DAJ Auction System

Fundamentos de Blockchains, 2020

D. Montes
IIMAS
UNAM
CDMX, México
niner90@ciencias.unam.mx

A. Davila
IIMAS
UNAM
CDMX, México
photographic_ren@comunidad.unam.mx

J. Solano
IIMAS
UNAM
CDMX, México
jrg.a.solano@gmail.com

Abstract

Este documento presenta una descripción del proyecto final de la asignatura Fundamentos de Blockchains, la aplicación descentralizada DAJ Auction System, el cual utiliza tecnología blockchain, en conjunto con las herramientas Solidity, Truffle Suite, Ganache, web3.js, Metamask, React, e IPFS como repositorio descentralizado de almacenamiento de archivos.

Index Terms

dApp, blockchains, auction

I. Introducción

Muchas de las actividades que existen desde la antigüedad se han ido modernizando con los avances tecnológicos, particularmente con software, para disminuir espacios físicos, costos de producción y ofrecer difusión masiva global con la finalidad de simplificar tareas que realizan las distintas sociedades en el mundo.

Aunque, en varias ocasiones se ha visto que los sistemas de software presentan situaciones polémicas similares a sus raíces analógicas, especialmente las injusticias, este contexto motiva que tanto estudiantes como investigadores, se encuentren interesados en proponer mecanismos o técnicas para mitigar esas situaciones en medida de lo posible.

Tomando como base lo anterior, el presente reporte tiene la intención de exponer una aplicación descentralizada en línea, que posee tecnología basada en el concepto de Blockchain, siguiendo el precepto de buscar mitigar situaciones que representan un problema en los esquemas de trabajo actuales.

Durante este reporte, se describirá la aplicación descentralizada desarrollada, las herramientas que utiliza, su composición, funcionamiento los alcances obtenidos y las situaciones de riesgo que logró manejar.

II. ¿Qué es DAJ?

A. Motivaciones y Descripción

Desde hace varios siglos las subastas se han utilizado como una manera de hacer intercambio de bienes, sin embargo, existen diversos elementos que pueden incidir sobre si ese intercambio se realiza de forma justa, por ejemplo, si realmente el mejor postor fue quien ganó la subasta, además, que se requiere invertir en infraestructura para el desarrollo de la subasta, inversión que dependiendo el contexto o los bienes puede variar en costo.

Actualmente, existen subastas de tipo electrónicas las cuales realizan el mismo tipo de intercambio, representando un costo menor para participantes y organizadores, no obstante, presentan detalles similares respecto a si el ejercicio se realizó de manera justa, en conjunto con los problemas de seguridad que enfrentan los sistemas de software si en su implementación no cuentan con protocolos o técnicas criptográficas.

Desde comienzos de la década pasada, se han hecho populares los sistemas basados en Blockchain, concebidos con filosofía OpenSource ofreciendo niveles de seguridad y justicia para la realización de transacciones.

Entonces, lo anterior establece la motivación para desarrollar una base que funge como alternativa para la realización de subastas electrónicas en línea, **DAJ Auction System**.

Esta aplicación descentralizada propuesto en el presente proyecto, utiliza tecnología Blockchain, para realizar transacciones de forma segura y justa, en él se pueden realizar las operaciones de crear una subasta, participar en una subasta haciendo pujas sobre bienes que se encuentren disponibles, de manera ágil, eficaz y segura.

B. Herramientas utilizadas

Para la implementación de la aplicación descentralizada, se hizo uso de las siguientes herramientas de desarrollo:

Fig 1. Herramientas.

Solidity

Es un lenguaje de alto nivel orientado a objetos para implementar contratos inteligentes, influenciado por los lenguajes C++, Python y JavaScript, está diseñado para apuntar a la máquina virtual de Ethereum (EVM), es estáticamente tipado, admite herencia, bibliotecas y tipos complejos definidos por el usuario, en él se pueden crear contratos para usos como votación, crowdfunding, subastas a ciegas y wallets con firma múltiple. [3]

web3.js

Es una colección de bibliotecas que permiten la interacción con un nodo ethereum local o remoto utilizando HTTP, IPC o WebSocket. [4]

Truffle

Es un entorno de desarrollo de clase mundial, un marco de prueba y una cartera de activos para blockchains utilizando la máquina virtual de Ethereum (EVM), con el objetivo de facilitar la vida para el desarrollador, es automatizado, personalizable, programable, administrable e interactivo. [1]

Ganache

Es una Blockchain personal para el desarrollo ágil de aplicaciones distribuidas de Ehtereum y Corda. Se puede usar en todo el ciclo de vida del desarrollo; permite desarrollar, implementar y probar las dApps *(Apps descentralizadas)* en un entorno seguro y determinista. [2]

MetaMask

Funge como una bóveda segura de llaves, ofrece inicio de sesión seguro y un wallet de tokens, ofrece varios niveles de seguridad, y ofrece intreracción con aplicaciones que requieran diversos tipos de wallet tipo Ethereum. [7]

React

Es una biblioteca JavaScript para crear interfaces de usuario, actualiza y renderiza de manera eficiente los componentes correctos cuando haya cambio en los datos. [5]

IPFS

Es un sistema de archivos distribuidos punto a punto que busca conectar todos los dispositivos digitales con el mismo sistema de archivos, proporciona un modelo de almacenamiento en bloque de alto rendimiento y contenido direccionado. [6]

III. Funcionalidad de DAJ

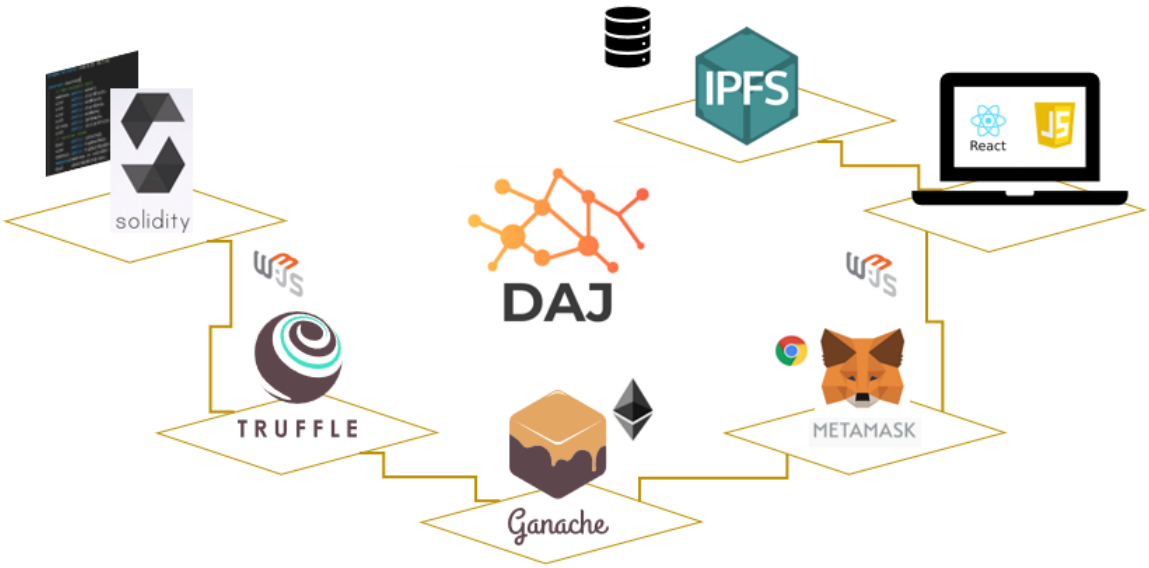

Fig 2. Arquitectura del Sistema.

DAJ Auction System tiene un diseño orientado a Ethereum. En la figura 2 se muestra un diagrama de la arquitectura general del sistema.

Parte con el desarrollo de los Smart Contracts implementados en Solidity, en este caso se desarrollaron dos contratos. Auction maneja toda la lógica de la subasta y el AuctionFactory permite controlar todas las subastas que se conciban en el sistema.

Para migrar los Smart Contracts en Ethereum se hace uso de la Suite Truffle. Truffle es un framework que permite, entre otras cosas, compilar los Smart Contracts, desplegarlos en la red y consultar características de los contratos desplegados en red (dirección de despliegue, llamada de métodos de los contratos, consultar el estado de la red, etc.).

En los archivos de configuración de la aplicación se indica a Truffle donde desplegar los Smart Contracts. Los contratos creados en este sistema se desplegaron en la herramienta Ganache, una blockchain Ethereum personal. Ganache cuenta con interfaz gráfica que permite monitorear transacciones, bloques, cuentas, eventos y contratos.

La interacción con Ethereum se hace mediante un administrador de cuentas. Para esta aplicación se utiliza MetaMask. Cuando se realiza una transacción que consume ether en la blockchain, MetaMask pide autorización al usuario y envía la petición.

React genera las vistas para el usuario y facilita las transacciones con MetaMask realizando peticiones a la blockchain.

La lógica del negocio se maneja en la blockchain de Ganache, pero, las imágenes se almacenan utilizando IPFS. Para esta aplicación descentralizada utilizamos el servidor ipfs.io para almacenar imágenes.

Finalmente, se utiliza web3.js es la API Ethereum JavaScript, para interactuar con redes Ethereum utilizando HTTP o IPC. Esta biblioteca permite a Truffle interactuar con Ganache. Así mismo, dentro en la aplicación front-end también se utiliza para realizar las transacciones.

Es necesario destacar que se requiere contar con el plugin de MetaMask en el navegador de internet, ya que mediante esa herramienta es posible poder realizar las operaciones que se describen en los siguientes apartados.

A. Crear una Subasta

Fig 3. Vistas para Crear.

Una vez que ya contamos con los requisitos de tener una cuenta participante, y tener acceso al servidor donde se encuentra alojada la aplicación, podemos realizar la siguiente secuencia de pasos:

- Se solicita el monto mínimo inicial, el rango de tiempo en que la subasta estará activa y la imagen del lote.
- Se le informa al creador que dicha operación solicita una comisión, la cual requiere su consentimiento.
- Se muestra el estado de su wallet para ver si es posible realizar esa transacción, y autorizar el movimiento.
- Muestra que se está procesando la transacción en la blockchain para concretar la operación.
- Una vez que ya fue minado se muestra el estado en el que se encuentra su subasta en la lista.

B. Participar en Subastas

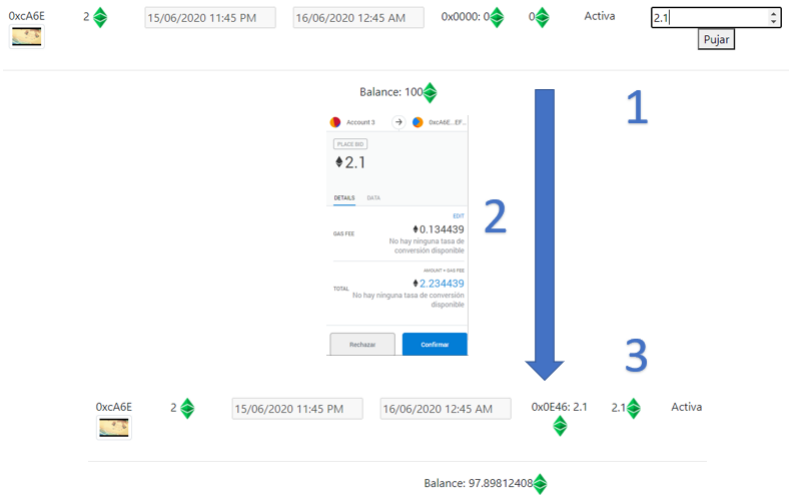

Fig 4. Vistas para Participar.

Para poder participar emitiendo pujas, se realiza la siguiente secuencia de pasos:

- Se ingresa el monto para pujar, este debe ser estrictamente mayor al precio actual de lo contrario la aplicación no va a permitir la puja.
- Se muestra el estado de su wallet para ver si es posible realizar esa transacción, y autorizar el movimiento.
- Se ve reflejada su puja en el lote, además de su nuevo balance disponible en su wallet.

C. Ganador de una Subasta

Fig 5. Vista para el Ganador.

El ganador está indicado por el usuario con el monto de la puja más alta, superando el precio actual, en la figura están indicados los campos.

D. Cancelar una Subasta

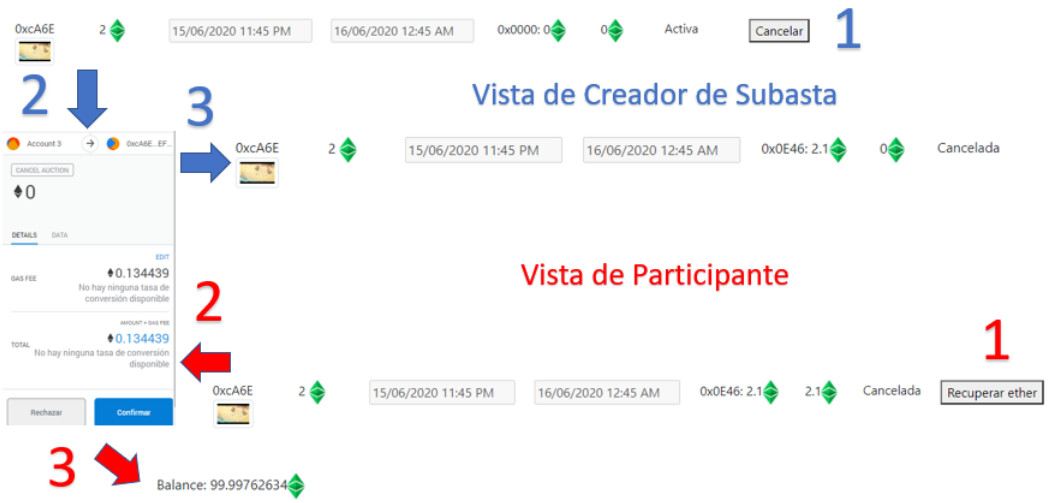

Fig 6. Vistas para Cancelar.

Para poder cancelar una subasta, se realiza la siguiente secuencia de pasos, desde la vista del creador de la subasta:

- Se selecciona el botón de cancelar.
- Se muestra su wallet para autorizar el movimiento.
- Se ve reflejado la cancelación en el lote.

Desde la vista del participante se puede solicitar el reintegro de su ether, a través de la siguiente secuencia de pasos:

- Se selecciona el botón de Recuperar ether.
- Se muestra su wallet para autorizar el movimiento.
- Se ve reflejado el movimiento en el balance de su wallet.

E. Smart Contracts

Estos son los Smart Contracts diseñados para este desarrollo:

```
contract Auction{
    // Non-mutable data
    address public owner;
    uint    public startBlock;
    uint    public endBlock;
    uint    public startDate;
    uint    public endDate;
    string  public ipfsHash;
    uint    public initialPrice;
    // Auction state
    bool    public canceled;
    uint    public highestBid;
    address public highestBidder;
    mapping(address => uint256) fundsByBidder;
    bool    public ownerHasWithdrawn;

    event LogBid(address bidder, uint bid, address highestBidder, uint highestBid, uint highestBindingBid);
    event LogWithdrawal(address withdrawer, address withdrawalAccount, uint amount);
    event LogCanceled();
```

Fig 7. Código Auction.

```
function createAuction(uint _startBlock, uint _endBlock, uint _startDate, uint _endDate,
                       string memory _ipfsHash, uint _initialPrice) public payable {
    require(msg.value >= 0.01 ether, "La comisión por el uso del sistema es de 0.01.");
    Auction newAuction = new Auction(msg.sender, _startBlock, _endBlock, _startDate, _endDate, _ipfsHash, _initialPrice);
    auctions.push(address(newAuction));
    emit AuctionCreated(address(newAuction), msg.sender, auctions.length, auctions);
}

function getAllAuctions() public view returns (address[] memory) {
    return auctions;
}

function getBalance() public view returns (uint256){
    return address(this).balance;
}

function withdraw() public onlyOwner returns (bool success){
    uint256 balance = address(this).balance;
    owner.transfer(address(this).balance);
    emit LogWithdrawal(msg.sender, balance);
    return true;
}

modifier onlyOwner {
    require(msg.sender == owner, "Solo el propietario del contrato puede ejecutar esta función.");
    _;
}
```

Fig 8. Código AuctionFactory.

Auction.sol

El Smart Contract Auction maneja la lógica de una subasta. Para llevar el control de la subasta se requiere guardar el creador de la misma (la cuenta que sube la subasta). Para evitar ataques Timestamp se utiliza aproximación por bloques (en Ethereum aproximadamente cada 15 segundos se mina un bloque). Se guardan las fechas inicial y final que eligió el usuario (informativas), el precio inicial del lote y el hash con el que se almacenó la imagen en IPFS. Esta información es inmutable.

Controlar el estado de la subasta se realiza mediante la creación de las variables Cancelado, puja más alta, dirección del ofertor de la puja más alta, retiro de la puja más alta y un mapa que relaciona las cuentas con sus pujas.

El contrato tiene 3 funcionalidades básicas: establecer una puja, cancelar una puja y retirar. Cualquier usuario (excepto el creador de la subasta) puede pujar por un lote. Solo el dueño de la subasta puede cancelarla. Un usuario que pujó y no ganó la subasta pueden reitrar, un creador al finalizar su subasta también puede retirar.

AuctionFactory.sol

El Smart Contract AuctionFactory sigue el patrón de diseño creacional que permite tener un solo punto de entrada (por ende control) de la creación de objetos (contratos en este caso).

El contrato tiene dos variables que una permiten llevar el control de los contratos creados y otra permite guardar al dueño del contrato, Auction y AuctionFactory (para obtener una remuneración por uso de la dApp).

La funcionalidad principal de este contrato es la creación de subastas utilizando el contrato Auction, lo que permite tener el control de las subastas que se creen.

IV. Alcances y Conceptos

Como vimos en el apartado anterior, es posible crear subastas, emitir una puja, consultar lotes, cancelar una subasta, esto se consiguió porque en el diseño de los Smart Contracts se definieron las reglas para esa lógica de negocio.

Adicionalmente a lo anterior, también se puede consultar los resultados de las subastas y sereembolsa a los participantes no ganadores, en las siguientes subsecciones se verán los desafíos presentes y la soiución propuesta.

A. Timestamping

Esta vulnerabilidad se produce cuando un contrato usa la *block.timestamp* como parte de la condición desencadenante para ejecutar una operación crítica (por ejemplo, transferencia de dinero) o como fuente de aleatoriedad, que sin embargo puede ser manipulada por un minero malicioso. La vulnerabilidad es causada por que Ethereum solo prescribe que una marca de tiempo debe ser mayor que la marca de tiempo de su bloque principal y debe estar dentro de los 900 segundos futuros del reloj actual. Si un contrato utiliza una condición basada en la marca de tiempo (por ejemplo, block.timestamp 25 == 0) para determinar si se transfiere o no dinero, un minero malintencionado puede cambiar ligeramente la marca de tiempo para satisfacer la condición en beneficio del atacante. Esta vulnerabilidad se puede prevenir al no usar block.timestamp en ninguna condición de toma de decisiones. [3]

La propuesta de solución está dada por la aproximación por bloques descrito en la lógica de negocio de los Smart Contracts.

B. Withdrawal Pattern

Asigna la responsabilidad de reclamar fondos al destinatario de los fondos: el destinatario debe enviar una transacción para retirar y obtener sus fondos.

Esto puede simplificar un contrato inteligente que envía fondos a los destinatarios, ya que el contrato no tiene que manejar los casos de qué hacer si falla el envío de fondos. Un contrato inteligente no sabe si el envío de los fondos falló debido a un error real, o si el destinatario es un contrato inteligente malicioso que se niega deliberadamente a aceptar los fondos.

Para evitar problemas de tipo re-entrancy, la mayoría de las funciones primero realizarán algunas verificaciones (quién llamó a la función, son los argumentos dentro del rango, si enviaron suficiente Ether, la persona tiene tokens, etc.). Estas verificaciones deben hacerse primero.

Como segundo paso, si se pasan todas las comprobaciones, se deben hacer efectos en las variables de estado del contrato actual. La interacción con otros contratos debe ser el último paso en cualquier función. [3]

En la lógica de negocio implementada en el diseño de los Smart Contracts, se consideraron estas implicaciones del problema de re-entrancy, ya que se establecieron reglas para el retiro de fondos manualmente por los usuarios, para evitar dobles gastos y por ende se vea reducida su cantidad de Ether en sus wallets, esta es la propuesta de solución ofrecida.

V. Conclusiones

El presente proyecto terminó de cimentar los conocimientos adquiridos a lo largo del semestre, permitió adquirir una amplia visión respecto a realizar implementaciones bajo el conocimiento que rodea el concepto de Blockchains, la importancia de establecer correctamente Smart Contracts, la posibilidad de analizar las interacciones entre los participantes y sus transacciones, visualizando el cumplimiento de las reglas establecidas en los Smart Contracts.

Otro aspecto a destacar es la adopción de otras herramientas, de las cuales se aprendió mucho durante la exploración de su funcionamiento, notar que pese a tener experiencia previa en desarrollo, en este esquema de trabajo, el enfoque cambia un poco, debido al proceso de minado el cual no puede ser controlado, representando algo nuevo.

REFERENCES

[1] Truffle Blockchain Group. (2020). Truffle. [Online]. Available: https://www.trufflesuite.com/docs/truffle/overview
[2] Truffle Blockchain Group. (2020). Ganache. [Online]. Available: https://www.trufflesuite.com/docs/ganache/overview
[3] Ethereum Community. (2020). Solidity. [Online]. Available: https://solidity.readthedocs.io/en/latest/index.html
[4] Ethereum Community. (2016). web3.js. [Online]. Available: https://web3js.readthedocs.io/en/v1.2.9/
[5] Facebook Inc. (2020). React. [Online]. Available: https://es.reactjs.org/docs/react-api.html
[6] Protocol Labs. (2020, May 29). IPFS. [Online]. Available: https://docs.ipfs.io/
[7] ConsenSys Formation. (2020). MetaMask. [Online]. Available: https://metamask.io/index.html
[8] B. Bellomy. (2017, June 29). Solidity tutorial: building a simple auction contract. [Online]. Available: https://medium.com/@bryn.bellomy/solidity-tutorial-building-a-simple-auction-contract-fcc918b0878a