SPOC (Small Private Online Course) de DLT (Distributed Ledger Technology)/Blockchain

(ID2019/219)

Memoria de Resultados

Convocatoria de Innovación Docente – Curso 2019-2020

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

30 de Junio de 2019

I. CONTENIDO

II. FIGURAS

III. DATOS DEL PROYECTO

TÍTULO: SPOC (Small Private Online Course) de DLT(Distributed Ledger Technology)/Blockchain
REFERENCIA: ID2019/219
CUANTÍA DE LA SUVENCIÓN: 540 €

PROFESORA COORDINADORA: Sara Rodríguez González

ORGANISMO: UNIVERSIDAD DE SALAMANCA
CENTRO: FACULTAD DE CIENCIAS

INVESTIGADORES QUE FORMAN EL EQUIPO:
Roberto Casado Vara, Diego Javier Valdeolmillos Villaverde, Alberto Rivas Camacho Javier José Martín Limorti, Elena Hernández Nieves , Inés Xiomara Sitton Candanedo, Ricardo S. Alonso Rincón.

Departamento de Informática y Automática
Universidad de Salamanca - Facultad de Ciencias
Plaza de la Merced, s/n
37008 Salamanca
DURACIÓN: Octubre 2019 a julio 2020

IV. INTRODUCCIÓN

Este proyecto tiene por objetivo el diseño de un SPOC para el aprendizaje de nuevas tecnologías de registro distribuido (DLT) como Blockchain.

Los SPOC son una modalidad de formación online que se caracterizan por ser una evolución de los MOOC para ajustarlos a necesidades específicas de una entidad educativa. Mientras los MOOC ayudan a la actualización de conocimientos generales y de grandes grupos, los SPOC permiten a los estudiantes aprovechar el potencial de este tipo de cursos, pero dirigidos a una comunidad particular, desarrollando un perfil específico. En este caso, se introduce una especialización a blockchain, una tecnología revolucionaria en temas de seguridad informática que actualmente está teniendo un gran impacto en la red debido a la aparición de monedas virtuales y contratos inteligentes (Smart Contracts). Más específicamente, prepara a los estudiantes con perfil principalmente de ingeniería para programar en entornos novedosos como Ethereum, ampliando conceptos de seguridad y programación concurrente de otras asignaturas de grado relacionadas (como Sistemas-Operativos-II, por ejemplo).

El SPOC tiene por objetivos secundarios proporcionar a los estudiantes:

- Comprensión y un conocimiento práctico de conceptos fundamentales de blockchain.
- Habilidades para diseñar SmartContract.
- Métodos para desarrollar aplicaciones descentralizadas en blockchain.
- Información sobre los marcos de blockchain específicos en la industria.

V. CONCEPTOS CLAVE

V) 1. SPOC

En los últimos años se han ido afianzado dentro de la comunidad educativa los formatos de cursos en formato MOOC, o lo que es lo mismo Cursos On-Line Masivos en Abierto (sus siglas en inglés Masive Open On-Line Course).

Esta nueva modalidad de curso online ha permitido que personas de muy diverso ámbito (no solo estudiantes de educación secundaria o universitarios) adquirieran conocimientos básicos e incluso avanzados de temas necesarios para su empleabilidad.

Es una forma sencilla de introducirse en temarios hasta hace poco desconocidos para aprender y conocer en qué se estaban moviendo los diferentes sectores laborales. También, ha ayudado a profesionales a reciclarse y ponerse al día con nueva metodologías, técnicas y conocimientos.

Aparecen aquí diferentes conceptos clave que debemos distinguir:

- Un MOOC (Masive Open On-Line Course) es un curso de acceso gratuito con itinerario, tutorización, dinamización y tiempo limitado. La certificación de superación del curso es lo único que suele ser de pago, y suele contar con pruebas de conocimiento.
- Un SPOOC (Self-Paced Open On-Line Course), se basa en el encadenamiento de módulos formativos en itinerario, ajustado la disponibilidad del usuario. Es decir, es el usuario el que marca su ritmo de aprendizaje, ya que este formato no fija un tiempo límite de realización. En este caso, se pierde la posible tutorización y dinamización del curso, aunque mantiene su poder de evaluación.
- Un NOOC (Nano Open On-Line Course), se puede considerar la versión micro de los anteriores, son pequeñas píldoras que den lecciones muy concretas. Son minicursos a la carta que pueden ser libres o con itinerarios.

Estos tres tipos de curso son cursos en abierto, y en ocasiones, son muy estandarizados, tanto que a veces no se ajustan a las necesidades de todos. Para ello están los cursos Privados:

- Blended: Estos cursos son los que se les llama de formación mixta, donde la parte presencial se integra con la on-line y al revés. Es una opción muy acogida entre PYME's para potenciar la formación de sus trabajadores en una doble dirección de aprendizaje.
- SPOC: No confundir con los SPOOC, son los Small Private On-Line Course. Pequeños cursos privados que se pueden hacer a medida para grupos, comunidades o empresas y que son perfectos para compaginar trabajo y formación.
- COOC: Por último, los Corporative Open On-Line Course. Es un paso más allá de la formación corporativa. Suele desarrollarse a medida, para empresas grandes o con muchos empleados para que la formación sea generalizada en todos los niveles e incluso personalizada en cada nivel jerárquico.

En el caso que nos ocupa en este proyecto de innovación, nos hemos centrado en la elaboración de un SPOC, orientado a alumnos e investigadores que pretenden ampliar sus conocimientos de seguridad informática con la tecnología Blockchain.

V) 2. BLOCKCHAIN

Uno de los primeros documentos en los que se menciona el termino Blockchain es el documento escrito por Satoshi Nakamoto[1] en el año 2008: "Bitcoin P2P e-Cash" . En ese artículo se describe Blockchain como una pieza tecnológica creada con el único propósito de hacer realidad el proyecto Bitcoin y que hace uso de distintas técnicas que han surgido, a lo largo de los últimos 40 años en el campo de la informática. Muchas de las técnicas descritas en el paper de Satoshi Nakamoto se crearon con propósitos militares y durante mucho tiempo fueron propiedad exclusiva de algunos gobiernos, aunque a mediados de la década de 1990, la mayoría de estas herramientas y tecnologías ya eran accesibles para las universidades y la población civil.

[1] Nakamoto, S. (2019). *Bitcoin: A peer-to-peer electronic cash system*.

V) 2. 1. LA CRIPTOGRAFÍA

Como es bien conocido, la criptografía tuvo origen militar. Durante la Segunda Guerra Mundial, los estados en guerra experimentaron necesidades de comunicarse por radio a través de canales que el enemigo podía escuchar para trasmitir mensajes secretos que solo pudiesen ser descifrados por los aliados. Alan Turing, matemático británico, es considerado uno de los padres de la informática y también uno de los primeros investigadores en el campo de la criptografía.

La mayoría de las redes blockchain utilizan un tipo de criptografía inventada en 1985 por Neal Koblitz y Victor S. Miller, conocida como criptografía de curvas elípticas.

V) 2. 2. EL MOVIMIENTO CYBERPUNK

En los años 90, con las bases técnicas de la criptografía consolidándose, tuvo lugar un movimiento social que ayudaría a que Bitcoin se hiciese realidad, dotándolo de recursos económicos y humanos: el movimiento Cyberpunk. Este movimiento tuvo su origen formal en el manifiesto cripto-anarquitas de Timothy C. May, publicado tras la reunión de Cyberpunks en Sillicon Valley que tuvo lugar en el año 1992. En este manifiesto, el autor realizaba un llamamiento para emprender una revolución social y económica de carácter libertaria con el objetivo de reducir la intervención del estado a su mínima expresión. Debido a que la protección que brinda el Estado en la firma de acuerdos y la relación entre individuos podía ser ahora garantizada por la tecnología, la sociedad podía avanzar hacia una forma de gobierno más libertaria y eficiente.

El paper de Satoshi Nakamoto describe a Bitcoin como un tipo de dinero electrónico gobernado por unas reglas criptográficas que controlan su creación y gestión en lugar de confiar en autoridades centrales como los gobiernos. No es de extrañar que el movimiento Cyberpunk apoyase el proyecto de Bitcoin desde su inicio.

V) 2. 3. LA TEORÍA DE JUEGOS

La Teoría de Juegos es una disciplina a medio camino entre las matemáticas y la economía y que, además de jugar un papel muy importante en redes de Blockchain como la de Bitcoin, es aplicable a situaciones en las que agentes racionales toman decisiones en un escenario bien definido. Una red de Blockchain no es solo un conjunto de usuarios ejecutando un programa de ordenador que les permite crear y transferir dinero electrónico. Los usuarios que participan de la red tienen un motivo para hacerlo.

La Teoría de Juegos es una disciplina matemática que, entre otras cosas, permite ajustar los incentivos de los participantes en una red de Blockchain para comportarse de una determinada manera, contribuyendo positivamente a la red. El motivo por el que en Bitcoin existen usuarios que validan transacciones es porque existe una recompensa.

V) 2. 4. LAS CRIPTOMONEDAS Y SUS ANTECEDENTES

Bitcoin es una moneda digital que tiene su origen en el año 2008 y que utiliza la criptografía para garantizar aspectos de su funcionamiento como la puesta en circulación de la moneda o el mecanismo de gasto que garantiza que una moneda no pueda ser gastada dos veces. Lo que conocemos como Blockchain es en realidad un conjunto de mecanismos que, en las redes públicas, sirven para resolver dos grandes problemas:

1. El mecanismo para poner en circulación una nueva moneda.

2. Evitar que un usuario gaste dos veces la misma cantidad de dinero.

El primer problema pudo resolverse gracias a la prueba de trabajo, inventada por Adam Back en el proyecto HashCash. La prueba de trabajo consiste en un problema matemático de resolución compleja, pero de verificación sencilla, como por ejemplo cálculos de raíces cuadradas de números grandes o colisiones parciales de hashes. El segundo problema pudo resolverse gracias al trabajo de investigación de Nick Szabo sobre los registros de activos tolerantes a fallos bizantinos. Sumado a la cadena de Bloques, estas investigaciones permitieron que varios usuarios de una red de Blockchain fuesen capaces de conocer y verificar el último estado global de la cadena de bloques.

V) 2. 5. LOS TOKENS Y LOS SMART CONTRACTS

En el año 2014, el programador Vitalik Buterin propuso y lideró la creación de una nueva red de Blockchain conocida como Ethereum. Además de permitir la transacción con una moneda digital propia, el Ether, esta red permite a cualquier usuario crear programas de ordenador y desplegarlos en ella. Al desplegarlos, el programa de ordenador se instala en todos y cada uno de los ordenadores que forman parte de la red.

Del mismo modo que una transacción en Bitcoin se considera legítima si la mayoría de los mineros de Bitcoin la consideran válida, el resultado de la ejecución de un Smart Contract será el obtenido por la mayoría de los mineros de la red de Ethereum. Eso significa que los mineros de la red Ethereum no solo validan transacciones e instalan estas aplicaciones, sino que además deben realizar todas las ejecuciones de cada una de ellas y compararlas con el resultado obtenido por el resto.

Como se puede deducir, a pesar de que el nombre Smart Contract signifique contrato inteligente, el nombre es desafortunado, ya que no se trata de un contrato sino de un programa de ordenador. Aunque los Smart Contracts de Ethereum pudiesen servir para garantizar una transacción comercial entre dos partes –siempre que esa transacción sea pagada Ether– lo cierto es que en la práctica no sirven para sustituir contratos complejos como un contrato de trabajo.

Uno de los principales usos de los Smart Contracts es la creación de tokens. Un token es un Smart Contract que sirve para llevar la cuenta de un determinado activo en una lista. Se trata de un pequeño libro contable creado sobre un gran libro contable: la cadena de bloques.

Uno de los fenómenos más populares alrededor de lo tokens es el de las ICO u oferta inicial de monedas. Se trata de una práctica seguida principalmente por las empresas de nueva creación, a través de la cual emiten sus propias monedas digitales para representar los servicios que prestarán en el futuro, cuando la compañía aún no presta ningún servicio. La empresa utiliza los fondos obtenidos en criptomonedas por la preventa de estos tickets —tokens— para financiarse y, de este modo, consigue crear una comunidad alrededor de su producto. Tan solo en el año 2017 la financiación obtenida por las empresas que han decidido financiarse mediante esta técnica supera los 5000 millones de euros. Una cifra sorprendente que no para de crecer teniendo en cuenta que esta forma de financiación no existía en 2016.

V) 2. 6. BLOCKCHAIN Y EL FENÓMENO DE LOS CONSORCIOS

Las criptomonedas tienen un origen con componentes políticos y sociales. Una de las primeras personas en percatarse de que este hecho no ayudaba a su adopción por parte de las empresas fue Don Tapscott, un ejecutivo canadiense con una larga trayectoria en finanzas que propuso separar las criptomonedas de la tecnología que subyace: Blockchain. Aunque pueda parecer poco relevante, el hecho de separar ambos conceptos fue clave para que la banca comenzase a experimentar con Blockchain.

La banca fue el primer sector en experimentar con Blockchain. Al fin y al cabo, Blockchain es una tecnología para gestionar transacciones económicas y este es el núcleo del sector financiero.

Otras empresas como IBM vieron en Blockchain una forma no solo de gestionar transacciones económicas, sino de gestionar información de todo tipo de manera compartida. Esto aportaba varias ventajas, ya que hasta ese momento cada una de las empresas que colaboran entre sí gestionaban la información por separado. Si alguna de ellas cometía un error era necesario ponerse en contacto con el resto para conciliar la información y asegurarse de que todas ellas llevaban la cuenta de lo mismo y de la misma forma.

VI. TECNOLOGÍAS Y TÉCNICAS EMPLEADAS

Para el desarrollo del proyecto ha sido necesario la evaluación de las principales tecnologías y técnicas computacionales que permitieran llevarlo a cabo. Este apartado recoge un resumen de las principales, aportando una descripción de cada uno y el uso dado en el proyecto.

VI) 1. REDES BLOCKCHAIN

En esta sección vamos a hablar sobre las 2 principales redes Blockchain que están disponibles ahora mismo. Aunque nos centraremos en Ethereum, ya que Hyperledger queda fuera del alcance de este curso.

VI) 1. 1. IBM HYPERLEDGER FABRIC

El proyecto Hyperledger nace como un esfuerzo colaborativo de código abierto creado para avanzar en el desarrollo de las tecnologías Blockchain. Nace en el seno de una fundación que nació para cuidar del Software Abierto, la Linux Foundation, en el que se mueven otros

proyectos como Docker, Kubernetes node.js, etc. En Hyperledger contribuyen empresas de diverso tipo; tecnológicas como IBM, Intel, Fujitsu, etc.; financieras, como JP Morgan, BBVA, SWIFT, etc.

Al hablar de hyperledger (HL) no lo hacemos de una sola tecnología Blockchain, o, en general, DLT (Distributed Ledger Technology), sino de varias: Fabric, Sawtooth, Burrow, Airoa, etc. En lo que sigue vamos a profundizar en una de ellas, el HL Fabric, que es un middleware que suministra soporte Blockchain, privado y permisionado. Por encima de Fabric se puede instalar un acelerador de aplicaciones que se denomina HyperLedger Composer, que permite crear fácilmente aplicaciones para Fabric.

El núcleo básico de la arquitectura de Hyperledger Fabric es una ejecución en tres fases del proceso tradicional order-execute sobre el que van a fluir las transacciones.

> *Definición: HL Fabric es un sistema operativo distribuido permisionado de operaciones sobre blockchains que ejecuta aplicaciones distribuidas escritas en lenguajes de programación de propósito general (por ejemplo, Go, Java, Node.js). Realiza un seguimiento seguro de su historial de ejecución en una estructura de datos llamada LEDGER (libro mayor replicado) y no tiene criptomoneda incorporada.*

HL Fabric presenta una arquitectura de ejecución en tres fases: execute-ordervalidate en vez de la tradicional order-execute por las razones explicadas anteriormente.

En pocas palabras, una aplicación distribuida para Fabric consta de dos partes:

- Un contrato inteligente, llamado chaincode, que es un programa código que implementa la lógica de la aplicación y se ejecuta durante la fase de ejecución. El chaincode es la parte central de una aplicación distribuida en Fabric y se puede escribir por un desarrollador no confiable. Existen chaincodes especiales para la gestión del sistema Blockchain (system chaincodes).
- Una política (normativa o policy) de aprobación que se evalúa en la fase inicial de validación. Estas políticas, llamadas de endosado (endorsing), no pueden ser elegidas o modificadas por los desarrolladores de aplicaciones, por definición no confiables, aun siendo parte del sistema. Una política de endosado (endorsing policy) actúa como una biblioteca estática para la validación de transacciones en Fabric, las cuales solo pueden ser parametrizadas por el chaincode. Solo ciertos administradores designados pueden ejecutar funciones de administración del sistema y tener el derecho de modificar la política de endosado.

A continuación, se pone una imagen de ejemplo de un chaincode simple:

Welcome org.acme.mynetwork.cto

```
/**
 * Write your model definitions here
 */

namespace org.acme.mynetwork

// 1 Create Participant
participant Professor identified by professorId {
  o String professorId
  o String professorFirstName
  o String professorLastName
}

//2 Create Asset
asset assetGrade identified by studentID {
  --> Professor professor
  o String studentID
  o String grade
}

// Create Transaction
transaction changeGrade {
  --> assetGrade asset
  o String newGrade
}
```

Figura 1. Ejemplo de chaincode para verificar títulos universitarios

Los nodos en una red Fabric toman uno de tres roles:

- **Clientes**: envían propuestas de transacciones para su ejecución, orquestan la fase de ejecución y, finalmente, transmiten transacciones para ordenar (consenso).
- **Peers:** ejecutan propuestas de transacciones y validan las transacciones. Los peers también mantienen el ledger en blockchain, estructura de datos de solo anexos que registra todas las transacciones en la forma de una cadena de hash, así como el estado, una sucinta representación del último estado contable. No todos los peer ejecutan todas las propuestas de transacción, solo un subconjunto de ellas llamados endorsing peers (o, simplemente, endorsers), según se especifique por la política de endosado del chaincode al que la transacción pertenece. Sin embargo, todos los peer mantienen el libro mayor completo.

- **Ordering Service Nodes (OSN)** (o, simplemente, orderers): son los nodos que forman colectivamente el servicio de orden (consenso). En pocas palabras, el servicio establece el orden completo de todas las transacciones en la red o topología HL Fabric, donde cada transacción contiene actualizaciones de estado y dependencias calculadas durante la fase de ejecución, junto con la firma criptográfica de los pares patrocinadores (endorsers) que los computaron. Los nodos de consenso u orderers, ignoran completamente el estado de la aplicación y no participan en la ejecución ni en la validación de transacciones. Esta elección de diseño permite un consenso modular (pluggable) y simplifica el reemplazo de protocolos de consenso en Fabric.

Dado que es posible ejecutar un nodo físico con múltiples roles, Hyperledger-Fabric también puede ser operado como un sistema peer-to-peer tradicional de Blockchain, en el que cada nodo mantiene el estado e invoca, valida y ordena transacciones.

VI) 1. 2. ETHEREUM

Ethereum nace a finales del 2013, a manos de Vitalik Buterin, un activo programador de Bitcoin y Mastercoin que pretendía extender las capacidades de dichas cadenas de bloques. Su primera propuesta incluía contratos flexibles mediante el uso de un lenguaje de scripting que mejoraba notablemente las limitaciones del código específico de Mastercoin.

En diciembre del 2013 publicaría el White Paper denominado «A Next-Generation Smart Contract and Decentralized Application Platform» que establecería las premisas de Ethereum y que evolucionaría gracias al apoyo de varios miembros de la comunidad. Entre ellos destacaría Gavin Wood (cofundador, diseñador y CTO) que publicaría la especificación técnica y formal de Ethereum en el denominado Yellow Paper: «Ethereum: A Secure Decentralised Generalised Transaction Ledger».

Se formaría así, el 30 de Julio del 2015, el «Ordenador Mundial» con el minado del primer bloque de la cadena de bloques de Ethereum.

Figura 2. Bloque 0 de Ethereum

Ethereum puede definirse como una máquina de estados infinitos con dos funcionalidades principales: publicar un estado único e implementar una máquina virtual que modifica dicho estado. De esta forma, se genera un entorno de computación distribuida que ejecuta programas denominados smart contracts, que a su vez almacenan y acceden a datos sincronizados mediante una cadena de bloques.

Los principales componentes de Ethereum son los siguientes:

- **Red peer-to-peer**: los nodos de Ethereum escuchan en el puerto 30303TCP usando el protocolo ÐΞVp2p
- **Blockchain**: Cadena de bloques cuya moneda intrínseca es el Ether y que está destinada a almacenar transacciones y el estado de la red. Esta permite almacenar en forma de datos el código compilado de los smart contracts que los usuarios comparten con la red. En Ethereum, la Blockchain se implementa con la estructura de datos arborescente Merkle Patricia Tree, donde cada nodo hoja contiene el hash de los datos y cada nodo no-hoja (incluida la raíz) contiene un hash de los nodos hijos.

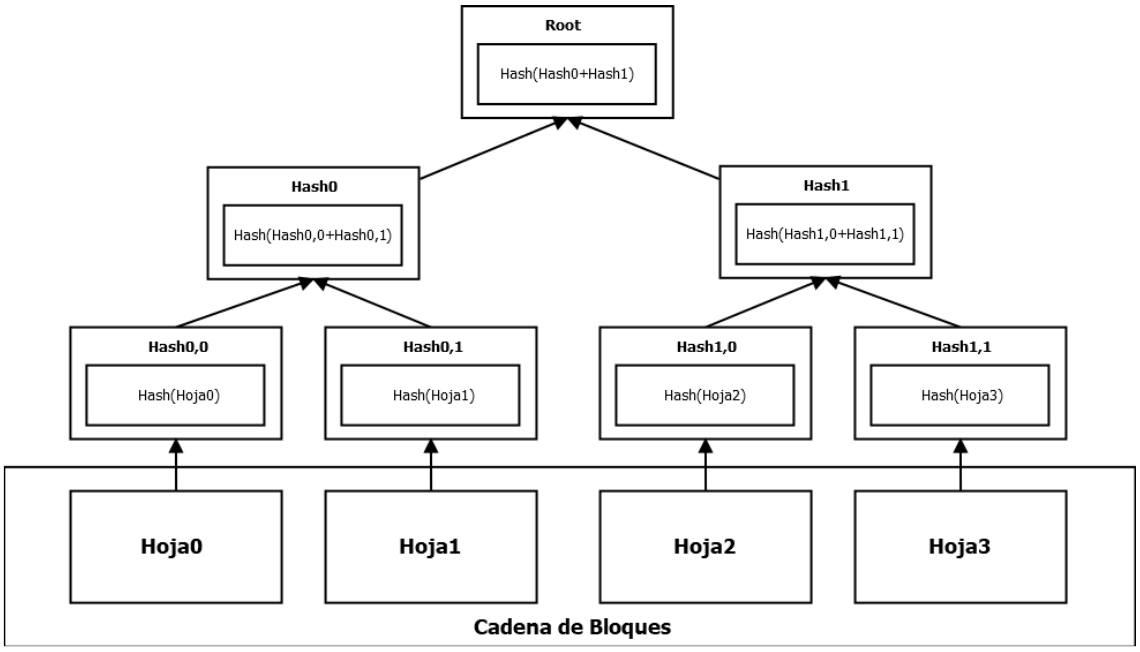

Figura 3. Estructura Merkle Patricia Tree

Gracias a esta estructura es posible validar la integridad de los datos almacenados en un nodo hoja solo con calcular los hashes de sus nodos padre, por lo que no es necesario disponer de los datos del resto de nodos hoja. A diferencia de otras cadenas de bloques, los bloques de Ethereum almacenan referencias a bloques uncle. Dichos bloques son bloques válidos por los que la red no apostó continuar y a los que no consideró parte de la cadena principal. Aunque en otras cadenas serían descartados, Ethereum los conserva para aumentar la seguridad de la Blockchain e intentar descentralizar la minería recompensando a los mineros por su generación.

- **Máquina Virtual**: Encargada de ejecutar los smart contracts que consultan y modifican el estado de la red. La EVM (Ethereum Virtual Machine) se ejecuta en los nodos de la red e interpreta el código máquina proveniente de la compilación de lenguajes de alto nivel (Solidity). De esta forma, la red se comporta como un sistema de ejecución distribuida, donde el código se almacena de manera inmutable en la cadena y donde los datos calculados son validados con las sucesivas ejecuciones de los mineros. Este consenso en la ejecución, aunque permite validar de forma distribuida los resultados de los smart contracts, restringe la ejecución del código no determinista en la cadena, por lo que hace que operaciones puramente aleatorias no sean posibles.
- **Algoritmo de consenso**: actualmente implementa un algoritmo de Proof-of-Work denominado Ethash. No obstante, existen planes de migrar a Proof-of-Stake.
- **Clientes**: Nodos de la red que interactúan con ella. Sus implementaciones más extendidas son Parity y Geth.

- **Transacciones**: Mensajes enviados por clientes de la red que incluyen origen, destino, valor y datos adicionales. Cada transacción en Ethereum requiere invertir una cantidad de gas determinado, este valor sirve para recompensar a los mineros con Ether por su cómputo y para evitar que las transacciones puedan ejecutarse infinitamente.

Desde un punto de vista práctico, Ethereum es un entorno de computación distribuido y descentralizado que emplea la tecnología Blockchain como medio de almacenamiento y sincronización de los datos. La forma de almacenar datos puede ser expresada como el conjunto Clave-Valor, al igual que el modelo de memoria RAM de un ordenador convencional. De igual manera, la memoria de Ethereum permite almacenar programas (smart contracts) que pueden ser ejecutados con el fin de leer y/o modificar datos. A diferencia de un ordenador común, todos los cambios en la memoria de Ethereum deben respetar el algoritmo de consenso.

Nota: Ethereum está aún en desarrollo y, por tanto, está sujeta a modificaciones y futuras evoluciones.

VII. IMPLICACIONES LEGALES DEL USO DEL BLOCKCHAIN

En esta sección se estudian las implicaciones legales del primer caso de uso de Blockchain, las criptodivisas, así como de otros casos de uso derivados del uso de blockchain y Smart contracts.

¿Qué son los smart contracts desde una perspectiva legal?

¿Nos llevan los smart contracts hacia una economía sin derecho contractual? A estas alturas todos hemos leído sobre smart contracts, qué son y cómo funcionan, pero poco se ha dicho aún sobre sus implicaciones en el mundo jurídico.

Actualmente, hemos encontrado en el mundo del derecho un fenómeno socioeconómico cada vez más generalizado: la sustitución del hombre por la máquina. En el ámbito económico, las relaciones comerciales (mediadas por el derecho) pretenden traernos la idea de que la maquinización y la informática pueden reemplazar el factor humano. Esto se trata de la sustitución del aparataje jurídico del que disponemos (cuerpos legales, documentos, registros, abogados, tribunales, etc.) por dispositivos y programas informáticos.

Cuando hablamos de un Smart Contract como un contrato que se ejecuta a sí mismo, ¿somos conscientes de que la única tarea de un programa informático es manejar información? Se trata de hacer operaciones con unos datos siguiendo instrucciones para dar como resultado unos nuevos datos.

El concepto de Smart Contract fue enunciado por Nick Szabo, en el año 1996, en la revista Extropy Nº 16: Smart Contracts: Building Blocks for Digital Free Markets. En este artículo Szabo trata de explicar los últimos avances en materia de protocolos criptográficos y el potencial que ello tenía sumado al concepto de Internet, donde podría revolucionarse el derecho contractual tradicional.

Szabo define los smart contracts como a set of promises, specified in digital form, including protocols within which the parties perform on the other promises. De esta definición podemos destacar tres elementos básicos:

- Conjunto de promesas
- Especificadas de forma digital.
- Con inclusión de protocolos que den cumplimiento a dichas promesas

Se trata de protocolos informáticos, meras instrucciones que producen cambios de estado en la información registrada en una BBDD. Se trata, pues, de traducir el clausulado de un contrato expresado en lenguaje natural, mediante el que se prometen determinadas prestaciones o se acuerdan determinadas consecuencias, si se producen ciertos eventos en una serie de instrucciones que rigen el comportamiento de dicho protocolo.

Así, siempre que las prestaciones establecidas en el contrato se puedan realizar en el ámbito informático, la programación del mismo automatiza su ejecución, haciendo que esta no dependa de la voluntad de las partes.

… Y después llegó Blockchain. Como se ha anticipado, el concepto de Smart Contract se enuncia en el año 1996, mientras que Blockchain aparece para dar vida a Bitcoin en el año 2008. No obstante, esta tecnología ha permitido dar vida a los smart contracts y, a continuación, explicaremos por qué.

Bitcoin ha sido el primer ejemplo de «tokenización» de activos o fichas susceptibles de tráfico en un entorno puramente digital, sin necesidad de intervención de terceros. En definitiva, se trata de dinero programable que puede ser gestionado desde un protocolo informático.

Por otra parte, está Blockchain. Su tecnología subyacente permite que el contrato o la relación económica programada en forma de código, que se autoejecuta sin necesidad de alojar información alguna en los servidores de una de las partes de forma manipulable u opaca, sino que se incorporará en un registro distribuido en red, repartido entre miles de ordenadores, de forma transparente y securizado mediante criptografía.

En el caso de Ethereum, dichos programas se ejecutan automáticamente y de forma transparente en miles de ordenadores. De ahí el sobrenombre de Ethereum, «The world computer». Gracias a ello Ethereum se ha convertido en la plataforma por excelencia para el desarrollo de smart contracts, pues se trata de una herramienta de código abierto para crear smart contracts. Desde combinaciones muy sencillas como un contrato de apuesta con dos partes, hasta relaciones mucho más complejas multiparte similares a un contrato de sociedad, como es el caso de DAO (Decentralized Autonomous Organization) que es una nueva modalidad de empresa: las relaciones entre los participantes se rigen por un smart contract complejo, escrito y programado sobre una plataforma Blockchain.

Una vez entendido cómo hemos llegado al concepto de smart contracts, vamos a hacer una serie de puntualizaciones acerca de los mismos.

¿Son realmente contratos los smart contracts? Para un abogado un contrato es un acuerdo de voluntades del que nacen obligaciones jurídicas. El smart contract, en principio, es ajeno a la idea de promesa y de obligación jurídica: uno no se obliga a algo cuya realización se ha automatizado.

¿Son realmente inteligentes? El contrato como vínculo jurídico es una entidad ideal, una cosa pensada. La inteligencia se predica de agentes, de seres animados (una persona, un animal) o capaces de un proceso o actividad (un robot, un programa informático).

Lo inteligente del smart contract es el programa que lo ejecuta, no el programa en sí mismo. Por lo tanto, se trata de un software informático, un programa, instrucciones con las que opera una máquina.

La cuestión relevante es si un Smart Contract puede ser calificado jurídicamente como un contrato o no.

Muchos consideran que el smart está más allá del concepto jurídico de contrato. De hecho, no es que no sea un contrato, es que no pretende serlo. Para estos, el smart contract es autosuficiente, no necesita ni demanda el apoyo de jurisdicción estatal alguna.

También se puede pensar que, de conformidad con nuestro derecho actual, una herramienta en la que se prescinde de la idea de obligarse uno jurídicamente en el marco de una determinada jurisdicción no sería un «contrato» y, por tanto, nunca podría buscar el apoyo del derecho del Estado. La propia lógica jurídica expulsaría al smart contract de su ámbito de actuación.

Un planteamiento intermedio entre ambos sería el siguiente: las partes de un smart contract quieren beneficiarse de las ventajas prácticas de este instrumento, pero no por ello han de renunciar a los remedios del derecho contractual tradicional si el mecanismo no llega a funcionar como habían previsto. Por lo tanto, en la medida en que el Smart Contract es el instrumento de relaciones económicas, el derecho –con su pretensión de orden y justicia– no puede desentenderse de lo que ahí está pasando.

VIII. RESULTADOS DEL PROYECTO

A continuación, se muestra algunas de las consideraciones más relevantes del desarrollo y resultados obtenidos del proyecto.

VIII) 1.PLATAFORMA DE E-LEARNING BASE

La metodología del SPOC tiene su base central en la utilización de materiales didácticos de corta duración para la transmisión teórica de conocimientos y actividades. A continuación, se presenta parte del contenido desarrollado para este curso y que forma la base del mismo. Los parámetros que se analizaros para la elección de las plataformas, tecnologías y contenidos son los siguientes: software actual, en la medida de lo posible libre, facilidad de instalación,

inserción de contenido de manera fácil, recursos didácticos de autoevaluación, patrones de comportamiento de los estudiantes para su análisis posterior, etc.

Después de realizar una búsqueda de plataformas que se ajustaran a nuestras necesidades para alojar nuestro SPOC, se encontraron diferentes alternativas:

- Moodle (module object-orientes dynamic learning environment): Moodle no es una plataforma creada de manera específica para realizar MOOC o cursos similares, pero debido a las propiedades que presenta, se postuló como primera alternativa dadas las necesidades del proyecto y a su adaptación como entorno de aprendizaje virtual. Las ventajas que ofrece esta plataforma son muchas, como por ejemplo, su compatibilidad con los distintos sistemas operativos y navegadores, la gran cantidad de material de ayuda, la comunidad de usuarios y técnicos, la posibilidad de implantar diversos tipos de recursos pedagógicos autoevaluables como test, lecciones, talleres, herramientas de comunicación síncronas y asíncronas (foros, chats, etc).

- Open edX: edX es una plataforma de cursos abiertos masivos en línea (MOOC, Massive Online Open Course), basada en software de código abierto. Fue fundada por el Instituto Tecnológico de Massachusetts y la Universidad de Harvard en mayo de 2012 para hospedar cursos en línea de nivel universitario de un amplio rango de disciplinas, para todo el mundo sin costos para propiciar la investigación y el aprendizaje. edX tiene más de 10 millones de usuarios registrados y es considerado el segundo proveedor más grande de MOOC en el mundo, después de Coursera[2] que cuenta con 23 millones de usuarios registrados. Los cursos de edX consisten en secuencias de aprendizaje. Cada secuencia se compone de distintos recursos de aprendizaje como videos, podcasts, lecturas, foros de discusión, infografías; y ejercicios de evaluación o interacción como evaluaciones de opción múltiple, preguntas de respuesta abierta, sondeos, preguntas de abiertas con revisión de pares, drag and drop, etc. Los participantes deberán completar estas actividades para alcanzar el puntaje requerido y así, de necesitarlo, obtener el certificado de aprobación. Existen dos tipos de modalidades de cursos: los cursos a "tu propio ritmo" y los cursos al "ritmo del profesor". La mayoría de los cursos de edX son al ritmo del profesor, suelen tener una duración de 4 a 8 semanas, con nuevo contenido publicado cada semana y fechas de entrega preestablecidas. Al analizar esta plataforma se observó que, aunque posee mucho de los requisitos necesarios para el proyecto, presenta algunas desventajas como por ejemplo la ausencia de herramientas de análisis y patrones posteriores, la dificultad de visionado de ejemplos específicos y vídeos.

[2] Coursera es una plataforma de educación virtual nacida en octubre de 2011 y desarrollada en la Universidad de Stanford con el fin de brindar oferta de educación masiva a la población (Massive Online Open Course).. Coursera ofrece cursos, tanto gratuitos como de pago, sobre temas variados a niveles universitarios, pero abiertos a todos los sectores de la población. Fue fundada por los profesores en ciencias de la computación Andrew Ng y Daphne Koller en octubre de 2011 con el lanzamiento de dos cursos gratuitos, "Aprendizaje automático" e "Introducción a las bases de datos".

- OpenMOOC: OpenMooc es una plataforma open source o recurso abierto, que se creó en 2012 y que fue financiada por la Universidad Nacional de Educación a Distancia (UNED) de España. Se trata de un recurso diseñado para cursos masivos en línea, cuya arquitectura integra funcionalidades a partir de otros recursos libres como Amazon S3, Google Analitycs, Sendgrid, etc. Las desventajas que encontramos en esta plataforma fue principalmente la falta de un equipo de desarrolladores activos, y una comunidad de apoyo, como si ocurre en Moodle y Open edX.

Por todo lo indicado, la plataforma elegida fue Moodle. Además, nos encontrábamos con la ventaja de tener a mano dos instalaciones de esta plataforma, la institucional de la universidad (Studium) y el campus del grupo de investigación de los miembros integrantes del proyecto (campus-bisite.usal.es), ambos accesibles y con una curva de aprendizaje mínimo por parte de los creadores del curso y también de los futuros estudiantes. Se llevó a cabo la creación de un curso en el campus del grupo de investigación, dividido en varios temas, con unidades temáticas, ejercicios y unidades de autoevaluación.

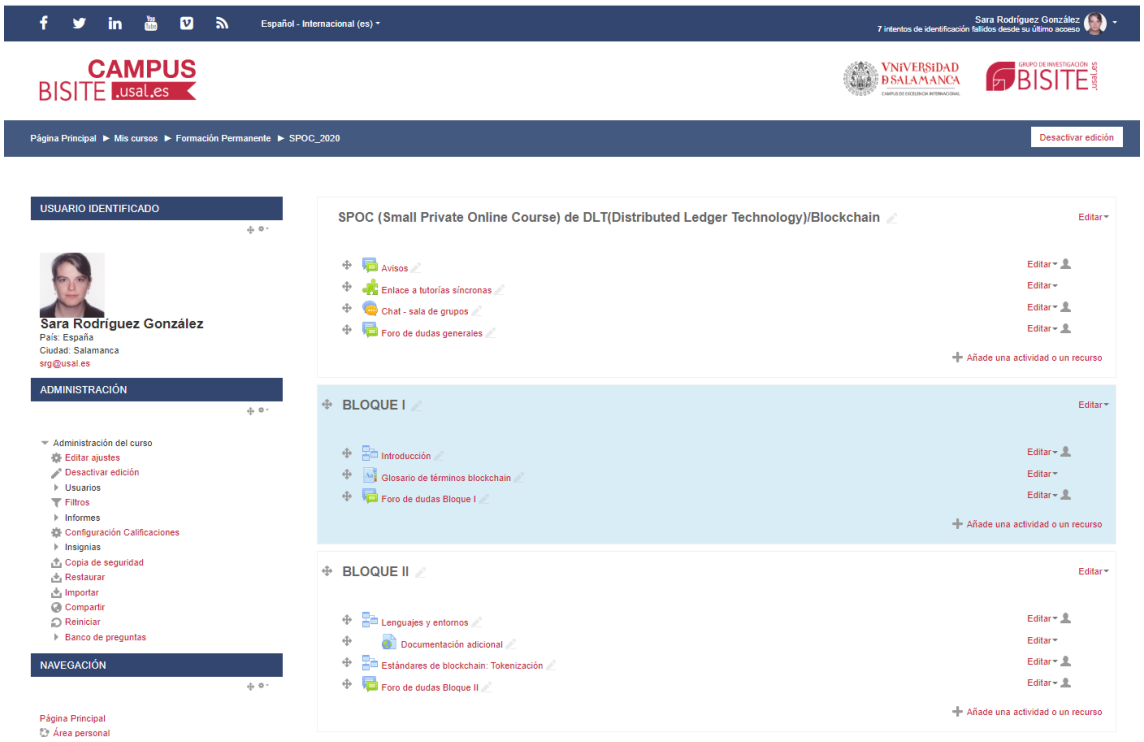

Figura 4. SPOC en Plataforma Moodle I

Figura 5. SPOC en Plataforma Moodle II

VIII) 2. DESARROLLO DE APLICACIONES BLOCKCHAIN

Se incluye en esta sección material de ejemplo del curso creado, con ejemplos de las plataformas específicas utilizadas e imágenes de los recursos pedagógicos.

Para el desarrollo de las aplicaciones blockchain el lenguaje que se va a utilizar es solidity. En el curso se introducen los conceptos básicos de este lenguaje de programación y, además, se dan una serie de recomendaciones relacionadas con las buenas prácticas y el uso de los algunos de los estándares de programación más comunes en blockchain.

VIII) 2. 1. LENGUAJE SOLIDITY

Solidity es un lenguaje orientando a contratos que permite la codificación de smart contracts en varias plataformas de Blockchain.

El compilador de Solidity, denominado Solc, es el encargado de transformar los programas escritos en Solidity en bytecode de la EVM y de generar los artefactos asociados al contrato (tales como la ABI). Actualmente, Solidity se encuentra en desarrollo y su versión actual puede sufrir cambios. Por este motivo, cada fichero codificado en dicho lenguaje debe especificar en la parte superior la versión con la que va a ser compilado.

```
pragma solidity ^0.5.2;
```

VIII) 2. 2. TIPOS DE DATOS

En el lenguaje Solidity existen los siguientes tipos de datos básicos:

- **int/uint**: entero con signo (int) o sin signo (uint) que permite almacenar 256 bits. En caso de querer limitar su tamaño, soporta sufijos numéricos correspondientes a múltiplos de 8: **uint8**, **int8**, **uint16**, **int16**, etc.

```
uint8 mark = 9;
```

- **bool**: valor booleano, **true** o **false** que soporta los siguientes operadores lógicos: **!** (negación), **&&** (y), **||** (o), **==** (igual), **!=** (distinto).

```
bool pass = true || false;
```

- **address**: dirección de Ethereum con tamaño de 20 bytes. Dispone de los siguientes atributos:
  - **balance**: valor con el saldo de la cuenta en Ether.
  - **transfer**: función que permite enviar Ether a otra cuenta. En caso de fallo, el contrato donde se ejecuta se detiene y devuelve un error.
  - **send**: función idéntica a **transfer**, pero que no se detiene en caso de fallo y únicamente devuelve **false**
  - **call**: llamada de bajo nivel que devuelve **false** en caso de error.

    - **call.value**: Permite especificar los fondos que se envían a la función (esta debe estar definida cómo **payable**).
    - **call.value.gas**: permite especificar el gas invertido para realizar la ejecución.

```
direccion.call.value(msg.value).gas(22000)();
```

- **bytes1, bytes2,…,bytes32**: array de bytes de longitud fija determinada por el número indicado en su nombre. Soportan operadores de bits tales como: & (y), | (o), ^ (xor), ~ (negación), << (desplazamiento izquierdo), >> (desplazamiento derecha). Los valores contenidos en el array pueden ser accedidos mediante índice

```
bytes4 prime;
prime[0] = 1;
prime[1] = 2;
prime[2] = 3;
prime[3] = 5;
```

- **bytes/string**: array dinámico de **bytes**. En caso de requerir el uso de caracteres UTF-8 utilizar **string.**

```
bytes surname = "Buterin";
string name="Vitalik";
```

- **enum**: valores discretos definidos por el usuario.

```
enum Hand {left, right}
Hand mihand = Hand.right;
```

- **struct**: contenedores de valores cuyo propósito es ser accedidos de forma conjunta.

```
struct Person {
    string name;
    string surname;
}
Person vitalik;
Person.name = "Vitalik";
Person.surname= "Buterin";
```

- **mapping**: tabla de hashes que permite almacenar la relación clave-valor.

```
mapping (address=>string) Identity;
Identity(0x01) = "Vitalik";
Identity(0x02) = "Gavin";
```

VIII) 2. 3. FUNCIONES Y VARIABLES PREDEFINIDAS

Las siguientes variables y funciones están disponibles de manera predeterminada en la EVM. Se trata de opcodes complejos que implementan funcionalidades típicas y recurrentes en el entorno Blockchain:

- **msg**: objeto con información relativa a la transacción o mensaje que originó la ejecución. Dispone de los siguientes atributos:
    - msg.sender: representa el identificador de cuenta que originó la transacción. Si la función se invoca desde otro contrato, el identificador corresponde a dicho contrato.
    - msg.gas: la cantidad de gas sin utilizar que queda para la ejecución de nuestro contrato.
    - msg.data: los datos en crudo de nuestra llamada.
    - msg.sig: los primeros cuatro bytes de msg.data que corresponden con la función a invocar.

```
if(msg.sender == 0x01)
string nombre = "Vitalik";
```

- **tx**: objeto con información de la transacción actual. Dispone de los siguientes campos:
    - tx.gasprice: el precio del gas para la transacción.
    - tx.origin: el origen de la pila de llamadas para la transacción.

```
if(tx.origin == msg.sender)
uint price = tx.gasprice;
```

- **block**: objeto con información del bloque actual. Dispone de los siguientes campos:
    - block.coinbase: la dirección del minero del bloque actual.
    - block.difficulty: la dificultad de Proof-Of-Work del actual bloque.
    - block.gaslimit: el límite de gas acordado para el actual bloque.
    - block.timestamp: la fecha fijada por el minero en formato Unix.

```
if(block.coinbase == msg.sender)
uint gaslimit = block.gaslimit;
```

- **addmod(x,y,z)/mulmod(x,y,z)**: calcula el módulo de la suma o el módulo de la multiplicación de la siguiente forma: (x+y)%z y (x*y)%z.

```
bool functionA = addmod(2,3,2) == (2+3)%2;
```

- **keccak256, sha256, sha3, ripemd160**: calculan las funciones de hash utilizando los algoritmos estándar.

```
bytes32 hash = sha256("abc");
```

- **ecrecover**: recupera el identificador de clave de un mensaje firmado.

```
bytes32 hash = sha3("abc");
bytes32 firma = web3_eth_sign_abc;
assembly {
    r := mload(add(firma, 32))
    s := mload(add(firma, 64))
    v := and(mload(add(firma, 65)), 255)
}
bytes memory prefijo = "\x19Ethereum Signed Message:\n32";
hash = sha3(prefijo, hash);
address firmante = ecrecover(hash,v,r,s);
```

VIII) 3.CONTRATOS

Siendo similares a objetos, los contratos permiten al desarrollador incluir datos y/o funciones. Aunque se trata del principal tipo de datos en Solidity, existen también los siguientes tipos:

- **interface**: son contratos donde las funciones pueden ser declaradas, pero no definidas. En caso de que un contrato herede de una interfaz (su principal uso en Solidity), este último deberá implementar todas sus funciones

```
interface calculator{
    function add(uint a, uint b) public returns(uint);
}

contract calculator_memory is calculator{
    int memorycal = 0;
    function add(uint a, uint b) public returns(uint){
            memorycal = a + b;
            return memorycal;
        }
}
```

- **library**: contratos que no pueden instanciarse y que carecen de estado (variables globales). En caso de llamar a una de sus funciones debe hacerse con delegatecall.

```
library calculator{
    function sumar(uint a, uint b) public returns(uint){
        return a+b;
    }
}
```

VIII) 3. 1. FUNCIONES PERSONALIZADAS

La definición de funciones dentro de un contrato tiene la siguiente forma:

function Name ([Params]) {Visibility} [Type] [Modifiers] [returns (<ReturnTypes>)]

- **Name**: el nombre de la función a escoger por el programador. Existe un tipo de función sin nombre que se denomina fallback, esta función no puede tener argumentos ni valor de retorno. Su ejecución sucede cuando no se especifica un nombre de función al llamar a un contrato.
- **Params**: lista de parámetros de la función, especificados con tipo y nombre.
- **Visibility**: por defecto, toma el valor public. Teniendo en cuenta que todo el código es visible en la cadena de bloques, la visibilidad afecta únicamente al origen desde el cual puede invocarse la función. Dicho campo puede tomar los siguientes valores:

- **Public**: las funciones de este tipo pueden ser llamadas desde otros contratos, por usuarios o dentro del propio contrato.
- **External**: las funciones de este tipo funcionan igual que **public** pero no pueden ser llamadas desde el mismo contrato sin especificar **this**.
- **Internal**: las funciones de este tipo únicamente se pueden llamar desde dentro del contrato o desde uno que herede de este.
- **Private**: las funciones de este tipo funcionan igual que **Internal** pero no se pueden llamar desde un contrato que herede de este.

- **Type**: Especifica el tipo de función con base en cómo afecta el estado de la cadena. Puede tomar los siguientes valores.
  - **constant/view**: indica que la función no modifica el estado y únicamente accede al mismo, por tanto, se ejecutará localmente sin ser reenviada a la red.
  - **pure**: indica que el estado de la red ni se accede ni se modifica. Al igual que el tipo **constant/view**, se ejecutará localmente sin ser reenviada a la red.
  - **payable**: indica que la función puede recibir pagos con las funciones **send**, **transfer** y **call.value**. Para enviar dinero al contrato, hacer payable la función **fallback**.

- **Modifiers**: lista de modifiers que se comprobarán antes de ejecutar la función. Los modifiers son un tipo de función especial cuyo propósito es el de crear condiciones que se comprueben previas a la ejecución de la función. Dentro de su definición se permite la sentencia especial _;. Con ella, se identifica el cuerpo de la función dentro de nuestro modifier para aplicar la lógica que requiramos y ejecutar la función en un determinado punto del código.

```
contract Helloworld {

modifier soloPares (int a, int b){
    if(a % 2 == 0 && b % 2 == 0)
        {
        _;
        }
    }
```

- **ReturnTypes**: lista de valores de retorno de la función, especificados con tipo y nombre.

```
function sumaYresta (int a, int b) public pure soloPares(a, b) returns (int, int){
    returns (a+b, a-b);
}
```

Existe un tipo de función especial denominada **constructor**. Esta función puede tomar el nombre constructor o tener el mismo nombre que el contrato. Dicha función será ejecutada en el momento de instanciar el contrato, por lo que, si el nombre del contrato cambiara, podremos encontrar un comportamiento inesperado o no deseado (llegando incluso a ser explotado por usuarios malintencionados). Por esta razón se recomienda el uso de constructor en lugar del nombre del contrato.

De manera opuesta, en caso de querer eliminar un contrato podremos llamar a la función **selfdestruct** con la cuenta donde queremos depositar el Ether devuelto como argumento.

La definición de variables complejas dentro de un contrato (arrays y structs) puede indicar también el tipo de almacenamiento que consumirán dichas variables. Estos tipos son los siguientes:

- **memory**: es el valor por defecto para parámetros y valores de retorno y su precio es menor que las del tipo **storage**. Su asignación provoca:
  - **memory=memory**: no crea copia, es un paso por referencia
  - **storage=memory**: realiza una copia completa.
- **storage**: es el valor por defecto para variables locales y variables de estado (en este último caso es el único valor posible). Su asignación provoca:
  - **memory=storage**: realiza una copia completa.
  - **storage=storage**: no crea copia, es un paso por referencia.

VIII) 3. 2. HERENCIA

Este mecanismo permite extender la funcionalidad de contratos ya definidos, de forma que el programador pueda comenzar con contratos simples e ir extendiendo su funcionalidad y complejidad a lo largo del tiempo. Solidiy soporta polimorfismo y herencia múltiple que se implementan mediante la sentencia **is**:

```
contract Hijo is Padre1, Padre2{
}
```

La herencia implica que los contratos hijos hereden todas las variables y funciones (incluidos modifiers) de los contratos padre. No obstante, estos pueden ser sobrescritos por el hijo en caso de requerirlo. El hijo puede invocar la función del padre utilizando la palabra reservada **super** dentro de la función heredada. En caso de herencia múltiple, donde los padres comparten nombre de función, es posible referirse a ejecutar la función de un contrato padre concreto indicando su nombre:

```
contract C{
    uint u;
    function f() { u = 1; }
}
contract B is C{
    function f() { u = 2; }
}
contract A is B{
    function f() { u = 3; } // u = 3
    function f1() { super.f(); } // u = 2
    function f2() { B.f(); } // u = 2
    function f3() { C.f(); } // u = 1
}
```

En caso de sobrescribir un constructor y requerir la ejecución del constructor del padre, es posible invocarlo en la propia definición de la función (al estilo modifier):

```
contract Helloworld {

contract Padre{
    uint x; function Padre(uint a) { x=a; }
}
contract Hijo{
    function Hijo(uint a) Padre (a*a) { }
}
```

VIII) 3. 3. CONTROL DE ERRORES

La ejecución de una función en un contrato puede acabar y devolver un error, en cuyo caso todos los cambios realizados en el estado son revertidos (variables, balances, etc.). Para ello, Solidity ofrece las siguientes funciones:

- **assert**: detiene la ejecución, genera un error en caso de no cumplirse la condición y no devuelve el gas no utilizado. Por convención, debe utilizarse para controlar errores internos y está enfocada al análisis estático (si sucede, es necesario corregir el código).
- **require**: detiene la ejecución, genera un error en caso de no cumplirse la condición y devuelve el gas no utilizado. Por convención, debe utilizarse para comprobar parámetros de entrada. Es posible incluir un mensaje de error como parámetro.
- **revert**: detiene la ejecución, genera un error y devuelve el gas no utilizado. Por convención, se utiliza para condiciones complejas.

```
contract Error{
    function SumaPares(uint a, uint b) returns (uint){
            require(a % 2 == 0 && b % 2 == 0, "No son pares");
            uint suma = a + b;
            assert(suma >= a && suma >= b);
            return suma;
        }
}
```

VIII) 3. 4. EVENTOS

Solidity ofrece eventos para generar logs en funciones que se ejecutan correctamente o no (queda constancia de ellos en la transacción, aunque se lance un error). Estos son especialmente útiles en aplicaciones que deben reaccionar ante cambios de estado (tales como una interfaz de usuario). La definición de un evento sigue la siguiente estructura:

event Name (Params)

- **Name**: nombre del evento a escoger por el desarrollador.
- **Params**: lista de parámetros del evento, especificados con tipo y nombre. El valor de estos parámetros se almacena en los logs generados.

```
contract Error{
    event SumaSatisfactoria(uint a, uint b, uint s); function SumaPares(uint a, uint b) returns (uint){
        require(a % 2 == 0 && b % 2 == 0, "No son pares");
        uint suma = a + b;
        assert(suma >= a && suma >= b);
        SumaSatisfactoria(a, b, suma);
        return suma;
    }
}
```

VIII) 3. 5. TAREA: PROGRAMACIÓN DE UNA CALCULADORA

Se presenta en esta sección un ejemplo de tarea, en la que los alumnos deben poner en práctica los conceptos vistos en los recursos explicativos.

En este caso, es necesario extender el funcionamiento del contrato *Calculadora.sol* con las siguientes funciones y características:

- **Memoria por usuario**: actualmente la calculadora dispone de una memoria global compartida por todos los usuarios. Es necesario que cada usuario que haga uso de la calculadora disponga de su propia memoria. El número de usuarios que pueden usar la calculadora no es conocido *a priori*.

- **Nuevas funciones a incluir**:
  - **public multiplica(int256 a, int256 b) public pure returns (int256)**
    - Devuelve el producto de *a* y *b*.
  - **public divide(int256 a, int256 b) public pure returns (int256)**
    - Devuelve el cociente de dividir *a* y *b*. En caso de que *b* sea 0, debe devolver un error con assert/require.
  - **public multiplicaMemoria(int256 a) public returns (int256)**
    - Devuelve el producto de *a* y el valor almacenado en memoria.
  - **public divideMemoria(int256 a) public returns (int256)**
    - Devuelve el cociente de dividir *a* entre el valor almacenado en memoria. En caso de que el valor de memoria sea 0, debe generar un evento **DivisionPorCero** indicando el identificador de usuario y el valor de *a*.
  - **public factorial(int256 a) public pure returns (int256)**
    - Devuelve el valor factorial de *a*. En caso de que el valor de *a* sea negativo, debe devolver un error con assert/require.

No será necesario entregar la interfaz de usuario web que haga uso de estas nuevas funcionalidades, únicamente se entregará el fichero CalculadoraExtendida.sol. El fichero entregado por el alumno será compilado y ejecutado mediante scripts que, a su vez, lanzarán baterías de pruebas. En estas pruebas se comprobarán los valores devueltos por cada una de las funciones para evaluar la nota del alumno.

Objetivos

El alumno deberá conocer y hacer uso de las estructuras de datos proporcionadas por Solidity para llevar a cabo la implementación de la memoria segmentada por usuario. Además, se utilizarán sentencias de control de errores para aquellas funciones que lo especifican (**divide** y **divideMemoria**). Por último, el alumno deberá resolver la función **factorial** evitando la recursividad, por lo que tendrá que utilizar sentencias iterativas ofrecidas por el lenguaje de programación.

```
contract Calculadora {
    //Atributos
    int256 private _memoria;
    //Constructor
    function Constructor() public
    {
        _memoria = 0;
    }
    //Funciones
    function suma(int256 a, int256 b) public pure returns (int256)
    {
        return a + b;
    }
    function sumaMemoria(int256 a) public view returns (int256)
    {
        return a + _memoria;
    }
    function resta(int256 a, int256 b) public pure returns (int256)
    {
        return a - b;
    }
    function restaMemoria(int256 a) public view returns (int256)
    {
        return a - _memoria;
    }
    function almacenaMemoria(int256 resultado) public returns (int256)
    {
        _memoria = resultado;
        return resultado;
    }
    function obtenerMemoria() public view returns (int256)
    {
        return _memoria;
    }
    function borrarMemoria() public
    {
        _memoria = 0;
    }
}
```

Figura 6. Código para CalculadoraExtendida.sol

VIII) 3. 6. POSIBLE SOLUCIÓN

```
pragma solidity ^0.5.2;

contract Calculadora {
    //Atributos
    //int256 private _memoria;

    //Constructor, ya no es necesario el constructor por los cambios realizados
   // function Constructor() public
    //{
    //    _ownerToMemoria[msg.sender] = 0;
    //}

    //Mappings
    mapping (address => int256) private _ownerToMemoria;

    //Eventos
    event DivisionPorCero(address _dir, int256 _n, string _error);

    //Funciones
    function suma(int256 a, int256 b) public pure returns (int256)
    {
        return a + b;
    }
```

```
    function sumaMemoria(int256 a) public view returns (int256) //se quita el view
porque almacena en memoria
    {
        return a + _ownerToMemoria[msg.sender]; //suma a la memoria pero no almacena
    }
    function resta(int256 a, int256 b) public pure returns (int256)
    {
        return a - b;
    }
    function restaMemoria(int256 a) public view returns (int256)//se quita el view
porque almacena en memoria
    {
        return a - _ownerToMemoria[msg.sender]; //resta al numero la memoria pero no
almacena
    }
    //*********************************************************************
    //****************** Funciones avanzadas ****************************
    //multiplicacion
    function multiplica(int256 a, int256 b) public pure returns (int256)
    {
        //Devuelve el producto de a y b.
        return a * b;
    }
    //multiplicacion con la memoria
    function multiplicaMemoria(int256 a) public returns (int256)
    {
        //Devuelve el producto de a y el valor almacenado en memoria.
        return a * _ownerToMemoria[msg.sender]; //memoria almacenada en msg.sender
    }
    //*********************************************************************
    //division con control de errores con require
    function divide(int256 a, int256 b) public pure returns (int256)
    {
        //Devuelve el cociente de dividir a y b.
        //En caso de que b sea 0, debe devolver un error con assert/require
        require(b != 0,"Error: No se puede dividir por 0");
        return a / b;
    }
    //*********************************************************************
    //division por memoria y control de errores con un evento
    function divideMemoria(int256 a) public returns (int256)
    {
        //Devuelve el cociente de dividir a entre el valor almacenado en memoria.
        //En caso de que el valor de memoria sea 0, debe generar un evento
DivisionPorCero
        //indicando el identificador de usuario y el valor de a.
        int256 aux;
        if(_ownerToMemoria[msg.sender]==0){
             emit DivisionPorCero(msg.sender, a, "Esta operaciÃ³n no se puede realizar,
se devuelve el valor introducido");
             aux =a; //se devuelve el valor de "a" al lanzar el evento, ya que sino el
compilador
                     //da un error porque el error aunque se lanze en el evento no se
controla
        }else{
            //en este caso el valor de la variable auxiliar toma el valor adecuado de la
division
            aux = a / _ownerToMemoria[msg.sender]; //memoria almacenada en msg.sender
        }
        return aux;
    }
    //*********************************************************************
    //*********************************************************************
    //factorial
    function factorial(int256 a) public pure returns (int256)
    {
        //Devuelve el valor factorial de a.
        //En caso de que el valor de a sea negativo,
        //debe devolver un error con assert/require.
        require(a >= 0,"Error: El valor introducido debe ser entero y positivo"); //
para evitar numeros negativos

        int256 aux_fact = a;

        if(a == 0)
            {
                aux_fact = 1; //aqui imponemos la condicion de que 0! = 1
```

```
            }
        else
            { // se calcula el factorial del resto de numeros
                for( int256 i = 1; i < a; i++ )
                {
                    aux_fact *= i;
                }
            }
        return int256(aux_fact); //se devuelve a valor pedido haciendo un casting a
int256 como se pide
    }
    //**********************************************************************
    //**********************************************************************

    //crear una memoria para cada usuario, almacenar la memoria en msg.sender para cada
usuario
    function almacenaMemoria(int256 _resultado) public returns (int256)
    {
        _ownerToMemoria[msg.sender] = _resultado;
        return _resultado;
    }
    function obtenerMemoria() public view returns (int256)
    {
        return _ownerToMemoria[msg.sender]; //memoria almacenada en msg.sender
    }
    function borrarMemoria() public
    {
        _ownerToMemoria[msg.sender] = 0; //memoria almacenada en msg.sender
    }
}
```

VIII) 4.ESTANDARES DE BLOCKCHAIN: TOKENIZACIÓN

En esta sección se muestra ejemplos de contenidos realizados para la explicación de los conceptos básicos para poder usar los estándares ERC20 y ERC721, para hacer uso de la tokenización en blockchain. Además, se verán las principales ventajas de la tokenización frente a los activos digitales tradicionales.

VIII) 4. 1. ¿QUÉ ES UN TOKEN?

Cuando nos referimos al ecosistema Blockchain, principalmente al de la Blockchain Ethereum, y hablamos de token, nos refierimos a un fragmento de código que busca representar valor, un activo dentro de la red. Mediante estos fragmentos, la tokenización busca establecer la forma de convertir los activos tradicionales en activos tokenizados, definiendo los procesos, parámetros y funcionalidades que deben representar estos tokens, así como los requerimientos legales que necesitan cumplir los activos para poder ser tokenizados.

Partiendo de esta base, surgen dentro de la tecnología Blockchain otros tokens, para ser intercambiados por tokens de criptomonedas y que representan activos como acciones, participaciones, derechos de voto, puntos de fidelización y gran cantidad de activos que pueden ser tokenizados

Una de las principales ventajas de la tokenización de activos es la liquidez que aportan a los mercados. Al tokenizar un activo, este se puede equiparar a un activo financiero, fácilmente intercambiable, lo cual genera liquidez.

Dentro de Ethereum se han creado dos estándares principales para la creación de tokens ERC20 y ERC223 (pero este estándar queda fuera del alcance de este curso), pensados para la creación de elementos de valor que pueden ser intercambiados por servicios, participaciones o cualquier otro activo al que se quiera vincular. Esta idea nos lleva a uno de los principales problemas actuales de los tokens y las ICO; no existe una regulación ni una normativa que dé a los usuarios de un token derecho real sobre su representación. En Ethereum existe también el estándar ER721, pensado para elementos no fungibles, que busca aportar al ecosistema Ethereum un formato para intercambiar activos, cuyo valor es propio al objeto (por ejemplo, cualquier elemento colección) y no a lo que representa.

A continuación, vamos a explicar brevemente los estándares ERC20 y ERC721 junto el código base de cada uno de ellos:

A) ERC20

El estándar de token ERC20 fue el primer estándar de token que se utilizó dentro de la plataforma de Ethereum, con el objetivo de tener una primera implementación de un token que fuera reutilizado por otras aplicaciones. Estas aplicaciones van desde wallets hasta Exchanges.

Contrato abstracto ERC-20 (interface)

Lo que vamos a hacer es crear un contrato abstracto que defina las funciones necesarias para nuestros contratos ERC-20. Para aquellos que no sepan que es un contrato abstracto, el concepto es similar a una clase abstracta. Un contrato abstracto es un contrato que no puede ser instanciado, pero puede ser usado como base para otros contratos. Un contrato pasa a ser abstracto cuando al menos una de sus funciones no incluye la implementación, dejando que las clases que hereden de dicho contrato sean los responsables de la implementación.

```
contract ERC20Interface {
    function totalSupply() public view returns (uint);
    function balanceOf(address tokenOwner) public view returns (uint balance);
    function allowance(address tokenOwner, address spender) public view returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Contrato token ERC-20

El siguiente contrato implementa las funciones de nuestro contrato ERC-20 y además añade las variables de estado para el símbolo, nombre, número total de tokens y número de decimales de nuestro token. Cuando se crea el contrato, los tokens son inicialmente asignados al creador del contrato.

```
contract ERC20Token is ERC20Interface {

    using SafeMath for uint; //use our safe math library

    string public symbol;
    string public  name;
    uint8 public decimals;
    uint _totalSupply;

    mapping(address => uint) balances;
    mapping(address => mapping(address => uint)) allowed;

    // ------------------------------------------------------------------------
    // Constructor
    // ------------------------------------------------------------------------
    constructor (
        uint256 _initialAmount,
        string _tokenName,
        uint8 _decimalUnits,
        string _tokenSymbol

    ) public {
        balances[msg.sender] = _initialAmount;
        _totalSupply = _initialAmount;
        name = _tokenName;
        decimals = _decimalUnits;
        symbol = _tokenSymbol;
    }

    // ------------------------------------------------------------------------
    // Fixed Total supply
    // ------------------------------------------------------------------------
    function totalSupply() public view returns (uint) {
        return _totalSupply;
    }

    // ------------------------------------------------------------------------
    // Get the token balance for account `tokenOwner`
    // ------------------------------------------------------------------------
    function balanceOf(address tokenOwner) public view returns (uint balance) {
        return balances[tokenOwner];
    }
```

```
    // ------------------------------------------------------------------------
    // Transfer the balance from token owner's account to `to` account
    // - Owner's account must have sufficient balance to transfer
    // - 0 value transfers are allowed
    // ------------------------------------------------------------------------
    function transfer(address to, uint tokens) public returns (bool success) {
        balances[msg.sender] = balances[msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        emit Transfer(msg.sender, to, tokens);
        return true;
    }

    // ------------------------------------------------------------------------
    // Token owner can approve for `spender` to transferFrom(...) `tokens`
    // from the token owner's account
    //
    // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
    // recommends that there are no checks for the approval double-spend attack
    // as this should be implemented in user interfaces
    // ------------------------------------------------------------------------
    function approve(address spender, uint tokens) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        emit Approval(msg.sender, spender, tokens);
        return true;
    }

    // ------------------------------------------------------------------------
    // Transfer `tokens` from the `from` account to the `to` account
    //
    // The calling account must already have sufficient tokens approve(...)-d
    // for spending from the `from` account and
    // - From account must have sufficient balance to transfer
    // - Spender must have sufficient allowance to transfer
    // - 0 value transfers are allowed
    // ------------------------------------------------------------------------
    function transferFrom(address from, address to, uint tokens) public returns (bool success) {
        balances[from] = balances[from].sub(tokens);
        allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        emit Transfer(from, to, tokens);
        return true;
    }


    // ------------------------------------------------------------------------
    // Returns the amount of tokens approved by the owner that can be
    // transferred to the spender's account
    // ------------------------------------------------------------------------
    function allowance(address tokenOwner, address spender) public view returns (uint remaining) {
        return allowed[tokenOwner][spender];
    }


    // ------------------------------------------------------------------------
    // Fallback function. Don't accept ETH
    // ------------------------------------------------------------------------
    function () public payable {
        revert();
    }
}
```

Tarea: Mirar el código del smart contract y explicar la funcionalidad de cada uno de los métodos del ERC20.

(1) ERC721

Los tokens ERC 721 de la red Ethereum basan su existencia y funcionamiento en la potenciación de la escasez digital para aprovechar el efecto que crean las ediciones limitadas de productos. A diferencia del token ERC 20, su atractivo radica en como su peculiaridad realza su faceta de ser coleccionado. El estándar fue diseñado con el objetivo de crear tokens intercambiables, pero con la particularidad de ser **únicos y no fungibles**. Es decir, cada token es único en toda su existencia y no puede deteriorarse o destruirse. En definitiva y en palabras más sencillas, un token ERC-721 no es más que un token "**coleccionable**". Gracias a ello, podemos definir el valor de un token ERC-721 en función de la rareza y particularidad de sus propiedades. Ello se traduce en que el mismo será más apetecible a sus futuros compradores o **"coleccionistas".**

En el mundo computacional **los Tokens son la representación digital de algún elemento o bien del mundo real o digital.** Algunos ejemplos de bienes no fungibles que pueden ser representados por tokens son:

- **Propiedades físicas** como: Casas, obras de arte, autos.
- **Coleccionables virtuales** como: CryptoKitties, **obras de arte digital** o tarjetas coleccionables.
- **Activos con valor negativo** como: Hipotecas.

Entre las principales diferencias que existen entre los tokens ERC-20 y ERC-721 destacamos:

- Al contrario que en los ERC 20, los tokens ERC-721 son tokens NFT o no fungibles (Not Fungible Token). Esto significa que los tokens ERC-721 no se deterioran o se destruyen como si pasa con los tokens ERC-20.
- Otra diferencia entre los tokens ERC-721 y ERC-20, es que los tokens ERC-721 no son divisibles o fraccionables. Los tokens ERC20 en cambio sí lo son.

Ahora pondremos un ejemplo de un ERC721:

```
pragma solidity ^0.4.20;

    /// @title ERC-721 Non-Fungible Token Standard
    /// @dev See https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md
    ///  Note: the ERC-165 identifier for this interface is 0x80ac58cd
    interface ERC721 /* is ERC165 */ {
        /// @dev This emits when ownership of any NFT changes by any mechanism.
        ///  This event emits when NFTs are created (`from` == 0) and destroyed
        ///  (`to` == 0). Exception: during contract creation, any number of NFTs
        ///  may be created and assigned without emitting Transfer. At the time of
        ///  any transfer, the approved address for that NFT (if any) is reset to none.
        event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);

        /// @dev This emits when the approved address for an NFT is changed or
        ///  reaffirmed. The zero address indicates there is no approved address.
        ///  When a Transfer event emits, this also indicates that the approved
        ///  address for that NFT (if any) is reset to none.
        event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);

        /// @dev This emits when an operator is enabled or disabled for an owner.
        ///  The operator can manage all NFTs of the owner.
        event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);

        /// @notice Count all NFTs assigned to an owner
        /// @dev NFTs assigned to the zero address are considered invalid, and this
        ///  function throws for queries about the zero address.
        /// @param _owner An address for whom to query the balance
        /// @return The number of NFTs owned by `_owner`, possibly zero
        function balanceOf(address _owner) external view returns (uint256);

        /// @notice Find the owner of an NFT
        /// @dev NFTs assigned to zero address are considered invalid, and queries
        ///  about them do throw.
        /// @param _tokenId The identifier for an NFT
        /// @return The address of the owner of the NFT
        function ownerOf(uint256 _tokenId) external view returns (address);


    /// @notice Transfers the ownership of an NFT from one address to another address
    /// @dev Throws unless `msg.sender` is the current owner, an authorized
    ///  operator, or the approved address for this NFT. Throws if `_from` is
    ///  not the current owner. Throws if `_to` is the zero address. Throws if
    ///  `_tokenId` is not a valid NFT. When transfer is complete, this function
    ///  checks if `_to` is a smart contract (code size > 0). If so, it calls
    ///  `onERC721Received` on `_to` and throws if the return value is not
    ///  `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`.
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    /// @param data Additional data with no specified format, sent in call to `_to`
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;

    /// @notice Transfers the ownership of an NFT from one address to another address
    /// @dev This works identically to the other function with an extra data parameter,
    ///  except this function just sets data to ""
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;

    /// @notice Transfer ownership of an NFT -- THE CALLER IS RESPONSIBLE
    ///  TO CONFIRM THAT `_to` IS CAPABLE OF RECEIVING NFTS OR ELSE
    ///  THEY MAY BE PERMANENTLY LOST
    /// @dev Throws unless `msg.sender` is the current owner, an authorized
    ///  operator, or the approved address for this NFT. Throws if `_from` is
    ///  not the current owner. Throws if `_to` is the zero address. Throws if
    ///  `_tokenId` is not a valid NFT.
    /// @param _from The current owner of the NFT
    /// @param _to The new owner
    /// @param _tokenId The NFT to transfer
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;

    /// @notice Set or reaffirm the approved address for an NFT
    /// @dev The zero address indicates there is no approved address.
    /// @dev Throws unless `msg.sender` is the current NFT owner, or an authorized
    ///  operator of the current owner.
    /// @param _approved The new approved NFT controller
    /// @param _tokenId The NFT to approve
    function approve(address _approved, uint256 _tokenId) external payable;
```

```
    /// @notice Enable or disable approval for a third party ("operator") to manage
    ///  all of `msg.sender`'s assets.
    /// @dev Emits the ApprovalForAll event. The contract MUST allow
    ///  multiple operators per owner.
    /// @param _operator Address to add to the set of authorized operators.
    /// @param _approved True if the operator is approved, false to revoke approval
    function setApprovalForAll(address _operator, bool _approved) external;

    /// @notice Get the approved address for a single NFT
    /// @dev Throws if `_tokenId` is not a valid NFT
    /// @param _tokenId The NFT to find the approved address for
    /// @return The approved address for this NFT, or the zero address if there is none
    function getApproved(uint256 _tokenId) external view returns (address);

    /// @notice Query if an address is an authorized operator for another address
    /// @param _owner The address that owns the NFTs
    /// @param _operator The address that acts on behalf of the owner
    /// @return True if `_operator` is an approved operator for `_owner`, false otherwise
    function isApprovedForAll(address _owner, address _operator) external view returns (bool);
}

interface ERC165 {
    /// @notice Query if a contract implements an interface
    /// @param interfaceID The interface identifier, as specified in ERC-165
    /// @dev Interface identification is specified in ERC-165. This function
    ///  uses less than 30,000 gas.
    /// @return `true` if the contract implements `interfaceID` and
    ///  `interfaceID` is not 0xffffffff, `false` otherwise
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}

interface ERC721TokenReceiver {
    /// @notice Handle the receipt of an NFT
    /// @dev The ERC721 smart contract calls this function on the
    /// recipient after a `transfer`. This function MAY throw to revert and reject the transfer. Return
    /// of other than the magic value MUST result in the transaction being reverted.
    /// @notice The contract address is always the message sender.
    /// @param _operator The address which called `safeTransferFrom` function
    /// @param _from The address which previously owned the token
    /// @param _tokenId The NFT identifier which is being transferred
    /// @param _data Additional data with no specified format
    /// @return `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    /// unless throwing
    function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data) external returns(bytes4);
}
```

Tarea: Como habéis podido comprobar, las funciones están vacías. Se recomienda al alumno de este curso intentar rellenarlas.

VIII) 4. 2. PRACTICA FINAL: SMART CONTRACTS PARA CONTROLAR EL ALQUILER DE UNA FLOTA DE DRONES

Para el SPOC, además del contenido referente a Blockchain y los ejercicios propuestos, se ha creado una práctica final, que se muestra a continuación.

Como inicio, y para motivar a los alumnos, se recomienda jugar también al juego cryptozombies: https://cryptozombies.io/es/

Figura 7. Web cryptozombies

Esta actividad está diseñada para poner a prueba todos los conceptos aprendidos durante el curso. El objetivo es que el alumno realice un diseño basado en blockchain. Se requiere que se presenten los smart contract que solucionen este problema.

Opcional: Aunque queda fuera de este curso, si los los alumnos quieren hacer el front de la aplicación se valorara positivamente, aunque si los smart contract presentados no están bien, el front no mejorara la nota de la practica final.

Descripción del problema:

Una empresa ha desarrollado un sistema de fumigación con drones y nos ha solicitado que desarrollemos una solución basada en la blockchain de Rinkeby (testnet de Ethereum) para su uso.

Los drones y las parcelas para fumigar van a estar representados como tokens dentro de la propia red.

Las características propias de los drones son:

- Un identificador único y ascendente, comenzando en 1 y que no puede repetirse.
- La empresa que lo gestiona, que será la única que pueda mandar acciones al dron.
- Altura máxima y mínima de vuelo.

- Autonomía de vuelo (expresada en metros).
- Una lista de pesticidas que puede suministrar. Los pesticidas existentes son cinco y sus nombres son: Pesticida A, Pesticida B, Pesticida C, Pesticida D y Pesticida E.
- Parcela en la que se encuentra actualmente (todos los drones al ser creados están en la parcela1). Al desplazarse para realizar un trabajo se quedan en dicha parcela.
- Coste.

Las operaciones que ofrece un dron son desplazarse a una parcela determinada desde aquella en la que se encuentra situado, a través de las parcelas indicadas por la empresa que lo gestiona, y suministrar el pesticida.

Las características de las parcelas son:

- Un identificador único y ascendente, que comienza en 1 y que no puede repetirse.
- Un propietario.
- La longitud que tiene (expresada en metros).
- Altura máxima y mínima de vuelo permitida.
- Pesticida aceptado, que va a ser uno de la lista de pesticidas descrita anteriormente.

Otras operaciones que debe suministrar la plataforma son:

- Contratar un dron a la empresa para desinfectar una parcela con un pesticida determinado.
- Pago de la operación realizada desde la cuenta del propietario a la de la empresa.

VIII) 4. 3. POSIBLE SOLUCIÓN

Primer se adjunta el esquema del diseño de los smart contract y a continuación el código:

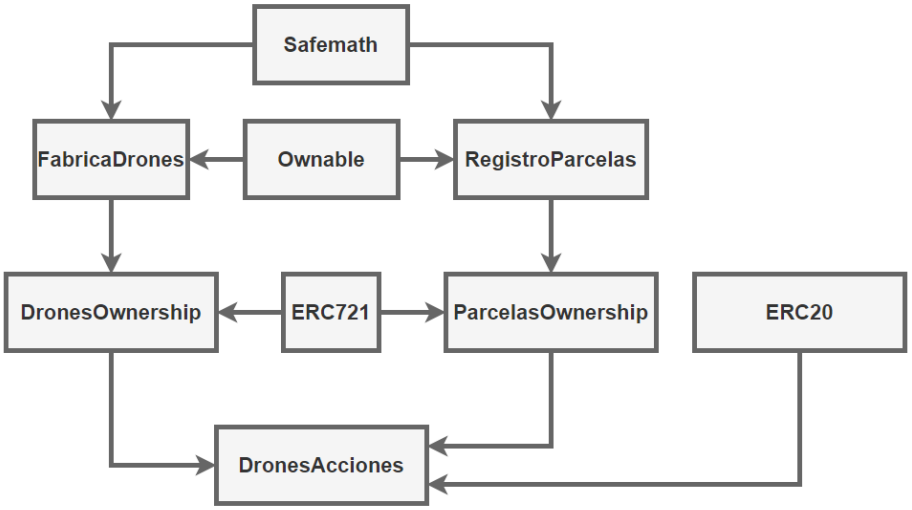

Figura 8. Diseño de Smart contract

Contrato safemath:

```
pragma solidity ^0.5.2;
/**
 * @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;
  }
}
```

Figura 9. Contrato safemath

Contrato ownable:

```
pragma solidity ^0.5.2;

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @return the address of the owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }


  /**
   * @dev Allows the current owner to relinquish control of the contract.
   * @notice Renouncing to ownership will leave the contract without an owner.
   * It will not be possible to call the functions with the `onlyOwner`
   * modifier anymore.
   */
  function renounceOwnership() public onlyOwner {
      emit OwnershipTransferred(_owner, address(0));
      _owner = address(0);
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
      _transferOwnership(newOwner);
  }

  /**
   * @dev Transfers control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function _transferOwnership(address newOwner) internal {
      require(newOwner != address(0));
      emit OwnershipTransferred(_owner, newOwner);
      _owner = newOwner;
  }
```

Figura 1. Contrato ownable

Contrato ERC20:

```
pragma solidity ^0.5.2;


contract ERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approveToken(address spender, uint256 value) external returns (bool);

    function transferFromToken(address from, address to, uint256 value) external returns (bool);

    function totalSupplyToken() external view returns (uint256);

    function balanceOfToken(address who) external view returns (uint256);

    function allowanceToken(address owner, address spender) external view returns (uint256);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

Figura 2. Contrato ERC20

Contrato Fabrica de drones:

```
pragma solidity ^0.5.2;

import "./Ownable.sol";
import "./safemath.sol";

contract FabricaDrones is Ownable {
    using SafeMath for uint256;

    //eventos
    event NewDron(uint256 _dronId,
                  address _empresaOwner,
                  uint256 _alturaMax,
                  uint256 _alturaMin,
                  uint256 _autonomiaVuelto,
                  string _pesticidas,
                  uint256 _parcela,
                  uint256 _coste);
    //variables
    uint256 _numDrones=1; //contador de drones

    struct Dron {
        uint256 dronId; //empieza en 1 y no puede repetirse
        address empresaOwner; // solo esta empresa puede mandarle acciones al dron
        uint256 alturaMax;
        uint256 alturaMin;
        uint256 autonomiaVuelto; // expresado en metros
        //string[5] pesticidas; // hay una lista de pesticidas permitdos desde el pesticida A al E
        string pesticidas; //HAY QUE MIRAR COMO PODER USAR AQUI UN ARRAY PARA LOS PESTICIDAS SIN EL ERROR DE MEMORY/STORAGE
        uint256 parcelaActual; // todos los drones creados empiezan en la parcela 1
        uint256 coste;
    }

    //mappings
    mapping (uint => Dron) _drones;
    mapping (uint => address) public dronToOwner;
    mapping (address => uint) ownerDronCount;
```

```
//funciones
function crearDron(uint256 _alturaMax,
                   uint256 _alturaMin,
                   uint256 _autonomiaVuelto,
                   //string[] memory _pesticidas
                   string memory _pesticidas,
                   uint256 _coste) public  {
    //se crea un nuevo dron con las caracteristicas requeridas
    //y se añade al final del array drones
    _drones[_numDrones].dronId = _numDrones;
    _drones[_numDrones].empresaOwner = msg.sender;
    _drones[_numDrones].alturaMax = _alturaMax;
    _drones[_numDrones].alturaMin = _alturaMin;
    _drones[_numDrones].autonomiaVuelto = _autonomiaVuelto;
    _drones[_numDrones].pesticidas = _pesticidas;
    _drones[_numDrones].parcelaActual = 1;
    _drones[_numDrones].coste = _coste;
    //se lanza un evento para informar al back que se ha creado
    //un nuevo dron en la aplicacion
    dronToOwner[_numDrones] = msg.sender;
    ownerDronCount[msg.sender]++;
    emit NewDron(_numDrones,
                 msg.sender,
                 _alturaMax,
                 _alturaMin,
                 _autonomiaVuelto,
                 _pesticidas,
                 1,
                 _coste);
    //se aumenta el numero de drones para la siguiente vez
    //que se cree un dron nuevo
    _numDrones++;
}


    //getters
    function getIdDron(uint256 dronId) public view returns(uint256){
            return(_drones[dronId].dronId);
    }

    function getDronAltMax(uint256 dronId) public view returns(uint256){
            return(_drones[dronId].alturaMax);
    }
    function getDronAltMin(uint256 dronId) public view returns(uint256){
            return(_drones[dronId].alturaMin);
    }
    function getDronVuelo(uint256 dronId) public view returns(uint256){
            return(_drones[dronId].autonomiaVuelto);
    }
    function getDronPesticida(uint256 dronId) public view returns(string memory){
            return(_drones[dronId].pesticidas);
    }
    function getDronParcelaActual(uint256 dronId) public view returns(uint256){
            return(_drones[dronId].parcelaActual);
    }

    function getDronCoste(uint256 dronId) public view returns(uint256){
            return(_drones[dronId].coste);
    }

}
```

Figura 3. Contrato Fabrica de drones

Contrato Registroparcelas:

```
pragma solidity ^0.5.2;

import "./Ownable.sol";
import "./safemath.sol";

contract RegistroParcelas is Ownable {
    using SafeMath for uint256;
    //eventos
    event NewParcela(uint256 id,
                     address propietario,
                     uint256 longitud,
                     uint256 alturaMax,
                     uint256 alturaMin,
                     string pesticidaAceptado);
    //variables
    uint256 _numParcelas=2; //contador de parcelas, la primera se crea con el constructor
    //ya que los drones siempre van a aparecer en esa parcela

    struct Parcela {
        uint256 parcelaId; //empieza en 1 y no puede repetirse
        address propietario; // propietario de la parcela
        uint256 longitud; // expresado en metros
        uint256 alturaMax;
        uint256 alturaMin;
        string pesticidaAceptado; // solo hay uno aceptado
    }


    //mappings
    mapping (uint256 => Parcela) _parcelas;
    mapping (uint => address) public parcelaToOwner;
    mapping (address => uint) ownerParcelaCount;
    //Constructor del contrato
    constructor() public {
        _parcelas[_numParcelas].parcelaId = 1;
        _parcelas[_numParcelas].propietario = msg.sender;
        _parcelas[_numParcelas].longitud=1;
        _parcelas[_numParcelas].alturaMax=1;
        _parcelas[_numParcelas].alturaMin=1;
    }

 //funciones
 function crearParcela(uint256 _longitud,
                       uint256 _alturaMax,
                       uint256 _alturaMin,
                       string memory _pesticidaAceptado) public {
     //se crea un nueva parcela con las caracteristicas requeridas
     //y se añade al final del array parcelas
     _parcelas[_numParcelas].parcelaId = _numParcelas;
     _parcelas[_numParcelas].propietario = msg.sender;
     _parcelas[_numParcelas].longitud = _longitud;
     _parcelas[_numParcelas].alturaMax = _alturaMax;
     _parcelas[_numParcelas].alturaMin = _alturaMin;
     _parcelas[_numParcelas].pesticidaAceptado = _pesticidaAceptado;

     //se lanza un evento para informar al back que se ha creado
     //una nueva parcela en la aplicacion
     parcelaToOwner[_numParcelas] = msg.sender;
     ownerParcelaCount[msg.sender]++;
     emit NewParcela( _numParcelas,
                      msg.sender,
                      _longitud,
                      _alturaMax,
                      _alturaMin,
                      _pesticidaAceptado);
     _numParcelas++;
 }
```

Figura 1. Contrato Registroparcelas

Contrato ERC721:

```
pragma solidity ^0.5.2;

contract ERC721 {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
  function balanceOf(address _owner) public view returns (uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns (address _owner);
  function transfer(address _to, uint256 _tokenId) public;
  function approve(address _to, uint256 _tokenId) public;
  function takeOwnership(uint256 _tokenId) public;
}
```

Figura 2. Contrato ERC721

Contrato DronesOwnership:

```
pragma solidity ^0.5.2;

import "./FabricaDrones.sol";
import "./ERC721.sol";
import "./safemath.sol";


contract DronesOwnership is FabricaDrones,ERC721 {
    using SafeMath for uint256;

    //modifiers
    modifier onlyOwnerOf(uint256 _dronId) {
      require(msg.sender == dronToOwner[_dronId]);
      _;
    }
    //mappings
    mapping (uint => address) dronApprovals;
    //esta funcion sirve para comprobar el numero de drones que tiene
    //cada una de las empresas
    function balanceOf(address _owner) public view returns (uint256 _balance){
      return ownerDronCount[_owner];
    }
    //funcion auxiliar para transferir la propiedad de los drones
    //se crea privada para que solo se puede usar desde fuera si
    //se llama desde una funcion publica pero con control de propiedad
    function _transfer(address _from, address _to, uint256 _dronId) private {
      ownerDronCount[_to] = ownerDronCount[_to].add(1);
      ownerDronCount[msg.sender] = ownerDronCount[msg.sender].sub(1);
      dronToOwner[_dronId] = _to;
      emit Transfer(_from, _to, _dronId);
    }
    //esta funcion se llama desde fuera del contrato por el dueño del dron
    //para transferir la propiedad del dron
    function transfer(address _to, uint256 _dronId) public onlyOwnerOf(_dronId) {
      _transfer(msg.sender, _to, _dronId);
    }
    //con esta funcion se sabe cual es el dueño del dronId que se de a la funcion
    function ownerOf(uint256 _dronId) public view returns (address _owner){
      return dronToOwner[_dronId];
    }
    //
    function approve(address _to, uint256 _dronId) public{
      dronApprovals[_dronId] = _to;
      //se lanza este evento al fron para informar de la aprobacion
      emit Approval(msg.sender, _to, _dronId);
    }
    //función para hacerse dueño de un dron, es necesario tener el aprobal para poder hacerlo
    function takeOwnership(uint256 _dronId) public{
      //se comprueba que el dron aprobal es el que esta invocando esta misma funcion
      require(dronApprovals[_dronId] == msg.sender);
      address owner = ownerOf(_dronId);
      //se transfiere la propiedad al que invoca la funcion
      _transfer(owner, msg.sender, _dronId);
    }

}
```

Figura 3. Contrato DronesOwnership

Contrato ParcelasOwnership:

```
pragma solidity ^0.5.2;

import "./RegistroParcelas.sol";
import "./ERC721.sol";
import "./safemath.sol";


contract ParcelasOwnership is RegistroParcelas,ERC721 {
    using SafeMath for uint256;

    //modifiers
    modifier onlyOwnerOf(uint256 _parcelaId) {
        require(msg.sender == parcelaToOwner[_parcelaId]);
        _;
    }
    //mappings
    mapping (uint => address) parcelaApprovals;
    //esta funcion sirve para comprobar el numero de parcelas que tiene
    //cada una de las empresas
    function balanceOf(address _owner) public view returns (uint256 _balance){
        return ownerParcelaCount[_owner];
    }
    //funcion auxiliar para transferir la propiedad de los parcelas
    //se crea privada para que solo se puede usar desde fuera si
    //se llama desde una funcion publica pero con control de propiedad
    function _transfer(address _from, address _to, uint256 _parcelaId) private {
        ownerParcelaCount[_to] = ownerParcelaCount[_to].add(1);
        ownerParcelaCount[msg.sender] = ownerParcelaCount[msg.sender].sub(1);
        parcelaToOwner[_parcelaId] = _to;
        emit Transfer(_from, _to, _parcelaId);
    }
    //esta funcion se llama desde fuera del contrato por el dueño de la parcela
    //para transferir la propiedad de la parcela
    function transfer(address _to, uint256 _parcelaId) public onlyOwnerOf(_parcelaId) {
        _transfer(msg.sender, _to, _parcelaId);
    }
    //con esta funcion se sabe cual es el dueño del parcelaId que se de a la funcion
    function ownerOf(uint256 _parcelaId) public view returns (address _owner){
        return parcelaToOwner[_parcelaId];
    }
    //
    function approve(address _to, uint256 _parcelaId) public{
        parcelaApprovals[_parcelaId] = _to;
        //se lanza este evento al front para informar de la aprobacion
        emit Approval(msg.sender, _to, _parcelaId);
    }
    //función para hacerse dueño de una parcela, es necesario tener el aprobal para poder hacerlo
    function takeOwnership(uint256 _parcelaId) public{
        //se comprueba que la parcela aprobal es el que esta invocando esta misma funcion
        require(parcelaApprovals[_parcelaId] == msg.sender);
        address owner = ownerOf(_parcelaId);
        //se transfiere la propiedad al que invoca la funcion
        _transfer(owner, msg.sender, _parcelaId);
    }

}
```

Figura 4. Contrato ParcelasOwnership

Contrato DronesAcciones:

```
pragma solidity ^0.5.2;

import "./DronesOwnership.sol";
import "./ParcelasOwnership.sol";
import "browser/ERC20.sol";
import "./safemath.sol";


contract DronesAcciones is DronesOwnership,ParcelasOwnership {
    using SafeMath for uint256;
    uint256 private tokenTotalSupply;

    //eventos
    event DronNuevaParcela(uint256 _dronId, uint256 _OldParcelaId, uint256 _NewParcelaId);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed _owner, address indexed _spender, uint256 value);

    //mappings
    mapping (address => uint256) private balances;
    mapping (address => mapping(address => uint256)) public approved;
    address _owner;

    //funcion contratar dron con un pesticida determinado

    function contrataDron(uint256 _dronId, uint256 _parcelaId) public {
      //precio del dron
      uint256 _value = _drones[_dronId].coste;
      //validar los requisitos
      require(_drones[_dronId].alturaMax <= _parcelas[_parcelaId].alturaMax);
      require(_drones[_dronId].alturaMin >= _parcelas[_parcelaId].alturaMin);
      require(_drones[_dronId].autonomiaVuelto <= _parcelas[_parcelaId].longitud);
      //ejecutar la funcion para mover el dron a la parcela indicada
      bool pagado = transferToken(_drones[_dronId].empresaOwner,_value);
      require(pagado == true);
      moverDron(_dronId, _parcelaId);
    }

    //funcion mover dron
    function moverDron(uint256 _dronId, uint256 _parcelaId) public{
      uint256 OldParcelaId = _drones[_dronId].parcelaActual;
      uint256 NewParcelaId = _parcelaId;
      _drones[_dronId].parcelaActual = NewParcelaId;

      emit DronNuevaParcela(_dronId, OldParcelaId, NewParcelaId);
    }
```

```
    //funciones ERC20

    //Constructor del contrato
    constructor(uint256 _totalSupply) public {
        tokenTotalSupply = _totalSupply;
        _owner = msg.sender;
        //Msg.sender or owner
        balances[msg.sender] = _totalSupply;
    }
    function transferToken(address to, uint256 value) public returns (bool) {
        require(balances[msg.sender]>=value);
        _transferTokens(msg.sender, to, value);
        return true;
    }

    function _transferTokens(address _from, address _to, uint256 _value) internal returns (bool) {
        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(_from, _to, _value);
        return true;
    }

    function approveToken(address spender, uint256 value) external returns (bool) {
        require(balances[msg.sender] >= value);
        _approveToken(spender,msg.sender, value);
        return true;
    }

    function _approveToken(address spender, address holder, uint256 value) internal {
        approved[holder][spender] = value;
    }

    function transferFromToken(address _from, address _to, uint256 _value) external returns (bool) {
        require((approved[_from][_to] >= _value) && (balances[_from]>=_value));
        approved[_from][_to] = approved[_from][_to].sub(_value);
        _transferTokens(_from,_to,_value);
        return true;
    }

    function totalSupplyToken() external view returns (uint256) {
        return tokenTotalSupply;
    }

    function balanceOfToken(address who) external view returns (uint256) {
        return balances[who];
    }

    function allowanceToken(address _owner, address spender) external view returns (uint256) {
        return 0;
    }


}
```

Figura 1. Contrato DronesAcciones

IX. CONCLUSIONES

A lo largo del proyecto se ha realizado un análisis tanto de las posibilidades de creación de cursos tipo SPOC, como de la tecnología necesaria para el uso de Blockchain, haciendo hincapié en las mejores condiciones para la enseñanza de ésta última de manera básica.

En el proyecto, se ha realizado una propuesta para la introducción de esta metodología y se ha creado el contenido esencial para su enseñanza a través de un curso.

Los resultados experimentales confirman que el modelo de curso tipo SPOC es una opción óptima para poder ofrecer a los estudiantes una visión general de una tecnología como Blockchain.

La contribución más relevante de este trabajo es la combinación de tecnologías emergentes dentro de la computación distribuida y la seguridad como es Blockchain con metodologías novedosas en e-learning.

En cuanto a la efectividad del curso se puede decir que en general es buena, aunque es cierto que existen limitaciones a tener en cuenta, como el hecho del uso de redes de blockchain específicas o el tiempo de tutorización específica que necesitan los alumnos para aprender una tecnología de este tipo.

IX) 1. LÍNEAS DE TRABAJO FUTURAS

La implantación de este proyecto y su posterior análisis de resultados puede suponer la base sobre la que crear una mejora en la metodología y materiales actualmente creados, pues como sabemos, la tecnología avanza a pasos agigantados y este tipo de recursos evolucionan para poder ofrecer una adaptación de la formación a las necesidades de los estudiantes interesados en el ámbito de la seguridad Informática.

Una posible futura versión sería ampliar los contenidos con el uso de otras redes blockchain. Además, podría adaptarse el uso del curso a investigadores y estudiantes de otras universidades que también utilicen este tipo de tecnología.

Otra mejora, podría ser la posibilidad de exportar y analizar todos los resultados obtenidos por parte de los estudiantes (tiempo y calidad de las tareas realizadas, uso de recursos, etc.)

Con la implementación de la aplicación y la experiencia de su funcionamiento en este proyecto se comprueba la validez de la utilización de iniciativas efectivas de aprendizaje invertido que incrementen el acervo audiovisual de materiales docentes. Como consecuencia del proceso de diseño del SPOC se establecen pruebas específicas que nos permitan establecer la validez de los resultados obtenidos en comparación con métodos tradicionales.

IX) 2. GRADO DE CUMPLIMIENTO

De acuerdo con la metodología y el plan de trabajo llevados a cabo, se tuvieron en cuenta dos tipos de indicadores para las medidas aplicadas para la evaluación de los resultados y su incidencia en la mejora del aprendizaje de los estudiantes:

- Indicadores del proceso clave: enseñanza/aprendizaje del alumno: indicadores sobre los procesos de apoyo, esto es, las etapas definidas para llevar a cabo el proyecto (revisión de literatura y análisis del estado del arte de las tecnologías empleadas, selección de técnicas y tecnologías a emplear, implementación de la aplicación en el dispositivo seleccionado).
- Revisiones del diseño y baterías de pruebas del modelo en las fases del proyecto tanto individuales como de forma integrada.

Durante toda la duración del proyecto se ha llevado a cabo una coordinación entre los profesores e investigadores del equipo. El equipo de investigación está especialmente satisfecho con la labor de innovación puesta en marcha del modelo y con las colaboraciones de investigación surgidas entre los integrantes.

Debido a la situación vivida durante el desarrollo del proyecto (estado de alarma, confinamiento y anulación de todas las actividades presenciales en la Universidad debido a la pandemia por el SARS-CoV-2 (COVID-19)), algunos de los objetivos iniciales como grabaciones de material específicas y reuniones de coordinación tuvieron que ser sustituidas por alternativas a distancia. En todo caso, no supuso ningún problema para el desarrollo del proyecto, que ha podido finalizarse en tiempo y con vistas a futuras ampliaciones.

De hecho, debido a los buenos resultados obtenidos, se asegura una línea de continuación que merece la pena ser explorada en sucesivos proyectos con una extrapolación del curso a diferentes asignaturas y ámbitos.

X. MEMORIA ECONÓMICA

En el proyecto de innovación presentado se solicitaban 626,54 Euros para licencias y material. En este caso la resolución solo tuvo en cuenta la consideración de la compra de licencias con una cantidad de 540 euros, que ha sido utilizada para el uso de redes de blockchain escalables.

XI. AGRADECIMIENTOS

Queremos agradecer a la institución, la Universidad de Salamanca, su esfuerzo por mantener este tipo de proyectos de innovación. También agradecer la colaboración desinteresada de aquellos alumnos e investigadores que se han prestado a colaborar en este proyecto.

XII. REFERENCIAS

1. Mar López, Juanita Pedraza, Javier Carbó, José M. Molina (2014). The awareness of Privacy issues in Ambient Intelligence. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
2. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014, July). A particle dyeing approach for track continuity for the SMC-PHD filter. In *17th International Conference on Information Fusion (FUSION)* (pp. 1-8). IEEE.
3. Bullon, Juan, et al. "Manufacturing processes in the textile industry. Expert Systems for fabrics production." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 6.4 (2017): 15-23.
4. Fdez-Riverola, F., Iglesias, E. L., Díaz, F., Méndez, J. R., & Corchado, J. M. (2007). Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications*, *33*(1), 36-48.
5. Souza de Castro, Lucas Fernando, Gleifer Vaz Alves, and André Pinz Borges. "Using trust degree for agents in order to assign spots in a Smart Parking." (2017).
6. Moung, Ervin. "A Comparison of the YCBCR Color Space with Gray Scale for Face Recognition for Surveillance Applications." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* [Online], 6.4 (2017): 25-33.
7. Morente-Molinera, J. A., Kou, G., González-Crespo, R., Corchado, J. M., & Herrera-Viedma, E. (2017). Solving multi-criteria group decision making problems under

environments with a high number of alternatives using fuzzy ontologies and multi-granular linguistic modelling methods. *Knowledge-Based Systems*, *137*, 54-64.
8. Kethareswaran, V., and C. SANKAR RAM. "An Indian Perspective on the adverse impact of Internet of Things (IoT)." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 6.4 (2017): 35-40.
9. Li, T., Sun, S., Bolić, M., & Corchado, J. M. (2016). Algorithm design for parallel implementation of the SMC-PHD filter. *Signal Processing*, *119*, 115-127.
10. Cunha, Rafhael, Cleo Billa, and Diana Adamatti. "Development of a Graphical Tool to integrate the Prometheus AEOlus methodology and Jason Platform." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 6.2 (2017): 57-70.
11. Coria, J. A. G., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. *Expert Systems with Applications*, *41*(4), 1189-1205.
12. Ming Fei Siyau, Tiancheng Li, Jonathan Loo (2014). A Novel Pilot Expansion Approach for MIMO Channel Estimation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
13. Tapia, D. I., Fraile, J. A., Rodríguez, S., Alonso, R. S., & Corchado, J. M. (2013). Integrating hardware agents into an enhanced multi-agent architecture for Ambient Intelligence systems. *Information Sciences*, *222*, 47-65.
14. Corchado, J. M., Pavón, J., Corchado, E. S., & Castillo, L. F. (2004, August). Development of CBR-BDI agents: a tourist guide application. In *European Conference on Case-based Reasoning* (pp. 547-559). Springer, Berlin, Heidelberg.
15. Lima, A. C. E., de Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. *Applied Mathematics and Computation*, *270*, 756-767.
16. Fdez-Riverola, F., & Corchado, J. M. (2004). Fsfrt: Forecasting system for red tides. *Applied Intelligence*, *21*(3), 251-264.
17. Fdez-Riverola, F., Iglesias, E. L., Díaz, F., Méndez, J. R., & Corchado, J. M. (2007). SpamHunting: An instance-based reasoning system for spam labelling and filtering. *Decision Support Systems*, *43*(3), 722-736.
18. Casado-Vara, R., Martin-del Rey, A., Affes, S., Prieto, J., & Corchado, J. M. (2020). IoT network slicing on virtual layers of homogeneous data for improved algorithm operation in smart buildings. *Future Generation Computer Systems*, *102*, 965-977.
19. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. *Information Sciences*, *180*(10), 2029-2043.
20. Casado-Vara, R., Prieto, J., De la Prieta, F., & Corchado, J. M. (2018). How blockchain improves the supply chain: case study alimentary supply chain. *Procedia computer science*, *134*, 393-398.
21. Corchado, J. M., & Aiken, J. (2002). Hybrid artificial intelligence methods in oceanographic forecast models. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *32*(4), 307-313.
22. González-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy optimization using a case-based reasoning strategy. *Sensors*, *18*(3), 865.
23. Corchado, J. M., Corchado, E. S., Aiken, J., Fyfe, C., Fernandez, F., & Gonzalez, M. (2003, June). Maximum likelihood hebbian learning based retrieval method for cbr systems. In *International Conference on Case-Based Reasoning* (pp. 107-121). Springer, Berlin, Heidelberg.
24. Ribeiro, Catarina, et al. "Customized normalization clustering meth-odology for consumers with heterogeneous characteristics." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 7.2 (2018): 53-69.
25. Guillén, J. H., del Rey, A. M., & Casado-Vara, R. (2019). Security Countermeasures of a SCIRAS Model for Advanced Malware Propagation. *IEEE Access*, *7*, 135472-135478.
26. Corchado, J. M., & Lees, B. (2001). A hybrid case-based model for forecasting. *Applied Artificial Intelligence*, *15*(2), 105-127.

27. Pawel Pawlewski, Kamila Kluska (2017). Modeling and simulation of bus assembling process using DES/ABS approach. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
28. Ricardo Azambuja Silveira, Rafaela Lunardi Comarella, Ronaldo Lima Rocha Campos, Jonas Vian, Fernando De La Prieta (2015). Learning Objects Recommendation System: Issues and Approaches for Retrieving, Indexing and Recomend Learning Objects. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 4
29. Fernández-Riverola, F., Diaz, F., & Corchado, J. M. (2006). Reducing the memory size of a fuzzy case-based reasoning system applying rough set techniques. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *37*(1), 138-146.
30. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. *International Journal of Ambient Computing and Intelligence (IJACI)*, *1*(1), 15-26.
31. Javier Gómez, Xavier Alamán, Germán Montoro, Juan C. Torrado, Adalberto Plaza (2013). AmICog – mobile technologies to assist people with cognitive disabilities in the workplace. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
32. Corchado, J. M., & Fyfe, C. (1999). Unsupervised neural method for temperature forecasting. *Artificial Intelligence in Engineering*, *13*(4), 351-357.
33. Mendez, J. R., Fdez-Riverola, F., Diaz, F., Iglesias, E. L., & Corchado, J. M. (2006, July). A comparative performance study of feature selection methods for the anti-spam filtering domain. In *Industrial Conference on Data Mining* (pp. 106-120). Springer, Berlin, Heidelberg.
34. Serna, Francisco José Aranda, and Javier Belda Iniesta. "The delimitation of freedom of speech on the Internet: the confrontation of rights and digital censorship." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 7.1 (2018): 5-12.
35. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. *Expert Systems with Applications*, *36*(4), 8239-8246.
36. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. *Wireless Communications and Mobile Computing*, *2018*.
37. Glez-Bedia, M., Corchado, J. M., Corchado, E. S., & Fyfe, C. (2002). Analytical model for constructing deliberative agents. *Engineering Intelligent Systems for Electrical Engineering and Communications*, *10*(3), 173-185.
38. Glez-Bedia, M., Corchado, J. M., Corchado, E. S., & Fyfe, C. (2002). Analytical model for constructing deliberative agents. *Engineering Intelligent Systems for Electrical Engineering and Communications*, *10*(3), 173-185.
39. Fyfe, C., & Corchado, J. M. (2001). Automating the construction of CBR Systems using Kernel Methods. *International Journal of Intelligent Systems*, *16*(4), 571-586.
40. Choon, Y. W., Mohamad, M. S., Safaai Deris, R. M., Illias, C. K. C., Chai, L. E., Omatu, S., & Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. *PloS one*, *9*(7).
41. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014, July). A particle dyeing approach for track continuity for the SMC-PHD filter. In *17th International Conference on Information Fusion (FUSION)* (pp. 1-8). IEEE.
42. Martín del Rey, A., Casado Vara, R., & Hernández Serrano, D. (2019). Reversibility of Symmetric Linear Cellular Automata with Radius r= 3. *Mathematics*, *7*(9), 816.
43. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. *IEEE Access*, *7*, 11972-11984.

44. Vera, Jefferson Stewart Espinosa. "Human rights in the ethical protection of youth in social networks-the case of Colombia and Peru." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 6.4 (2017): 71-79.
45. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto, J., & Corchado, J. M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. *Information Fusion*, *49*, 227-239.
46. Farias, Giovani Parente, et al. "Predicting Plan Failure by Monitoring Action Sequences and Duration." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 6.4 (2017): 55-69.
47. Van Haare Heijmeijer, Alexis, and Gleifer Vaz Alves. "Development of a Middleware between SUMO simulation tool and JaCaMo framework." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 7.2: 5-15.
48. Bogdan Okresa Durik. (2017) Organisational Metamodel for Large-Scale Multi-Agent Systems: First Steps Towards Modelling Organisation Dynamics. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
49. Glaeser, Stefania da Silveira, et al. "Modeling of Circadian Rhythm under influence of Pain: an approach based on Multi-agent Simulation." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 7.2 (2018): 17-25.
50. Srivastava, Varun, and Ravindra Purwar. "An extension of local mesh peak valley edge based feature descriptor for image retrieval in bio-medical images." *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 7.1 (2018): 77-89.
51. Ricardo Silveira, Guilherme Klein Da Silva Bitencourt, Thiago Ângelo Gelaim, Jerusa Marchi, Fernando De La Prieta (2015). Towards a Model of Open and Reliable Cognitive Multiagent Systems: Dealing with Trust and Emotions. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 3
52. Carolina González, Juan Carlos Burguillo, Martín Llamas, Rosalía Laza (2013). Designing Intelligent Tutoring Systems: A Personalization Strategy using Case-Based Reasoning and Multi-Agent Systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 1
53. Daniel Ayala, Juan C. Roldán, David Ruiz, Fernando O. Gallego (2015). An approach for discovering keywords from Spanish tweets using Wikipedia. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 2
54. Ángel Martín del Rey, F. K. Batista, A. Queiruga Dios (2017). Malware propagation in Wireless Sensor Networks: global models vs Individual-based models. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
55. Vanessa N. Cooper, Hisham M. Haddad, Hossain Shahriar (2014). Android Malware Detection Using Kullback-Leibler Divergence. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
56. Saadi Bin Ahmad Kamaruddin, Nor Azura Md Ghanib, Choong-Yeun Liong, Abdul Aziz Jemain (2012). Firearm Classification using Neural Networks on Ring of Firing Pin Impression Images. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 1, n. 3
57. José Antonio Castellanos Garzón, Juan Ramos González (2015). A Gene Selection Approach based on Clustering for Classification Tasks in Colon Cancer. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 3

58. Miki Ueno, Naoki Mori, Keinosuke Matsumoto (2014). Picture models for 2-scene comics creating system. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2
59. Giovani Parente Farias, Ramon Fraga Pereira, Lucas W. Hilgert, Felipe Meneguzzi, Renata Vieira, Rafael H. Bordini (2017). Predicting Plan Failure by Monitoring Action Sequences and Duration. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
60. Vergara, D.; Extremera, J.; Rubio, M.P.; Dávila, L.P. Meaningful Learning Through Virtual Reality Learning Environments: A Case Study in Materials Engineering. *Appl. Sci.* **2019**, *9*, 4625.