Memoria

1. Justificación del uso de la tecnología blockchain para resolver el problema propuesto.

Realmente, según mi punto de vista, no tiene mucho sentido utilizar blockchain para este caso de uso ya que con una solución tradicional con una base de datos relacional sería suficiente para resolver el problema. No se comparte información entre distintas empresas o entidades que necesite ser guardada por miedo a ser modificada por uno de los actores sin el permiso de los demás, tampoco se llevan a cabo procesos donde se comparta información entre varios actores y el caso de uso no obtiene ningún beneficio al utilizar tokens y NFT´s.

1. Análisis y modelo del sistema propuesto.

Para resolver el problema, he utilizado la aproximación ProxyContract (aunque no me ha terminado de funcionar, están los Smart contracts que tendrían que hacer de proxy, están desplegados, pero luego desde el código javascript se llama directamente a los Smart contracts con la lógica, no a los proxies, que es a quienes se debería llamar), también he separado la lógica de negocio de los datos, utilizando Smart contract solo para los datos.

Para la parte de los drones tenemos tres Smart contract, uno es el proxy, otro el que tiene la lógica de negocio, y otro el que tiene los datos. El Smart contract que tiene la lógica de negocio necesita el address del Smart contract de los datos para poder acceder a los datos, a su vez, el Smart contract que hace de proxy, necesita el address del Smart contract con la lógica para delegar las llamadas a este.

La parte de las parcelas es exactamente igual que la de los drones.

Los Smart contract con la lógica de negocio (ParcelasERC721 y DronesERC721) heredan de la interface del ERC721 de openZeppelin, además de otras interfaces o clases (Context, IERC721Metadata, ERC165, Ownable).

Los Smart contract de datos (EternalDronesStorage y EternalParcelasStorage) son Smart contracts normales que guardan todos los datos de los Smart contract que representan a los drones y parcelas (que hemos dicho que eran ERC721). Utilizo un Smart contract de utilidades de Counters para asignar un id único a cada nft que se crea (dron o parcela)

Los Smart contract de proxy también son Smart contract normales que tienen el address de los ERC721 para delegar las llamadas

Todos los Smart contracts heredan de Ownable para no permitir que se cambien las referencias del address de datos o lógica a cualquier usuario, sino solamente al dueño de los Smart contracts (la cuenta que los ha desplegado).

Además también tenemos el Smart contract MyTokenERC20 que sirve para realizar los pagos (es Smart contract que hereda del Smart contract ERC20) y no tiene nada en especial.

También tenemos el Smart contract OperacionesDronesParcelas, que es el Smart contract que tiene la lógica que permite “alquilar un dron” al dueño de la parcela, realizándose el pago de este mediante el token MyTokenERC20 (primero se llama al método approve desde javascript para permitir al Smart contract realizar el pago, y luego desde el Smart contract se hace la transferencia de fondos).

Adjunto varias address con las que he realizado las pruebas, que tienen tokens de MyTokenERC20. La forma de distribuir los tokens no forma parte de este trabajo, pero creo que sería interesante hacer de vez en cuando ICO´s con los token haciendo descuentos o algún otro tipo de promoción para hacer más interesante el modelo de negocio, así como ofrecer ventajas si se paga con el token MyTokenERC20.

Por último tenemos un Smart contract llamado DataLibrary que tiene dos structs, uno representa a los drones y otro a las parcelas, y se utilizan en el resto de Smart contracts como esttructuras de datos.

Adjunto diagramas UML de las clases (herencia), solo se muestran los Smart contracts que tienen algún tipo de herencia, descendencia… y solo se muestran los Smart contracts que he realizado yo (Por ejemplo, MyTokenERC20 hereda del smartcontract ERC20, que a su vez, implementa la interface IERC20, esa interface no se muestra).

A su vez, ParcelasERC721 y DronesERC721 también heredan de los Smart contracts Context, ERC721 y Ownable. MyTokenERC20 hereda del Smart contract ERC20.

Diagrama

Descripción generada automáticamente

ParcelasERC721 y DronesERC721 implementan una interface y no heredan directamente de la clase ERC721 porque la clase ERC721 tiene el almacenamiento de la información en la propia clase, por lo que únicamente implemento la interface y gestiono el almacenamiento por mi cuenta con otros Smart contracts (los Smart contract de storage).

La idea hubiera sido hacerlo también con MyTokenERC20 pero no he tenido tiempo, por lo que la información está almacenada en este Smart contract directamente. Lo ideal hubiera sido implementar la interface IERC20 y tener una clase storage de almacenamiento.

A continuación muestro el diagrama UML con la relación de clases.

Diagrama

Descripción generada automáticamente

1. Descripción del entorno de desarrollo utilizado

He utilizado como editor de código Visual Studio Code, como framework de desarrollo he utilizado truffle, para las pruebas locales he utilizado ganache, que es parte de truffle.

La solución está hecha con Solidity 8 (la red local de alastria no funcionaba, además solo admite hasta la versión 5 de solidity, que es una versión totalmente desfasada y no tiene ningún sentido hacerlo en esta versión. Si se intenta aprender una tecnología nueva y que está cambiando continuamente, hacer una práctica en una versión que está desfasada desde hace varios años, me parece ridículo). Además he utilizado openZeppelin como librería (openZeppelin por ejemplo no admite solidity 5) y node js y html para la parte web. Para la gestión de paquetes en node, he utilizado npm.

Aunque la solución es 100% funcional, hay muchas validaciones y facilidades de uso que no están implementadas, además no está hecha teniendo en cuenta ningún tipo de diseño. La solución está realizada para demostrar los conocimientos adquiridos en el desarrollo de Smart contracts, así que hay mucho que mejorar y depurar, aunque como he dicho, es 100% funcional.

1. Instrucciones de despliegue

La solución ya está desplegada (adjunto debajo cuentas y address de todos los Smart contracts). Si se quisiera volver a desplegar, se podría utilizando el comando:

Truffle compile (para compilar)

Truffle migrate –network ropsten (En vez de ropsten se podría configurar cualquier otra red).

Para levantar la parte web, desde la carpeta raíz del programa, hay que hacer node index.js, a partir de entonces, tendremos la aplicación disponible en http://localhost:8080

Además de a continuación, en el fichero útiles.js del código, esta toda esta información.

// Address despliegue 0xF0D3216cba4a15F374f6cF6C2f823051Ecb5B268

// Private Key address despliegue 306ea9284bae55f5c1eec4f6eff8f5628d707fdce072f52ef2f4ce2610db55c6

// Address empresa Drones 0xC17137ADf4Eccc08c13eeea81eFc13b9D8A72d32

// Private Key empresa drones b7f1085ef71ced11f1a5b6aa7add6eb678ad2200871ab016135913c28cd4395e

// Address Empresa parcelas 0xAb5943ED1894B6FdCCab1438eE6153a4f1B9aF54

// Private key empresa parcelas 2e131e47b5649f0d81ebb02605572e27e37b93f93b048a7adfba72976f30ccd0

Además hay creados dos drones

// Dron 1, altura mínima: 10, altura máxima: 500, coste: 3, pesticida aceptado: A

// Dron 2, altura mínima: 20, altura máxima: 1000, coste: 5, pesticida aceptado: B

Y dos parcelas

// Parcela 1, altura mínima: 1, altura máxima: 2000, pesticida aceptado: A

// Parcela 2, altura mínima: 10, altura máxima: 2000, pesticida aceptado: B

Todo el código está disponible en: <https://github.com/jubox70/UnirDrones>

No me dejaba adjuntarlo a la plataforma de UNIR así que lo he dejado en github.