MARGA GABARD USLENGHI

Memoria trabajo final curso Blockchain - UNIR

Experto universitario en desarrollo Blockchain

Índice

Capítulo 1 - Introducción

1.1 - Objetivos

El objetivo principal de este trabajo es el desarrollo de un sistema basado en Blockchain a través de la cual se apliquen los conocimientos adquiridos en el curso realizado de Experto Universitario Blockchain.

En este trabajo final se plantea un MVP (Producto mínimo viable) donde se priorizará las principales funcionalidades imprescindibles para que el producto cumpla sus funciones vitales y se pueda operar todo el ciclo de vida de una solicitud de contratación, dejando para siguientes fases la evolución del resto de funcionalidades.

1.2 - Detalle del caso de uso

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

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.
- 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.
- Coste. Las operación de fumigación es inmediata y debe lanzar un evento de parcela fumigada con el ID de la parcela

Las características de las parcelas son:

- Un identificador único y ascendente, que comienza en 1 y que no puede repetirse.
- Un propietario.
- 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. Esta operación debe lanzar un evento con el ID del Dron que va a realizar la fumigación y el ID de la parcela que se va a fumigar.
- Pago de la operación realizada desde la cuenta del propietario a la de la empresa. Para la gestión de pagos se debe crear un token propio basado en el estándar ERC 20. Además, los

drones y las parcelas pueden gestionarse mediante tokens no fungibles basados en el estándar ERC 721

La empresa solicita tener una interfaz web que le permita registrar los drones y asignarles trabajos. A su vez, también se debe proporcionar una interfaz web para que los propietarios de las parcelas las puedan registrar y tengan la posibilidad de contratar un dron con las características que requiere su parcela y que pueda desplazarse hasta la misma.

1.3 - Justificación de la tecnología Blockchain en este caso de uso

Blockchain está revolucionando muchos sectores gracias a brindar posibilidades que ninguna otra tecnología ofrecía previamente, las cuales se resumen en la posibilidad de que distintos actores con intereses diversos trabajen sobre un mismo registro de información y en la no necesidad de que un tercero controle dicho registro. En otras palabras, genera CONFIANZA entre los diferentes actores implicados.

El uso de Blockchain en el caso de uso permitirá generar a la empresa de Fumigación ventajas competitivas al dar servicio a través de una plataforma confiable y por tanto diferenciarse del resto de la competencia, así como al mismo tiempo reducir sus costes operativos y de mantenimiento ya que ciertas operativas críticas se podrán gestionar de forma dinámica sin necesidad de intervenciones humanas (a través de Smart Contracts).

También puede llegar a permitir mejorar la experiencia del usuario final ya que los drones (como elementos IoT) podrían llegar a registrar todos sus movimientos en la Blockchain y así ofrecer al cliente (propietario de las parcelas) información fiable de la situación de la fumigación contratada (cuanto tiempo ha requerido, si se ha ejecutado, costes…etc). Esa transparencia, trazabilidad y garantías que estaría dando la empresa de Fumigación redundaría en fidelización de sus clientes y en marketing que permitirá adquirir nuevos clientes y poder ampliar sus servicios a otros sectores/actividades.

Asimismo, le permite a la empresa disponer de información fiable (no alterada) de los patrones de vuelo de los drones, así como el análisis de datos en tiempo real, para explotar dicha información en beneficio del negocio.

Dada la capacidad de sistema distribuido de Blockchain permite mantener una copia de todos los registros preservando la disponibilidad y el rendimiento para hacer que la arquitectura de la solución sea más robusta y resistente a cualquier potencial problema o ataque. Este sentido se está implementado un sistema más robusto y seguro que los tradicionales.

Para este caso de uso se implementará como forma de pago de los servicios una criptomoneda (Utility Token) basada en el estándar ERC20. De esta forma se genera una demanda real de dicha criptomoneda al servir como medio de pago para contratar los servicios de fumigación. Las ventajas que aportara la utilización de este medio de pago son:

-No hace falta implantar pasarelas de pago (con los ahorres de costes correspondientes)

-Cualquier persona que disponga de dicha criptomoneda (este donde este) puede realizar el pago del servicio.

-Permite a la empresa dispone de financiación (se puede plantear una ICO u IDO).

-Puede servir como medio de marketing.

El token ERC20 se implementa con la **UTILIDAD** de financiar el proyecto y permitir expandir por tanto las áreas de negocio y ámbitos. Tiene un **valor subyacente** para un determinado target de personas que quieran utilizar los servicios de la empresa y al mismo tiempo para aquellos que quieran especular con su valor ya que podrá ir vinculado a la evolución de la empresa.

Investigando por internet de los diferentes casos de uso que ya existe hoy en día (similar al de este trabajo, IoT y Blockchain) quiero destacar uno especial (Gónzalez, 2020) que pone en relevancia como la tecnología Blockchain en el caso de rastreo de la actividad de los drones puede llegar a ofrecer como gran valor añadido la transparencia, con lo cual se vuelve a poner en valor la confianza.

Capitulo 2 - Desarrollo de la solución propuesta

A continuación se explican los detalles de la solución propuesta para cubrir los casos de uso planteados y como se ha desarrollado dicha solución.

NOTA: el código del proyecto desarrollado se encuentra en:

https://github.com/MargaGabard/Trabajo-final-curso-ExpertoBlockchain.git

2.1 - Abstracción y nivel conceptual del problema a resolver

En los siguientes apartados se explicará a nivel conceptual la solución que se va a desarrollar, recurriendo en algunos casos a la notación UML para analizar los detalles conceptuales.

2.1.1 - Fases del desarrollo

Dado que se trata de un trabajo de final de curso donde el principal objetivo es aplicar los conocimientos adquiridos, se ha acotado el alcance de esta entrega a un MVP con determinada completitud funcional, dejando para siguientes fases la completitud mejora de la operativa.

Se implementará desde la primera entrega una aplicación Web3 donde el usuario interactuará a través de sus address y su Wallet (en este caso Metamask).

En esta primera fase se contempla la entrega de las siguientes funcionalidades:

- Crear y registrar nuevo Dron
- Obtener información Dron

- Crear y registrar nueva Parcela
- Obtener información Parcela

- Contratar fumigación (con todas las validaciones requeridas)
- Calcular compatibilidad tupla Dron/Parcela
- Pagar servicio de fumigación
- Fumigar parcela

- Implementación de nuevo Token ERC20
- Calcular coste fumigación (con Token ERC20)
- Mostrar situación balance empresa y propietario (vinculada a su Token ERC20)

Se implementará un back-end con lógica de negocio en la blockchain de Ethereum a través de Smart Contracts y dos front-end's donde los usuarios y la empresa (owner) puedan operar el sistema.

En las siguientes fases se contempla conseguir una completitud funcional/operativa e implantar las siguientes mejoras/acciones:

- Evolución de las funcionalidades de negocio (mantenimiento de información de Drones y Parcelas,).

- Optimización de los smart contract para mejorar consumo de GAS

- Adaptar los tokens ERC721 para que se puedan guardar los datos menos críticos/importantes off-chain (posiblemente en *IPFS: InterPlanetary File System o Swarm*). El almacenamiento en Blockchain es muy costoso y el protocolo IPFS permite subir información e indexar el hash del fichero subido con los metadatos (en este caso del Dron y/o Parcela) a la transacción correspondiente en la cadena de bloques.

  Guardar datos en IPFS proporciona un hash único. En lugar de almacenar los datos en el contrato, solo almacenaremos el hash en el contrato y luego podremos usar el hash para recuperar los datos por lo que la reducción de gas será notoria.

- Agregar en los Smart Contracts el patrón Proxy de OpenZeppelin para permitir evolucionar la lógica de negocio sin perder los estados previos.

- Agregar patrón a los Smart Contracts para poder detener la operativa de la aplicación (en caso de ser necesario)

- Control de actores y activos digitales (Drones y Parcelas) a través de protocolos de identidad (Veramo, antes Uport o similar).

- Profesionalizar el front-end para que sea más usable y atractivo por parte de usuarios y empresa. Agregar front-end para móvil.

- Hacer uso de la tecnología QR para facilitar la usabilidad.

- Comercialización en la propia empresa del Token ERC20 (hacer un pequeño intercambio centralizado).

- Pruebas exhaustivas del sistema

Fases más avanzadas y pendientes de estudio:

- Analizar qué mecanismos e infraestructura se requeriría para poder disponer de información en directo de la trazabilidad de la actividad del Dron y poder explotar dicha información, tanto por parte de la empresa como usuarios.

2.1.2 - Descripción del entorno de desarrollo utilizado

Para desarrollar el proyecto se han utilizado las siguientes herramientas/versiones:

Herramienta	Versión
NVM	1.1.9 (para Windows 10)
NPM	v6.14.11
Remix IDE y Remixd (plugin)	V0.21.4
Solidity	0.8.1
Framework de desarrollo Ethereum - Truffle	v5.5.4
Ganache	v7.0.3
Nodejs	12.20.2
Web3js	web3.js@1.0.0-beta.35
HTML server (Python)	3.10.2

Using Testin:

Mocha
Truffle

Browser tools:

Metamask
Browser (Edge/Firefox/Chrome)

Herramientas Front end:

HTML5
CSS
Framework MUI

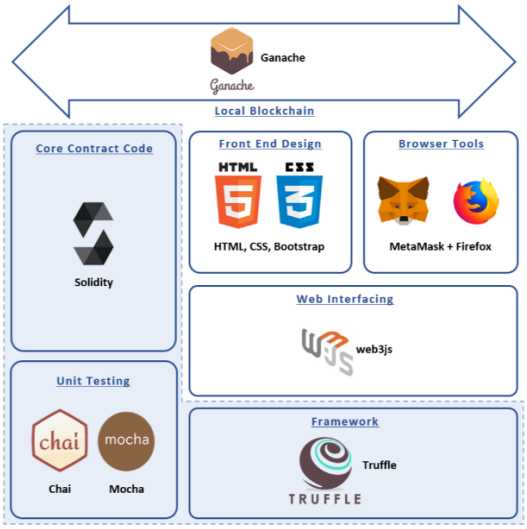


Imagen de las diferentes tecnologías utilizadas

2.1.3 - Arquitectura del sistema

Para desarrollar la dapp se han tenido que utilizar diferentes herramientas (detalladas en apartado "Descripción del entorno de desarrollo utilizado") de forma integrada y con la visión puesta en una arquitectura que cubra las necesidades de escalabilidad del sistema.

La siguiente imagen detalla los principales componentes de esta arquitectura:

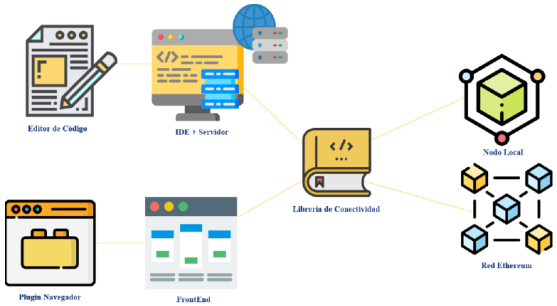


Diagrama de arquitectura de alto nivel

2.1.4 - Actores implicados

Se han identificado los siguientes actores directos que intervendrán en el sistema:

- **Empresa** que suministra el servicio de fumigación (Owner del negocio)
- **Propietarios** de las parcelas

2.1.5 - Componentes del sistema (diagrama de clases)

Smart Contracts

Los Smart Contracts desarrollados en la Dapp están escritos en Solidity. Se han implementado los siguientes Smart Contracts:

- FumigaSA.sol
- FumiTokenColeccion.sol
- DronTokenColeccion.sol
- ParcelaTokenColeccion.sol
- FumToken.sol
- Owneable.sol

A continuación se detallan algunas particularidades de los Smart Contracts para clarificar el comportamiento de la solución:

El contrato **FumigaSA.sol** es donde se centra la principal lógica de negocio de la aplicación. Guarda las address de la colección de tokens de Drones y Parcelas, así como una relación (mapping) de Drones y su estado de asignación (contratación fumigación).

Las principales funciones de este contrato son:

altaDron

lanza la función mint del token para de esa forma incrementar la secuencia y también guardar la información en las variables de estados correspondientes al token

```
function crearDron(address to, uint alturaMax, uint alturaMin, uint costeVuelo, uint
autonomia, uint m2_por_minuto, uint[] memory fertilizantes) public returns(uint256){

        require(alturaMax > alturaMin, "Valores alturas incorrectos");
        DronTokenColeccion _dronToken = DronTokenColeccion(contract_DronToken);
        uint256 tokenId = _dronToken.mint(to, alturaMax, alturaMin, costeVuelo,
autonomia,m2_por_minuto,fertilizantes);
        emit crearDronLog(tokenId);
        return tokenId;
  }
```

altaParcela

Idem que el punto anterior pero con las Parcelas

```
function crearParcela(address to, uint alturaMax, uint alturaMin, uint256 m2superfice,
uint[] memory fertilizantes) public returns(uint256 result){
        require(alturaMax > alturaMin, "Valores alturas incorrectos");
        ParcelaTokenColeccion _parcelaToken =
ParcelaTokenColeccion(contract_ParcelaToken);
        uint256 tokenId = _parcelaToken.mint(to, alturaMax, alturaMin, m2superfice,
fertilizantes);
        emit crearParcelaLog(tokenId);
        return tokenId;
  }
```

contratarFumigacion

Verifica perviamente que el dron este disponible y que tiene asignada la parcela que se solicita fumigar, y que dicha asignación este activa. Finalmente actualiza estado contratacion y lanza evento.

```
function contratarFumigacion (uint256 idDron, uint256 idParcela, uint idPestizida)
external returns (bool result)
{
    //Mirar si el dron esta disponible
    require (dronDisponible(idDron), "Dron no disponible");
    require (compatibilidadDronParcela(idDron,idParcela), "Dron no compatible");


  //Agrego la información a la relacion de dron-contratacion
    listaFumigaciones[idDron].idParcela= idParcela;
    listaFumigaciones[idDron].estado= estadoContratacion.Activa;
    listaFumigaciones[idDron].idPestizida= idPestizida;
    emit contratarFumigacionLog(idDron,idParcela, idPestizida, msg.sender, true);
    return true;

    }
```

fumigacionParcela

Únicamente el Owner del contrato puede ejecutar esta función por eso tiene el modificador correspondiente.

Actualiza situación y deja el dron listo para otra contratación.

```
function fumigacionParcela (uint256 idDronFumigar, uint256 idParcelaFumigar) external
onlyOwner returns (bool result)
{

    //Mirar si el dron esta asignado a esa parcela y esta activa la contratación
    require(listaFumigaciones[idDronFumigar].idParcela== idParcelaFumigar, "Dron no
esta asignado a parcela");
    require(listaFumigaciones[idDronFumigar].estado== estadoContratacion.Activa,
"Dron-Parcela no activa");


  //Se asume en este punto que la fumigación se ha ejecutado ya que es inmediata y se
marca como tal
    listaFumigaciones[idDronFumigar].estado= estadoContratacion.Ejecutada;
    emit
ParcelaFumigadaLog(idDronFumigar,idParcelaFumigar,listaFumigaciones[idDronFumigar].est
ado);
    return true;

    }
```

compatibilidadDronParcela

Esta función es muy importante porque esta parte de la lógica que controla que un dron pueda o no fumigar una parcela.

La lógica para determinar si es compatible o no se base en las alturas máximas y minimas del dron y la parcela (tiene que estar dentro del rango), los pesticidas que ambas

suministran/requieren (el dron tiene que suministrar todos los pesticidas que requiere la parcela).

```
function compatibilidadDronParcela (uint256 idDron, uint256 idParcela) public view
returns (bool result)
{
   // uint minutosVueloDron;
    DronTokenColeccion _DronToken = DronTokenColeccion(contract_DronToken);
    ParcelaTokenColeccion _ParcelaToken =
ParcelaTokenColeccion(contract_ParcelaToken);


    if (_DronToken.getAlturaMinima(idDron) > _ParcelaToken.getAlturaMaxima(idParcela))
         return false;

    if (_ParcelaToken.getAlturaMinima(idParcela) > _DronToken.getAlturaMaxima(idDron))
         return false;

    uint[] memory pestizidasDron =  _DronToken.getListaPesticidas(idDron);
    uint[] memory pestizidasParcela =  _ParcelaToken.getListaPesticidas(idParcela);

    bool tienePestizida;

    //Si la parcela requiere mas pestizidas de los que soporta el Dron no son
compatibles
    if (pestizidasParcela.length > pestizidasDron.length){
       return false;
    }

    //Si la parcela no requiere de ningun pestizida en particular
    if (pestizidasParcela.length==0){
      return true;
    }

    for (uint i = 0; i < pestizidasParcela.length ; i++) {
              tienePestizida=false;
              for (uint j = 0; j < pestizidasDron.length ; j++) {
                    if (pestizidasParcela[i] == pestizidasDron[j] ) {
                      tienePestizida=true;
                    }
               }
               if (tienePestizida==false)
               {
                 return false;
              }
      }

     return true;

}
```

CosteFumigacionDronParcela

En esta función se calcula el coste que tendrá la fumigación del dron para una parcela concreta. El coste se calcula en base a:

-La cantidad de veces que hay que recargar la batería del dron en función de los m2 de superficie de la parcela y la autonomía del dron.

-El coste de vuelo del dron

Importe = (cantidad de veces que se recarga el dron * el coste de cada vuelo)

```
function costeFumigacionDronParcela (uint256 idDron, uint256 idParcela) external returns (uint
result)
{

    uint minutosVueloDron;

    DronTokenColeccion _DronToken = DronTokenColeccion(contract_DronToken);
    ParcelaTokenColeccion _ParcelaToken = ParcelaTokenColeccion(contract_ParcelaToken);

    //minutosVueloDron=10;
    minutosVueloDron= (_ParcelaToken.getSuperficieParcela(idParcela) /
_DronToken.getm2MinutoDron(idDron));

    emit costeFumiLog (_DronToken.getCosteVueloDron(idDron), 1
,_DronToken.getAutonomiaDron(idDron), minutosVueloDron);
    if (minutosVueloDron <= _DronToken.getAutonomiaDron(idDron))
        return (_DronToken.getCosteVueloDron(idDron));


    uint recargasDron=  (minutosVueloDron / _DronToken.getAutonomiaDron(idDron));
    emit costeFumiLog(_DronToken.getCosteVueloDron(idDron), 1
,_DronToken.getAutonomiaDron(idDron), minutosVueloDron);

    return (_DronToken.getCosteVueloDron(idDron) * recargasDron);

}
```

El contrato **FumiTokenColeccion** se basa en el estándar **ERC721** y es donde se agrupan la información y funcionalidades comunes a Parcelas y Drones.

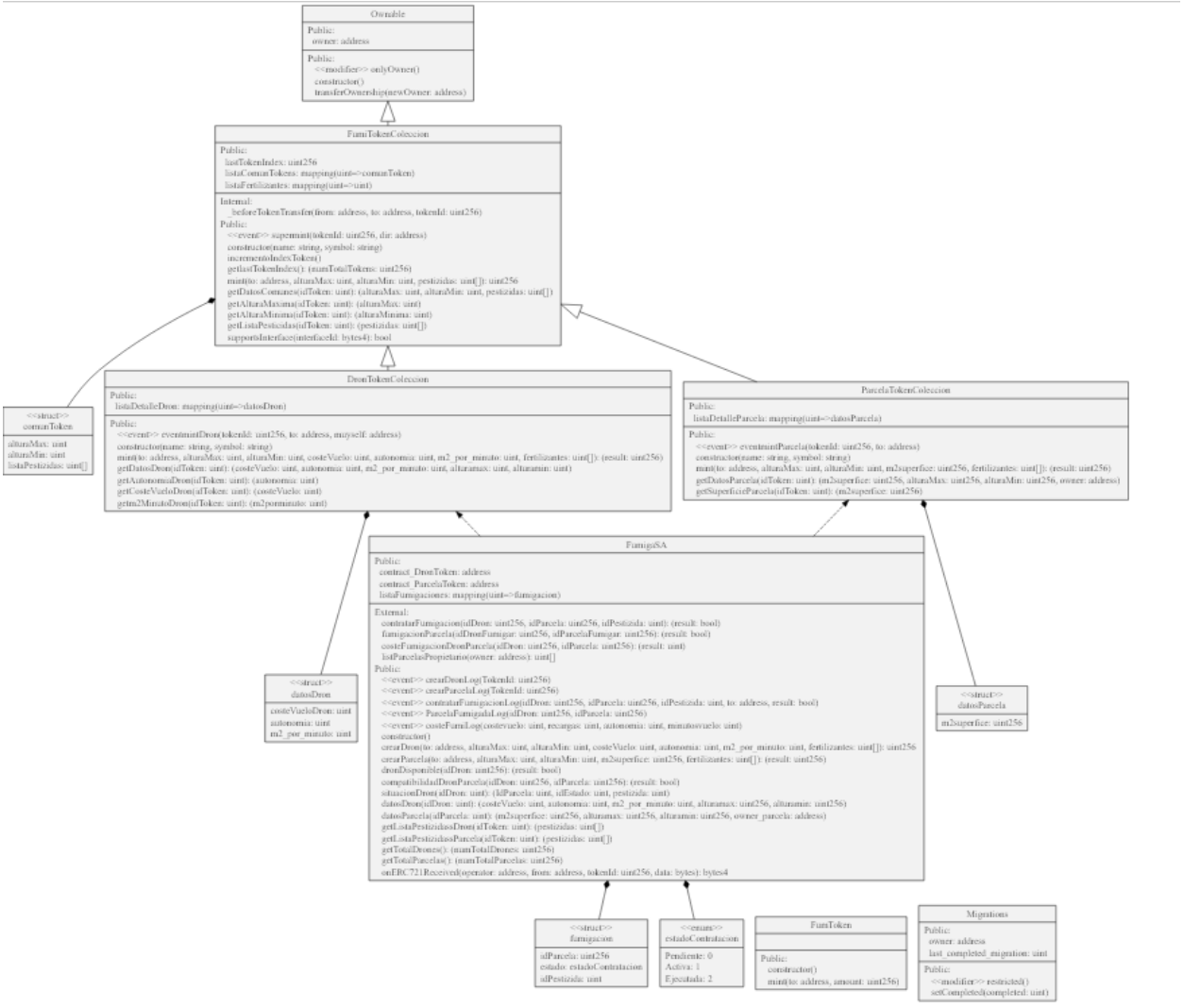
Los contratos **DronTokenColeccion** y **ParcelaTokenColeccion** heredan de FumiTokenColeccion e implementan las particularidades para cada uno.

El contrato **FumToken** es el token **ERC20** implementado como forma de pago de las contrataciones de los propietarios.

Diagrama de clases

Durante el diseño de la solución se ha utilizado el diagrama de clases para tener visión de los contratos y estructuras necesarios.

A continuación se visualiza dicho diagrama resultado que se ha conseguido con la herramienta de ingeniería inversa **(sol2uml)** para obtener el diagrama de clases definitivo.



2.1.6 - Diagramas de estados

Para registrar y controlar las contrataciones de los drones por parte de los propietarios se ha establecido una variable de estados (mapping) que relaciona el dron con la situación de contratación:

IdDron -- Estructura contratación

- IdParcela
- Estado
- idPestizida

Los estados por los que puede pasar la contratación son:

-**Pendiente** - cuando se da de alta por primera vez se le asigna este estado. En este estado el dron esta disponible para su contratación.

-**Activa** – una vez que el propietario ha solicitado y pagado la contratación de la fumigación

-**Ejecutada** – una vez que el owner ejecuta la fumigación y esta ha finalizado

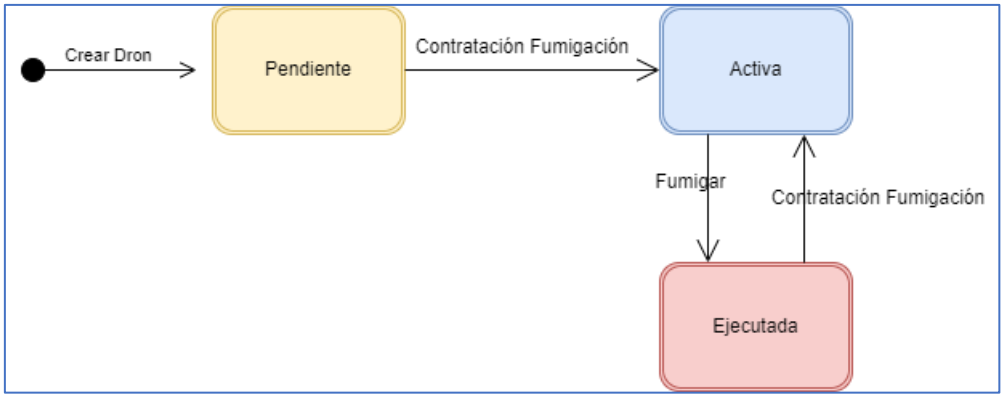
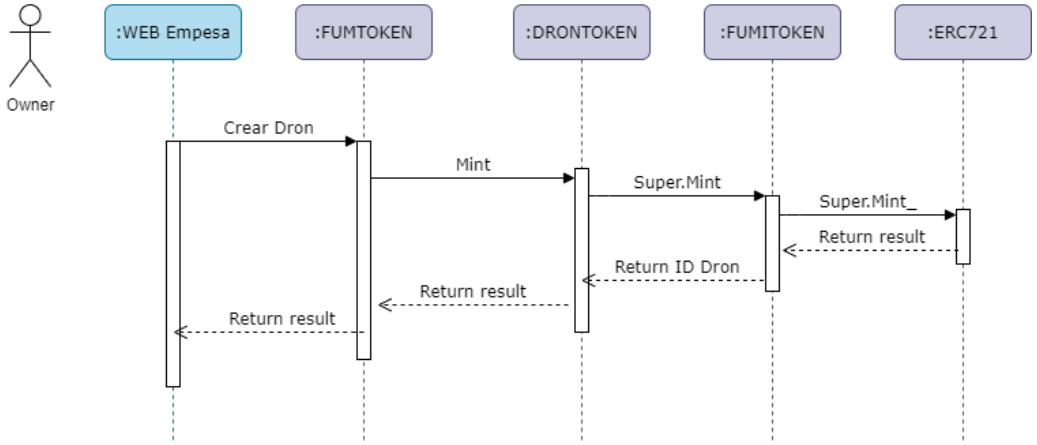


Diagrama de estados

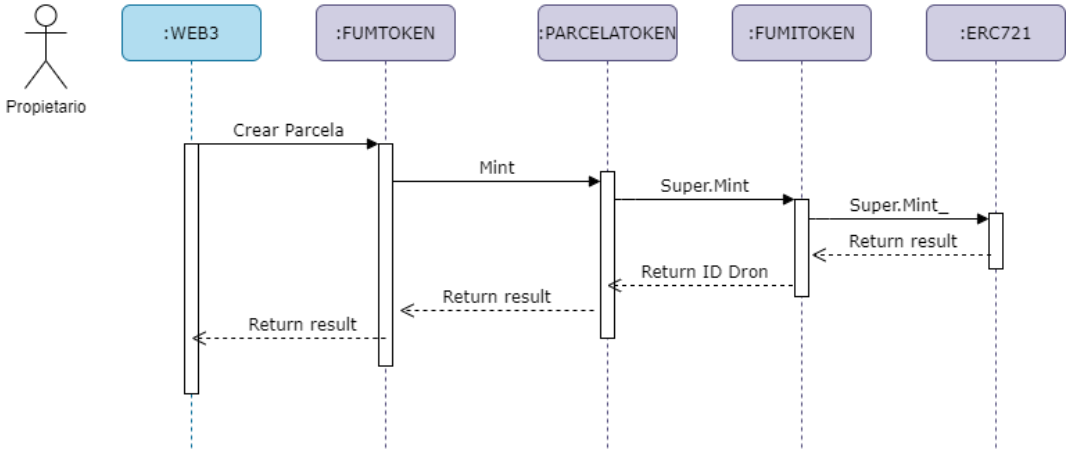
2.1.6 - Diagramas de secuencia

A continuación, se detallan los **diagramas de secuencia de instancias** de las principales funcionalidades de la aplicación. Se ha utilizado para la realización de los mismos la herramienta Draw.io (https://app.diagrams.net/).

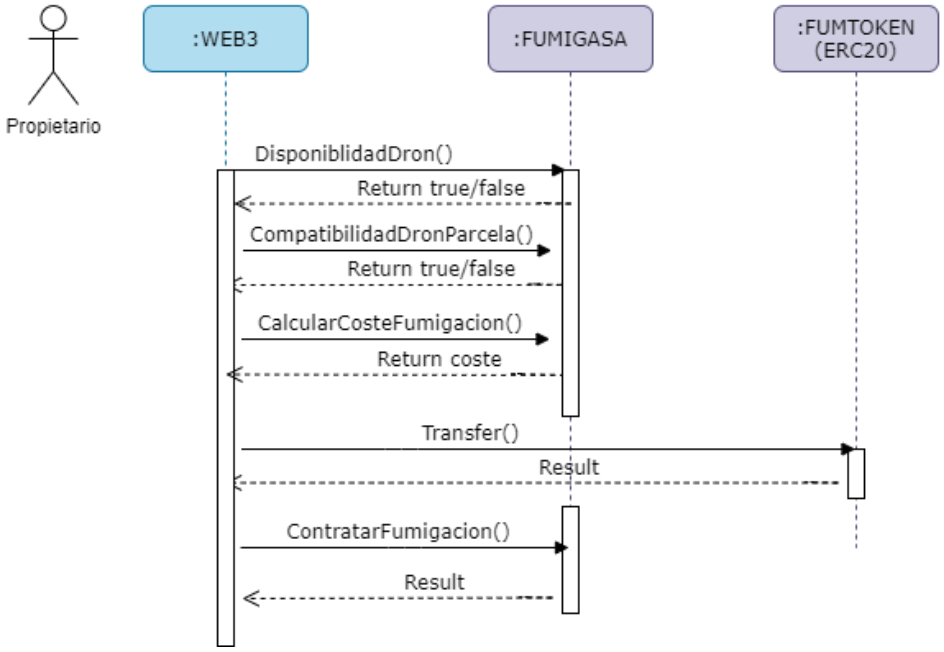
Secuencia para crear nuevo Dron



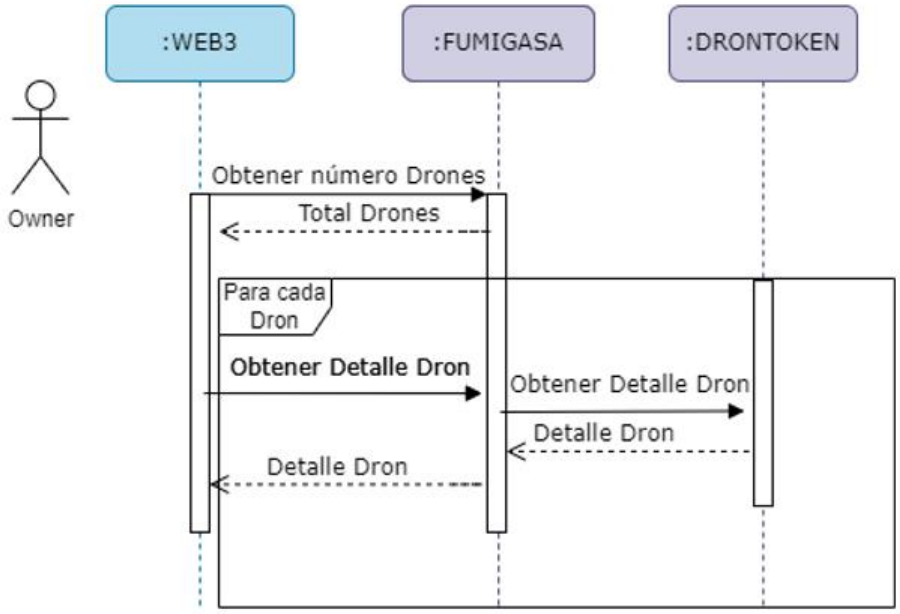
Secuencia para crear nueva Parcela



Secuencia para contratar un dron para fumigar una parcela



Secuencia para listar Drones existenes



3 - Despliegue del sistema

A continuación se presenta el diagrama de despliegue de la aplicación. Es una visión de cómo sería en caso de implantarse en una red Ethereum, para este caso solo se ha desplegado en una red local de pruebas (Ganache).

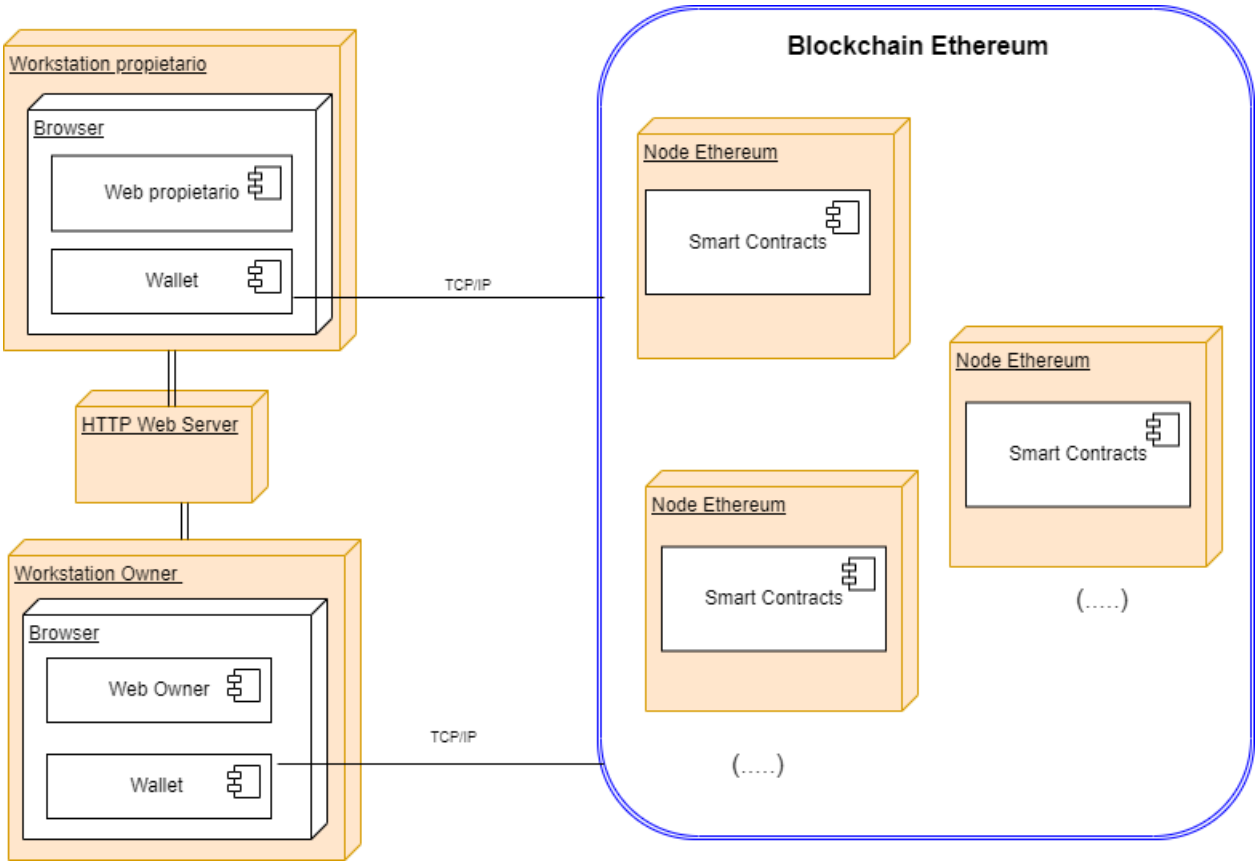


Diagrama de despliegue

NOTA: el código del proyecto desarrollado se encuentra en:

https://github.com/MargaGabard/Trabajo-final-curso-ExpertoBlockchain.git

Para poder desplegar la aplicación se deben seguir los siguientes pasos:

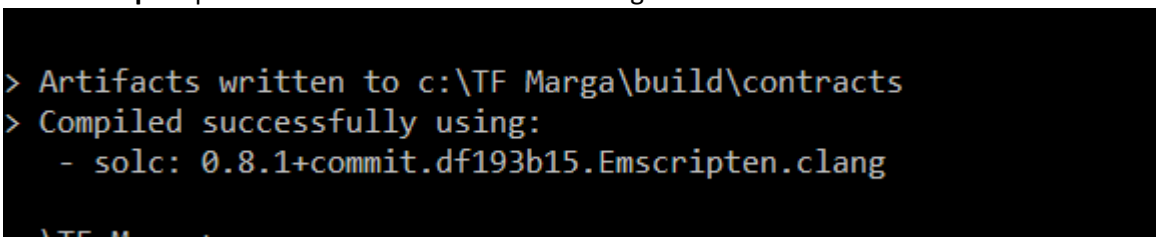
1. Instalar **Node.js** para lo cual se recomienda instalar el gestor de versiones **NVM** y a partir de este instalar la versión de node necesaria. En este caso nos hemos basado en la versión para Windows del siguiente repositorio:
2. https://github.com/coreybutler/nvm-windows)
3. Una vez instalado NVM se instalará la versión del nodo requerida con el comando (**NVM install [versión])**
4. **Creación de un directorio para el proyecto** donde se copiara la estructura de directorios y ficheros facilitados y nos movemos a dicho directorio.

5. Instalación de Truffle (**npm install truffle**)
6. Instalación de la red de pruebas de ganache **(install ganache-cli**)
7. Instalación de las librerías de openzeppelin (**npm install @openzeppelin/contracts**)
8. Instalación de las librerías de pruebas:
    a. **npm install mocha**
    b. **npm install chai**
9. **Instalación de http-server en puerto 8000**. En este caso se utilizo paython pero podría ser cualquiera.
10. **Configuración del fichero truffle-config** agregando la siguiente información de la red donde promocionar:

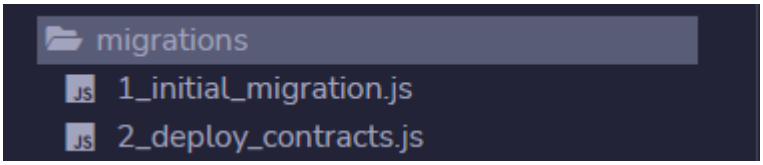
```
development: {
   host: "127.0.0.1",     // Localhost (default: none)
   port: 8545,            // Standard Ethereum port (default: none)
  network_id: "*",       // Any network (default: none)
 },
```

11. **Ejecutar la red de ganache** desde la línea de comandos (ganache-cli)

12. Una vez instaladas todas las herramientas/componentes y arrancado los servicios se deberá proceder a desplegar los contratos en este caso en la red ganache lanzando los siguientes comandos:
    truffle compile que mostrará un resultado como el siguiente:



    truffle migrate (que ejecutara los siguientes comandos)




```
const Migrations = artifacts.require("Migrations");

module.exports = function (deployer) {
  deployer.deploy(Migrations);
};
```

```
const FumigaSA = artifacts.require("FumigaSA");
const FumToken = artifacts.require("FumToken");
```

```
module.exports = function (deployer) {
  deployer.deploy(FumToken);
  deployer.deploy(FumigaSA);
};
```

El resultado que se vera será el siguiente:

```
Starting migrations...
======================
> Network name:    'development'
> Network id:      1647512754798
> Block gas limit: 90000000 (0x55d4a80)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0x699cfa191bd0102c77626662a88e45f180ea55fee0ff7b1d8b24dc55b8c5f9f8
   > Blocks: 0            Seconds: 0
   > contract address:    0x10992237F43EB5c17A0b677eA0Cf031C00FaA549
   > block number:        1
   > block timestamp:     1647512906
   > account:             0x0187b46F89CB9c1B1aC8fb03662FD45A4af1f693
   > balance:             99.99506192
   > gas used:            246904 (0x3c478)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00493808 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00493808 ETH
```

```
   Deploying 'FumToken'
   --------------------
   > transaction hash:    0x92203d1fc88eadb62c107303f9bf992d73fdc059f5cf2195392051266c244de4
   > Blocks: 0            Seconds: 0
   > contract address:    0xD596FE932D22548e7226088A4E0a8D0eFdE7242b
   > block number:        3
   > block timestamp:     1647512908
   > account:             0x0187b46F89CB9c1B1aC8fb03662FD45A4af1f693
   > balance:             99.96754074
   > gas used:            1333546 (0x14592a)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.02667092 ETH


   Deploying 'FumigaSA'
   --------------------
   > transaction hash:    0x471cf549631e233b04a88968e70cc00009c5f18a17050d40f7e3017348fb835d
   > Blocks: 0            Seconds: 0
   > contract address:    0x156754De38768bec31389c4DA77cAC0B412a99cC
   > block number:        4
   > block timestamp:     1647512909
   > account:             0x0187b46F89CB9c1B1aC8fb03662FD45A4af1f693
   > balance:             99.77438702
   > gas used:            9657686 (0x935d56)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.19315372 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.21982464 ETH

Summary
=======
> Total deployments:   3
> Final cost:          0.22476272 ETH
```

13. Modificar en el front-end las direcciones de los contratos y del owner (que es quien recibirá los tokens de las contrataciones). Ir a los ficheros index.html e index-owner.html y modificar las siguientes líneas:

```
cuentaEmpresa="0x0187b46F89CB9c1B1aC8fb03662FD45A4af1f693";
instanciaTokenERC20= await new web3.eth.Contract(ABI_FUMTOKEN, "0xD596FE932D22548e7226088A4E0a8D0eFdE7242b");
instanciaEmpresa= await new web3.eth.Contract(ABI_FUMIGASA, "0x156754De38768bec31389c4DA77cAC0B412a99cC");
```

14. Ejecutar el http-server en el puerto 8000

15. Abrir la web del owner y la del propietario (previamente se tendrá que importar al menos dos cuentas a Metamask y el token FumToken)
    localhost:8000 (web propietarios)
    localhost:8000/index-owner.html (web owner)

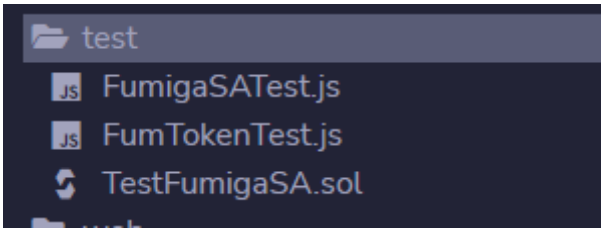
3 - Pruebas de validación

Las pruebas se han realizado de diferentes formas. En primer lugar, en fases tempranas del desarrollo se iban realizando a través de de Remix (probando la funcionalidad de los Smart Contract sin interacción front-end). A medida que se avanzaba con el desarrollo se comenzó a realizar de forma automática a través del framwork de Truffle y desde el propio front-end.

Además se utilizo el modulo solidity-coverage para comprobar el % de cobertura de las pruebas realizadas y el resultado fue del 71%.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	71.31	52.94	69.05	71.54	
DronTokenColeccion.sol	72.73	100	50	72.73	43,48,54
FumToken.sol	50	100	50	50	15
FumiTokenColeccion.sol	85.71	100	80	85.71	54,83
FumigaSA.sol	69.05	53.33	70.59	69.05	... 270,271,274
Ownable.sol	50	50	66.67	60	23,24
ParcelaTokenColeccion.sol	85.71	100	75	85.71	41
All files	71.31	52.94	69.05	71.54	

Los casos de pruebas automatizados se encuentran en la carpeta **Test** del proyecto.



Pruebas javascript utilizando Chai y Truffle-assertions.

```
const fumigaSA = artifacts.require("FumigaSA");
const assert = require("chai").assert;
const truffleAssert = require('truffle-assertions');

contract("fumigaSA", (accounts) => {

var instancia;
var owner;

describe('Pruebas TF', async () => {

    beforeEach(async function () {
        instancia = await fumigaSA.deployed();


        let totalDrones=await instancia.getTotalDrones.call();
        let totalParcelas=await instancia.getTotalParcelas.call();

        owner= await instancia.owner();
        console.log("Numero de drones BEFORE:",totalDrones.toNumber());
        console.log("Numero de parcelas BEFORE:",totalParcelas.toNumber());


    });

describe('Happy path', async () => {

  it('Deberia crear una nueva Parcela', async () => {

    let totalParcelas_before=await instancia.getTotalParcelas();

    // Creo nueva Parcela y compruebo si el número de parcelas totales se ha incrementado
    const pestizidas = [1,2];

    await instancia.crearParcela(accounts[2],7,2,10,pestizidas, { from: accounts[2] });

    var totalParcelas_new=await instancia.getTotalParcelas();
    totalParcelas_before=parseInt(totalParcelas_before)+parseInt(1);

    assert.equal(totalParcelas_new, totalParcelas_before, "No se ha incrementado el numero de parcelas");


    // Obtengo los datos de la parcela creada y comparo con los esperados
    var datosParcela;
    datosParcela = await instancia.datosParcela(totalParcelas_new);
//     console.log("Total parcelas new",totalParcelas_new.toNumber());
    assert.equal(datosParcela.alturamax, 7, 'La altura maxima deberia ser 7');
    assert.equal(datosParcela.alturamin, 2, 'La altura minima deberia ser 2');
    assert.equal(datosParcela.m2superfice, 10, 'Los m2 de superficie deberían ser 10');
    assert.equal(datosParcela.owner_parcela, accounts[2] , 'El popietario es incorrecto');

  });

  it('Deberia crear un nuevo Dron', async () => {


    let totalDrones_before=await instancia.getTotalDrones();
    const pestizidas = [1,2];

    // Creo nuevo Dron y compruebo si el número de drones totales se ha incrementado
    await instancia.crearDron(accounts[1],10,5,10,120,1,pestizidas, { from: accounts[1] });

    var totalDrones_new=await instancia.getTotalDrones();
    totalDrones_before=parseInt(totalDrones_before)+parseInt(1);

    assert.equal(totalDrones_new, totalDrones_before, "No se ha incrementado el numero de drones");
```

```
    // Obtengo los datos del dron creado y comparo con los esperados
    var datosDron;
    datosDron = await instancia.datosDron(totalDrones_new);

    assert.equal(datosDron.costeVuelo, 10, "El coste del vuelo deberia ser 10");
    assert.equal(datosDron.autonomia,120, "La autonomia deberia ser 120");
    assert.equal(datosDron.m2_por_minuto,1, "Los m2 de fumigacion por minuto deberia ser 1");
    assert.equal(datosDron.alturamax,10, "La altura maxima del Dron creado deberia ser 10");
    assert.equal(datosDron.alturamin, 5, "La altura minima del Dron creado deberia ser 5");

  });

  it('Deberia contratar fumigacion Dron/Parcela', async () => {
       //instancia = await fumigaSA.deployed();

       let totalDrones=await instancia.getTotalDrones.call();
       let totalParcelas=await instancia.getTotalParcelas.call();

    // Contrato fumigacion con el ultimo Dron y Parcelas creados
    await instancia.contratarFumigacion(totalParcelas.toNumber(),totalDrones.toNumber(),1, {from: accounts[2] });

    var estado_expected=1; //Activo
    let situacionDron = await instancia.situacionDron.call(totalDrones.toNumber());

    assert.equal(situacionDron.idEstado, estado_expected, "La tuplca Dron/Parcela no esta activa");

    });

  });


describe('Dron/Parcela no compatibles', async () => {

 it('Deberia crear una nueva Parcela', async () => {

    let totalParcelas_before=await instancia.getTotalParcelas();

    // Creo nueva Parcela y compruebo si el número de parcelas totales se ha incrementado
    const pestizidas = [2,3];

    await instancia.crearParcela(accounts[2],2,1,10,pestizidas, { from: accounts[2] });

    var totalParcelas_new=await instancia.getTotalParcelas();
    totalParcelas_before=parseInt(totalParcelas_before)+parseInt(1);

    assert.equal(totalParcelas_new, totalParcelas_before, "No se ha incrementado el numero de parcelas");

  });

  it('Deberia crear un nuevo Dron', async () => {


    let totalDrones_before=await instancia.getTotalDrones();
    const pestizidas = [1,4];

    // Creo nuevo Dron y compruebo si el número de drones totales se ha incrementado
    await instancia.crearDron(accounts[1],10,5,10,120,1,pestizidas, { from: accounts[1] });

    var totalDrones_new=await instancia.getTotalDrones();
    totalDrones_before=parseInt(totalDrones_before)+parseInt(1);

    assert.equal(totalDrones_new, totalDrones_before, "No se ha incrementado el numero de drones");

  });

  it('No deberia haber compatiblidad entre Dron y Parcela', async () => {

       let totalDrones=await instancia.getTotalDrones.call();
       let totalParcelas=await instancia.getTotalParcelas.call();

    // Compruebo compatiblidad entre el ultimo Dron y Parcelas creados
    var result= await instancia.compatibilidadDronParcela(totalParcelas.toNumber(),totalDrones.toNumber());
    console.log("resultado:",result.valueOf());
```

```
    assert.equal(result.valueOf(),false, "Dron/Parcela no deberian ser compatibles");
    });

  });

////////////////////////////////////////////////////////////////////////////////////////////////

describe('OnlyOwner puede Fumigar', async () => {


  it('Deberia dejar Fumigar porque es el Owner', async () => {


        var tx= await instancia.fumigacionParcela(1,1, {from: owner});

        //console.log("result:", result.valueOf());
        //assert.equal(result.status, true, "Debería haber dejado fumigar");

        truffleAssert.eventEmitted(tx, 'ParcelaFumigadaLog', (ev) => {
        return (ev.idParcela == 1 && ev.idDron== 1);
        });



    });

    it('No deberia dejar Fumigar porque NO es el Owner', async () => {


        var tx= await instancia.fumigacionParcela(1,1, {from: accounts[2]});
        truffleAssert.eventNotEmitted(tx, 'ParcelaFumigadaLog', (ev) => {
            return (ev.idParcela == 1 && ev.idDron== 1);
        });


    });


  });

});
});
```

Pruebas token ERC20 (FumToken):

```
const Token = artifacts.require("FumToken");

var chai = require("chai");

const BN = web3.utils.BN;
const chaiBN = require('chai-bn')(BN);
chai.use(chaiBN);

var chaiAsPromised = require("chai-as-promised");
chai.use(chaiAsPromised);

const expect = chai.expect;

contract("Token Test", async accounts => {
    const [ initialHolder, recipient, anotherAccount ] = accounts;


    it("Todos los tokens deberian estar en la cuenta inicial", async () => {
    let instance = await Token.deployed();
    let totalSupply = await instance.totalSupply();
```

```
    await
expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(totalSupply);
    });


   it("I can send tokens from Account 1 to Account 2", async () => {
      const sendTokens = 1;
      let instance = await Token.deployed();
      let totalSupply = await instance.totalSupply();
      await
expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(totalSupply);
      await expect(instance.transfer(recipient, sendTokens)).to.eventually.be.fulfilled;
      await
expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(totalSupply.sub(new
BN(sendTokens)));
      await expect(instance.balanceOf(recipient)).to.eventually.be.a.bignumber.equal(new
BN(sendTokens));
    });


    it("It's not possible to send more tokens than account 1 has", async () => {
      let instance = await Token.deployed();
      let balanceOfAccount = await instance.balanceOf(initialHolder);
      await expect(instance.transfer(recipient, new
BN(balanceOfAccount+1))).to.eventually.be.rejected;

      //check if the balance is still the same
      await
expect(instance.balanceOf(initialHolder)).to.eventually.be.a.bignumber.equal(balanceOfAccount);

    });



});
```

Las pruebas de los contratos se realizaron principalmente desde Remix pero para aprender a desarrollar test automaticos con Trufflee se realizo implemento este caso concreto:

```
pragma solidity ^0.8.1;
import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/fumigaSA.sol";

contract TestFumigaSA {

struct datosDron{

    uint costeVuelo;
    uint autonomia;
    uint m2_por_minuto;
    uint alturaMax;
    uint alturaMin;
}


uint[] pestizidas;
uint[] pestizidas_expected;

 function testCrearDron() public {
    FumigaSA instancia = FumigaSA(DeployedAddresses.FumigaSA());
```

```
    datosDron memory datos;
    datosDron memory datosExpected;
    uint expected = 0;

    uint totalDrones= instancia.getTotalDrones();
    Assert.equal(totalDrones, expected, "Ya hay otros Drones creados");

    //Datos expected
    datosExpected.alturaMax = 10;
    datosExpected.alturaMin = 5;
    datosExpected.costeVuelo=10;
    datosExpected.autonomia=120;
    datosExpected.m2_por_minuto=1;

    //Datos para dar de alta el Dron de prueba
    datos.alturaMax = 10;
    datos.alturaMin = 5;
    datos.costeVuelo=10;
    datos.autonomia=120;
    datos.m2_por_minuto=1;


    uint idDron= instancia.crearDron(tx.origin, datos.alturaMax , datos.alturaMin, datos.costeVuelo ,
datos.autonomia, datos.m2_por_minuto, pestizidas);

    uint totalDrones_expected=totalDrones+1;
    totalDrones= instancia.getTotalDrones();
    Assert.equal(totalDrones, totalDrones_expected , "La cantidad de Drones no se ha incrementado");


    //Se valida que todos los datos del Dron se hayan guardado correctamente
    (datos.costeVuelo, datos.autonomia, datos.m2_por_minuto,datos.alturaMax,datos.alturaMin) =
instancia.datosDron(idDron);
    Assert.equal(datos.costeVuelo, datosExpected.costeVuelo, "El coste del vuelo deberia ser 10");
    Assert.equal(datos.autonomia,datosExpected.autonomia, "La autonomia deberia ser 120");
    Assert.equal(datos.m2_por_minuto,datosExpected.m2_por_minuto, "Los m2 de fumigacion por minuto deberia ser 1");
    Assert.equal(datos.alturaMax,datosExpected.alturaMax, "La altura maxima del Dron creado deberia ser 10");
    Assert.equal(datos.alturaMin, datosExpected.alturaMin, "La altura minima del Dron creado deberia ser 5");


    //Se valida que la situación del Dron sea "Pendiente"

    uint estado_expected=0;

    uint idParcela;
    uint estado_dron;
    uint pestizida;

    (idParcela,estado_dron,pestizida) = instancia.situacionDron(idDron);

    Assert.equal(estado_dron, estado_expected, "La situacion del Dron creado no es correcta");

  }


}
```

Para ejecutar las pruebas se debe realizar los siguientes pasos:

truffle compile

truffle migrate

truffle test

```
  Contract: fumigaSA
    Pruebas TF
      Happy path
Numero de drones BEFORE: 0
Numero de parcelas BEFORE: 0
        √ Deberia crear una nueva Parcela (876ms)
Numero de drones BEFORE: 0
Numero de parcelas BEFORE: 1
        √ Deberia crear un nuevo Dron (1150ms)
Numero de drones BEFORE: 1
Numero de parcelas BEFORE: 1
        √ Deberia contratar fumigacion Dron/Parcela (841ms)
      Dron/Parcela no compatibles
Numero de drones BEFORE: 1
Numero de parcelas BEFORE: 1
        √ Deberia crear una nueva Parcela (1296ms)
Numero de drones BEFORE: 1
Numero de parcelas BEFORE: 2
        √ Deberia crear un nuevo Dron (1169ms)
Numero de drones BEFORE: 2
Numero de parcelas BEFORE: 2
resultado: false
        √ No deberia haber compatiblidad entre Dron y Parcela (294ms)
      OnlyOwner puede Fumigar
Numero de drones BEFORE: 2
Numero de parcelas BEFORE: 2
        √ Deberia dejar Fumigar porque es el Owner (372ms)
Numero de drones BEFORE: 2
Numero de parcelas BEFORE: 2
        √ No deberia dejar Fumigar porque NO es el Owner (281ms)

  Contract: Token Test
    √ Todos los tokens deberian estar en la cuenta inicial (252ms)
    √ I can send tokens from Account 1 to Account 2 (627ms)
    √ It's not possible to send more tokens than account 1 has (675ms)


  11 passing (12s)
```

4 - Funcionamiento de la aplicación

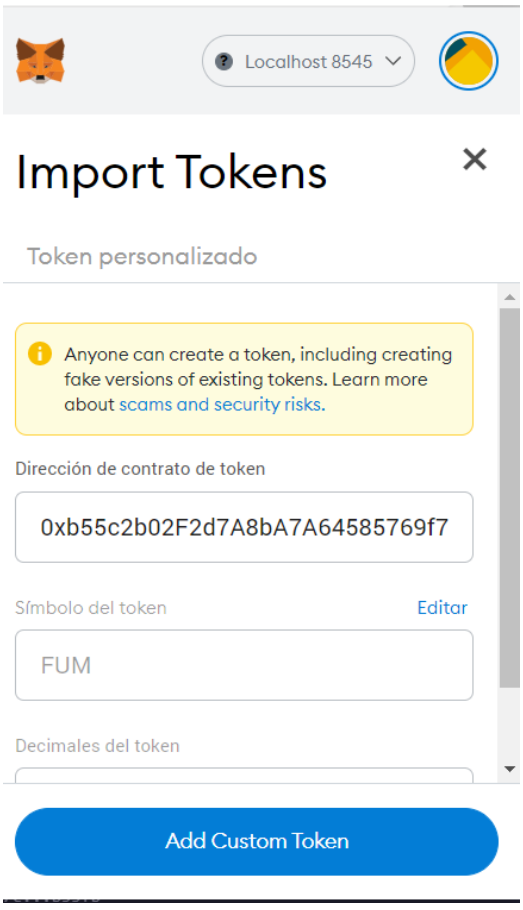
A continuación se presentan una secuencia de ejecución con las pantallas para que se pueda visualizar la operativa que soporta la aplicación en esta fase inicial.

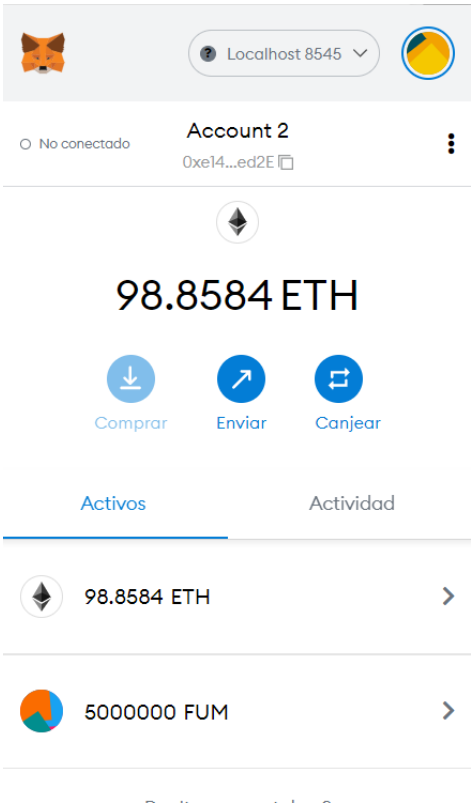
Esta secuencia se ha realizado con una red de prueba (cliente ganache) y se ha seguido el ciclo de un "***happy path***".

Como pre-requisitos para ejecutar la aplicación se importaran dos cuentas a metamask (de las que suministra ganache) y se importaran también los tokens (FumToken) a la cuenta del owner y se transferirá desde metamask una parte de estos a la cuenta del propietario para que así este ultimo pueda solicitar la contratación de fumigación.

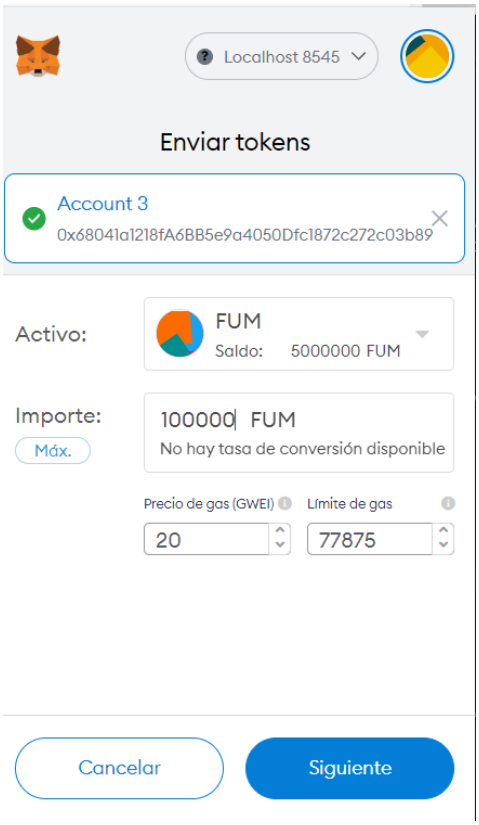
Nota: Si el usuario al conectarse a la web no ha logeado en matamask se le abrirá la ventana para solicitar dicho login y la conexión la/s cuentas con las que quiere operarar.

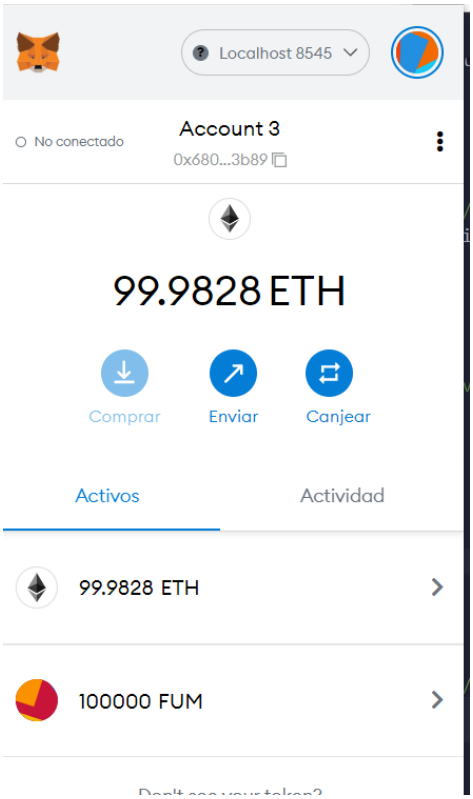
Cuenta owner:



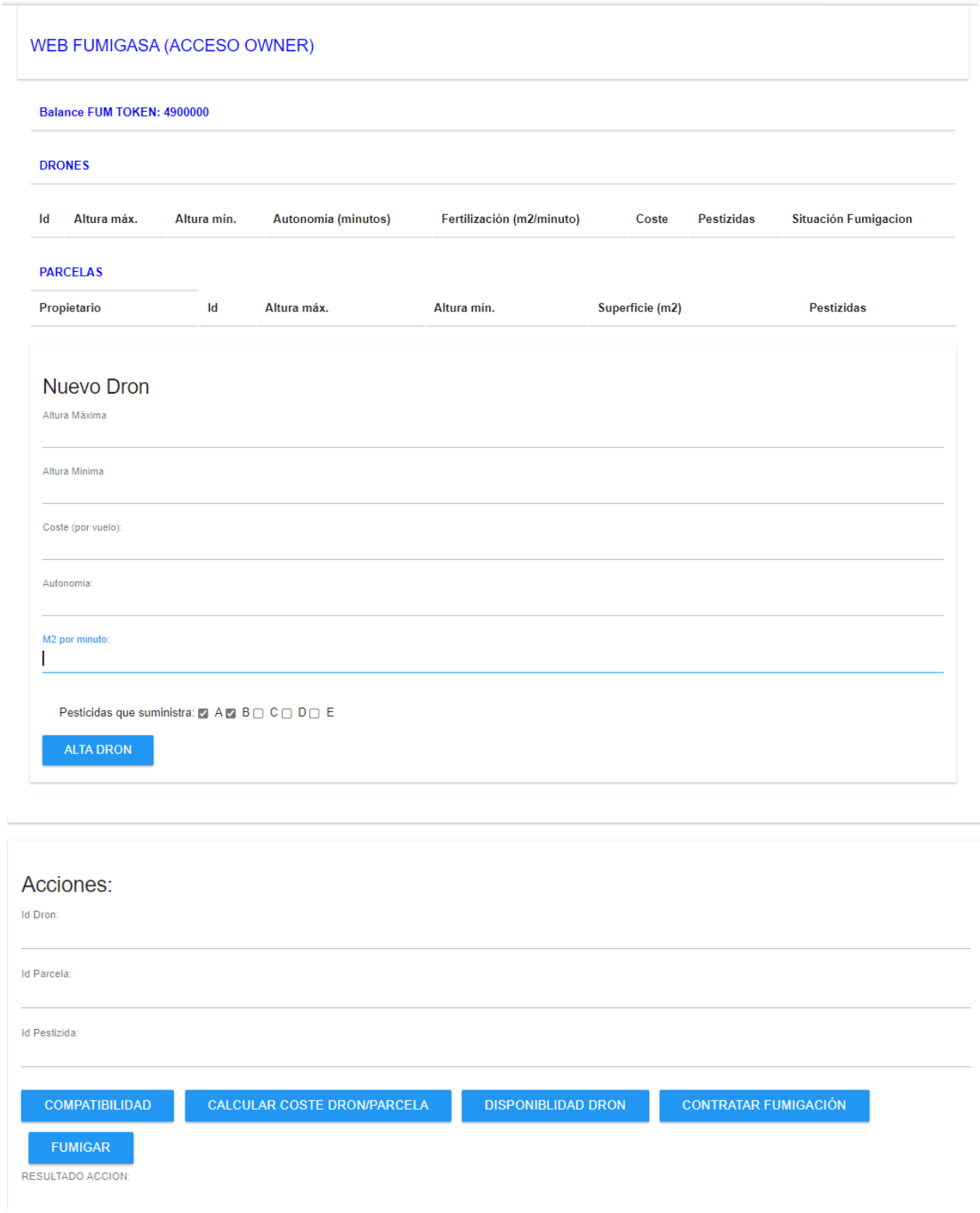


Cuenta propietario:

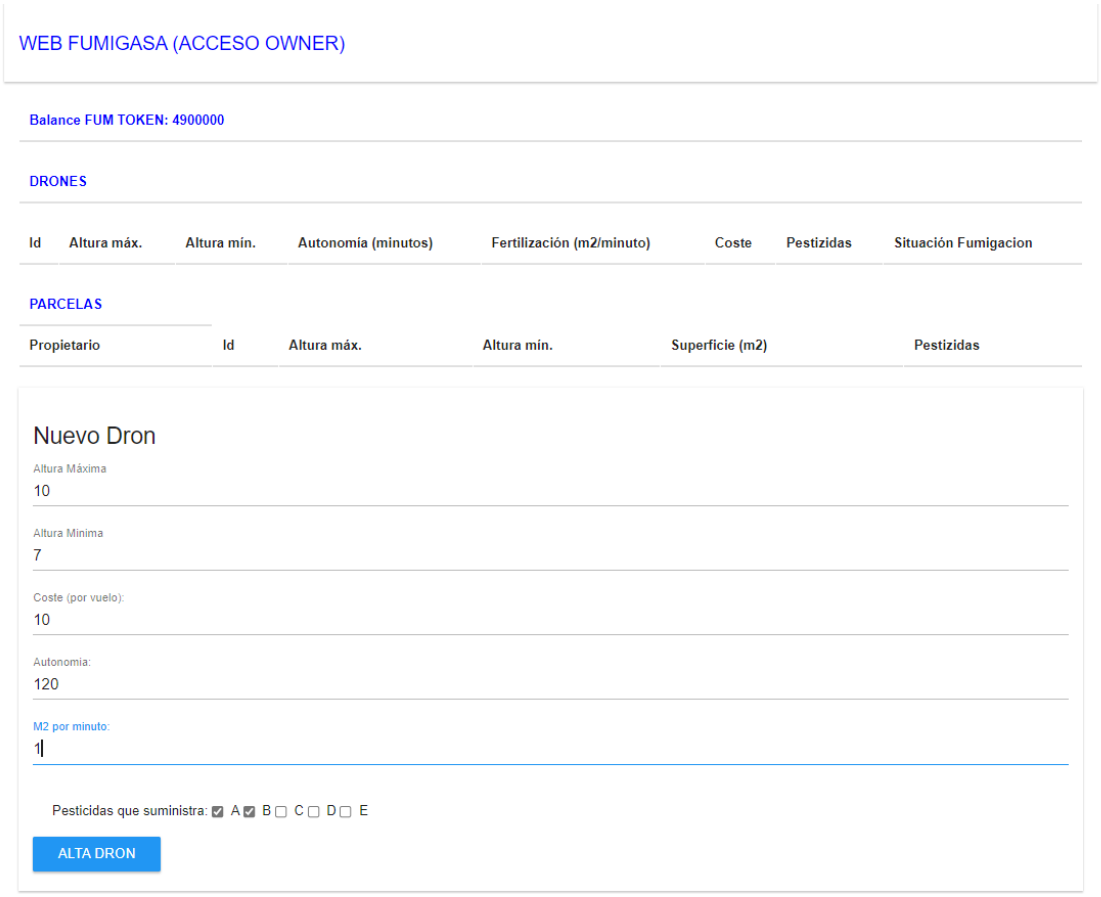




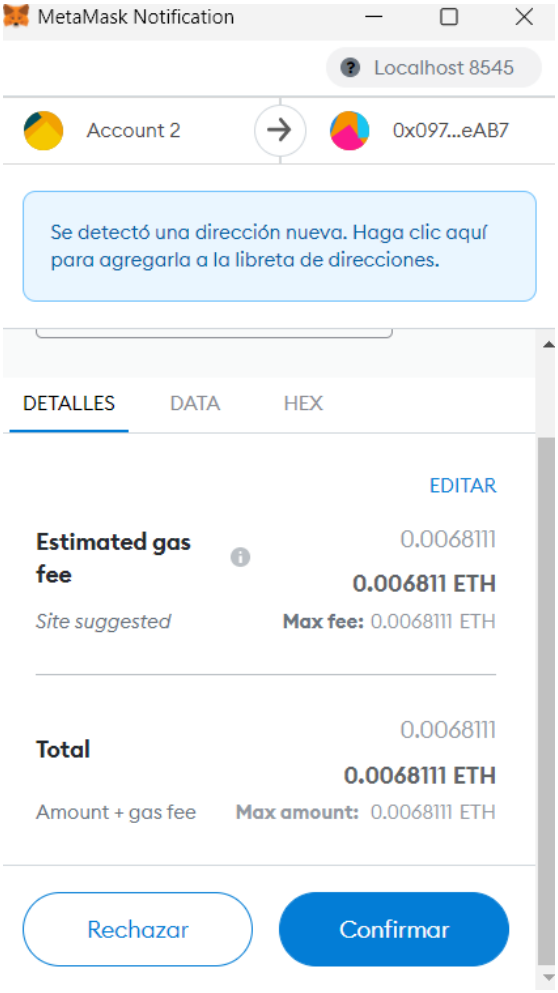
1-El owner de la empresa se conecta al sitio web especifico para administración (localhost:8000/index_owner.html) donde puede visualizar toda la lista de drones con su situación, parcelas (con sus propietarios y demás datos) y tiene diferentes operativas que puede realizar (comprobar disponibilidad Dron, comprobar compatibilidad dron/parcela, Calcular coste fumigación, asignarle trabajo a un dron (contratar fumigación) y fumigar).



2-El owner da de alta un Dron con la información requerida.

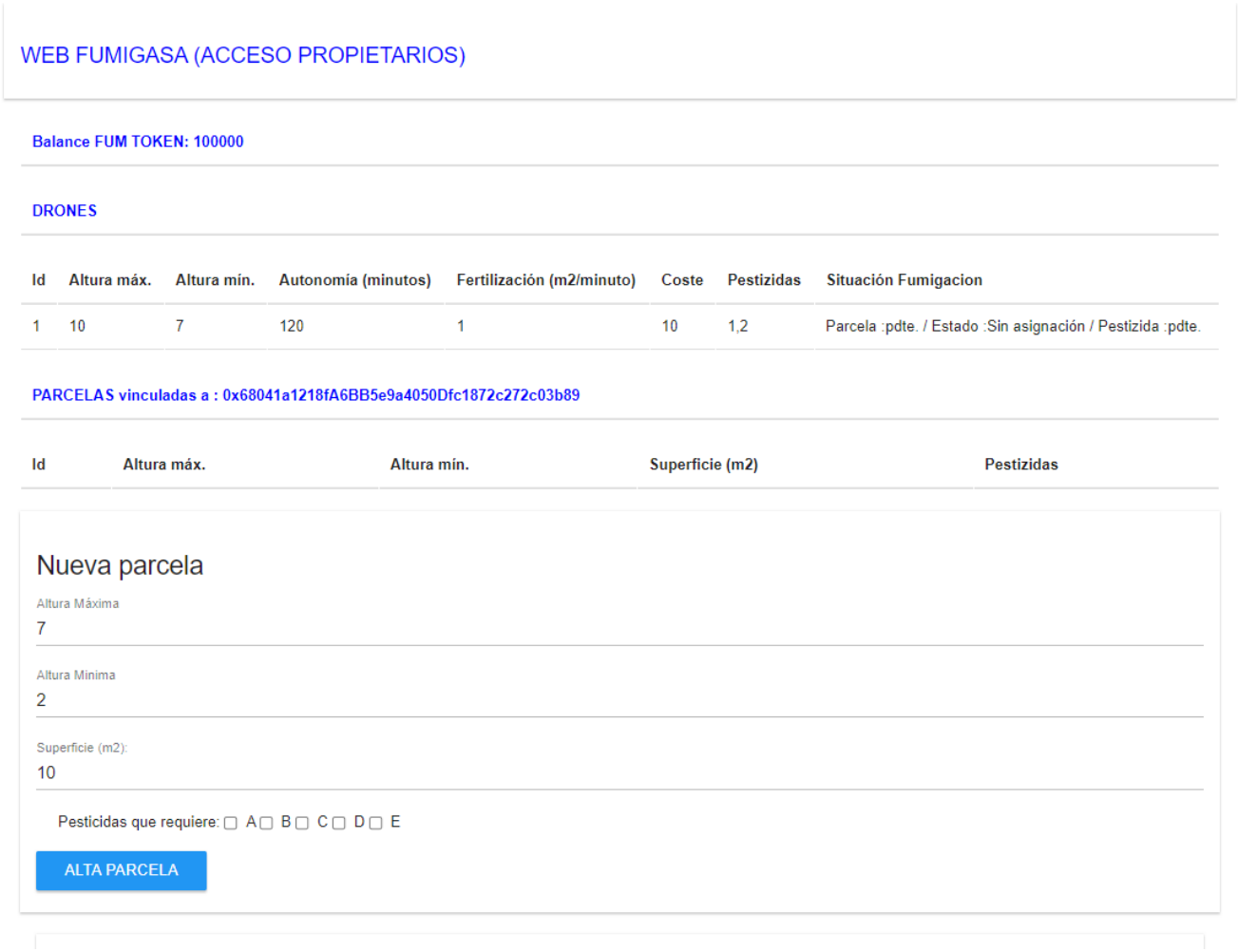


Se ejecuta la transacción y se pide autorización al usuario para ejecutar la transacción y consumir el gas correspondiente.

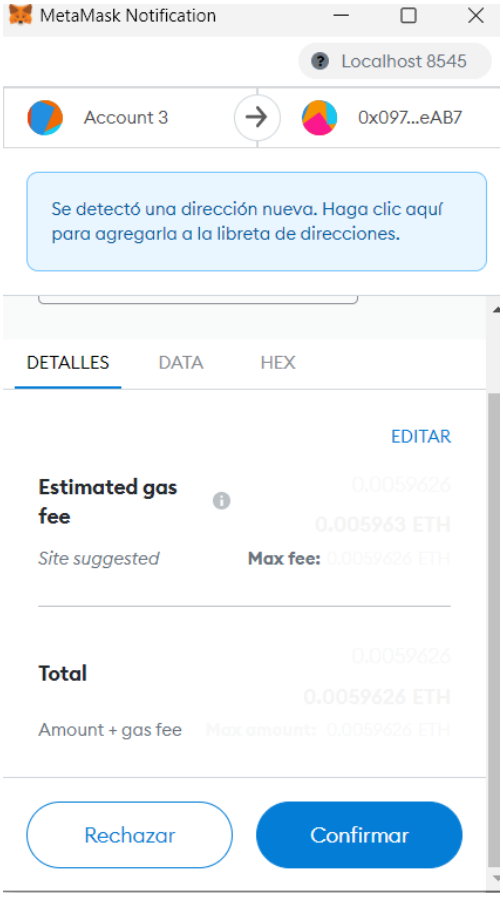


Una vez confirmada la transacción y siendo esta ok se actualiza automáticamente la pantalla para mostrar el nuevo Dron y si situación.

3-El propietario entra a la web (localhost:8000/index.html) donde puede ver el saldo de tokens que tiene, todos los drones que hay y su situación(de contratación) y las parcelas que el address activa en su wallet tiene asociados.

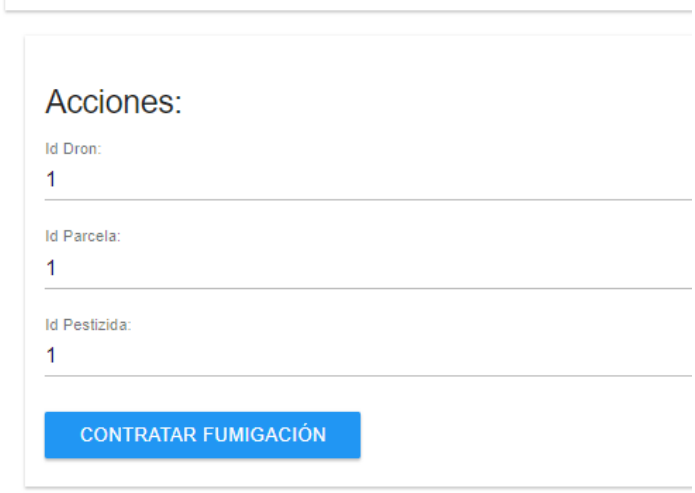


El propietario solicita el alta de la nueva parcela para lo cual consumirá un gas y requerirá autorización en su wallet.

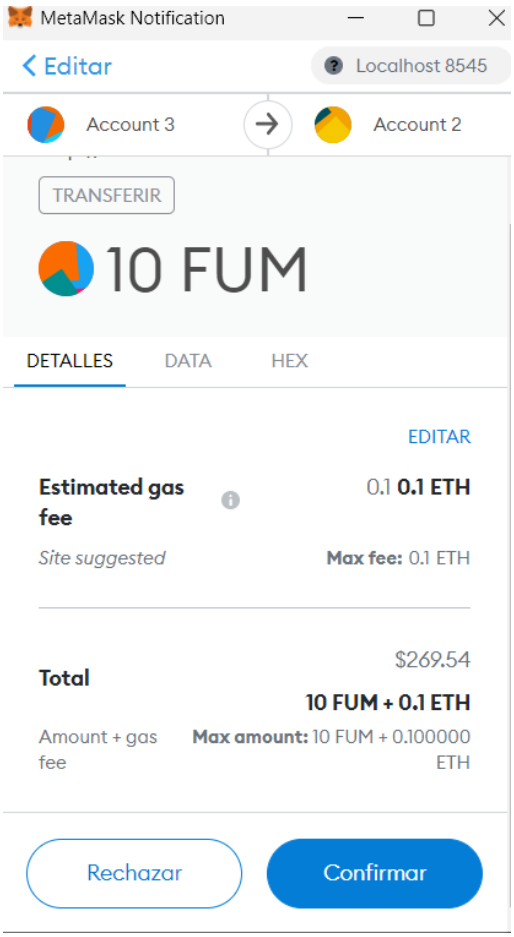


Se actualiza automáticamente la nueva parcela en la lista de parcelas del propietario:

4-El propietario solicita la contratación de la fumigación de la Parcela 1 con el Dron 1 (ambos compatibles). Se le pedirá en su wallet la confirmación primero de la transferencia de FUM tokens y luego la confirmación de la contratación (siempre que la primera haya ido bien). Informa previamente del id de dron y parcela y de pesticida con el que quiere contratar.



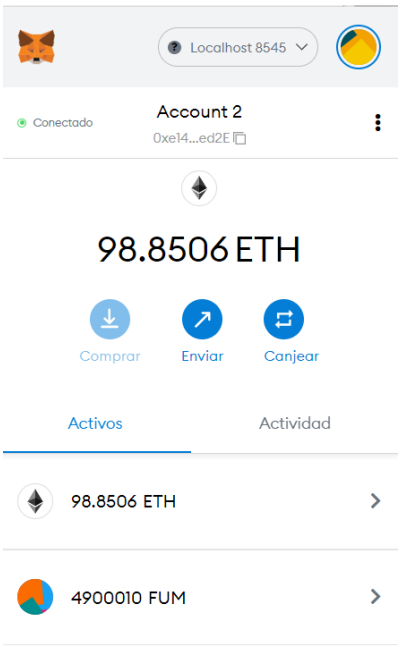
Se le presenta la confirmación de la transferencia a la cuenta del owner de la empresa. El coste son 10 FUM tokens calculados en base al coste del dron y a las veces que tiene que realizar una recarga de autonomía.



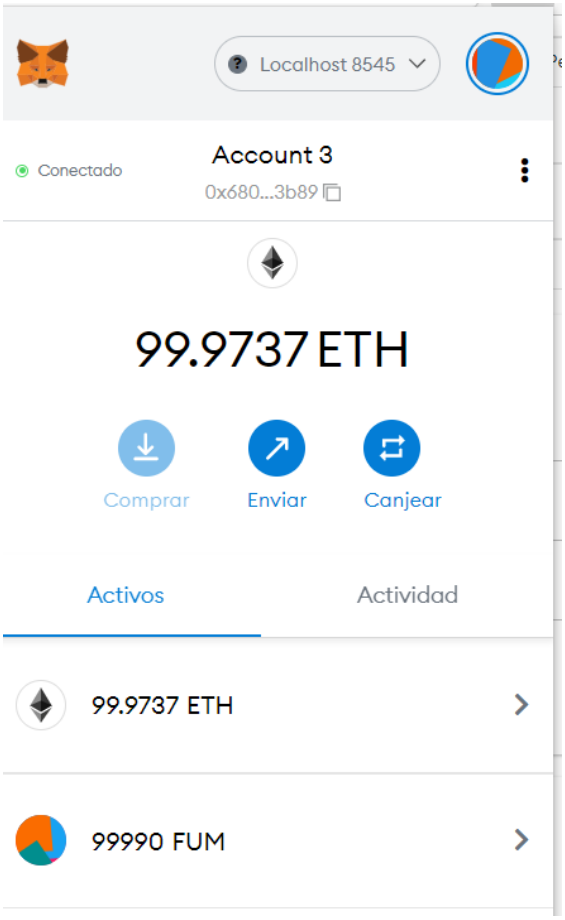
Aprobación de la transacción de contratación de la fumigación

Las cuentas ahora quedan de la siguiente manera:

Owner



Propietario



La situación del dron por pantalla ahora ha cambiado a “Fumigación activa” ya que está contratada y pendiente de que el owner de la orden de fumigar.

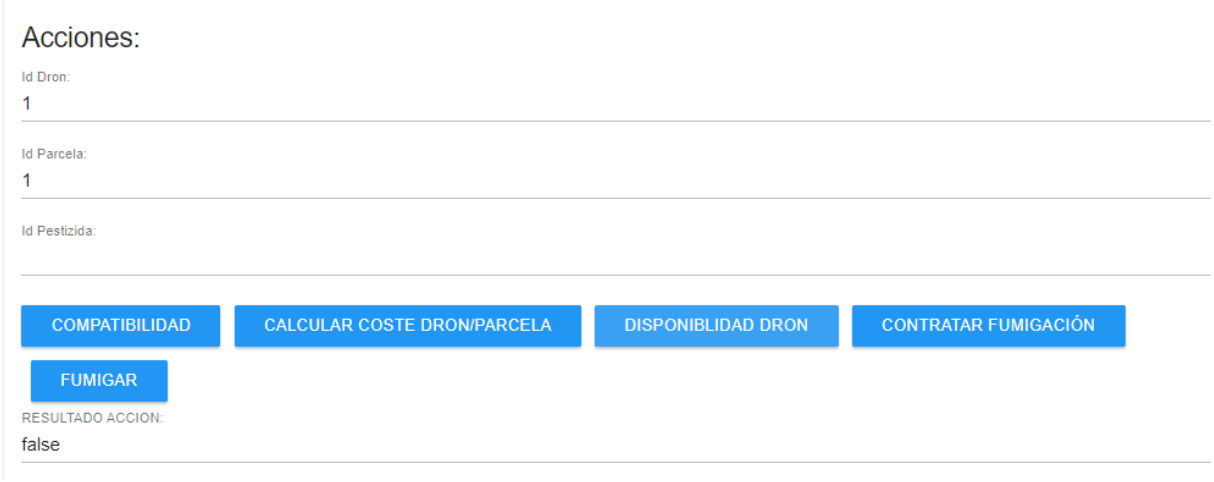
WEB FUMIGASA (ACCESO PROPIETARIOS)

Balance FUM TOKEN: 99990

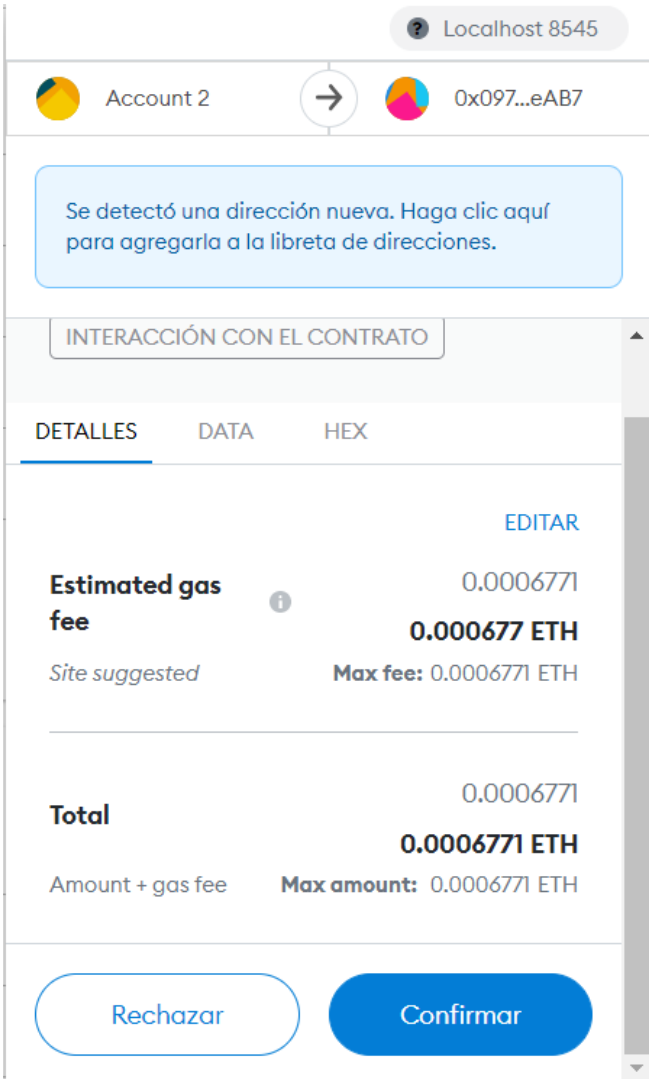
DRONES

Id	Altura máx.	Altura mín.	Autonomía (minutos)	Fertilización (m2/minuto)	Coste	Pestizidas	Situación Fumigacion
1	10	7	120	1	10	1,2	Parcela :1 / Estado :Fumigación activa / Pestizida :A

El owner comprueba disponibilidad del dron y le da como resultado que no esta disponible (false)



6-El owner da la orden de fumigar la parcela (informando la tupla dron/parcela) y se le pide autorización para ejecutar la transacción y consumir gas:



Una vez ejecutada la transacción de forma satisfactoria se muestra automáticamente por pantalla la nueva situación del dron (Fumigación finalizada) por lo que dicho dron ya queda disponible para otra contratación

WEB FUMIGASA (ACCESO OWNER)

Balance FUM TOKEN: 4900010

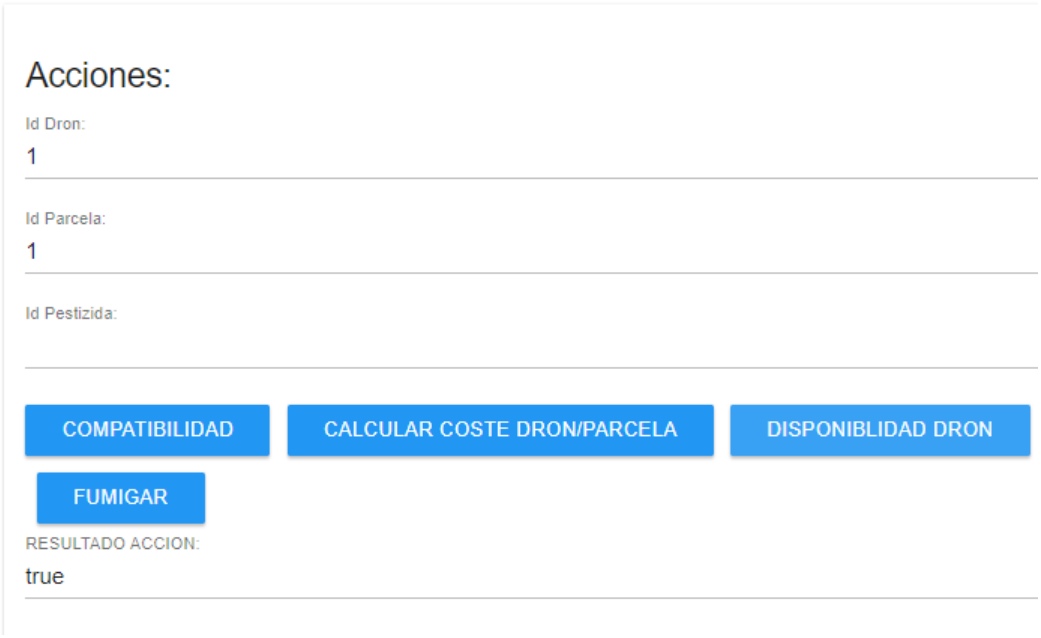
DRONES

Id	Altura máx.	Altura mín.	Autonomía (minutos)	Fertilización (m2/minuto)	Coste	Pestizidas	Situación Fumigacion
1	10	7	120	1	10	1,2	Parcela :1 / Estado :Fumigacion finalizada / Pestizida :A

PARCELAS

Propietario	Id	Altura máx.	Altura mín.	Superficie (m2)	Pestizidas
0x68041a1218fA6BB5e9a4050Dfc1872c272c03b89	1	7	2	10	1,2

El owner vuelve a consultar el estado de disponibilidad del dron y esta vez le sale disponible.



7- El propietario al entrar a la web puede ver como la fumigación se ha llevado a cabo (en la situación del dron)

5 – Conclusiones

Con la consecución del análisis, desarrollo y pruebas de esta aplicación basada en Blockchain **he logrado el objetivo inicial del mismo que era el de aplicar y llevar a la práctica todos los conocimientos que se fueron brindado durante la realización del curso**.

Al pasar por todas las etapas de desarrollo te enfrentas a diferentes cuestiones particulares que surgen al utilizar esta tecnología, lo cual facilita, en cierta medida (teniendo en cuenta que se trata de un trabajo académico), el **cambio de mentalidad necesaria** para afrontar este tipo de proyectos con los resultados esperados.

Llevo muchos años sin programar, pero siempre con el ojo puesto en las tecnologías desde el punto de vista más técnico-funcional (soy Project Manager) y el volver a estar en contacto directo con la tecnología y sumergirme de pleno en este nuevo mundo que es Blockchain **me ha generado mucha satisfacción personal y profesional**. No obstante **he pasado por momentos complicados y muy difíciles, dada la importante curva de aprendizaje** que representa el desarrollar y entender los diferentes componentes que intervienen y como conectarlos de tal forma que se lleve a cabo el desarrollo correctamente y en el menor tiempo posible.

Bajo mi simple y humilde opinión, **creo que esta tecnología ha venido para quedarse pero estamos en fases muy tempranas** y al igual que paso con internet en los 90's ahora estamos viviendo una nueva época donde hay muchas cosas por construir para facilitar el acceso no solo a los usuarios finales sino también a los desarrolladores para lograr disminuir el "time to market" de la implementación de soluciones y la barrera de entrada para abordar este tipo de proyectos.

A pesar de la dificultad, hay algo que destaco en mi caso particular que me ayudo mucho a poder realizar este trabajo y es la **gran comunidad que hay detrás de esta tecnología**. Sin ella yo personalmente no hubiera podido (al menos en el tiempo que disponíamos) llevarlo a cabo, ya que muchos de los problemas que me iba encontrando lograba solventarlos revisando en foros por internet (principalmente https://stackoverflow.com/).

Me quedo con muchas ganas de continuar aprendiendo (mi siguiente paso es continuar con este proyecto) y seguir este camino que he comenzado y que creo no tiene vuelta atrás porque cuando entras en este fascinante mundo de Blockchain ya no puedes dejarlo.