UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE CONTRATOS INTELIGENTES BASADOS EN EL SISTEMA ETHEREUM

Diego Sales Bellés

Tutor: Antonio León Fernández

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 18 de junio de 2019

Resumen

En este documento se proporciona una aproximación teórica a la tecnología de cadena de bloques, tecnología en fase de investigación y desarrollo en la actualidad y con una gran perspectiva de futuro. El documento también proporcionará una aproximación teórica al diseño y funcionamiento de la plataforma Ethereum, la segunda más grande del mundo relacionada con la tecnología de cadena de bloques, enfocada especialmente para el desarrollo e implementación de aplicaciones descentralizadas. También se proporcionará una aproximación teórica al concepto de contrato inteligente, sus utilidades, ventajas y aplicaciones, girando entorno a la plataforma de Ethereum.

Una vez realizada la introducción teórica de estas tecnologías, se desarrollarán dos aplicaciones enfocadas para el sistema informático de un hotel cápsula. Una de ellas será un contrato inteligente en el lenguaje de programación Solidity, enfocado especialmente en la plataforma Ethereum. Por otro lado, se desarrollará otra aplicación, en el lenguaje de programación Python, diseñado para interactuar con el contrato inteligente. Así mismo, se hará una propuesta del hardware necesario para trabajar con dichos programas.

Finalmente, se realizará una simulación de ejecución de las aplicaciones diseñadas en una simulación de la plataforma Ethereum.

Resum

En aquest document es proporciona una aproximació teòrica a la tecnologia de cadena de blocs, tecnologia en fase d'investigació i desenvolupament en l'actualitat i amb una gran perspectiva de futur. El document també proporcionarà una aproximació teòrica al disseny i funcionament de la plataforma Ethereum, la segona més gran del món relacionada amb la tecnologia de cadena de blocs, enfocada especialment per al desenvolupament i implementació d'aplicacions descentralitzades. També es proporcionarà una aproximació teòrica al concepte de contracte intel·ligent, les seves utilitats, avantatges i aplicacions, girant al voltant de la plataforma de Ethereum.

Un cop realitzada la introducció teòrica d'aquestes tecnologies, es desenvoluparan dues aplicacions enfocades per al sistema informàtic d'un hotel càpsula. Una d'elles serà un contracte intel·ligent en el llenguatge de programació Solidity, enfocat especialment a la plataforma Ethereum. D'altra banda, es desenvoluparà una altra aplicació, en el llenguatge de programació Python, dissenyat per interactuar amb el contracte intel·ligent. A més, es farà una proposta d'equip informàtic per a treballar en els programes desenvolupats.

Finalment, es realitzarà una simulació d'execució de les aplicacions dissenyades en una simulació de la plataforma Ethereum.

Abstract

This document provides a theoretical approach to blockchain technology, technology currently in the research and development phase and with a great perspective for the future. The document will also provide a theoretical approach to the design and operation of the Ethereum platform, the second largest in the world related to blockchain technology, especially focused on the development and implementation of decentralized applications. A theoretical approach to the concept of smart contracts, its utilities, advantages and applications will also be provided, revolving around the Ethereum platform.

Once the theoretical introduction of these technologies has been carried out, two applications focused on the computer system of a capsule hotel will be developed. One of them will be an intelligent contract in the programming language Solidity, focused especially on the Ethereum platform. On the other hand, another application will be developed, in the Python programming

language, designed to interact with the smart contract. Also, a hardware system will be proposed, in order to execute the designed applications.

Finally, a simulation of the execution of the applications designed in a simulation of the Ethereum platform will be carried out.

Índice

Capítulo 1. Introducción, objetivos y metodología

1.1 Introducción del proyecto

Un contrato inteligente es un protocolo informático con el objetivo de facilitar digitalmente, verificar y reforzar la negociación o aplicación de un contrato. Esta clase de contratos permiten la realización de transacciones fiables sin terceros. Las transacciones realizadas a través de contratos inteligentes son rastreables e irreversibles [1].

El principal objetivo de un contrato inteligente es conseguir un nivel superior de seguridad frente a los contratos tradicionales. Además, estos contratos pueden reducir los costes de transacción asociados a los negocios. Para programar un contrato inteligente, una herramienta interesante es el lenguaje de programación Solidity [2].

Muchas criptomonedas han implementado esta clase de contratos en sus sistemas de cadenas de bloques. Un ejemplo de ello es el sistema Ethereum, una plataforma descentralizada de código abierto, donde cualquier desarrollador puede crear y publicar aplicaciones distribuidas que empleen contratos inteligentes, propuesta por Vitalik Buterin. Esta plataforma usa como criptomoneda Ether, la cual puede ser empleada como instrumento de intercambio y como recompensa para los mineros del sistema [3].

Este proyecto explicará algunos conceptos de la tecnología de cadena de bloques, la plataforma Ethereum y los contratos inteligentes. Además, en este proyecto, el estudiante desarrollará un contrato inteligente con Solidity, con el propósito de resolver o mejorar un problema de la vida real. Para interactuar con dicho contrato, se diseñará un código en Python.

Los contratos e interfaces diseñados en este proyecto podrán ser aplicables al sistema informático de un hotel cápsula. Además, se realizará una propuesta de instalación del hardware necesario para la ejecución de dichos códigos.

1.2 Objetivos del proyecto

Los objetivos principales del proyecto son:

- Comprender cómo funciona la tecnología de cadena de bloques.
- Comprender cómo funcionan las plataformas descentralizadas como Bitcoin o Ethereum.
- Comprender cómo pueden ser aplicados los contratos inteligentes en la vida real.
- Desarrollar un contrato inteligente como solución técnica de una situación de la vida real, empleando la plataforma Etherum y Solidity.

Finalmente, el proyecto ayudará al estudiante a aprender sobre programación con la tecnología de cadena de bloques. Este es el objetivo principal del proyecto, debido a su creciente importancia en el mercado laboral internacional.

1.3 Metodología del proyecto

El proyecto seguirá las siguientes fases:

- En primer lugar, se realizará una breve investigación sobre conceptos básicos necesarios para la comprensión del proyecto. Para esta fase se destinará una semana.
- En segundo lugar, el estudiante investigará cómo funciona la tecnología de cadena de bloques desde un punto de vista técnico. Esta fase durará una semana.
- En tercer lugar, el estudiante investigará cómo funciona una plataforma basada en la tecnología de cadena de bloques. Dicha plataforma será Ethereum. Esta fase durará una semana.

- A continuación, el estudiante investigará qué es un contrato inteligente, y cómo funciona. Esta fase durará otra semana.
- Una vez todo el material teórico necesario haya sido explicado, el estudiante creará una situación real donde los contratos inteligentes puedan ser útiles. Esta situación será resuelta mediante la programación de un contrato inteligente mediante Solidity. Para crear la situación, se destinará otra semana.
- La siguiente fase consistirá en el desarrollo de dicha solución. Esta fase se compone de dos partes:
    - La primera se basará en aprender a programar en Solidity. Esta fase se realizará al mismo tiempo que las fases de investigación ya mencionadas. Esta fase tomará unas cinco semanas.
    - La segunda se basará en el desarrollo del código. Para ello, se creará un contrato inteligente en Solidity y, además, una aplicación en Python para interactuar con dicho contrato desarrollado. Para esta fase se estima que se emplearán unas tres semanas.
- Finalmente, se desarrollará una conclusión, donde se explicará cómo la solución propuesta puede mejorar nuestra situación creada a partir de contratos inteligentes. Esta fase se desarrollará en una semana.
- Así mismo, la redacción del documento final se realizará coetáneamente al desarrollo del proyecto.

Toda esta información nos dice que el proyecto se puede realizar en nueve semanas. Esta información se resume en el siguiente diagrama de Gantt.

Semana	1	2	3	4	5	6	7	8	9
Investigación de conceptos básicos	■								
Investigación de la tecnología Blockchain o cadena de bloques		■							
Investigación de la plataforma Ethereum			■						
Investigación del protocolo de Contratos inteligentes o Smart Contracts				■					
Investigación de una situación de la vida real					■				
Aprendizaje de programación mediante Solidity	■	■	■	■	■				
Creación del código solución en Solidity+Python						■	■	■	
Conclusión									■
Redacción del documento	■	■	■	■	■	■	■	■	■

Figura 1. Diagrama de Gantt

1.4 Conocimientos necesarios para la comprensión del trabajo

En primer lugar, el lector del documento debe conocer algunos conceptos básicos para la comprensión del proyecto. Estos conceptos son:

- **Criptografía:** estudio y práctica de técnicas para la comunicación segura en presencia de terceros o espías. El principal objetivo de la criptografía es construir protocolos para evitar la intromisión de terceros [4].
- **Hash**: algoritmo matemático que transforma cada bloque arbitrario en una nueva serie de caracteres con una longitud fija. Indistintamente de la longitud del valor de entrada, el valor de salida siempre tendrá la misma longitud [5]. Existen algunos algoritmos para obtener una función Hash, como el MD5 o el SHA-1.
    - Como ejemplo de función hash, el resultado de codificar la cadena “Diego” con SHA-1 es “1B47E87D02781E58A171F3221A7E4C032E67F2D0” [6]. Esta función puede ser útil para obtener contraseñas seguras, o como forma de verificar la integridad de un canal de comunicación.

- **Descentralización:** proceso de distribución o dispersión de funciones, poderes o cosas fuera de una ubicación. De acuerdo con el campo de estudio, el significado puede variar.
- **Red P2P:** red sin servidores ni clientes. Funciona con una serie de nodos haciendo la función de cliente y servidor al mismo tiempo con otros nodos de la red. Estas redes son:
    - **Autoescalables:** incrementar el número de nodos incrementa la demanda y la cantidad de recursos. Por ello, incrementar el número de nodos mejora el comportamiento de la red.
    - **Robustas:** la distribución del poder de la red hace que la caída de la misma sea más complicada.
    - Cada usuario puede ser un nodo, y ningún nodo es esencial.
    - **Los costes se distribuyen entre usuarios:** existe un intercambio de resursos. Estos recursos pueden ser diferentes clases de datos, como archivos.
    - **Anonimato:** es positivo para la red que el usuario origen de un archivo se mantenga en el anonimato.
    - **Seguro:** algunos objetivos de una red P2P segura giran entorno la identificación y eliminación de nodos malvados, evitar contenido infectado, evitar el espionaje de comunicaciones, la protección de los recursos de la red, etc.

    Estas redes, de acuerdo con su distribución final, pueden ser:

    - **Redes P2P centralizadas:** intercambios entre nodos realizados por un servidor central, el cual establece conexiones entre nodos y, además, almacena y distribuye información como quién almacena un archivo determinado.
    - **Redes P2P descentralizadas:** estas redes no necesitan ningún gestor central. Por ello, un nodo, para comunicarse con otro, tiene que buscar por sí mismo un camino, incluso a través de otros nodos.
    - **Redes P2P híbridas:** estas redes tienen un servidor central y nodos, y ambos interactúan, pero sin conocer la existencia el uno del otro, y sin almacenar información. El servidor no comparte ningún documento con ningún nodo.
- **Criptomoneda:** activo digital diseñado para funcionar como método de intercambio. Emplea la criptografía para mantener las transacciones financieras seguras, controlar unidades adicionales y verificar la transferencia de los activos. Las criptomonedas funcionan con una distribución descentralizada, opuestamente a las monedas digitales centralizadas y a los sistemas bancarios centrales [7].
- **Árbol de Merkle:** en informática, es un árbol en que cada nodo (hoja) está etiquetado con el hash de un bloque de datos, y cada nodo no hoja (rama) está etiquetado con el hash criptográfico de las etiquetas de sus nodos anteriores.
    - Esta tecnología permite una verificación eficiente y segura de los contenidos de grandes estructuras de datos [8].
- **Árbol de Merkle-Patricia:** como en el árbol de Merkle, cada nodo tiene un valor hash. El hash de cada nodo se decide por el valor de hash SHA3 de su contenido. En este árbol, hay más tipos de nodos, aparte de los nodos de rama y los nodos de hoja. También hay nodos de extensión, que son un nodo optimizado del nodo de rama. La diferencia entre cada tipo de nodo es un prefijo a su ruta [9].
- **Libro mayor distribuido (distributed ledger):** consenso de replicado, compartidos y sincronizados datos digitales propagados geográficamente a través de múltiples lugares, países o instituciones. Para obtener el consenso, se emplea un algoritmo, por lo que una red P2P es necesaria. Un ejemplo de libro mayor distribuido es el empleado por la tecnología de cadena de bloques [10].

- **Máquina de Touring:** modelo matemático de computación que define una máquina abstracta. Esta máquina manipula una cadena de símbolos de acuerdo con las normas establecidas en una tabla. Una máquina de Touring es completa cuando es capaz de gestionar todas las tareas realizabless por un ordenador.
    - No todos los lenguajes de programación son máquinas completas de Touring, debido a las limitaciones de memoria de la mayoría de estos.
- **Ofrecimiento inicial de moneda (Initial Coin Offering, ICO):** en el mundo de la cadena de bloques, es un proceso en el que una nueva aplicación, moneda o servicio busca inversores para su lanzamiento. Como intercambio por la inversión, los inversores reciben un token específico de una criptomoneda, con el objetivo de devolver una gran rentabilidad en el futuro, cosa que debería suceder si el lanzamiento es exitoso [11].

Capítulo 2. Cadena de bloques, explicación teórica

Las cadenas de bloques, como su propio nombre indica, son listas crecientes de registros (bloques) interrelacionados entre sí empleando criptografía (cadenas). Cada bloque contiene la siguiente información:

- Un hash criptográfico, empleado como link del anterior bloque.
- Una marca temporal.
- Datos de transacción, normalmente representados como la raíz de un árbol de Merkle hasheado.

Las cadenas de bloques, por su estructura, son resistentes a modificaciones de datos. Son libros mayores distribuidos abiertos que recogen transacciones entre dos partes eficientemente de forma verificable y permanente. Así mismo, pueden ser programados para automatizar las transacciones [12].

Las cadenas de bloques, normalmente, son empleadas en redes P2P con protocolos para la comunicación entre nodos y la validación de nuevos bloques. Todos los datos registrados no pueden ser alterados sin alterar los bloques siguientes. Esta acción requiere el acuerdo de gran parte de la red.

Debido a esto, las cadenas de bloques se consideran seguras por diseño, y son un gran ejemplo de sistemas computacionales distribuidos con gran tolerancia a errores en avalancha (o defecto bizantino, información imperfecta debido al fallo de componentes) [13].

Esta tecnología fue descrita por primera vez por Satoshi Nakamoto en 2008, con el propósito de emplearla como libro mayor público de transacciones de la criptomoneda Bitcoin, la primera moneda digital que resolvió el problema del doble gasto contable sin una autoridad confiable avalista.

2.1 Estructura

Una cadena de bloques es un libro mayor público descentralizado, distribuido y público empleado para registrar transacciones entre muchos ordenadores, por lo que cualquier registro no puede ser modificado retroactivamente sin alterar los bloques consecuentes, dando a los participantes involucrados la habilidad de verificar y auditar transacciones independientemente [14].

La base de datos de una cadena de bloques es gestionada autónomamente con una red P2P y un servidor distribuido para las marcas de tiempo. Cada nodo es autenticado por colaboración masiva.

El empleo de cadenas de bloques elimina la característica de infinitas copias de un activo digital, confirmando que cada unidad de valor fue transferida únicamente una vez, resolviendo el problema del doble gasto contable [15].

2.1.1 Bloques

Los bloques contienen conjuntos de transacciones válidas que están codificadas en un árbol Merkle. Cada bloque incluye el hash criptográfico del bloque anterior, creando un enlace o cadena entre bloques.

Las iteraciones continuas del proceso proporcionan la integridad del sistema de los bloques anteriores, hasta el primer bloque [16].

El proceso de creación de un bloque se denomina minado. Los usuarios que crean nuevos bloques se llaman mineros.

Durante el proceso de minado, los mineros están aprobando transacciones y codificándolas. El primer minero que crea un hash que codifica todas las transacciones aprobadas, y lo comparte con el resto de la red (añadiendo el nuevo bloque a la cadena), es premiado con un premio económico (normalmente una cantidad de criptomoneda, dependiendo de la plataforma) y una comisión por las transacciones que codificó.

Crear un nuevo bloque es un proceso de varios minutos que requiere una gran cantidad de recursos. Además, existe una gran competitividad entre mineros, quienes luchan contrarreloj por ser el minero más rápido en encontrar un hash válido [17].

Para crear un bloque, toma una cantidad de tiempo dependiendo de la plataforma. Por ejemplo, en la plataforma Bitcoin, se necesitan unos 10 minutos, mientras que en otras plataformas se necesitan entorno a 15 segundos. Esta cantidad de tiempo se denomina bloque de tiempo (block time), y es el tiempo que toma una transacción en ser completada. Un bloque de tiempo corto significa que las transacciones pueden realizarse en poco tiempo [18].

A veces, bloques separados pueden crearse en el mismo momento, creando una bifurcación temporal. Las razones de que esto suceda pueden ser cambios en las reglas de la plataforma (o en el software) acordadas por la comunidad. Si estos cambios son aceptados por el software antiguo, se denomina bifurcación suave (soft fork). Pero, si el software antiguo no acepta el nuevo software debido a la existencia de incompatibilidades, se denomina bifurcación dura (hard fork). Si un grupo de nodos emplea el nuevo software, mientras que el resto de nodos emplea el viejo, ocurre una separación. Si la bifurcación es exitosa, dos sistemas diferentes de origen común pueden trabajar independientemente, tal y como ocurrió con la división entre Ethereum y Ethereum Classic en el año 2016 [19].

2.1.2 Descentralización y seguridad

Almacenando datos en la red P2P de forma compartida, las cadenas de bloques eliminan el riesgo inherente de mantener centralizado el almacenamiento de datos. Para mantener los datos, se emplean redes distribuidas y comunicaciones ad-hoc.

La tecnología de cadena de bloques, debido a su distribución por nodos, no tiene puntos centralizados de vulnerabilidad, por lo que los hackers informáticos no pueden explotarlos.

Como método de seguridad, la tecnología de cadena de bloques emplea un sistema de criptografía de clave pública. Este sistema funciona con claves públicas o direcciones en la cadena de bloques. Cada propietario tiene su propia clave pública. Todas las transacciones se registran con su cadena pública, por lo que la cantidad de tokens que cada propietario tiene puede especificarse buscando las transacciones realizadas en los bloques, vinculando con las direcciones públicas.

Para realizar transacciones, se necesita una clave privada. Cada dueño tiene también una, que debe mantener secreta por la seguridad de sus activos criptográficos. Esta clave privada es totalmente necesaria para tener acceso a los activos digitales, y operar con ellos.

Las claves públicas y privadas son dos cadenas diferentes, creadas con números y letras aleatorias.

Debido a la distribución y uso de las claves públicas y privadas, los datos almacenados en la cadena de bloques se consideran generalmente incorruptibles [20].

Cada nodo en el sistema posee una copia de la cadena de bloques. La calidad de los datos depende de la confianza computacional y la replicación masiva de datos. Ningún nodo es más o menos confiable que otro, y no existe una copia centralizada de los datos.

Todas las transacciones son entregadas a la red por software. Los nodos mineros validan transacciones y las añaden al bloque que están creando, y cuando el bloque ha sido creado, se reparte al resto de nodos.

Para entrar al sistema como nodo, en la mayor parte de sistemas de cadenas de bloques, no se requiere permiso. Por ello, cualquiera puede entrar al sistema y empezar a añadir nuevos bloques sin la aprobación del resto de nodos. Sin embargo, algunas redes (como Bitcoin) requieren una prueba de trabajo para entrar al sistema como nodo [21].

2.2 Tipos

Hoy en día, existen tres clases distintas de redes de cadenas de bloques:

- **Cadenas de bloques públicas**: no tiene restricciones de acceso. Cualquiera con una conexión a Internet puede mandar transacciones a ella, al igual que convertirse en un nodo. Las transacciones son públicas. Un ejemplo es Bitcoin.
- **Cadenas de bloques privadas**: requieren de permiso. La única forma de acceder a la misma es por invitación de los administradores de la red. Las transacciones son privadas. Ejemplos de cadenas de bloques privadas son Hyperledger (Fundación Linux), R3 (bancario) o Ripple (protocolo de transferencia de dinero).
- **Cadenas de bloques híbridas**: combinación entre redes privadas y públicas. Todos los nodos entran al sistema por invitación, pero todas las transacciones son públicas, independientemente de su pertenencia a la plataforma. Algunos ejemplos son BigchainDB (proveedor de servicios de cadenas de bloques) o Evernym (identidad auto-soberana) [22].

2.3 Aplicaciones

La tecnología de cadenas de bloques puede ser empleada con muchas finalidades. Su uso primario es como libro mayor distribuido para criptomonedas, como Bitcoin o Ethereum. Estos son algunos ejemplos de aplicaciones de la tecnología de cadena de bloques:

- **Criptomonedas:** empleando las cadenas de bloques, se pueden registrar las transacciones de criptomonedas.
- **Contratos inteligentes**: contratos propuestos que pueden ser ejecutados parcial o totalmente sin interacción humana. Estos contratos se basan en la cadena de bloques [23].
- **Servicios financieros**: la industria financiera está implementando libros mayores distribuidos en sus servicios bancarios. Bancos como UBS están invirtiendo en laboratorios de investigación centrados en la tecnología de cadena de bloques.
- **Videojuegos:** algunos videojuegos basados en la cadena de bloques, como Huntercoin o Cryptokitties [24].
    - Este último videojuego ha demostrado que los videojuegos basados en la cadena de bloques pueden ser catalogados como activos digitales, debido a que, en 2017, un personaje fue vendido por USD100,000.
    - Además, la tecnología de cadena de bloques puede ser útil para plataformas de pagos en juegos como el FIFA Ultimate Team o el Fortnite, donde los jugadores pueden adquirir artículos especiales para sus personajes.
- **Cadenas de suministros**: la cadena de bloques puede ser empleada como método de monitorización de cadenas de suministros en la industria, para mejorar el servicio de rastreo o los estándares de calidad.
- **Compilador de datos:** las cadenas de datos pueden usarse como método de registro de datos de ventas, rastreo de uso digital y pagos para creadores de contenido (músicos, por

ejemplo), seguros P2P, seguros paramétricos, microseguros, votaciones online (Tezos), etc… [25][26].

- **Economía compartida**: modelo de mercado híbrido con intercambios nodo a nodo. Las transacciones son, a menudo, facilitadas por los servicios online de una comunidad. Basándose en la premisa de que cuando la información sobre bienes es compartida, el valor de dichos bienes puede incrementarse para negocios, para individuales y para la comunidad y la sociedad en general [27]. Las cadenas de bloques pueden ser plataformas de libros mayores para estas comunidades que comparten bienes para mejorar su valor para la sociedad.
- **Internet de las cosas (IoT):** concepto de extender la conectividad de Internet más allá de las plataformas computacionales convencionales. Estos "nuevos" dispositivos pueden comunicarse e interactuar con otros a través de Internet, y pueden ser temotamente monitorizados y controlados [28].

Capítulo 3. Ethereum, explicación teórica

Ethereum es una plataforma descentralizada de código abierto que permite a los usuarios desarrollar contratos inteligentes entre dos partes, basado en la tecnología de cadena de bloques. Cada desarrollador puede crear y publicar aplicaciones distribuidas en la plataforma [29].

La plataforma Ethereum proporciona un token o moneda de intercambio para realizar transacciones, llamado Ether. El Ether puede ser empleado para realizar transferencias y como premio o incentivo para los mineros de la plataforma [30]. Además, la plataforma proporciona una máquina virtual descentralizada, llamada Máquina Virtual de Ethereum (Ethereum Virtual Machine, EVM), que ejecuta scripts empleando una red de nodos. Este sistema es casi una máquina completa de Touring, debido a la limitación en el número de posibles operaciones computacionales que puede desarrollar, determinado por el gas, concepto que explicaremos después.

Ethereum fue propuesto por Vitalik Buterin en el año 2013, y su desarrollo fue financiado por donaciones vía Internet en 2014. En 2015, la plataforma entró en funcionamiento.

3.1 Plataforma

La plataforma Ethereum tiene cuatro elementos claves que la distinguen de otras plataformas basadas en cadenas de bloques. Estos elementos son:

- La máquina virtual de Ethereum.
- El Ether.
- El gas.
- Las aplicaciones y el desarrollo de software, como los contratos inteligentes.

3.1.1 Máquina Virtual de Ethereum (EVM)

La máquina virtual de Ethereum es un entorno de ejecución para contratos inteligentes en Ethereum. Es una pila de registro de 256 bits, diseñada para ejecutar el mismo código exactamente como se pretende. Se ha implementado en algunos lenguajes de codificación, como C ++, Java, JavaScript o Python. Después de codificar en estos idiomas, se compila en el bytecode EVM, el idioma real con el que trabaja EVM [31].

La máquina virtual de Ethereum es una máquina virtual orientada a la seguridad. Está diseñada para ejecutar código no confiable por una red global de ordenadores. Para ejectutarla de forma segura, impone varias restricciones:

- Cada operación computacional ordenada en la ejecución de un programa debe pagar previamente una tasa, para así prevenir ataques de denegación de servicio (DNS).
- Los programas únicamente pueden interactuar unos con otros transmitiendo un único array de bits de longitud arbitraria. No pueden acceder a los estados de otros programas.
- La ejecución de un programa es aislada, por lo que un programa EVM únicamente puede acceder y modificar su propio estado interno, y activar la ejecución de otros programas EVM.
- La ejecución de los programas es totalmente determinista, y produce idénticas transiciones de estado para cualquier implementación que comience en un estado idéntico [32].

La máquina virtual de Ethereum es responsable de manejar el estado interno y las computaciones en la red. Además, debe manejar la información de la cuenta en relación con las direcciones, los saldos, el precio actual del gas y la información de bloque.

El EVM puede ser considerado como una máquina de estado. Las máquinas de estado pueden leer una serie de entradas y, en función de esas entradas, cambiar a un nuevo estado. Por lo tanto, en Ethereum, el punto de partida antes de realizar una entrada está en blanco.

A medida que se realizan las transacciones en la red, el estado cambia. Pero, para cambiar su estado, la transacción debe ser válida. Las transacciones se consideran válidas cuando se validan con éxito a través del proceso de minería.

Este proceso de minería de datos, como en otras plataformas de cadenas de bloques, involucra a los nodos mineros de la plataforma, quienes compiten entre sí para ser el primer creador del bloque y llevarse una recompensa económica.

La máquina virtual Ethereum debe realizar un seguimiento de:

- **Estados de las cuentas**: las cuentas en Ethereum pueden ser subdivididas en:
    - **Cuentas poseídas externamente**: controladas por claves privadas. No existe un código asociadas a estas. Son capaces de enviar mensajes a otras cuentas poseídas externamente y a otras cuentas de contrato, empleando una transacción firmada digitalmente y una clave privada. Sin embargo, para la comunicación con una cuenta asociada a un contrato, tanto el contrato como la comunicación se ejecutan, por lo que acciones no incluidas en el contrato pueden ser ejecutadas, como la transferencia de tokens. Las cuentas poseídas externamente son cuentas proactivas.
    - **Cuentas asociadas a contratos**: controladas por su propio código asociado a ellas. No son capaces por sí mismas de iniciar una nueva transacción. Por tanto, son cuentas reactivas.

  Para clasificar cada clase de cuenta, existen tres elementos:
    - **Nonce:** su valor depende del tipo de cuenta. En las cuentas poseídas externamente, este valor es igual al número de transacciones enviadas por la misma. En las cuentas asociadas a contratos, el valor es igual al número de contratos creados por la cuenta.
    - **Balance:** igual a la cantidad de Weis poseídos por la cuenta del contrato. ($10^{18} Wei = 1 Ether$)
    - **CodeHash:** valor inmutable del hash asociado a la cuenta.
- **Estado mundial:** consiste en una asignación entre entre los identificadores de dirección de 160 bits y el estado de la cuenta, que se mantiene en un árbol Merkle Patricia. Esta estructura de datos se compone de un conjunto de nodos que cumplen:
    - Tienen una gran cantidad de hojas o nodos en la base del árbol alojando los datos subyacentes.
    - Tienen una serie de nodos intermedios, donde cada nodo es el valor de hash de dos nodos hijos.
    - Tienen un solo valor hash de raíz formado a partir del hash de los dos nodos secundarios anteriores, que representan la parte superior de la estructura de árbol.
- Estado de almacenamiento: mantenido por el entorno de ejecución del EVM, contiene información específica de estado.
- Información de bloque: requerida para apoyar una transacción. Cada bloque incluye:
    - **Hash de bloque**: el hash del último bloque previamente completado.
    - **Coinbase**: dirección del destinatario.
    - **Marca de tiempo**: del bloque actual.
    - **Número:** del bloque actual.
    - **Dificultad:** del bloque actual.

  - **Límite de gas**: del bloque actual.
- **Información del entorno de ejecución:** información empleada para ejecutar una transacción. Incluye:
  - **Precio del gas:** precio actual especificado por el precursor de la transacción.
  - **Tamaño del código:** tamaño de la base de código de la transacción.
  - **Llamador:** dirección de la cuenta que está ejecutando la transacción.
  - **Origen:** dirección del emisor origirnal de la transacción [33].

3.1.2 Ether

El Ether es el token que se usa en la plataforma Ethereum. El Ether puede ser considerado como una criptomoneda debido a que provee de un libro mayor distribuido para transacciones (pese a que popularmente se conoce a la criptomoneda como Ethereum). Con Ether, los usuarios pueden pagar por el gas y las comisiones para completar transacciones.

La validez de cada token está garantizada por la cadena de bloques. Sin embargo, como se ha comentado previamente, Ethereum opera con transiciones de estados. Estos estados no están almacenados en la cadena de bloques, sino en un árbol de Merkle Patricia separado.

El suministro total de Ether a la red ronda los 100 millones, y su capitalización de mercado, a 19 de marzo, es de casi 15 billones de dólares americanos, siendo la segunda criptomoneda más grande del mercado en cuanto a capitalización de mercado (por detrás de Bitcoin). La emisión de Ether anualmente está limitada a 18 millones.

Por cada bloque minado, se crean 5 Ether, y se entregan al minero que creó el bloque. Si otro minero consigue crear otro bloque no incluído en la cadena de bloques, se le recompensa con 3 Ether.

Para medir la cantidad de Ether, se emplean diferentes escalas de nombres:

- Wei
- 1 Lovelace = 1000 Wei.
- 1 Babbage = 1000 Lovelace.
- 1 Shannon = 1000 Babbage
- 1 Szabo = 1000 Shannon
- 1 Finney = 1000 Szabo
- **1 Ether = 1000 Finney**
- 1 Kether = 1000 Ether
- 1 Mether = 1000 Kether
- 1 Gether = 1000 Mether
- 1 Tether = 1000 Gether [34]

3.1.3 Gas

El Gas puede ser considerado como equivalente a una tasa o comisión. Cada transacción realizada en la plataforma Ethereum requiere esta comisión.

El concepto de gas puede dividirse en dos subconceptos:

- Gas como herramienta para pagar por transacciones y operaciones computacionales.
- Precio del gas: la cantidad de Ether que un individuo está dispuesto a pagar por cada unidad de gas. Se evalúa en Wei, equivalente a 10^{-18} Ether.

Por ello, para ejecutar una transacción, el emisor debe establecer un límite de gas y un precio que está dispuesto a pagar por ese gas adjunto a la transacción. Si el emisor se queda sin gas, la transacción es inválida.

El gas es un elemento limitante para la Máquina Virtual de Ethereum. Los bloques minados en la cadena de bloques de Ethereum tienen un límite de gas adjunto, por lo que la cantidad de gas empleado por todas las transacciones no pueden exceder dicha cantidad de gas. Además, algunas transacciones son complejas. Dichas transacciones se convierten en impracticables económicamente debido a la gran cantidad de gas requerida para ejecutarlas. De esta forma, se evitan operaciones computacionalmente costosas que puedan poner en entredicho el funcionamiento de la plataforma.

3.1.4 Aplicaciones y desarrollo de software

Como lenguaje completo de Touring, los desarrolladores pueden escribir código en 7 lenguajes de programación distintos. Las aplicaciones desarrolladas en la plataforma se denominan Aplicaciones Descentralizadas (DApps), y están basadas en la Máquina Virtual de Ethereum y los contratos inteligentes.

Existen multitud de propuestas respecto a posibles aplicaciones desarrollables en la plataforma, relacionadas con los campos de las finanzas, el Internet de las Cosas, apuestas deportivas, videojuegos, etc.

Debido a la gran cantidad de seguidores de la plataforma, la plataforma Ethereum es la plataforma de cadena de bloques con mayor cantidad de Ofrecimientos Iniciales de Monedas (ICO) [35][36].

Mutitud de empresas están dedicando recursos a investigación y desarrollo sobre la tecnología Ethereum. Mayormente, las empresas tecnológicas están experimentando con la cadena de bloques de Ethereum para automatizar y mejorar los procesos de pagos y las actividades mercantiles [37].

3.2 Procedimiento de las transacciones

La mayoría de carteras de criptomonedas almacenan una clave pública y privada, gracias a las cuales pueden realizarse operaciones en las plataformas. En el caso de Ethereum, esto también es así.

Como en otros sistemas de cadenas de bloques, en una transacción, el emisor envía el Ether directamente a la clave pública del receptor. Para realizar dicha operación, el emisor necesita su clave privada.

Desde el punto de vista del usuario, realizar una transacción de Ether es muy similar a enviar un correo electrónico. Para enviar un correo electrónico, el emisor debe entrar en su cuenta. Para ello, necesita una contraseña (clave privada del emisor).

Entoncces, el emisor envía el correo (Ether) dirigido a la dirección de correo electrónico del receptor (clave pública del receptor).

Cuando el receptor recibe el correo electrónico (ether), este sabe que tiene nuevo correo (Ether) en su cuenta (cartera). Pero, para poder gestionarlo, necesita entrar en su cuenta de correo (cartera), y para ello necesita su contraseña (clave privada del receptor) [38].

Al igual que en una cuenta de correo electrónico, es muy importante mantener la contraseña (clave privada) privada y segura, debido a que si un tercero la consiguiera, tendría acceso a la cuenta (cartera con todo el Ether).

La cantidad de Ether no está relacionada al nombre de su dueño, sino a su contraseña pública. Por ello, la mayoría de plataformas de cadenas de bloques (incluida Ethereum) son anónimas.

Los dueños pueden mantener sus claves de diferentes maneras, como un software o incluso escritas en papel. Estas direcciones están compuestas por el prefijo "0x" concatenado con una función hash de 20 bytes (40 dígitos hexadecimales). Para crear el hash, el procedimiento de encriptación empleado es el Keccak-256 (SHA-3).

Como ejemplo de dirección de una cuenta de Ether, la dirección de la cartera fría de la casa de cambio de criptomonedas Poloniex es "0xb794F5eA0ba39494cE839613fffBA74279579268". Esta dirección se determina empleando como entradas el nonce y el nonce de la creación de la transacción [39].

Como en otras plataformas de cadenas de bloques, todas las transacciones válidas se encriptan en un nuevo bloque.

3.3 Organizaciones Autónomas Descentralizadas (DAO)

Las Organizaciones Autónomas Descentralizadas (DAO) son organizaciones representadas por reglas codificadas en programas de ordenador que son transparentes y controladas por participantes de la organización, y no influenciadas por gobiernos centrales.

Estas reglas se guardan en la cadena de bloques, y su estado legal es incierto [40].

El objetivo principal de las DAO es proveer de un libro mayor digital seguro para rastrear interacciones financieras a través de Internet, para así prevenir falsificaciones, gracias a una marcación temporal confiable y la distribución de las bases de datos.

Como consecuencia de ello, una tercera parte confiable se convierte en innecesaria, simplificando el proceso y convirtiendo las transacciones financieras en más baratas.

Con una correcta y bien organizada DAO, las personas se pueden convertir en innecesarias en algunas tareas en plataformas como Ethereum. La hipótesis fue propuesta por Vitalik Buterin, uno de los cofundadores de la plataforma.

Algunos ejemplos de DAO son Dash, The DAO y Digix [41][42].

Sin embargo, debido a su incierto estado legal, las implicaciones de este estado en los contratos escritos y algunos problemas de seguridad, las DAO no están creciendo a la velocidad que cabría esperar de ellas. Por ello, siguen en una fase de investigación, al igual que Ethereum y el resto de plataformas de cadenas de bloques.

Capítulo 4. Contratos inteligentes, explicación teórica

El concepto de contrato inteligente fue ideado por primera vez por Nick Szabo, un catedrático de ciencia computacional de la universidad George Washington en la década de los 90. Definió contrato inteligente como un acuerdo no garantizado por la ley, sino por hardware o software [43].

Con esta clase de acuerdos, cualquier tercera parte con el objetivo de monitorizar el cumplimiento de lo acordado pierde su sentido de existencia debido a que su función la realiza una máquina.

El objetivo de los contratos inteligentes es dotar de una seguridad superior al de los contratos tradicionales y reducir los costes de transacción.

El concepto que ideó Szabo se ha desarrollado hasta nuestros días, momento en el que se entiende que un contrato inteligente es cualquier tipo de programa informático aplicado a la tecnología de cadena de bloques. Hoy en día, las plataformas Bitcoin y Ethereum son los ejemplos más conocidos de plataformas capaces de aplicar los contratos inteligentes en sus cadenas de bloques.

- En la plataforma Bitcoin, es posible implementar contratos inteligentes para crear cuentas multifirma, canales de pago, entre otras.
- En la plataforma Ethereum, existe una gran estructura relativa a la implementación de contratos inteligentes.

Con la tecnología de cadena de bloques, todos los contratos implementados en los bloques son públicos para todos los usuarios. Con esta transparencia, se pueden poner de relieve algunos errores de código o diseño. Por tanto, es importante garantizar un gran nivel de seguridad antes de publicar un contrato.

Pese a toda esta transparencia, las partes de un contrato se mantienen anónimas para el resto de la comunidad.

4.1 Cómo funcionan

Un buen símil de cómo funciona un contrato inteligente es el funcionamiento de una máquina expendedora.

En una máquina expendedora, si alguien quiere comprar un vaso de café, primero, debe definir los parámetros del mismo, como la cantidad de azúcar, leche, tipo de café, etc. Una vez definido, se introduce el dinero necesario para pagar el café. Finalmente, se sirve.

Un contrato inteligente funciona de manera similar. Se introducen los parámetros necesarios para que el código pueda trabajar, y una vez definido, se paga el Ether necesario para ejecutar la transacción.

En este proceso, existe un contrato predefinido (o acuerdo) definido por el programador de la máquina expendedora (programador del contrato inteligente), con unos posibles valores de entrada elegibles por el cliente. Cuando el evento desencadenante del proceso ocurre (definir los parámetros, introducir la cantidad de dinero suficiente), el contrato se ejecuta por si mismo. Finalmente, el cliente obtiene su producto o servicio.

Por tanto, para que un contrato se ejecute por si mismo, es necesario que ocurra un evento desencadenante.

4.2 Beneficios del uso de contratos inteligentes

Algunos de los beneficios que proporciona el uso de contratos inteligentes son:

- **Autonomía**: terceras partes como abogados no son necesarios.

- **Confianza**: los documentos están encriptados en un bloque de la cadena. Por ello, y combinando esto con la naturaleza distribuida de la cadena de bloques, la pérdida de información es casi imposible.
- **Valor de reserva**: la posibilidad de un fallo catastrófico del sistema (y de perder información) es muy baja.
- **Seguridad**: debido a su arquitectura, hackear un bloque de la cadena es muy difícil Por ello, es muy difícil hackear un contrato inteligente.
- **Velocidad:** la automatización de los procesos puede llegar a ahorrar gran cantidad de trabajo de oficina.
- **Ahorro:** al no existir la necesidad de terceras partes confiables, su tasa o parte de la transacción es innecesaria. Esto ahorra dinero a las partes del contrato. Además, debido a la automatización, gran cantidad de trabajo desaparece, ahorrando también tiempo.
- **Exactitud:** lo escrito en el código del contrato será lo que se ejecute. Problemas relacionados con errores humanos se minimizan a los errores del programador que, para aportar un servicio de calidad, debe minimizarlos [44].

4.3 Aplicaciones

Las posibles aplicaciones de los contratos inteligentes aplicados a la cadena de bloques son casi infinitos. A continuación, se recogen algunas aplicaciones:

- Sistemas electorales: a partir del uso de contratos inteligentes, se puede proporcionar servicio a un proceso electoral con garantías, dado que los votos serían públicos a la par que anónimos.
    - Como ejemplo, para el pasado referéndum de independencia de Cataluña del 1 de octubre de 2017, se diseño un sistema de votación basado en contratos inteligentes en Ethereum, pese a que finalmente el creador del mismo decidió no aportarlo al proceso [45].
- Gestión empresarial: los contratos inteligentes pueden resultar útiles en el mundo de los negocios, debido a su exactitud, transparencia y automatización. Además, también puede ahorrarse tiempo, ya que pueden evitarse retrasos debido a la necesidad de aprobaciones por parte de los equipos directivos. Además, discrepancias de carácter subjetivo respecto a la aplicación del contrato desaparecen.
- Cadenas de suministro: los contratos inteligentes pueden ser útiles en las cadenas de suministro, debido a que están basados en la lógica if-then, ahorrando tiempo y recursos en fábricas y servicios de reparto.
- Seguros: gracias a los avances en los sensores tecnológicos, la causa de un accidente puede ser detectada automáticamente, posibilitando que un contrato de seguros pueda ejecutarse por sí mismo.
- Alquiler: con contratos inteligentes, los servicios de alquiler (coches, apartamentos, etc.) pueden realizarse de forma automática.

Resumiendo, en todos los sectores, la aplicación de contratos inteligentes puede ayudar en la automatización de servicios, facilitando las actuaciones a los consumidores finales y a las empresas.

Capítulo 5. Caso a resolver

5.1 Sistema de reserva y pago de cápsulas de un hotel cápsula

Se pretende implementar un sistema informático para la gestión y pago de un hotel cápsula, así como el hardware necesario para su utilización.

Un hotel cápsula es un establecimiento hotelero muy típico en algunos países asiáticos como Japón, diseñados para una estancia de tiempo reducida. Dichos sistemas se crearon para los viajeros que tenían que hacer noche en los alrededores de una estación, o que habían padecido algún retraso en sus trenes o vuelos.

Las habitaciones o cápsulas de dichos establecimientos son pequeñas habitaciones (en nuestro caso, 2 metros de largo, 1,5 metros de ancho y 1 de alto), proveídas de diferentes elementos, como pueden ser un colchón, televisión, enchufes, conexión a Internet, luz, etc.

Figura 2. Imagen de una cápsula de hotel.

En nuestro caso, se nos pide desarrollar el sistema de alquiler de un hotel con 10 cápsulas, tanto por su parte de hardware como de software. Dicho sistema permitirá el pago mediante Ether. Las cantidades a cobrar vendrán en función de la cantidad de minutos que el usuario emplea el sistema. Además, en el caso de exceder una cantidad determinada de minutos, el sistema cobrará una penalización al usuario.

El procedimiento de alquiler que se nos pide sigue los siguientes pasos:

- El usuario accede a la zona de las cápsulas.
- Una vez llegue a una cápsula libre, tiene ante él una pantalla, con la que podrá interactuar. En el caso de que la cápsula esté libre, el usuario escaneará, a través de una cámara ubicada junto a la pantalla, un código QR, a través del cual el sistema podrá conocer la cuenta pública del usuario que desea alquilar una cápsula.
- Una vez el sistema haya reconocido la clave pública del usuario, se iniciará el proceso de alquiler por minutos. Además, el propio sistema iniciará un temporizador, para así contar la cantidad de minutos que el usuario emplea el sistema.

- Para finalizar el alquiler de la cápsula, el usuario escaneará de nuevo su código QR con su clave privada.
- Una vez reconocidas dichas cuentas, el sistema calculará la cantidad monetaria a cobrar, la mostrará por pantalla y gestionará automáticamente dicha transacción.

Las cantidades monetarias que se nos pide aplicar se muestran a continuación:

- Cantidad a cobrar por minuto de alquiler = 0,01 Eth.
- Cantidad a cobrar por gestión del sistema = 0,01 Eth.
    - Con esta cantidad se pretende pagar el consumo de gas de la operación.
- Cantidad a cobrar por uso excesivo de la cápsula = 1 Eth.
- Tiempo máximo de uso = 360 minutos.

En la siguiente imagen, se muestra el alzado, planta y perfil del establecimiento donde se encuentran las cápsulas.

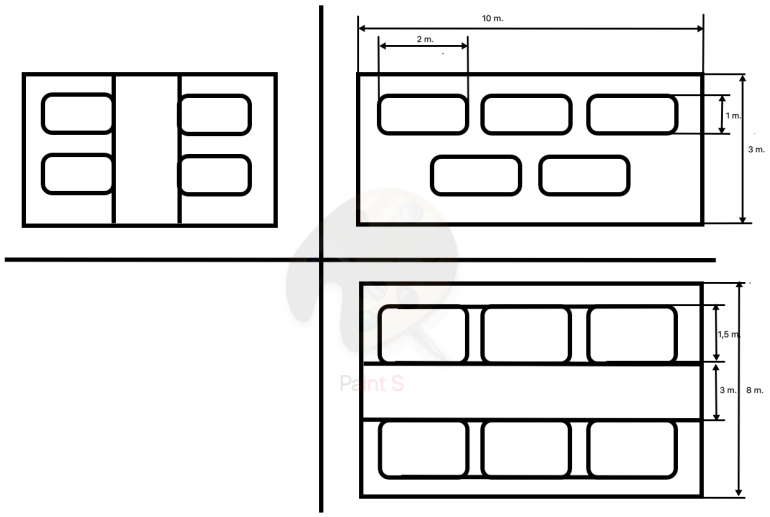

Figura 3. Perspectiva caballera del hotel cápsula.

Se nos proporciona la información de que las paredes del establecimiento están diseñadas con chapa de plástico extraíble, de forma que su extracción es sencilla y no costosa.

Capítulo 6. Solución propuesta

En este capítulo, se detallará en profundidad el proceso realizado para solventar el caso que se ha propuesto en el apartado anterior.

Para resolver el caso propuesto, se han diseñado:

- Un equipo físico para que el cliente pueda interactuar con el sistema. A partir de dicho hardware, el cliente podrá introducir sus claves pública y privada, así como iniciar y finalizar el alquiler de la cápsula.
- Un contrato inteligente preparado para la plataforma Ethereum. Dicho contrato se ha escrito en el lenguaje de programación Solidity, diseñado específicamente para este tipo de códigos.
    - Dicho contrato incluirá todas las funciones que nuestra aplicación front-end ejecutará en el proceso de alquiler.
- Una aplicación front-end, con la que interactuaremos para poder ejecutar el contrato inteligente escrito en la cadena de bloques. Esta aplicación se ha escrito en el lenguaje de programación Python.
    - Dicha aplicación será la que se mostrará por pantalla, y será la propia aplicación la que incluya los parámetros a aplicar al contrato inteligente.

Complementariamente:

- Para simular el comportamiento de nuestros códigos, se ha empleado el simulador de la cadena de bloques de Ethereum Ganache.
    - Ganache aporta información sobre los bloques y transacciones minados durante la ejecución de la simulación como si de la propia plataforma Ethereum se tratara. Además, proporciona al usuario 10 simulaciones de cuentas diferentes de Ethereum para poder simular operaciones, con la ventaja de que no maneja Ether de verdad, por lo que no hay gasto económico real alguno.
- Para evitar posibles fallos de ejecución, es necesario crear un entorno virtual de trabajo.

6.1 Equipo físico

El hardware del sistema propuesto consistirá en una pantalla conectada a una cámara y un pequeño equipo informático, ubicado junto a cada cápsula. Como hardware, se emplearán los siguientes dispositivos:

- Diez Raspberry Pi 3 Model B+ [46].
- Diez Raspberry Pi Universal Power Supply [47].
- Diez Raspberry Pi camera module V2 [48].
- Diez Pi Foundation 7” touchscreen display [49].
- Dos Switches TPLink de 5 puertos Ethernet, Gigabit [50].
- Router (proporcionado por el suministrador de Internet).
- 13 Cables Ethernet NANOCABLE 10.20.0405 de 5 metros [51].
- Un adaptador RJ45 hembra-hembra Equip 221159 [52].

La instalación se realizará de la siguiente manera:

- Al router, a través de dos puertos conmutados, se conectarán dos cables Ethernet, en direcciones opuestas, y se conectarán a un switch de una entrada y cinco salidas. Cada switch estará bajo las cápsulas, uno en cada pared (por dentro). Dado que nuestros cables

son de 5 metros, uno deberá ser empalmado con otro cable Ethernet mediante el adaptador RJ45, para así llegar al switch.

- Del switch saldrán cinco cables Ethernet, los cuales se conectarán al módulo informático de cada cápsula. Habrá un módulo por cápsula, y la ejecución de cada módulo será independiente.
- Cada módulo informático constará de:
    - Una pantalla táctil.
    - Una cámara.
    - Una Raspberry, a través de la cual se ejecutarán las aplicaciones informáticas.
    - Una fuente de alimentación.

A partir del plano en vista caballera del que disponemos, se ha decidido realizar el siguiente montaje.

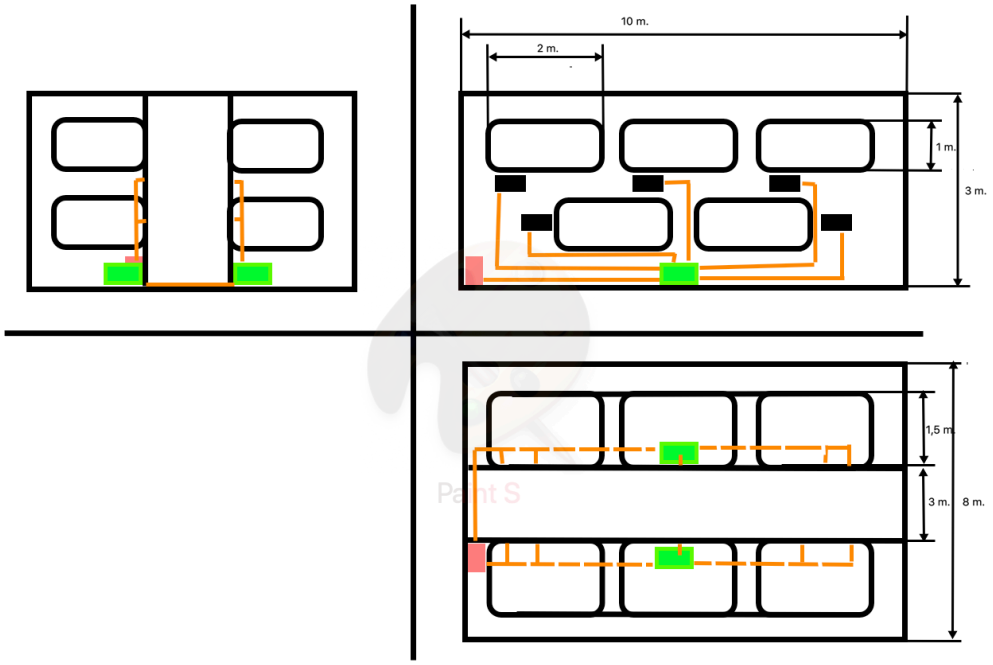

Figura 4. Vista caballera del hotel cápsula con instalación del hardware necesario.

En dicho dibujo, se muestra:

- En color rosa, el router.
- En color naranja, los cables Ethernet.
- En color verde, los switches.
- En color negro, la pantalla junto a la cámara.

Como se puede ver en el diseño propuesto, tanto el router como el cableado y los switches quedan fuera de la vista del cliente (dentro de la pared).

A la vista del cliente, únicamente quedará la pantalla táctil y la cámara, con la cual el cliente podrá interactuar. Dicha configuración quedará clavada en la pared, cada bloque junto a su cápsula correspondiente.

6.2 Preparación del entorno de trabajo

Para desarrollar el software necesario, se empleará un dispositivo con macOs Mojave versión 10.14.3. Una vez haya sido probado y los resultados demostrados satisfactorios, se indicarán los cambios pertinentes para adaptar el software necesario a Raspbian, así como para subir el contrato inteligente a la red principal de Ethereum.

6.2.1 Descarga e instalación de los programas necesarios

Antes de empezar a diseñar el código necesario, se deben instalar varios programas y liberias. A continuación se detallan los pasos a seguir para ello, diseñados para un macOS Mojave versión 10.14.3, sistema operativo con el que se realizarán las simulaciones.

- En primer lugar, se debe instalar el programa informático Homebrew. Homebrew es un gestor de paquetes, que nos será útil para nuestro proceso de instalación de los programas necesarios.
    - Desde la ventana terminal del ordenador, introducir el siguiente comando:
        - $ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" [53].
- Una vez finalizada la instalación, es el momento de instalar Python. Necesitamos la versión 3 del mismo. Para ello:
    - Desde la ventana terminal del ordenador, introducir el siguiente comando:
        - $ brew install python3
- A continuación, debemos instalar el instalador de paquetes para Python (Python Package Installer). Para ello:
    - Desde la ventana terminal del ordenador, introducir el siguiente comando:
        - $ sudo easy_install pip
- Una vez tengamos el pip instalado, es la hora de instalar el virtualenv. Este programa nos dará un entorno virtual para ejecutar nuestros códigos sin problemas. Para ello:
    - Desde la ventana terminal del ordenador, introducir el siguiente comando:
        - $ sudo pip install virtualenv
- Una vez instalado, debemos activarlo. Para ello:
    - Desde la ventana terminal del ordenador:
        - Buscar la carpeta en la que queramos trabajar, mediante los comandos ls (muestra las carpetas) y cd nombre_carpeta (las abre).
    - Una vez encontrada la carpeta, se debe activar el comando:
        - $ source env/bin/actívate [54].
- Ahora podremos instalar la librería web3.py, útil para trabajar con Python y interactuar con un nodo de la plataforma Ethereum. Para instalarlo,
    - Desde la ventana terminal del ordenador:
        - $ pip install web3
- Para nuestro lector QR, debemos instalar las liberías zbar, pyzbar e imutils.
    - Desde la ventana terminal del ordenador:
        - $ brew install zbar
        - $ pip3 install pyzbar

        - $ pip3 install imutils
- Finalmente, nos queda por instalar Ganache. Para ello:
    - Desde su página oficial, podemos descargar el programa informático con una interfaz de uso agradable a la vista. [55]
- Mención especial merece el Remix de Solidity, programa a través del cuál podremos escribir, compilar y subir nuestro contrato inteligente. Se encuentra en la siguiente página web.
    - http://remix.ethereum.org
    - Se debe recalcar que, para realizar las simulaciones en Ganache, la página https da problemas.

6.3 Código del contrato inteligente

En el anexo IV se muestra el código diseñado como contrato inteligente (aplicación back-end), a la vez que comentarios del mismo. Los comentarios se muestran en color verde. Dicho código está diseñado en el lenguaje de programación Solidity, empleando los recursos de Remix disponibles en http://www.remix.org.

Como librería, hemos empleado la librería SafeMath, con la finalidad de evitar problemas de overflow al realizar operaciones matemáticas en nuestro contrato inteligente. En el anexo V se muestra el código [56].

6.4 Código de la interface de usuario

En el anexo VI se muestra el código desarrollado como aplicación front-end o interface. A través de ella, se ejecutarán todas las funciones definidas en el contrato inteligente. Dicho código ha sido desarrollado en el lenguaje de programación Python.

Los comentarios se muestran en verde, como en el contrato inteligente.

En cuanto al código de Python para la lectura de códigos QR en tiempo real, el código tomará la dirección escaneada a partir del código QR proporcionado, y la escribirá en un archivo .csv. Después, el programa de Python que hemos diseñado se encargará de leer dicha dirección del archivo. Cada vez que se escanee un código QR, el archivo .csv se actualizará. Para cerrar la cámara, debe presionarse la tecla ‘q’ Dicho código se muestra en el anexo VII [57].

6.5 Instrucciones de ejecución

Para ejecutar la simulación de nuestra solución en un ordenador con macOS Mojave 10.14.3, debemos aplicar los siguientes pasos:

- En primer lugar, abrir el navegador, y entrar en la página web de Remix.
    - http://remix.ethereum.org
    - Copiar y pegar en dos archivos .sol distintos los códigos renting.sol y SafeMath.sol.
- En segundo lugar, compilar el código de Solidity. Para ello, pinchar en el botón “Start to compile (Ctrl+S)”.

Figura 5. Botón a pinchar.

- En tercer lugar, abrir la pestaña “Details” del programa, y copiar los valores de abi y bytecode. Estos valores deben pegarse en el código de Python. Una vez hecho, compilar el código de Python.

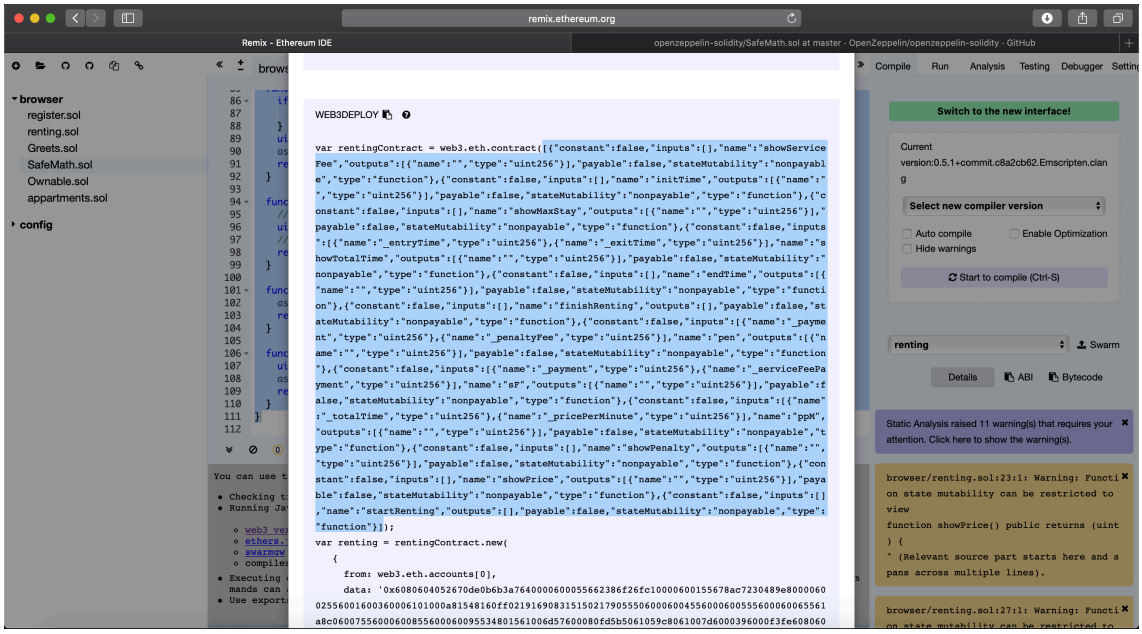

Figura 6. Se muestra el abi subrayado.

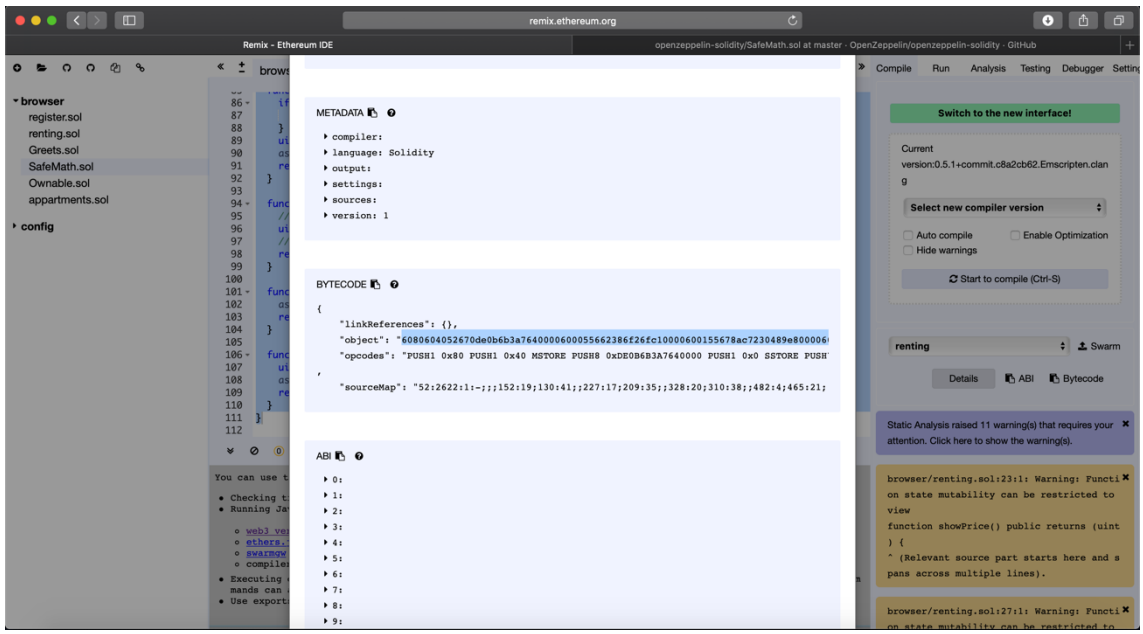

Figura 7. Se muestra parte del bytecode subrayado.

- En cuarto lugar, abrir Ganache, y pinchar en Quickstart.
  - Al abrir Ganache, conviene fijarse en el RPC Server.
  - Este valor marca la dirección del nodo a la que tenemos que conectar nuestro programa de Python. Equivaldría al valor ganache_url de nuestro código.

- Por otro lado, también hay que copiar las claves de las cuentas que queremos usar.
  - En nuestro diseño, empleamos como cuenta propietaria la de index 4.
  - Las claves de los clientes vendrán definidas por el escaneo de códigos QR.

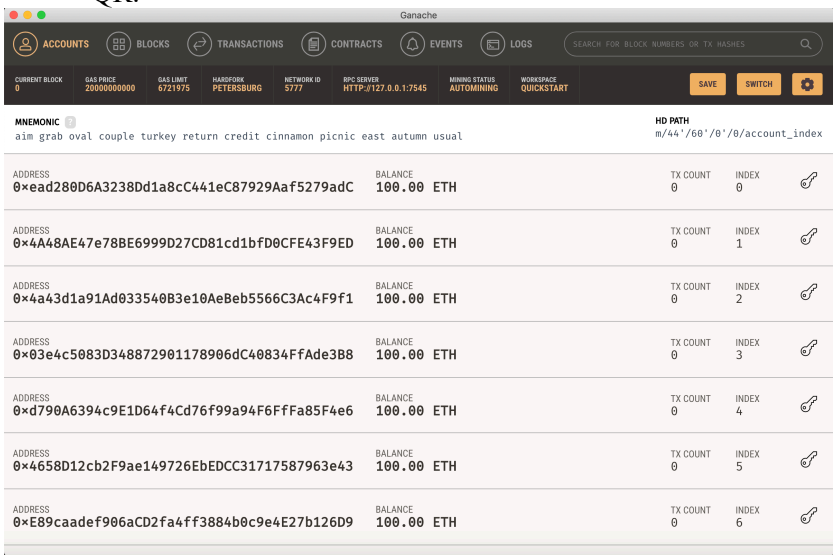

Figura 8. Ganache en ejecución, antes de realizar ninguna transacción.

- En quinto lugar, volviendo al Remix, pulsar en la pestaña "Run", y en la pestaña "Environment", señalar la opción "Web3 provider". Como "Web3 Provider Endpoint", escribir "http://localhost:7545", y pulsar "OK". Pulsar también "Deploy" para renting.sol.
- En sexto lugar, debe activarse un entorno virtual. Se ha explicado previamente cómo hacerse.
  - Una vez creado, en la ventana de terminal se mostrará el nombre del entorno virtual entre paréntesis al inicio de cada línea.

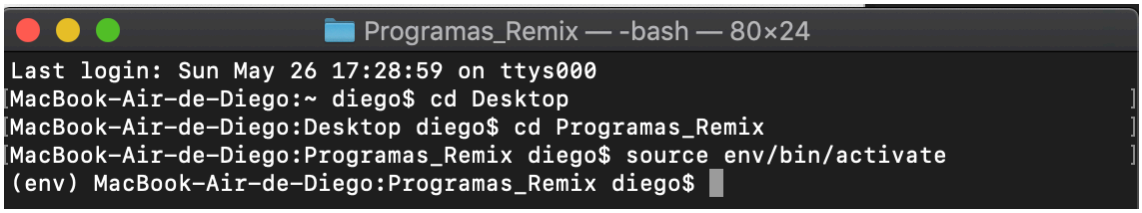

Figura 9. Ejemplo de activación del entorno virtual.

- Por último, en la ventana Terminal del ordenador, ejecutar el programa de Python. Para ello, introducir el comando:
  - $ python3 PythonRenting.py
  - PythonRenting.py es el nombre del archivo.
- Una vez introducido, se ejecuta el programa.

6.6 Resultados

En este apartado, procedemos a simular la solución propuesta. Dicha simulación consistirá en realizar el proceso de alquiler de una cápsula. Para realizar la simulación, se empleará como cuenta cliente la de índice 2 de Ganache, y tal y como se ha definido en la aplicación de Python, la de índice 4 como destinatario. A continuación, se muestran los siguientes códigos QR empleados, correspondientes a las claves pública y privada de la cuenta 2.

Figura 10. Código QR con la clave pública de la cuenta de índice 2 de Ganache.

Figura 11. Código QR con la clave privada de la cuenta de índice 2 de Ganache.

En el anexo I se muestran los códigos QR de todas las cuentas disponibles en Ganache.

En primer lugar, procedemos a iniciar el sistema. Cuando el sistema se inicie, aparecerá la siguiente pantalla:

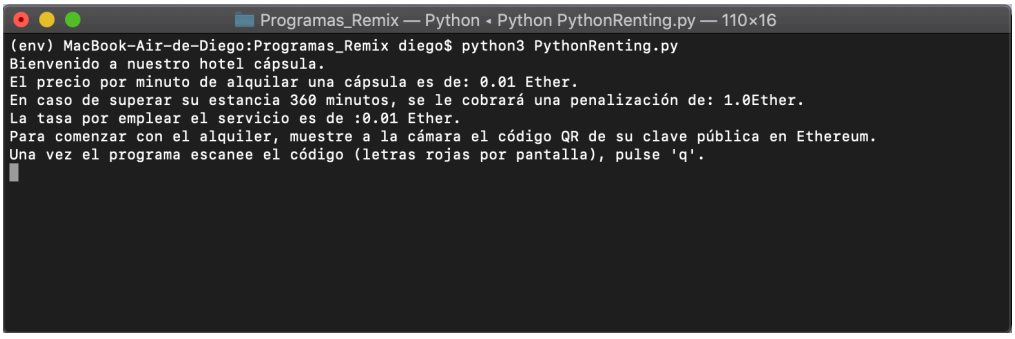

Figura 10. Inicio de la aplicación de alquiler.

Así mismo, se iniciará la cámara automáticamente. Por ello, procedemos a escanear nuestra clave pública con un código QR.

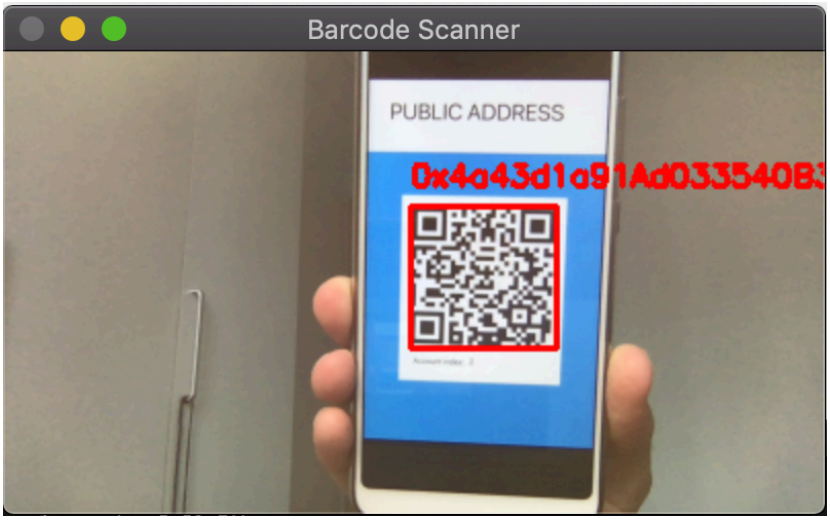

Figura 11. Escaneo de la clave pública de la cuenta de índice 2 con su código QR.

Como indicador de que el sistema ha detectado y decodificado el código QR, aparecerán las letras rojas, tal y como se aprecia en la imagen. Dichas letras son el propio string escaneado. Una vez la clave pública ha sido detectada (y la tecla 'q' pulsada para cerrar la cámara), se inicia el alquiler. La interfaz muestra los siguientes mensajes.

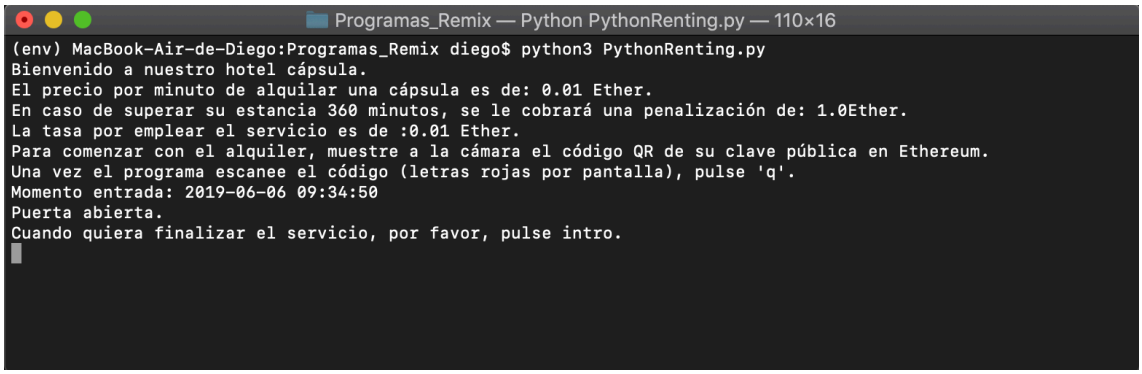

Figura 12. Inicio del alquiler.

En este momento, el alquiler está iniciado, y el temporizador activado. Cuando queramos finalizar el servicio, deberemos de apretar la tecla intro, así como escanear el código QR de la cuenta privada.

Procediendo a finalizar el servicio de alquiler:

```
Programas_Remix — Python ◂ Python PythonRenting.py — 110×16
Una vez el programa escanee el código (letras rojas por pantalla), pulse 'q'.
Momento entrada: 2019-06-06 09:34:50
Puerta abierta.
Cuando quiera finalizar el servicio, por favor, pulse intro.

Por favor, firme digitalmente la operación con su clave Ethereum privada.
Una vez el programa escanee el código (letras rojas por pantalla), pulse 'q'.
```

Figura 13. Inicio del proceso de finalización del alquiler.

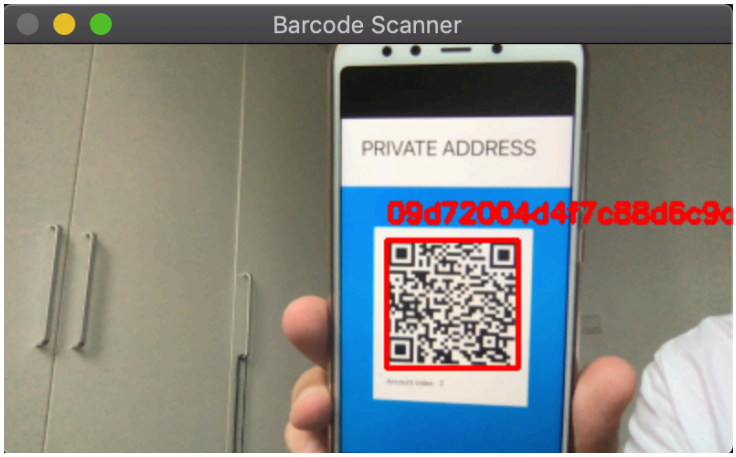

Figura 14. Escaneo de la clave privada de la cuenta de índice 2 con su código QR.

Tras presionar la tecla ‘q’.

```
Programas_Remix — -bash — 110×22
(env) MacBook-Air-de-Diego:Programas_Remix diego$ python3 PythonRenting.py
Bienvenido a nuestro hotel cápsula.
El precio por minuto de alquilar una cápsula es de: 0.01 Ether.
En caso de superar su estancia 360 minutos, se le cobrará una penalización de: 1.0Ether.
La tasa por emplear el servicio es de :0.01 Ether.
Para comenzar con el alquiler, muestre a la cámara el código QR de su clave pública en Ethereum.
Una vez el programa escanee el código (letras rojas por pantalla), pulse 'q'.
Momento entrada: 2019-06-06 09:34:50
Puerta abierta.
Cuando quiera finalizar el servicio, por favor, pulse intro.

Por favor, firme digitalmente la operación con su clave Ethereum privada.
Una vez el programa escanee el código (letras rojas por pantalla), pulse 'q'.
Momento salida: 2019-06-06 10:04:21
Estancia total: 29 minutos.
Puerta cerrada.
La cantidad a pagar es de: 0.3 Ether.
0x6f565568c0b71bc0b85136d52d0ed30930b3e7ca086788a557cb5739aab50681
Pago realizado con éxito.
(env) MacBook-Air-de-Diego:Programas_Remix diego$
```

Figura 15. Finalización del alquiler.

Una vez llegados aquí, el alquiler se ha finalizado con éxito. La estancia ha durado 29 minutos, y con las tarifas establecidas, 0,29 ETH corresponden al tiempo de estancia, y 0,01 a los gastos de gestión (pago de gas por parte del cliente). Por último, se muestra el hash de la operación de pago.

Una vez se ha simulado el proceso, vamos a analizar los efectos que ha tenido dicha operación en nuestra simulación de la cadena de bloques.

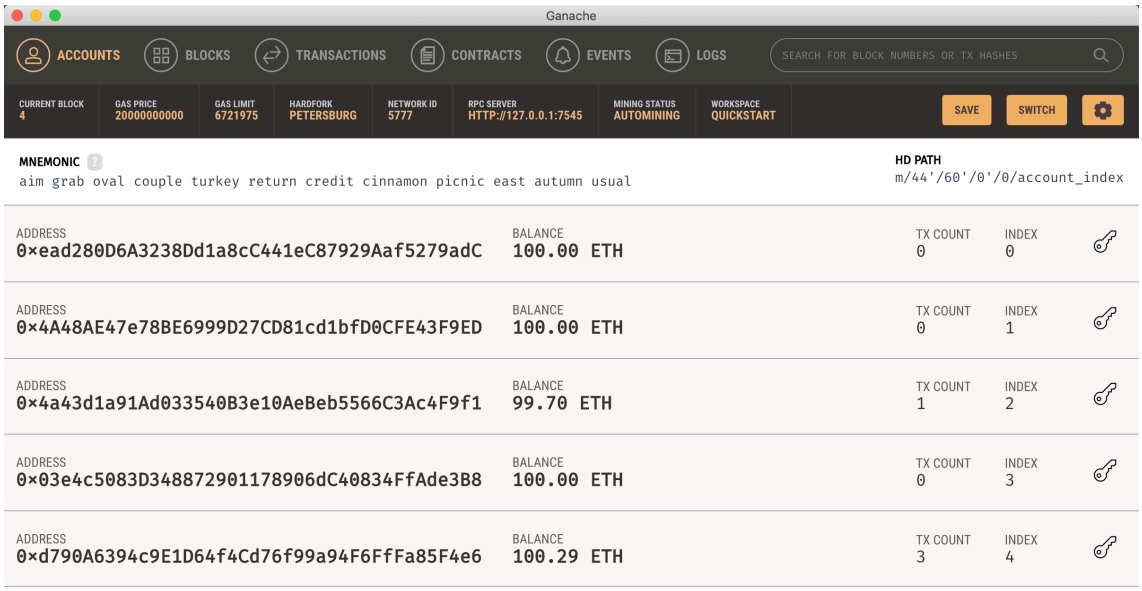

Figura 16. Estado de las cuentas ETH una vez ha finalizado el proceso de alquiler.

Si comparamos el estado de las cuentas ETH una vez finalizado el alquiler, y comparamos con el estado de las mismas antes de iniciar la operación (figura 8), vemos que:

- La cuenta de índice 2 (cliente) ha disminuido su balance en 0,3 ETH. Así mismo, ha realizado una transacción.
- La cuenta de índice 4 (corresponde a la cuenta del establecimiento) ha incrementado su balance en 0,29 ETH.

De esta información, podemos intuir que se ha realizado una transacción de 0,3 ETH desde la cuenta 2 a la 4, y la 4, además ha tenido que pagar de gas 0,01 ETH. Vamos a comprobar esta hipótesis observando el registro de transacciones y los bloques creados.

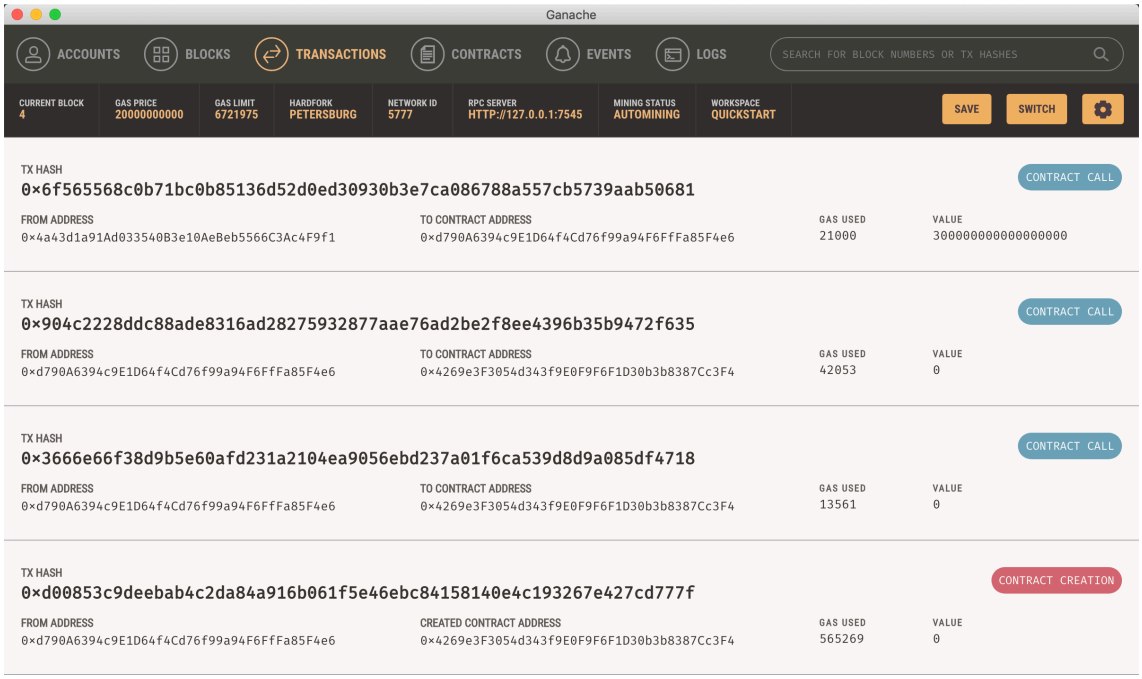

Figura 17. Transacciones realizadas durante la ejecución del alquiler.

Podemos observar que se han realizado 4 transacciones. Leyendo desde abajo, la primera corresponde con la creación del contrato inteligente, tiene de origen la cuenta de índice 4 (hotel, configurada además como cuenta por defecto en nuestro código de Python). El valor de la transacción es 0, pese a haber un consumo de gas.

La segunda transacción registrada corresponde a una transacción con el contrato inteligente, de valor 0. Tiene como origen la cuenta 4 (hotel), y como destinatario el contrato. Esta operación es similar a la tercera transacción, y corresponderían respectivamente al inicio y final del alquiler.

Por último, tenemos una transacción, esta vez con un valor monetario de 0,3 ETH (expresado en Wei), y correspondería al proceso de pago del alquiler entre la cuenta 2 (origen) y la 4 (destinatario).

Ahora, vamos a analizar la información proporcionada por los bloques creados durante el proceso de alquiler.

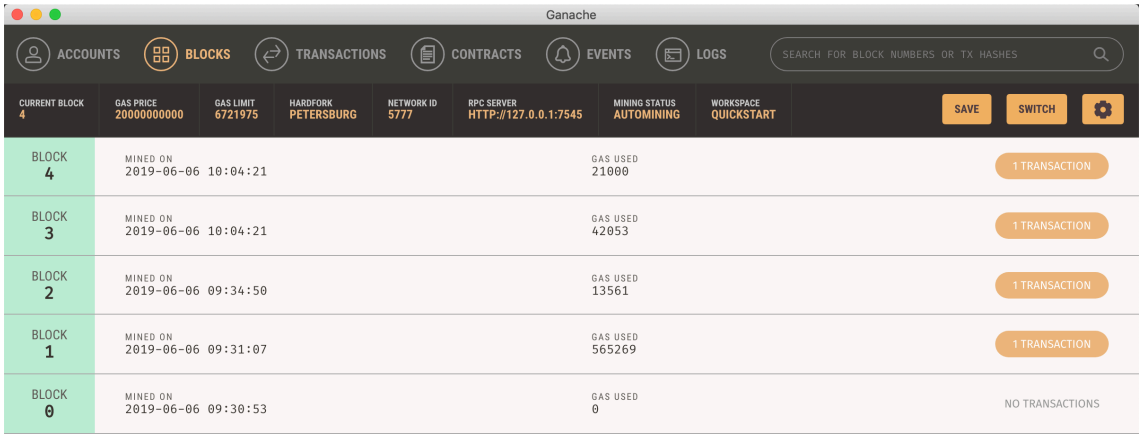

Figura 18. Bloques creados durante la ejecución del alquiler.

Podemos observar que durante la ejecución del contrato se han creado 4 bloques, a partir del bloque madre (0, creado al iniciarse el programa). Cada bloque tiene codificado una transacción de las descritas previamente. Si analizamos en profundidad cada bloque, tenemos lo siguiente.

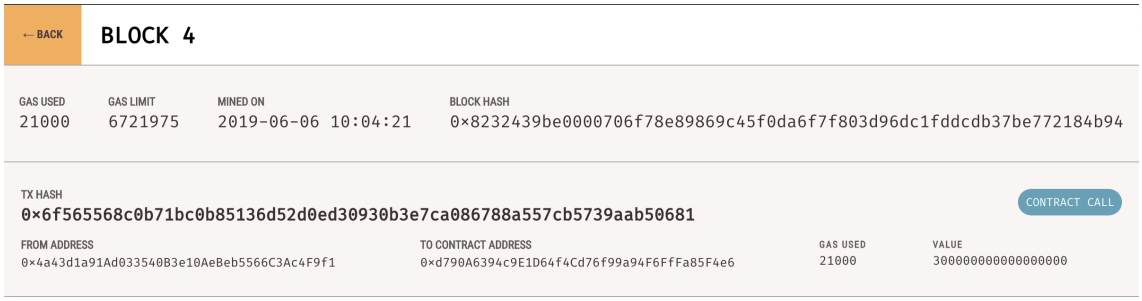

Figura 19. Bloque 4 creado durante el alquiler.

Figura 20. Bloque 3 creado durante el alquiler.

Figura 21. Bloque 2 creado durante el alquiler.

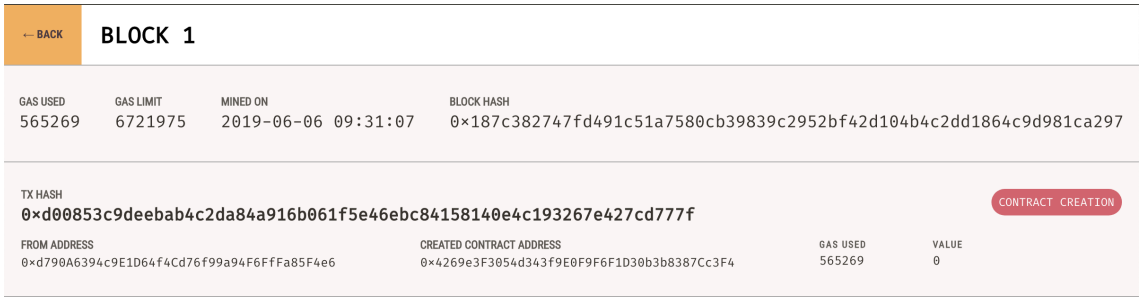

Figura 22. Bloque 1 creado durante el alquiler.

Figura 23. Bloque 0 creado durante el alquiler.

Con toda esta información, podemos confirmar que:

- El bloque 0 se crea una vez iniciada la aplicación Ganache.
- El bloque 1 corresponde a la creación del contrato, la primera transacción realizada.
- El bloque 2 corresponde al inicio del alquiler.
- El bloque 3 corresponde al final del alquiler.
- El bloque 4 corresponde a la transacción monetaria.

Por tanto, podemos confirmar la hipótesis que nos habíamos planteado en primer lugar.

6.7 Cambios a realizar para adaptar la solución a nuestro hardware

Hasta ahora, hemos trabajado con una cadena de bloques simulada (Ganache) y un MacOS. Sin embargo, nuestro hardware consiste en una Raspberry Pi (funcionando con Raspbian), y nuestro contrato inteligente debe subirse a la cadena de bloques de Ethereum.

6.7.1 Cambios para Raspbian

Para ejecutar nuestro código de Python en Raspbian, así como emplear la cámara de la Raspberry Pi, debemos ejecutar cambios en dos campos del sistema:

- Entorno de trabajo: los procedimientos de instalación de bibliotecas, así como la creación del entorno de trabajo difieren respecto a MacOS.
- Código de python: los cambios a realizar son:
    - Cambiar la cámara a emplear para leer los códigos QR. Dicho cambio ha sido señalado en el propio código (barcode_scanner_image.py).
    - Cambiar la url de la cadena de bloques al inicio de la interface (PythonRenting.py). Cambio comentado en el propio código.
        - Para ello, necesitamos conectarnos a un nodo de la red principal. En el código, se emplea Infura [62].
        - El cambio sería descomentar las siguientes líneas:
            - infura_url = “https://mainnet.infura.io/v3/c1c040269cc44901950d8553f8413a1e”
            - web3 = Web3(Web3.HTTPProvider(infura_url))
        - Y comentar las siguientes líneas:
            - ganache_url = "http://127.0.0.1:7545"
            - web3 = Web3(Web3.HTTPProvider(ganache_url))

Profundizando en los cambios referentes a nuestro entorno de trabajo, desde la ventana del terminal:

- Para instalar Python 3.6.
    - $ sudo apt-get update
    - $ sudo apt-get install build-essential tk-dev libncurses5-dev libncursesw5-dev libreadline6-dev libdb5.3-dev libgdbm-dev libsqlite3-dev libssl-dev libbz2-dev libexpat1-dev liblzma-dev zlib1g-dev
    - $ wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tar.xz
    - $ tar xf Python-3.6.5.tar.xz
    - $ cd Python-3.6.5
    - $ ./configure
    - $ make
    - $ sudo make altinstall [58]
- Para instalar el Python Installer Packages.
    - $ sudo apt-get install python3-pip [59]
- Para instalar el entorno virtual de trabajo.
    - $ sudo pip install virtualenv
- Para crear el entorno virtual de trabajo.
    - En la carpeta donde se ubiquen los archivos Python a ejecutar.
        - $ virtualenv env

- Para activar el entorno virtual de trabajo.
  - En la misma carpeta donde se ubique dicho entorno creado previamente.
    - $ source env/bin/activate
  - Al igual que en Mac, se puede acceder a las carpetas con el comando cd nombre_carpeta. [60]
- Para instalar la librería web3.py, y así poder interactuar con el contrato inteligente.
  - $ pip install web3
- Para la librería zbar y pyzbar.
  - $ sudo apt-get install libzbar0
- Para utilizar la cámara, necesitamos la librería OpenCV. Para ello:
  - Descargar los archivos .zip disponibles en:
    - https://github.com/opencv/opencv
    - https://github.com/opencv/opencv_contrib
  - A continuación, ejecutar los siguientes comandos:
    - $ cd ~
    - $ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.0.0.zip
    - $ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.0.0.zip
    - $ unzip opencv.zip
    - $ unzip opencv_contrib.zip
    - $ mv opencv-4.0.0 opencv
    - $ mv opencv_contrib-4.0.0 opencv_contrib
    - $ pip install numpy [61]
- Para la librería imutils, ejecutar en el terminal el siguiente comando.
  - $ pip install imutils
- Para poder emplear la cámara de la Raspberry Pi
  - sudo apt-get install python3-picamera [59]
- Finalmente, para ejecutar el programa:
  - $ python3 PythonRenting.py

6.7.2 Publicar el contrato inteligente a la cadena de bloques de Ethereum

Publicar un contrato en la red principal de Ethereum implica que el creador del mismo manda una transacción a la red desde su dirección personal. En dicha transacción, se transmite todo el código del contrato inteligente, y dicha información se almacenará en la cadena de bloques.

Tras la publicación del contrato, todos los datos almacenados estarán disponibles para futuras transacciones realizadas a través de llamadas a funciones del contrato. La publicación del contrato tiene un coste de transacción, así como un consumo de gas. Por tanto, requiere de una inversión económica inicial.

Para publicar un contrato inteligente a la red principal de Ethereum, tenemos dos opciones:

- Podemos montar y conectar un nodo propio a la red.
- Podemos emplear servicios que dan acceso a la red, como Infura, Metamask o MyEtherWallet [62][63][68].

De estas dos opciones, hemos de tener en cuenta que montar un nodo propio conllevará gastos extra en hardware, así como mantenimiento. Mientras tanto, los servicios online como Infura, Metamask o MyEtherWallet nos liberan de esos posibles contratiempos y gastos extra.

En nuestro caso, proponemos dos soluciones para publicar el contrato. La primera consistirá en los servicios de MyEtherWallet, mientras que la segunda consistirá en utilizar el propio Remix de Solidity.

Para publicar nuestro contrato a la red principal a través de MyEtherWallet, necesitamos una cuenta en Ethereum. La creación de la misma se explica en el anexo II.

Una vez creada y accedida la cuenta, podemos publicar el contrato inteligente a la red principal de Ethereum. Para ello, debemos ir al apartado “Contract”, “Deploy Contract”. Una vez ahí, nos pedirá el Bytecode, el ABI y el nombre del contrato.

Firmamos la transacción, y el contrato estará subido a la red principal.

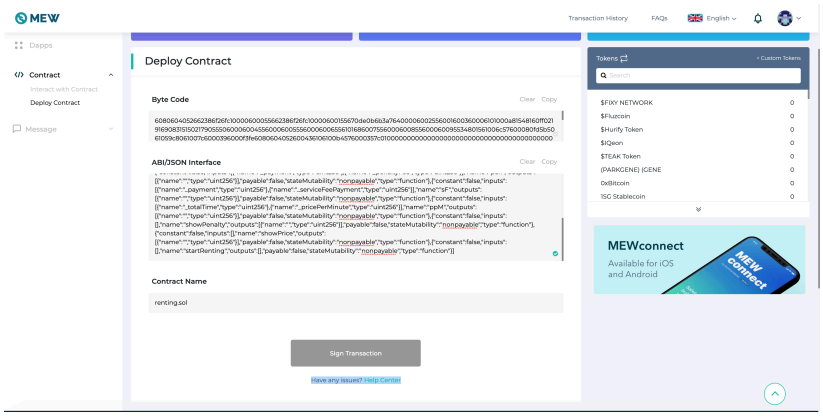

Figura 24. Publicación de contrato a través de MyEtherWallet.

Otra opción más sencilla es, teniendo una cuenta en MetaMask, seleccionar en Remix “Run”, “Injected Web3” en la pestaña “Provider”, y como cuenta seleccionamos una que esté configurada en MetaMask. Pulsaríamos en “Deploy”, y nos saldrá un mensaje de confirmación de la transacción, en el que tendremos que especificar el precio de gas que estamos dispuestos a pagar.

Figura 25. Mensaje de confirmación de la transacción para publicar el contrato.

Confirmamos la operación, y el contrato inteligente quedaría publicado en la cadena de bloques de Ethereum.

6.8 Presupuesto del proyecto

Como parte final del proyecto, vamos a detallar un presupuesto para el diseño e implementación del proyecto. Dicho presupuesto tendrá en cuenta gastos varios, como la instalación, el diseño de los programas o los materiales.

Los gastos concernientes al gas por operaciones como la subida del contrato a la red principal o las transacciones derivadas de las llamadas a funciones del contrato inteligente irán a cargo de la cuenta de Ethereum del hotel cápsula, por lo que no se tendrán en cuenta para la realización del presupuesto.

En primer lugar, recordamos los elementos necesarios para la instalación del sistema:

- Diez Raspberry Pi 3 Model B+.
- Diez Raspberry Pi Universal Power Supply.
- Diez Raspberry Pi Camera Module V2.
- Diez Pi Foundation 7” Touchscreen display.
- Dos switches TPLink de una entrada y cinco salidas Ethernet, Gigabit.
- Trece cables Ethernet NANOCABLE 10.20.0405 de cinco metros de longitud.
- Un adaptador RJ45 hembra-hembra Equip 221159.

Además de todos estos elementos, debemos tener en cuenta servicios como la mano de obra, el cobro por el diseño del proyecto y el servicio mensual de Internet.

Como servicio de Internet, se propone el servicio de fibra de Vodafone de 100 Mbps.

Por tanto, se ha diseñado el siguiente presupuesto.

Elemento	Coste unitario (€)	Cantidad	Coste total (€)	Fuente
Raspberry Pi 3 Model B+	32,82 €	10	328,20 €	[46]
Raspberry Pi Universal Power Supply	8,68 €	10	86,80 €	[47]
Raspberry Pi Camera Module V2	27,94 €	10	279,40 €	[48]
Pi Foundation 7" Touchscreen display	60,00 €	10	600,00 €	[49]
Switch TPLink 1I-5O Ethernet, Gigabit	15,99 €	2	31,98 €	[50]
Cable Ethernet NANOCABLE 10.20.0405 5 metros	4,95 €	13	64,35 €	[51]
Adaptador RJ45 hembra-hembra Equip 221159	4,75 €	1	4,75 €	[52]
TOTAL HARDWARE			**1.395,48 €**	
Mano de obra	50,00 €	1	50,00 €	Estimado
Diseño de los códigos	250,00 €	1	250,00 €	Estimado
TOTAL GASTOS INICIALES			**1.695,48 €**	
Servicio Mensual Internet (tarifa 600Mbps, 7 dispositivos)	**30,99 €**	**1**	**30,99 €**	[64]

Tabla 1. Presupuesto propuesto para el proyecto.

Por tanto, el presupuesto final sería de 1.695,48€ como gastos iniciales, y 30,99€ como gastos mensuales por el servicio de Internet.

Capítulo 7. Conclusión

En primer lugar, tras haber finalizado todos los apartados del proyecto previstos, vamos a analizar la consecución de los diversos objetivos propuestos al inicio del trabajo.

- Comprender cómo funciona la tecnología de cadena de bloques.
    - En este proyecto se ha aportado bastante información acerca de la tecnología de cadena de bloques. De toda esta información, podemos concluir que:
        - La tecnología se basa en bloques interconectados entre sí mediante hashes.
        - Cada bloque se compone del hash del bloque anterior, una marca de tiempo y los datos de las transacciones realizadas.
        - La tecnología es resistente a cambios y hackeos de datos por su naturaleza descentralizada, gozando así de una gran seguridad.
        - Los bloques se crean mediante el proceso de minado, en un entorno de competitividad absoluta entre mineros.
        - La tecnología tiene multitud de aplicaciones.
- Comprender cómo funcionan las plataformas descentralizadas como Bitcoin o Ethereum.
    - En este proyecto se ha estudiado en profundidad la plataforma Ethereum, debido a su gran correlación con los contratos inteligentes. Podemos concluir que:
        - La plataforma tiene cuatro elementos diferenciadores respecto a otras plataformas de cadenas de bloques:
            - La máquina virtual de Ethereum
            - El Ether.
            - El gas.
            - Capacidades de implementación de aplicaciones descentralizadas.
- Comprender cómo pueden ser aplicados los contratos inteligentes en la vida real.
    - En este proyecto se han aportado varios ejemplos de aplicación de los contratos inteligentes en situaciones cotidianas de la vida real y del mundo de los negocios.
    - Además, se ha llegado a desarrollar un contrato inteligente implementable como sistema de reserva de un hotel cápsula.
- Desarrollar un contrato inteligente como solución técnica de una situación de la vida real, empleando la plataforma Etherum y Solidity.
    - No sólo se ha desarrollado un contrato inteligente como propuesta del sistema informático de reserva de un hotel cápsula, sino que además se ha desarrollado e implementado con éxito una interface en Python capaz de interactuar con dicho contrato, poniendo el mismo en práctica. Además, se ha propuesto un hardware con el que se podría poner en práctica el sistema diseñado.
    - El proceso de ejecución del sistema está inspirado en los procedimientos de pago con dinero electrónico que se practican habitualmente en Asia, mediante códigos QR. Dicho sistema es también aplicable al mundo de los pagos con criptomonedas.
        - En este aspecto hemos de señalar una salvedad. En la vida real, únicamente se debería escanear el código QR con la clave pública. Por ello, la transacción debería ordenarse desde el dispositivo móvil del cliente, y no del servicio hotelero.
        - Sin embargo, como forma de simplificar las transacciones monetarias, así como forma de ilustrar el procedimiento de las mismas, hemos implementado dichas operaciones en el propio sistema informático. Así,

el cliente firma las operaciones desde el sistema del hotel, y no desde su propio dispositivo.

- Finalmente, el proyecto ayudará al estudiante a aprender sobre programación con la tecnología de cadena de bloques. Este es el objetivo principal del proyecto, debido a su creciente importancia en el mercado laboral internacional.
    - Por lo comentado en los dos últimos puntos, consideramos que el objetivo principal del proyecto se ha cumplido.

En segundo y último lugar, podemos decir que en el proyecto se ha implementado con éxito un contrato inteligente con su correspondiente interface para posibilitar la aplicación del mismo, en un caso que podría ser aplicable en la vida real. Además, se ha aportado información a través de un ejemplo práctico de la información que manejan los bloques de la plataforma, pese a haberse realizado en una simulación.

La ejecución del sistema diseñado no se ha practicado en la cadena de bloques de Ethereum del mundo real debido a los costes económicos que ello conlleva (bajos pero existentes). Sin embargo, se aportan todas las instrucciones necesarias para su aplicación, aplicables tanto para un sistema Mac OS (donde se ha realizado la simulación) como en Raspbian, sistema operativo de la Raspberry Pi, modelo de hardware propuesto para la aplicación práctica del sistema.

Capítulo 8. Bibliografía

[1] Wikipedia, the free encyclopedia. “Smart Contract”. Wikipedia.org. https://en.wikipedia.org/wiki/Smart_contract. 26 septiembre, 2018.

[2] Wikipedia, the free encyclopedia. “Solidity”. Wikipedia.org. https://en.wikipedia.org/wiki/Solidity. 26 septiembre, 2018.

[3] Wikipedia, the free encyclopedia. “Ethereum”. Wikipedia.org. https://es.wikipedia.org/wiki/Ethereum. 26 septiembre, 2018.

[4] Bellare, Mihir; Rogaway, Phillip (2005). "Introduction". Introduction to Modern Cryptography.

[5] Kaspersky latinoamerica, ¿Qué es un Hash y cómo funciona?. latam.kaspersky.com.

https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/. 8 marzo, 2019

[6] PasswordGenerator.net, SHA1 Hash Generator. passwordgenerator.net. https://passwordsgenerator.net/sha1-hash-generator/

[7] León, A. (2018). Diseño de Sistemas Telemáticos, Tema 10, Servicios peer to peer. Universitat Politècnica de València, Valencia.

[8] Choban, U. (2017). Cryptocurrencies: A Brief Thematic Review. 1st ed. Canberra: School of Business and Economics, University of New South Wales, Canberra.

[9] Kim, K. (2018). Modified Merkle Patricia Trie - How Ethereum saves a state. medium.com. [Accessed 18 March 2019]. https://medium.com/codechain/modified-merkle-patricia-trie-how-ethereum-saves-a-state-e6d7555078dd

[10] Hancock, M. and Vaizey, E. (2016). Distributed Ledger Technology: beyond block chain. 1st ed. [ebook] United Kingdom: UK Government. Disponible en: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf [Acceso 15 Marzo 2019].

[11] Frankenfield, J. (2018). “Initial Coin Offering (ICO)”. investopedia.com. [Acceso 19 March 2019]. https://www.investopedia.com/terms/i/initial-coin-offering-ico.asp

[12] Iansiti, M.; Lakhani, K. (2017). The truth about Blockchain. Harvard Business Review.

[13] Lamport, L.; Shostak, R.; Pease, M. (1982). The Byzantine Generals Problem.

[14] ACM Transactions on Programming Languages and Systems. pp. 382-401. Disponible en: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Fbyz.pdf

[15] Catalini, Christian; Gans, Joshua S. (2016). "Some Simple Economics of the Blockchain".

[16] Tapscott, D; Tapscott, A. (2016). Here’s why Blockchains will change the world. Fortune

[17] ¿CÓMO MINAR CRIPTOMONEDAS?. (n.d.). www.criptonoticias.com. [online] Disponible en: https://www.criptonoticias.com/informacion/como-minar-criptomonedas/ [Acceso 14 Marzo 2019].

[18] Madeira, A. (2018). "Why is Ethereum different to Bitcoin?". CryptoCompare.

[19] Wikipedia, the free encyclopedia. “Ethereum Classic”. Wikipedia.org. https://en.wikipedia.org/wiki/Ethereum_Classic. [Acceso 14 Marzo 2019]

[20] The Economist (2015). The great chain of being sure about things.

[21] Antonopoulos, A. M. (2014). Mastering Bitcoin. Unlocking Digital Cryptocurrencies. Sebastopol, CA: O'Reilly Media.

[22] Preukschat, A. (2017). Los tipos de Blockchain: públicas, privadas e híbridas. [online] iecisa.com. Disponible en: http://www.iecisa.com/es/blog/Post/Los-tipos-de-Blockchain-publicas-privadas-e-hibridas-y-II/ [Acceso 15 Marzo 2019].

[23] Franco, P. (2014). Understanding Bitcoin: Cryptography, Engineering and Economics. p.9

[24] Cheng, E. (2017). Meet CryptoKitties, the $100,000 digital beanie babies epitomizing the cryptocurrency mania. CNBC

[25] ASCAP, PRS and SACEM join forces for blockchain copyright system. Music Business Worldwide. (2017)

[26] Chandra, P. (2018). Reimagining Democracy: what if votes were a cryptocurrency? democracywithoutborders.org

[27] Geron, T. (2012). Airbnb had $56 Million Impact on San Francisco: Study. Forbes.

[28] Brown, E. (2016). Who needs the Internet of Things? Linux.com. [Acceso 15 Marzo 2019]

[29] Wall, E. and Malm, G. (2016). Using Blockchain Technology and Smart Contracts to Create a Distributed Securities Depository. Department of Electrical and Information Technology, Lund University.

[30] Cryptocurrencies: A Brief Thematic Review Archived 25 December 2017 at the Wayback Machine. Social Science Research Network. [Acceso 15 Marzo 2019]

[31] Wood, G. (2018). "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER (EIP-150)". yellowpaper.io. [Acceso 18 Marzo 2019]

[32] ebuchman and zmanian (2019). Notes on the EVM. [online] github.com. Disponible en: https://github.com/CoinCulture/evm-tools/blob/master/analysis/guide.md [Acceso 6 Junio 2019].

[33] Asolo, B. (2018). Ethereum Virtual Machine Explained. Mycryptopedia.com [Acceso 18 Marzo 2019] https://www.mycryptopedia.com/ethereum-virtual-machine-explained/

[34] La moneda: el Ether (ETH). miethereum.com. [Acceso 19 Marzo 2019] https://miethereum.com/ether/

[35] "Ethereum vs. Bitcoin" (PDF). Economist. 2018.

[36] "This Is Your Company on Blockchain". Bloomberg Businessweek. (2016) [Acceso 19 Marzo 2019]

[37] “Settlement using blockchain to Automate Foreign Exchange in a Regulated environment (SAFER).” UK Research and Innovation. (2016). https://gtr.ukri.org/projects?ref=720735

[38] ¿Qué son las claves personales de Ethereum?. luno.com. [Acceso 19 Marzo 2019]. https://www.luno.com/learn/es/article/what-are-ethereum-private-keys

[39] https://etherscan.io/address/0xb794f5ea0ba39494ce839613fffba74279579268

[40] Buterin, V. (2013). "Bootstrapping A Decentralized Autonomous Corporation: Part I". Bitcoin Magazine (News). [Acceso 19 Marzo 2019] https://bitcoinmagazine.com/articles/bootstrapping-a-decentralized-autonomous-corporation-part-i-1379644274/

[41] Chohan, U.W. (2017). "The Decentralized Autonomous Organization and Governance Issues". Regulation of Financial Institutions Journal: Social Science Research Network (SSRN).

[42] Norta, A. (2015). "Creation of Smart-Contracting Collaborations for Decentralized Autonomous Organizations." Perspectives in Business Informatics Research. Volume 229 of the series Lecture Notes in Business Information Processing pp 3-17

[43] Szabo, N. (1997). Formilizing and securing relationships on public networks. [online] ojphi.org. Disponible en: https://ojphi.org/ojs/index.php/fm/article/view/548/469 [Acceso 19 Marzo 2019].

[44] blockgeeks.com. (2017). "Smart Contracts: The Blockchain Technology That Will Replace Lawyers". [online] Disponible en: https://blockgeeks.com/guides/smart-contracts/ [Acceso 19 Marzo 2019].

[45] Alsedo, Q. (2017). "El 'tecnohippie' que dijo 'no' a Puigdemont." ElMundo.es. [online] Disponible en: https://www.elmundo.es/espana/2017/12/12/5a3011fd46163f38558b4684.html [Acceso 19 Marzo. 2019].

[46] raspberrypi.org. (n.d.). Raspberry Pi 3 Model B+. [online] Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/ [Acceso 6 Junio 2019]

[47] raspberrypi.org. (n.d.). Raspberry Pi Universal Power Supply. [online] Disponible en: https://www.raspberrypi.org/products/raspberry-pi-universal-power-supply/ [Acceso 6 Junio 2019].

[48] raspberrypi.org. (n.d.). Camera Module V2. [online] Disponible en: https://www.raspberrypi.org/products/camera-module-v2/ [Acceso 6 Junio 2019].

[49] raspberrypi.org. (n.d.). Raspberry Pi Touch Display. [online] Disponible en: https://www.raspberrypi.org/products/raspberry-pi-touch-display/ [Acceso 6 Junio. 2019].

[50] mediamarkt.es. (n.d.). Switch - TPLink, 5 puertos Ethernet, Gigabit. [online] Disponible en: https://www.mediamarkt.es/es/product/_switch-tplink-5-puertos-ethernet-gigabit-1196975.html [Acceso 6 Junio 2019].

[51] amazon.es. (n.d.). NANOCABLE 10.20.0405 - Cable de Red Ethernet RJ45 Cat.6 UTP AWG24, 100% Cobre, Gris, latiguillo de 5mts. [online] Disponible en: https://www.amazon.es/NANOCABLE-10-20-0405-Cable-Ethernet-latiguillo/dp/B0091F5XQ6/ref=sr_1_4?__mk_es_ES=ÅMÅŽÕÑ&keywords=cable+ethernet+6+metros&qid=1559741820&s=gateway&sr=8-4 [Acceso 6 Junio 2019].

[52] amazon.es. (n.d.). Equip 221159 - Adaptador RJ45 apantallado Hembra-Hembra, Color Blanco. [online] Disponible en: https://www.amazon.es/Equip-221159-Adaptador-apantallado-hembra-

hembra/dp/B002UNFPIC/ref=sr_1_6?keywords=empalme+rj45&qid=1559781183&s=gateway&sr=8-6 [Acceso 6 Junio 2019].

[53] Howell, M. (n.d.). Homebrew. GitHub.

[54] Mitchel, J. (2017). Install Python 3.6, Virtualenv, & Django on Mac. [online] codIngforentrepreneurs. Disponible en: https://www.codingforentrepreneurs.com/blog/install-django-on-mac-or-linux [Acceso 26 Mayo 2019].

[55] Ganache. (2019). Truffle Blockchain Group.

[56] SafeMath. (2016). GitHub: OpenZeppelin.

[57] Rosebrock, A. (2018). An OpenCV barcode and QR code scanner with ZBar. [online] pyimagesearch.com. Disponible en: https://www.pyimagesearch.com/2018/05/21/an-opencv-barcode-and-qr-code-scanner-with-zbar/ [Acceso 6 Junio 2019].

[58] GitHub.com. (n.d.). Installing Python 3.6 on Raspbian. [online] Disponible en: https://gist.github.com/dschep/24aa61672a2092246eaca2824400d37f [Acceso 6 Junio 2019].

[59] raspberrypi.org. (n.d.). Installing Python packages - Raspberry Pi Documentation. [online] Disponible en: https://www.raspberrypi.org/documentation/linux/software/python.md [Acceso 6 Junio 2019].

[60] raspberrypi-aa.github.com. (n.d.). Python Virtual Environments (virtualenv). [online] Disponible en: http://raspberrypi-aa.github.io/session4/venv.html [Acceso 6 Junio 2019].

[61] Rosebrock, A. (2018). Install OpenCV 4 on your Raspberry Pi. [online] pyimagesearch.com. Disponible en: https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/ [Acceso 6 Junio 2019].

[62] infura.io. (n.d.). INFURA. [online] Disponible en: http://www.infura.io [Acceso 6 Junio 2019].

[63] Finlay, D., Huang, T., Pangilinan, F., Serrano, K., Moreau, J., kumavis and Jeria, C. (n.d.). MetaMask. [online] metamask.io. Disponible en: https://metamask.io [Acceso 6 Junio 2019].

[64] vodafone.es. (n.d.). Tarifas de fibra. [online] Disponible en: https://www.vodafone.es/c/particulares/es/productos-y-servicios/internet-y-fijo/ [Acceso 6 Junio 2019].

[65] MyEtherWallet.io. (n.d.). MyEtherWallet | MEW. [online] Disponible en: http://https://www.myetherwallet.com [Acceso 6 Junio 2019].

[66] Coinbase.com (n.d.) Coinbase. [online] Disponible en: https://www.coinbase.com [Acceso 6 Junio 2019]

[67] Team, U. (2017). How to buy ETH using Metamask + Coinbase. [online] Disponible en: https://blog.ujomusic.com/how-to-buy-eth-using-metamask-coinbase-ecffe9ede78e [Acceso 6 Junio 2019]

[68] Metamask.io (n.d) MetaMask. [online] Disponible en: https://metamask.io [Acceso 6 Junio 2019]

Capítulo 9. Anexo I. Códigos QR de las cuentas de Ganache

En este anexo, se proporcionan los códigos QR de todas las cuentas disponibles en Ganache. En cada figura, se muestra la clase de clave codificada (pública o privada), así como el propio código QR y el índice de la cuenta.

Dichos códigos son útiles para realizar la simulación del alquiler de una cápsula.

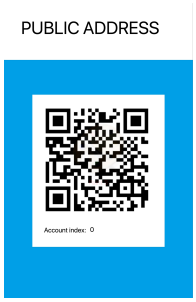

Figura 26. Código QR con la clave pública de la cuenta con índice 0.

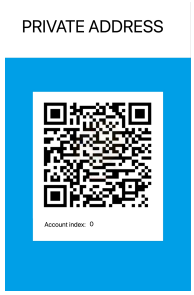

Figura 27. Código QR con la clave privada de la cuenta con índice 0.

Figura 28. Código QR con la clave pública de la cuenta con índice 1.

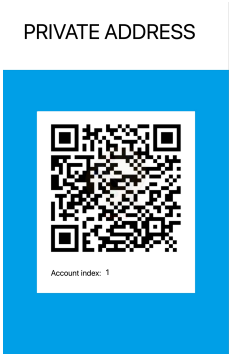

Figura 29. Código QR con la clave privada de la cuenta con índice 1.

Figura 30. Código QR con la clave pública de la cuenta con índice 2.

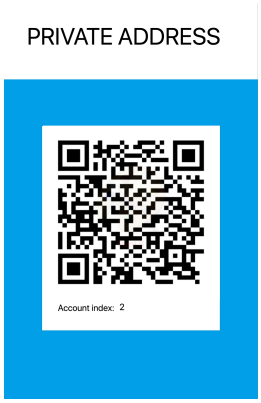

Figura 31. Código QR con la clave privada de la cuenta con índice 2.

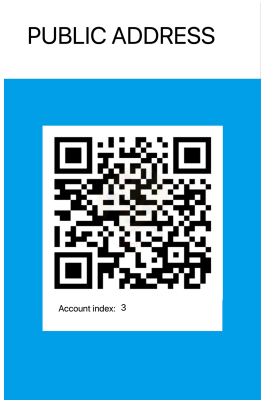

Figura 32. Código QR con la clave pública de la cuenta con índice 3.

Figura 33. Código QR con la clave privada de la cuenta con índice 3.

Figura 34. Código QR con la clave pública de la cuenta con índice 4.

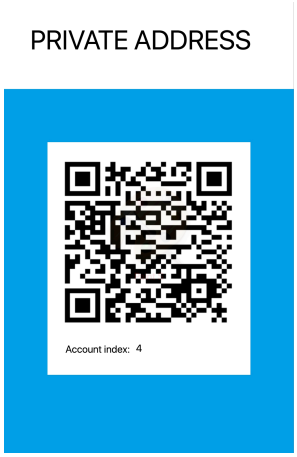

Figura 35. Código QR con la clave privada de la cuenta con índice 4.

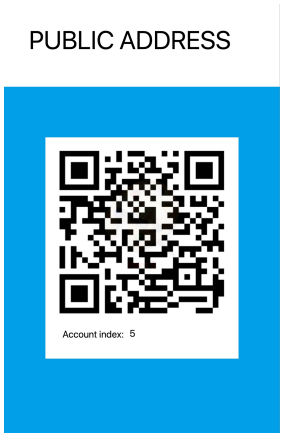

Figura 36. Código QR con la clave pública de la cuenta con índice 5.

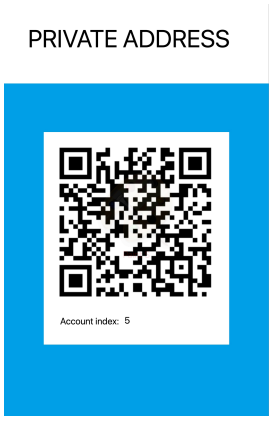

Figura 37. Código QR con la clave privada de la cuenta con índice 5.

Figura 38. Código QR con la clave pública de la cuenta con índice 6.

Figura 39. Código QR con la clave privada de la cuenta con índice 6.

Figura 40. Código QR con la clave pública de la cuenta con índice 7.

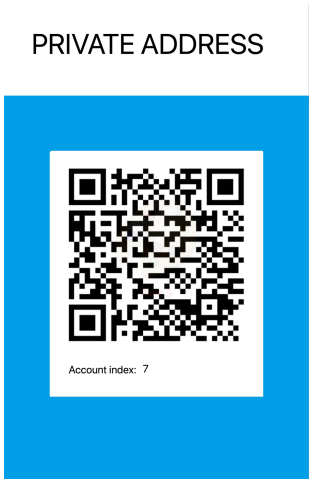

Figura 41. Código QR con la clave privada de la cuenta con índice 7.

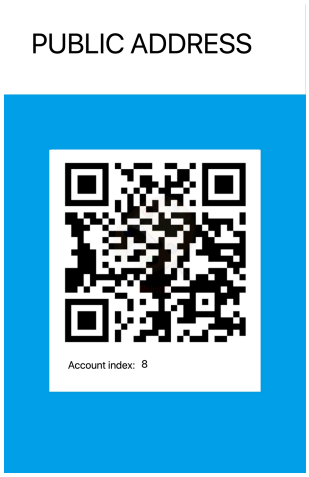

Figura 42. Código QR con la clave pública de la cuenta con índice 8.

Figura 43. Código QR con la clave privada de la cuenta con índice 8.

Figura 44. Código QR con la clave pública de la cuenta con índice 9.

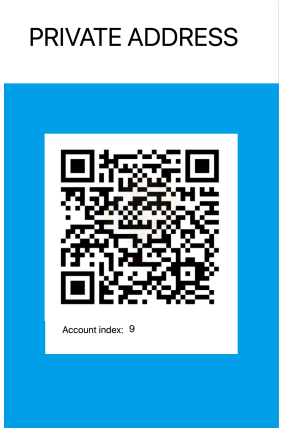

Figura 45. Código QR con la clave privada de la cuenta con índice 9.

Capítulo 10. Anexo II. Creación de una cuenta Ethereum

Para crear una cuenta Ethereum, podemos emplear varios servicios, como casas de cambio (exchanges como Coinbase o Binance), donde podremos adquirir criptomonedas, o crear nuestra propia cuenta mediante servicios como MyEthereumWallet.com o Metamask [65].

En primer lugar, explicaremos el procedimiento de creación de una cuenta Eth mediante MyEthereumWallet.

MyEthereumWallet es un servicio gratuito que ayuda al cliente a interactuar con la cadena de bloques de Ethereum. Dicho servicio ayuda a generar carteras Ethereum e interactuar con contratos inteligentes, entre otras cosas.

Para crear una cuenta Ethereum, en primer lugar, accedemos a la página web https://www.myetherwallet.com.

Una vez accedamos a la página, pulsamos en “Create a New Wallet”.

Una vez iniciemos el proceso, la propia página nos dará varios mensajes de advertencia acerca de lo que implica crear una cuenta Eth con sus servicios.

MyEthereumWallet no es un banco ni una casa de cambio, por lo que no almacena claves, fondos ni información personal. Por ello, es el usuario el encargado de guardar dicha información, y el servicio no será capaz de recuperar dicha información.

MyEthereumWallet también da algunas normas de seguridad, como no compartir la clave privada de la cuenta Eth (consejo que no tomaremos en cuenta en este proyecto), así como no confiar en recibir Ether gratis, comprobar los certificados de seguridad de las páginas web. Además, la página recomienda comprar un monedero de Ether físico.

A continuación, una vez leídas todas las recomendaciones de la página, el servicio proporciona tres formas de crear una contraseña para acceder a la cartera de Ethereum a crear.

- Empleando MEWconnect (aplicación móvil).
- A través de un archivo que almacene las claves.
- Por una frase nemotécnica.

En este caso, emplearemos la clave nemotécnica. Se nos proporcionan las siguientes palabras:

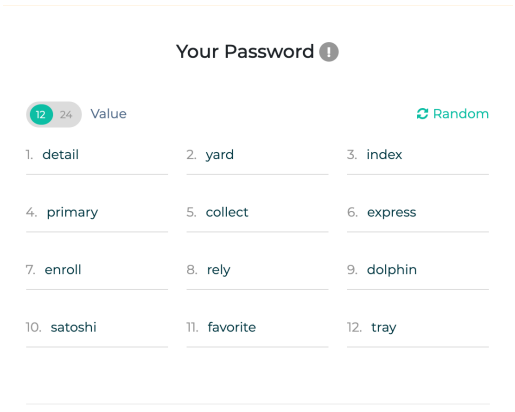

Figura 46: palabras que forman la clave nemotécnica de la cuenta ejemplo a crear.

Las palabras proporcionadas deben ser mantenidas en privado. Para este trabajo, dado que el objetivo principal es explicativo y no se introducirá ninguna cantidad de Ether en la misma, se publican las 12 palabras de la clave.

A continuación, la página nos pide introducir algunas de las palabras que nos ha proporcionado, como método de verificación.

Una vez introducidas, la cartera se ha creado. Para acceder a dicha cartera, la página web nos da cuatro opciones:

- Emplear MEWconnect (aplicación móvil).
- Emplear un dispositivo hardware como monedero físico.
- Emplear MetaMask.
- Emplear el software con el que nos hemos verificado. En este caso, elegimos la clave nemotécnica.

Introducimos todas las palabras de la contraseña, y podremos interactuar con nuestra cuenta creada, para realizar transacciones (una vez aceptemos los términos de uso).

Una vez hemos accedido, podemos realizar transacciones, cambiar Ether por Bitcoin, publicar contratos, etc. Dicho procedimiento no nos permite conocer nuestra clave privada.

La clave pública de la cuenta creada es la siguiente.

Figura 47. Clave pública de la cuenta ejemplo creada.

Otra solución mucho más sencilla sería empleando el plugin Metamask, disponible en navegadores como Chrome, Mozilla o Opera. Para ello, abrimos el plugin, y pulsamos en el siguiente símbolo.

Figura X. Botón para cambiar, crear o añadir una cuenta en Metamask.

A continuación, pulsamos en “Create Account”. Con ello, se crea automáticamente una cuenta Eth, con la que podremos interactuar a través de Metamask.

Ejecutando dicho proceso, tenemos como cuenta creada ejemplo la clave pública 0x80b050f60105133adb5dc02be5cc84fa940a5381.

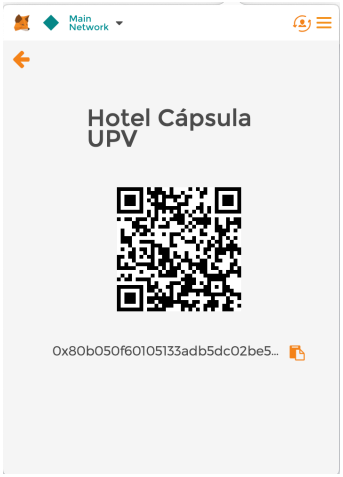

Figura 48. Clave pública de la cuenta ejemplo creada.

Por último, vamos a explicar cómo conseguir una cuenta Ethereum mediante una casa de cambio, empleando como casa de cambio Coinbase [66].

Para ello, únicamente debemos de registrarnos en la casa de cambio. Nos pedirá nuestro nombre, una dirección de correo electrónico y una contraseña. La plataforma mandará un correo de confirmación a la dirección de correo proporcionada automáticamente.

A continuación, la casa de cambio nos pedirá un número de teléfono. Una vez introducido, nos enviará un SMS con un código numérico, que debemos de introducir en Coinbase.

Una vez completado este paso, ya tenemos una cuenta en la casa de cambio operativa. Dicha casa de cambio nos proporciona varias cuentas de criptomonedas, entre ellas una de Ether.

Figura 49. Clave pública de la cuenta ejemplo creada.

Debe tenerse en cuenta que dicha cuenta no será operativa hasta que pasemos un control de identidad en la plataforma, introduciendo algunos datos personales.

Capítulo 11. Anexo III. Obtención de Ether

Para obtener Ether, tenemos tres opciones:

- Minar bloques. Esta opción requiere una alta cantidad de recursos.
- Recibir una transferencia de Ether en nuestra cartera.
- Comprar Ether en una casa de cambio.

Para esta última opción, podemos utilizar la casa de cambio Coinbase, entre otras.

Coinbase es una casa de cambio que permite el intercambio de criptomonedas. Como elemento diferenciador, la casa de cambio permite adquirir criptomonedas con tarjeta de crédito.

Para poder comprar criptomonedas en Coinbase, nos pedirá en primer lugar una verificación de identidad, en la que se nos pedirán datos como el nombre y apellidos, fecha de nacimiento, dirección postal, ocupación o propósito de la adquisición.

Desde Metamask también podemos comprar Ethereum a partir de Coinbase. Para ello:

- Entramos en el plugin de Metamask.
- Seleccionamos la red principal de Ethereum (Ethereum Main Net).
- Pulsamos en “Buy”.
- Pulsamos en “Coinbase” y “Continue to Coinbase”.
- En este momento, se abre la página web de Coinbase. Debemos tener una cuenta creada en la casa de cambio, con la identidad verificada.
- Introducimos la cantidad (en USD) que queremos comprar, y nuestra dirección de correo electrónico.
- Coinbase nos mandará vía email un código de seguridad numérico. Lo introducimos.
- Introducimos nuestro número de teléfono. Coinbase mandará un código numérico de seguridad vía SMS.
- Introducimos dicho código.
- Introducimos el método de pago que preferimos.
    - Transferencia bancaria. Tarda en hacerse efectiva la operación entre 4 y 5 días.
    - Tarjeta de débito o crédito. Operación inmediata.
- Confirmamos la compra.

Tras este proceso, podemos ver que en nuestra cuenta de Metamask figura la cantidad de Ether que hemos adquirido [67].

Capítulo 12. Anexo IV. Código del contrato inteligente.

```
//Código diseñado por Diego Sales Bellés, para su trabajo de fin de grado.
//Este código muestra el contrato inteligente para un hotel cápsula.
//Versión del compilador de Solidity.
pragma solidity ^0.5.1;
//Importamos la librería SafeMath.sol, diseñada para realizar operaciones matemáticas
//sin problemas de desbordamiento de bits.
import "./SafeMath.sol";
//Inicio del contrato.
contract renting  {
//Empleamos la librería para uints de 256 bits.
using SafeMath for uint;
//Establecemos valores a algunas variables prefijadas antes de la ejecución del programa.
//Las cantidades monetarias se expresan en Weis, para así evitar trabajar con floats
//hasta el paso final del proceso.
uint pricePerMinute = 10000000000000000; //cantidad en Wei, equivale a 0,01 ETH
uint serviceFee = 10000000000000000; //Por usar el servicio pongo una tasa de 0,01 ETH, expresado en Wei
uint penaltyFee = 1000000000000000000; //Una multa por exceso de servicio de 1 ETH, expresado en Wei
bool freePlace = true; //Esta variable nos marcará el estado del apartamento, libre u ocupado.
uint entryTime = uint(0); //Esta variable marcará el momento de inicio del alquiler. Inicializamos a 0.
uint exitTime = uint(0); //Esta variable marcará el momento de finalización del alquiler. Inicializamos a 0.
uint totalTime = uint(0); //Esta variable marcará la cantidad de minutos que ha durado el proceso de alquiler.
uint maxStay = uint(360); //Fijamos un periodo de 360 minutos, equivalente a 6 horas, como máximo de estancia sin penalización.
uint paymentPerMinutes = uint(0); //Esta variable nos ayudará a calcular la cantidad a pagar únicamente por los minutos de estancia.
uint serviceFeePayment = uint(0); //Esta variable nos ayudará a calcular la cantidad total a pagar.
//-----------------------------------------------------------------------------------------------------------
//A continuación, se muestran las funciones a ejecutar por la front-end app.
//Todas las funciones son públicas, por lo que cualquier usuario podrá ejecutarlas.
```

```
//-------------------------------------------------------------------------------------------------------------
//Esta función, una vez llamada, devuelve el valor de pago por minuto del alquiler.
function showPrice() public returns (uint) {
   return pricePerMinute;
}
//Esta función, una vez llamada, devuelve el valor de la penalización.
function showPenalty() public returns (uint) {
   return penaltyFee;
}
//Esta función, una vez llamada, devuelve el valor de la tasa de servicio.
function showServiceFee() public returns (uint) {
   return serviceFee;
}
//Esta función, una vez llamada, devuelve el valor de estancia máxima permitida
//sin penalización en el apartamento.
function showMaxStay() public returns (uint) {
   return maxStay;
}
//Esta función, una vez llamada, marca el momento de inicio del alquiler.
//Devuelve dicho momento.
//Se emplea para marcar el inicio del alquiler en la front-end app.
function initTime() public returns (uint) {
   entryTime = now; //El valor es el número de segundos transcurridos entre la ejecución y el momento inicial del año 1970
   return entryTime;
}
//Esta función, una vez llamada, marca el momento de salida del alquiler.
//Devuelve dicho momento. Se emplea para marcar el final del alquiler en la front-end app.
function endTime() public returns (uint) {
   exitTime = now; //El valor es el número de segundos transcurridos entre la ejecución y el momento inicial del año 1970
   return exitTime;
}
//Esta función, una vez llamada, calcula, a partir de los valores de entrada,
```

```
//la diferencia entre dichos valores, y los divide entre 60. Devuelve dicho valor.
//Esta función es útil para calcular el tiempo de estancia total.
function showTotalTime(uint _entryTime, uint _exitTime) public returns (uint) {
    totalTime = _exitTime;
    totalTime = totalTime.sub(_entryTime);
    totalTime = totalTime.div(60);
    return totalTime;
}
//-------------------------------------------------------------------------------------------------------------
//A continuación, se muestran las funciones empleadas para marcar el inicio y final del alquiler.
//-------------------------------------------------------------------------------------------------------------
//Esta función es llamada en el inicio del alquiler. Marca la cápsula como ocupada.
//Para poder ejecutarse, la cápsula debe estar marcada como libre.
function startRenting() public {
   require(freePlace == true);
   freePlace = false;
}
//Esta función es llamada al final del alquiler. Marca la cápsula como libre. Para poder ejecutarse,
la cápsula debe estar marcada como ocupada.
function finishRenting() public {
    require(freePlace == false);
    freePlace = true;
}
//-------------------------------------------------------------------------------------------------------------
//A continuación, se muestran las funciones empleadas para calcular el pago final.
//-------------------------------------------------------------------------------------------------------------
//Esta función multiplica los valores de entrada, y devuelve el resultado.
//Será empleada para calcular la cantidad a pagar por el concepto de estancia.
function ppM(uint _totalTime, uint _pricePerMinute) public returns (uint) {
    paymentPerMinutes = _totalTime;
    paymentPerMinutes = uint(paymentPerMinutes.mul(_pricePerMinute));
    return paymentPerMinutes;
}
```

```
//Esta función suma los valores de entrada, y devuelve el resultado. Será empleada para calcular la cantidad a pagar por el concepto de estancia más las tasas.
function sF(uint _payment, uint _serviceFeePayment) public returns (uint) {
    serviceFeePayment = uint(_serviceFeePayment.add(_payment));
    return serviceFeePayment;
}
//Esta función suma los valores de entrada, y devuelve el resultado. Será empleada para calcular la cantidad total a pagar en el caso de que haya una penalización.
function pen(uint _payment, uint _penaltyFee) public returns (uint) {
    serviceFeePayment = _payment;
    serviceFeePayment = serviceFeePayment.add(_penaltyFee);
    return serviceFeePayment;

}
}
```

Capítulo 13. Anexo V. Código de la biblioteca SafeMath

```
pragma solidity ^0.5.1;
/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }
  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }
  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
```

```
  }
  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
library SafeMath32 {
  function mul(uint32 a, uint32 b) internal pure returns (uint32) {
    if (a == 0) {
      return 0;
    }
    uint32 c = a * b;
    assert(c / a == b);
    return c;
  }
  function div(uint32 a, uint32 b) internal pure returns (uint32) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint32 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }
  function sub(uint32 a, uint32 b) internal pure returns (uint32) {
    assert(b <= a);
    return a - b;
  }
  function add(uint32 a, uint32 b) internal pure returns (uint32) {
    uint32 c = a + b;
    assert(c >= a);
    return c;
```

```
  }
}
/**
 * @title SafeMath16
 * @dev SafeMath library implemented for uint16
 */
library SafeMath16 {
  function mul(uint16 a, uint16 b) internal pure returns (uint16) {
    if (a == 0) {
      return 0;
    }
    uint16 c = a * b;
    assert(c / a == b);
    return c;
  }
  function div(uint16 a, uint16 b) internal pure returns (uint16) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint16 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }
  function sub(uint16 a, uint16 b) internal pure returns (uint16) {
    assert(b <= a);
    return a - b;
  }
  function add(uint16 a, uint16 b) internal pure returns (uint16) {
    uint16 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

Capítulo 14. Anexo VI. Código de la interface de usuario.

```
#Código diseñado por Diego Sales Bellés, para su trabajo de fin de grado.
#Este código muestra la interface a través de la cual el usuario interactuará con el smart contract
#En primer lugar, se importan varias librerias de Python
import json #Librería JavaScript Object Notation
import web3 #Librería Web3.py
import time #Librería time
from web3 import Web3 #Importamos la librería web3 al proyecto
from datetime import datetime #Importamos la librería datetime
import os #Librería os
import csv #Librería csv
from imutils.video import VideoStream # Librería imutils.video
from pyzbar import pyzbar #Librería pyzbar
import argparse #Librería argparse
import imutils #Librería imutils
import cv2 #Librería csv2
import sys #Librería sys
#A continuación, conectamos el programa al nodo de la plataforma Ethereum.
#En nuestro caso, lo conectamos a Ganache, nuestro simulador.
#En el caso de que conectemos la interface al contrato subido a la red principal, debemos cambiar dicha url.
# infura_url = “https://mainnet.infura.io/v3/c1c040269cc44901950d8553f8413a1e”
# web3 = Web3(Web3.HTTPProvider(infura_url))
ganache_url = "http://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(ganache_url))
#A continuación, establecemos las claves de las cuentas que vamos a utilizar
#en nuestra simulación.
addressReceiver = "0xd790A6394c9E1D64f4Cd76f99a94F6FfFa85F4e6" #Clave pública del propietario (Ganache), index 4.
web3.eth.defaultAccount = addressReceiver #Ponemos a la cuenta receptora como cuenta por defecto. Dicha cuenta se hará cargo de los pagos de gas.
#A continuación, debemos conectar nuestro programa de Python al contrato inteligente.
#Para ello, copiamos y pegamos los valores de la abi y el bytecode del contrato inteligente ya subido a la plataforma. Dicho valor se puede encontrar en el Remix de Solidity.
```

abi =
‘[{"constant":false,"inputs":[],"name":"showServiceFee","outputs":[{"name":"","type":"uint256
"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs"
:[],"name":"initTime","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutabilit
y":"nonpayable","type":"function"},{"constant":false,"inputs":[],"name":"showMaxStay","outp
uts":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"nonpayable","type":"fun
ction"},{"constant":false,"inputs":[{"name":"_entryTime","type":"uint256"},{"name":"_exitTi
me","type":"uint256"}],"name":"showTotalTime","outputs":[{"name":"","type":"uint256"}],"pa
yable":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs":[],"na
me":"endTime","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"no
npayable","type":"function"},{"constant":false,"inputs":[],"name":"finishRenting","outputs":[],"
payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs":[{"
name":"_payment","type":"uint256"},{"name":"_penaltyFee","type":"uint256"}],"name":"pen",
"outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"nonpayable","type"
:"function"},{"constant":false,"inputs":[{"name":"_payment","type":"uint256"},{"name":"_serv
iceFeePayment","type":"uint256"}],"name":"sF","outputs":[{"name":"","type":"uint256"}],"pay
able":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs":[{"nam
e":"_totalTime","type":"uint256"},{"name":"_pricePerMinute","type":"uint256"}],"name":"pp
M","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"nonpayable","t
ype":"function"},{"constant":false,"inputs":[],"name":"showPenalty","outputs":[{"name":"","ty
pe":"uint256"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":
false,"inputs":[],"name":"showPrice","outputs":[{"name":"","type":"uint256"}],"payable":false,
"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs":[],"name":"startRe
nting","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}]’

bytecode =
‘6080604052662386f26fc10000600055662386f26fc10000600155670de0b6b3a7640000600255
6001600360006101000a81548160ff0219169083151502179055506000600455600060055560006
0065561016860075560006008556000600955348015610 06c57600080fd5b5061059c8061007
c6000396000f3fe6080604052600436106100b4576000357c0100000000000000000000000000000
0000000000000000000000000000000900480630401e869146100b95780630411518 7146100e457
80631592b75e1461010f5780632b3e40ff1461013a5780633197cbb6146101935780633320d95c
146101be5780633da72eb2146101d5578063428d8c3d1461022e5780634962463c14610287578
0634ccefcaf146102e05780638e0e0abb1461030b5780638eca00ba14610336575b600080fd5b34
80156100c557600080fd5b506100ce61034d565b604051808281526020019150506040518091
390f35b3480156100f057600080fd5b506100f9610357565b604051808281526020019150506040
5180910390f35b34801561011b57600080fd5b506101246103685 65b604051808281526020019
150506040518091039 0f35b34801561014657600080fd5b5061017d600480360360408110156
1015d57600080fd5b810190808035906020019092919080359060200190929190505050610372
565b604051808281526020019150506040518091039 0f35b34801561019f57600080fd5b506101
a86103bd565b604051808281526020019150506040518091039 0f35b3480156101ca57600080fd
5b506101d36103ce565b005b3480156101e157600080fd5b506102186004803603604081101 56
101f857600080fd5b810190808035906020019092919080359060200190929190505050610 40d
565b604051808281526020019150506040518091039 0f35b34801561023a57600080fd5b506102
7160048036036040811015610251 57600080fd5b810190808035906020019092919080359060 2
00190929190505050610 43c565b604051808281526020019150506040518091039 0f35b348015
61029357600080fd5b506102ca600480360360408110156102aa57600080fd5b81019080803590
602001909291908035906020019092919050505061046 2565b604051808281526020019150506
0405180910390f35b3480156102ec57600080fd5b506102f5610491565b604051808281526020 0
191505060405180910390f35b34801561031757600080fd5b5061032061049b565b6040518082
64

```
8152602001915050604051809103905f35b34801561034257600080fd5b5061034b6104a4565b00
5b600060015490509056 5b600042600481905550600454905090565b600060075490509056 5b6
000816006819055506103908360065461 04e390919063ffffffff16565b600681905550610 3ac603
c6006546104fc90919063ffffffff16565b600681905550600654905092915050565b60004260058
190555060055490509056 5b600015156003600090549061010 00a900460ff1615151415156103f
057600080fd5b600160036000610100 0a81548160ff021916908315150217905550565b6000826
009819055506104 2b826009546105179091906 3ffffffff16565b600981905550600954905092915
0505056 5b600061045183836105179091906 3ffffffff16565b600981905550600954905092915050
0565b600082600881905550610480826008546105359091906 3ffffffff16565b60088190555060
0854905092915050565b600060025490509056 5b600080549050905 65b6001151560036000905
4906101000a900460ff1615151415156104c657600080fd5b600060036000610100 0a81548160ff
021916908315150217905550565b600082821115151561 04f157fe5b818303905092915050565b
600080828481151561050a57fe5b049050809150509291505056 5b600080828401905083811101
5151561052b57fe5b809150509291505056 5b600080831415610548576000905061056a565b60
008284029050828481281151561055b57fe5b0414151561056557fe5b809150505b9291505056fe
a165627a7a72305820cdd4672334dc290c12f9a902bd9dd0115389890c463312461b36ae7b05e4
e2310029'
```

```
#A continuación, instanciamos al contrato.
renting = web3.eth.contract(abi=abi, bytecode=bytecode)
#Ahora, creamos el contrato en nuestra simulación.
#Para ello, se realiza una transacción del contrato.
#Dicha transacción debe ser respondida, como confirmación del éxito.
tx_hash = renting.constructor().transact()
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
#Ahora, debemos de instanciar de nuevo el contrato, una vez ha sido subido a la red.
contract = web3.eth.contract(
    address=tx_receipt.contractAddress, #Dirección del contrato.
    abi=abi
)
#-----------------------------------------------------------------------------------------------------------
#Una vez el contrato ha sido ya subido correctamente a la red,
#podemos empezar a trabajar con él.
#-----------------------------------------------------------------------------------------------------------
#En primer lugar, llamamos a las variables fijadas en el contrato inteligente.
precioPorMinuto = contract.functions.showPrice().call()
precioPorMinutoScreen = precioPorMinuto/1000000000000000000 #Convertimos a 1 Ether para
mostrarlo en pantalla.
penalty = contract.functions.showPenalty().call()
```

```
penaltyScreen = penalty/1000000000000000000 #1Convertimos a 10 Ether para mostrarlo en pantalla.

tasaServicio = contract.functions.showServiceFee().call()

tasaServicioScreen = tasaServicio/10000000000000000 #Convertimos a 0,01 Ether para mostrarlo en pantalla.

estanciaMaxima = contract.functions.showMaxStay().call()

#A continuación, se muestra la interfaz de la pantalla a través de la cual el usuario interactuará.

print("Bienvenido a nuestro hotel cápsula.")

print("El precio por minuto de alquilar una cápsula es de: " +str(precioPorMinutoScreen)+ " Ether.")

print("En caso de superar su estancia " +str(estanciaMaxima)+ " minutos, se le cobrará una penalización de: " +str(penaltyScreen)+ "Ether.")

print("La tasa por emplear el servicio es de :" +str(tasaServicioScreen)+ " Ether.")

# Lectura de claves

print("Para comenzar con el alquiler, muestre a la cámara el código QR de su clave pública en Ethereum.")

print("Una vez el programa escanee el código (letras rojas por pantalla), pulse 'q'.")

# Escaneo de la clave pública del cliente

os.system("python3 barcode_scanner_image.py") #Dicho programa se describirá a continuación.

addressSender = lectura(44) # El último carácter a tomar es el número 44.

#Se da la orden de iniciar el alquiler.

#En primer lugar, se marca temporalmente el momento de inicio (segundos).

momentoEntrada = contract.functions.initTime().call()

#En segundo lugar, se convierte dicho valor a la hora local.

momentoEntradaScreen = time.localtime(momentoEntrada)

#Se imprime la hora local de inicio.

print("Momento entrada: " +time.strftime("%Y-%m-%d %H:%M:%S", momentoEntradaScreen))

tx_hash = contract.functions.startRenting().transact()

tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)

print("Puerta abierta.")

#Una vez se llega aquí, el alquiler está iniciado.



#-----------------------------------------------------------------------------------------------------------
```

```
#PROCESO DE FINALIZACIÓN DE ALQUILER
#-----------------------------------------------------------------------------------------------------------
print("Cuando quiera finalizar el servicio, por favor, pulse intro.")
tecla = input()
# Escaneo de la clave privada del cliente
print("Por favor, firme digitalmente la operación con su clave Ethereum privada.")
print("Una vez el programa escanee el código (letras rojas por pantalla), pulse 'q'.")
#Escaneo de la clave privada del cliente
os.system("python3 barcode_scanner_image.py")
addressPrivateSender = lectura(66) #El último carácter a tomar es el 66
remove(“barcodes.csv”) #De esta forma se evita mantener el archivo con claves valiosas para el usuario
#Se da la orden de finalización del alquiler.
#En primer lugar, se marca el momento de salida (segundos).
momentoSalida = contract.functions.endTime().call()
#En segundo lugar, se convierte dicho valor a la hora local.
momentoSalidaScreen = time.localtime(momentoSalida)
#Se imprime la hora local de finalización.
print("Momento salida: " +time.strftime("%Y-%m-%d %H:%M:%S", momentoSalidaScreen))
#Se calcula la cantidad de minutos que ha durado el alquiler.
tiempoTotal = contract.functions.showTotalTime(momentoEntrada, momentoSalida).call()
print("Estancia total: " +str(tiempoTotal)+ " minutos.")
tx_hash = contract.functions.finishRenting().transact()
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
#Una vez se llega aquí el alquiler está finalizado.
#-----------------------------------------------------------------------------------------------------------
#PROCESO DE CÁLCULO DE CANTIDAD A PAGAR
#-----------------------------------------------------------------------------------------------------------
#En primer lugar, se calcula la cantidad a pagar únicamente por la estancia.
pago1 = contract.functions.ppM(tiempoTotal, precioPorMinuto).call()
#En segundo lugar, se calcula la cantidad a pagar añadiendo el efecto de la tasa por servicio.
pago2 = contract.functions.sF(pago1, tasaServicio).call()
#A continuación, se evalúa si debe aplicarse una penalización por exceso de tiempo o no.
if(tiempoTotal>estanciaMaxima):
```

```
    pago3 = contract.functions.pen(pago2, penalty).call() #Se aplica penalización.
    pago3 = pago3/1000000000000000000
else:
    pago3 = pago2/1000000000000000000 #No se aplica penalización.
#Se muestra por pantalla la cantidad a pagar.
print("La cantidad a pagar es de: " +str(pago3)+ " Ether.")
#-----------------------------------------------------------------------------------------------------------------
#PROCESO DE PAGO AUTOMÁTICO
#-----------------------------------------------------------------------------------------------------------------
#En primer lugar, se debe cubrir la posibilidad de que surja una excepción por falta de fondos.
try:
    #Obtención del nonce de la transacción a realizar.
    nonce = web3.eth.getTransactionCount(addressSender)
    #Se construye la transacción.
    tx = {
        'nonce': nonce, #Nonce.
        'to': addressReceiver, #Destinatario.
        'value': web3.toWei(pago3, 'ether'), #Cantidad.
        'gas': 2000000, #Límite de gas de la operación.
        'gasPrice': web3.toWei('50', 'gwei') #Precio que proponemos de gas.
    }
    #Se firma la transacción.
    signed_tx = web3.eth.account.signTransaction(tx, addressPrivateSender)
    #Se envía la transacción.
    tx_hash = web3.eth.sendRawTransaction(signed_tx.rawTransaction)
    #Se imprime el hash de la transacción.
    print(web3.toHex(tx_hash))
    #Se confirma el éxito de la operación.
    print("Pago realizado con éxito.")
except:
    pass
    #Si saliera este mensaje, es que ha surgido un error durante el pago, como la falta de fondos.
    print("Error al ejecutar el pago. No dispone de suficientes fondos.")
```

Capítulo 15. Anexo VII. Código del lector de códigos QR.

```
# Importar las bibliotecas necesarias
from imutils.video import VideoStream
from pyzbar import pyzbar
import argparse
import datetime
import imutils
import time
import cv2
import csv
import os
# construir el analizador de argumentos y analizar los argumentos
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", type=str, default="barcodes.csv",
	help="path to output CSV file containing barcodes")
args = vars(ap.parse_args())
# Inicialice el flujo de video y permita que el sensor de la cámara se encienda.
vs = VideoStream(src=0).start() #En el caso de emplear la cámara del ordenador
# En la siguiente línea comentada, se da el comando para activar la cámara de la raspberry.
# En el montaje real, se debería comentar la línea previa y descomentar la próxima.
#vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
# Abra el archivo CSV de salida para escribir e inicialice el conjunto de códigos de barras encontrados
csv = open(args["output"], "w")
found = set()
# bucle sobre los instantes del flujo de video
while True:
	# toma el instante de video y cambia el tamaño máximo de 400 píxeles
	frame = vs.read()
	frame = imutils.resize(frame, width=400)
	# encuentra los códigos y decodificalos
	barcodes = pyzbar.decode(frame)
```

```
    # bucle sobre los códigos encontrados
        for barcode in barcodes:
            # Extraer la ubicación del cuadro delimitador del código de barras y dibujar
            # el cuadro delimitador que rodea el código de barras en la imagen.
            (x, y, w, h) = barcode.rect
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
            # Los datos del código de barras son un objeto de bytes, así que si queremos dibujarlos en nuestra imagen de salida necesitamos convertirlo primero en una cadena
            barcodeData = barcode.data.decode("utf-8")
            barcodeType = barcode.type
            # dibujar los datos de código de barras y el tipo de código de barras en la imagen
            text = (barcodeData) # Editar escritura de datos
            cv2.putText(frame, text, (x, y - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
            # Si el texto del código de barras no está actualmente en nuestro archivo CSV, escriba la marca de tiempo + código de barras en el disco y actualizar el conjunto
            if barcodeData not in found:
                csv.write(barcodeData)
                csv.flush()
                found.add(barcodeData)
        # mostrar la salida
        cv2.imshow("Barcode Scanner", frame)
        key = cv2.waitKey(1) & 0xFF
    # si se presiona la tecla 'q', se sale del bucle
        if key == ord("q"):
            break
# Cierra el archivo CSV
csv.close()
cv2.destroyAllWindows()
vs.stop()
```