UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería de Telecomunicación
y Electrónica

eite

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Utilización de Blockchain para la validación de documentos

Titulación:	Grado en Ingeniería en Tecnologías de la Telecomunicación
Mención:	Telemática
Autor:	D. Marta Jiménez Rodríguez
Tutores:	D. Álvaro Suárez Sarmiento
	Dña. Elsa María Macías López
	Felipe Raúl Mendoza Álamo
Fecha:	Julio 2019

Índice

Índices

Índice general

Índice de tablas

Índice de figuras

Índice de códigos

1 Introducción

En este capítulo presentamos el problema a resolver en este *Trabajo Fin de Grado* (*TFG*), el cual ha sido planteado por una empresa, *The Singular Factory*, después de realizar en dicha empresa las prácticas curriculares. También se presenta una breve introducción a la tecnología *Blockchain*, clave para resolver el problema, y la estructura de la memoria.

1.1 Blockchain

Una *Blockchain* es una red *Peer*-to-*Peer* (*P2P*), red basada en *nodos* distribuidos donde no existen nodos centrales que gestionen a los demás. En el caso de la *Blockchain* cada *nodo* de la red tiene una copia de todas las operaciones realizadas en la red. Esta información se almacena en forma de una cadena de bloques. Las operaciones realizadas son denominadas transacciones, pero no siempre son utilizadas para un intercambio monetario, también pueden ser utilizadas para el intercambio de información, que es el caso que nos interesa.

Estas transacciones se introducen en los bloques de la cadena a través de un proceso denominado *minar*. Este trabajo lo realizan miembros de la red llamados mineros. Éstos reciben un pago en criptomoneda como incentivos para realizar este trabajo. Este proceso requiere la resolución de complejas operaciones matemáticas.

Para evitar que estos mineros falseen las transacciones que se introducen en los bloques existen varios mecanismos, conocidos como mecanismos de consenso. El más utilizado de ellos es *Proof-of-Work* (*PoW*) [3]. Esto consiste en hacer que los mineros compitan por minar los bloques de forma que si algún minero intenta falsear algo se quede retrasado con respecto a los demás y no llegue a crear un bloque valido.

De esta forma se obtiene una un nivel elevado de seguridad al tener todos los nodos una copia de la información y al ser muy improbable que los mineros falseen la información (porque deberían falsear todas las copias de forma sincronizada). Esta seguridad y la falta de centralización, que implica la necesidad de confiar en un tercero, son los principales incentivos de la tecnología *Blockchain*. Pero esta tecnología aún no está madura y uno de los objetivos de este TFG es valorar su utilidad actual.

La tecnología *Blockchain* fue descrita, por primera vez, en 1991 por Stuart Haber and W. Scott Sornita [1]. Pero fue implementada, por primera vez, con *Bitcoin* [2], creado bajo el seudónimo de Satoshi Nakamoto, para generar la criptomoneda con el mismo nombre. Este protocolo tiene como objetivo descentralizar la actividad financiera en la Economía, mediante un sistema de consenso.

Al tener *Bitcoin* como único objetivo los pagos electrónicos, tiene ciertas limitaciones que no lo hacen viable para nuestro objetivo. Las cuatro principales limitaciones son:

- El tamaño de un bloque es muy pequeño. Este es de un máximo de 1 MB, aunque existe cierto debate sobre ampliarlo.

- Las cuotas transferencia pueden llegar a ser muy altas, llegando a alcanzarse un pico de 55 dólares norteamericanos (USD) en diciembre de 2017. Actualmente, consultado en mayo de 2018, se encuentran entre 1-2 USD. Esto se puede observar en las figuras 1.1 y 1.2.
- La velocidad de validación de una transacción puede ser muy lenta, llegando a tardar varias horas.
- Falta de flexibilidad para realizar otras operaciones que no sean trasferencias de la criptomoneda.

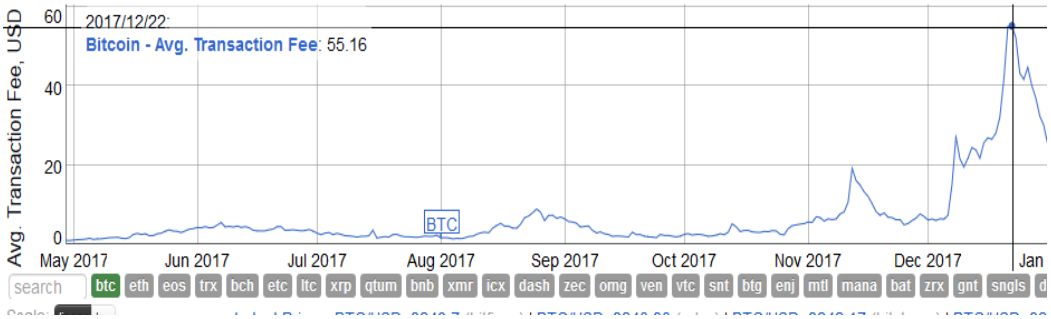


Figura 1.1 Cuota de transacción Bitcoin en diciembre 2017
Fuente: https://bitinfocharts.com

Figura 1.2 Cuota de transacción Bitcoin en mayo 2018
Fuente: https://bitinfocharts.com

1.2 *Ethereum* y Smart Contracts

Debido a las limitaciones de *Bitcoin* y, a que no estamos interesados en los pagos electrónicos, *Bitcoin* no coincide con nuestros intereses. En cambio, la plataforma *Ethereum* [4] permite bloques de mayor tamaño, unas cuotas menores y una velocidad de transferencia muchísimo menor. Y, especialmente relevante para nuestro caso, permite crear las denominadas aplicaciones descentralizadas o *Decentralized Applications* (*DApp*) que se ejecutan en la *Blockchain*. La *Ethereum Virtual Machine* (*EVM*) implementa el código *Smart Contract* (*SC*) de estas DApp en la *Blockchain* [5] (dado que la EVM es *Turing-Complete*, en ella se podría llevar a cabo cualquier tarea si se dispone de la potencia de cálculo necesaria). Cada SC almacena datos: en una cola *Last In First Out* (*LIFO*), en una memoria de acceso aleatorio, y en almacenamiento persistente (permanece tras la ejecución) de clave-valor (campo de almacenamiento es que se usaría para almacenar los datos de los documentos). Existen diversos lenguajes de programación de SC; utilizamos *Solidity* [8] por ser el más parecido a *JavaScript* y ser el más utilizado y documentado.

Para interactuar con este SC hay distintos medios, aunque siempre es necesario tener acceso a un nodo de la red. En primer lugar, tenemos la librería de *javascript Web3* [6], pero como queremos realizar una *Application Programming Interface* (*API*) en *Hypertext Preprocessor* (*PHP*), y no existen bibliotecas completas como Web3 para PHP, debemos ir un nivel más abajo y comunicarnos directamente con el nodo mediante *Remote Procedure Call* (*RPC*) utilizando *JavaScript Object Notation* (*JSON*) con un formato definido por *Ethereum*.

La posibilidad de utilizar una red de pruebas (*Rinkeby*) de *Ethereum* [7] para evitar pagar por el uso de la red principal de *Ethereum*, y obtener unos resultados con cierto grado de realismo, fue otro de los motivos que nos inclinó por el uso de Ethereum.

1.3 Objetivo

El principal objetivo de este trabajo de fin de grado es almacenar una serie de datos asociados a documentos en la red *Rinkeby* de la plataforma *Ethereum* para obtener una validación con un alto grado de confianza y valorar el rendimiento de la *Blockchain*. Este objetivo está marcado por la empresa *The Singular Factory* [19].

Esta empresa *of*rece un servicio denominado *Wupplier* [20], en beta cerrada actualmente, que gestiona facturas. Una de las funcionalidades de este servicio es adelantar el

pago de la factura a cambio de una comisión por parte de la entidad financiera. Pero puede darse la situación en la que un proveedor solicite a varias entidades el pago por adelantado de una misma factura aceptada. Por esto se requiere un sistema confiable que permita a las entidades de financiación comprobar si una factura está en trámites de un proceso de adelanto de pago o ya se ha pagado para evitar este tipo de fraude. Aquí es donde entra *Blockchain*, es un espacio de almacenamiento, en el que todas las entidades de financiación y empresas pagadoras en general pueden consultar el estado real del pago de una factura de una manera confiable y segura dejando un registro de los cambios que se han producido sobre el documento desde su inserción en la cadena de bloques por parte de una entidad autorizada.

Para gestionar la forma en la que se almacena la información y la lógica que se le aplicaría, se utilizaran los SC. Estos tienen como objetivo original sustituir a los contratos más tradicionales eliminando intermediarios y ambigüedades, pero en nuestro caso queremos valernos de su capacidad para almacenar información en la *Blockchain* con una lógica asociada y simular una base de datos, pero con la confianza que da la *Blockchain*.

Como desventajas con respecto a bases de datos tradicionales se puede esperar una velocidad de respuesta lenta, una capacidad de almacenamiento limitada y un coste elevado debido a las cuotas de la red.

Para acceder a la *Blockchain* y al SC se desea crear una API utilizando el *framework Symfony* que se aloja en un servidor (que también despliega a un nodo de la red *Ethereum*); de esta forma no sería necesario disponer de un nodo para acceder a la información y resultaría más sencillo manejarla.

Este objetivo se desglosa en los siguientes objetivos específicos:

- *A*nálisis del funcionamiento de *Blockchain* para conocer sus posibilidades en relación con el objetivo del TFG. Esto incluye determinar la capacidad de almacenamiento de información estimada, la velocidad de respuesta de la red y su eficiencia con respecto a las cuotas existentes en la *Blockchain*.
- Creación de los SC para gestionar los datos a almacenar.
- Creación de una *Application Programming Interface* (*API*) para interactuar con la *Blockchain* y los datos, en PHP utilizando el *framework Symfony*.

1.4 Estructura del documento

En el capítulo 2 se presenta la *Blockchain* en más detalle junto con otras herramientas utilizadas. Estas otras herramientas son *Smart Contracts*, el *framework* Symfony, el s*oft*ware *Go Ethereum* (*Geth*) para gestionar el nodo de la *Blockchain* y la API *RPC* utilizada para comunicarnos con el nodo.

En el capítulo 3 se presenta el problema planteado y la solución propuesta. Se presenta la solución implementada desde el punto de vista del SC y de la API en *Symfony*, detallando los métodos y llamadas de la arquitectura *REpresentational State Transfer* (*REST*) en ambos casos.

En el capítulo 4 se detalla la implementación de la API y se analizan las decisiones tomadas.

En el capítulo 5 se analiza el rendimiento de la API y se considera si nuestra solución es válida para el problema planteado.

Finalmente, en el capítulo 6, se muestran las conclusiones y la utilidad de la *Blockchain* en su estado actual con respecto a nuestro problema.

2 Tecnologías utilizadas

En este capítulo se presenta la *Blockchain* en más detalle junto con otras herramientas utilizadas. Estas otras herramientas son *SC*, *el framework Symfony*, *el software Go Ethereum* para gestionar el nodo de la *Blockchain*.

2.1 Blockchain

Para hacer entender la *Blockchain* primero presentamos qué es una red P2P y como se almacena la información en bloques. Los nodos de las redes P2P cumplen los dos roles a la vez: clientes y servidores, e intercambian información de forma directa. En el caso de una *Blockchain* todos los nodos tienen la misma información (bloques). Y, como no existen nodos centrales, la caída de un nodo no afecta al funcionamiento de la red.

2.1.1 Minado de bloques

Todas las operaciones realizadas en la *Blockchain* se denominan *transacciones*; utilizadas para el intercambio de información entre nodos.

Estas transacciones se incluyen en una lista de transacciones pendiente de procesar, y de ser considerar validadas, hasta que un minero coge un grupo de estas transacciones para añadirlas a la cadena en forma de bloque. La elección de estas transacciones depende de la cuota asociada a esta transacción. De forma que transacciones dispuestas a pagar una cuota más alta son procesadas más rápidamente. Este bloque no resulta único; cada minero crea un bloque en función del tamaño máximo de bloque del protocolo correspondiente.

Cuando se completa el tamaño del bloque con transacciones, se debe resolver un complejo problema matemático de cifrado criptográfico, único para el bloque. El primer minero que resuelve este problema emite la solución y el bloque y el resto de los mineros comprueba que la solución es válida para ese bloque. Si la mayoría de los mineros comprueban esta condición el bloque se añade a la cadena y se confirman las transacciones incluidas.

Tras incluirse un nuevo bloque los mineros descartan el bloque en el que estaban trabajando, ya que este podría incluir transacciones ya procesadas que resultarían no validas en la red, y por lo tanto el bloque no sería válido y esto resultaría en una pérdida de trabajo de procesado.

No entramos en detalle de los métodos de criptografía y los conceptos matemáticos necesario para el minado de bloques ya que en este TFG no estamos analizando el grado de seguridad de la *Blockchain*, sino su capacidad para almacenar y gestionar información. Sin embargo, si explicamos el concepto de hash ya que nos resulta necesario.

2.1.2 Hash

Una función hash es un proceso matemático mediante el cual se transforma una entrada de información de cualquier tamaño en una salida de información de tamaño fijo. La idea detrás de

esta función es que la salida represente la entrada de forma única. Pero que no se pueda obtener la entrada a partir de la salida.

Es decir, conociendo la salida de un hash podemos determinar si una información es idéntica a la entrada original aplicando hash a esta segunda información y comparando la salida resultante. La única forma de obtener la entrada a partir de una salida sería probar todas las posibilidades posibles para la entrada usando fuerza bruta. Si tenemos en cuenta que la entrada puede ser de cualquier tamaño existe una gran cantidad de posibilidades de entrada y resulta extremadamente complicado descifrar un hash.

No explicamos los conceptos matemáticos para obtener estas funciones por las mismas razones que no entramos en los detalles del puzle criptográfico para minar bloques.

Existen diferentes funciones hash disponibles, pero en nuestro caso usamos principalmente, *Secure Hash Algorithm* (*SHA*). En concreto utilizamos 2 variaciones de este hash, SHA-256 y SHA-3. SHA-3 también se denomina keccak-256 en *Ethereum*.

2.2 Smart *Contracts*

Un SC se realiza a través de un programa informático (*script*) y es capaz de ejecutarse y hacerse cumplir a sí mismo sin intermediario. La ventaja es que no tienen el problema de ser mal interpretado como los contratos tradicionales. Por otro lado, al formar parte de la red *Blockchain*, no puede ser cambiado sin que la misma tecnología lo rechace, siendo así descentralizado, inmutable y transparente.

Sin embargo, no fue hasta la llegada del *Blockchain* que esta idea fuese llevada a la práctica, sobre todo con la llegada del *Bitcoin*, que permitía un sistema de pagos descentralizados. Pero *Bitcoin* no había sido creada para algo más que ser una criptomoneda. Es aquí donde aparece *Ethereum*, una plataforma *Blockchain* que permite la creación e implementación de SC creados en el lenguaje de programación *Solidity* [8], que, tras ser desplegados en la red, no son controlados por ninguna de las partes firmantes debido a la naturaleza descentralizada de la red, actuando de forma automática tras cumplir las condiciones estipuladas en el mismo. Además, implementa un estado de seguridad mayor al de un contrato tradicional, reduce los costes de notariado de un contrato impreso y reduce el tiempo asociado a este tipo de interacciones.

2.2.1 Solidity

Solidity es un lenguaje de alto nivel o "*Turing Complete*" para la implementación de contratos inteligentes en la red de *Ethereum*. Fue influenciado por C ++, Python y *Javascript* y está diseñado para trabajar en la *Máquina Virtual Ethereum* (*EVM*). El concepto de Turing Complete significa que puede realizar cualquier operación si dispone del poder computacional, hardware, suficiente. Su ejecución se aplica sobre EVM.

Admite herencia, bibliotecas y tipos complejos definidos por el usuario, entre otras características. Está diseñado y compilado en código de bytes (*bytecode*).

2.2.2 Ethereum Virtual Machine

Ethereum permite a los usuarios crear sus propias operaciones de cualquier complejidad que deseen. De esta manera, sirve como una plataforma para muchos tipos diferentes de aplicaciones de *Blockchain* descentralizadas, que incluyen, pero no se limitan a las criptomonedas.

Ethereum en el sentido estricto se refiere a un conjunto de protocolos que definen una plataforma para aplicaciones descentralizadas. En el corazón de la plataforma está EVM que puede ejecutar código de complejidad algorítmica arbitraria, siendo esta un sistema de Turing completo. Los desarrolladores pueden crear aplicaciones que se ejecutan en EVM utilizando lenguajes de programación basados en lenguajes existentes como *Javascript* y Python. Todos y cada uno de los nodos de la red ejecutan el EVM y ejecutan las mismas instrucciones para mantener el consenso a través de la cadena de bloques.

El consenso descentralizado le otorga a *Ethereum* niveles extremos de tolerancia a fallos, asegura cero tiempos de inactividad y hace que los datos almacenados en la cadena de bloques permanezcan invariables y resistentes a la censura.

2.2.3 Gas

Una de las ventajas de *Ethereum* de trabajar con un lenguaje *Turing Complete* es esa posibilidad de utilizar bucles que permiten crear aplicaciones más complejas de una manera más simple y eficiente. Como desventaja, esto implica el tener que disponer de un mecanismo como el '*Gas*' para evitar que el sistema pueda ser colapsado.

El Gas es la forma que tiene *Ethereum* de calcular el coste de las transacciones y no es ni más ni menos que el gasto computacional de procesar una transacción o SC en la red, y las necesidades de su almacenamiento.

Es un método que surge para evitar que se den bucles infinitos y colapsar así el sistema. De no existir el Gas, nada impediría que una aplicación mal programada pudiera dejar el sistema inservible.

Se requiere que cada transacción incluya un límite de Gas y una tarifa que se esté dispuesto a pagar por G Los mineros tienen la opción de incluir la transacción al bloque y cobrar la tarifa o no. Si el número total de Gas utilizado por el cálculo generado por la transacción es menor o igual que el límite de Gas, entonces la transacción procesa. Si el total de Gas excede el límite de Gas, todos los cambios se revierten, excepto que la transacción aún es válida y el minero aún puede cobrar la tarifa.

2.3 El nodo y *Geth*

Para disponer de acceso a la *Blockchain* debemos tener acceso a un nodo de ésta. Existen servicios que nos permiten hacer esto de forma remota como *Infura* [25] o *Metamask* [26]. Pero en nuestro caso queremos disponer de un nodo propio que se aloje en la misma máquina que el código de nuestra API. Para ellos debemos instalar la implementación *Go* de *Ethereum* (*Geth*) [18].

En nuestro caso lo hemos instalado como parte del S*of*tware *Mist* [27] ya que posteriormente utilizamos ese s*of*tware para el despliegue del SC.

La instalación de un nodo puede llevar una gran cantidad de tiempo y necesita una gran cantidad de almacenamiento ya que requiere la descarga de la *Blockchain* completa. Por supuesto también necesita una buena conexión a Internet.

A la hora de instalar nuestro nodo tenemos una serie de opciones. Principalmente utilizamos 2 de ellas.

La nuestra primera opción es seleccionar a que red pertenece nuestro nodo. *Ethereum* dispone de una red principal y 3 redes de prueba públicas, *Rinkeby*, *Ropsten* y *Kovan*. También tendríamos la posibilidad de montar nuestra propia red privada.

La otra opción que nos interesa es configurar un servidor *RPC* con unas API determinadas [12] para comunicarnos con *Ethereum*. Con estas API podemos comprobar el estado del nodo, leer datos y escribir datos gastando Gas.

2.4 Framework Symfony

Symfony [14] es un *framework* para aplicaciones y servicios Web en PHP. Como *framework*, *Symfony* es una colección de componentes reutilizables que cumplen una tarea específica. Estos componentes automatizan las tareas más básicas y generan un base sobre la que resulta más sencillo desarrollar.

Symfony fue lanzado en 2005 como *open source* y ha desarrollado una buena reputación debido a su alta flexibilidad y fiabilidad. Plataformas conocidas como *Drupal* y *Magento* usan componentes de *Symfony*.

La versión más reciente de *Symfony* considerada estable es la 4.1. Pero en nuestro TFG usamos la 3.4, a recomendación de la empresa para facilitar posibles integraciones con sus proyectos existentes en esta versión. Para utilizar *Symfony* en nuestro TFG utilizamos el gestor de paquetes en PHP *Compose*r [21].

Symfony utiliza una arquitectura *Model View Controller* (*MVC*). Esta arquitectura divide las aplicaciones en 3 partes.

- La Vista da forma a la interfaz de usuario. Determina la posición de los componentes de una Web, los colores, el formato del texto…
- El Modelo gestiona los datos. Realiza búsquedas en la base de datos y contiene la lógica de la aplicación.
- El Controlador actúa de intermediario entre el modelo de la vista. Comunica al modelo los movimientos del usuario y vuelca los datos del Modelo en la Vista.

En nuestro TFG, al tratarse solamente de una API REST, no utilizamos el componente de vista ya que no existe una interfaz gráfica de usuario.

2.5 Postman

Postman [11] es la *API Development Environment* (ADE) que utilizamos para interactuar con nuestra API. En nuestro caso utilizamos este software de forma superficial ya que solo lo utilizamos para realizar llamadas a nuestra API. En las respuestas a estas llamadas *Postman* nos permite analizar el tiempo de respuesta, el tamaño y el código de estado HTTP.

Además, *Postman* nos permite automatizar baterías de tests para nuestra API y gestionar distintos entornos para estos tests. En un entorno podemos configurar distintas variables globales, como la URL base o datos de autenticación del cliente.

3 Solución propuesta

En este capítulo se presenta el problema planteado y la solución sugerida por la empresa. Además, se detallan la especificación del *SC*, con sus cambios respecto a los datos iniciales debido a las limitaciones de *Solidity*. Y la especificación de la API donde se detallan los métodos REST con sus rutas y las respuestas dadas.

3.1 Problema planteado

Como se presentó en el capítulo 1, la empresa The Singular Factory *of*rece un servicio denominado *Wupplier* que gestiona facturas. Una de las funcionalidades de este servicio es adelantar el pago de la factura a cambio de una comisión por parte de la entidad financiera. Esto da la oportunidad de fraude si un proveedor solicita a varias entidades el pago por adelantado de una factura aceptada. Por esto se requiere un sistema confiable que permita a las entidades de financiación comprobar si una factura está en trámites de un proceso de adelanto de pago o ya se ha pagado para evitar este tipo de fraude. Aquí es donde entra *Blockchain*, proporcionando un lugar donde todas las entidades de financiación y empresas pagadoras en general puedan consultar el estado real del pago de una factura de una manera confiable y segura dejando un registro de los cambios que se han producido sobre el documento desde su inserción en la cadena de bloques por parte de una entidad autorizada.

Para gestionar la forma en la que se almacena la información y la lógica que se le aplica a esta se utilizan los SC por su capacidad para almacenar información confiable en la *Blockchain* y simular una base de datos.

Una factura debe tener un identificador único por parte de *Wupplier* y un identificador único por parte del propietario de la factura. Este segundo identificador debe ser único por empresa y por año fiscal. Además, debe almacenar la moneda utilizada, la forma de pago utilizada y los días de plazo para realizar el pago. También se almacenan identificadores del proveedor y del cliente de la factura.

Con respecto a las fechas, se almacena el año fiscal, la fecha de emisión de la factura, la fecha de expiración del plazo de pago y la fecha en la que se realiza el pago.

Si se solicita y se acepta un adelanto se debe añadir el total del adelanto, el nombre de la institución financiera que lo emitió y la fecha de vencimiento.

Por último, cada factura mantiene 2 estados, uno general y otro relacionando los adelantos. El ciclo de estos estados se encuentra en la figura 3.1, junto con un texto explicativo.

En la tabla 3.1 estos datos se encuentran más detallados e identificados con los nombres que se utilizan en este documento y en el código del SC y de la API.

Tabla 3.1 Campos de un documento

Nombre del campo	**Descripción**
documentUniqueId	Valor alfanumérico que representa el Identificador único de la factura generado con SHA256 a partir del *invoiceNumber*, *total*, *supplierName*, *customerName y fiscalYear*. No se puede cambiar.
invoiceNumber	Campo que almacena el número de la factura. Es un valor alfanumérico generado por el propietario de la factura, cuyo valor debe ser único dentro de la empresa y año fiscal. No se puede cambiar.
fiscalYear	Valor entero que representa el año fiscal. No se puede cambiar.
total	Importe total de la factura. No se puede cambiar.
factoringTotal	El importe que recibiría el proveedor tras descontar la comisión de la Institución financiera que adelanta el pago. Solo se le asignaría un valor al confirmarse que el adelanto está aceptado y no se podría cambiar posteriormente.
state	Estado de la factura. Los estados disponibles son: *pending*, *accepted* y *paid*.
currency	Moneda en la que se realiza el pago (euros, dólares...). No se puede cambiar.
paymentType	Tipo de pago, normalmente es transferencia. No se puede cambiar.
supplierName	Identificador del proveedor. No se puede cambiar.
customerName	Identificador del cliente. No se puede cambiar.
financialInstitutionName	Identificador de la Institución financiera. Solo se le asignaría un valor al confirmarse que el adelanto está aceptado y no se podría cambiar posteriormente.

factoringState	Estado del pago adelantado. Los estados disponibles son: *fact_pending*, *fact_requested* y *fact_accepted*.
paymentTerms	Días de plazo para realizar el pago. No se puede cambiar.
invoiceDate	Fecha y hora en la que se creó la factura. No se puede cambiar.
paymentDate	Fecha y hora en la que se realizó el pago. Solo se le asignaría un valor al confirmarse que la factura está pagada y no se podría cambiar posteriormente.
expirationDate	Fecha y hora en la que se acaba el plazo para realizar el pago. No se puede cambiar.
factoringExpirationDate	Fecha y hora en la que se acaba el plazo para pagar a la institución financiera. Solo se le asignaría un valor al confirmarse que el adelanto está aceptado y no se podría cambiar posteriormente.

3.2 Especificación SC

Dentro de nuestro SC se definen:

1. Qué es un documento: un objeto que tiene un *identificador* (*ID*), una serie de datos y la posición de su ID en un *array*.
2. Un *mapping* que contiene documentos: tabla hash, una estructura de datos que asocia claves con valores, cuyas claves son los ID de los documentos. El problema del *mapping* es que no nos permite saber cuántos documentos hay dentro y por lo tanto no podemos obtener una lista de los documentos para gestionarlos.
3. Un *array* con los ID de los documentos: contiene solamente los IDs de los documentos y no los documentos en sí ya que entonces sería necesario realizar un bucle, que podría ser costoso, para buscar documentos completos según el ID sin conocer la posición.

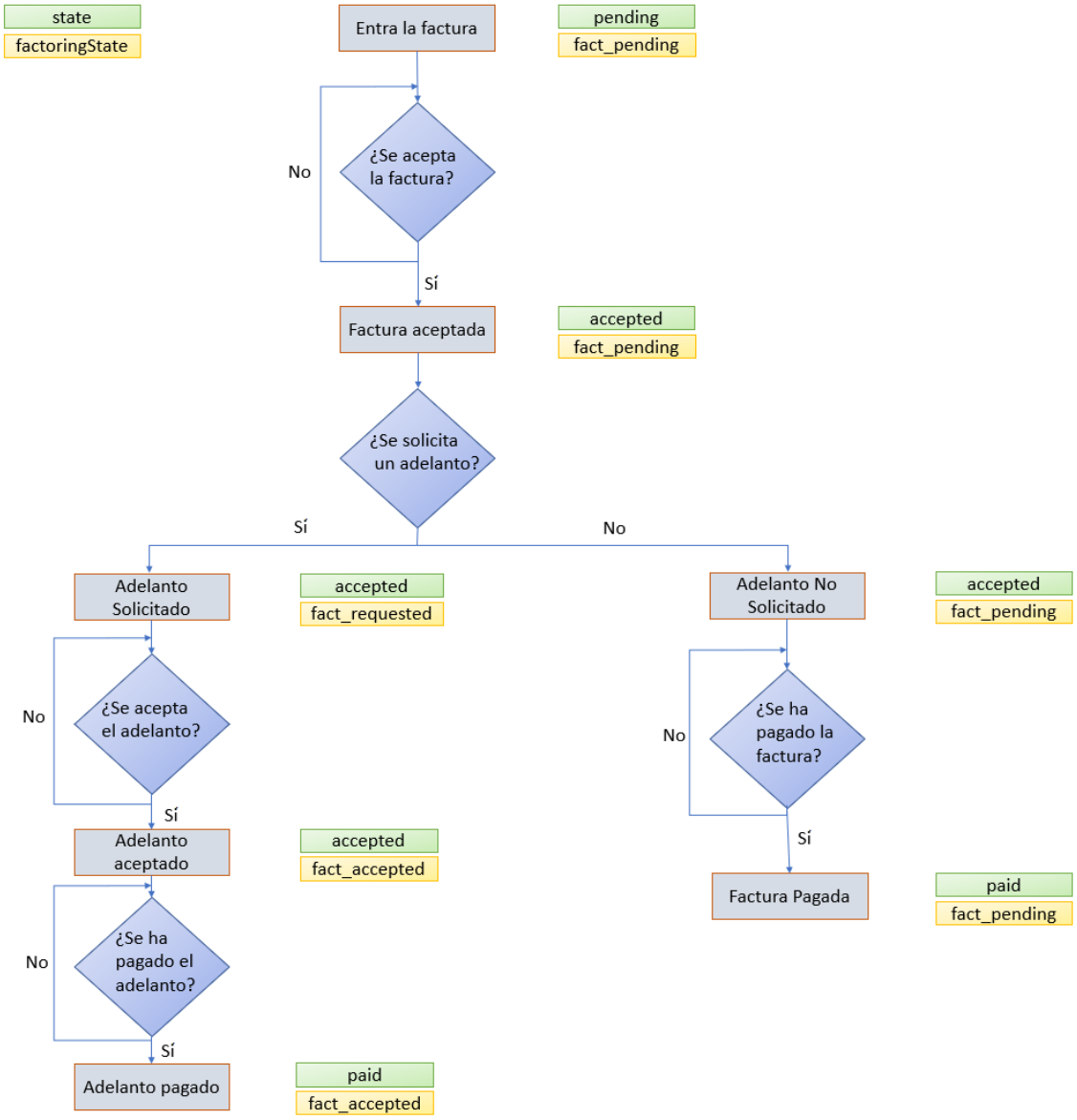


Figura 3.1 Diagrama del ciclo de estados

Ambos estados empiezan como pendientes (*pending* y *fact_pending*), una vez que el cliente acepte la factura *state* pasa a ser aceptado y *factoringState* continúa pendiente (*accepted y fact_pending*). Si no se solicita un adelanto, una vez que se pague la factura *state* pasa a ser pagado, y *factoringState* continúa pendiente (*paid y fact_pending*).

En el caso contrario, cuando se solicita un adelanto, *factoringState* pasa a ser solicitado y *state* se mantiene en aceptado (*accepted y fact_requested*). Cuando se acepta el adelanto *factoringState* pasa a ser aceptado y *state* se mantiene en aceptado (*accepted y fact_accepted*). Finalmente, cuando se paga la factura *state* pasa a ser pagado y *factoringState* continúa aceptado (*paid y fact_requested*).

El objetivo de tener estas 2 estructuras, *mapping* y *array*, en lugar de sólo una de ellas es obtener una lista de los ID de los documentos y que no sea necesario realizar un bucle para buscar un documento concreto. Combinándolos el *mapping* nos evita iterar y el *array* nos permite tener una lista.

Entrando ahora en los datos a almacenar, primero mencionar que el *documentUniqueId* se genera fuera del SC, en la API, ya que sería costoso concatenar *strings* y crear un hash dentro del contrato. Además, los datos *fiscalYear*, *invoiceDate* y *expirationDate* se fusionan en un único dato llamado dates para reducir el número de parámetros debido a limitaciones de *Solidity*. Y aparte de los datos de la tabla 3.1 se añade otro dato (variable) llamado *index* que almacena la posición en el *array*.

Por último, debemos definir cómo almacenamos estos datos ya que otra de las limitaciones de *Solidity* son los tipos de datos soportados [8]. En nuestro caso utilizamos *bytes*, *byte<X>* y *uint*:

- *bytes<X>*: *array* de bytes de tamaño fijo, este tamaño viene determinado por X, resulta mucho menos costoso en términos de Gas que bytes. Sin embargo, solo permite almacenar hasta 32 B o caracteres (*bytes32*).
- *bytes*: *array* de bytes de tamaño dinámico, resulta más costoso que bytes<X> pero como muchos de nuestros datos superan 32 B es necesario.
- *uint*: número entero no negativo, en concreto de 256 b. Se pueden especificar enteros de menor tamaño en saltos de 8 b de igual manera que en *bytes<X>* con *int<X>* (*int8*, *int16*, *int24*...). Pero en nuestro caso no limitamos el tamaño.

De esta forma se usa bytes para la mayoría de los datos ya que no sabemos su tamaño, o su tamaño máximo estimado por la empresa supera los 32 B. El tamaño máximo estimado se obtiene en base a los tipos de datos utilizados en la base de datos.

En otros casos podemos saber su tamaño máximo, y este se encuentra por debajo de 32 B, y usamos una variación de bytes<X>.Y en un par de ocasiones usamos *uint*. Una lista con los tipos de datos usados en cada dato y el razonamiento de su uso se puede encontrar en Tabla 3.2.

Tabla 3.2 Tipos asignados a los datos

Nombre del dato	**Tipo de dato**	**Razonamiento**
documentUniqueId	bytes	Hash generado de 64 B, pese a tener un tamaño fijo debemos usar el *array* dinámico porque supera 32 B.
invoiceNumber	bytes	El tamaño máximo de este campo es de 100 caracteres, por lo que se usa el *array* dinámico.
total	bytes	El tipo de dato usado originalmente es float que no está soportado en *Solidity* y por lo tanto se usa bytes.
factoringTotal	bytes	El tipo de dato usado originalmente es *float* que no está soportado en *Solidity* y por lo tanto se usa bytes.
state	bytes8	Al haber solo ciertos valores permitidos se toma de tamaño máximo el tamaño del valor más largo, en este caso *accepted* con 8 caracteres.
currency	bytes3	Se almacena un identificador de la moneda formado por 3 caracteres (por ejemplo, EUR).
paymentType	bytes	El tamaño de este campo es de 50 caracteres, por lo que se usa el *array* dinámico.
supplierName	bytes	El tamaño de este campo es de 255 caracteres, por lo que se usa el *array* dinámico.
customerName	bytes	El tamaño de este campo es de 255 caracteres, por lo que se usa el *array* dinámico.
financialInstitutionName	bytes	El tamaño de este campo es de 255 caracteres, por lo que se usa el *array* dinámico.
factoringState	bytes14	Al haber solo ciertos valores permitidos se toma de tamaño máximo el tamaño del valor más largo, en este caso *fact_requested* con 14 caracteres.

paymentTerms	uint	El tipo de dato usado originalmente es *integer*.
paymentDate	bytes19	Fecha y hora que se almacena de la siguiente manera: AAA/MM/DD HH:MM:SS. Utilizando así 19 caracteres.
dates	bytes	En este campo se concatenan *fiscalYear*, *invoceDate y expirationDate* de la siguiente forma: FYFY-AAA/MM/DD HH:MM:SS-AAA/MM/DD HH:MM:SS. Al superarse 32 caracteres se debe utilizar bytes.
factoringExpirationDate	bytes19	Fecha y hora que se almacena de la siguiente manera: AAA/MM/DD HH:MM:SS. Utilizando así 19 caracteres.
index	uint	Entero que indica la posición del ID en el *array*.

Aún con la reducción de parámetros hecha con el dato *dates* siguen existiendo demasiados parámetros como para crear un documento completo en un solo método. Sin embargo, varios de estos datos no son necesarios en un nuevo documento. Los datos de un adelanto no existirían aun y se añadirían al ser aceptado el adelanto. De la misma manera la fecha de pago no se introduce hasta que se paga la factura. Y los estados empezaran como *pending* y *fact_pending* sin necesidad de pasarlos como parámetros.

De esta forma el contrato cuenta con un método para insertar nuevos documentos, una serie de métodos para comprobar la existencia y los estados de un documento, y una serie de métodos para obtener los datos de un documento. Además, existen métodos para modificar los estados de un documento y los datos relacionados con un adelanto de acuerdo a estos estados. También hay un método para borrar un documento y un método que permite obtener el ID de un documento a partir de su *index* en el *array*. Y, finalmente, existe un método para obtener la cuenta de los documentos. Una definición más completa de estos métodos se puede encontrar en la tabla 3.3.

Tabla 3.3 Métodos del Smart Contract

Nombre	**Parámetros**	**Descripción**
insertDocument	- documentUniqueId - invoiceNumber - total - currency - paymentType - supplierName - customerName - paymentTerms - dates	Añade un nuevo documento con los parámetros dados. Los estados se inicializan como pendientes, ”pending” *y “fact_pending”*, y el resto de los datos relacionados con los adelantos y el pago se asignan una vez que se llegue a esos estados. Antes de ejecutarse se comprueba que documentUniqueId no exista ya.
exists	- documentUniqueId	Devuelve un booleano que es verdadero si el documento existe o falso si el documento no existe.
documentIsPending	- documentUniqueId	Devuelve un booleano que es verdadero si es documento está pendiente, ”pending”, o falso en caso contrario. Antes de ejecutarse se comprueba que el documentUniqueId exista.
documentIsAccepted	- documentUniqueId	Devuelve un booleano que es verdadero si el documento está aceptado, ”accepted”, o falso en caso contrario. Antes de ejecutarse se comprueba que el documentUniqueId exista.

documentFactoringIsPending	- documentUniqueId	Devuelve un booleano que es verdadero si el adelanto está pendiente, ”fact_pending”, o falso en caso contrario. Antes de ejecutarse se comprueba que el documentUniqueId exista.
documentFactoringIsRequested	- documentUniqueId	Devuelve un booleano que es verdadero si el adelanto está solicitado, ”fact_requested”, o falso en caso contrario. Antes de ejecutarse se comprueba que el documentUniqueId exista.
documentFactoringIsAccepted	- documentUniqueId	Devuelve un booleano que es verdadero si el adelanto está aceptado, ”fact_accepted”, o falso en caso contrario. Antes de ejecutarse se comprueba que el documentUniqueId exista.
documentIsPaid	- documentUniqueId	Devuelve un booleano que es verdadero si el documento está pagado, “paid”, o falso en caso contrario. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setFactoringTotal	- documentUniqueId factoringTotal	Establece el dato factoringTotal de un documento si se comprueba que se ha aceptado el adelanto, ”fact_accepted”.

		Antes de ejecutarse se comprueba que el documentUniqueId exista.
setFactoringExpirationDate	- documentUniqueId - factoringExpirationDate	Establece el dato factoringExpirationDate de un documento si se comprueba que se ha aceptado el adelanto, ”fact_accepted”. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setFinancialInstitutionName	- documentUniqueId - financialInstitutionName	Establece el dato financialInstitutionName de un documento si se comprueba que se ha aceptado el adelanto, ”fact_accepted”. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setPaymentDate	- documentUniqueId - paymentDate	Establece el dato factoringTotal de un documento si se comprueba que se ha pagado, “paid”. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setStateAcceptedFromPending	- documentUniqueId	Se marca el documento dado como aceptado, ”accepted”, si se comprueba que antes estaba pendiente, “pending”. Antes de ejecutarse se comprueba que el documentUniqueId exista.

setStatePaidFromAccepted	- documentUniqueId	Se marca el documento dado como pagado, “paid”, si se comprueba que antes estaba aceptado, “accepted”. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setStatePaidFrom AcceptedPlusInfo	- documentUniqueId - paymentDate	Se marca el documento dado como pagado, “paid”, y se establece el dato paymentDate si se comprueba que antes estaba aceptado, ”accepted”. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setFactoringStateRequested	- documentUniqueId	Se marca el adelanto del documento como solicitado, “fact_requested”, si se comprueba que el documento se encuentra aceptado, “accepted”, y estado del adelanto es pendiente, ”fact_pending”. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setFactoringStateAccepted FromRequested	- documentUniqueId	Se marca el adelanto del documento como aceptado, “fact_accepted”, si se comprueba que el documento se encuentra aceptado, “accepted”, y estado del adelanto es solicitado,

		”fact_requested”. Antes de ejecutarse se comprueba que el documentUniqueId exista.
setFactoringStateAccepted FromRequestedPlusInfo	- documentUniqueId - factoringTotal - factoringExpirationDate - financialInstitutionName	Se marca el adelanto del documento como aceptado, “fact_accepted”, si se comprueba que el documento se encuentra aceptado, “accepted”, y estado del adelanto es solicitado, ”fact_requested”. Ademas, se establecen los datos factoringTotal, factoringExpirationDate y financialInstitutionName. Antes de ejecutarse se comprueba que el documentUniqueId exista.
deleteDocument	- documentUniqueId	Se elimina el documento con el id dado. Antes de ejecutarse se comprueba que el documentUniqueId exista.
getDocumentCount		Devuelve el número de documentos almacenados.
getDocumentAtIndex	- index	Devuelve el documentUniqueId del documento con el índice dado.
get<Dato>*	- documentUniqueId	Devuelve el dato correspondiente del documento dado por

		documentUniqueId. Antes de ejecutarse se comprueba que el documentUniqueId exista.

*Para evitar una repetición innecesaria no se listan todos los métodos *getter* para obtener datos.

En el método para insertar un nuevo documento se usan de parámetros se crea un nuevo documento con los datos dados y se asignan *state y factoringState* como pendientes. Antes de ejecutarse este método se comprueba que no existe un documento con el mismo ID.

Los métodos para comprobar la existencia y el estado de un documento reciben el ID de este como parámetro y devuelve un booleano que es verdadero si se cumple la condición correspondiente.

Los métodos que para obtener datos de un documento reciben el ID de este como parámetro y devuelve el dato correspondiente. Antes de ejecutarse el método se comprueba que existe un documento con el ID dado.

Finalmente, los métodos para modificar los estados de un documento y los datos relacionados con un adelanto reciben el ID del documento como parámetro. Estos métodos realizan comprobaciones de los estados del documento de forma que se cumpla el flujo del diagrama de la Figura 3.1.

Por último, todos estos datos comprueban la condición *onlyOwner*. Esta condición impide que estos métodos sean ejecutados si no los llama la cuenta que desplegó el contrato originalmente.

3.3 Especificación API

Hemos diseñado una API *RESTful* con llamadas y respuestas en formato JSON. Está creada con el *framework* de PHP *Symfony* y para comunicarse con el contrato desplegado en la *Blockchain* se hace uso de otra API de *Ethereum* con la realizamos llamadas *RPC* utilizando la librería *cURL* [10]. Las rutas de las llamadas de la API se han hecho siguiendo las pautas de buenas prácticas de *APIgee* [13].

Fuera de las llamadas relacionadas directamente con los documentos nuestra API tiene dos llamadas para comprobar el estado de la red, una de ellas para comprobar la conexión y

otra para la sincronización. Además, existe otra llamada para comprobar el estado de una transferencia.

Entrando ahora en las llamadas relacionadas con los documentos tenemos, en primer lugar, una llamada para insertar un nuevo documento con los parámetros *invoiceNumber*, *fiscalYear*, *total*, *currency*, *paymentType*, *supplierName*, *customerName*, *paymentTerms*, *invoiceDate y expirationDate*. De estos parámetros se toman *invoiceNumber*, *fiscalYear*, *total*, *paymentType y supplierName* para generar el identificador del documento concatenado los parámetros y aplicando *sha256* para asegurar un identificador único, llamamos a estos parámetros datos identificativos. Esta llamada para insertar un documento devuelve el hash de la transferencia realizada, el ID generado y los datos identificativos en el caso de que el ID generado no exista. Si el ID generado ya existe se devuelve un error junto con el ID y los datos identificativos.

También existen una serie de llamadas para comprobar la existencia y el estado de un documento. Estos métodos requieren el ID del documento y devuelven un booleano que es verdadero si la condición correspondiente se cumple. Al ser llamadas de sólo lectura no generan una transferencia.

El siguiente grupo de llamadas son *setters* para modificar los estados de acuerdo con el diagrama de la figura 3.1 y establecer datos relacionados con los adelantos y los pagos. Estos métodos requieren el ID del documento, así como los datos a establecer cuando corresponda. Devuelven el hash de la transferencia y el dato establecido cuando lo hay. Si no se cumple las condiciones de los estados también se devuelven mensajes de error.

El último grupo de llamadas son aquellas utilizadas para obtener los datos almacenados. Estos métodos requieren el ID y devuelven el dato correspondiente. Un caso especial dentro de este grupo es una llamada que devuelve todos los datos de un documento.

La última llamada de escritura consiste en eliminar un documento. Esta llamada requiere el ID y devuelve el hash de la transferencia.

Por último, tenemos una llamada que devuelve el número de documento almacenados y otra llamada que devuelve el ID de un documento según un índice. Combinando estas dos llamadas tenemos una última llamada que devuelve una lista de los ID existentes.

De forma genérica todas las llamadas que requieran el ID como parámetro se podrían llamar utilizar el identificador o los cinco datos que lo generan, *invoiceNumber*, *fiscalYear*, *total*, *paymentType y supplierName*, que llamamos datos identificativos. Estos métodos devuelven un error si el ID no se encuentra.

En el caso de utilizarse solo el ID este se pasa como parte de la ruta. Y se devolvería solamente el ID sin los datos además de las respuestas específicas de la llamada correspondiente.

En el caso de utilizarse los datos, estos podrán pasarse como JSON en el caso de las llamadas POST o como parámetros de la ruta en las llamadas GET. Se devolvería el ID y los datos identificativos además de las respuestas específicas de la llamada correspondiente

Todos estos métodos se encuentran más detallados en la tabla 3.4. En esta tabla se pueden ver las rutas y métodos *Hiper Text Transfer Protocol* (*HTTP*) utilizadas y una descripción más detallada de cada llamada. Para obtener una descripción más completa de las llamadas se puede consultar el Anexo 1. Este anexo incluye ejemplos de los JSON de entrada y de salida. Finalmente, en la tabla 3.5 se pueden consultar los distintos errores emitidos por la API.

Tabla 3.4 Métodos de la API

método http y ruta	**Descripción**
GET /net/connection	Esta llamada devuelve CONNECTED si hay conexión con la red y NOT_CONNECTED en caso contrario.
GET /net/sync	Esta llamada devuelve NODE_SYNC si el nodo está sincronizado y NODE_NOT_SYNC en caso contrario.
GET /transaction/{hashTx}	Devuelve el estado de una transferencia con SUCCESS o FAILURE. También incluye el gas usado y el bloque que incluye la transferencia.
POST /documents/	Añade un nuevo documento con los parámetros dados. Los estados se inicializan como pendientes, *pending* y *fact_pending*, y el resto de los datos relacionados con los adelantos y el pago se asignan una vez que se llegue a esos estados. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos no exista ya.
GET /documents/{id}/exists ***GET /documents/exists***	Devuelve un booleano que es verdadero si el documento existe o falso si el documento no existe.
GET /documents/{id}/pending ***GET /documents/pending***	Devuelve un booleano que es verdadero si es documento está pendiente, *pending*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.

GET /documents/{id}/accepted ***GET /documents/accepted***	Devuelve un booleano que es verdadero si el documento está aceptado, *accepted*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
GET /documents/{id}/factoring/pending ***GET /documents/factoring/pending***	Devuelve un booleano que es verdadero si el adelanto está pendiente, *fact_pending*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
GET /documents/{id}/factoring/requested ***GET /documents/factoring/requested***	Devuelve un booleano que es verdadero si el adelanto está solicitado, *fact_requested*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
GET /documents/{id}/factoring/accepted ***GET /documents/factoring/accepted***	Devuelve un booleano que es verdadero si el adelanto está aceptado, *fact_accepted*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
GET /documents/{id}/paid ***GET /documents/paid***	Devuelve un booleano que es verdadero si el documento está pagado, *paid*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/factoring/total ***POST /documents/factoring/total***	Establece el dato factoringTotal de un documento si se comprueba que se ha aceptado el adelanto, *fact_accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/factoring/expirationDate ***POST /documents/factoring/expirationDate***	Establece el dato factoringExpirationDate de un documento si se comprueba que se ha aceptado el adelanto, *fact_accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/factoring/financialInstitutionName	Establece el dato financialInstitutionName de un documento si se comprueba que se ha aceptado el adelanto, *fact_accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.

POST /documents/factoring/financialInstitutionName	
POST /documents/{id}/paymentDate ***POST /documents/paymentDate***	Establece el dato factoringTotal de un documento si se comprueba que se ha pagado, *paid*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/state/accepted ***POST /documents/state/accepted***	Se marca el documento dado como aceptado, *accepted*, si se comprueba que antes estaba pendiente, *pending*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/state/paid ***POST /documents/state/paid***	Se marca el documento dado como pagado, *paid*, si se comprueba que antes estaba aceptado, *accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/state/paidPlus ***POST /documents/state/paidPlus***	Se marca el documento dado como pagado, *paid*, y se establece el dato paymentDate si se comprueba que antes estaba aceptado, *accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/factoringState/requested ***POST /documents/factoringState/requested***	Se marca el adelanto del documento como solicitado, *fact_requested*, si se comprueba que el documento se encuentra aceptado, *accepted*, y estado del adelanto es pendiente, *fact_pending*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/factoringState/accepted ***POST /documents/factoringState/accepted***	Se marca el adelanto del documento como aceptado, *fact_accepted*, si se comprueba que el documento se encuentra aceptado, *accepted*, y estado del adelanto es solicitado, *fact_requested*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
POST /documents/{id}/factoringState/acceptedPlus ***POST /documents/factoringState/acceptedPlus***	Se marca el adelanto del documento como aceptado, fact_accepted, si se comprueba que el documento se encuentra aceptado, accepted, y estado del adelanto es solicitado, *fact_requested*. Además, se establecen los datos *factoringTotal*, *factoringExpirationDate y financialInstitutionName*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.

DELETE /documents/{id} ***DELETE /documents***	Se elimina el documento con el id dado. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
GET /documents/count	Devuelve el número de documentos almacenados.
GET /documents/index/{index}	Devuelve el *documentUniqueId* del documento con el índice dado.
GET /documents	Devuelve un *array* con todos los *documentUniqueId* almacenados.
GET /documents/{id}/<Dato> ***GET /documents/<Dato>***	Devuelve el dato correspondiente del documento. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.
GET /documents/{id}/all ***GET /documents/all***	Devuelve todos los datos de un documento. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista.

*Para evitar una repetición innecesaria no se listan todas las llamadas para obtener datos.

Tabla 3.5 Errores de la API

Error	**Descripción**
NOT_NET*WORK*_CONNECTION	Este error se produce cuando no hay conexión con la red.
ID_ALREADY_EXIST	Este error se produce cuando se intenta insertar un documento que genera un identificador que ya existe.
ID_NOT_EXIST	Este error se produce cuando identificador no existe.
DOCUMENT_FACTORING _NOT_ACCEPTED	Este error se produce cuando se quieren establecer datos del adelanto y este no se encuentra aceptado, *fact_accepted*.
DOCUMENT_NOT_PAID	Este error se produce cuando se quieren establecer datos del pago y el documento no está marcado como pagado, *paid*.
DOCUMENT_NOT_PENDING	Este error se produce cuando se quiere cambiar el estado de un documento ha aceptado, *accepted*, sin estar marcado como pendiente, *pending*, previamente.

DOCUMENT_NOT_ACCEPTED	Este error se produce cuando se quiere cambiar el estado de un documento a pagado, *paid*, sin estar marcado como aceptado, *accepted*, previamente.
DOCUMENT_NOT_FACTORING_PENDING	Este error se produce cuando se quiere cambiar el estado del adelanto ha solicitado, *fact_requested*, sin estar marcado como pendiente, *fact_pending*, previamente.
DOCUMENT_NOT_FACTORING_REQUESTED	Este error se produce cuando se quiere cambiar el estado del adelanto ha aceptado, *fact_accepted*, sin estar solicitado, *fact_requested*, previamente.

4 Análisis e implementación

En este capítulo se detallan los pasos seguidos para implementar la API y se razonan las decisiones tomadas.

4.1 Instalación del nodo, go *Ethereum* y *Mist*

La gestión del nodo de *Ethereum* se ha realizado a través del s*of*tware Go *Ethereum*, al que denominamos *Geth*. Este s*of*tware se puede descargar directamente de su página *of*icial *Geth*.*Ethereum*.org [22]. Pero también viene incluido en otro s*of*tware llamado *Mist*.

Mist es una *wallet* de *Ethereum* que, aparte de permitir enviar y recibir *ether* (Gas), nos da una interfaz gráfica que hace más cómodas muchas de las tareas a realizar, como desplegar el SC y comprobar su funcionamiento.

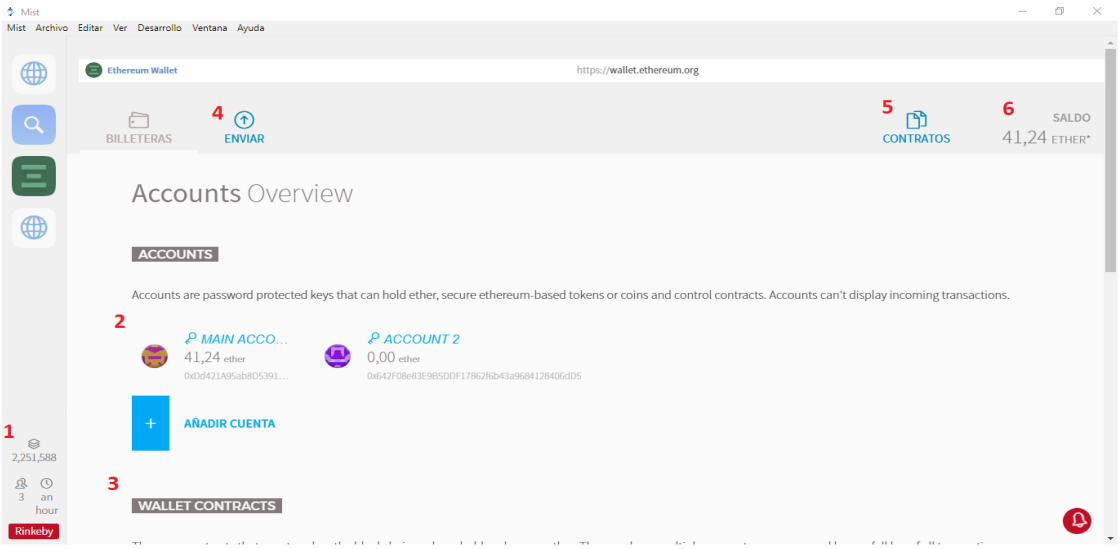
La primera vez que ejecutamos *Mist* se inicia el proceso de sincronizado con la *Blockchain*. Este proceso puede tardar días y llega a ocupar varios GB ya que se está descargando toda la cadena existente. Esta sincronización se puede interrumpir sin tener que reiniciar el proceso. En nuestro caso la sincronización del nodo tardo unas dos semanas, con interrupciones al tener que trasladar el equipo, y llegó a ocupar más de 40GB. Cabe destacar que el equipo no tiene las condiciones óptimas, como un disco *Solid State Drive* y una buena conexión a Internet en todo momento, que harían el proceso mucho más rápido.

Una vez sincronizado el nodo, *Mist* nos solicita una contraseña para generar nuestra cuenta. En un caso real esta contraseña debería ser lo más segura posible ya que controlaría el acceso a nuestro *ether*. Pero en nuestro caso optamos por una contraseña más sencilla y fácil de recordar, especialmente porque esta contraseña no se puede modificar en ningún caso y no hay forma de recuperarla. La cuenta se identifica con un hexadecimal generado por la *Blockchain*.

En nuestro caso la cuenta es *0xDd421A95ab8D53919092Cf2A144815905C2BC4Db* y la contraseña es *bleSurfu*.

En la figura 4.1 se muestra la ventana principal de *Mist*. En esta interfaz se muestra información sobre la *Blockchain* (1), gestionar nuestras cuentas (2), ver nuestros SC asociados (3), enviar *ether* a otras cuentas (4), desplegar e interactuar con SC (5) y consultar nuestro saldo en *ether* (6).

Destacar que en la sección de datos de la *Blockchain* donde se muestra el bloque en el que nos encontramos y la red en la que estamos, *Rinkeby*, también podemos cambiar nuestra red.



1 - Datos de la blockchain
2 - Gestion de Cuentas
3 - SC asociados
4 - Enviar ether
5 - Gestión de SC
6 - Saldo de ether

Figura 4.1 Interfaz principal Mist

Para obtener saldo de *ether*, necesario para realizar las distintas operaciones, en nuestra cuenta de *Ethereum* en la *testnet* debemos crear una publicación en Facebook, Twitter o Google+ con nuestra cuenta de *Ethereum* y copiar el enlace de la publicación en la Web de *Rinkeby* [16], nuestra *testnet*, como se observa en la Figura 4.2. Esto nos permite ingresar 3 *ether*s cada 8 horas, 7.5 *ether* cada día y 18.75 *ether* cada 3 días. Estas limitaciones de tiempo existen para impedir ataques contra la *testnet*. En nuestro caso usamos un post de Google+, servicio que ya no existe, y realizamos varias peticiones de 18.75 *ether*.

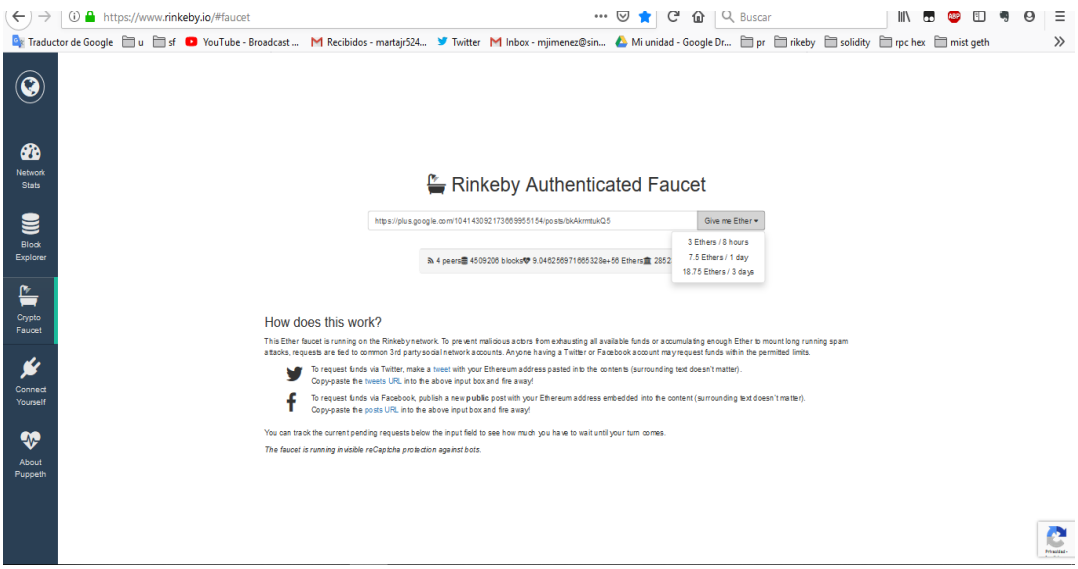


Figura 4.2 Obtener ether Rinkeby

Volviendo a *Geth*, cuando iniciamos *Mist* también se inicia *Geth*, pero para poder comunicarnos con la API *RPC* de *Ethereum* debemos activar unas opciones que no se activan por defecto con *Mist*. Por eso debemos lanzar *Geth* manualmente en la consola. La instrucción utilizada para lanzar *Geth* con las opciones que deseamos se puede ver en la instrucción siguiente y contiene las siguientes opciones:

```
geth --rinkeby --rpc -rpcapi="eth,net,personal"
```

- --*rinkeby*: indica que nos queremos conectar con la red *Rinkeby*.
- --*rpc*: indica que habilitarmos el servidor HTTP-*RPC*.
- -*rpcapi*="eth,net,personal": indica las APIs disponibles por *RPC*. "eth" para gestionar *ether*, "personal" para gestionar la autenticación de cuentas y "net" para comunicarnos con la red y poder llamar al SC.

Para consultar otras opciones de *Geth* debe consultarse su documentación [18].

En la figura 4.3 se muestra el lanzamiento de *Geth* en la consola. Aquí cabe destacar la URL base para realizar llamas a la API *RPC*, http://127.0.0.1:8345 o http://localhost:8345.

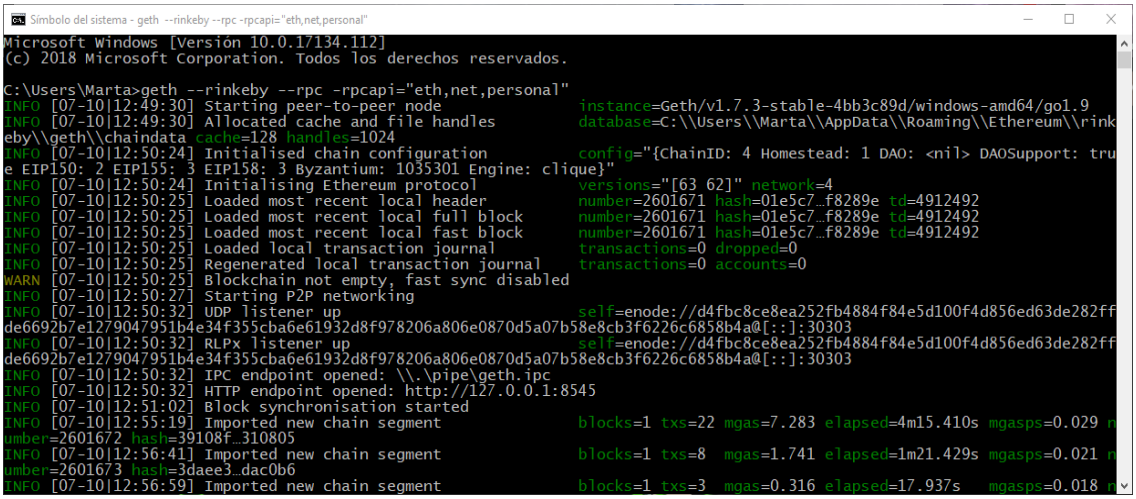


Figura 4.3 Lanzamiento del nodo con Geth

4.2 Creación del SC

El IDE que se ha utilizado para crear el SC ha sido *Remix* [15]. Este *Integrated Development Environment* (*IDE*) se encuentra en un entorno Web y permite comprobar si el código compila en *Solidity*. Además, *Remix* permite desplegar el SC y comprobar su funcionamiento, pero para estas tareas utilizamos *Mist*.

Nuestro SC se llama *DocumentCrud* y compila utilizado *Solidity* 0.4.4. Además, hereda de un SC denominado Mortal que da ciertas funcionalidades consideradas buenas prácticas. Los principales bloques de funcionabilidad forman el contrato son:

- Definir la estructura de un objeto document y el *array* que los contiene.
- Insertar un nuevo document.
- Comprobar si un documento ya existe.
- Borrar un documento.
- Obtener la cuenta de documentos totales.
- Obtener el identificador de un documento según su posición en el *array*.
- Comprobar el estado de un documento.
- Insertar determinada información en un documento.
- Cambiar el estado de un documento comprobando que el cambio es válido.
- Obtener información específica o toda la información de un documento.

4.2.1 Contrato Mortal

En primer lugar, exponemos el SC padre denominado Mortal. En este SC definimos una variable *owner* de tipo *address*. Este tipo identifica una dirección dentro de la red de *Ethereum*, esta dirección puede estar asociada a un SC o a una cuenta. A continuación, definimos el constructor del SC, esta función se ejecutaría automáticamente al desplegar el SC. En este caso el constructor asigna la dirección de la cuenta que desplego el SC a la variable owner.

Seguidamente se crea un modificador denominado *onlyOwner*. Un modificador es una función que se puede aplicar a otras funciones con el objetivo de no duplicar código. El modificador *onlyOwner* comprueba que la llamada a la función requerida proviene del dueño, variable owner, del SC antes de ejecutar dicha función. Este modificador se aplicaría a las funciones sobre las que queremos tener más control.

Finalmente creamos la función kill que permite eliminar el SC con el objetivo de liberar carga de la red en un futuro si determinamos que el SC no se utilizara nunca más. También podemos observar que se le aplica el modificador *onlyOwner*, por lo que solo puede llamar a esta función la cuenta que desplego el SC en primer lugar. La implementación de este SC se muestra en el bloque de código 4.1.

```
contract Mortal {
 address owner;

 function Mortal() public { owner = msg.sender; }
 modifier onlyOwner(){
 require(msg.sender == owner);
 _;
 }

 function kill() public onlyOwner { selfdestruct(owner);}
}
```

Bloque de código 4.1 Contrato Mortal

Estas funcionalidades, funciones limitadas al dueño y la habilidad de eliminar el SC, son útiles en cualquier SC. Esta es la razón de que se desarrollaran en un SC padre del que nuestro SC heredaría. Esta herencia se puede ven en el bloque de código 4.2

```
contract DocumentCrud is Mortal{}
```

Bloque de código 4.2 Herencia de contrato

4.2.2 Definir la estructura de datos

Lo primero que hacemos dentro de nuestro SC es definir una estructura de datos para almacenar documentos y definir el objeto que representa un documento.

Definimos un documento definiendo una estructura denominada DocumentStruct. Esta definición se hace estableciendo unas variables con unos tipos de datos determinados. Los tipos de datos que utilizamos son:

- **bytes<X>**: *Array* de bytes de tamaño fijo, este tamaño viene determinado por X, resulta mucho menos costoso en términos de gas que bytes. Sin embargo, solo permite almacenar hasta 32 B o caracteres (bytes32).
- **bytes**: *Array* de bytes de tamaño dinámico, resulta más costoso que bytes<X> pero como muchos de nuestros datos superan 32 B es necesario.
- **uint**: número entero no negativo en concreto de 256 b. Se pueden especificar enteros de menor tamaño en saltos de 8 b de igual manera que en bytes<X> con int<X>(int8,int16,int24...). Pero en nuestro caso no limitamos el tamaño.

Las variables que forman la estructura del documento se encuentran listadas en la tabla 4.1.

Destacar de entre estos datos la variable *dates*, que agrupa 3 fechas para reducir el número de parámetros necesarios para introducir un nuevo documento ya que *Solidity* limita el número de parámetros de una llamada. Y también hay que destacar la variable *index*, que no aporta información del documento, pero se utiliza para mantener la estructura de datos.

Tabla 4.1 Variables de la estructura de un documento

Nombre del dato	**Tipo de dato**	**Razonamiento**
documentUniqueId	bytes	Hash generado de 64 bytes, pese a tener un tamaño fijo debemos usar el *array* dinámico porque supera 32 bytes.
invoiceNumber	bytes	El tamaño máximo de este campo es de 100 caracteres, por lo que se usa el *array* dinámico.
total	bytes	El tipo de dato usado originalmente es *float* que no está soportado en *Solidity* y por lo tanto se usa bytes.
factoringTotal	bytes	El tipo de dato usado originalmente es *float* que no está soportado en *Solidity* y por lo tanto se usa bytes.
state	bytes8	Al haber solo ciertos valores permitidos se toma de tamaño máximo el tamaño del valor más largo, en este caso “accepted” con 8 caracteres.
currency	bytes3	Se almacena un identificador de la moneda formado por 3 caracteres (por ejemplo: EUR)
paymentType	bytes	El tamaño de este campo es de 50 caracteres, por lo que se usa el *array* dinámico.
supplierName	bytes	El tamaño de este campo es de 255 caracteres, por lo que se usa el *array* dinámico.
customerName	bytes	El tamaño de este campo es de 255 caracteres, por lo que se usa el *array* dinámico.

financialInstitutionName	bytes	El tamaño de este campo es de 255 caracteres, por lo que se usa el *array* dinámico.
factoringState	bytes14	Al haber solo ciertos valores permitidos se toma de tamaño máximo el tamaño del valor más largo, en este caso “fact_requested” con 14 caracteres.
paymentTerms	uint	El tipo de dato usado originalmente es *integer*.
paymentDate	bytes19	Fecha y hora que se almacena de la siguiente manera: AAA/MM/DD HH:MM:SS. Utilizando así 19 caracteres.
dates	bytes	En este campo se concatenan *fiscalYear*,*invoceDate y expirationDate* de la siguiente forma : FYFY-AAA/MM/DD HH:MM:SS-AAA/MM/DD HH:MM:SS. Al superarse 32 caracteres se debe utilizar bytes
factoringExpirationDate	bytes19	Fecha y hora que se almacena de la siguiente manera: AAA/MM/DD HH:MM:SS. Utilizando así 19 caracteres.
index	uint	Entero que indica la posición del ID en el *array*.

A continuación, se define un *mapping*, llamado `documentMapping`, que almacena el conjunto de documentos en una estructura de clave valor donde la clave es una variable de tipo byte, el identificador único del documento en nuestro caso, y el valor es un DocumentStruct. Este código se muestra en el bloque de código 4.3

Esto nos permite obtener un documento si conocemos su identificador, pero no nos permite obtener un listado de documentos. Para obtener este listado creamos un *array*, llamado `documentIndex`, que contiene la lista de identificadores existentes. Este *array* no contiene los documentos en sí ya que entonces sería necesario realizar un bucle, que podría ser costoso, para buscar documentos completos sin conocer la posición.

El objetivo de tener estas 2 estructuras, *mapping* y *array*, en lugar de sólo una de ellas es obtener una lista de los ID de los documentos y que no sea necesario realizar un bucle para buscar un documento concreto.

```
struct DocumentStruct {
 bytes documentUniqueId; //hash de 64 bytes
 bytes invoiceNumber; // tamaño maximo 100 char utf8
 bytes total; // float
 bytes factoringTotal; // float
 bytes8 state; // tamaño maximo de 8 caracteres(accepted)
 bytes3 currency; // 3 char utf8 (ISO)
 bytes paymentType; //50 char utf8
 bytes supplierName; // tamaño maximo 255 char utf8
 bytes customerName; // tamaño maximo 255 char utf8
 bytes financialInstitutionName; // tamaño maximo 255 char utf8
 bytes14 factoringState; // tamaño maximo de 14 (fact_requested)
 uint paymentTerms; // integer
 //fiscalYear,invoiceDate y expirationDate
 bytes dates; //mas de 32 caracteres (FYFY-AAAA/MM/DD HH:MM:SS-
AAAA/MM/DD HH:MM:SS)
 bytes19 factoringExpirationDate; // tamaño maximo de 19 (AAAA/MM/DD
HH:MM:SS)
 bytes19 paymentDate; // tamaño maximo de 19 (AAAA/MM/DD HH:MM:SS)

 uint index;
 }
 mapping(bytes => DocumentStruct) private documentMapping;

 bytes[] private documentIndex;
```

Bloque de código 4.3 Definir la estructura de datos del contrato

4.2.3 Insertar un nuevo document

A la función de insertar un documento se le pasan como parámetros 9 de las 16 variables que formar un documento. Las variables que faltan no se introducen en este paso por que *Solidity* limita el tamaño de esta llamada.

Dentro de las variables faltantes tenemos 3 casos:

- Los estados, estos se inicializan como pendientes sin necesidad de pasar un parámetro.
- La variable index que se asigna automáticamente al documento de forma que este se sitúa al final del *array*

- Variables sin valor en un nuevo documento, como la fecha de devolución de un préstamo que aun no se ha solicitado.

A esta función de insertar se le aplica el modificador *onlyOwner*, por lo que solo puede ser llamada por el dueño del contrato. La función también de vuelve el índice asignado al nuevo documento y antes de ejecutarse comprueba que no existe un documento con el mismo identificador único. La función se puede ver en el bloque de código 4.4

```
 function insertDocument(
 bytes documentUniqueId,
 bytes invoiceNumber,
 bytes total,
 bytes3 currency,
 bytes paymentType,
 bytes supplierName,
 bytes customerName,
 uint paymentTerms,
 bytes dates
 )
 public
 onlyOwner
 returns(uint index)
 {
 require(!exists(documentUniqueId));
 documentMapping[documentUniqueId].documentUniqueId =
documentUniqueId;
 documentMapping[documentUniqueId].invoiceNumber = invoiceNumber;
 documentMapping[documentUniqueId].total = total;
 documentMapping[documentUniqueId].currency = currency;
 documentMapping[documentUniqueId].paymentType = paymentType;
 documentMapping[documentUniqueId].supplierName = supplierName;
 documentMapping[documentUniqueId].customerName = customerName;
 documentMapping[documentUniqueId].paymentTerms = paymentTerms;
 documentMapping[documentUniqueId].dates = dates;
 documentMapping[documentUniqueId].state = "pending";
 documentMapping[documentUniqueId].factoringState = "fact_pending";
 documentMapping[documentUniqueId].index =
documentIndex.push(documentUniqueId)-1;
 return documentIndex.length-1;
 }
```

Bloque de código 4.4 función para insertar un nuevo documento

4.2.4 Comprobar si un documento ya existe

Se crea una función llamada *exists* que devuelve un booleano indicando si el identificador dado existe. Para comprobar la existencia de un documento con ese identificador se realizan 3 pasos:

1. Se encuentra ese documento en el *mapping* con su ID.
2. Se obtiene la variable *index* del documento que contiene su posición en el *array*.
3. Se comprueba si el identificador almacenado en esa posición coincide con el identificador utilizado para obtener el documento.

Se comprueba tanto en el *mapping* y en el *array* porque en un *mapping* no podemos borrar completamente elementos y determinamos si un elemento existe de verdad según su existencia en el *array*. Pero buscar directamente en el *array* puede ser muy costoso por que deberíamos iterar por un gran número de elementos. Por lo que se utiliza el *mapping* para obtener el documento y el *array* determinar si consideramos el documento como existente.

Además, para poder comparar los ID, debemos aplicarles *keccak256* porque *Solidity* no permite comparar variables del tipo byte. La función se puede ver en el bloque de código 4.5.

```
 function exists(bytes documentUniqueId)
 public
 constant
 returns(bool isIndeed)
 {
 if (documentIndex.length == 0) {
 return false;}
 return
(keccak256(documentIndex[documentMapping[documentUniqueId].index]) ==
keccak256(documentUniqueId));
 }
```

Bloque de código 4.5 Determinar la existencia de un documento

4.2.5 Borrar un documento

Esta función elimina un documento según su ID, pero como ya mencionamos, no podemos eliminar completamente documentos del *mapping*, por lo que los documentos que consideramos existentes son aquellos que se encuentran en el *array*. Por lo tanto, el proceso de borrar un documento es el proceso de eliminar un elemento del *array*.

Pero tampoco nos interesa dejar huecos vacíos en el *array*, por lo que el hueco dejado por el elemento eliminado se rellena copiando el último elemento del *array* y se actualiza la

variable *index* del documento copiado. Finalmente se reduce el tamaño del *array* en 1 eliminando el último documento del *array* que acabamos de mover y se encuentra duplicado.

Finalmente borramos el documento del *mapping*, pero esto no elimina el documento. Al aplicar *delete* a un elemento del *mapping* lo que hacemos es que deje de referenciarse por el *mapping* actual. Pero el elemento aún existe y puede referenciarse por antiguos bloques de la *Blockchain*. La función se puede ver en el bloque de código 4.6

```
function deleteDocument(bytes documentUniqueId)
public
onlyOwner
returns(uint index)
{
require(exists(documentUniqueId));
// borrar del array
uint rowToDelete = documentMapping[documentUniqueId].index;
bytes storage keyToMove = documentIndex[documentIndex.length-1];
documentIndex[rowToDelete] = keyToMove;
documentMapping[keyToMove].index = rowToDelete;
documentIndex.length--;
// borrar del mapping
delete documentMapping[documentUniqueId];
return rowToDelete;
}
```

Bloque de código 4.6 Borrar un documento

4.2.6 Gestión del *array*

Con respecto al *array* tenemos 2 funciones muy simples. Una de ellas devuelve el número total de documentos existentes y la otra devuelve el identificador almacenado en una posición dada del *array*. Las funciones se pueden ver en los bloques de código 4.7 y 4.8

```
function getDocumentAtIndex(uint
index)
public
constant
returns(bytes documentUniqueId)
{
return documentIndex[index];
}
```

Bloque de código 4.8 Obtener el identificador de un documento

```
function getDocumentCount()
public
constant
returns(uint count)
{
return documentIndex.length;
}
```

Bloque de código 4.7 Obtener la cuenta de documentos

4.2.7 Insertar información en un documento

A la hora de insertar información en un documento tenemos 2 casos. El primero de ellos consiste en asignar valor a alguna de las variables a las que no se le asignaba al insertar un nuevo documento. El segundo caso es cambiar el estado de un documento, donde este cambio despende del estado anterior para ser permitido o no.

En el primer caso se trata de una función setter sencilla, se pasan como parámetros el identificador del documento y el valor de la variable. Se asigna el valor a la variable y se devuelve un booleano a true en caso de éxito. La función se puede ver en el bloque de código 4.9.

```
 function setFactoringTotal(bytes documentUniqueId, bytes
factoringTotal)
 public
 onlyOwner
 returns(bool success)
 {
 require(exists(documentUniqueId));
 require(documentFactoringIsAccepted(documentUniqueId));
 documentMapping[documentUniqueId].factoringTotal = factoringTotal;
 return true;
 }
```

Bloque de código 4.9 Ejemplo de asignar valor a un atributo

En el segundo caso solo se requiere el identificador del documento y antes de realizar el cambio de estado se verifica que el estado anterior permite este cambio. En el ejemplo dado solo se permite cambiar el estado ha aceptado si anteriormente el estado era pendiente. La función se puede ver en el bloque de código 4.10

En ambos casos se comprueba la existencia del documento antes de ejecutarse y son funciones solo permitidas por el dueño del contrato.

```
function setStateAcceptedFromPending(bytes documentUniqueId)
 public
 onlyOwner
 returns(bool success)
 {
 require(exists(documentUniqueId));
 require(documentIsPending(documentUniqueId));
 documentMapping[documentUniqueId].state = "accepted";
 return true;
 }
```

Bloque de código 4.10 Ejemplo de cambio de estado

4.2.8 Obtener información de un documento

A la hora de obtener información de un documento también tenemos 2 casos. En el primer caso simplemente se pasa como parámetro el identificador del documento y se devuelve el valor de la variable. La función se puede ver en el bloque de código 4.11.

```
function getInvoiceNumber(bytes documentUniqueId)
public
constant
returns(bytes invoiceNumber)
{
require(exists(documentUniqueId));
return(documentMapping[documentUniqueId].invoiceNumber);
}
```

Bloque de código 4.11 Ejemplo de obtener el valor de una variable

En el segundo caso, el caso de los estados, no se devuelve el valor del estado. En su lugar existen una serie de métodos que devuelven un booleano indicando si un documento se encuentra en ese estado o no. Se ha hecho de esta forma por que las comprobaciones de estado son algo frecuente fuera y dentro del contrato y de esta forma se evita replicar el mismo condicional. La función se puede ver en el bloque de código 4.12

```
function documentIsPending(bytes documentUniqueId)
public
view
onlyOwner
returns(bool success)
{
require(exists(documentUniqueId));
bool result = false;
if((documentMapping[documentUniqueId].state == "pending")){
result = true;
}
return result;
}
```

Bloque de código 4.12 Ejemplo de comprobar el estado de un documento

4.3 Despliegue del SC

Para poder interactuar con el SC primero debemos desplegarlo en la *Blockchain*. Hay varias formas de hacer esto.

4.3.1 Despliegue con *Mist*

Para el despliegue de nuestro SC se ha utilizado la aplicación *Mist*. Existen varias opciones para desplegar SC, pero hemos optado por *Mist* debido a su comodidad en nuestro caso. Estas otras opciones se discuten la final de este apartado.

En la figura 4.4 se muestra la pantalla principal de *Mist*. En ella se muestran las cuentas y el *ether* que contienen. También podemos asociar SC existentes a nuestra *wallet* y manejar transacciones. Se muestra, en la esquina inferior izquierda, que nos encontramos en la red de pruebas *Rinkeby* y que estamos sincronizados. En caso de no estar sincronizados los bloques se presentan con un número negativo.

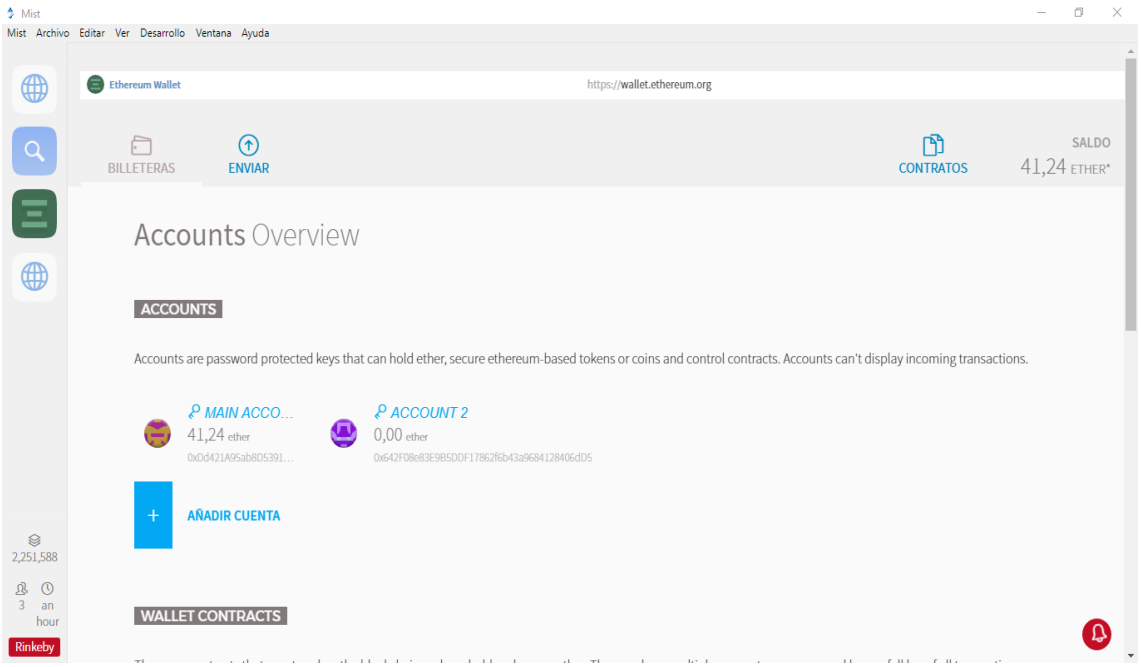


Figura 4.4 Pantalla principal de Mist

Si nos desplazamos a la pestaña de contratos (esquina superior derecha) llegamos a la pantalla mostrada en la figura 4.5 donde podemos ver nuestros contratos y la opción de desplegar un nuevo contrato.

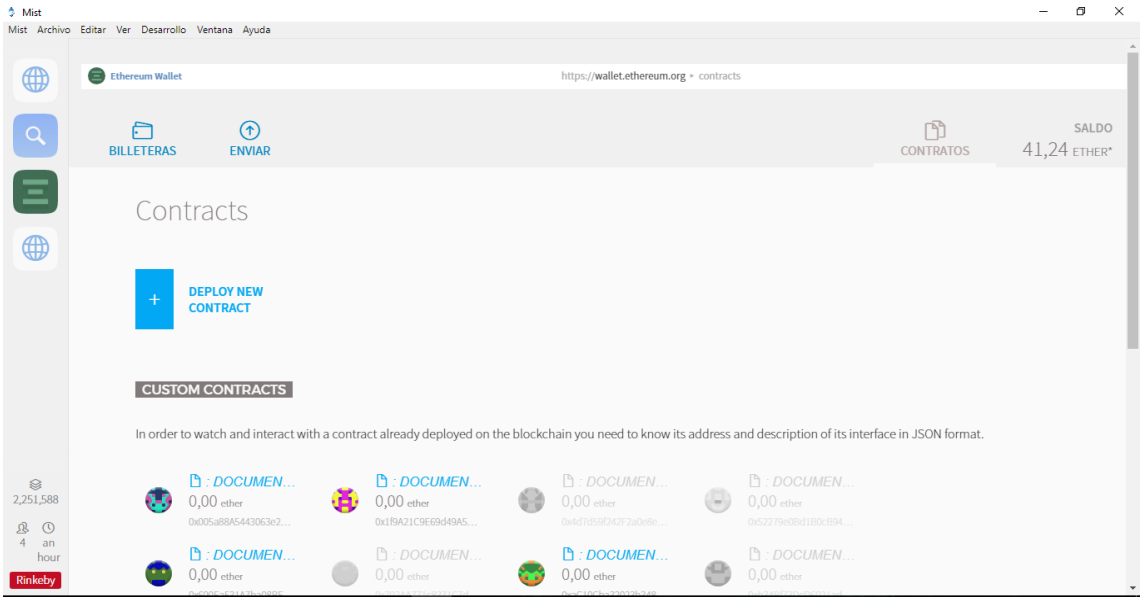


Figura 4.5 Pantalla contratos de Mist

En la figura 4.6 se muestra que al desplegar un contrato debemos dar una cuenta que pague por su despliegue. También podemos darle *ether* al contrato, pero en nuestro caso no es necesario que el contrato tenga *ether* propio.

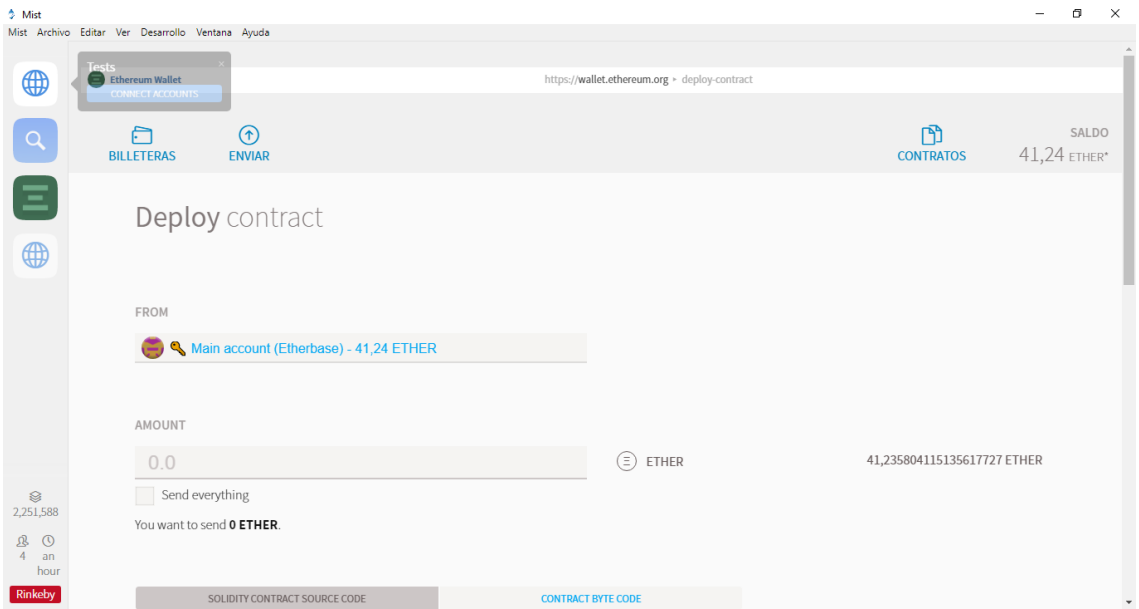


Figura 4.6 Pantalla desplegar contrato

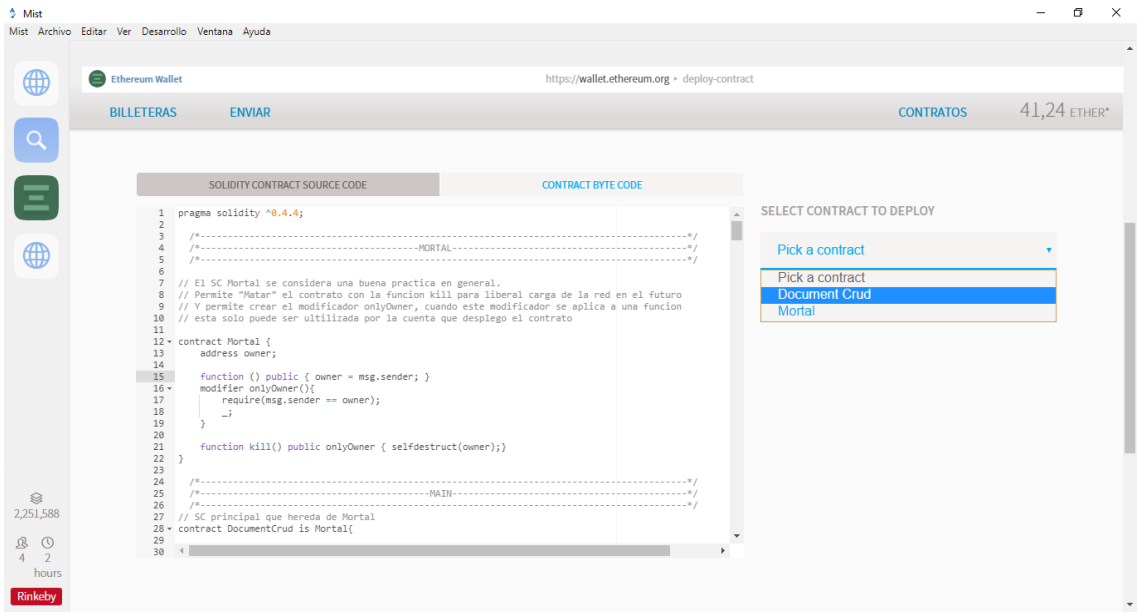


Figura 4.7 Pantalla introducir código del contrato

A continuación, debemos introducir el codigo de nuestro SC y seleccionar el contrato que queremos desplegar, *DocumentCrud* en nuestro caso como se observa en la figura 4.7. Si hubiera algún error de compilación se nos avisaría y no nos dejaría desplegar el SC.

A continuación, podemos elegir la cantidad de *ether* utilizada para desplegar el contrato. El despliegue utilizara la misma cantidad de gas, en cualquier caso, pero si elegimos pagar más por unidad de Gas es probable que nuestra transacción de despliegue se realice más rápido al resultar más atractiva a los mineros. Como no utilizamos *ether* real elegimos pagar una cuota alta como se ve en la figura 4.8.

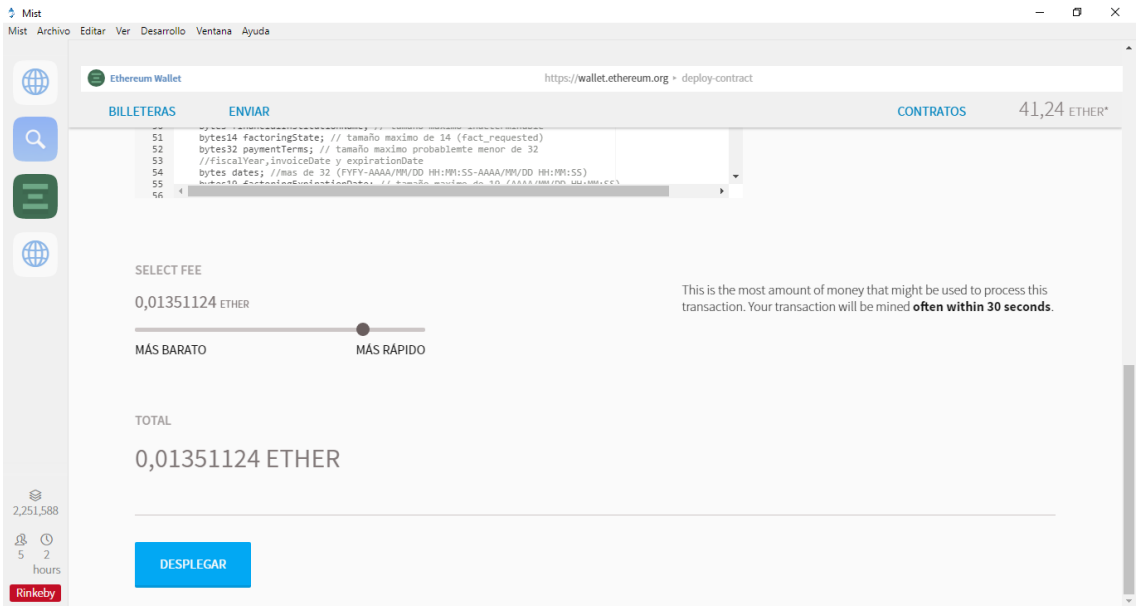


Figura 4.8 Pantalla cuota de despliegue

Tras pulsar *DESPLEGAR* se solicita que confirmamos la transacción pidiéndonos la contraseña. En la figura 4.9 se observa que se nos avisa que la transacción puede fallar usar mucho Gas debido al tamaño. Pero esto no sucede y el contrato se despliega con éxito.

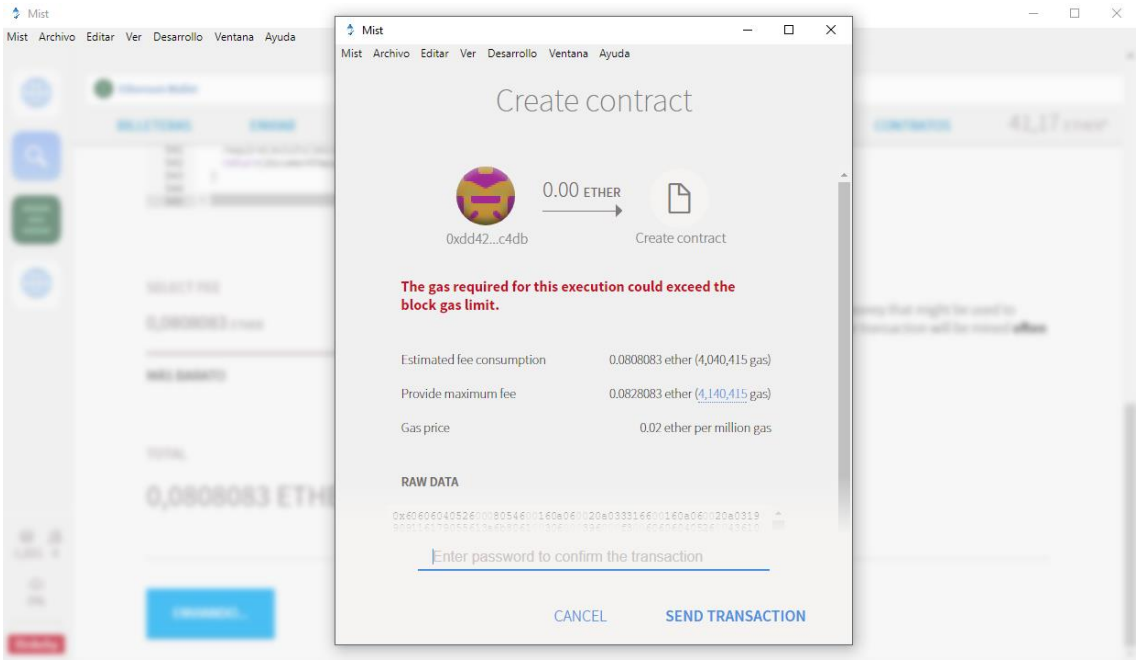


Figura 4.9 Pantalla confirmar despliegue

Si tras confirmar la transacción volvemos a la pantalla principal y vamos a la sección de transacciones podemos ver en la figura 4.10 que el contrato se está creando y se muestra difuminado. Se espera a que se confirme 12 veces para considerarse creado.

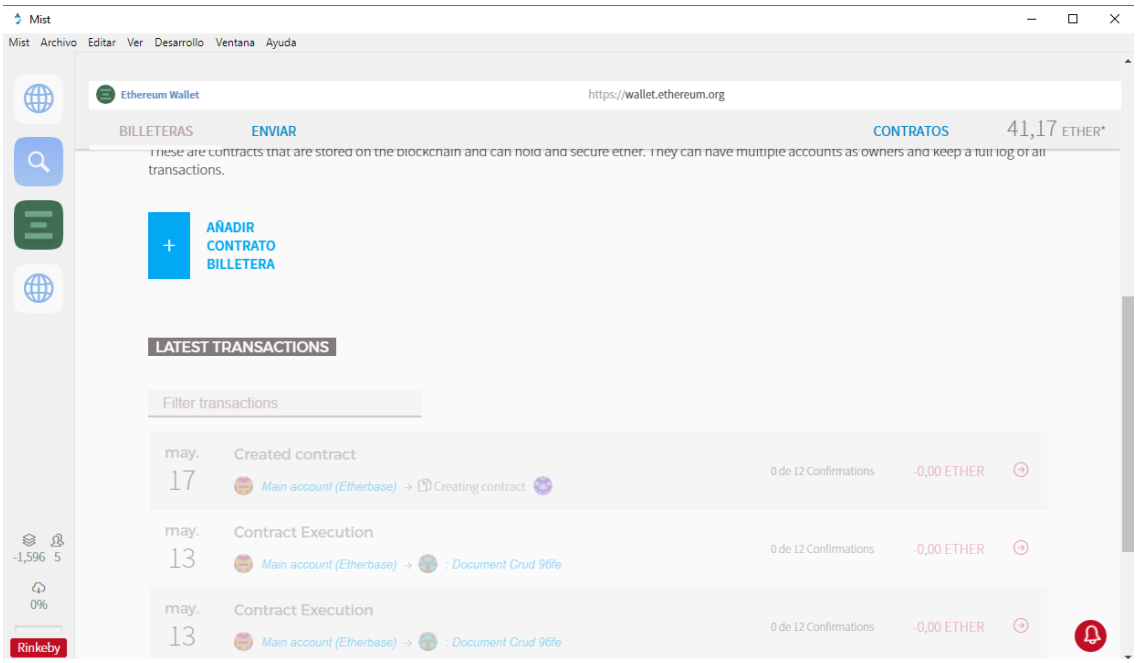


Figura 4.10 Pantalla creando contrato

Tras crearse el SC podemos acceder a él e interactuar con el cómo se ve en la figura 4.11. En la parte derecha de la captura se pueden ver las acciones disponibles. Estas acciones son: depositar *ether*, copiar la dirección del contrato, generar un código QR y ver la interfaz del contrato. En la parte inferior podemos interactuar con las funciones que programamos en el contrato como vemos en la figura 4.12.

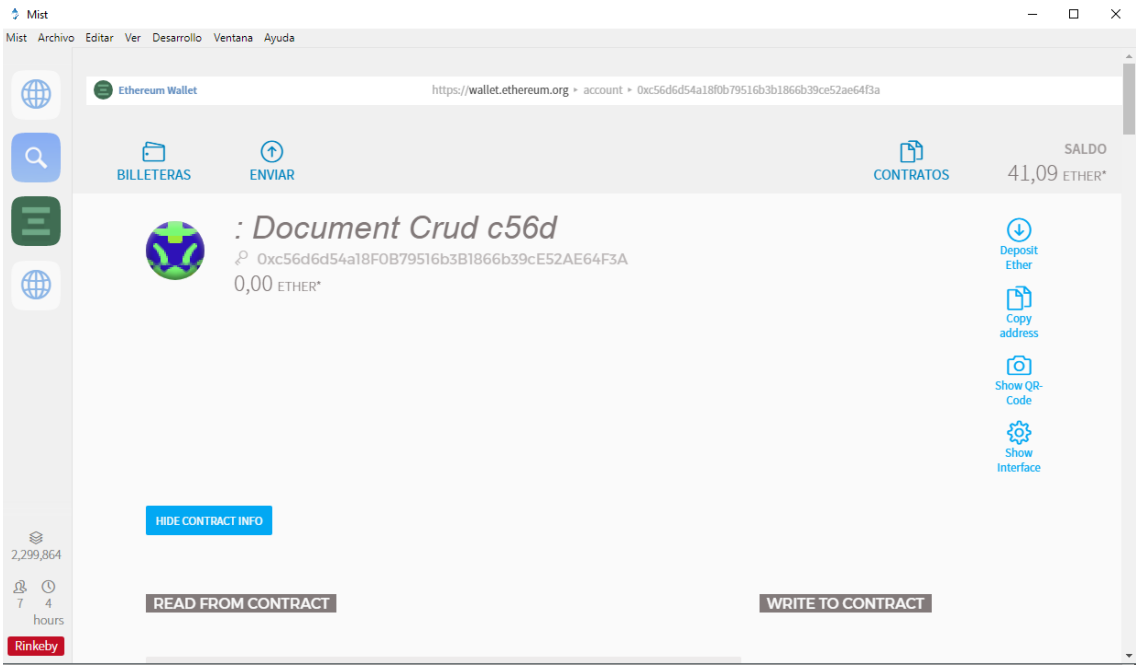


Figura 4.11 Pantalla contrato

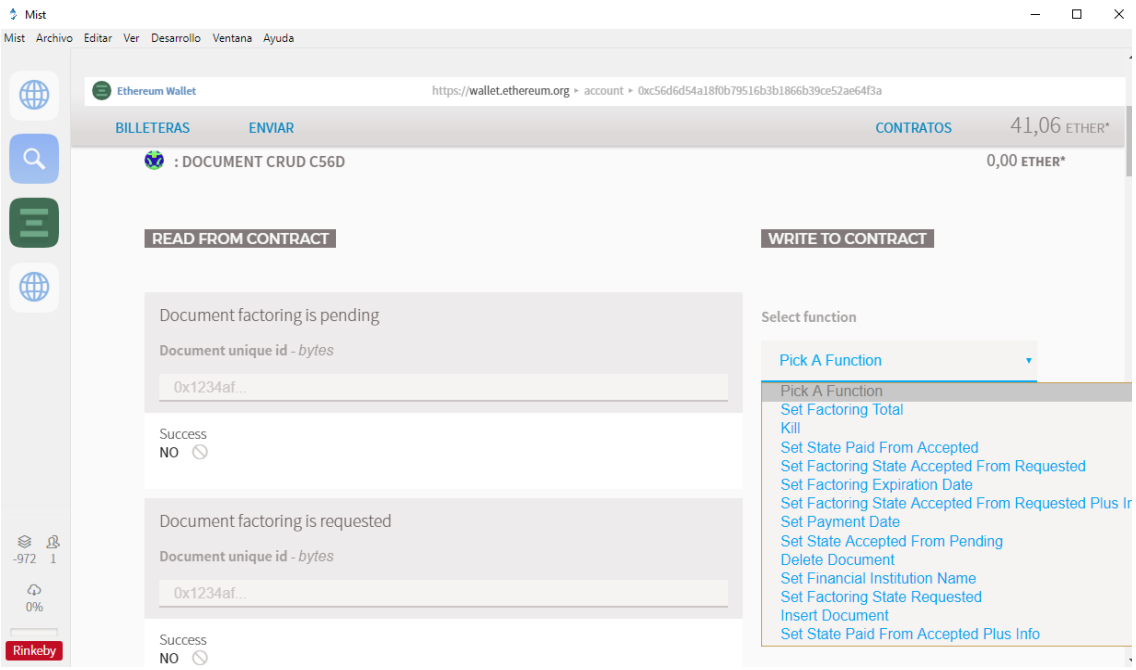


Figura 4.12 Pantalla funciones del contrato

Una de las acciones disponibles, ver la interfaz del contrato, requiere algo más de explicación. El resultado de esta acción se muestra en la figura 4.13.

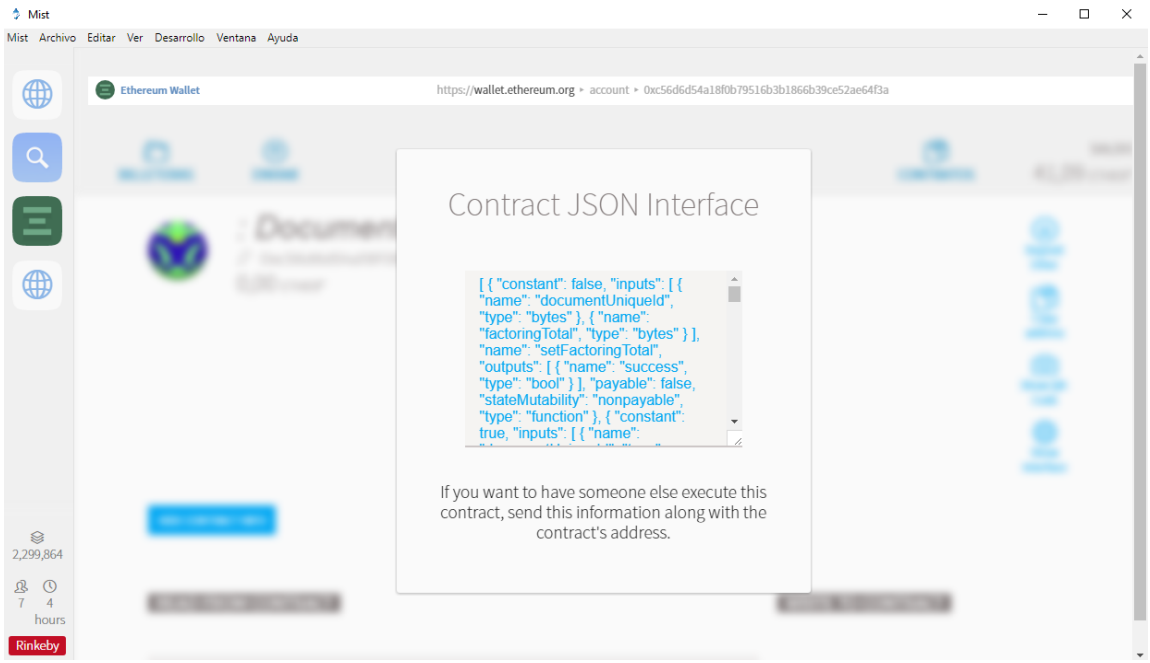


Figura 4.11 Pantalla para mostrar la interfaz

Este JSON define las funciones de nuestro SC. En nuestro caso no es necesario, pero si quisiéramos interactuar con nuestro contrato por otros medios, como la librería de *javascript* Web3, sería necesario para poder comunicarse con el SC.

4.3.2 Otros medios de despliegue

Otra forma de desplegar un contrato bastante cómoda es a través del IDE online de *Solidity Remix* [15]. La interfaz de *Remix* se muestra en la figura 4.14.

Al igual que en *Mist*, tenemos que introducir el código de nuestro SC y dar una cuenta que pagué por el despliegue. Pero en este caso tenemos el campo *Environment*. Este campo es para indicar cómo queremos desplegar el contrato. Hay tres opciones:

- *JavaScript VM*: si seleccionamos este *environment* no se llega a desplegar el SC. Esta opción se usa para probar el SC.
- *Injected Web3*: si seleccionamos este environment necesitamos el plugin *Metamask* para poder inyectar un acceso a la *Blockchain*.
- *Web3Provider*: si seleccionamos este environment necesitamos acceso a un nodo, si tenemos un nodo local podemos utilizar *http://localhost:8545*.

Estas dos formas comentadas resultan las más sencilla y *of*recen una interfaz gráfica. Pero fuera de estas podemos desplegar contratos mediante comandos con so*f*tware como *truffle* o el

propio *Geth*, o mediante código con Web3. Dentro de estas otras formas, de forma genérica, se han de seguir estos dos pasos

1. Compilar el código del contrato y obtener el bytecode y *Application Binary Interface* (*ABI*) [9]. para poder compilar el contrato es necesario tener instalado un compilador, en nuestro caso de *Solidity*.
2. Desplegar el contrato con el bytecode y la ABI.

Estos 2 pasos son genéricos. Dependiendo de cómo lo hagamos, el despliegue puede llevar varios comandos. Se mencionan estas otras formas más complejas, pese a no utilizarlas, para destacar su utilidad en el caso de *Blockchain* privadas.

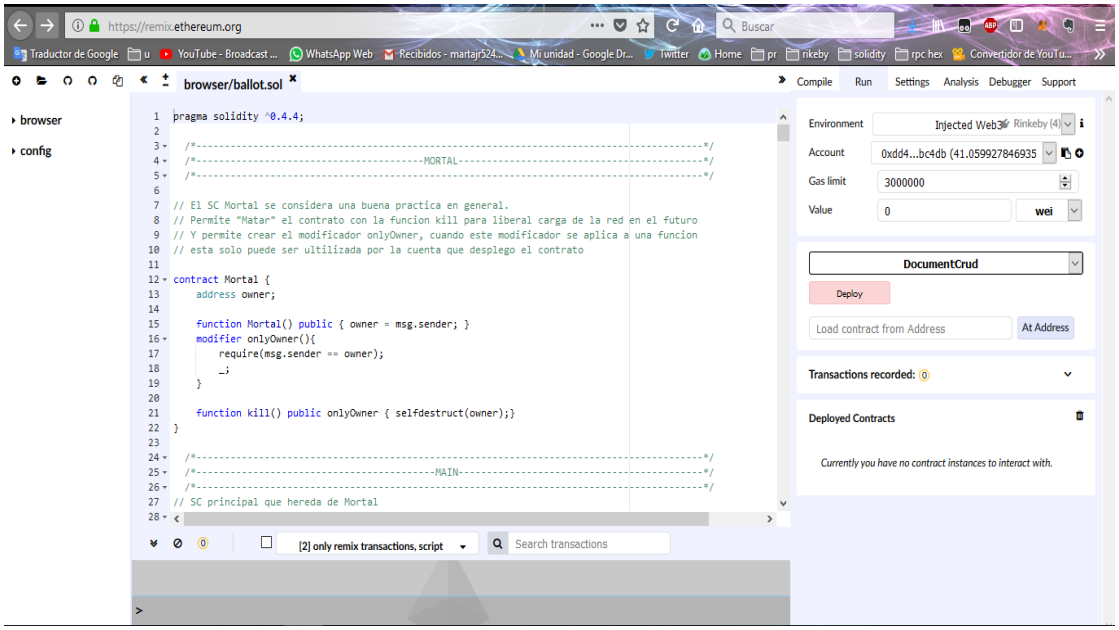


Figura 4.12 Captura de remix

4.4 Creación de la API

Symfony consiste en un conjunto de componentes de PHP y nos permite integrar más componentes a través *Composer*. Para instalar *Symfony 3.4* hemos utilizado el gestor de componentes para PHP *Composer*. La instrucción para instalar *Symfony* se puede ver en la siguiente Instrucción.

```
composer create-project symfony/framework-standard-edition
my_project_name "3.4.*"
```

Symfony utiliza un modelo Vista-Modelo-Controlador, pero en nuestro caso, al tratarse de una *API REST*, ignoramos la vista y solo trabajamos con el modelo y el controlador. El modelo gestiona los datos y el controlador gestiona la comunicación con el usuario de la API.

En nuestro caso hemos añadido 2 componentes a parte de los que vienen integrados en *Symfony*. El primero de estos componentes es *kornrunner/keccak* [24]. Este componente nos da la funcionalidad añadida de aplicar el *keccak 256*. El otro componente que añadimos es *friendsofSymfony/rest-bundle* [23] para facilitar la creación de rutas para las llamadas a nuestra API.

Para añadir estos componentes utilizamos la instrucción de composer *require* que añade los componentes al archivo *composer.JSON* e instala el componente o lo actualiza si ya hay una versión previa instalada. El archivo *composer.JSON* documenta los componente del proyecto de forma que si se instala el proyecto basta con ejecutar la instrucción composer *update* para obtener los componentes necesarios. La instalación de estos componentes se puede ver en la siguiente instrucción.

```
composer require friendsofsymfony/rest-bundle

composer require kornrunner/keccak
```

4.4.1 Modelo

Dentro de nuestro modelo tenemos 3 archivos, Utils.php, Access*Contract*.php y Model.php. Dentro de Utils tenemos una serie de funciones que se utilizaran frecuentemente y funciones para comunicarse con la *Blockchain* que no tienen relación con nuestro contrato. En Access*Contract* implementamos funciones para comunicarnos con nuestro contrato, esta comunicación requiere ciertas transformaciones a hexadecimal y un formato muy concreto. Finalmente, en Model utilizamos las funciones de Access*Contract* gestionando sus parámetros y realizamos control de errores.

Access*Contract* existe separado de Model en lugar de tener toda la funcionalidad en Model porque las transformaciones para poder comunicarnos con *Ethereum*, aunque algo repetitivas, resultan complejas y largas, por lo que harían el código de Model menos legible.

A continuación, entramos más en detalle en cada uno de los 3 componentes.

Utils

Lo primero que hacemos en Utils es definir 3 constantes: *URL*, *ACCOUNT* y CONTRACT. *URL* define la dirección donde está situado nuestro nodo de *Ethereum*, http://localhost:8545 en nuestro caso. *ACCOUNT* y CONTRACT contiene las direcciones hexadecimales a nuestra cuenta y a nuestro contrato, 0xDd421A95ab8D53919092Cf2A144815905C2BC4Db y 0xc56d6d54a18F0B79516b3B1866b39cE52AE64F3A, respectivamente. La declaración de estas constantes se puede ver en el bloque de código 4.13.

```
const URl="http://localhost:8545";
const ACCOUNT="0xDd421A95ab8D53919092Cf2A144815905C2BC4Db";
const CONTRACT="0xc56d6d54a18F0B79516b3B1866b39cE52AE64F3A";
```

Bloque de código 4.14 Constantes en Utils

A continuación, implementamos una serie de funciones que utilizan *cURL* para comunicarse con nuestro nodo de *Ethereum*.

La primera función a mencionar es *unlockAccount*. Esta llamada desbloquea nuestra cuenta de *Ethereum* y es necesario utilizarla antes de realizar cualquier operación que gaste Gas. La función se puede ver en el bloque de código 4.15.

Para hacer una llamada *cURL* utilizamos un handler al que configuramos utilizando los siguientes parámetros:

- *CURLOPT_URL*: URL hacia la que realizamos la llamada, en nuestro caso es la constante previamente definida como http://localhost:8545.
- *CURL*OPT_HTTPHEADER: indicamos las cabeceras de nuestra llamada. En nuestro caso simplemente indicamos que los parámetros de nuestra llamada son de tipo JSON
- *CURL*OPT_POST: marcando este campo a *true* indicamos que la llamada se realiza con el método HTTP POST.
- *CURL*OPT_POSTFIELDS: contiene los datos de nuestra llamada en formato JSON.
- *CURL*OPT_RETURNTRANSFER: marcando este campo a *true* indicamos que queremos recuperar el resultado real de la llamada. Si no lo indicamos recibiríamos *true* en caso de éxito y false en caso de fallo.

Estos parámetros de *cURL* son siempre los mismos en todas nuestras llamadas, con la excepción del valor de *CURL*OPT_POSTFIELDS que varía según nuestra llamada.

Los datos de nuestra llamada se configuran en un *array* que posteriormente se codifica como un JSON. Dentro de Este *array* hay 4 campos:

- json*rpc*: versión del servidor *RPC* del nodo, versión 2 en nuestro caso.
- *method*: método a invocar, en concreto *personal_unlockAccount*. Este método pertenece a la API personal.
- *params*: parámetros necesitados por el método, en ese caso, la dirección de la cuenta indicada en la constante, la contraseña *bleSurfu* y el el tiempo durante el que la cuenta se encontrara desbloqueada en segundos, 10 minutos en nuestro caso. Se ha puesto un tiempo mucho más largo del necesario debido a que el equipo en el que se ha ejecutado el proyecto puede llegar a ralentizarse considerablemente.
- *Id*: identifica la llamada que hemos hecho al servidor *RPC*. Pero como no nos interesa mantener un registro de estas llamadas, e implementar una gestión de identificadores añadiría una complejidad innecesaria en la que necesitaríamos una base de datos, siempre asignamos el ID 67 a nuestras llamadas.

La ejecución de *cURL* nos devuelve una respuesta en formato JSON que decodificamos y devolvemos como respuesta a nuestra función.

```
private function unlockAccount(){
 $data = [
 'jsonrpc'=>'2.0','method'=> 'personal_unlockAccount'
_,'params'=>[self::ACCOUNT,'bleSurfu',600],'id'=>67];

 $params= json_encode($data)
;
 $handler = curl_init();
 curl_setopt($handler, CURLOPT_URL, self::Url);
 curl_setopt($handler, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
 curl_setopt($handler, CURLOPT_POST,true);
 curl_setopt($handler, CURLOPT_POSTFIELDS, $params);
 curl_setopt($handler, CURLOPT_RETURNTRANSFER, true);
 $response = curl_exec ($handler);
 curl_close($handler);

 $json = json_decode($response, true);
 return $json;
}
```

Bloque de código 4.15 Función unlockAccount

Las siguientes 2 llamadas, *cURL*RequestCall y *cURL*RequestSendTransaction, tienen como objetivo llamar a las funciones del SC. *cURL*RequestCall lee de la *Blockchain* sin gastar Gas y se utiliza para las operaciones de solo lectura. Y *cURL*RequestSendTransaction si gasta Gas,

pero nos permite escribir en la *Blockchain* por lo que se utilizara para las funciones de escritura. Ambas funciones se muestran en los bloques de código 4.16 y 4.17. Ambas funciones reciben como parámetro un hexadecimal que representa la función del contrato y sus parámetros. Estos hexadecimales se generan en Access*Contract*. También mostramos que las funciones son extremadamente similares a la función *unlockAccount*. Los únicos cambios con respecto a esta llamada son los campos *method*; y *params* dentro de los datos de la llamada.

En el caso de la función *cURLRequestCall* el method es *eth_call*. Este método realiza una llamada al nodo sin crear una transacción por lo que solo pueden ser llamadas de lectura y no cuestan Gas. Los parámetros del método son *from* (dirección desde la que realiza la llamada), *to* (dirección del que recibe la llamada) y *data* (hexadecimal que define los detalles de la llamada al SC). Adicionalmente en este método se indica que la lectura se quiere realizar en el último bloque disponible de la cadena añadiendo *latest* al componente *params*.

```
function curlRequestCall($bytecode)
{
 $data = [
 'jsonrpc' => '2.0', 'method' => 'eth_call', 'params' => [[
 "from" => self::ACCOUNT, "to" => self::CONTRACT, "data" =>
$bytecode], 'latest'
 ], 'id' => 67
 ];

 $this->unlockAccount();
 $params = json_encode($data);
 $handler = curl_init();
 curl_setopt($handler, CURLOPT_URL, self::URl);
 curl_setopt($handler, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
 curl_setopt($handler, CURLOPT_POST, true);
 curl_setopt($handler, CURLOPT_POSTFIELDS, $params);
 curl_setopt($handler, CURLOPT_RETURNTRANSFER, true);
 $response = curl_exec($handler);

 curl_close($handler);
 $json = json_decode($response, true);
 return $json;
```

Bloque de código 4.16 Función cURLRequestCall

En el caso de la función *cURLRequestSendTransaction* el method es *eth_ sendTransaction*. Este método crea una transacción en el nodo y por lo tanto cuesta Gas, se utiliza para llamadas de escritura. Los parámetros del método son *from*, *to*, *gas* (*marca un límite de gas para la transacción*) y *data*. El límite de Gas existe para que si surge algún problema con nuestra función no se consuma una cantidad infinita de Gas y falle la transacción en su lugar. Hemos establecido un límite de 600000 Gas, 0x927c0 en hexadecimal. Este límite de Gas lo

hemos puesto basándonos en la operación que más Gas consume, insertar un nuevo documento. Esta operación consume algo menos de 400000 Gas.

```
function curlRequestSendTransaction($bytecode)
{

 $data = [
 'jsonrpc' => '2.0', 'method' => 'eth_sendTransaction', 'params' =>
[[
 "from" => self::ACCOUNT, "to" => self::CONTRACT, "gas" =>
"0x927c0", "data" => $bytecode]], 'id' => 67
 ];
 $params = json_encode($data);
 $this->unlockAccount();
 $handler = curl_init();
 curl_setopt($handler, CURLOPT_URL, self::URl);
 curl_setopt($handler, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
 curl_setopt($handler, CURLOPT_POST, true);
 curl_setopt($handler, CURLOPT_POSTFIELDS, $params);
 curl_setopt($handler, CURLOPT_RETURNTRANSFER, true);
 $response = curl_exec($handler);
 curl_close($handler);
 $json = json_decode($response, true);
 return $json;
}
```

Bloque de código 4.17 Función cURLRequestSendTransaction

En las 2 primeras llamadas la respuesta solo resultaba relevante en caso de error. Pero en esta tercera llamada la respuesta nos da un hexadecimal que identifica la transacción, este hexadecimal se puede utilizar para comprobar que la transacción ha tenido éxito.

Seguidamente tenemos otra agrupación de 3 funciones que tienen como objetivo comprobar el estado del nodo y de la conexión a éste:

1. *curlCheckNetConection*: comprueba que estamos conectado a la red usando el método net_listening. Se devuelve true o false (código 4.18).
2. *curlCheckNetSync*: determina si nos encontramos sincronizados con *Ethereum* y devuelve true o false usando el método eth_syncing (código 4.19).
3. curl*CheckTransaction*: utiliza el método eth_getTransactionReceipt para comprobar si una transacción ha tenido éxito o no (código 4.20).

```
function curlCheckNetConection()
{

 $data = [
 'jsonrpc' => '2.0', 'method' => 'net_listening', 'params' => [],
'id' => 67];
 $params = json_encode($data);
 $handler = curl_init();
 curl_setopt($handler, CURLOPT_URL, self::URl);
 curl_setopt($handler, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
 curl_setopt($handler, CURLOPT_POST, true);
 curl_setopt($handler, CURLOPT_POSTFIELDS, $params);
 curl_setopt($handler, CURLOPT_RETURNTRANSFER, true);
 $response = curl_exec($handler);
 curl_close($handler);
 $json = json_decode($response, true);
 $result=$json['result'];
 if($result){
 return true;
 }else {
 return false;
 }
}
```

Bloque de código 4.18 Función CurlCheckNetConection

```
function curlCheckNetSync()
{
 $data = [
 'jsonrpc' => '2.0', 'method' => 'eth_syncing', 'params' => [], 'id'
=> 67];
 $params = json_encode($data);
 $handler = curl_init();
 curl_setopt($handler, CURLOPT_URL, self::URl);
 curl_setopt($handler, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
 curl_setopt($handler, CURLOPT_POST, true);
 curl_setopt($handler, CURLOPT_POSTFIELDS, $params);
 curl_setopt($handler, CURLOPT_RETURNTRANSFER, true);
 $response = curl_exec($handler);
 curl_close($handler);
 $json = json_decode($response, true);
 $result=$json['result'];
 if(!$result){
 return true;
 }else {
 return false;
 }
}
```

Bloque de código 4.19 Función curlCheckNetSync

```
function curlCheckTransaction($hashTx)
{
 $data = [
 'jsonrpc' => '2.0', 'method' => 'eth_getTransactionReceipt',
'params' => [$hashTx], 'id' => 67
 ];
 $params = json_encode($data);
 $this->unlockAccount();
 $handler = curl_init();
 curl_setopt($handler, CURLOPT_URL, self::URl);
 curl_setopt($handler, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
 curl_setopt($handler, CURLOPT_POST, true);
 curl_setopt($handler, CURLOPT_POSTFIELDS, $params);
 curl_setopt($handler, CURLOPT_RETURNTRANSFER, true);
 $response = curl_exec($handler);
 curl_close($handler);
 $json = json_decode($response, true);
 $result=$json['result'];
 return $result;
}
```

*Bloque de código 4.20 Función curl*CheckTransaction

A continuación, tenemos 4 funciones que nos ayudaran a realizar las transformaciones a hexadecimal dentro de la clase Access*Contract*:

1. *lengthBytesOnBlocksOf32*: devuelve la longitud de un argumento hexadecimal en bloques de 32 b. Por ejemplo, un argumento que ocupe 33 b nos daría un resultado de 64 b. Esto se hace por que el hexadecimal que enviamos a *Ethereum* se divide en bloques de 32 B (código 4.21).

```
function lengthBytesOnBlocksOf32($argHex){
 $length = strlen($argHex)/2;
 $length32 = 0;
 $i = 0;
 while(true){
 if ($length <= 32*$i){
 $length32 = 32*$i;
 break;
 }else{$i++;}
 }
 return $length32;
}
```

Bloque de código 4.21 Función lengthBytesOnBlocksOf32

2. hexMethodSignature. Para identificar la función del contrato a la que queremos llamar primeros hemos de obtener la firma de esta. Esta firma consiste en los primeros 8 caracteres hexadecimales de un hash *keccak256*. Este hash se obtiene aplicándoselo a un *string* formado por el nombre de la función en el contrato y los tipos de sus parámetros de la siguiente forma: *<nombre de la*

función>(<tipo del parámetro 1>,<tipo del parámetro 2>). Por ejemplo, *setPaymentDate(bytes,bytes19)*. Obtenemos como parámetro el *string* con la función y sus parámetros, aplicamos *keccak256* y obtenemos los 8 primeros caracteres. También se añade *0x* al inicio para indicar que se trata de una cadena hexadecimal (código 4.22).

```
function hexMethodSignature($methodSignature){
 $hash = Keccak::hash($methodSignature, 256);
 $beginnningHash = substr ( $hash ,0, 8 );
 return "0x".$beginnningHash;
}
```

Bloque de código 4.22 Función hexMethodSignature

3. *stringTo*Hex: transforma de hexadecimal a *string*. Se recorre la cadena a trasformar con un bucle. En cada iteración coge un carácter y le aplica *ord* para obtener un número decimal que se corresponde con un carácter ASCII. Seguidamente este número decimas se pasa a hexadecimal *dechex* (código 4.23).

```
function stringToHex($string){
 $hex='';
 for ($i=0; $i < strlen($string); $i++){
 $hex .= dechex(ord($string[$i]));
 }
 return $hex;
}
```

Bloque de código 4.23 Función stringToHex

4. hex*ToString*: transforma de *string a* hexadecimal. Se recorre la cadena a trasformar con un bucle. cada iteración del bucle coge 2 caracteres y les aplica hexdec para obtener un numero decimal. A su vez a este número decimal le aplicamos *chr* para obtener el carácter que se corresponde con ese número en ASCII (código 4.24).

```
function hexToString($hex){
 $string='';
 for ($i=0; $i < strlen($hex)-1; $i+=2){
 $string .= chr(hexdec($hex[$i].$hex[$i+1]));
 }
 return $string;
}
```

Bloque de código 4.24 Función hexToString

Finalmente tenemos la función *generateId* que genera el ID de un documento. Para obtener este identificador concatenamos los atributos del documento *invoiceNumber*, *total*, *supplierName*, *customerName* y *fiscalYear* en un *string* y aplicando el hash *sha256* para obtener un hexadecimal (código 4.25).

```
function generateId($invoiceNumber, $total, $supplierName,
$customerName,$fiscalYear){
 $data=$invoiceNumber.$total.$supplierName.$customerName.$fiscalYear;
 return hash ( "sha256",$data,false );
}
```

Bloque de código 4.25 Función generateId

Access*Contract*

En esta clase generamos los distintos hexadecimales que utilizamos para comunicarnos con nuestro SC y *Ethereum* a través de las funciones de la clase Utils *cURL*RequestSendTransaction y *cURL*RequestCall. Como tenemos una gran cantidad de funciones, y el proceso de transformar a hexadecimal resulta algo repetitivo, aquí solo mostramos unos pocos ejemplos significativos.

Estos ejemplos son:

- Insertar un nuevo documento: mostramos una operación de escritura y cómo se tratan y organizan los distintos tipos de parámetros en la cadena hexadecimal a enviar.
- Comprobar el estado del documento: presentamos una operación de lectura que devuelve un argumento del tipo booleano.
- Obtener el índice del documento según su identificador: presentamos una operación de lectura que devuelve un argumento del tipo *array* de bytes.
- Obtener la cuenta de documentos: presentamos una operación de lectura que devuelve un argumento del tipo *integer*.

Antes de entrar en el código de las funciones, explicamos los bloques en los que se divide la cadena hexadecimal.

Empezamos declarando que es una cadena hexadecimal con *0x* y a continuación se añade la firma de la función del contrato. Y a continuación empezamos a definir los parámetros. Tenemos 2 tipos de parámetros, tipo dinámico y no dinámico.

Los parámetros de tipo dinámico son aquellos que no tienen una longitud definida por su tipo, un ejemplo de esto es bytes. Los parámetros de tipo no dinámico si tienen una longitud definida, como *byte10* que tiene una longitud de 10 B.

En la figura 4.15 podemos ver que los parámetros no dinámicos simplemente se añaden a continuación de la firma y se ocupa el resto del bloque de 32 con 0 desplazando el parámetro a la derecha.

En el caso de los dinámicos lo que se añade tras la firma es un *off*set, también desplazado a la derecha, que indica en que bloque de la cadena empieza a definirse el parámetro. En el caso del primer parámetro dinámico indicamos que empieza a definirse en 60 en hexadecimal, 96 en decimal, es decir a 3 saltos de 32 B desde el primer bloque de 32 tras la firma. El segundo parámetro está en A0, en 160 decimal, 5 saltos.

A su vez los parámetros de tipo dinámico se definen en 2 partes. En primer lugar, se ocupa un bloque indicando el tamaño del parámetro, desplazado a la derecha, y a continuación se declara el parámetro en sí. El parámetro puede ocupar más de un bloque y se encuentra desplazado a la derecha en lugar de a la izquierda.

En el ejemplo de la figura 4.15 podemos ver que el primer parámetro tiene un tamaño de 2 en hexadecimal y decimal. Esto indica que ocupa 2 B, por lo tanto 4 caracteres en hexadecimal. En el segundo parámetro indicamos un tamaño e 40, 64 en decimal, por lo tanto, ocupa 2 bloques completos de 32 B y está formado por 128 caracteres.

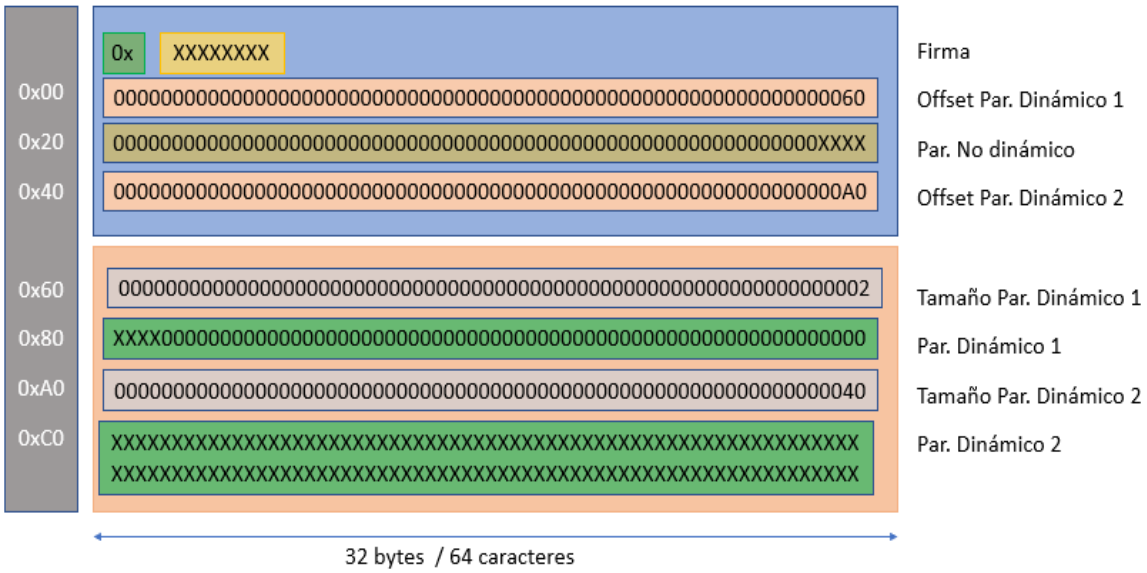


Figura 4.13 Diagrama hexadecimal

En la función de insertar un nuevo documento, *insertDocument*, tenemos los parámetros *documentUniqueId*, *invoiceNumber*, *total*, *currency*, *paymentType*, *supplierName*, *customerName*, *paymentTerms* y *dates*. La función devuelve el hash de la transacción (código 4.26).

De los 9 parámetros de la función 7 son de tipo dinámico y 2 de tipo no dinámico. Los 2 argumentos de tipo no dinámico son *currency* y *paymentTerms*. Estos parámetros son de tipo byte3 y uint256 (8 B). Los otros parámetros son de tipo bytes y no tienen una longitud definida.

Lo primero que hacemos en la función es obtener un objeto de la clase *Utils* para tener acceso a sus funciones y utilizar *hexMethodSignature* para obtener la firma del método, con *0x* incluido, con el *string* *insertDocument(bytes,bytes,bytes,bytes3,bytes,bytes,bytes,uint256,bytes)*. Seguidamente obtenemos el equivalente hexadecimal de los distintos parámetros.

A continuación, determinados la longitud de los argumentos dinámicos en 2 casos. En el primer caso se obtiene el tamaño en bytes dividiendo la longitud de la cadena hexadecimal por 2 y desplazando el resultado a la derecha. Este caso debe indicarse al principio de la declaración del parámetro. En el segundo caso se calcula la longitud que ocuparía en bloques de 32 B para calcular los distintos *offsets* de los parámetros dinámicos y completar los bloques.

A continuación, calculamos los distintos *offsets* de los parámetros. Para esto sumamos los primeros 9 bloques ocupados por los *offset* y los parámetros no dinámicos, más el bloque del tamaño del parámetro dinámico anterior y el parámetro anterior. Estos resultados se transforman a hexadecimal y se desplazan a la derecha.

Seguidamente se desplazan los argumentos no dinámicos a la derecha. Y los argumentos dinámicos se rellenan con 0 hasta ocupar bloques completos.

Finalmente se concatenan todos los hexadecimales. En primer lugar, tenemos la firma, luego tenemos loas *offsets* y los no dinámicos en el orden establecido y al final tenemos los distintos parámetros dinámicos emparejados con su tamaño. Este hexadecimal final se envía como parámetro a la función *curlRequestSendTransaction* de Utils para obtener la respuesta del nodo.

```
function insertDocument($documentUniqueId,$invoiceNumber,$total,
$currency,$paymentType, $supplierName,
 $customerName,$paymentTerms, $dates){
 $utils= new Utils();
 // firma del metodo
 $signature = $utils-
>hexMethodSignature("insertDocument(bytes,bytes,bytes,bytes3,bytes,
bytes,bytes,uint256,bytes)");
 // argumentos en hexadecimal
 $hexDocumentUniqueId = $utils-> stringToHex($documentUniqueId);
 $hexInvoiceNumber = $utils-> stringToHex($invoiceNumber);
 $hexTotal = $utils-> stringToHex($total);
 $hexCurrency = $utils-> stringToHex($currency);
 $hexPaymentType = $utils-> stringToHex($paymentType);
 $hexSupplierName = $utils-> stringToHex($supplierName);
 $hexCustomerName = $utils-> stringToHex($customerName);
 $hexPaymentTerms = dechex($paymentTerms);
 $hexDates = $utils-> stringToHex($dates);
 //tamaño de los argumentos dinamicos(bytes) sin dezplazar y
desplazado
 $rawHexDocumentUniqueIdLength = strlen($hexDocumentUniqueId)/2;
 $hexDocumentUniqueIdLengthPad =
str_pad(dechex($rawHexDocumentUniqueIdLength), 64, "0",
STR_PAD_LEFT);
 $rawHexInvoiceNumberLength = strlen($hexInvoiceNumber)/2;
 $hexInvoiceNumberLengthPad =
str_pad(dechex($rawHexInvoiceNumberLength), 64, "0", STR_PAD_LEFT);
 $rawHexTotalLength = strlen($hexTotal)/2;
 $hexTotalLengthPad = str_pad(dechex($rawHexTotalLength), 64, "0",
STR_PAD_LEFT);
 $rawHexPaymentTypeLength = strlen($hexPaymentType)/2;
 $hexPaymentTypeLengthPad =
str_pad(dechex($rawHexPaymentTypeLength), 64, "0", STR_PAD_LEFT);
 $rawHexSupplierNameLength = strlen($hexSupplierName )/2;
 $hexSupplierNameLengthPad =
str_pad(dechex($rawHexSupplierNameLength), 64, "0", STR_PAD_LEFT);
 $rawHexCustomerNameLength = strlen($hexCustomerName )/2;
 $hexCustomerNameLengthPad =
str_pad(dechex($rawHexCustomerNameLength), 64, "0", STR_PAD_LEFT);
 $rawHexDatesLength = strlen($hexDates)/2;
 $hexDatesLengthPad = str_pad(dechex($rawHexDatesLength), 64, "0",
STR_PAD_LEFT)
//tamaño de los argumentos dinamicos(bytes) en bloques de 32 bytes
 $hexDocumentUniqueIdLength32 = $utils-
>lengthBytesOnBlocksOf32($hexDocumentUniqueId);
 $hexInvoiceNumberLength32 = $utils-
>lengthBytesOnBlocksOf32($hexInvoiceNumber);
 $hexTotalLength32 = $utils->lengthBytesOnBlocksOf32($hexTotal);
 $hexPaymentTypeLength32 = $utils-
>lengthBytesOnBlocksOf32($hexPaymentType);
 $hexSupplierNameLength32 = $utils-
>lengthBytesOnBlocksOf32($hexSupplierName);
 $hexCustomerNameLength32 = $utils-
>lengthBytesOnBlocksOf32($hexCustomerName);
 $hexDatesLength32 = $utils->lengthBytesOnBlocksOf32($hexDates);
```

```
 $offsetDocumentUniqueID = str_pad(dechex(32*9), 64, "0",
STR_PAD_LEFT);//numero de argumentos
 $offsetInvoiceNumber =str_pad(dechex(32*9
+32+$hexDocumentUniqueIdLength32), 64, "0", STR_PAD_LEFT);
 $offsetTotal =str_pad(dechex(32*9 +32+$hexDocumentUniqueIdLength32
+32+$hexInvoiceNumberLength32), 64, "0", STR_PAD_LEFT);
 $offsetPaymentType =str_pad(dechex(32*9
+32+$hexDocumentUniqueIdLength32 +32+$hexInvoiceNumberLength32
+32+$hexTotalLength32), 64, "0", STR_PAD_LEFT);
 $offsetSupplierName =str_pad(dechex(32*9
+32+$hexDocumentUniqueIdLength32 +32+$hexInvoiceNumberLength32
+32+$hexTotalLength32 +32+$hexPaymentTypeLength32), 64, "0",
STR_PAD_LEFT);
 $offsetCustomerName =str_pad(dechex(32*9
+32+$hexDocumentUniqueIdLength32 +32+$hexInvoiceNumberLength32
+32+$hexTotalLength32
+32+$hexPaymentTypeLength32+32+$hexSupplierNameLength32), 64, "0",
STR_PAD_LEFT);
 $offsetDates =str_pad(dechex(32*9 +32+$hexDocumentUniqueIdLength32
+32+$hexInvoiceNumberLength32 +32+$hexTotalLength32
+32+$hexPaymentTypeLength32+32+$hexSupplierNameLength32
+32+$hexCustomerNameLength32), 64, "0", STR_PAD_LEFT);
 //argumentos no dinamicos desplazados
 $hexCurrencyPad = str_pad($hexCurrency, 64, "0");
 $hexPaymentTermsPad = str_pad($hexPaymentTerms, 64, "0",STR_PAD_LEFT);
 //argumentos dinamicos completados con ceros
 $hexDocumentUniqueIdFull = str_pad($hexDocumentUniqueId,
$hexDocumentUniqueIdLength32*2 , "0");
 $hexInvoiceNumberFull = str_pad($hexInvoiceNumber,
$hexInvoiceNumberLength32*2 , "0");
 $hexTotalFull = str_pad($hexTotal, $hexTotalLength32*2 , "0");
 $hexPaymentTypeFull = str_pad($hexPaymentType,
$hexPaymentTypeLength32*2 , "0");
 $hexSupplierNameFull = str_pad($hexSupplierName,
$hexSupplierNameLength32*2 , "0");
 $hexCustomerNameFull = str_pad($hexCustomerName,
$hexCustomerNameLength32*2 , "0");
 $hexDatesFull = str_pad($hexDates, $hexDatesLength32*2 , "0");

 $bytecode=$signature.
// argumentos no dinamicos y offsets de los dinamicos
$offsetDocumentUniqueID.$offsetInvoiceNumber.$offsetTotal.$hexCurrencyP
ad.$offsetPaymentType.
$offsetSupplierName.$offsetCustomerName.$hexPaymentTermsPad.$offsetDate
s.
//argumentos dinamicos
 $hexDocumentUniqueIdLengthPad.$hexDocumentUniqueIdFull.
 $hexInvoiceNumberLengthPad.$hexInvoiceNumberFull.
 $hexTotalLengthPad.$hexTotalFull.
 $hexPaymentTypeLengthPad.$hexPaymentTypeFull.
 $hexSupplierNameLengthPad.$hexSupplierNameFull.
 $hexCustomerNameLengthPad.$hexCustomerNameFull.
 $hexDatesLengthPad.$hexDatesFull;
 $answer = $utils->curlRequestSendTransaction($bytecode);
 return $answer['result'];
}
```

Bloque de código 4.26 Función insertDocument

A continuación, tenemos 3 funciones de lectura donde ya no analizamos la creación de nuestro hexadecimal, sino como interpretamos el resultado devuelto. Estas funciones son: *getDocumentCount*, *documentFactoringIsPending* y *getDocumentAtIndex*.

1. *getDocumentCount*: obtenemos un *integer* precedido por 0x y desplazado a la derecha con 0 a la izquierda. Eliminamos el 0x sabiendo que son los 2 primeros caracteres de la cadena y obtenemos el decimal del parámetro sin necesidad de eliminar los 0 ya que se encuentran a la izquierda. (código 4.27)

```
function getDocumentCount()
{
 $utils= new Utils();
 // firma del metodo
 $signature = $utils->hexMethodSignature("getDocumentCount()");
 $bytecode= $signature;
 $answer = $utils->curlRequestCall($bytecode);
 $result=$answer['result'];
 // ejemplo de resultado
 //0x
 //0000000000000000000000000000000000000000000000000000000000000004
 $argResult = substr($result, 2);// eliminar 0x
 return hexdec($argResult);
}
```

Bloque de código 4.27 Función getDocumentCount

2. *documentFactoringIsPending:* devuelve un booleano. El resultado de esta función es verdadero si se obtiene un 1 y falso si se obtiene 0. Simplemente obtenemos el último elemento de la cadena y comprobamos si es 1 o no. (código 4.28)

3. *getDocumentAtIndex*: obtenemos un parámetro de tipo dinámico como resultado. El resultado vendría precedido por 0x y se dividiría en 3 bloques. El primero de estos sería el *of*fset, que en nuestro caso siempre es 32 B ya que únicamente recibimos un parámetro a la vez. A continuación, tenemos el tamaño del parámetro desplazado a la derecha. Y finalmente el parámetro en sí.

   Para analizar la respuesta primero elimínanos el 0x y el *of*fset. Del resultado obtenemos los primeros 64 caracteres para obtener el tamaño del parámetro. Seguidamente obtenemos el parámetro sabiendo que empieza a continuación del carácter 64 y ocupa los caracteres del tamaño obtenido transformado a

hexadecimal multiplicado por 2. Finalmente se transforma el resultado de hexadecimal a *string* y se devuelve. (código 4.29)

```
function documentFactoringIsPending($documentUniqueId){
 $utils= new Utils();
 $signature = $utils-
>hexMethodSignature("documentFactoringIsPending(bytes)");
 $hexDocumentUniqueId = $utils-> stringToHex($documentUniqueId);
 $rawHexDocumentUniqueIdLength = strlen($hexDocumentUniqueId)/2;
 $hexDocumentUniqueIdLengthPad =
str_pad(dechex($rawHexDocumentUniqueIdLength), 64, "0",
STR_PAD_LEFT);
 $hexDocumentUniqueIdLength32 = $utils-
>lengthBytesOnBlocksOf32($hexDocumentUniqueId);
 $offsetDocumentUniqueID = str_pad(dechex(32*1), 64, "0",
STR_PAD_LEFT);
 $hexDocumentUniqueIdFull= str_pad($hexDocumentUniqueId,
$hexDocumentUniqueIdLength32*2 , "0");
$bytecode=$signature.$offsetDocumentUniqueID.$hexDocumentUniqueIdLe
ngthPad.$hexDocumentUniqueIdFull;
 $answer = $utils->curlRequestCall($bytecode);
 $result=$answer['result'];
 // ejemplo de resultado
 //0x
 //0000000000000000000000000000000000000000000000000000000000000001
 $arg = substr($result,65);//
 if ($arg=="1"){
 return true;
 }else{
 return false;
 }
 }
```

Bloque de código 4.28 Función documentFactoringIsPending

```
function getDocumentAtIndex($index){
 $utils= new Utils();
 $signature = $utils-
>hexMethodSignature("getDocumentAtIndex(uint256)");
 $indexPad =str_pad($index, 64, "0", STR_PAD_LEFT);
 $bytecode=$signature.$indexPad;
 $answer = $utils->curlRequestCall($bytecode);
 $result=$answer['result'];
 // ejemplo de resultado
 //0x
 //0000000000000000000000000000000000000000000000000000000000000020
 //0000000000000000000000000000000000000000000000000000000000000040
 //6636343436613831616566613436613165323537383535663262623935356563
 //3037313830616439366334376263383562386661663236353662393138343539

 $lenghtAndArgResult = substr($result,66);// string con la longitud
de la respuesta y la respuesta
 $lenghtResult= substr($lenghtAndArgResult,0,64);// logitud del
resultado
 $argResult=substr($lenghtAndArgResult,64,hexdec($lenghtResult)*2);
// argumento
 return $utils->HexToString($argResult);
}
```

Bloque de código 4.29 Función getDocumentAtIndex

Model

Dentro de la clase Model gestionamos las llamadas de Utils que comprueban la conexión a *Ethereum* y las llamadas de Access*Contract* que se comunican con nuestro contrato. Organizamos las respuestas de estas llamadas para hacerlas más explicitas añadimos control de errores.

En el caso de las funciones que devuelven el estado de la red no devolvemos simplemente un booleano true o false, sino que devolvemos un JSON que contiene un texto que describe más explícitamente el estado de la red.

En el caso de la función checkConection, bloque de código 4.30, devolvemos *CONNECTED* o *NOT_CONNECTED*. Y en el caso de checkSync, bloque de código 4.31, devolvemos *NODE_SYNC* o *NODE_NOT_SYNC*.

```
function checkConection(){
 $utils = new Utils();
 if($utils->curlCheckNetConection()){
 return ['status'=>'CONNECTED'];
 }else{
 return ['status'=>'NOT_CONNECTED'];
 }
}
```

Bloque de código 4.30 Función checkConection

```
function checkSync(){
 $utils = new Utils();
 if($utils->curlCheckNetSync()){
 return ['status'=>'NODE_SYNC'];
 }else{
 return ['status'=>'NODE_NOT_SYNC'];
 }
}
```

Bloque de código 4.31 Función checkSync

Nuestra siguiente función es *checkTransaction*, bloque de código 4.32. Esta función devuelve si una transacción ha tenido éxito o fallado con los textos *SUCCESS* o *FAILURE* y se devuelven otros parámetros de la transacción como el gas utilizado y el boque que la contiene. Además, en esta función, y en la mayoría de las otras funciones, se comprueba la conexión a la red antes de ejecutar el código principal. Si no hay conexión se devuelve el texto de error *NOT_NETWORK_CONNECTION*. También resultaría recomendable comprobar la sincronización del nodo, pero debido a las condiciones del equipo, no siempre se encuentra sincronizado, se

ha omitido este paso. Aunque en este caso no es completamente necesario encontrarse completamente sincronizado con la red al tratarse de una operación de lectura.

```
function checkTransaction($hashTx){
 $utils = new Utils();
 if($utils->curlCheckNetConection()){
 $result =$utils->curlCheckTransaction($hashTx);
 if($result['status']=='0x1') {
 $status = 'SUCCESS';
 }else{
 $status = 'FAILURE';
 }
 $hashTx=$result['transactionHash'];
 $gasUsed=hexdec(trim(substr($result['gasUsed'],2),"0"));
 $blockNumber=hexdec(trim(substr($result['blockNumber'],2),"0"));
 $blockHash=$result['blockHash'];
 return
['status'=>$status,'hashTx'=>$hashTx,'gasUsed'=>$gasUsed,'blockNumbe
r'=>$blockNumber,'blockHash'=>$blockHash];
 }else{
 return ['error' =>'NOT_NETWORK_CONNECTION'];
 }
}
```

Bloque de código 4.32 Función checkTransaction

A continuación, tenemos la función *insertNewDocument*, código 4.33. Aquí podemos ver que utilizamos *generateId* de *Utils* para obtener el ID a partir de los parámetros *invoiceNumber*, *total*, *supplierName*, *customerName* y *fiscalYear*. Si el identificador ya se encuentra utilizando la función *exists* se devuelve el texto de error *ID_ALREADY_EXIST*.

A la hora de insertan el documento podemos ver que fusionamos los parámetros *fiscalYear*, *invoiceDate* y *expirationDate* para obtener una variable dates que es la que almacenamos en el contrato. Recordamos que esto se debe a las limitaciones de *Solidity* en el número de parámetros.

Finalmente, la función devuelve el identificador generado, el hash de la transacción y los parámetros utilizados para generar el identificador.

A continuación, tendríamos una serie de funciones que comprueban la existencia y los estados del documento. La función de ejemplo que utilizamos para este grupo de funciones es *exists*, bloque de código 4.34. En estas funciones comprobamos la conexión y devolvemos el identificador y los parámetros que lo generan.

También introducimos otra funcionalidad. En caso de no conocerse el identificador este se puede volver a generar con los parámetros si la función recibe un identificador vacío.

```
function insertNewDocument($invoiceNumber,$fiscalYear,$total
,$currency,$paymentType,$supplierName,$customerName,$paymentTerms,
 $invoiceDate,$expirationDate){
 $contract = new AccessContract();
 $utils = new Utils();
 if($utils->curlCheckNetConection()){

 $documentUniqueId = $utils->generateId($invoiceNumber, $total,
$supplierName, $customerName,$fiscalYear);
 if(!($contract->exists($documentUniqueId))){
 $dates = $fiscalYear . "-" . $invoiceDate . "-" . $expirationDate;
//(FYFY-AAAA/MM/DD HH:MM:SS-AAAA/MM/DD HH:MM:SS)
 $hashTx = $contract->insertDocument($documentUniqueId,
$invoiceNumber, $total, $currency, $paymentType,
 $supplierName, $customerName, $paymentTerms, $dates);
 return ['documentUniqueId' => $documentUniqueId, 'hashTx' =>
$hashTx,'invoiceNumber'=>$invoiceNumber

,'total'=>$total,'supplierName'=>$supplierName,'customerName'=>$cust
omerName,"fiscalYear"=>$fiscalYear];
 }else{
 return ['documentUniqueId' => $documentUniqueId,'error'
=>'ID_ALREADY_EXIST',

'invoiceNumber'=>$invoiceNumber,'total'=>$total,'supplierName'=>$sup
plierName,'customerName'=>$customerName,"fiscalYear"=>$fiscalYear];
 }

 }else{
 return ['error' =>'NOT_NETWORK_CONNECTION'];
 }
}
```

Bloque de código 4.33 Función insertNewDocument

```
 function exists($id,$invoiceNumber, $total, $supplierName,
$customerName,$fiscalYear){
 $contract = new AccessContract();
 $utils = new Utils();
 if($utils->curlCheckNetConection()){
 if(($id=="")){
 $documentUniqueId = $utils->generateId($invoiceNumber, $total,
$supplierName, $customerName,$fiscalYear);
 }else{
 $documentUniqueId = $id;
 }
 $answer = $contract->exists($documentUniqueId);
 return ['documentUniqueId' => $documentUniqueId,"exists" =>
$answer,
'invoiceNumber'=>$invoiceNumber,'total'=>$total,'supplierName'=>$sup
plierName,'customerName'=>$customerName,"fiscalYear"=>$fiscalYear];
 }else{
 return ['error' =>'NOT_NETWORK_CONNECTION'];
 }
 }
```

Bloque de código 4.34 Función exists

A continuación, tenemos una serie de funciones que asignan valores a los atributos de un documento. Si permitimos realizar esta asignación depende del estado en el que se encuentra el documento. En caso de no encontrarse en el estado adecuando se devolvería un texto de error comunicando en qué estado se necesitaría estar para poder ejecutar la llamada. Estos podrían ser: *DOCUMENT_FACTORING_NOT_ACCEPTED*, *DOCUMENT_NOT_PAID*, *DOCUMENT_NOT_PENDING*, *DOCUMENT_NOT_ACCEPTED*, *DOCUMENT_NOT_FACTORING_PENDING* y *DOCUMENT_NOT_FACTORING_REQUESTED*.

Dentro de este grupo de funciones tenemos 3 casos. En el primero caso insertaríamos un valor a un atributo como *financialInstitutionName* en la función que requiere que el préstamo esté aceptado. En el segundo caso cambiaríamos uno de los estados del documento como pasar del préstamo solicitado al préstamo aceptado en la función.

En el tercer caso fusionaríamos la funcionalidad de los otros 2 casos, insertando un dato y cambiando el estado en una misma función, un ejemplo de este caso es la función *setFactoringStateAcceptedFromRequestedPlusInfo*, bloque de código 4.35.

Estas funciones devuelven el identificador único y sus parámetros, el hash de la transacción y el dato o datos insertados en el caso necesario. También podemos ver que se comprueba la existencia del documento y si este no existe se devuelve el texto de error *ID_NOT_EXIST*.

Seguidamente tenemos una serie de funciones utilizadas para gestionar la lista de documentos. Una de estas funciones devuelve la cuenta de los documentos, otra función da el identificador de un documento según su índice. Y una tercera función que devuelve la lista de identificadores de los documentos, función *getAllDocumentId*, bloque de código 4.36.

Finalmente tenemos una serie de funciones que obtienen datos del documento. Dentro de estas funciones tenemos un grupo de funciones que solamente devuelve un dato, como la función. Pero también tenemos una función que devuelve todos los datos de un documento llamada *getAll* y que podemos ver en el bloque de código 4.37. Estas funciones devuelven el identificador y sus parámetros: generador y el dato, o datos, solicitado.

Al final de esta sección se incluye la tabla 4.2 con los posibles textos de error devueltos por el modelo.

```
Function setFactoringStateAcceptedFromRequestedPlusInfo(
$id,$invoiceNumber, $total, $supplierName,$customerName,$fiscalYear,
$factoringTotal,$factoringExpirationDate,$financialInstitutionName){
 $contract = new AccessContract();
 $utils = new Utils();
 if($utils->curlCheckNetConection()){
 if(($id=="")){
 $documentUniqueId = $utils->generateId($invoiceNumber, $total,
$supplierName, $customerName,$fiscalYear);
 }else{
 $documentUniqueId = $id;
 }
 if($contract->exists($documentUniqueId)){
 if($contract->documentFactoringIsRequested($documentUniqueId)){
 $hashTx = $contract-
>setFactoringStateAcceptedFromRequestedPlusInfo($documentUniqueId,$f
actoringTotal, $factoringExpirationDate, $financialInstitutionName);
 return ['documentUniqueId' =>$documentUniqueId,
'hashTx'=>$hashTx,'factoringTotal'=>$factoringTotal
 ,'factoringExpirationDate'=> $factoringExpirationDate,
'financialInstitutionName'=> $financialInstitutionName,
'invoiceNumber'=>$invoiceNumber,'total'=>$total,'supplierName'=>$sup
plierName,'customerName'=>$customerName,"fiscalYear"=>$fiscalYear];
 }else{
 return ['documentUniqueId' => $documentUniqueId,'error'
=>'DOCUMENT_NOT_FACTORING_REQUESTED',
'invoiceNumber'=>$invoiceNumber,'total'=>$total,'supplierName'=>$sup
plierName,'customerName'=>$customerName,"fiscalYear"=>$fiscalYear];
 }
 }else{
 return ['documentUniqueId' => $documentUniqueId,'error'
=>'ID_NOT_EXISTS',
'invoiceNumber'=>$invoiceNumber,'total'=>$total,'supplierName'=>$sup
plierName,'customerName'=>$customerName,"fiscalYear"=>$fiscalYear];
 }
 }else{
 return ['error' =>'NOT_NETWORK_CONNECTION'];
 }
}
```

Bloque de código 4.35 Función setFactoringStateAcceptedFromRequestedPlusInfo

```
function getAllDocumentId(){
 $contract = new AccessContract();
 $utils = new Utils();
 if($utils->curlCheckNetConection()){
 $count =$contract->getDocumentCount();
 $data =array();
 for ($i=0;$i<$count;$i++){
 $id=$contract->getDocumentAtIndex($i);
 array_push($data,$id);
 }
 return $data;
 }else{
 return ['error' =>'NOT_NETWORK_CONNECTION'];
 }
}
```

Bloque de código 4.36 Función getAllDocumentId

```
function getAll($id,$invoiceNumber, $total, $supplierName,
$customerName,$fiscalYear){
 $contract = new AccessContract();
 $utils = new Utils();
 if($utils->curlCheckNetConection()){
 if(($id=="")){
 $documentUniqueId = $utils->generateId($invoiceNumber, $total,
$supplierName, $customerName,$fiscalYear);
 }else{
 $documentUniqueId = $id;
 }
 if($contract->exists($documentUniqueId)){
 return ['documentUniqueId' => $documentUniqueId,
 'InvoiceNumber'=> $contract->getInvoiceNumber($documentUniqueId),
 'Total'=> $contract->getTotal($documentUniqueId),
 'FactoringTotal'=> $contract-
>getFactoringTotal($documentUniqueId),
 'State'=> $contract->getState($documentUniqueId),
 'Currency'=> $contract->getCurrency($documentUniqueId),
 'PaymentType'=> $contract->getPaymentType($documentUniqueId),
 'SupplierName'=> $contract->getSupplierName($documentUniqueId),
 'CustomerName'=> $contract->getCustomerName($documentUniqueId),
 'CustomerName'=> $contract->getCustomerName($documentUniqueId),
 'FinancialInstitutionName'=> $contract-
>getFinancialInstitutionName($documentUniqueId),
 'FactoringState'=> $contract-
>getFactoringState($documentUniqueId),
 'PaymentTerms'=> $contract->getPaymentTerms($documentUniqueId),
 'FiscalYear'=> substr($contract->getDates($documentUniqueId),0,4),
 'InvoiceDate'=> substr($contract-
>getDates($documentUniqueId),5,19),
 'ExpirationDate'=> substr($contract-
>getDates($documentUniqueId),25,19),
 'FactoringExpirationDate' => $contract-
>getFactoringExpirationDate($documentUniqueId) ,
 'PaymentDate' => $contract->getPaymentDate($documentUniqueId),
'invoiceNumber'=>$invoiceNumber,'total'=>$total,'supplierName'=>$su
pplierName,'customerName'=>$customerName,"fiscalYear"=>$fiscalYear]
;
 }else{
 return ['documentUniqueId' => $documentUniqueId,'error'
=>'ID_NOT_EXISTS',

'invoiceNumber'=>$invoiceNumber,'total'=>$total,'supplierName'=>$su
pplierName,'customerName'=>$customerName,"fiscalYear"=>$fiscalYear]
;
 }
 }else{
 return ['error' =>'NOT_NETWORK_CONNECTION'];
 }
}


function getAll($id,$invoiceNumber, $total, $supplierName,
$customerName,$fiscalYear){
 $contract = new AccessContract();
 $utils = new Utils();
 if($utils->curlCheckNetConection()){
 if(($id=="")){
```

Bloque de código 4.37 Función getAll

Tabla 4.2 Errores modelo

Error	**Descripción**
NOT_NET*WORK*_CONNECTION	Este error se produce cuando no hay conexión con la red.
ID_ALREADY_EXIST	Este error se produce cuando se intenta insertar un documento que genera un identificador que ya existe.
ID_NOT_EXIST	Este error se produce cuando identificador no existe.
DOCUMENT_FACTORING _NOT_ACCEPTED	Este error se produce cuando se quieren establecer datos del adelanto y este no se encuentra aceptado.
DOCUMENT_NOT_PAID	Este error se produce cuando se quieren establecer datos del pago y el documento no está marcado como pagado.
DOCUMENT_NOT_PENDING	Este error se produce cuando se quiere cambiar el estado de un documento ha aceptado sin estar marcado como pendiente previamente.
DOCUMENT_NOT_ACCEPTED	Este error se produce cuando se quiere cambiar el estado de un documento a pagado sin estar marcado como aceptado previamente.
DOCUMENT_NOT_FACTORING _PENDING	Este error se produce cuando se quiere cambiar el estado del adelanto ha solicitado sin estar marcado como pendiente previamente.
DOCUMENT_NOT_FACTORING _REQUESTED	Este error se produce cuando se quiere cambiar el estado del adelanto ha aceptado sin estar solicitado previamente.

4.4.2 Controlador

Nuestro controlador consiste en un único archivo PHP llamado *AccessContractController.php*. Dentro de este archivo se establece como deben ser las llamadas de la API y se implementan estas llamadas llamando a su vez al modelo. Nuestro controlador simplemente recoge los parámetros que llegan en la llamada de la API y los gestiona para pasárselos al modelo. Tras esto devuelve la respuesta del modelo.

Para configurar las rutas y el método HTTP utilizado en cada llamada utilizamos la funcionalidad del componente *friendsofSymfony/rest-bundle* para utilizar anotaciones. También podríamos configurar las rutas en un archivo a parte usando los formatos *xml* o *yaml*, pero consideramos más sencillo utilizar las anotaciones.

Como las funciones del controlador no tienen casi ninguna funcionalidad solo mostramos 3 ejemplos para mostrar un ejemplo de llamada POST y GET y ver como actuamos en caso de obtener el identificador del documento o los datos que lo generan.

Nuestro primer ejemplo es la función *insertDocument*, bloque de codigo 4.38. En la función podemos observar que lo único que se hace es recoger los parámetros y pasárselos al modelo. También vemos en la anotación que configuramos la ruta en */documents* y que debemos usar el método POST para utilizarla.

```
/**
 * @Rest\Post("/documents")
 */
function insertDocument(Request $request){
 $model= new Model();
 $invoiceNumber = $request->get('invoiceNumber');
 $fiscalYear = $request->get('fiscalYear');
 $total = $request->get('total');
 $currency =$request->get('currency');
 $paymentType=$request->get('paymentType');
 $supplierName = $request->get('supplierName');
 $customerName = $request->get('customerName');
 $paymentTerms=$request->get('paymentTerms');
 $invoiceDate=$request->get('invoiceDate');
 $expirationDate=$request->get('expirationDate');
 return $model-
>insertNewDocument($invoiceNumber,$fiscalYear,$total,$currency,$pay
mentType,

$supplierName,$customerName,$paymentTerms,$invoiceDate,$expirationD
ate);
}
```

Bloque de código 4.38 Función insertDocument

Nuestros siguientes 2 ejemplos son *exists* y *existsData*, ambas usan el método GET. La diferencia entre ellas depende de si conocemos el identificador o los parámetros que lo generan.

En *exists*, bloque de código 4.39, conocemos el identificador del documento y este se incluye en la ruta de esta forma */documents/<id>/exists*. Al modelo le pasamos este identificador y los parámetros los pasamos vacíos.

```
/**
 * @Rest\Get("/documents/{id}/exists")
 */
function exists($id){
 $model= new Model();
 return $model->exists($id,"","","","","");
}
```

Bloque de código 4.39 Función exists

En cambio, en *existsData*, bloque de código 4.40, no conocemos el ID. La ruta pasaría a ser */documents/exists* y los parámetros se pasan en el cuerpo de la llamada. Al modelo le pasaríamos los parámetros y el identificador vacío.

```
/**
 * @Rest\Get("/documents/exists")
 */
function existsData(Request $request){
 $model= new Model();
 $invoiceNumber = $request->get('invoiceNumber');
 $total = $request->get('total');
 $supplierName = $request->get('supplierName');
 $customerName = $request->get('customerName');
 $fiscalYear = $request->get('fiscalYear');
 return $model->exists("",$invoiceNumber ,$total,
$supplierName,$customerName,$fiscalYear);
}
```

Bloque de código 4.40 Función existsData

El resto de las funciones se compartan de igual forma, obtienen los parámetros de la llamada y llaman al modelo usando el ID o sus parámetros generadores dependiendo del caso.

En el Anexo de este documento se puede consultar una lista completa de las llamadas de la API, con ejemplos de la llamada y de la respuesta.

5 Evaluación

En este capítulo se evalúa el rendimiento de la API y se considera si nuestra solución es válida para el problema planteado.

5.1 Rendimiento de la API

Para comprobar el funcionamiento de la API hemos usado el s*oft*ware *Postman*. Una captura de la interfaz de dicho s*oft*ware se puede ver la figura 5.1.

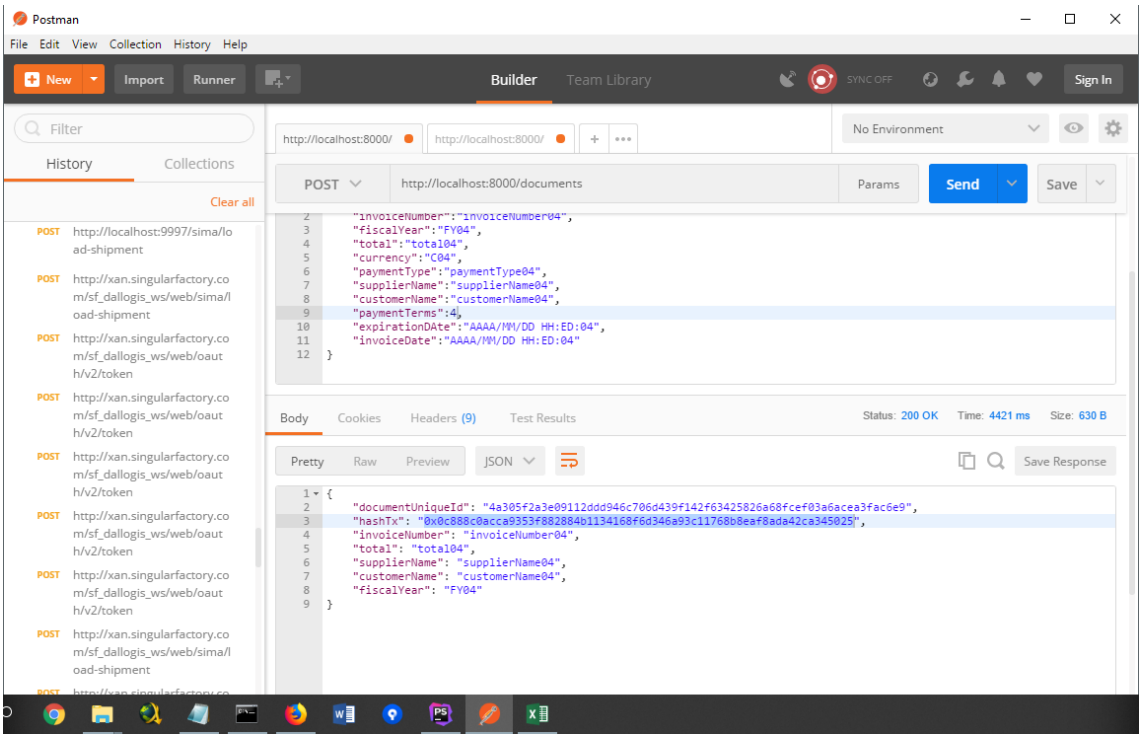


Figura 5.1 Captura interfaz Postman

En la parte superior de la captura se muestra que estamos realizando una llamada POST a la ruta *http://localhost:8000/documents* para insertar un documento. Debajo se puede ver el JSON que pasamos como parte de la llamada.

En la mitad inferior de la pantalla mostramos la respuesta de la API, en este caso un JSON. En la esquina superior derecha de esta sección podemos ver información sobre la respuesta. En concreto podemos ver el tiempo de respuesta en milisegundos, el tamaño de la respuesta y su estado.

El estado se corresponde con los códigos de estado HTTP donde 200 indica un a petición correcta y 500 un error en el servidor.

En nuestro caso el tamaño de la respuesta no resulta una información muy relevante, ya que nuestras respuestas siempre son JSON de pocos bytes. Por lo tanto, este aspecto no se analiza en profundidad. La mayor respuesta, de 1 KB, se da cuando se solicitan todos los datos de un contrato.

El aspecto que más analizamos es la velocidad de respuesta. Para esto hemos capturado 10 medidas de tiempo por llamada y hemos obtenido un promedio con una gran margen de error ya que las medidas pueden llegar a ser muy dispares. estas medidas se pueden observar en la tabla 5.1.

Las medidas se han realizado alojando la API, el nodo de *Ethereum* y Postman en un portátil Lenovo modelo G50-80 con 8 GB de RAM, sistema operativo Windows 10, un procesador Intel i5-5200U 2.20GHz y 4 años de uso. Se ha utilizado este equipo ya que el nodo requiere mucho almacenamiento, 40 GB actualmente, y no se ha encontrado ningún servidor que permita alojarlo de forma gratuita. Debido a esto los tiempos que se obtienen son mayores de los que se podrían obtener en un servidor y este análisis se centra más en la diferencia de tiempo entre llamadas y comprobar que estas diferencias son coherentes con las gestiones realizadas.

A continuación, enumeramos las llamadas por orden decente de tiempo de llamada y mencionamos las gestiones que realizan las llamadas para explicar las diferencias en tiempos.

- Con 1300 y 1500 milisegundos, las llamadas que menos tardan en responder son aquella que comprueban la conexión con el nodo. Esto es debido a que estas llamadas no gestionan datos ni realiza ninguna comprobación previa.
- Con 2500 ms, la siguiente llamada es aquella que devuelve el identificador de un documento según su índice. Esta llamada recibe solamente un dato y devuelve solo también solo uno, índice e identificador respectivamente. No comprueba la existencia del índice, pero si comprueba la conexión de la red.
- Con 3000 ms tenemos 4 llamadas. Estas llamadas son: comprobar la existencia de un documento, comprobar el estado de una transferencia, devolver un dato de un documento y devolver el número de documentos almacenados. Todas estas llamadas comprueban la conexión con la red y reciben un dato, el ID o el hash de la transferencia. En las 2 llamadas que devuelven datos también se comprueba la existencia del identificador, esto se equilibra en la llamada de la transferencia ya que esta resulta más compleja y lenta por parte del nodo. La llamada para comprobar la existencia debería tener un menor tiempo de respuesta y el promedio obtenido podría ser equivocado al haber una medida, de 4598 ms, mucho mayor que las otras. Como ya se ha mencionado, el equipo donde se realizan estas medidas no es ideal y pueden darse discrepancias.
- Con 4000 ms, tenemos la llamada que elimina un documento. Esta llamada comprueba la red y la existencia del documento, y recibe el identificador. Tiene

un mayor tiempo de respuesta la ser una llamada que interactúa con un método más complejo por parte del contrato.

- Las siguientes llamadas, con 4200 ms, son aquellas que comprueba los estados de un documento. Estas llamadas reciben el identificador y comprueban la red y la existencia del documento. Al igual que con eliminar, el tiempo es mayor al ser funciones más complejas por parte del contrato.
- Con 4500 ms, tenemos la llamada que inserta un nuevo documento. Esta llamada recibe varios parámetros y comprueba la red y la no existencia del documento. También tiene una mayor complejidad por parte el contrato.
- Con 5000 ms a 5500 ms tenemos las llamadas que establecen datos ligados al adelanto y al pago una vez que se comprueban los estados correspondientes. Y las llamadas que cambian los estados del documento. Estas llamadas comprueban la red, la existencia del documento y los estados correspondientes. Esta comprobación de los estados incrementa el tiempo de respuesta. Esta diferencia de 500 ms se puede deber a las condiciones del equipo.
- Con 6000 ms tenemos las llamadas que cambian los estados del documento y establecen datos. la diferencia en tiempo con el caso anterior se debe a que, al establecer los datos, el método en el contrato es más complejo.
- Con 9400 ms tenemos la llamada que devuelve la lista de identificadores existentes. esta medida se realizó con 6 documentos. Si hubiera más documentos el tiempo sería aún mayor. Y este tiempo es grande debido a que primero ha de buscarse el número de documento almacenados y luego ha de recuperarse el identificador de cada documento según su índice por separado
- Con 26200 ms, la llamada más lenta, tenemos la llamada que devuelve todos los datos de un documento. este tiempo se debe a que ha realizarse una llamada al contrato por cada uno de los 16 dato que forman un documento ya que el contrato no permite devolverlos todos en una única llamada.

Se comprueba que los tiempos de respuesta son coherentes con las operaciones realizadas por las llamadas pese a no contar con un equipo ideal.

Tabla 5.1 Tiempos de respuesta de la API

Método	Tiempos (ms)										Promedio
checkConnection	1814	1221	1147	1129	1635	1547	1325	1798	1664	1327	1500
checkSync	1223	1446	1658	1419	1226	1215	1398	1227	1465	1189	1300
transaction	3131	3395	3362	2516	3051	2957	2896	3045	3267	3369	3000
insertDocument	5096	4985	4254	4448	4679	5124	5045	4378	4225	5102	4500
exists	3355	4598	2653	2647	2813	3165	2796	2857	2963	3095	3000
IsPending	3893	4339	3896	4474	4229	3884	3981	3928	4434	3987	4200
IsAccepted	3987	4178	3861	3856	4106	4209	3745	3952	4312	4265	4200
IsFactPending	3954	4025	4006	4189	4304	4157	3985	3756	4013	3985	4200
IsFactRequest	3852	3965	4108	3857	4256	3875	4005	4256	3856	3898	4200
IsFactAccepted	3975	3798	3965	4185	3907	4324	4212	4189	4265	3902	4200
IsPaid	3804	4048	3922	3998	4135	4298	4326	3864	3987	4306	4200
setFactoringTotal	4986	4897	5104	4986	5124	4853	4965	5268	4789	5007	5000
SetExpirationDate	5264	4986	5136	5251	4865	4789	5109	4768	5126	4851	5000
setFinancial InstituitionName	4867	5026	4756	5478	4852	4982	5031	5127	5067	4967	5000
setPaymentDate	5145	4793	5089	4985	5268	5147	4856	4786	4894	5106	5000
SetAccepted	5278	5774	6445	4964	5522	5648	4899	5328	4932	5439	5500
setPaid	5782	4968	5487	5687	5706	5464	4992	6023	5781	5678	5500
setPaidPlus	6443	5469	5631	5863	5781	6215	6205	6089	6147	5879	6000
setFactRequest	6549	5395	5522	5948	5362	4987	5028	6124	6083	5641	5500
setFactAccepted	5479	4932	5367	4998	5689	4998	5521	5643	6048	5347	5500
setFactAcceptedPlus	6461	5494	5597	5432	5708	5723	5393	6098	5459	5846	6000
delete	3756	4159	4059	4256	3978	4187	3956	4124	4089	3856	4000
count	3372	2856	2509	2789	2897	3105	2654	3264	2864	3021	3000
getAtIndex	2948	2729	2424	2568	2527	2147	2985	2142	2017	2442	2500
getAllIds	9305	9677	9431	9258	9748	9564	9157	9756	9651	9267	9400

getDato	2894	3158	2984	2985	3147	3025	3164	2986	2874	3050	3000
getAll	28919	26207	26283	26247	26057	26178	26205	26396	26109	26148	26200

5.2 Uso de Gas, coste y tiempo de confirmación

Para comprobar el coste del Gas de las operaciones de escritura que requiere el pago de una cuota de transferencia se ha utilizado la Web *etherscan* de *Rinkeby* [16]. Esta información también se podría obtener a través de nuestra API, pero resulta más sencillo a través de la Web. Una captura de la Web se puede ver en la figura 5.2.

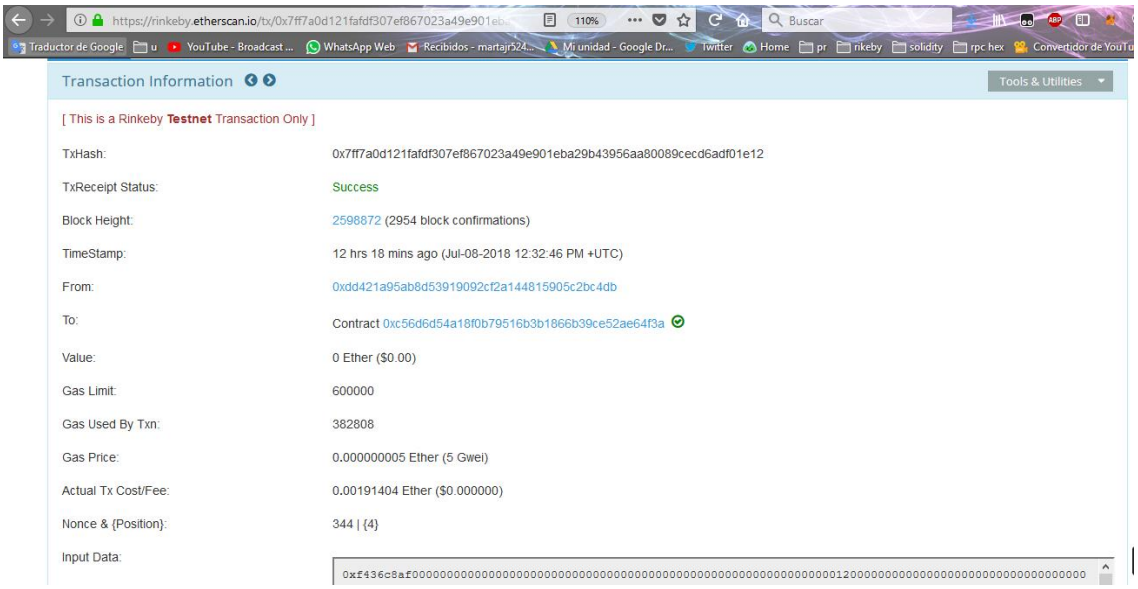


Figura 5.2 Captura rinkeby scan
Fuente :https://rinkeby.etherscan.io/

La Web *ether*scan nos muestra si la transferencia ha sido exitosa, el Gas utilizado, el precio del Gas, el contenido de la transferencia y el bloque que la contiene.

En la tabla 5.2 mostramos el Gas utilizado por las distintas operaciones de escritura. Solo se toma una medida ya que no hay cambios significativos entre una llamada y otra.

Tabla 5.2 Coste en gas por transferencia

Método	Hash de la transferencia	Gas
insertDocument	0x7ff7a0d121fafdf307ef867023a49e901eba29b4395 6aa80089cecd6adf01e12	382808
setFactoringTotal	0x8fe9b1d8c16767e37f766a87a56f6f437126e2dcbe9 21371503f42f2b4974824	56598
SetExpirationDate	0x6494c2c73ca5ef789ae0288c46c4bd49e10fe00fb75 bc854171b3f0f194c4263	56615
setFinancialInstituitionName	0xbbaa0b2ccc1db4cf52964607c39d32297f3c11c86b93 7b59947feb2818342da9	55596
setPaymentDate	0x8625a56b357c710c1115cd0ab34cbee5a6096cac3a1b c2d521ff8b78ac7293cf	55602
SetAccepted	0x0d563e0731742550a824d7c457547f19e4a088aee67d6 18c2f1bb3647f5072fb	39485
setPaid	0x0648003331ad38759d605eebae63dc0440336ca2519 e8116f55e7d6c3c75d85c	39156
setPaidPlus	0x45cf2730c96780f32dff7b7961307991a2412eb4a84f37 d454571cab91061daf	61675
setFactRequest	0x660cd9579639c5ce26a09b98d3c84509be8a804fa887a95 b5232c0f84a87c180	40418
setFactAccepted	0x4636f2e1e0448ba5fa66181fc3266dc7956fdf1cef4223f81 61b7befc5fd719b	39955
setFactAcceptedPlus	0x8c601c16d0b5aa6b7607e06ef5df38e8eeef2c65993c1e60 bbffb78a5cb090ed	108628
delete	0x6117bb8b5d302c326be6672235ec9d7951a8ac2d29 c1c0a4067ada0d96bd9584	87290

En la tabla 5.2 se observa que aquellas operaciones más complejas, como insertar un documento y cambiar estados a la vez que se introducen datos, consumen más Gas.

Con estas medidas del Gas usado podemos ir a la Web *ether* Gas station [17], y obtener estimaciones del precio en dólares y el tiempo que tardaría esta transferencia en confirmarse. A mayor precio menos tiempo de espera ya que los mineros estarán más dispuestos a gestionar la transferencia. Estas estimaciones son volubles ya que el precio del *ether* y las cuotas fluctúan.

En la figura 5.3 se puede ver un a captura de la interfaz donde se nos permite obtener estas estimaciones.

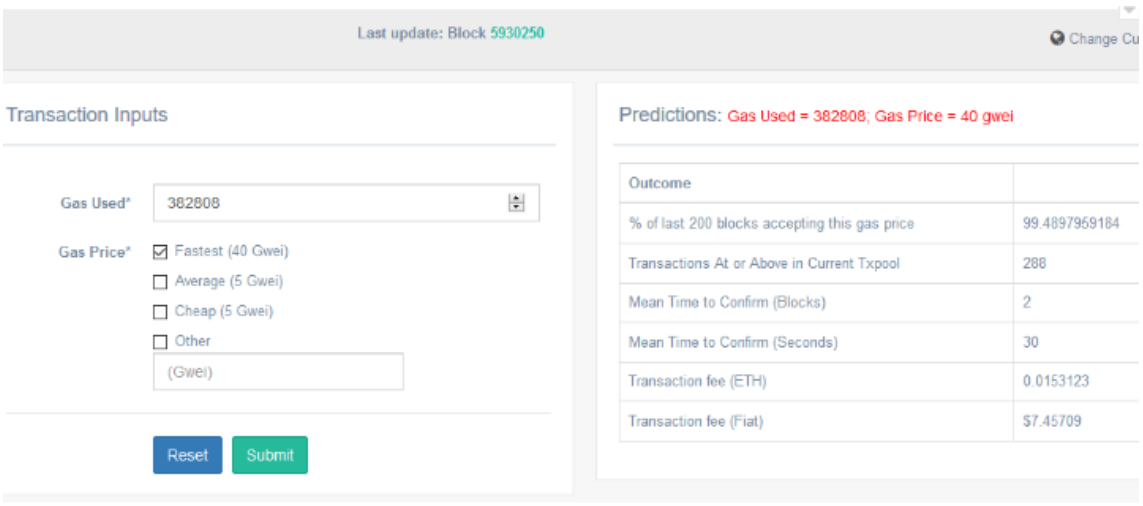


Figura 5.3 captura eth gas station
Fuente: https://ethgasstation.info/calculatorTxV.php

En la captura de Gas station, realizada el 5 de julio de 2018, observamos que el precio promedio del gas es de 5 *Gwei*, y 40 *Gwei* para una transferencia rápida. un *Gwei* es equivalente a 0.000000001 *ether*. En el caso de una transferencia rápida la Web considera que se confirma tras 2 bloques y en el caso promedio se consideran 30 bloques. Esta confirmación indica el tiempo en el que la transferencia se puede considerar completamente integrada en la *Blockchain* y segura.

En la tabla 5.3 mostramos las estimaciones de tiempo, en segundos, y cuota de transferencia, en dólares, para el precio de gas promedio. Y en la tabla 5.4 tenemos lo mismo, pero con un precio de gas de 40 *Gwei*.

Si seguimos el ciclo de vida de documento, insertar, modificar los estados, introducir datos y borrar, obtenemos que, en el caso promedio cambiando estados y estableciendo datos por separado, se obtiene un tiempo de confirmación total de más de 50 m y un precio de cuota total de más de 2 $. Si cambiamos los estados y establecemos los datos a la vez con los métodos plus el tiempo se reduce a más de 30 m y el precio a 1,7 $.

Si utilizamos 40 Gwei obtenemos un tiempo de 5 m y un precio de 16,6 $ para establecer los datos por separado. Y un tiempo de 3 m y un precio de 14 $ para los métodos plus.

Una conclusión que podemos obtener es que resulta mucho más eficiente cambiar los estados y establecer los datos utilizando los métodos plus.

Pero también podemos concluir que la falta de eficacia de la *Blockchain* actual, y los altos coste de las cuotas de transferencia, hacen nuestra solución poco eficiente para el problema planteado.

Tabla 5.3 tiempos y costes con precio del gas promedio

Método	**Gas**	**Precio del gas (Gwei)**	**Tiempo estimado (seg)**	**Cuota de transferencia (USD)**
insertDocument	382808	5	304	0,93212
setFactoringTotal	56598	5	304	0,13782
SetExpirationDate	56615	5	304	0,13787
setFinancialInstituitionName	55596	5	304	0,13539
setPaymentDate	55602	5	304	0,13539
SetAccepted	39485	5	304	0,09613
setPaid	39156	5	304	0,09535
setPaidPlus	61675	5	304	0,15019
setFactRequest	40418	5	304	0,09842
setFactAccepted	39955	5	304	0,0973
setFactAcceptedPlus	108628	5	304	0,26449
delete	87290	5	304	0,21258

Tabla 5.4 Tiempos y costes con precio del gas alto

Método	**Gas**	**Precio del gas (Gwei)**	**Tiempo estimado (seg)**	**Cuota de transferencia (USD)**
insertDocument	382808	40	30	7,45709
setFactoringTotal	56598	40	30	1,10252
SetExpirationDate	56615	40	30	1,10286
setFinancialInstituitionName	55596	40	30	1,08299
setPaymentDate	55602	40	30	1,08314
SetAccepted	39485	40	30	0,76917

setPaid	39156	40	30	0,76274
setPaidPlus	61675	40	30	1,20143
setFactRequest	40418	40	30	0,78733
setFactAccepted	39955	40	30	0,77832
setFactAcceptedPlus	108628	40	30	2,11606
delete	87290	40	30	1,70041

6 Conclusión

En este apartado se expone si la solución es viable o no en este momento y su posible viabilidad en el futuro.

6.1 Conclusión

En el capítulo 5 evaluamos los costes y la eficacia de nuestro SC y llegamos a la conclusión de que nuestra solución no resulta eficiente. Nuestros principales obstáculos son las altas cuotas de transacción, la limitada capacidad de nuestra comunicación con *Ethereum* y la falta de velocidad en la escritura de los datos.

El coste de las cuotas podría disminuir en un futuro si *Ethereum* se extiende y surgen más mineros de forma que ya no sería necesario competir con cuotas más altas para que se mine nuestra transacción.

La limitación en nuestra capacidad de comunicación se debe al tamaño de los bloques y solo podría solucionarse aumentando el tamaño del bloque. Pero este causaría que la velocidad de minado de los bloques fuera aún más lenta. Además, hay que tener en cuenta que la cantidad de datos que necesitamos siempre seguiría aumentando.

Finalmente tenemos la velocidad de minado de un bloque. Esta es menor en *Ethereum* que en *Bitcoin*, pero aun así se nota, sobre todo al comparar los tiempos de respuesta con bases de datos tradicionales. Hay dos formas de disminuir este tiempo. Una de ellas sería hacer la encriptación mucho más sencilla, pero esto comprometería la seguridad de la cadena. La otra forma sería reducir el tamaño de los bloques, pero entonces podríamos transmitir incluso menos información.

De los 3 problemas que tenemos uno de ellos podría solucionarse en un futuro cercano. Pero los otros 2 problemas no tienen una solución sencilla e incluso se influyen negativamente entre sí.

Nuestra conclusión es que la *Blockchain* podría ser utilizada para aplicaciones que no dependan de un gran volumen de datos en las transacciones y que se puedan permitir largos tiempos de escritura. Pero no es una solución eficiente para nuestro problema actualmente.

6.2 Posibles ampliaciones

Las posibles ampliaciones y mejoras de este proyecto son las siguientes:

1. Mejorar la estructura de datos: la estructura de datos que hemos utilizado simula, de forma simplificada, la estructura de datos utilizada por *Wupplier,* que cuenta con un servidor y una base de datos tradicional y para *Solidity* estos datos pueden llegar a ser pesados de manejar. Podría realizarse un estudio mas profundo de los datos para obtener solamente aquellos que sean imprescindibles. Así como establecer un formato

más definido que nos permitiera usar más datos de tamaño fijo. De esta forma se podría crear un Smart Contract más eficiente y consumir menos Gas.

2. Mejora del equipo: el equipo donde se ha realizado este proyecto es un portátil Lenovo con 4 años de antigüedad que tiene un rendimiento reducido. Un equipo en mejores condiciones obtendría mejores tiempos de respuesta.
3. Distribuir la red: en este proyecto se ha planteado la utilización de un único nodo al que se accedería a través de la API desarrollada. Se opto por no distribuir la red de forma que cada usuario tuviera un nodo porque se consideró que dichos usuarios no querían mantener un nodo propio. Pero si los usuarios tuvieran nodos propios se eliminaría la necesidad de la API y los tiempos de respuestas mejorarían.

Pliego de condiciones

En este capítulo se especifican los aspectos relacionados con la información de licencias, los derechos de autoría y las responsabilidades. Así como las condiciones bajo las que se ha desarrollado el presente TFG y las que se establecen al usuario para poder usar el software que se ofrece.

Pl.1 Condiciones Hardware

Durante el desarrollo de este TFG se han usado los dispositivos hardware recogidos en la Tabla Pl.1.

Equipo	Modelo	Fabricante
Ordenador portátil	Lenovo G50-80, CPU Intel® Core™ i5-5200U CPU @ 2.20GHz, 8 GB RAM, 500 GB HDD	Lenovo

Tabla Pl. 1 Condiciones Hardware

Pl.2 Condiciones S*of*tware

Por otro lado, en la Tabla Pl.2 se exponen las herramientas s*of*tware utilizadas, especificando su versión.

Aplicación	Versión
JetBrains PHPStorm	2017.2.4
Sistema Operativo Windows	Micros*of*t Windows 10 Home
Micros*of*t *Of*fice [83]	2016
Solidity	0.4.4
PHP	7.1.11
Symfony	3.4

Tabla Pl. 2 Condiciones Software

Pl.3 Condiciones de uso por parte de usuario

El objetivo de este apartado es comentar las condiciones hardware y software necesarios para que el usuario pueda utilizar el servicio que brinda la aplicación.

El usuario necesitaría una maquina capaz de contener un nodo de *Ethereum*. La limitación principal para esto es el espacio de almacenamiento. Se recomienda un espacio de almacenamiento de 100GB como mínimo para la red de pruebas que hemos utilizado.

Pl.4 Condiciones de licencia

El código desarrollado en este trabajo es propiedad de la Universidad de Las Palmas de Gran Canaria y todos los que pretendan hacer uso de él deben aceptar todas las cláusulas establecidas en esta licencia. El uso de las plataformas se podría hacer bajo autorización del autor, tutores del TFG, y la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Pl.5 Derechos de autor

Tanto el código como la información que se adjunta están protegidos por las leyes de propiedad intelectual que les sean aplicables, así como las disposiciones de los tratados internacionales. Por tanto, el código se considera un producto protegido por derechos de autor. Esto no es óbice para que una persona pueda copiar o utilizar el código bajo la autorización del autor, tutores del TFG, y la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Pl.6 Restricciones

No se permite el uso de ingeniería inversa. Sí se permite la transferencia del código a un tercero siempre que no se conserve ninguna copia, incluyendo actualizaciones o material escrito adicional.

Pl.7 Garantía

El autor del TFG lo presenta *AS IS* (tal cual), sin garantía implícita de ningún tipo. No se responsabiliza de los daños que pudieran causar a equipos o personas por el uso del código o la documentación. El autor no asegura, garantiza, o realiza ninguna declaración sobre el uso y resultados derivados de la utilización del código y/o del resto de la información proporcionada.

El código no está exento de errores y no está diseñado para entornos de riesgo que requieran de un funcionamiento a prueba de fallos. El autor rechaza expresamente cualquier garantía explícita e implícita de adecuación del código para actividades de riesgo.

Pl.8 Limitación de responsabilidad

En ningún caso el autor ni los tutores, ni la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de las Palmas de Gran Canaria serían responsables de los perjuicios directos, indirectos incidentales o consiguientes, gastos, lucro cesante, pérdida de ahorros, interrupción de negocios, pérdida de información comercial o de negocio, o cualquier otra pérdida que resulte del uso o de la incapacidad de usar el código o la documentación. El usuario conoce y acepta este riesgo, así como el resto de las cláusulas y restricciones. El autor no reconoce otra garantía que no haya sido indicada anteriormente.

Pl.9 Otras consideraciones

En el supuesto de que cualquier disposición de esta licencia sea declarada total o parcialmente inválida, las cláusulas afectadas serán modificadas convenientemente de manera que sea

ejecutable una vez modificada, permaneciendo el resto de este contrato en vigencia. Este contrato se rige por las leyes de España. El usuario acepta la jurisdicción exclusiva de los tribunales españoles con relación a las disputas derivadas de la presente licencia.

Presupuesto

Este capítulo contiene el presupuesto que recoge los gastos generados en la realización del presente TFG.

Pr.1 Componentes del presupuesto

El presupuesto calculado se divide en las siguientes partes:

- Recursos materiales.
- Trabajo tarifado por tiempo empleado.
- Material Fungible.
- Redacción de la documentación.
- Derechos de visado del *Colegio Oficial de Ingenieros Técnicos de Telecomunicación* (*COITT*).
- Gastos de tramitación y envío.
- Aplicación de impuestos y coste total.

Pr.2 Recursos Materiales

Para la correcta ejecución y desarrollo del TFG han sido necesarios diversos recursos tanto de hardware como herramientas de s*of*tware, las cuales llevan asociadas un coste en sus licencias. Entre estos recursos, se podrían destacar el paquete de Micros*of*t *Of*fice para la redacción de esta memoria, el ordenador portátil y la impresora. Así cabe destacar que el IDE PHPStorm, con el que se desarrolló todo el sistema, son gratuitos, por lo que no conllevas gastos.

Con el fin de establecer un cálculo de la amortización, se presupone el sistema de amortización como lineal, de tal forma que se asume que el inmovilizado material se desprecia de forma constante a lo largo de su vida útil.

Así, para llevar a cabo el cálculo de la cuota de amortización anual, se calcula usando la Ecuación Pr.1.

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Número\ de\ años\ de\ vida\ útil}$$

Ecuación Pr. 1

De esta manera, la amortización total aparece reflejada en la Tabla Pr.1, así como los diferentes valor y cuotas de los diferentes recursos empleados en este TFG.

Recurso					Uso (meses)	

	Valor de adquisición (€)	Valor residual (*) (€)	Vida útil (años)	Cuota anual (€)		Cuota aplicable (€)
Lenovo G50-80, CPU Intel® Core™ i5-5200U CPU @ 2.20GHz, 8 GB RAM, 500 GB HDD	1099	330	5	153,8	4	51,3
Licencia Paquete *Off*ice Windows (2017)	150	0	5	30	4	10
MATERIALES AMORTIZABLES	TOTAL					61,3

Tabla Pr. 1 Amortización de materiales

(*) El valor residual puede ser definido como el valor teórico supuesto que tendría el elemento después de su vida útil. En el caso de un terminal móvil y portátil se deprecian en una tasa del 30 % con respecto a su valor. Según *Artículo 34*, *LISR*.

El coste de los materiales amortizables es de un total de: **sesenta y un euros con treinta céntimos** (61,3€).

Pr.3 Trabajo tarificado por tiempo empleado

Este concepto contabiliza los gastos que corresponden a la mano de obra, según el salario correspondiente a la hora de trabajo de un Ingeniero Técnico de Telecomunicación.

Según la tabla retributiva de personal contratado en TFGs de investigación elaborada por la ULPGC en el año 2016, este salario, con una dedicación de 20 horas semanales, asciende a 896,31€ mensuales. Lo que se aproxima a una retribución de 11,20 €/h. Este TFG se tal como comprende el TFG Docente ha conllevado 300 horas, por lo que se calcula el coste total por tiempo empleado en:

$$11{,}20 \cdot 300 = 3.360{,}00\,€$$

Por lo tanto, el trabajo tarifado por tiempo empleado asciende a la cantidad de ***tres mil trescientos sesenta euros*** (*3.360,00€*).

Pr.4 Redacción del trabajo

Se ha utilizado la Ecuación P.2 para determinar el coste asociado a la redacción de la memoria del presente Trabajo Fin de Grado.

$$R = 0{,}07 \times P \times C_n,$$

Ecuación Pr. 2

Donde:

- R son los honorarios por la redacción del trabajo.
- P es el presupuesto.
- C_n es el coeficiente de ponderación en función del presupuesto.

El valor del presupuesto P se calcula sumando los costes del trabajo tarifado por tiempo empleado y de la amortización del inmovilizado material, tanto hardware como s*of*tware. El resultado de los costes se muestra en la Tabla Pr.2.

Concepto	Coste (€)
Trabajo tarifado por tiempo empleado	3.360,00
Amortización del material	61,3
Total	3.521,3

Tabla Pr. 2 Valor del presupuesto

Como el coeficiente de ponderación Cn para presupuestos menores de 30.050,00€ viene definido por el COITT con un valor de 1.00, el coste derivado de la redacción del TFG es de:

$$R = 0{,}07 \times 3.521{,}3\,€ \times 1 = 246{,}5\,€$$

Ascendiendo de esta forma el coste de la redacción del trabajo a ***doscientos cuarenta y seis euros con cincuenta céntimos*** $(246{,}5€)$.

Pr.5 Material Fungible

Los costes asociados a la de edición de documentos, así como los gastos del material de *of*icina se recogen en la Tabla Pr.3.

Descripción de gastos	Coste (€)
Impresión	60
CD-ROM x 3	6
Encuadernación	3,80
Total	69,80

Tabla Pr. 3 Costes de material fungible

Los gastos asociados al material fungible ascienden a un total de: **sesenta y nueve euros con ochenta céntimos** (69,80 €).

Pr.6 Derechos de visado del COITT

El COITT establece que, para TFGs técnicos de carácter general, los derechos de visado para 2016 se calculan en base a la Ecuación Pr.3

$$V = 0{,}006 \times P_1 \times C_1 + 0{,}003 \times P_2 \times C_2,$$

Ecuación Pr. 3

Donde:

- V es el coste de visado del trabajo.
- P_1 es el presupuesto del TFG.
- C_1 es el coeficiente reductor en función del presupuesto.
- P_2 es el presupuesto de ejecución material correspondiente a la obra civil.
- C_2, es el coeficiente reductor en función a P2.

El valor del presupuesto P_1 se halla sumando los costes de las secciones correspondientes al trabajo tarifado por tiempo empleado, a la amortización del inmovilizado material y a la redacción del documento. Esta suma se muestra en la Tabla Pr.4. Al igual que en el caso anterior, el coeficiente C_1 para proyectos de presupuesto inferior a 30.050€ es de 1, asimismo el valor de P_2 es de 0€ ya que no se realiza ninguna obra.

Concepto	Coste (€)
Trabajo tarifado por tiempo empleado	3.360,00
Amortización del material	61,3
Redacción del trabajo	246,5
Total	3.667,8

Tabla Pr. 4 Presupuesto P1

De esta forma, aplicando a la Ecuación P.3 los datos de la Tabla P.4 y el coeficiente especificado se obtiene:

$$V = 0{,}006 \times 3.667{,}8\,€ \times 1 = 22\,€$$

Los costes por derechos de visado del presupuesto ascienden a ***veintidós euros*** (*22 €*).

Pr.7 Gastos de tramitación y envío

Los gastos de tramitación y envió están estipulados en seis euros (6,00€) por cada documento visado de forma telemática.

Pr.8 Aplicación de impuestos y coste total

El presupuesto total del presente TFG está gravado por el *Impues*to *General Indirec*to *Canario* (*IGIC*), que está establecido en la actualidad en un siete por ciento (7%). El coste total de este TFG se encuentra desglosado en la Tabla Pr.5.

Descripción	Subtotal (€)
Amortización de materiales	61,3
Trabajo tarifado por tiempo empleado	3.360,00
Costes de material fungible	69,80
Redacción del trabajo	246,5
Derechos de visado del COIT	22
Gastos de tramitación y envío	6,00
Suma (€)	3.765,6
IGIC (7%)	263,6
TOTAL	4.029,2

Tabla Pr. 5 Coste total

El presupuesto total del TFG *Sistema de localización y orientación en zonas de exteriores con escasos puntos de referencia* asciende a: **cuatro mil veinte y nueve euros con veinte céntimos** (4.029,2€).

Fdo.: Marta Jiménez Rodríguez

En Las Palmas de Gran Canaria a 29 de julio de 2019

Bibliografía

[1] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," Journal *of* Cryptology, vol 3, no 2, p 99-111, 1991.

[2] Satoshi Nakamoto, "*Bitcoin*: A *Peer*-to-*Peer* Electronic Cash System," https://*Bitcoin*.org/*Bitcoin*.pdf, 2008, disponible en marzo de 2018.

[3] *Bitcoin*: *Proof-ofWork*, Khan Academy, https://es.khanacademy.org/economics-finance-domain/core-finance/money-and-banking/*Bitcoin*/v/*Bitcoin-proof-of-work*, disponible en febrero de 2018.

[4] "A Next-Generation Smart *Contract* and *Decentralized Application* Platform," https://github.com/*Ethereum*/wiki/wiki/White-Paper, 2018, disponible en marzo de 2018.

[5] *Ether*um.org: Primer *Contrac*to inteligente en *Ether*um, https://www.*Ethereum*.org/greeter, disponible en febrero de 2018.

[6] Librería Web3 de *Javascript* para *Ether*um, https://github.com/*Ethereum*/Web3.js/, disponible en febrero de 2018.

[7] *rinkeby*: *Ether*um test net: https://www.*Rinkeby*.io/#stats, disponible en febrero de 2018.

[8] Lenguaje de programación *Solidity*: https://*Solidity*.readthedocs.io/en/develop/*Solidity*-in-depth.html, disponible en febrero de 2018.

[9] Documentación ABI https://*Solidity*.readthedocs.io/en/develop/abi-spec.html disponible en febrero de 2018.

[10] Documentación cURL:https://curl.haxx.se/docs/manpage.html disponible en febrero de 2018.

[11] SoftWare Postman https://www.getpostman.com/

[12] Documentación API Ethereum https://github.com/ethereum/wiki/wiki/JSON-RPC disponible en febrero de 2018.

[13] Brian Mulloy, "Web API Design, Crafting *Interface*s that Developers Love"

[14] Documentación Symfony: http://librosWeb.es/libro/Symfony_2_4/ disponible en febrero de 2018.

[15] IDE Remix: https://*remix.Ethereum*.org/ disponible en febrero de 2018.

[16] Transacciones: https://*rinkeby.ether*scan.io/ disponible en febrero de 2018.

[17] Estimaciones de ether: https://ethgasstation.info/index.PHP disponible en febrero de 2018.

[18] Documentación Geth:https://github.com/*Ethereum*/go-*Ethereum*/wiki/Installing-*Geth* disponible en febrero de 2018.

[19] The Singular Factory: https://www.singularfactory.com/ disponible en febrero de 2018.

[20] Wupplier: https://www.*Wupplier*.com/es/ disponible en febrero de 2018.
[21] Documentación Composer: https://get*Composer*.org/ disponible en febrero de 2018.
[22] Descargar Geth: https://*Geth*.*Ethereum*.org/downloads/ disponible en febrero de 2018.
[23] Componente rest-bundle: https://packagist.org/packages/friends*ofSymfony*/rest-bundle disponible en febrero de 2018.
[24] Componente keccak: https://packagist.org/packages/kornrunner/keccak disponible en febrero de 2018.
[25] Infura: https://infura.io/ disponible en febrero de 2018.
[26] Metamask: https://metamask.io/ disponible en febrero de 2018.
[27] Documentación Mist: https://github.com/*Ethereum*/*Mist* disponible en febrero de 2018.

Anexos

Anexo 1: Listado de llamadas de la API

En este anexo se definen todas las llamadas de la API, utilizando el identificador o los datos identificativos. También se dan ejemplos de las llamadas utilizadas y de inputs y *Out*puts.

A1.1 Comprobar la conexión de la red

Esta llamada devuelve *CONNECTED* si hay conexión con la red y *NOT_CONNECTED* en caso contrario:

- *Ruta*: GET /net/connection
- *Ejemplo de llamada*: http://localhost:8000/net/connection
- *Ejemplo de Output*: {"status": "CONNECTED"}

A1.2 Comprobar la sincronización del nodo

Esta llamada devuelve *NODE_SYNC* si el nodo está sincronizado y *NODE_NOT_SYNC* en caso contrario:

- *Ruta*: GET /net/sync
- *Ejemplo de llamada*: http://localhost:8000/net/sync
- *Ejemplo de Output*: {"status": "NODE_SYNC"}

A1.3 Comprobar el estado de una transacción

Devuelve el estado de una transferencia con *SUCCESS* o *FAILURE*. También incluye el gas usado y el bloque que incluye la transferencia:

- *Ruta*: GET /transaction/{hashTx}
- *Ejemplo de llamada*:

http://localhost:8000/transaction/0x7ff7a0d121fafdf307ef867023a49e901eba29b43956aa80089cecd6adf01e12

- *Ejemplo de Output*:

{"status": "SUCCESS",
"hashTx": "0x7ff7a0d121fafdf307ef867023a49e901eba29b43956aa80089cecd6adf01e12",
"gasUsed": 382808,

"blockNumber": 2598872,

"blockHash":
"0xd14863b18e7c2fb88b7d7d83017cfabf108d1096df3054c689369dc2944e9f71"}

A1.4 Insertar un nuevo documento

Añade un nuevo documento con los parámetros dados. Los estados se inicializan como pendientes, *pending y* fact_pending, y el resto de los datos relacionados con los adelantos y el pago se asignan una vez que se llegue a esos estados. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos no exista ya:

- *Ruta*: POST /documents/
- *Ejemplo de input*:

{"invoiceNumber":"invoiceNumber05",

"fiscalYear":"FY05",

"total":"total05",

"currency":"C05",

"paymentType":"paymentType05",

"supplierName":"supplierName05",

"customerName":"customerName05",

"paymentTerms":5,

"expirationDate":"AAAA/MM/DD HH:MM:SS",

"invoiceDate":"AAAA/MM/DD HH:MM:SS"}

- *Ejemplo de Output*:

{"documentUniqueId":
"6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",

"hashTx":
"0x7ff7a0d121fafdf307ef867023a49e901eba29b43956aa80089cecd6adf01e12
",

"invoiceNumber": "invoiceNumber05",

"total": "total05",

"supplierName": "supplierName05",

"customerName": "customerName05",

"fiscalYear": "FY05"}

A1.5 Comprobar la existencia de un documento

Devuelve un booleano que es verdadero si el documento existe o falso si el documento no existe:

- *Ruta*: GET /documents/{id}/exists
- *Ejemplo de llamada*:

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/exists

- *Ejemplo de Output*:

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"exists": true,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.6 Comprobar si un documento está pendiente

Devuelve un booleano que es verdadero si es documento está pendiente, *pending*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta*: GET /documents/{id}/pending
- *Ejemplo de llamada*:

http://localhost:8000/documents/*6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08*/pending

- *Ejemplo de Output*:

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"documentIsPending": true,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.7 Comprobar si un documento está aceptado

Devuelve un booleano que es verdadero si el documento está aceptado, *accepted*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta*: GET /documents/{id}/accepted
- *Ejemplo de llamada*:

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/accepted

- *Ejemplo de Output*:

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"documentIsAccepted": false,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.8 Comprobar si un documento está pagado

Devuelve un booleano que es verdadero si el documento está pagado, *paid*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta*: GET /documents/{id}/paid
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/paid

- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"documentIsPaid": false,
"invoiceNumber": "",
"total": "",
"supplierName": "",

"customerName": "",
"fiscalYear": ""}

A1.9 Comprobar si un documento tiene adelanto pendiente

Devuelve un booleano que es verdadero si el adelanto está pendiente, *fact_pending*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* GET /documents/{id}/ factoring/pending
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoring/pending

- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"documentFactoringIsPending": true,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.10 Comprobar si un documento tiene un adelanto solicitado

Devuelve un booleano que es verdadero si el adelanto está solicitado, *fact_requested*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* GET /documents/{id}/ factoring/requested
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoring/requested

- *Ejemplo de Output:*

{"documentUniqueId":
"6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"documentFactoringIsRequested": false,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.11 Comprobar si un documento tiene un adelanto aceptado

Devuelve un booleano que es verdadero si el adelanto está aceptado, *fact_accepted*, o falso en caso contrario. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* GET /documents/{id}/ factoring/accepted
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoring/accepted

- *Ejemplo de Output:*

{"documentUniqueId":
"6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"documentFactoringIsAccepted ": false,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.12 Establecer el total del adelanto

Establece el dato *factoringTotal* de un documento si se comprueba que se ha aceptado el adelanto, *fact_accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/factoring/total
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoring/total

- *Ejemplo de input:* {"factoringTotal": “newFactoringTotal05”}
- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx": "0x8fe9b1d8c16767e37f766a87a56f6f437126e2dcbe921371503f42f2b4974824",
"factoringTotal": "newFactoringTotal05",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.13 Establecer la fecha de expiración del adelanto

Establece el dato *factoringExpirationDate* de un documento si se comprueba que se ha aceptado el adelanto, *fact_accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/factoring/expirationDate
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoring/expirationDate

- *Ejemplo de input:* {"factoringExpirationDate": “AA/MM/DD HH:MM:SS”}
- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx": "0xfbef9b420449c5fb8ee62f425b10e9ba12fd43675fae2153ea00aaa3f7621d2b",
"factoringExpirationDate": “AA/MM/DD HH:MM:SS”,
"invoiceNumber": "",
"total": "",
"supplierName": "",

"customerName": "",
"fiscalYear": ""}

A1.14 Establecer la institución financiera del adelanto

Establece el dato *financialInstitutionName* de un documento si se comprueba que se ha aceptado el adelanto, *fact_accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta*: POST /documents/{id}/factoring/financialInstitutionName
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoring/financialInstitutionName

- *Ejemplo de input:* {"financialInstitutionName": “newfinancialInstitutionName05”}
- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx": "0x43f20b8b3b5c976476332557400120be6c37cf49022b677e4ef31422b229e37c",
"financialInstitutionName": “newfinancialInstitutionName05”,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.15 Establecer la fecha de pago

Establece el dato *paymentDate* de un documento si se comprueba que se ha pagado, *paid*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/paymentDate
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/paymentDate

- *Ejemplo de input:* {"paymentDate": “AA/MM/DD HH:MM:SS”}

- *Ejemplo de Output:*

{"documentUniqueId":
"6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx":
"0x8625a56b357c710c1115cd0ab34cbee5a6096cac3a1bc2d521ff8b78ac7293cf",
"paymentDate": “AA/MM/DD HH:MM:SS”,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.16 Establecer un documento como aceptado

Se marca el documento dado como aceptado, *accepted*, si se comprueba que antes estaba pendiente, *pending*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/state/accepted
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/state/accepted

- *Ejemplo de Output:*

{"documentUniqueId":
"6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx":
"0x0d563e0731742550a824d7c457547f19e4a088aee67d618c2f1bb3647f5072fb",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.17 Establecer un documento como pagado

Se marca el documento dado como pagado, *paid*, si se comprueba que antes estaba aceptado, *accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/state/paid
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/state/paid

- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx": " “0x0648003331ad38759d605eebae63dc0440336ca2519e8116f55e7d6c3c75d85c",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.18 Establecer un documento como pagado y establecer datos

Se marca el documento dado como pagado, *paid*, y se establece el dato *paymentDate* si se comprueba que antes estaba aceptado, *accepted*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/state/paidPlus
- *Ejemplo de llamada:*

http://localhost:8000/documents/ebddd8a3f40e4d665d5ef4616af531f9aeb01f53e32536df5af72eefbd46d01b/state/paidPlus

- *Ejemplo de input:* {"paymentDate": “AA/MM/DD HH:MM:SS”}
- *Ejemplo de Output:*

{"documentUniqueId":
"ebddd8a3f40e4d665d5ef4616af531f9aeb01f53e32536df5af72eefbd46d01b",
"hashTx":
"0x45cf2730c96780f32dff7b7961307991a2412eb4a84f37d454571cab91061daf",
"paymentDate": “AA/MM/DD HH:MM:SS”,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.19 Establecer un adelanto como solicitado

Se marca el adelanto del documento como solicitado, *fact_requested*, si se comprueba que el documento se encuentra aceptado, *accepted*, y estado del adelanto es pendiente, *fact_pending*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/factoringState/requested
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoringState/requested

- *Ejemplo de Output:*

{"documentUniqueId":
"6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx":
"0x660cd9579639c5ce26a09b98d3c84509be8a804fa887a95b5232c0f84a87c180",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.20 Establecer un adelanto como aceptado

Se marca el adelanto del documento como aceptado, *fact_accepted*, si se comprueba que el documento se encuentra aceptado, *accepted*, y estado del adelanto es solicitado, *fact_requested*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/factoringState/accepted
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/factoringState/accepted

- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"hashTx": "0x4636f2e1e0448ba5fa66181fc3266dc7956fdf1cef4223f8161b7befc5fd719b",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.21 Establecer un adelanto como aceptado y establecer datos

Se marca el adelanto del documento como aceptado, *fact_accepted*, si se comprueba que el documento se encuentra aceptado, *accepted*, y estado del adelanto es solicitado, *fact_requested*. Además, se establecen los datos *factoringTotal*, *factoringExpirationDate y financialInstitutionName*. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* POST /documents/{id}/factoringState/acceptedPlus
- *Ejemplo de llamada:*

http://localhost:8000/documents/ebddd8a3f40e4d665d5ef4616af531f9aeb01f53e32536df5af72eefbd46d01b/factoringState/acceptedPlus

- *Ejemplo de input:*

{"factoringTotal”: “newFactorinTotal06”,
"factoringExpirationDate”: “AA/MM/DD HH:MM:SS”,
"financialInstitutionName”: “newfinancialInstitutionName06”}

- *Ejemplo de Output:*

{"documentUniqueId": "ebddd8a3f40e4d665d5ef4616af531f9aeb01f53e32536df5af72eefbd46d01b",
"hashTx": "0x8c601c16d0b5aa6b7607e06ef5df38e8eeef2c65993c1e60bbffb78a5cb090ed",
"factoringTotal”: “newFactorinTotal06”,
"factoringExpirationDate”: “AA/MM/DD HH:MM:SS”,
"financialInstitutionName”: “newfinancialInstitutionName06”,
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.22 Eliminar un Documento

Se elimina el documento con el identificador dado. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* DELETE /documents/{id}
- *Ejemplo de llamada:*

http://localhost:8000/documents/cace11b81442bd7e117996c57b1177b1405a9135ab1a2c8cbc0fab5fc8567331

- *Ejemplo de Output:*

{"documentUniqueId": "cace11b81442bd7e117996c57b1177b1405a9135ab1a2c8cbc0fab5fc8567331",
"hashTx":"0x6117bb8b5d302c326be6672235ec9d7951a8ac2d29c1c0a4067ada0d96bd9584",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.23 Obtener número de documentos

Devuelve el número de documentos almacenados:

- *Ruta*: GET /documents/count
- *Ejemplo de llamada:* http://localhost:8000/documents/count
- *Ejemplo de Output:* {"Count": 4}

A1.24 Obtener el identificador de un documento

Devuelve el *documentUniqueId* del documento con el índice dado:

- *Ruta:* GET /documents/index/{index}
- *Ejemplo de llamada*: http://localhost:8000/documents/1
- *Ejemplo de Output:*

  {"documentUniqueId":
"62acdf9b69831b2efe5a292b5b2b6d5f740ca1e7d91542de9a077eef1797f889"}

A1.25 Obtener una lista de identificadores de los documentos

Devuelve un *array* con todos los *documentUniqueId* de los documentos almacenados:

- *Ruta:* GET /documents
- *Ejemplo de llamada:* http://localhost:8000/documents
- *Ejemplo de Output:*

  ["6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08”,
  "62acdf9b69831b2efe5a292b5b2b6d5f740ca1e7d91542de9a077eef1797f889",
  "23f5e6f918bae7f6885bcbf8358d3c527aba0d5843db01d287083e5c4e1dfdbd",
  "4a305f2a3e09112ddd946c706d439f142f63425826a68fcef03a6acea3fac6e9"]

A1.26 Obtener un dato de un documento

Devuelve el dato correspondiente del documento. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* GET /documents/{id}/<Dato>
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/state

- *Ejemplo de Output:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"State": "paid",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

A1.27 Obtener todos los datos de un documento

Devuelve todos los datos de un documento. Antes de ejecutarse se comprueba que el identificador generado con los datos identificativos exista:

- *Ruta:* GET /documents/{id}/all
- *Ejemplo de llamada:*

http://localhost:8000/documents/6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08/all

- *Ejemplo de Output con identificador:*

{"documentUniqueId": "6cbe1fdac2f40d1979e2f2b1bff261883b94e18169956fa6f1f8ad24daca8a08",
"InvoiceNumber": "invoiceNumber05",
"Total": "total05",
"FactoringTotal": "newfactoringTotal",
"State": "paid",
"Currency":"C05",
"PaymentType": "paymentType05",
"SupplierName": "supplierName05",
"CustomerName": "customerName05",
"FinancialInstitutionName": "newfinancialInstitutionName",
"FactoringState": "fact_accepted",
"PaymentTerms": 5,
"FiscalYear": "FY05",

"InvoiceDate": "AA/MM/DD HH:MM:SS",
"ExpirationDate": "AA/MM/DD HH:MM:SS",
"FactoringExpirationDate":"AA/MM/DD HH:MM:SS",
"PaymentDate": "AA/MM/DD HH:MM:SS",
"invoiceNumber": "",
"total": "",
"supplierName": "",
"customerName": "",
"fiscalYear": ""}

Glosario

Siglas	Expresión
ABI	Application Binary Interface
ADE	API Development Environment
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
DApp	Decentralized Application
EVM	Ethereum Virtual Machine
Geth	Go Ethereum
HTTP	HyperText Transfer Protocol
ID	Identificador
IDE	integrated development environment
JSON	JavaScript Object Notation
LIFO	Last In First Out
MVC	Model View Controller
P2P	Peer to Peer
PHP	Hypertext Preprocessor
PoW	Proof of Work
REST	Representational State Transfer
RPC	Remote Procedure Call
SC	Smart Contract
SHA	Secure Hash Algorithm
URL	Uniform Resource Locator