

# Trabajo Tutelado BC

## Sistema de Votación

María Andrea Ugarte Valencia

Marcos Villar Avión

# ÍNDICE

<b>1 - Introducción.....</b>	<b>4</b>
1.1 - Motivación.....	4
1.2 - Definición del Escenario.....	4
1.3 - Objetivos.....	5
<b>2 - Estudio del estado del arte.....</b>	<b>6</b>
2.1 - Análisis de soluciones parecidas.....	6
2.2 - Comparativa con la solución propuesta.....	6
2.2.1 - Funcionalidades.....	6
2.2.2 - Ciberseguridad.....	7
<b>3 - Metodología.....</b>	<b>8</b>
<b>4 - Análisis.....</b>	<b>9</b>
4.1 - Actores.....	9
4.2 - Requisitos.....	9
4.3 - Justificación del uso de Ethereum.....	13
<b>5 - Diseño.....</b>	<b>14</b>
5.1 - Casos de uso.....	14
5.2 - Diagramas de secuencia.....	17
5.2.1 - Mostrar candidatos disponibles.....	17
5.2.2 - Votar a un candidato.....	17
5.2.3 - Modo administrador - Mostrar resultados.....	18
5.2.4 - Modo administrador - Subir Censo.....	18
5.3 - Arquitectura de comunicaciones.....	19
5.4 - Modelo de datos.....	20
5.5 - Tecnologías utilizadas.....	21
5.5.1 - IPFS.....	21
5.5.2 - Smart Contract.....	21
5.5.3 - React.....	21
5.5.4 - Docker.....	21
5.5.5 - OrbitDB.....	22
5.5.6 - Metamask.....	22
<b>6 - Implementación.....</b>	<b>23</b>
6.1 - Smart Contract - Solidity.....	23
6.1.1 - Variables globales y Structs.....	23
6.1.2 - Funciones y modifiers.....	25
6.2 - Interfaz de usuario.....	26
<b>7 - Demostración del funcionamiento del sistema.....</b>	<b>30</b>
7.1 - Despliegue.....	30
<b>8 - Análisis de los riesgos de seguridad.....</b>	<b>33</b>
<b>9 - Evaluación y pruebas.....</b>	<b>34</b>
9.1 - Smart Contract.....	34

9.1 - Interfaz de usuario.....	43
<b>10 - Lean canvas.....</b>	<b>45</b>
<b>11 - Aspectos Legales.....</b>	<b>46</b>
<b>12 - Manual de usuario.....</b>	<b>47</b>
<b>13 - Conclusiones.....</b>	<b>51</b>
<b>14 - Líneas futuras.....</b>	<b>52</b>
14.1 - Mejoras a corto plazo.....	52
14.2 - Mejoras a medio plazo.....	52
14.3 - Mejoras a largo plazo.....	52
<b>15 - Lecciones aprendidas.....</b>	<b>53</b>
15.1 - Incidencias.....	53
<b>16 - Planificación.....</b>	<b>55</b>
<b>17 - Referencias.....</b>	<b>56</b>

# 1 - Introducción

El derecho a voto, más conocido como sufragio universal, ha sido uno de los derechos más buscados por parte de los ciudadanos para poder realizar elecciones de forma transparente y justa. Dicho derecho es necesario para que un estado/país tenga un recuento de las preferencias de todos y cada uno de sus ciudadanos, con independencia de su sexo, raza, religión y creencias.

En España, concretamente, el sufragio universal masculino se consiguió a partir del año 1890. Sin embargo, el femenino no se logró hasta 1933, con la nueva constitución de la II República.

Con los avances de la tecnología y los recursos que tenemos hoy en día, surge la siguiente pregunta: ¿por qué a día de hoy seguimos realizando las votaciones en papel? ¿Por qué no trasladamos este sistema al mundo digital para lograr así mayor seguridad, comodidad y eficiencia?

Como posible solución a este problema, se podría proponer un sistema centralizado para cambiar el papel por los sistemas digitales. Sin embargo, este enfoque no sería el más adecuado debido a que no tendríamos la redundancia, fiabilidad, y anonimidad que nos proporciona un sistema descentralizado.

## 1.1 - Motivación

La motivación principal para realizar este trabajo es conseguir realizar un sistema de votación de forma descentralizada, de forma que se consiga con ello que este escrutinio sea más fiable, cómodo, inmediato y seguro que el realizado actualmente en papel y forma presencial.

## 1.2 - Definición del Escenario

En este escenario se utilizarán diferentes tecnologías para lograr la correcta funcionalidad del sistema. En primer lugar, se utilizarán los *smart contracts* creados en *Solidity* para implementar la funcionalidad del sistema. En segundo lugar, se creará una interfaz web para que los diferentes ciudadanos puedan interactuar con el sistema. Y finalmente, el sistema contendrá una base de datos descentralizada para poder ejecutar las peticiones de información correspondientes.

### 1.3 - Objetivos

El principal objetivo del presente trabajo será desarrollar una herramienta capaz de realizar el escrutinio de los votos de manera anónima y automática. Dicha herramienta constará de una interfaz de usuario donde los diferentes ciudadanos podrán realizar la votación a su candidato favorito además de mostrar los resultados de la votación realizada.

Como objetivos secundarios tendríamos los siguientes:

- Estudiar y comprender el lenguaje Solidity para realizar *smart contracts* capaces de ejecutar las funciones responsables de la votación
- Estudiar e investigar el uso de una base de datos descentralizada como es OrbitDB así como sus limitaciones
- Investigar las soluciones propuestas disponibles para realizar una votación con un sistema descentralizado
- Diseñar e implementar la interfaz de usuario mediante la cual los ciudadanos serán capaces de votar a su candidato favorito
- Realizar una implementación segura y completamente anónima para que ningún ciudadano sufra una vulnerabilidad de su integridad
- Desarrollar un sistema de pruebas donde se ejecute todas las herramientas responsables del trabajo.

Entre los objetivos complementarios a este proyecto se encuentran:

- Desarrollo del sistema de forma iterativa
- Implementar la solución del sistema de forma modular para que en el futuro se pueda añadir ampliaciones e incorporaciones de nuevas características
- Preferencia por uso de herramientas y librerías de código abierto
- Separación entre la vista y el modelo

## 2 - Estudio del estado del arte

### 2.1 - Análisis de soluciones parecidas

Actualmente, diversas empresas se dedican a la integración de la tecnología blockchain en sistemas de votación, entre las cuales se destaca ScytI. Esta compañía ha participado en proyectos piloto y colaboraciones gubernamentales con el objetivo de explorar el potencial de la tecnología blockchain en contextos electorales. Una de sus colaboraciones más reconocidas fue con Estonia, país pionero en la utilización de la tecnología blockchain para llevar a cabo elecciones avanzadas y seguras. También, destacó su colaboración con Suiza en programas piloto destinados a modernizar el proceso de votación en el país.

Además de Suiza, varios países han realizado proyectos piloto para evaluar el potencial de la tecnología blockchain. Algunos ejemplos son Rusia o Japón. Sin embargo, estas pruebas no han sido implementadas aún a gran escala.

Voatz es una empresa destacada en el ámbito de los sistemas de votación basados en blockchain. Su aplicación móvil permite a los usuarios emitir sus votos de manera remota combinando tecnología blockchain con verificación biométrica. Esta empresa ha mencionado el uso de la red Ethereum para ejecutar sus sistemas de votación electrónica. Voatz ha sido parte de elecciones piloto y pruebas en varios estados de EE. UU. Un ejemplo notable fue su colaboración con el Partido Demócrata en 2020, donde la tecnología de Voatz se empleó para las votaciones internas.

Existen diversos estudios donde se implementa Blockchain para un sistema de votación como el que se plantea en este documento [3][4]<sup>1</sup> y que se han leído superficialmente antes de realizar este proyecto.

### 2.2 - Comparativa con la solución propuesta

#### 2.2.1 - Funcionalidades

El sistema de votación estándar basado en tecnología blockchain se centra en los votantes, quienes, después de una identificación segura, podrán enviar su voto al candidato de su elección.

Nuestro sistema contará con la capacidad de agregar y eliminar candidatos y administradores. Además, permitirá un voto por persona y mostrará a los ganadores seleccionados. Incluso tendrá una pequeña interfaz web para mostrar cómo podemos interactuar con algunas funciones del *smart contract*. Para que un usuario pueda utilizar nuestro sistema, este ha de contar con una cuenta Metamask con Ethereum disponible.

---

<sup>1</sup> Mirar el apartado: [17 - Referencias](#)

De una forma humorística y para explotar las características de Solidity, nuestro sistema también da la posibilidad de amañar las votaciones si se aporta la cantidad económica suficiente. Sin embargo, somos conscientes de que en un sistema real esta práctica no se debería llevar a cabo.

### 2.2.2 - Ciberseguridad

Al igual que en nuestro caso, las soluciones anteriores se han llevado a cabo con la intención de que las votaciones sean íntegras y no sean manipuladas (a excepción de la parte de amaño de votos mediante pago, que normalmente no se incluiría). Además, se garantizará la confidencialidad de los votos.

Estos aspectos han sido claves durante el desarrollo del proyecto y han estado en consideración durante todo este proceso. Sin embargo, siempre pueden surgir problemas de seguridad debido a causas como el factor humano y una posible mala gestión de las claves privadas.

Nuestro sistema plantea rasgos básicos de la implementación de la tecnología blockchain, por lo que cuenta con flaquezas en seguridad, sobre todo en el apartado de la interfaz, donde un atacante podría aprovechar varias vulnerabilidades de aplicación como puede ser la subida de archivos no controlada.

### 3 -Metodología

Para el desarrollo de este proceso se ha seleccionado una metodología ágil por su facilidad de uso y debido a la incertidumbre inicial del proyecto. Además, se ha tenido enfoque de desarrollo evolutivo para generar múltiples prototipos hasta lograr el sistema final.

Los beneficios de usar esta metodología son varios. En primer lugar, dado que el proyecto se desarrolla en *sprint*, nos permite realizar una mejor organización de las tareas y de las funcionalidades a implementar en cada etapa. En segundo lugar, gracias a desarrollar basándonos en el desarrollo evolutivo, si surge alguna complicación en el proyecto podremos hacer las modificaciones necesarias para remediarlo. Y en tercer y último lugar, el resultado tras varias iteraciones será un producto muy robusto y fiable puesto que en cada etapa se han llevado a cabo varias pruebas unitarias, de integración y de sistema.

La división de los sprints se ha dividido como muestra la siguiente tabla:

<b>Sprint</b>	<b>Tareas a realizar</b>
Sprint 1	1-Investigación inicial de la tecnología a emplear. 2-Aprendizaje del lenguaje y entorno
Sprint 2	1- Implementación del contrato inteligente con las funcionalidades básicas e intermedias 2 - Pruebas unitarias del contrato
Sprint 3	1-Mejora del contrato inteligente 2-Implementación de IPFS 3-Pruebas de integración contrato - IPFS 4- Pruebas de unitarias y de integración
Sprint 4	1-Mejora del contrato inteligente 2-Implementación de la interfaz de usuario 3-Integración con el contrato inteligente 4-Pruebas de integración y de sistema 5- Redacción de documentación



## 4 - Análisis

Antes de implementar el proyecto, hemos tenido que realizar un análisis completo de los actores, requisitos, tecnologías que van a ser usadas... Es por ello que vemos necesario explicar cada uno de estos aspectos y justificar su uso ante un proyecto de tales características.

### 4.1 - Actores

Nuestro sistema tiene varios actores con funcionalidades y privilegios diferentes para separar así por roles las acciones que pueden llevar a cabo y evitar que usuarios no privilegiados realicen acciones con privilegios.

Los actores de nuestro sistema son los siguientes:

- **Usuario:** Representa al ciudadano estándar que utilizará la aplicación para votar o para consultar los resultados.
- **Owner:** Es el actor que lanza el contrato. Tiene privilegios de administración del sistema completo.
- **Administrador:** Son personas cuyo cargo es delegado por el Owner para que lleven tareas de gestión.
- **Candidato:** aunque goza de un papel más pasivo, también es otro usuario diferente que contiene nuestra aplicación. Este usuario puede ser votado y puede realizar las acciones que un usuario normal. Incluso en un futuro se podría añadir más funcionalidades que aporten valor y que se especifican en el apartado correspondiente. Este usuario no se ha tenido en cuenta ya que no tiene un comportamiento activo en el sistema pero sí cabe destacar su parte en el proceso de votaciones.

En el apartado [5 - Diseño](#) se especifican los casos de uso que cada usuario puede realizar

### 4.2 - Requisitos

Hemos dedicado gran cantidad de horas a la identificación de los requisitos que este sistema contiene. Es por ello que a continuación se especifican de forma detallada los requisitos recopilados. Dichos requisitos fueron analizados como historias de usuario debido a la comodidad que aporta frente al análisis de casos de uso clásico

H01	Votar a un candidato			
Descripción				
El usuario sin privilegios puede realizar su voto a su candidato favorito				
Criterios de aceptación				
<ul style="list-style-type: none"><li>El voto se emite correctamente</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Alta	2 horas	2	Finalizada	2 horas

H02	Mostrar los candidatos disponibles			
Descripción				
El usuario sin privilegios puede ver los candidatos actuales de las elecciones				
Criterios de aceptación				
<ul style="list-style-type: none"><li>El usuario visualiza el listado de nombres</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Alta	2 horas	2	Finalizada	2 horas

H03	Mostrar los resultados de las elecciones			
Descripción				
El usuario sin privilegios puede ver los resultados de las elecciones.				
Criterios de aceptación				
<ul style="list-style-type: none"><li>El usuario visualiza los resultados</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Alta	3 horas	3	Finalizada	3 horas

H04	Mostrar el ganador de las elecciones			
Descripción				
El usuario sin privilegios puede visualizar el ganador de las elecciones				
Criterios de aceptación				
<ul style="list-style-type: none"><li>El usuario visualiza al ganador</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Media	1 hora	2	Finalizada	1 hora

H05	Manipular los resultados de las elecciones.			
Descripción				
El usuario sin privilegios puede manipular los resultados de las elecciones				
Criterios de aceptación				
<ul style="list-style-type: none"><li>El usuario manipula el resultado</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Baja	3 horas	4	Finalizada	5 horas

A continuación se muestran los casos de uso que puede realizar explícitamente un usuario con rol de **Administrador**

H06	Gestionar candidatos			
Descripción				
Los administradores pueden añadir, eliminar y modificar candidatos de las elecciones				
Criterios de aceptación				
<ul style="list-style-type: none"><li>Los administradores modifica los candidatos de las elecciones</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Alta	3 horas	4	Finalizada	4 horas

H07	Gestionar censos			
Descripción				
Los administradores pueden añadir, eliminar y modificar archivos del censo				
Criterios de aceptación				
<ul style="list-style-type: none"><li>Los administradores modifica los censo</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Media	1 hora	5	Finalizada	2 horas

A continuación se muestran los casos de uso que puede realizar explícitamente un usuario con rol de **Owner**

H08	Gestionar administradores			
Descripción				
El owner del contrato puede añadir, eliminar o modificar los administradores del contrato				
Criterios de aceptación				
<ul style="list-style-type: none"><li>El owner modifica los administradores del contrato</li></ul>				
Prioridad	Duración estimada	Sprint	Estado	Duración real
Alta	2 horas	4	Finalizada	4 horas

### 4.3 - Justificación del uso de Ethereum

En este proyecto se emplea Ethereum por su capacidad para ejecutar contratos inteligentes. Es una red blockchain bien establecida, lo que garantiza un alto nivel de seguridad y resistencia a la manipulación. Ha sido utilizada en numerosos casos previos (ver [2.1 - Análisis de soluciones parecidas](#)), demostrando su funcionalidad con éxito en diversas aplicaciones. Ethereum cuenta con una comunidad activa y muy buena documentación, por lo que hay herramientas, bibliotecas y recursos disponibles para desarrollar aplicaciones.

## 5 - Diseño

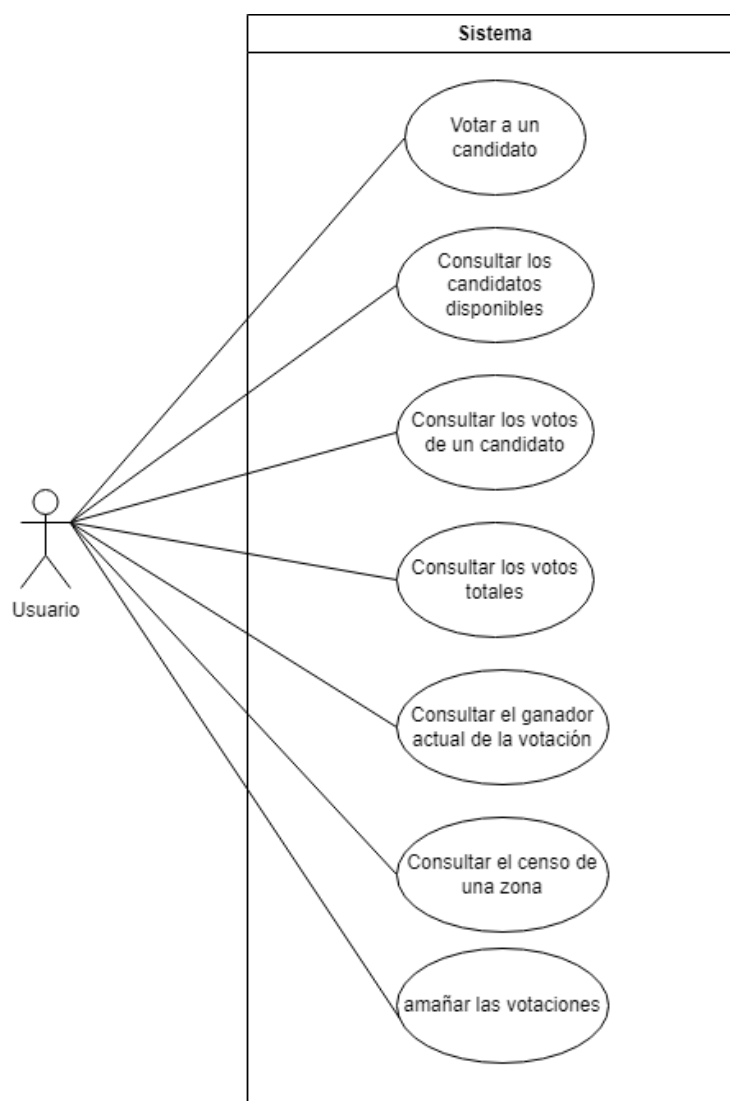
Se ha dedicado una determinada cantidad de tiempo a diseñar el sistema para su correcta implementación y desarrollo. Es por ello que a continuación se muestran los diagramas más típicos que todo proyecto tiene.

### 5.1 - Casos de uso

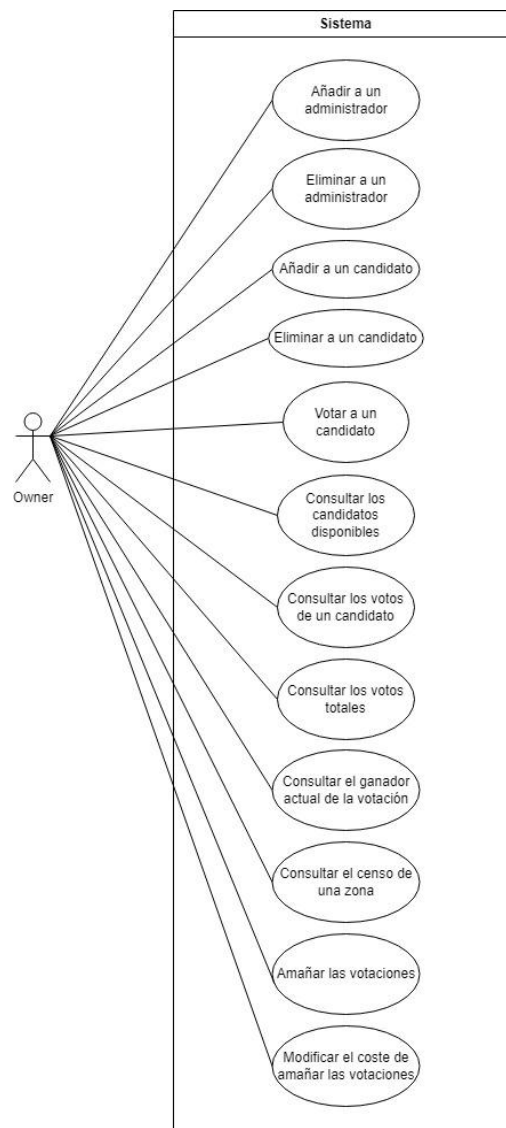
Un diagrama de casos de uso es una herramienta visual utilizada para el análisis y diseño de sistemas de software para representar las interacciones entre usuarios y sistemas. Este diagrama identifica y describe las diversas acciones que los actores externos pueden realizar en el sistema, mostrando cómo se relacionan entre sí.

Hemos dividido los diagramas de casos de uso dependiendo del usuario para una mayor comprensión de este diagrama.

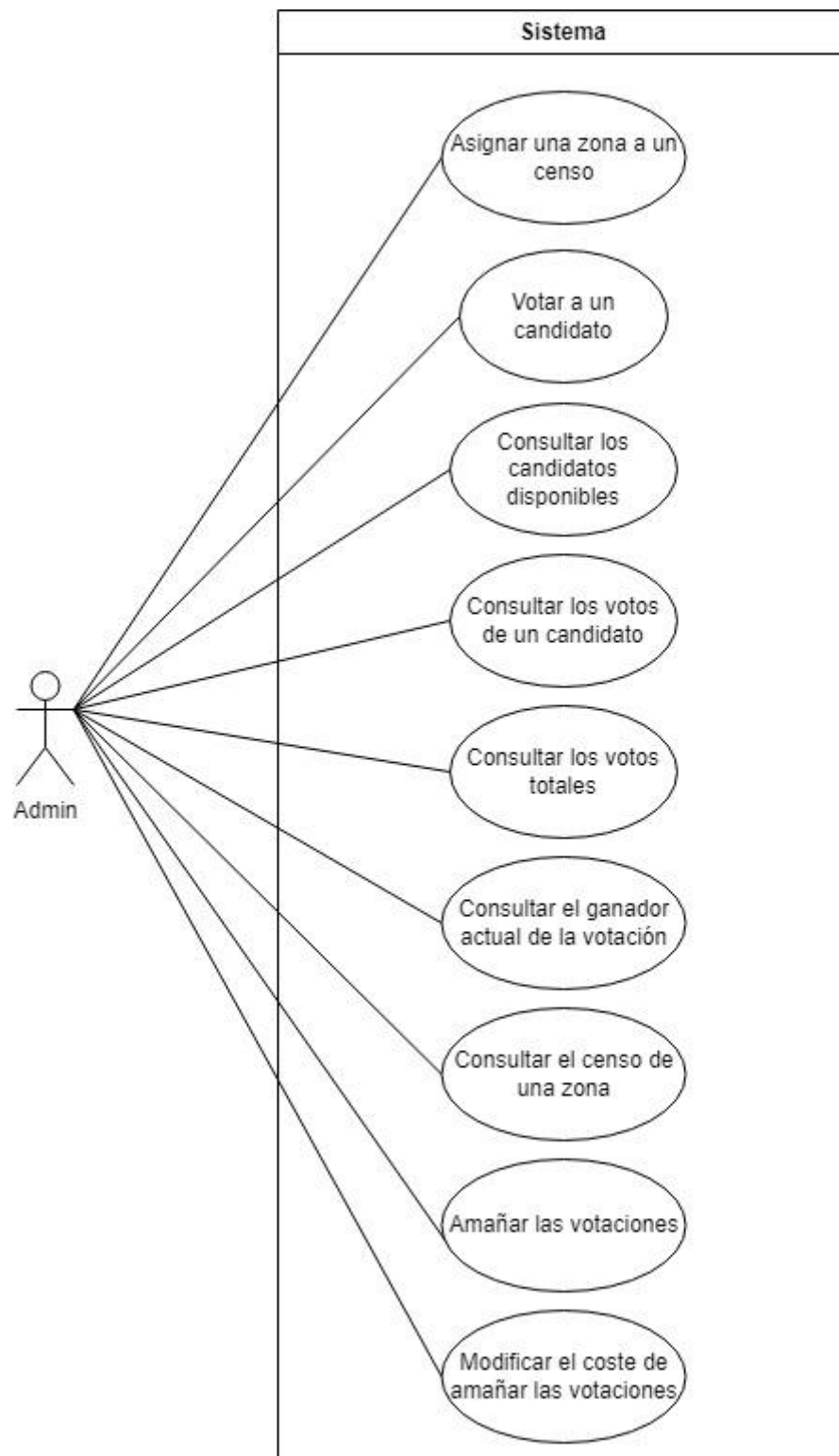
A continuación se muestra el diagrama de casos de uso para el cliente



A continuación se muestra el diagrama de casos de uso para el usuario **owner**



El siguiente diagrama representa los casos de uso de un usuario administrador del sistema



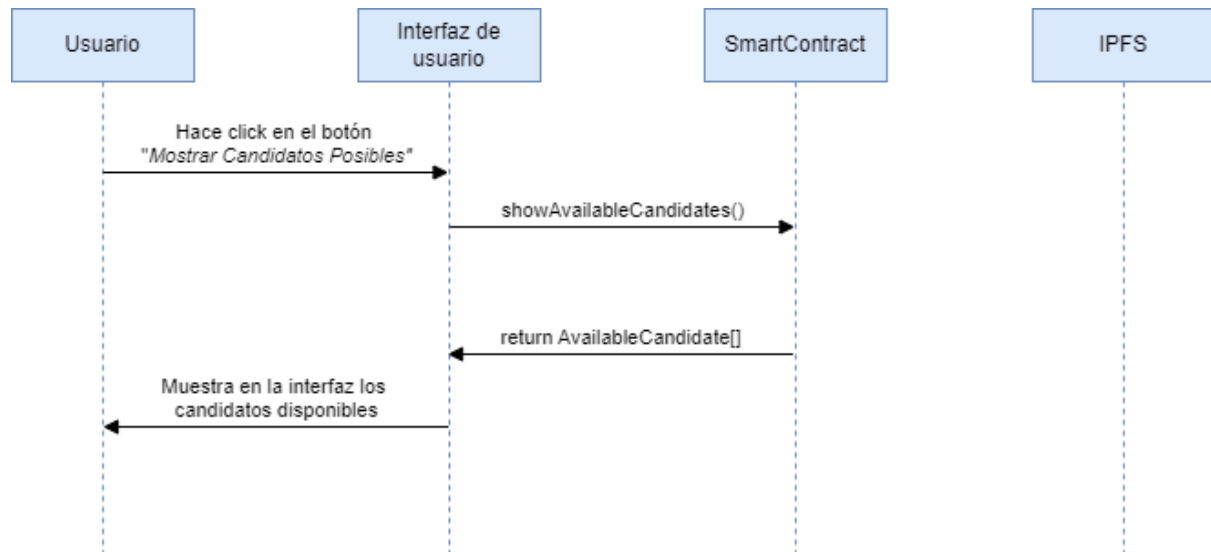


## 5.2 - Diagramas de secuencia

Para una mejor comprensión del código se ha diseñado un diagrama de secuencia en donde se muestra el flujo normal del sistema diseñado.

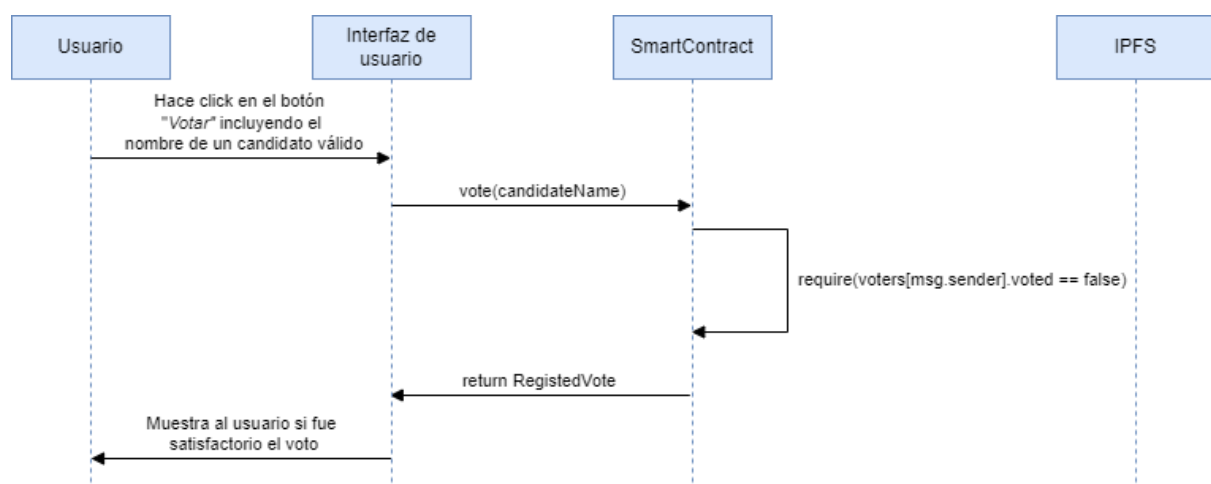
### 5.2.1 - Mostrar candidatos disponibles

A continuación se muestra el diagrama de secuencia para mostrar los candidatos disponibles:



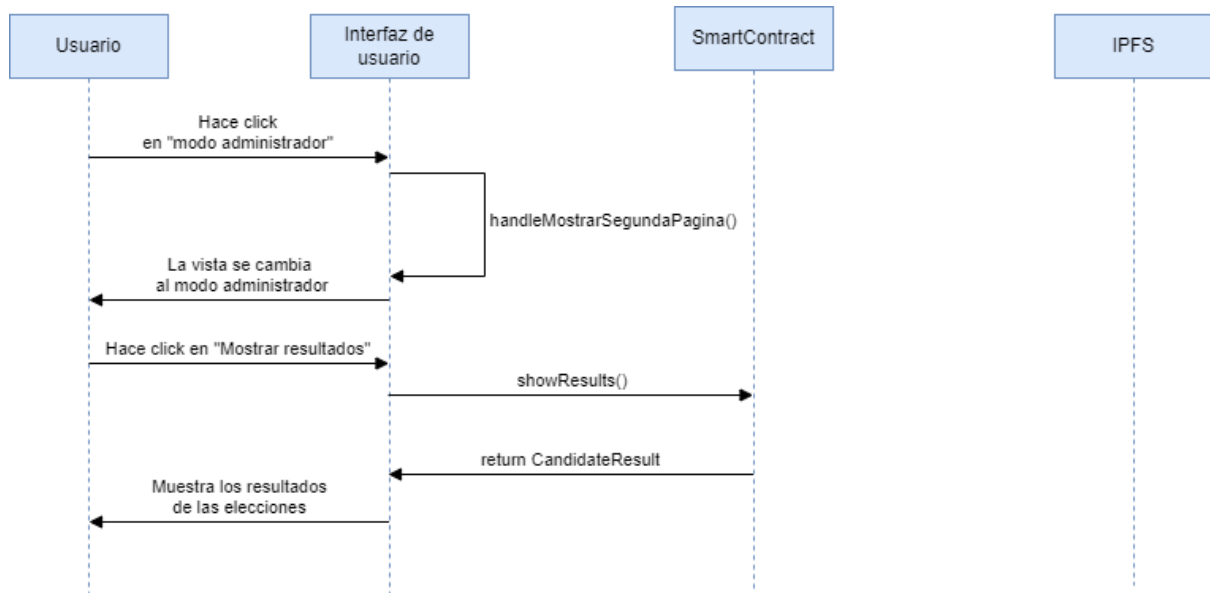
### 5.2.2 - Votar a un candidato

A continuación se muestra el diagrama de secuencia para votar a un candidato de los candidatos disponibles:



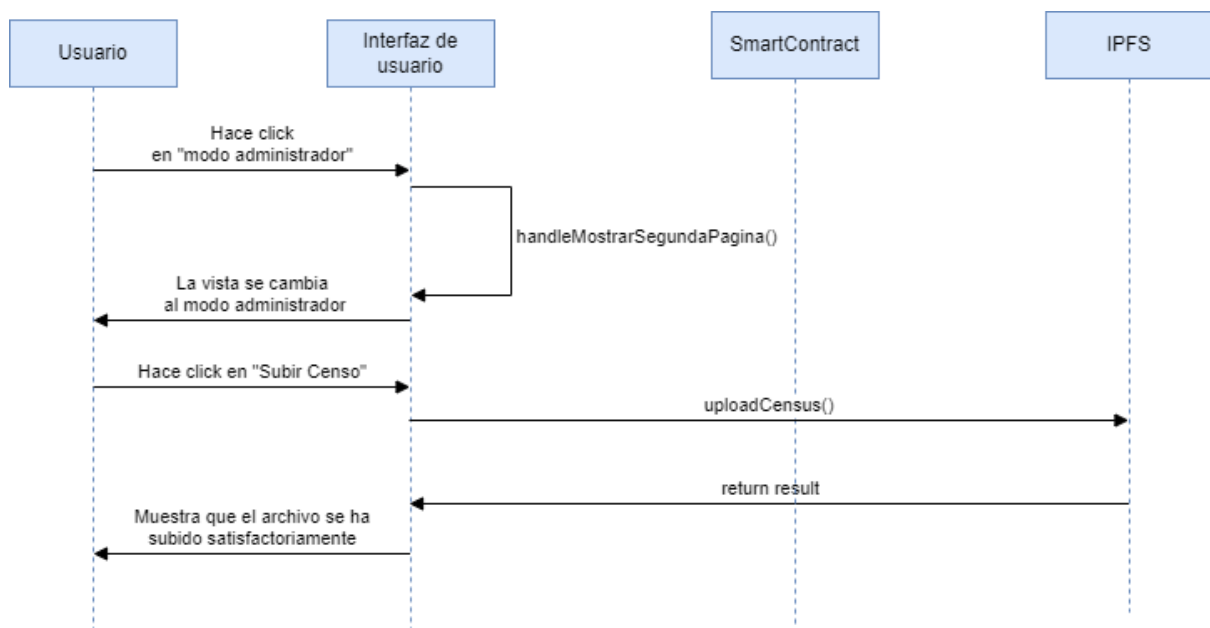
### 5.2.3 - Modo administrador - Mostrar resultados

A continuación se muestra el diagrama de secuencia para cambiar la vista de administrador y mostrar los resultados de las elecciones



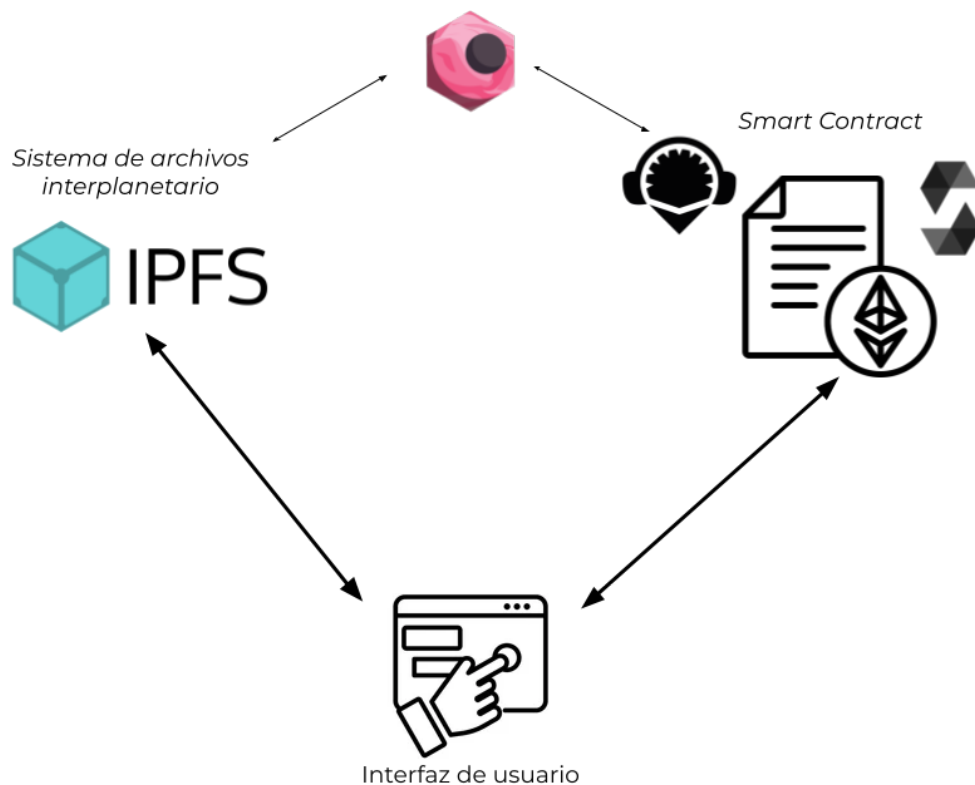
### 5.2.4 - Modo administrador - Subir Censo

A continuación se muestra el diagrama de secuencia para cambiar la vista de administrador y subir un censo de las elecciones



### 5.3 - Arquitectura de comunicaciones

A continuación se muestra el diagrama de arquitectura de comunicaciones donde se puede visualizar las tecnologías empleadas y cómo a través de la interfaz de usuario se puede comunicar con el Sistema de archivos interplanetario, IPFS, y con la lógica que es el *smart contract*



De forma intuitiva se muestra la arquitectura de las comunicaciones de nuestra aplicación y proyecto.

## 5.4 - Modelo de datos

En este proyecto se ha intentado implementar una base de datos descentralizada como OrbitDB pero por cuestiones de tiempo y de dificultad, se ha realizado otro enfoque del problema debido a los inconvenientes que han ido surgiendo a lo largo del desarrollo del proyecto. Es por ello que finalmente se ha decidido almacenar la propia información en el propio contrato inteligente.

En dicho contrato tendremos las siguientes estructuras:

Candidate		
Parámetro	Tipo	Explicación
id	uint	Identificador de candidato
name	string	Nombre del candidato
votos	uint	Número de votos que recibe

Voter		
Parámetro	Tipo	Explicación
voted	bool	Booleano para guardar si el votante ya ha votado
candidateId	uint	Identificador del candidato que ha votado el votante

## 5.5 - Tecnologías utilizadas

En la implementación de este proyecto se han usado varias tecnologías para lograr satisfacer todos los requisitos planteados. Dichas tecnologías se especifican a continuación:

### 5.5.1 - IPFS

*IPFS*, o Sistema de Archivos InterPlanetario (InterPlanetary File System), es un protocolo de red diseñado para crear un sistema de almacenamiento y distribución descentralizado de información. Fue desarrollado con el objetivo de superar las limitaciones de los sistemas de archivos tradicionales y centralizados en la web. En nuestro caso esta tecnología es usada para guardar archivos que representan el censo de la votación

### 5.5.2 - Smart Contract

Un smart contract en Ethereum es un programa informático autoejecutable diseñado para ejecutar automáticamente, gestionar y hacer cumplir acuerdos digitales en la blockchain de Ethereum. Estos contratos inteligentes son escritos en un lenguaje de programación específico para Ethereum, como Solidity, y se ejecutan en la máquina virtual Ethereum (EVM). En nuestro caso se ha utilizado esta tecnología para implementar toda la lógica de negocio de todo el sistema de votaciones.

### 5.5.3 - React

React es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y eficientes. Es especialmente popular en el desarrollo de aplicaciones web de una sola página (SPA), donde se busca una experiencia de usuario fluida y receptiva. En el caso de este proyecto se ha empleado esta tecnología para interactuar a través de la interfaz de usuario con el *smart contract* y proporcionar al usuario una manera cómoda y sencilla de emplear la lógica de negocio implementada en solidity.

### 5.5.4 - Docker

Docker es una plataforma de código abierto diseñada para facilitar la creación, implementación y ejecución de aplicaciones en entornos livianos y portátiles llamados "contenedores". Estos contenedores encapsulan una aplicación y todas sus dependencias, proporcionando un entorno consistente y aislado para su ejecución. Docker simplifica el desarrollo y la implementación al eliminar las diferencias entre los entornos de desarrollo y producción.

En nuestro caso se ha usado docker para lanzar el nodo *IPFS* con la DAPP.

### 5.5.5 - OrbitDB

OrbitDB es una base de datos descentralizada construida sobre la tecnología de la cadena de bloques, específicamente diseñada para entornos peer-to-peer (de igual a igual o P2P). Se desarrolló como parte del ecosistema IPFS (InterPlanetary File System) para proporcionar una solución descentralizada y resistente a la censura para el almacenamiento y la gestión de datos.

En este proyecto se ha intentado implementar OrbitDB pero por diversos errores encontrados no se ha logrado integrar esta herramienta en el proyecto. Este caso se explica mejor en el apartado de [15.1 - Incidencias](#)

### 5.5.6- Metamask

MetaMask es una extensión de navegador y una billetera digital que permite a los usuarios gestionar sus activos criptográficos e interactuar con aplicaciones descentralizadas (dApps) en la cadena de bloques de Ethereum. Funciona como un puente entre el navegador web y la blockchain, permitiendo a los usuarios realizar transacciones, almacenar y gestionar sus criptomonedas, así como acceder a diversas aplicaciones descentralizadas sin necesidad de una infraestructura adicional.

En este proyecto se ha empleado esta herramienta como puente entre la interfaz de usuario y el smart contract.

## 6 - Implementación

Este sistema se ha desarrollado utilizando las tecnologías mencionadas en el apartado anterior [5.5 - Tecnologías utilizadas](#). Es por ello que hemos dividido este apartado en tres partes:

- *Smart Contract*
- Interfaz de usuario
- IPFS

### 6.1 - Smart Contract - Solidity

#### 6.1.1 - Variables globales y Structs

Para que nuestro *smart contract* funcione correctamente tendremos que definir una serie de variables globales. En concreto, las variables globales se especifican y describen en la tabla siguiente:

<u>Variable global</u>	<u>Descripción</u>
tamperValue	Valor de ether mínimo para modificar los resultados de las elecciones
candidates	Array para guardar todos los objetos candidatos
voters	Diccionario/Mapping para guardar una relación entre la dirección del votante y el Votante
isAdmin	Diccionario/Mapping para guardar la información de si un usuario es administrador
censusForZone	Diccionario/Mapping para guardar una relación entre un censo y su zona correspondiente

En cuanto a los structs utilizados, solo hemos definido dos. Uno para guardar los valores importantes del Votante y otro del Candidato:

<u>Struct</u>	<u>Atributos</u>
Candidate	- <u>id</u> : identificador del candidato - <u>name</u> : nombre - <u>votes</u> : votos recibidos
Voter	- <u>voted</u> : booleano para comprobar si ya ha votado - <u>candidateId</u> : el identificador del candidato



### 6.1.2 - Funciones y modifiers

En la implementación del sistema de votación utilizando un *smart contract* se han creado diversas funciones para satisfacer los objetivos del sistema. Además, se han creado *modifiers* nuevos y se han utilizado los heredados de **Ownable**. En la siguiente tabla se especifican los modificadores utilizados:

<u>Modificador</u>	<u>Descripción</u>
existCandidateName	Para verificar que el usuario realmente existe
notExistCandidateName	Para verificar que el usuario realmente no existe
onlyOwner	Heredada de Ownable
onlyAdmin	Para verificar que un usuario es administrador

En la siguiente tabla se especifican las funciones implementadas y una breve descripción de ellas:

<u>Nombre</u>	<u>Funcionalidad</u>	<u>Control de Acceso</u>
addCandidate()	Añade un candidato	Owner
removeCandidate()	Elimina un determinado candidato	Owner
addAdmin()	Añade un administrador.	Owner
removeAdmin()	Elimina un administrador.	Owner
addCensusForZone()	Permite a un administrador asignar una zona a un censo.	Admin
getCensusForZone()	Muestra el hash del censo asignado a una zona específica.	
vote()	Permite a un usuario votar por un candidato mediante el nombre. El usuario solo puede votar una vez y se comprueba que el nombre del candidato sea válido.	
showResults()	Muestra los resultados de todos los candidatos	
showAvailableCandidates()	Muestra todos los candidatos disponibles para ser votados	

showWinner()	Muestra el ganador de las elecciones.	
_hashCompareWithLengthCheck()	Compara dos strings pasadas como parametros	
isNameInCandidates()	Verifica que el nombre del candidato exista	
getByIdByName()	Devuelve el identificador del candidato a partir de su nombre	
tamperElections()	Función que nos permite modificar el ganador de las elecciones	
setTamperValue()	Método para cambiar el valor de ether que hay que pagar para modificar el ganador de las elecciones.	Owner

## 6.2 - Interfaz de usuario

La interfaz de usuario se ha implementado utilizando [React](#) que es una tecnología basada en javascript gracias a la cual se pueden crear aplicaciones SPA. Es por ello que se ha decidido utilizar esta tecnología para la creación de la interfaz web.

El código se ha estructurado por partes, las mostraremos a continuación.

Aquí, gestionamos los mensajes temporales que se muestran al usuario:

```

22 //Message
23 const [message, setMessage] = useState(null);
24 const [messageTimeout, setMessageTimeout] = useState(null);
25
26 const showMessage = (msg, duration = 3000) => {
27   setMessage(msg);
28   clearTimeout(messageTimeout);
29   const timeout = setTimeout(() => {
30     setMessage(null);
31   }, duration);
32   setMessageTimeout(timeout);
33 };

```

Después, esta parte del código la usamos para subir archivos IPFS:

```

37 //IPFS
38 const [ipfsHash, setIpfsHash] = useState("");
39 const [file, setFile] = useState(null);
40 const fileInputRef = useRef(null);
41
42 useEffect(() => {
43   async function readFile() {
44     try {
45       const result = await ipfsContract.getCensusForZone(defaultProvider.getSigner().getAddress());
46       if (result !== ZERO_ADDRESS) setIpfsHash(result);
47     } catch (error) {
48       console.error("Error fetching census for zone:", error.message);
49     }
50   }
51   readFile();
52 }, [defaultProvider]);
53
54 const setFileIPFS = async (hash) => {
55   const ipfsWithSigner = ipfsContract.connect(defaultProvider.getSigner());
56   console.log("TX contract");
57   const tx = await ipfsWithSigner.setFileIPFS(hash);
58   console.log({ tx });
59   setIpfsHash(hash);
60 };
61
62 const handleFileChange = (e) => {
63   const data = e.target.files[0];
64   const reader = new window.FileReader();
65   reader.readAsArrayBuffer(data);
66   reader.onloadend = () => {
67     setFile(Buffer(reader.result));
68   };
69
70   const fileName = fileInputRef.current.files[0]?.name;
71   const label = document.getElementById('file-label');
72   if (label) label.textContent = fileName || 'Browse';
73 };
74
75 const handleSubmit = async (e) => {
76   e.preventDefault();
77   try {
78     const client = await create('/ip4/0.0.0.0/tcp/5001');
79     const result = await client.add(file);
80
81     await client.files.cp(`/ipfs/${result.cid}`, `/${result.cid}`);
82     console.log(result.cid);
83     await setFileIPFS(result.cid.toString());
84   } catch (error) {
85     console.log(error.message);
86   }
87 };
88
89 const retrieveFile = (e) => {
90   const data = e.target.files[0];
91   const reader = new window.FileReader();
92   reader.readAsArrayBuffer(data);
93   reader.onloadend = () => {
94     console.log("Buffer data: ", Buffer(reader.result));
95     setFile(Buffer(reader.result));
96   }
97   e.preventDefault();
98 };

```

Posteriormente, mostramos los candidatos gracias a este fragmento:

```

100 //Candidates
101 const [candidateName, setCandidateName] = useState("");
102 const [candidates, setCandidates] = useState([]);
103 const [executedFetch, setExecutedFetch] = useState(false);
104 const [showCandidates, setShowCandidates] = useState(false);
105
106 const fetchCandidates = useCallback(async () => {
107   try {
108     const provider = new ethers.providers.Web3Provider(window.ethereum);
109     const signer = provider.getSigner();
110     const contract = new ethers.Contract(addresses.ipfs, abis.ipfs, signer);
111
112     const eventListener = contract.on("AvailableCandidate", (candidateName, candidateIndex) => {
113       setCandidates(prevCandidates => [
114         ...prevCandidates,
115         { name: candidateName, index: candidateIndex }
116       ]);
117     });
118
119     await contract.showAvailableCandidates();
120
121     return () => {
122       eventListener.removeAllListeners();
123     };
124   } catch (error) {
125     console.error("Error fetching candidates:", error);
126   }
127 }, []);
128
129 useEffect(() => {
130   if (showCandidates && !executedFetch) {
131     fetchCandidates();
132     setExecutedFetch(true);
133   }
134 }, [showCandidates, executedFetch, fetchCandidates]);
135
136 const handleShowCandidates = () => {
137   setShowCandidates(prevShowCandidates => !prevShowCandidates);
138 };

```

Esto nos sirve para intercalar entre la vista de usuario y la de administrador:

```

141 //Change of page
142 const [mostrarPrimeraPagina, setMostrarPrimeraPagina] = useState(true);
143
144 const handleMostrarSegundaPagina = () => {
145   setMostrarPrimeraPagina(false);
146 };
147
148 const handleVolverPrimeraPagina = () => {
149   setMostrarPrimeraPagina(true);
150 };

```

Esta parte se utiliza para mostrar los resultados actuales:

```

153 //Results
154 const [showResults, setShowResults] = useState(false);
155 const [resultados, setResultados] = useState([]);
156
157 const handleShowResults = async () => {
158   try {
159     const provider = new ethers.providers.Web3Provider(window.ethereum);
160     const signer = provider.getSigner();
161     const contract = new ethers.Contract(addresses.ipfs, abis.ipfs, signer);
162
163     const tx = await contract.showResults();
164     tx.wait().then(() => {
165       contract.on("CandidateResult", (candidateName, votes) => {
166         setResultados(prevResults => [...prevResults, { name: candidateName, votes: votes }]);
167       });
168     });
169   } catch (error) {
170     console.error("Error al mostrar resultados:", error);
171   }
172 };
173
174 const handleToggleResults = () => {
175   setShowResults(prev => !prev);
176   if (!showResults) {
177     handleShowResults();
178   }
179 };

```

Aquí, está el fragmento del código para realizar votaciones:

```
182 //Vote
183
184 const handleVote = async () => {
185   try {
186     const provider = new ethers.providers.Web3Provider(window.ethereum);
187     const signer = provider.getSigner();
188     const contract = new ethers.Contract(addresses.ipfs, abis.ipfs, signer);
189
190     await contract.vote(candidateName);
191     showMessage(`Votaste por ${candidateName}`);
192   } catch (error) {
193     showMessage(`Error al votar`);
194     console.error("Error al votar:", error);
195   }
196 };
```

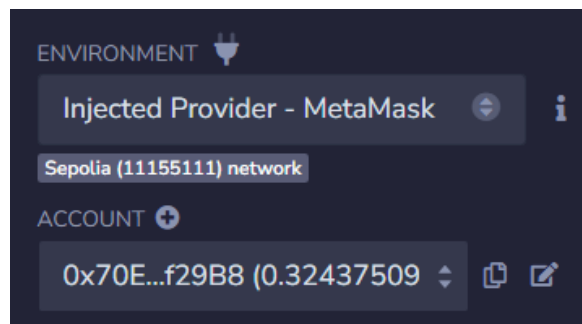
Por último, tenemos el código para el diseño de la interfaz, el cuál no vamos a incluir en este documento por su papel secundario en el proyecto. Sin embargo, se puede visualizar rápidamente el resultado conseguido en el apartado de [12 - Manual de usuario](#)

## 7 - Demostración del funcionamiento del sistema

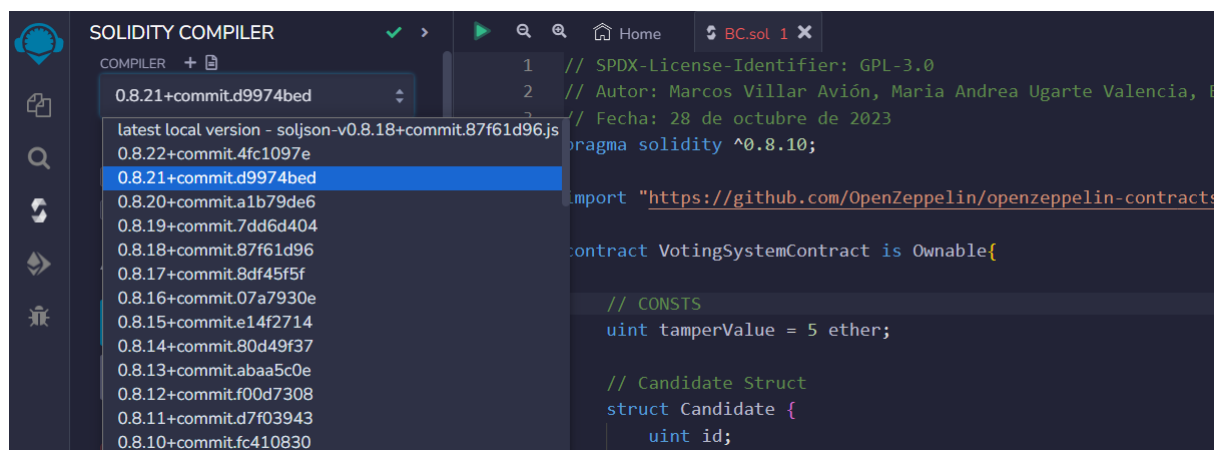
### 7.1 - Despliegue

#### **Smart Contract**

Para ejecutar el *smart contract* desarrollado, necesitaremos desplegarlo con nuestra cuenta de Metamask. Esto nos dará la capacidad de realizar operaciones que conllevan algún tipo de costo.



También, tendremos que seleccionar la versión 0.8.21 que es con la que hemos realizado todas las pruebas.



Ahora, añadimos candidatos y el contrato debería crearse sin ningún inconveniente:

**GAS LIMIT**

3000000

**VALUE**

0 Ether

**CONTRACT**

VotingSystemContract - contracts/BC

evm version: shanghai

Deploy ["Marcos","Andrea","Bahdon"]

☐ Publish to IPFS

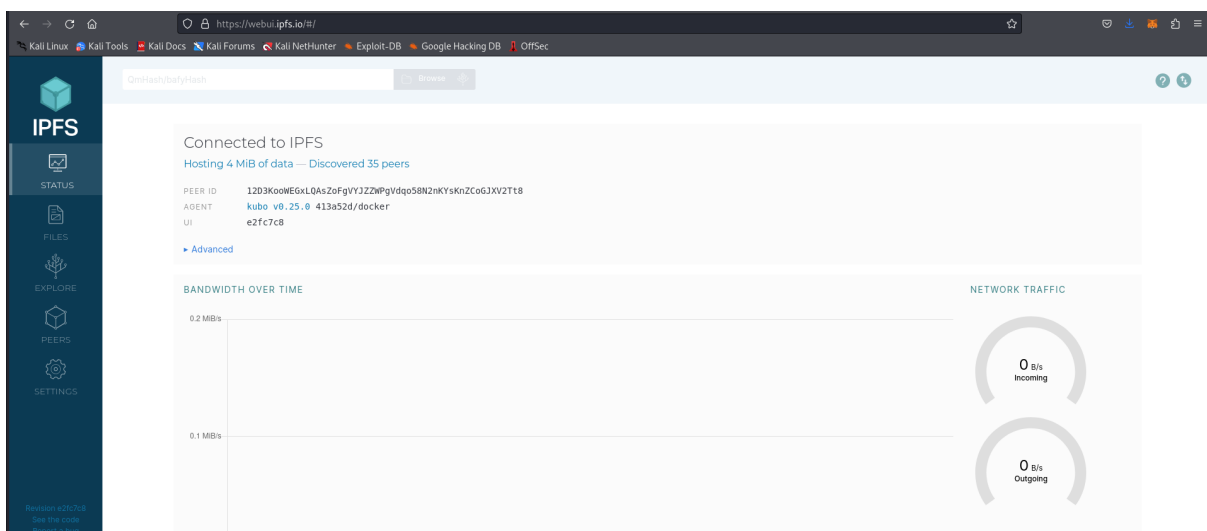
## IPFS

Lanzamos un nodo IPFS:

```
(kali@kali)~$ sudo docker run -d --name ipfs_host -v $PWD:/export -v $PWD:/data/ipfs -p 4001:4001 -p 4001:4001/udp -p 127.0.0.1:8080:8080 -p 127.0.0.1:5001:5001 ipfs/kubo
aefb58aa41e6b7dc3bee3389f934b280527da3c949fa45c5ea1b6f934eb01c

(kali@kali)~$ sudo docker exec ipfs_host ipfs config --json API.HTTPHeaders.AccessControl-Allow-Origin '["http://0.0.0.0:5001", "http://localhost:3000", "http://127.0.0.1:5001", "https://webui.ipfs.io"]'
```

Comprobamos que estamos conectados:



## Interfaz de usuario

Nos movemos a la carpeta `dapp_ipfs` y ejecutamos `npm start`:

```
(kali@kali)-[~]:5001:5001 ipfs/kubo
$ cd dapp_ipfs

(kali@kali)-[~/dapp_ipfs]
$ npm start
> dapp_ipfs@0.1.0 start
> react-scripts start
```

La interfaz ha cargado correctamente:





## 8 - Análisis de los riesgos de seguridad

Hemos estudiado los posibles riesgos de seguridad en un sistema de estas características y hemos destacado los siguientes:

- **Ataques de 51%:** Se produce en el momento en que un solo actor o grupo controla el 51% o más del poder computacional de la red, lo que podría permitir la manipulación de votos o la interrupción del proceso electoral. Una red pequeña como la nuestra podría ser susceptible a este tipo de ataques.
- **Riesgos de identidad:** La autenticación de identidades en un sistema de votación en línea es crucial. Si se produce un robo de identidad o suplantación, se podrían emitir votos fraudulentos.
- **Phishing y malware:** Los votantes podrían ser susceptibles a ataques de phishing o malware que comprometan sus claves privadas o información de acceso, permitiendo a terceros manipular sus votos.
- **Escalabilidad y congestión de red:** Ethereum ha experimentado problemas de escalabilidad y congestión de red durante momentos de alta demanda. Esto podría afectar la velocidad y eficiencia del proceso de votación.
- **Errores pasados por alto debido a la inexperiencia:** Es la primera vez que realizamos un proyecto de este tipo, por lo que podemos pasar por alto detalles que desconocemos que pueden traer problemas.

## 9 - Evaluación y pruebas

### 9.1 - Smart Contract

#### Añadir a un candidato

Insertamos una nueva candidata llamada Paula (solo el owner del contrato podrá añadir nuevos candidatos):



```

    },
    {
      "from": "0x5FD6eB55D12E759a21C09eF703fe0C8a1DC9d88D",
      "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eae6d10da3ca4d2ed6",
      "event": "AvallaleCandidate",
      "args": {
        "0": "Paula",
        "1": "3",
        "name": "Paula",
        "id": "3"
      }
    }
  ]
}

```

Si otro usuario que no fuera el propietario del *smart Contract* intentará ejecutar la función, se le mostraría el siguiente error

```

transact to VotingSystemContract.addCandidate pending ...

transact to VotingSystemContract.addCandidate errored: Error occured: revert.

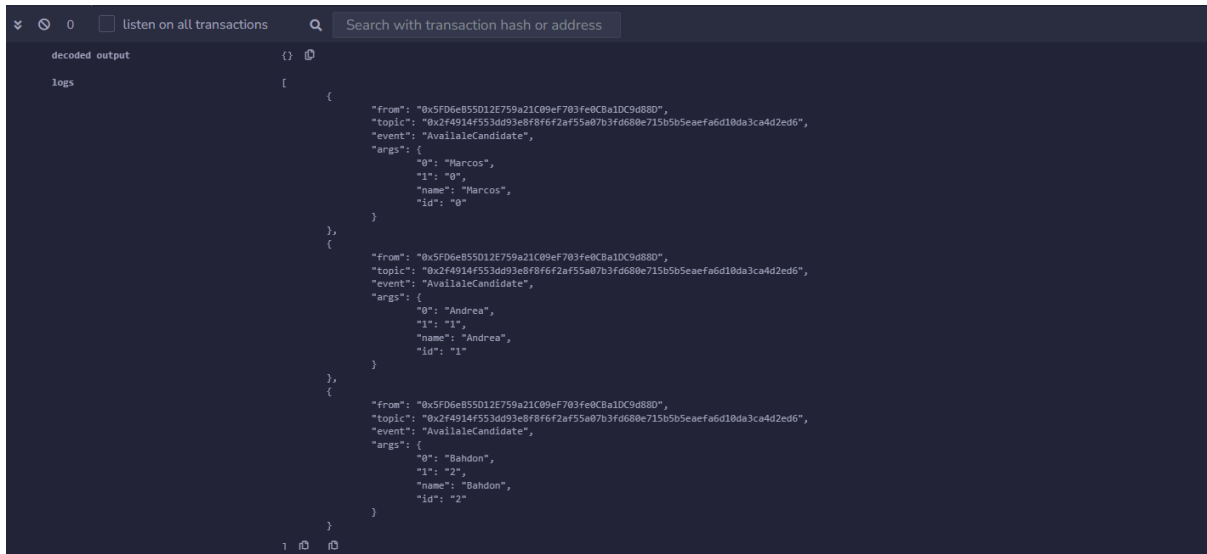
revert
  The transaction has been reverted to the initial state.
Error provided by the contract:
OwnableUnauthorizedAccount
Parameters:
{
  "account": {
    "value": "0xAb8483F64d9C6d1FcF9b849Ae677d03315835cb2"
  }
}
Debug the transaction to get more information.

[vm] from: 0xAb8...35cb2 to: VotingSystemContract.addCandidate(string) 0x5FD...9d88D value: 0 wei data: 0x462...00000 logs: 0 hash: 0xebf...10466

```

## Mostrar candidatos disponibles

Ejecutamos la función de `showAvailableCandidates()` para mostrar los candidatos que tenemos disponibles en nuestro sistema y sus respectivos identificadores:



```

decoded output {}
logs [
  {
    "from": "0x5FD6e855D12E759a21C09eF703fe0C8a1DC9d88D",
    "topic": "0x2f4914f553dd93e8f8f6f2af55e07b3fd680e715b5b5eae6d10da3ca4d2ed6",
    "event": "AvailableCandidate",
    "args": {
      "0": "Marcos",
      "1": "0",
      "name": "Marcos",
      "id": "0"
    }
  },
  {
    "from": "0x5FD6e855D12E759a21C09eF703fe0C8a1DC9d88D",
    "topic": "0x2f4914f553dd93e8f8f6f2af55e07b3fd680e715b5b5eae6d10da3ca4d2ed6",
    "event": "AvailableCandidate",
    "args": {
      "0": "Andrea",
      "1": "1",
      "name": "Andrea",
      "id": "1"
    }
  },
  {
    "from": "0x5FD6e855D12E759a21C09eF703fe0C8a1DC9d88D",
    "topic": "0x2f4914f553dd93e8f8f6f2af55e07b3fd680e715b5b5eae6d10da3ca4d2ed6",
    "event": "AvailableCandidate",
    "args": {
      "0": "Bahdon",
      "1": "2",
      "name": "Bahdon",
      "id": "2"
    }
  }
]

```

## Eliminar a un candidato

Tras verificar que la adición de candidatos funciona correctamente, vamos a verificar que solamente el *owner* del *smart contract* puede ejecutar dicha función y que se elimina correctamente el usuario especificado.

En primer lugar, vamos a verificar que un usuario cualquiera no puede eliminar candidatos ya que le saldría el siguiente error:



```

transact to VotingSystemContract.removeCandidate pending ...

transact to VotingSystemContract.removeCandidate errored: Error occurred: revert.

revert
  The transaction has been reverted to the initial state.
Error provided by the contract:
OwnableUnauthorizedAccount
Parameters:
{
  "account": {
    "value": "0xab8483f64d9c6d1ecf9b849ae677d03315835cb2"
  }
}
Debug the transaction to get more information.

[vm] from: 0xab8...35cb2 to: VotingSystemContract.removeCandidate(string) 0x5FD...9d88D value: 0 wei data: 0x01c...00000 logs: 0 hash: 0x80e...6c86e

```

A continuación, vamos a verificar que si se introduce un nombre erróneo de un candidato que no existe en el *smart contract*, se visualizará un error indicado dicho problema.

```

transact to VotingSystemContract.removeCandidate pending ...

transact to VotingSystemContract.removeCandidate errored: Error occurred: revert.

revert
  The transaction has been reverted to the initial state.
  Reason provided by the contract: "This candidate does not exist!".
  Debug the transaction to get more information.

[vm] from: 0x5B3...eddC4 to: VotingSystemContract.removeCandidate(string) 0x5FD...9d88D value: 0 wei data: 0x01c...00000 logs: 0 hash: 0xea8...cc4f4

```

Como se puede ver, se nos indica que el usuario no existe mediante el mensaje: *"This candidate does not exist"*. Por lo tanto, vamos ahora a comprobar que se puede eliminar un candidato que realmente exista y que la modificación se hace efectiva.

```

transact to VotingSystemContract.removeCandidate pending ...

[vm] from: 0x5B3...eddC4 to: VotingSystemContract.removeCandidate(string) 0x5FD...9d88D value: 0 wei data: 0x01c...00000 logs: 0 hash: 0x7fe...b431d

```

Y si visualizamos ahora los candidatos disponibles:

```

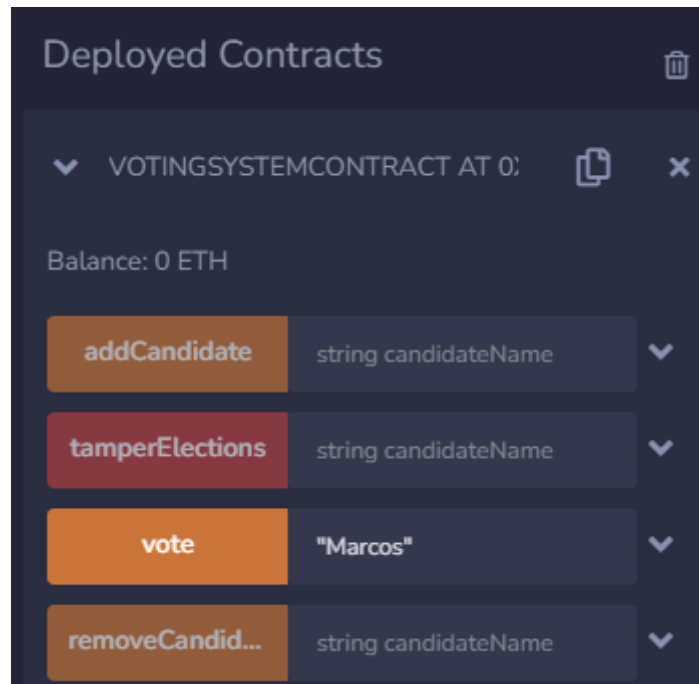
{
  "from": "0x5FD6eB55D12E759a21C09eF703fe0C8a1DC9d88D",
  "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eaeafa6d10da3ca4d2ed6",
  "event": "AvailaleCandidate",
  "args": {
    "0": "Paula",
    "1": "0",
    "name": "Paula",
    "id": "0"
  }
},
{
  "from": "0x5FD6eB55D12E759a21C09eF703fe0C8a1DC9d88D",
  "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eaeafa6d10da3ca4d2ed6",
  "event": "AvailaleCandidate",
  "args": {
    "0": "Andrea",
    "1": "1",
    "name": "Andrea",
    "id": "1"
  }
},
{
  "from": "0x5FD6eB55D12E759a21C09eF703fe0C8a1DC9d88D",
  "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eaeafa6d10da3ca4d2ed6",
  "event": "AvailaleCandidate",
  "args": {
    "0": "Bahdon",
    "1": "2",
    "name": "Bahdon",
    "id": "2"
  }
}

```

Confirmando así que nuestro sistema funciona correctamente.

### Votar a un candidato

Vamos a ver cómo votar ahora a un candidato. En nuestro caso, vamos a votar al “Marcos”



Y si vemos los logs:

```
transact to VotingSystemContract.vote pending ...  
  
[vm] from: 0x5B3...eddC4 to: VotingSystemContract.vote(string) 0xEc2...cF142 value: 0 wei data: 0xfc3...00000 logs: 1 hash: 0x5dc...7ccfd
```

Si intentáramos hacer esto otra vez, lógicamente la función nos mostraría un error ya que el mismo usuario no puede votar dos veces

```
transact to VotingSystemContract.vote pending ...  
  
transact to VotingSystemContract.vote errored: Error occurred: revert.  
  
revert  
  The transaction has been reverted to the initial state.  
Reason provided by the contract: "You have already voted".  
Debug the transaction to get more information.  
  
[vm] from: 0x5B3...eddC4 to: VotingSystemContract.vote(string) 0xEc2...cF142 value: 0 wei data: 0xfc3...00000 logs: 0 hash: 0xf35...5ff6e
```

Y si intentamos votar a un usuario que no existe:

```
transact to VotingSystemContract.vote pending ...

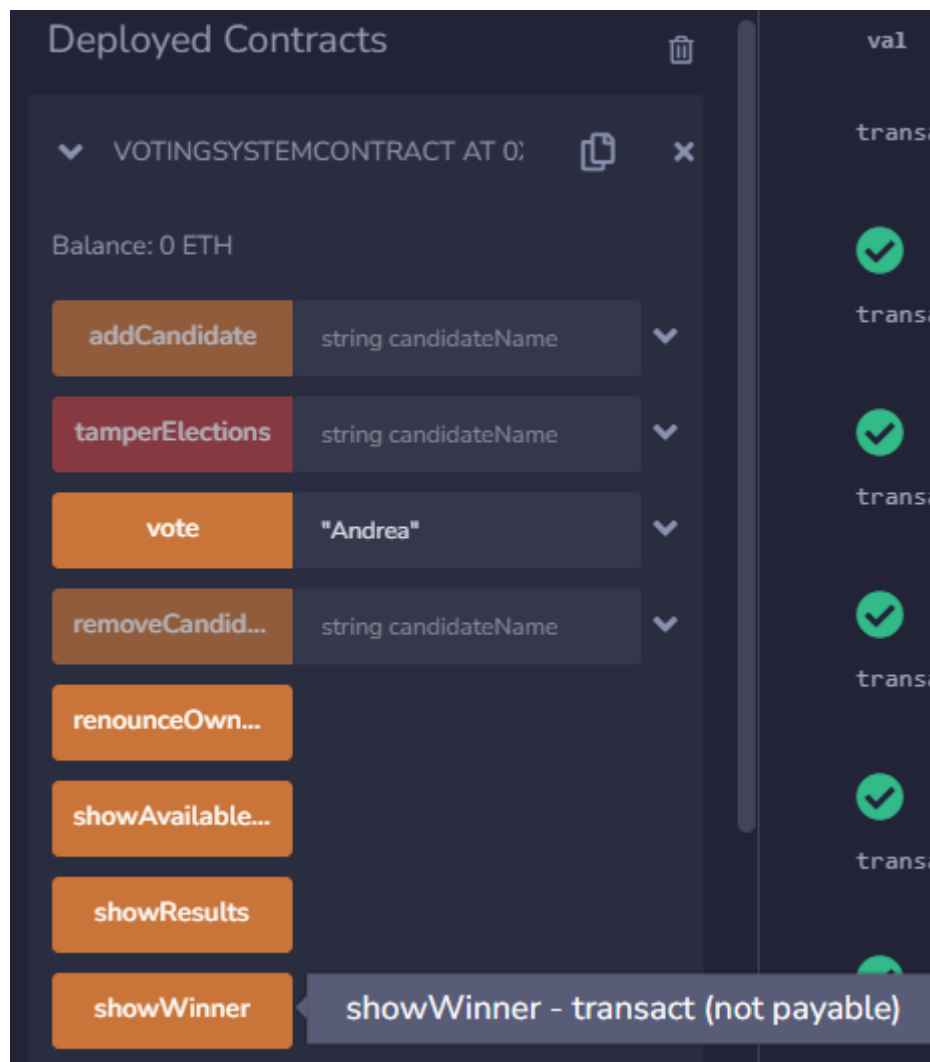
transact to VotingSystemContract.vote errored: Error occurred: revert.

revert
  The transaction has been reverted to the initial state.
  Reason provided by the contract: "This candidate does not exist!".
  Debug the transaction to get more information.

[vm] from: 0xAb8...35cb2 to: VotingSystemContract.vote(string) 0xEC2...cF142 value: 0 wei data: 0xfc3...00000 logs: 0 hash: 0xbd0...9db32
```

### Mostrar al ganador

Vamos a ver como se muestra el resultado de las elecciones una vez todos los usuarios han votado. Para ello se hará uso del método `showWinner()` que crea un evento para mostrar el ganador de las elecciones y cuántos votos ha conseguido



Y este es el resultado tras las elecciones:

```
logs
[
  {
    "from": "0xEc29164D68c4992cEdd1D386118A47143FdcF142",
    "topic": "0xec7b07f92bbe36adaa91eaf451da184b543d99b53d1d2f4826d9de457750d2",
    "event": "Winner",
    "args": {
      "0": "Andrea",
      "1": "4",
      "name": "Andrea",
      "votes": "4"
    }
  }
]
```

### Mostrar todos los resultados

Para mostrar todos los resultados de las elecciones recopilador ejecutaremos la función `showResult()` que mostraría algo como lo siguiente:

```
logs
[
  {
    "from": "0xEc29164D68c4992cEdd1D386118A47143FdcF142",
    "topic": "0xb6daccab904ec230aea8840fa0f8e907d3a955340b279897f211bd6ba2354067",
    "event": "CandidateResult",
    "args": {
      "0": "Marcos",
      "1": "1",
      "name": "Marcos",
      "votes": "1"
    }
  },
  {
    "from": "0xEc29164D68c4992cEdd1D386118A47143FdcF142",
    "topic": "0xb6daccab904ec230aea8840fa0f8e907d3a955340b279897f211bd6ba2354067",
    "event": "CandidateResult",
    "args": {
      "0": "Andrea",
      "1": "5",
      "name": "Andrea",
      "votes": "5"
    }
  },
  {
    "from": "0xEc29164D68c4992cEdd1D386118A47143FdcF142",
    "topic": "0xb6daccab904ec230aea8840fa0f8e907d3a955340b279897f211bd6ba2354067",
    "event": "CandidateResult",
    "args": {
      "0": "Bahdon",
      "1": "3",
      "name": "Bahdon",
      "votes": "3"
    }
  }
]
```

### Amañar las elecciones pagando

Hemos implementado una manera de modificar el resultado de las elecciones en caso de que alguien esté dispuesto a pagar la cantidad solicitada. Para ello, hay un método llamado `tamperElections()` donde se tiene que pagar **5 ether** para poder ejecutarla. Veamos un ejemplo para que el candidato Bahdon gane las elecciones en vez de Andrea



Si no se paga suficiente ether, el programa mostrará un error *"You must pay more!"*

```
transact to VotingSystemContract.tamperElections pending ...
transact to VotingSystemContract.tamperElections errored: Error occurred: revert.

revert
  The transaction has been reverted to the initial state.
Reason provided by the contract: "You must pay more!".
Debug the transaction to get more information.

[vm] from: 0xCA3...a733c to: VotingSystemContract.tamperElections(string) 0xEc2...cF142 value: 0 wei data: 0x106...00000 logs: 0 hash: 0x76c...84c87
```

Incluso si el usuario especificado no existe se mostrará un error:

```
transact to VotingSystemContract.tamperElections pending ...
transact to VotingSystemContract.tamperElections errored: Error occurred: revert.

revert
  The transaction has been reverted to the initial state.
Reason provided by the contract: "This candidate does not exist!".
Debug the transaction to get more information.

[vm] from: 0xCA3...a733c to: VotingSystemContract.tamperElections(string) 0xEc2...cF142 value: 8000000000000000000 wei data: 0x106...00000 logs: 0 hash: 0x569...90e7c
```

Pero en caso de que el usuario pague lo suficiente se puede llegar a modificar los resultados:

```
transact to VotingSystemContract.tamperElections pending ...

[vm] from: 0xCA3...a733c to: VotingSystemContract.tamperElections(string) 0xEc2...cF142 value: 6000000000000000000 wei data: 0x106...00000 logs: 0 hash: 0xeca...7ccec
```

Y ahora si mostramos el resultado de *showWinner()*



```
logs
[
  {
    "from": "0xEc29164D68c4992cEdd1D386118A47143FdcF142",
    "topic": "0xec7b07f92bbe36adaa91eaf451da184b543d99b53d1d2f4826d9de457750d2",
    "event": "Winner",
    "args": {
      "0": "Bahdon",
      "1": "6",
      "name": "Bahdon",
      "votes": "6"
    }
  }
]
```

### Añadir a un administrador

Añadimos a un administrador para que pueda realizar la gestión de censos:

**addAdmin**

La operación se realiza correctamente:

✓ [block:4783854 txIndex:4] from: 0x70e...f29b8 to: VotingSystemContract.addAdmin(address) 0x3c9...0adb2 value: 0 wei data: 0x704...f29b8 logs: 0 hash: 0x521...13b45

### Eliminar administradores

Eliminamos a un administrador:




**removeAdmin**



El administrador se elimina con éxito:

✓ [block:4783872 txIndex:7] from: 0x70e...f29b8 to: VotingSystemContract.removeAdmin(address) 0x3c9...0adb2 value: 0 wei data: 0x178...f29b8 logs: 0 hash: 0x6d4...58de5

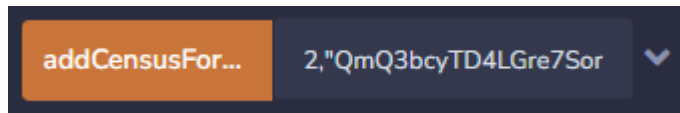
### Asignar una zona específica a un censo

Copiamos el CID del censo:

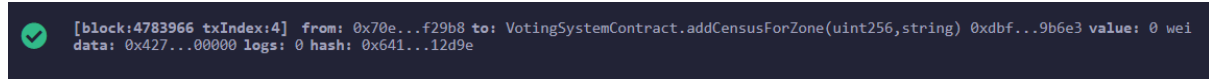
Name ↑	Pin Status	Size
 <b>QmQ3bcyTD4LGre7SoruRNkStcmF2MpewqCcxTvuzH4r5S</b> <small>QmQ3bcyTD4LGre7SoruRNkStcmF2MpewqCcxTvuzH4r5S</small>		194 B
 <b>QmRsyeco2JRANoJFEixktfbTEhATn7rWwfoCgSrk5Dtr</b> <small>QmRsyeco2JRANoJFEixktfbTEhATn7rWwfoCgSrk5Dtr</small>		
 <b>QmS4zeXryphvVfspfaDXEzZmy3Dn2cuQCRhCPgg7qZqV5F</b> <small>QmS4zeXryphvVfspfaDXEzZmy3Dn2cuQCRhCPgg7qZqV5F</small>		

 Share link  
 Copy CID

Y desde el rol de administrador le asignamos una zona específica:

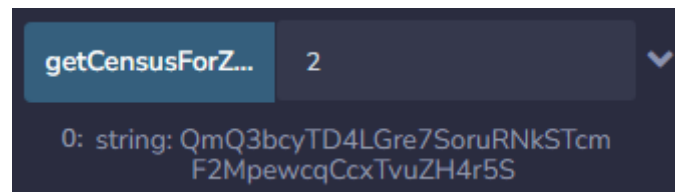


La asignación se ha realizado de manera exitosa.



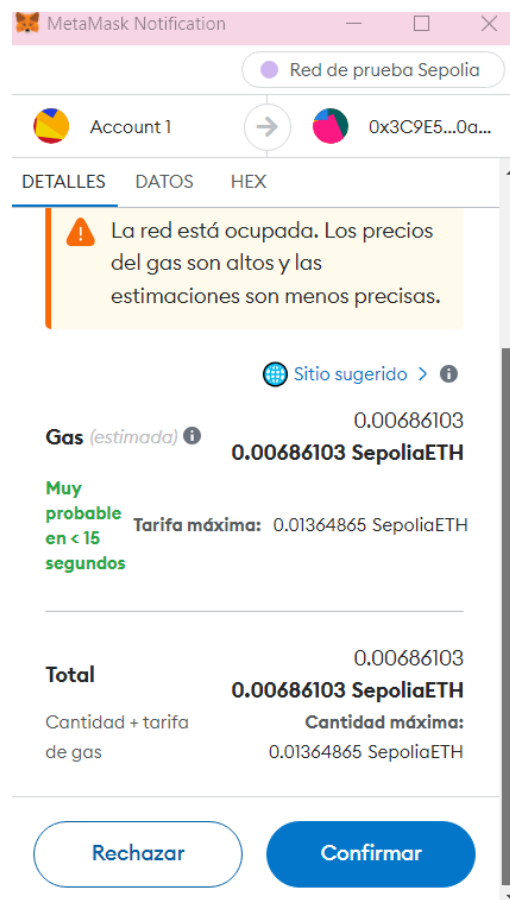
### Mostrar el censo de una zona específica

Gracias al id de una zona, podemos ver el hash de su censo correspondiente:



### Metamask

Aquí podemos observar la notificación de metamask cada vez que realicemos una acción:

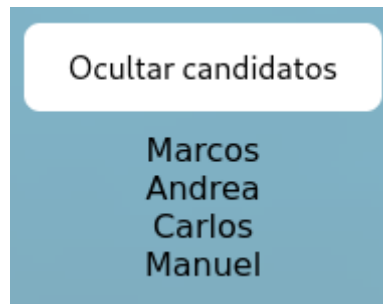


## 9.1 - Interfaz de usuario

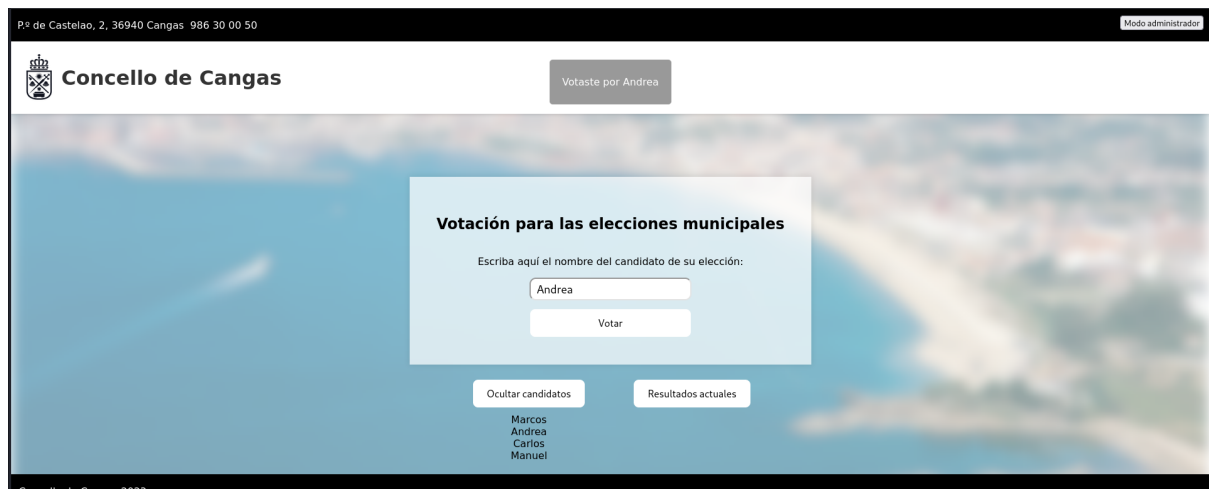
### Vista de usuario

Desde la interfaz se han implementado para el usuario estándar las funcionalidades de mostrar los candidatos disponibles, los resultados actuales y votar.

El usuario visualiza los candidatos disponibles:



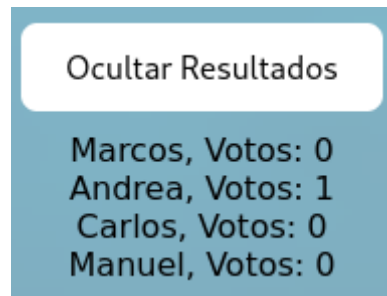
El usuario vota (correctamente):



El usuario vota (incorrectamente):



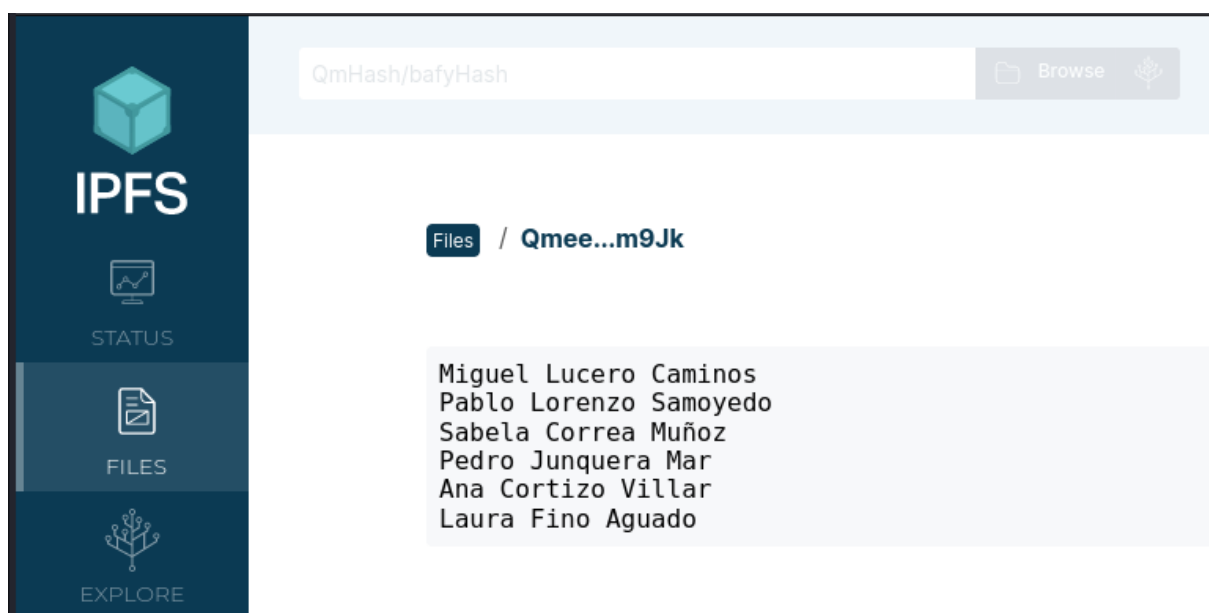
El usuario visualiza los resultados actuales:



### Vista de administrador:

Un administrador podrá subir censos con IPFS desde la interfaz.

El administrador sube un censo:



## 10 - Lean canvas

Lean Canvas es una herramienta de planificación estratégica utilizada en el ámbito empresarial y emprendedor para diseñar y visualizar modelos de negocio de manera concisa y ágil. Fue propuesto por Ash Maurya como una adaptación del Business Model Canvas de Alexander Osterwalder para ser más específico y centrado en el proceso de desarrollo de startups y proyectos innovadores.

A continuación se muestra nuestro modelo Lean Canva donde se presentan diversos aspectos clave de nuestro proyecto. En tal modelo canva se enumeran los recursos clave, la propuesta de valor que queremos transmitir o las relaciones con clientes que queremos mantener.



## 11 - Aspectos Legales

A lo largo del desarrollo de este proyecto se han analizado los aspectos legales que un sistema de votación tiene que tener para que sea justo, legal y oficial. Es por ello que tras varias horas de investigación y argumentación, se ha logrado identificar varias cláusulas legales que son necesarias en un proyecto de este tipo.

En primer lugar, como en cualquier sistema de votación clásico, el voto tendrá que ser anónimo. Esta es la primera cláusula legal que se ha mantenido en la implementación de este proyecto y se ha logrado mediante un estudio refinado de la estructura a implementar en el *Smart Contract*. Para lograr este resultado, no se guarda ninguna información correlacionada del votante y el nombre/identificador del candidato que vota, proporcionando así la anonimidad buscada.

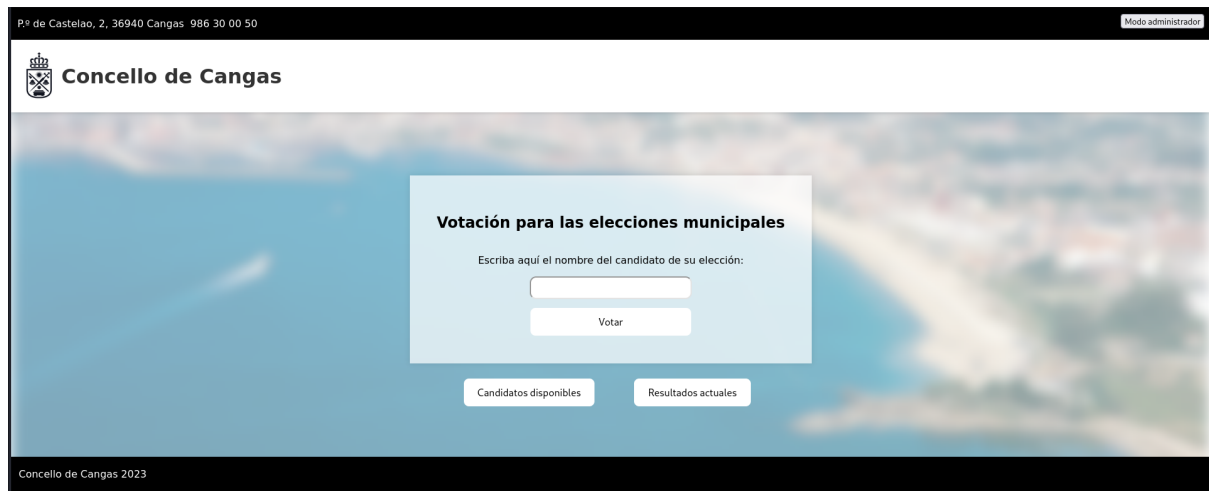
En segundo lugar, la información de los votantes se gestiona de forma segura cumpliendo la actual **Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales**. De forma que las direcciones, nombres completos, dnis, ubicación del censo se maneja cumpliendo dicha normativa.

En tercer lugar, gracias a tener un sistema distribuido, tenemos completa redundancia de los datos del escrutinio. Esto nos brinda la seguridad de que los datos no podrán ser manipulados, borrados o modificados logrando así mayor seguridad y fiabilidad en un sistema de votaciones como este.

Finalmente, también se ha tenido en consideración aspectos como Leyes electorales actuales, accesibilidad y comunicación de toda la información relevante respecto al sistema.

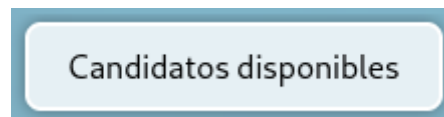
## 12 - Manual de usuario

### **Vista de usuario estándar:**

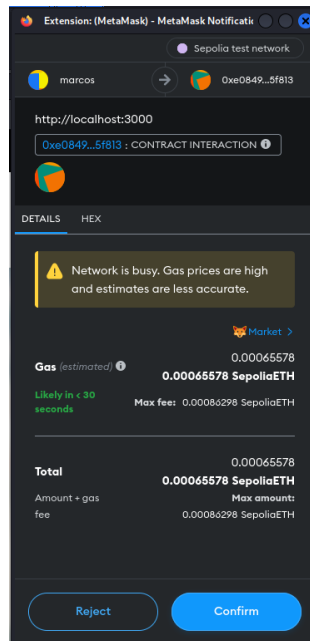


El usuario estándar podrá ver los candidatos disponibles, votar y ver los resultados actuales.

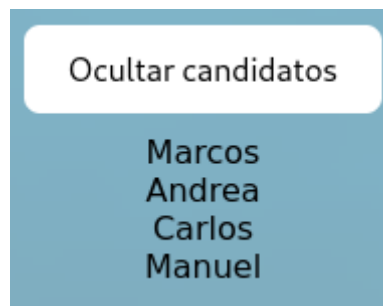
El usuario podrá ver los candidatos disponibles pulsando en el botón “Candidatos disponibles”:



Una vez pulsado, se le pedirá al usuario que realice una transacción:



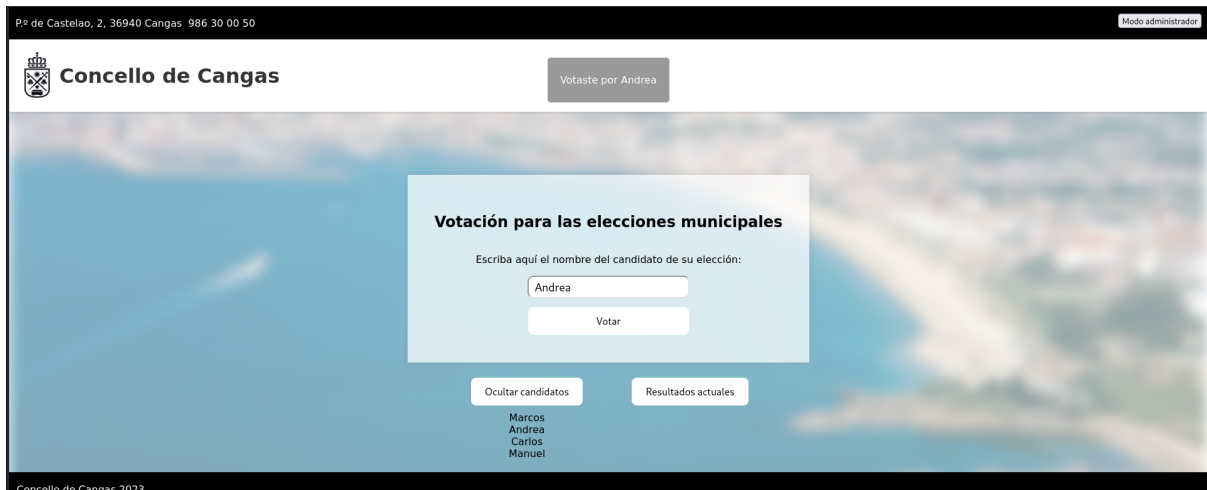
Realizada la transacción, se mostrarán los candidatos disponibles:



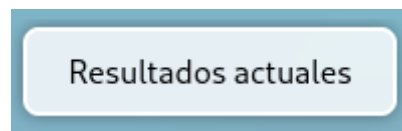
También podrá ocultarlos pulsando de nuevo el botón.

El usuario podrá votar escribiendo el nombre del candidato de su elección y posteriormente deberá pulsar el botón de “Votar”. Una vez más, se le solicitará la transacción. Si se ha hecho correctamente, saldrá un mensaje temporal de confirmación, si hay algún problema con el voto (ha escrito un nombre que no existe, ya ha votado, etc) también se avisará por pantalla al usuario.

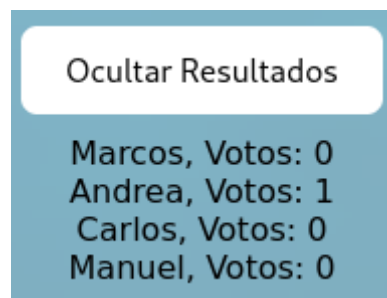




El usuario podrá consultar los resultados actuales de las votaciones si pulsa el botón de “Resultados Actuales”:



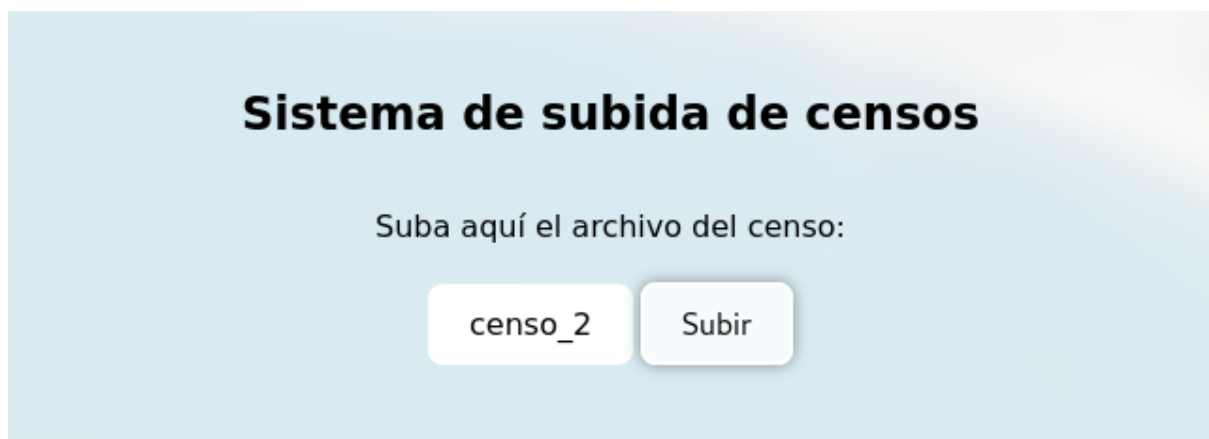
Una vez más, se le solicitará una transacción y finalmente se mostrarán los resultados:



### Vista de administrador:



Un administrador podrá subir censos si pulsa “Browse”, elige el archivo de su elección y posteriormente pulsa “Subir”:



## 13 - Conclusiones

En este proyecto se ha realizado e implementado un sistema de votaciones distribuido de forma segura, flexible y de fácil uso y despliegue. Dicho sistema podría ser empleado para realizar votaciones a niveles locales donde no haya un gran número de votantes.

Durante el desarrollo de este proyecto hemos aprendido a utilizar diversas tecnologías nunca antes vistas. En primer lugar se ha realizado una investigación profunda de las necesidades esenciales de un sistema de votación distribuido. En segundo lugar, se ha implementado la lógica de negocio en un *smart contract* programado en *Solidity*, lenguaje que no se había usado anteriormente por parte de los alumnos. En tercer lugar, se ha empleado IPFS para guardar los archivos de censo de forma distribuida.

Todo este aprendizaje técnico por parte de los alumnos ha sido esencial para lograr implementar un sistema de estas características donde se buscaba conseguir un sistema fiable, seguro y que brindara la anonimidad de un sistema de votación clásico.

Gracias a la metodología empleada, el sistema final ha sido el resultado de varias iteraciones de prototipos con sus respectivas fases de diseño, análisis, implementación y pruebas. Es por ello que podemos calificar el resultado final como robusto y fiable ya que ha sufrido numerosas pruebas unitarias, de integración y de sistema a lo largo de las iteraciones realizadas. Podemos determinar así que la metodología utilizada ha sido realmente acertada para este tipo de proyecto

La interfaz de usuario creada es realmente fácil e intuitiva de utilizar. Cualquier usuario con un dispositivo inteligente sería capaz de realizar su voto a su candidato favorito. Esto proporciona incluso mayor accesibilidad que un sistema de votación clásico donde se debe ir a un sitio indicado a realizar su voto.

En una época donde todos los ciudadanos tienen acceso a un dispositivo electrónico con conexión a internet, se llevará a cabo una transición orgánica a este tipo de sistemas de votación distribuido donde la comodidad, fiabilidad y accesibilidad son una de las miles de ventajas que tienen este tipo de sistemas.

## 14 - Líneas futuras

Tras la finalización de este proyecto existen diversas mejoras que podrían realizarse para lograr un sistema más completo y robusto. Esto se podría organizar en tres partes:

- Mejoras a corto plazo
- Mejorar a medio plazo
- Mejoras a largo plazo

### 14.1 - Mejoras a corto plazo

Una de las posibles mejoras a corto plazo podría ser la implementación de la autenticación del administrador a través de un token como JWT. De esta forma se realizaría un control de acceso bueno ya que el implementado actualmente carece de este sistema ya que simplemente se ha realizado simbólicamente.

Otra posible mejora pequeña que podría sufrir nuestro sistema podría ser la modificación de la interfaz para añadir un apartado para el candidato. De esta forma dicho candidato podría añadir su programa electoral con sus propuestas, medidas y decisiones futuras o cualquier otra información de valor que le ayude a ganar votos.

### 14.2 - Mejoras a medio plazo

Las mejoras a medio plazo son mejoras más consistentes y difíciles de lograr pero que aportan mayor valor al sistema implementado. En primer lugar una posible mejora sería la adición de Orbit DB a este proyecto para guardar la información de los candidatos de una forma más segura y más descentralizado

Otra posible mejora a medio plazo sería la adición de ficheros de log del docker para mantener siempre un registro de que el sistema está siempre activo y evitar así cualquier inconveniente por denegación de servicio

### 14.3 - Mejoras a largo plazo

Finalmente, las mejores a largo plazo son las que mayor utilidad brindan pero las más costosas de realizar. Es por ello que se plantean pero realmente será complicado de lograr.

Una posible mejora a largo plazo podría ser la inclusión de la base de datos del gobierno a nuestro sistema para realizar una primera prueba de este sistema en un caso real.

## 15 - Lecciones aprendidas

Gracias a la implementación y realización de este sistema se han logrado múltiples aprendizajes de gran valor para los alumnos. En primer lugar, se ha aprendido a desarrollar *smart contracts* de una dificultad media con sus respectivas funciones y sus tipos de datos. Incluso llegando a incluir herencia y funciones *payables* en el sistema implementado. En segundo lugar, se ha adquirido conocimiento respecto a IPFS, un sistema de almacenamiento distribuido de ficheros para así lograr redundancia de los datos. Finalmente, en tercer lugar los alumnos han aprendido a integrar estas dos tecnologías anteriormente mencionadas a través de una interfaz web desarrollada por ellos mismos, ampliando así el aprendizaje y el conocimiento adquirido.

Sin embargo, no todo el aprendizaje durante la realización de este proyecto ha sido técnico. El diseño de diagramas inicialmente y del estudio del arte respecto a este tipo de sistemas ha sido realmente útil y de valor para los estudiantes. Además, implementar este proyecto en parejas ha sido todo un reto ya que la coordinación entre las dos partes ha sido realmente esencial para lograr el resultado conseguido. Mediante reuniones cortas y concisas para actualizar el estado del proyecto y la ayuda en caso de que uno de los estudiantes estuviera atascado/a en la tarea asignada, ha sido de gran utilidad para conseguir implementar correctamente un sistema de tal características

### 15.1 - Incidencias

En todo proyecto surgen incidencias y dificultades y este no iba a ser menos. En la implementación de este sistema de votaciones los alumnos se han ido encontrando con diversas dificultades pero realmente la mayor incidencia se produjo con OrbitDB que dado a las limitaciones de tiempo y la falta de documentación de valor sobre esta tecnología, fue imposible conseguir que funcionara integrado con el sistema diseñado.

La segunda incidencia importante que hemos tenido fue respecto a la interfaz web, ya que se tiene que realizar dos llamadas al método “`hadleShowResults()`” para que la propia interfaz muestra los resultados de las elecciones.

En el primer prototipo del proyecto tuvimos problemas al encontrar el motivo del error generado cuando se importaba el fichero *Ownable.sol* ya que decía que no estaba realmente compilado para esa versión y Remix nos permitía compilar nuestro fichero con dicho error. Tras una búsqueda por internet encontramos la solución cambiando la versión del compilador de remix a una versión más adecuada, en concreto la 0.8.21 y el error se solventó de inmediato.

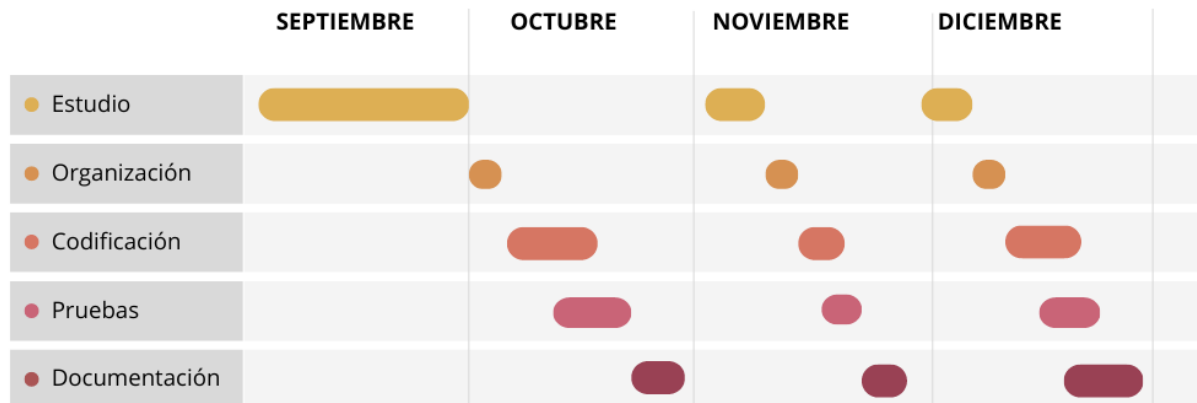
También, tuvimos que darnos cuenta de que solidity no tienen un método nativo bueno para comparar dos cadenas de texto. Esto es realmente frustrante porque

hay que buscar una solución alternativa creando un hash y comparándolos para así concluir que ambas cadenas son iguales.

Además, otro problema apareció cuando estábamos desarrollando el constructor del *smart contract* ya que nos indicaba que el constructor de Ownable tenía que recibir un parámetro pero no sabíamos cómo realizar esta acción ya que no la habíamos visto en clase. Realizando una búsqueda rápida en internet encontramos la solución efectiva.

## 16 - Planificación

Para plasmar la planificación de este proyecto, hemos diseñado un diagrama de Gantt que comienza al inicio de la asignatura:



El primer mes fue de aprendizaje y familiarización con Solidity y Smart Contracts. Una vez adquirimos todos los conocimientos básicos, decidimos cómo orientar nuestro contrato inteligente y organizamos el trabajo entre los miembros del equipo y procedimos a la codificación. La codificación de octubre fue la que más tiempo llevó ya que tuvimos que sentar las bases de nuestro proyecto, a medida que codificamos, realizamos pruebas para asegurarnos de que íbamos por un buen camino. Una vez todo listo, documentamos todo lo que hemos realizado. Para las siguientes entregas seguimos un proceso similar, sin embargo, en el último mes comenzamos la documentación antes de la finalización del código y de las pruebas ya que fuimos abordando apartados en los que no era necesario tener el proyecto finalizado.

## 17 - Referencias

1. Sufragio universal: [https://es.wikipedia.org/wiki/Sufragio\\_universal](https://es.wikipedia.org/wiki/Sufragio_universal)
2. Sufragio universal español:  
<https://blog.congreso.es/sufragio-universal-espana/>
3. Blockchain for Electronic Voting System—Review and Open Research Challenges: <https://www.mdpi.com/1424-8220/21/17/5874>
4. Blockchain-Based E-Voting System:  
<https://ieeexplore.ieee.org/abstract/document/8457919>
5. Empresas dirigidas a la votación usando tecnología blockchain  
<https://www.criptonoticias.com/comunidad/transparencia-electoral-5-plataformas-blockchain-para-votaciones/>
6. Sistema de votación blockchain en Estonia:  
<https://www.bbva.com/es/innovacion/mundo-gobiernos-nube/>
7. Países que probaron sistemas de votación blockchain:  
<https://medium.com/couger-blog/what-countries-and-states-that-have-trialed-blockchain-voting-learned-f0a9f5e98a43>