**MVP** (Minimun Viable Product

 es decir, una serie de funcionalidades imprescindibles que aseguren que el producto cumpla sus funciones vitales.

Comienzo pensando en la taxonomía

Esta taxonomía está destinada a ayudar con importantes arquitectura

consideraciones sobre los atributos de rendimiento y calidad

(por ejemplo, disponibilidad, seguridad y rendimiento) de sistemas basados ​​en blockchain.

Porque blockchain?

La institución pone un ejemplo: en el 2019 un **dron entregó un riñón que luego fue trasplantado con éxito en la Universidad de Maryland, en 10 minutos**, realizando un recorrido de 2.6 millas, y que**en automóvil, se hubiese tardado entre 15 y 20 minutos**.

El informe resalta que **no es el tiempo el único problema, sino que el hospital receptor no conoce normalmente el progreso del órgano**, lo que puede llevar a retrasos innecesarios,**es aquí donde la tecnología blockchain permitiría el acceso a la información de todas las partes involucradas**.

La confiabilidad y la confianza son esenciales para el uso de máquinas voladoras automatizadas Aquí es donde entra en juego la cadena de bloques y brinda soluciones a las inquietudes planteadas.

Seguimiento de los patrones de vuelo de los drones, así como el análisis de datos en tiempo real, para permitir que el dron adapte su plan de vuelo de acuerdo con la ruta más segura hasta su destino.

El token ERC20 lo implemento con la **UTILIDAD** de financiar el proyecto.

El **valor subyacente** (tiene valor para una cierta gente?) En que subgrupo tiene valor

**Estado legal** del token

**Capa técnica** en la que se implementa

**Propósito** – cual es propósito de mi token?

Tokenomics?

Token – Representación virtual del valor de un activo

El valor de los tokens fungibles recide en cuantos tokens tengo de esos y no en las caracteristicas de cada uno

Abstracción y nivel conceptual del problema a resolver

**¿Quién usará el sistema?**

Los actores que utilizarán el sistema serán:

-Propietaria de las parcelas

-Empresa que suministra el servicio ofrecido (riego de parcelas a través de Drones)

**Diagramas de casos de uso (User Stories)**

**Diagramas de secuencia (como pasan los mensajes de unos a otros)**

Temporal

Instancias

Diagrama de estados en que puede estar el contrato a lo largo del tiempo

(dentro de un propio contrato)

Que componentes tendrá el sistema?

Objetos –> Diagrama de clases

Herencia , asociaciones (una clase llama a otra clase)

(los atributos no deben ser objetos, usar relaciones)

Patrones utilizados?

Como funcionará el sistema?

En el tiempo 🡪 Diagrama de secuencia

En el espacio 🡪 Diagrama de estados

Donde vamos a poner el sistema?

Estructura –> Diagrama de Despliegue

Arquitecutra Eth Alastria…etc

Que alguien que lo vea entiendo como hacer el despliegue de nuestros Smart contracts. Por ejemplo si alguno de los conratos hablan con algún actor externo habría que ponerlo

**IDEAS**

[API](https://academy.binance.com/en/glossary/application-programming-interface) para que los usuarios creen sus propios tokens no fungibles y codifiquen [metadatos](https://academy.binance.com/en/glossary/metadata) (imágenes, descripciones, etc.).

Hola

He comenzado a definir el sistema mediante las metodologías que nos han explicado en el curso y en base también a mi experiencia (hace tiempo que no programo ni diseño).

Me gustaría que me dieras tu feedback para ver si voy bien orientada.

Mi idea es crear un MVP básico para luego de la entrega continuar enriqueciéndolo pero dado el tiempo que disponemos me veo en la necesidad de acotar las funcionalidades.

Tal como lo veo yo diseñaría un sistema que tuviera:

**Actores:**

-Empresa de fumigación (a partir de ahora FUMIGAS)

-Propietario de las parcelas

**Casos de uso (que contemplo implementar en este MVP):**

|  |  |  |
| --- | --- | --- |
| **Caso de uso** | **Actores** | Observaciones |
| Crear nuevo dron | Fumigas | Se crean y luego se marcan como “contratables” |
| Habilitar dron para contratación | Fumigas |  |
| Registrar nueva parcela | Propietario y Fumigas | Fumigas valida/acepta dicha parcela |
| Contratar riego | Propietario/Fumigas | Validaciones:  -Altura máx. parcela vs altura máxima dron  -m2 parcela vs autonomía dron (desde origen)  -Que propietario no tenga deudas previas |
| Calcular compatibilidad dron con parcela | Fumigas | Valida que dicho dron pueda regar una parcela en cuestión |
| Pagar servicio | Propietario y Fumigas |  |
| Obtener listado de drones | Fumigas |  |
| Obtener listado de parcelas | Propietario/Fumigas |  |
| Obtener lista/situación de contrataciones | Fumigas |  |
| Obtener listado de pesticidas | Fumigas |  |
| Mostrar situación balance (saldo) | Fumigas |  |

**Que componentes tendrá el sistema:**

**Diagrama de clases**

He realizado un principio de diagrama de clases que he identificado:

<https://gitmind.com/app/flowchart/e578713888>

Password: 2746

**Aplicación WEB**

Interactuará con los contratos que daré de alta en la testnet.

Había pensado en hacer una web que tuviera un apartado **para la propia empresa Fumigas** donde esta pudiera:

-Ver contrataciones activas y gestionarlas

-Ver saldo

-Ver y gestionar validación de alta de parcelas

-Ver situación drones y dar de alta nuevos

Otro apartado para los propietarios de las parcelas (se identificarán con un código y podrán ver las contrataciones que tienen y realizar nuevas visualizando los diferentes drones disponibles y sus características).

Que es lo que NO contemplare en esta fase (si me da tiempo si claro) y que quedara como evolución del sistema:

-Gestión de usuarios (controlar que el acceso a la web solo de aquellos propietarios que están identificados en el sistema)

-Todo lo que sea comandar el dron para ejecutar la operación de riego y desplazamiento

-Desvincular un dron de una parcela

-Identificación de los actores a través de protocolos de identidad digital

**Metamask**

Se utilizará metamask como billetera para probar el pago de la contratación con el nuevo token.

**REQUERIMIENTOS PREVIOS**

Installa NS

Installa Truffle

NPM Install openzeppelin

En truffle

Truffle compile

Truffle migration

Truffle deploy

Diagrama

Descripción generada automáticamente con confianza media

var erc20Instance = new web3.eth.Contract(abi,Token\_address);

erc20Instance.methods.approve(contractAddress, amount).send({from: userAddress},

function(err, transactionHash) {

//some code

});

0

Las asignaciones no almacenan sus claves, solo el valor que se almacena en la dirección de la memoria de estado. Para obtener la lista de datos, utilice una matriz.

address[] public addresses;

ahora defina una función para obtener la longitud de esta matriz:

function getAddressCount() public view returns(uint){

return addresses.length;

}

Defina también una función para obtener el elemento por índice:

function getAddressByIndex(uint index) public view returns(address){

return addresses[index]

}

ahora necesita escribir código para extraer el obtener la matriz uno por oen. Así es como se hace en javascript con librería:web3

let addresses,addressCount;

try {

addressesCount = await ContractName.methods.getCampaignCounts().call();

addresses = await Promise.all(

Array(parseInt(addressesCount))

.fill()

.map((element, index) => {

return ContractName.methods.getAddressByIndex(index).call();

})

);

} catch (e) {

console.log("error in pulling array list", e);

}