Descentralized Voting System

Índice

Objetivo de la aplicación

El objetivo es crear un sistema de votos descentralizado, utilizando tecnología blockchain , mediante el cual los seguidores del canal puedan votar de manera totalmente segura y transparente el contenido y el rumbo del canal.

Para conseguir este objetivo, vamos a crear un sistema de votos descentralizados, desplegado sobre la red de polygon, mediante el cual poder realizar las votaciones de una manera totalmente transparente, segura y confiable.

Para facilitar el voto, el sistema vendrá acompañado de una interfaz de usuario, la cual enmascara todos los detalles al usuario, facilitando así el uso de la aplicación

StakeHolders

Los stakeholders del proyecto son :

***Suscriptor/Seguidor :** Es cualquier persona que es seguidora del canal, tanto en youtube como en las redes sociales, y que puede proponer ideas para ser votadas y posteriormente participar en la votación. Son los que se ven más afectados por las mismas, ya que dependiendo del resultado de las votaciones se subirá un contenido u otro.*

***Votantes :** Un votante es cualquier persona, seguidor o no, que vota por una de las opciones que se encuentran disponibles en la aplicación. El votante previamente habrá adquirido tokens de la aplicación y realizará los votos necesarios*

***Administrador** : El administrador será la persona propietaria de la aplicación, que se encargará de seleccionar las opciones a ser votadas, iniciara una votación y anunciará el resultado de la misma. Pueden haber uno o más administradores*

***Nodos :** Los nodos serán los computadores de la red blockchain, que se encargará de enviar las peticiones a la red y de verificar las transacciones para garantizar la veracidad de las mismas*

***Mineros :** Su función principal será la de introducir las transacciones que se emitan en el sistema de votación en un nodo, para que así esta quede registrada de manera inmutable en la blockchain*

Requisitos de Usuario

Una vez tenemos identificado el objetivo y los diferentes stakeholders del proyecto, es hora de empezar a recopilar los requisitos de la aplicación, para ello empezaremos recopilando requisitos de usuario en forma de user stories.

Las user stories, son una descripción a alto nivel de los requisitos que la aplicación tienen que cumplir, en esta descripción no entramos en detalles técnicos sinó que simplemente tiene por objetivo

el recopilar las funcionalidades que la aplicación debería incluir para cumplir con las necesidades de los diferentes stakeholders.

User Story 1

Como seguidor, me gustaría poder hacer propuestas sobre el contenido qué ver en el canal, para así poder someterla a votación.

User Story 2

Como Administrador, me gustaría seleccionar las opciones que más me gusten y someterlas a votación

User Story 3

Como votante, una vez elegidas las opciones finales, me gustaría votar por la que más me guste.

User Story 4

Como Administrador, me gustaría poder poner una fecha límite de votación, para así pasado un cierto tiempo poder parar la votación y saber cual es la opción más votada

User Story 5

Como Seguidor, me gustaría poder ver el estado actual de las votaciones, así como poder visualizar los votos, para que el proceso sea lo más transparente posible

User Story 6

Como votante, me gustaría que mi voto sea anónimo y no se pueda vincular a mi identidad, para así poder proteger mi privacidad

User Story 7

Como Usuario, me gustaría ver de manera transparente y clara el resultado de las votaciones, junto con la opción ganadora

User Story 8

Como votante, me gustaría que mi voto sea ponderado al número de tokens que tengo, para así si una opciones me gusta mucho, hacer que tenga más opciones de ser elegida

User Story 9

Como Usuario, me gustaría poder comprar tokens de la aplicación de manera cómoda y transparente, para así poder realizar las votaciones

Requisitos de Sistema

Una vez definidas las historias de usuario, procederemos a definir los requisitos de sistema,
Estos son una descripción más detallada de los requisitos de usuario, donde describimos aspectos técnicos relacionados con la implementación de las funcionalidades descritas en las user story.

Etiqueta	RFS-Administrador-01
Título	Añadir opciones de votación
Descripción	Una vez se han realizado las diferentes propuestas, y el administrador ha seleccionado las que se van a someter a votación, este deberá introducirlas en la aplicación para que así se pueda proceder a su votación. Para ello, la aplicación provee al administrador de una interfaz en la cual podrá introducir las opciones para ser votadas. El proceso será el siguiente : ❖ El administrador selecciona las opciones que más le gustan ❖ Las introduce en la aplicación : ➢ Se guarda el nombre completo en la base de datos ➢ Se asocia cada nombre con un id de opción Por lo general siempre habrá 4 opciones para elegir, en caso de que este número cambie el smart contract ofrecerá las funciones necesarias para adaptarse a la votación actual.
Objetivo	El objetivo es iniciar una votación, con las opciones que el administrador haya elegido
Dependencia	

Etiqueta	RFS-Administrador-02
Título	Creación de la votación
Descripción	Una vez se han añadido las opciones de votación se procede a la creación de la misma para ello ❖ Reiniciamos los votos asociados a cada id ❖ Especificamos el número de opciones para adaptar los rangos de id aceptados para esta votación ❖ Especificamos la fecha de fin de la votación
Objetivo	El objetivo es permitir al administrador iniciar una nueva votación
Dependencia	RFS-Administrador-01

Etiqueta	RFS-Seguidor-01
Título	Seguimiento de la votación
Descripción	La aplicación tiene que permitir que los usuarios puedan visualizar el estado actual de las votaciones, para ello la aplicación deberá mostrar las opciones, junto con el número de votos que ha recibido cada una, además deberá ofrecer la opción de poder visualizar una vista más detallada, donde muestre los votos que ha realizado cada votante
Objetivo	El objetivo permitir realizar un seguimiento de las votaciones
Dependencia	RFS-Administrador-02

Etiqueta	RFS-Tokens-01
Título	Comprar Tokens
Descripción	La aplicación deberá permitir a los usuarios comprar tokens, los cuales les den derecho a realizar votos. Además de comprar tokens, la aplicación deberá permitir que un usuario pueda devolver tokens a la aplicación y que le devuelva el dinero que se ha gastado en los tokens que no ha utilizado.
Objetivo	El objetivo es tener un sistema de tokens, de manera que los votos puedan ir ponderados a la cantidad de tokens que tiene el usuario
Dependencia	

Etiqueta	RFS-Votante-01
Título	Realizar votos
Descripción	La aplicación deberá permitir que cualquier usuario que mande tokens pueda realizar una votación, para ello en la interfaz de la aplicación ,el votante va a elegir la opción y mandar un token, lo cual le dará derecho a realizar la votación, en nuestro caso los votos no van a ir ponderados al número de tokens, sinó que solo se podrá mandar 1 token por usuario que representa 1 voto.
Objetivo	El objetivo es permitir a un usuario votar
Dependencia	RFS-Tokens-01

Diseño Smart Contract Tokens

Necesitamos diseñar un smart contract que nos permita la distribución y creación de tokens ERC 20, los cuales darán derecho a voto.

Una opción podría ser que los votos sean ponderados al número de tokens que tengas, así cuantos más tokens mandes a la plataforma, mayor peso tendrá tu voto, esto podría ser injusto y podría provocar situaciones en las que los que tengan más tokens sean los que decidan el rumbo del canal, para ello en una votación una misma dirección solo puede votar 1 vez y para ello necesita mandar 1 token.

Inicialmente, todos los tokens son propiedad del smart contract, el cual se encargará de distribuirlos a aquellas direcciones que quieran comprar tokens.

Cuando un usuario quiere comprar un token, este mandará una cantidad de matic, en concreto cada token se vende a 0,2 matic + 0,02 matic para la plataforma.

Una vez comprado el token, en cualquier momento antes de ser usado se puede devolver a la aplicación, para ello será necesario que el smart contract tenga matic suficiente para reembolsar a todas aquellas cuentas que aún no se hayan gastado los tokens.

Cuando una cuenta decida realizar un voto, se consultará el saldo de esa cuenta y se mandará el token al smart contract el cual lo podrá volver a vender a posteriores usuarios.

Para implementar este token, no haría falta usar el estándar ERC 20, podriamos programar las funciones nosotros mismos sin necesidad de heredar todo el estándar, sin embargo, para facilitar que wallets como meta mask puedan reconocer nuestro token, es recomendable usar dicho estándar

Contrato MyToken

Este contrato hereda de ERC20 y de ReentrantGuard y representa la implementación de los tokens ERC 20 de la aplicación.

Cómo hacemos una herencia de ERC 20, el contrato automáticamente incorpora todas las funciones definidas en el estándar ERC 20, en nuestro caso hemos decidido sobreescribir las funciones transfer.transferFrom y approve para manejar un registro de suscriptores.

Consideramos suscriptor cualquier dirección que tenga, o haya tenido tokens alguna vez, podemos observar que en el smart contract tenemos una función que permite al owner regalar tokens a un suscriptor, el objetivo del mapping es precisamente ese, que en algún momento determinado, el owner pueda regalar un token a los suscriptores.

Nuestro token, usa 18 decimales, que son el mismo número de decimales que usa el estándar ERC 20, esto quiere decir que un token se representa con 1000000000000000000.

Cuando llamamos al constructor todo el balance, es destinado al smart contract, que será el responsable de vender los tokens a los usuarios.

Se establece un precio, 0,1 matic por cada token, ***en el constructor el valor se debe especificar en wei***

Observemos como la gestión de la venta de tokens, la podríamos realizar en otro contrato externo, conceptualmente sería una buena opción, ya que tendríamos un diseño más modular, pero tendríamos los siguientes problemas :

- *A la hora de comprar tokens, vender token y ver el balance de tokens, deberíamos realizar llamadas constantes a otro smart contract, lo cual implica un mayor gasto de gas*

- *Para realizar la devolución de tokens, es decir un usuario quiere devolver sus tokens y recuperar su dinero, el usuario tendría que ejecutar la función approve, y autorizar al contrato el uso de los tokens que el usuario quiere retirar*

Tenerlo en contratos diferentes también tiene sus ventajas :

- *Permite tener un diseño más modular y más legible, ya que no mezclamos funcionalidades diferentes en un mismo smart contract.Además si se detecta un fallo en uno de los contratos, es suficiente cambiar uno de ellos y el otro no sufre ningún cambio*

- *Tenemos contratos más pequeños, por lo tanto más fáciles de manejar y con un menor coste de deployment*

En nuestro caso, he optado por fusionar los smart contract en uno solo, ya que sino el número de llamadas externas que teníamos que realizar era muy grande, y así la interacción entre el usuario y la aplicación será mucho más simple

En el smart contract MyToken, además de las funciones propias del token ERC 20 encontramos :

buyToken : *Es la función que ejecutará un suscriptor cuando quiera comprar tokens de la aplicación, para así tener derecho a voto. El usuario mandará una cantidad de matic, se comprueba que la cantidad de matic que se manda es suficiente y que el sistema dispone de tokens suficientes para realizar la operación y realiza la transferencia, antes de terminar la operación comprobamos si hay que devolver cambio o no.* ***Amount es un valor que viene expresado en wei.***

```
function buyToken(uint256 amount) public payable nonReentrant {
       require(getBlanceContract()>=amount,"Shop don't have enough tokens");
       uint256 total_price = (amount*price)/10**18;
       require(msg.value>=total_price,"Money you are sending is not enough");
      _transfer(address(this),msg.sender,amount);
       uint256 return_value=msg.value-total_price;
       if(return_value>0){
           payable(msg.sender).transfer(return_value);
       }
       if(!is_sub[msg.sender]){
          is_sub[msg.sender]=true;
       }
       emit PurchaseTokens(msg.sender,amount);
   }
```

returnToken : *Esta función tiene por objetivo devolver tokens y recuperar el dinero que se pagó por el. La función verifica, que el sender tenga tokens suficientes para devolver, se verifica que el smart contract tenga dinero suficiente para pagar por los tokens y se realiza la transferencia de tokens desde el sender hasta el smart contract y se realiza la transferencia de matic desde el smart contract hacia el sender.* ***Amount es una cantidad expresa en wei***

```
function returnToken(uint256 amount) public nonReentrant {
       require(balanceOf(msg.sender)>=amount,"You don't have enough Tokens");
       uint256 total_matic = (amount*price)/10**18;
       require(getBalanceContractMatic()>=total_matic, "Smart contract don't have
enough money");
       _transfer(msg.sender,address(this),amount);
       payable(msg.sender).transfer(total_matic);
       emit ReturnTokens(msg.sender,amount);
   }
```

removeMatic *: Es una función que solo puede ser ejecutada por el propietario del contrato y tiene por objetivo recuperar matic para él owner. Es importante que el smart contract disponga de fondos suficientes para poder devolver el dinero a todos aquellos usuarios que tengan tokens, para ello :*
Obtenemos el balance del smart contract
Obtenemos los tokens en circulación ,es decir todos aquellos tokens que no son propietarios del smart contract, como tenemos 18 decimales, dividimos ese valor entre 10^18 para tener un número entero.
Calculamos el dinero que necesitamos para reembolsar tokens, para ello multiplicamos el número de tokens en circulación por el precio de cada token
Obtenemos el dinero restante, restando el balance menos el valor anterior.

*Veamos un ejemplo : Supongamos que tenemos 1,2*10^18 wei, 7 tokens en circulación y cada token vale 0,1 * 10^18 wei.*

*El dinero que necesitamos para devolver los tokens es ((7*10^18)*(0,1*10^18) /1*10^18) = 7*10^17 = 0,7*10^18 wei deben quedar en el smart contract, por si nos quieren devolver los 7 tokens que tienen los suscriptores, es decir necesitamos 0,7 matic en el smart contract.*

*Por lo tanto, el owner puede recuperar 1,2*10^18-0,7*10^18 = 0,5*10^18 = 0,5 matic y 0,7 quedarán en el smart contract*

```
function removeMatic() public onlyOwner nonReentrant {
       uint256 total_balance=address(this).balance;
       uint256 total_return=(getCirculationTokens() *price) /10**18;
       uint256 total_for_remove=total_balance-total_return;
       require (total_for_remove>0, "There's not enough Matic");
       payable(owner).transfer(total_for_remove);
   }
```

Podemos ver, como así el smart contract siempre dispone de fondos suficientes en caso de que un suscriptor quiera devolver un token.

<u>Contrato PollingStation</u>

Este contrato es el encargado de gestionar las votaciones y recibir las votaciones de los votantes.
En nuestro caso, solo vamos a votar opciones, la descripción de cada opción la tendremos en una base de datos separada, esto es así ya que si queremos guardar nombres y descripciones en un smart contract puede ser muy caro de mantener, y si además estos datos se tienen que ir cambiando de manera periódica, el coste puede ser aún superior.

Por eso a nivel de smart contract únicamente votamos opciones, a nivel de aplicación, votaremos una propuesta acompañada de una descripción y vinculada a una opción.

A continuación veamos las variables más importantes junto con el uso que tienen :

- ***voters :*** *Es un array que contiene a todos los los votantes que han participado en la votación actual, el objetivo de este array es cuando se inicia una nueva votación, poder limpiar el mapping has_voted e iniciar una nueva votación. Notemos cómo cada vez que se crea una votación este array es eliminado, ya que no nos interesa mantener un historial en la blockchain de todos los votos que se han realizado, para ello tenemos un evento que nos permitirá realizar dicho seguimiento.*
- ***options*** *: Es un array, que representa los votos de cada opción*
- ***min_index*** *: Indica el índice de la primera opción, siempre será 1, por lo tanto como los array están indexados por 0, deberemos normalizar el voto que nos llegue por parámetro, restándole min_index.*
- ***max_index*** *: Representa la opción más alta que se permite votar, generalmente siempre tendremos 4 opciones, aunque según la votación este índice puede variar.*
- ***votation_id*** *: Indica el índice de la votación actual, la primera votación tendrá el índice 1 y así sucesivamente, el objetivo de esta variable es emitir un evento, que nos permitirá realizar un seguimiento de las opciones que se han votado.*
- ***tokens_for_vote*** *: Representa el número de tokens que el votante tiene que mandar para poder emitir 1 voto, generalmente este valor será 1000000000000000000, que representa 1 token.*
- ***has_Voted :*** *Es un mapping que nos indica si una cuenta ha votado, el objetivo de este mapping es evitar que un votante pueda votar más de 1 vez.*

También se ha emitido un evento, que será de gran importancia para realizar el seguimiento de la aplicación , es el evento NewVote, que nos permite saber en cada votación el voto que ha emitido cada wallet. Notemos como la privacidad está garantizada, la identidad de la persona que realiza la votación está asegurada, excepto si esa persona decide revelar su identidad y asociar esa dirección a su identidad, la anonimización de los votantes y de la opción elegida será el siguiente paso de la aplicación, para ello será necesario introducir el concepto de firmas ciegas.

Veamos a continuación las funciones más importantes del smart contract :

setVotation *: Esta función sólo puede ser ejecutada por el owner de la polling station, y tiene por objetivo iniciar una nueva votación, como parámetros recibe :*

- *El número máximo de opciones que se podrán elegir en la votación*
- *La fecha límite de la votación*

Reiniciamos el array de votaciones adaptado al número máximo de opciones de la votación actual.
Como en solidity no podemos reiniciar un mapping, debemos iterar sobre todas sus claves para reiniciar sus valores, para ello lo que hacemos es iterar sobre el array voters, que contiene las

direcciones de todos los votantes de la votación anterior y ponemos el valor has_vote a false, así permitimos que si quiere votar otra vez lo podrá hacer.
Finalmente reiniciamos el array voters, ya que en esta nueva votación de momento no hemos tenido ningún voto e incrementamos votation_id, indicando que empezamos una nueva votación.

```
function setVotation(uint256 _max_index, uint256 _endVotation) public onlyOwner
nonReentrant {
        max_index=_max_index;
        options=new uint256[](max_index);
        for(uint256 i=0;i<voters.length;i++){
            has_voted[voters[i]]=false;
        }
        votingEndTime = block.timestamp + (_endVotation * 1 minutes);
        delete voters;
        votation_id++;
    }
```

Vote : *Esta función tiene por objetivo emitir un voto, recibe por parámetro la opción que se quiere votar, antes de realizar el voto, se realizan las siguientes comprobaciones :*

- *Se verifica,que la opción sea válida, es decir que esté en el rango de min_index, max_index.*
- *Se verifica que el votante no haya votado anteriormente en esta votación.*
- *Se verifica que el votante dispone de fondos suficientes para realizar la votación, importante que desde el contrato MyToken, autorice a la polling station a votar.*

Una vez se se han hecho todas las comprobaciones :

- *Se transfieren fondos hacia el contrato MyToken*
- *Se realiza la votación*
- *Guardamos al votante en el array voters*
- *Indicamos que ha votado*
- *Emitimos el evento newVote, indicando el id de la votación, la dirección del votante y la opción*

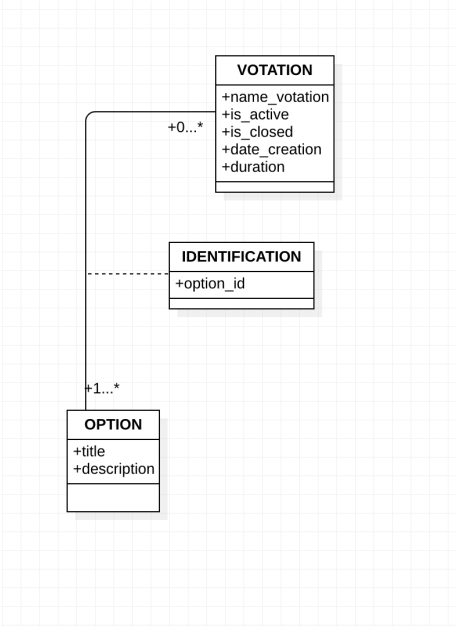
```
function vote(uint256 option) public votingOpen nonReentrant {
        require(option >= min_index && option <= max_index, "Invalid option");
        require(!has_voted[msg.sender],"You just can vote once");
          require(token.allowance(msg.sender,address(this)) >= tokens_for_vote, "You
don't have enough tokens");
        token.transferFrom(msg.sender,token_address,tokens_for_vote);
        options[option - min_index]++;
        voters.push(msg.sender);
        has_voted[msg.sender]=true;
        emit newVote(votation_id,msg.sender, option);
    }
```

El resto de funciones son triviales, así que no hace falta que sean documentadas.

Ahora que hemos explicado los smart contract con detalle, procederemos a explicar la implementación de la base de datos de soporte, para poder brindar a la aplicación una funcionalidad completa.

Diseño Base de Datos

El objetivo de la base de datos, es liberar a la cadena de bloques de almacenar descripciones y strings, a la vez que minimizamos el número de llamadas a la blockchain aumentando así la performance de la aplicación.



Hemos supuesto que una opción puede aparecer en más de una votación, ya que es posible que una vez realizada la votación, una opción la queramos volver a someter a votación en un futuro.

Podemos observar, como la información que tenemos en la clase Votation, también la tenemos en la blockchain, pero a nivel conceptual lo necesitamos tener en la base de datos para vincular un voto a una opción y a una votación.

Otro enfoque sería una vez realizada la votación ,eliminar todos los registros y reiniciar el sistema de nuevo, también sería un enfoque válido, pero en nuestro caso nos interesa mantener una constancia de las votaciones que se han realizado en el sistema.

Con este enfoque, evitamos que el sistema este sobrecargado, evitamos almacenar todo el historial en la blockchain, a la vez que contamos con las ventajas de la tecnología blockchain, ya que todas las transacciones que genere la aplicación quedarán almacenadas de manera permanente e inmutable en la blockchain, para que cualquiera lo pueda verificar.

Notemos cómo, en cada momento solo podremos tener una votación activada, para ello en el momento en el que se cree una nueva votación ,todas las demás se marcaran como canceladas

Notemos cómo la clase IDENTIFICATION, nos permite asociar un identificador a una opción determinada en una votación en concreto, así conseguimos que una opción en una votación tenga un identificador y en otra pueda tener otro.

Código SQL de creación de tablas

```
CREATE TABLE VOTATION(
    idVotation int PRIMARY key AUTO_INCREMENT,
    end_date date,
    is_active tinyInt,
    is_closed tinyInt
);

CREATE TABLE OPTIONS(
    idOption int PRIMARY Key AUTO_INCREMENT,
    title varchar(60),
    description varchar(140)
);
CREATE TABLE IDENTIFICATION (
    idOption int ,
    idVotation int,
    option_id int,
    FOREIGN KEY (idOption) REFERENCES OPTIONS(idOption) ON DELETE CASCADE,
    FOREIGN KEY (idVotation) REFERENCES VOTATION(idVotation) ON DELETE CASCADE
);
```