

# Memoria Laboratorios BC

## Ejercicio 6

María Andrea Ugarte Valencia

Bahdon Barkhad

Marcos Villar Avión

# ÍNDICE

<b>1 - Análisis y Definición del Escenario.....</b>	<b>3</b>
1.1 - Definición del Escenario.....	3
1.2 - Análisis.....	3
<b>2 - Diseño.....</b>	<b>4</b>
2.1 - Casos de Uso.....	4
2.2 - Contrato inteligente.....	6
2.2.1 - Variables globales y Structs.....	6
2.2.2 - Funciones y modifiers.....	6
2.3 - Usuarios del sistema.....	8
<b>3 - Implementación.....</b>	<b>9</b>
<b>4 - Pruebas.....</b>	<b>10</b>
4.1 - Añadir Candidatos.....	10
4.2 - Eliminar Candidatos.....	12
4.3 - Votar a un Candidato.....	14
4.4 - Mostrar el ganador.....	15
4.5 - Mostrar Todos los Resultados.....	16
4.6 - Modificar los Resultados Pagando.....	17
<b>5 - Problemas encontrados.....</b>	<b>19</b>

# 1 - Análisis y Definición del Escenario

## 1.1 - Definición del Escenario

Para la realización de este ejercicio se ha seleccionado un sistema de votación. Nuestro sistema serviría como una plataforma para que los distintos usuarios logren votar a su candidato preferido a través de la utilización de una blockchain.

Se ha decidido implementar este sistema debido a su naturaleza y los requisitos del sistema, logrando así distintos beneficios enumerados a continuación:

- Seguridad y Confianza
  - Los datos no pueden ser modificados una vez almacenados en la blockchain proporcionando así un alto nivel de confianza e integridad
- Transparencia
  - Los datos son visibles para todos los nodos de la red
- Reducción de Fraude
  - Al no ser un sistema centralizado, la inmutabilidad de los datos reduce el riesgo de fraude
- Registro de historial
  - Contiene un registro histórico completo de todas las transacciones.

Es por ello que los beneficios que nos otorga un blockchain son tan adecuados para un sistema de votación y siendo este el principal motivo de nuestra elección. Sin embargo, en nuestro caso hemos querido añadirle un toque humorístico y dar la posibilidad de alterar el resultado de las elecciones si alguien paga por ello.

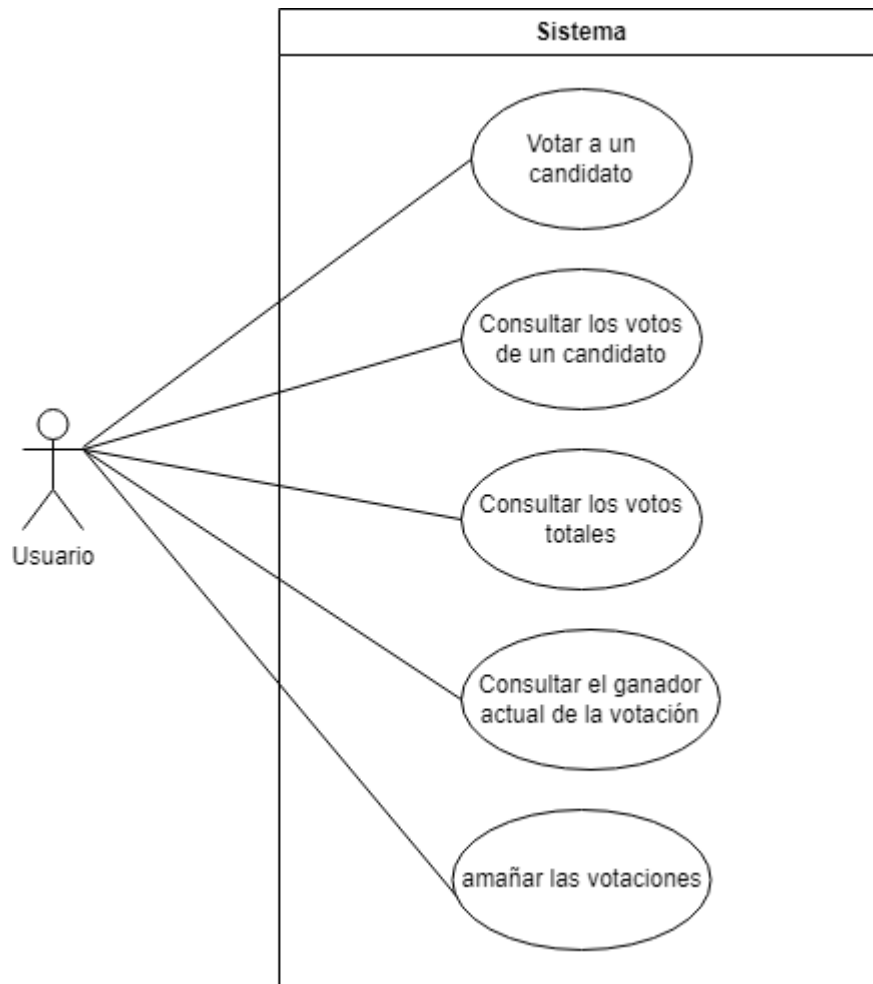
## 1.2 - Análisis

Tras analizar un sistema de votación en profundidad, hemos identificado que la blockchain desarrollada debería ser pública como Ethereum, ya que todos los usuarios deben de ser capaces de votar al presidente que ellos deseen. Además, este sistema tiene que ser fiable, tolerante a fallos, transparente y seguro, siendo el desarrollo de una blockchain idóneo para este problema.

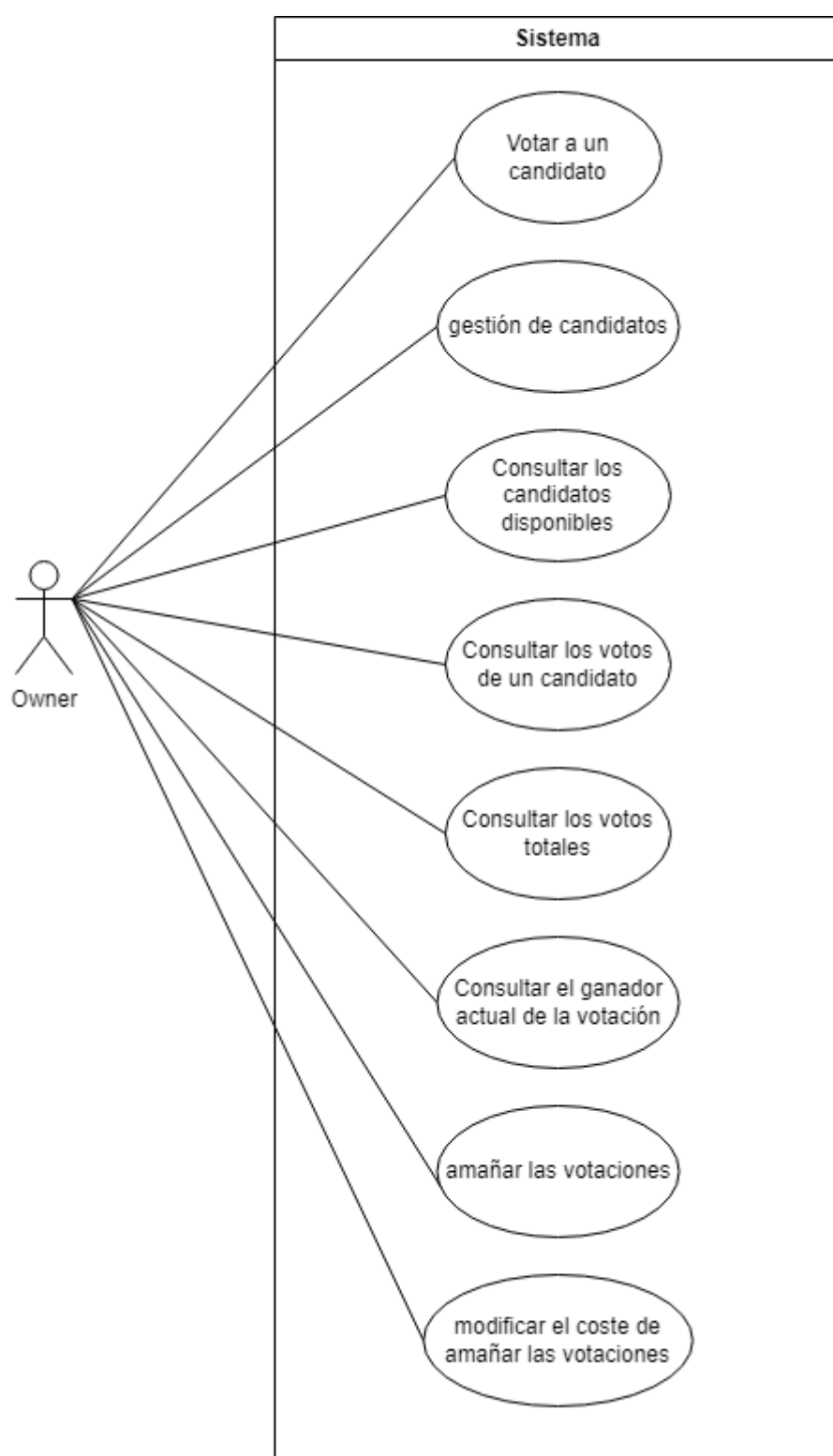
## 2 - Diseño

### 2.1 - Casos de Uso

- Votante



- Owner



## 2.2 - Contrato inteligente

### 2.2.1 - 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

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

### 2.2.2 - 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

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
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

## 2.3 - Usuarios del sistema

En nuestro sistema como se ha indicado con anterioridad, tendremos 3 tipos de usuarios:

- Votante
- Candidato
- Owner / Propietario del *smart Contract*

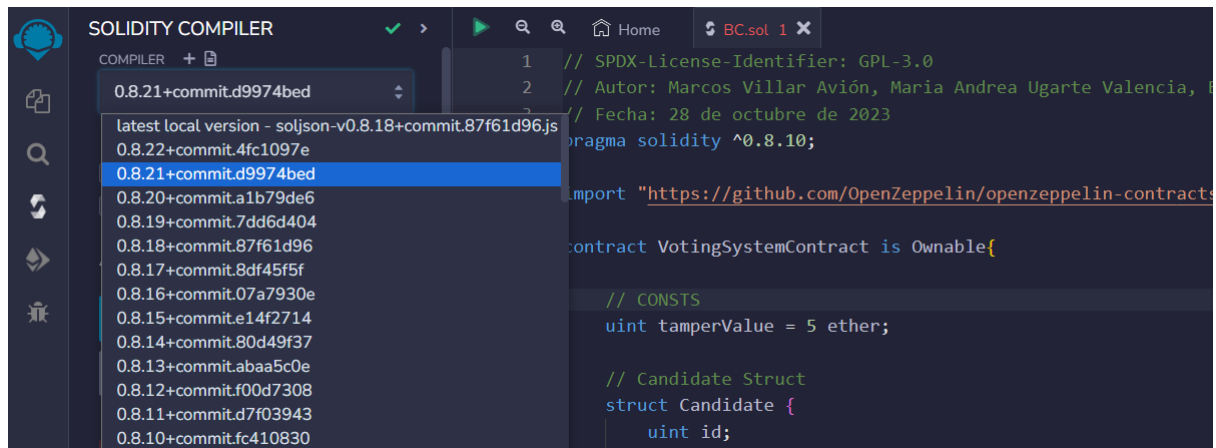
La función de candidato es completamente pasiva ya que no tienen funciones diferentes a un usuario normal y se comportan como un votante más que puede ser votado.

Respecto a un propietario del *smart contract*, este puede realizar una serie de acciones de forma privilegiada en el sistema tal como añadir nuevos candidatos o eliminar un determinado candidato pero también puede votar y comportarse como un votante más.



### 3 - Implementación

Para compilar y ejecutar el *smart contract* desarrollado tendremos que realizar una tarea para indicarle a remix la versión del compilador de Solidity que queremos usar. Para ello, una vez importado el fichero, vamos a la pestaña de compilación y



Y vamos a seleccionar la versión 0.8.21 que es con la que hemos realizado todas las pruebas. Con esta modificación nuestro contrato debería de compilarse sin problema alguno

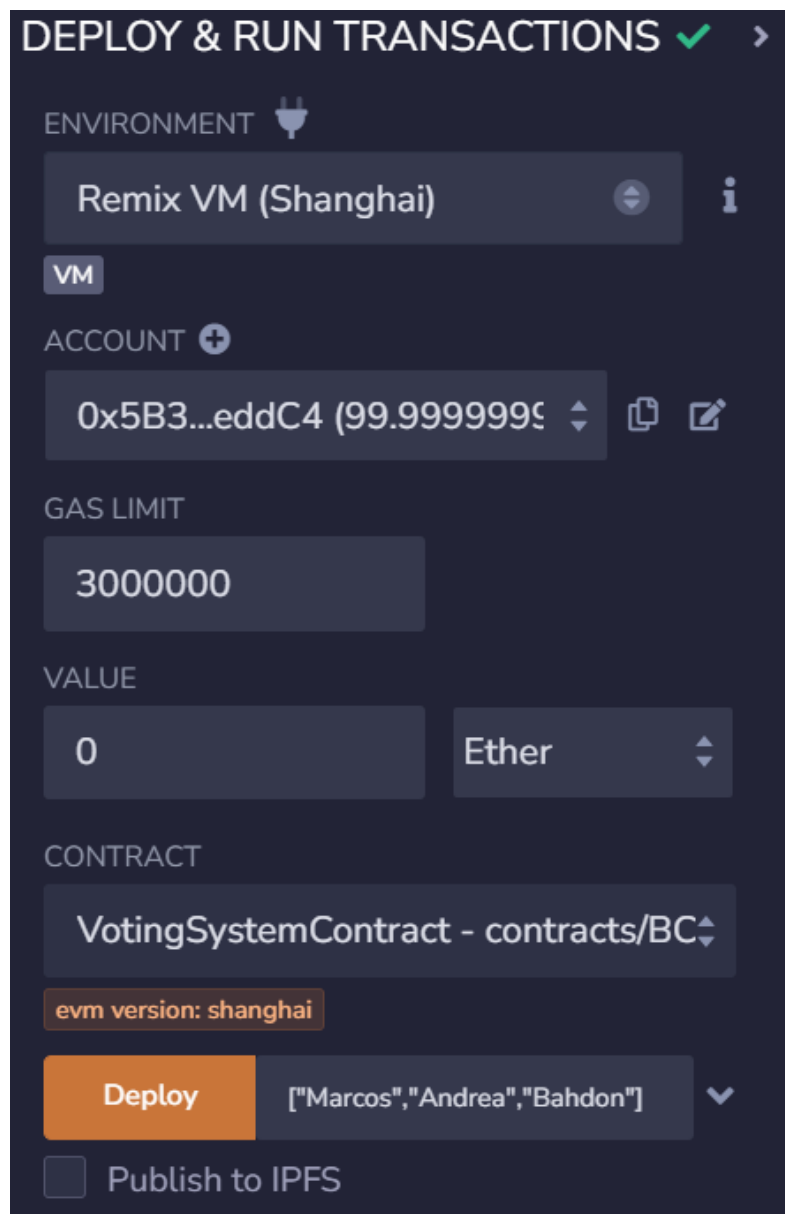
## 4 - Pruebas

### 4.1 - Añadir Candidatos


Una de las primeras pruebas que vamos a realizar es que las funciones de añadir y de eliminar candidatos funcionen correctamente. Más en detalle, la función de eliminar candidato solo podrá eliminar el nombre especificado en caso de que exista en nuestra base de datos, si no la función avisará del error.

Primero de todo, vamos a compilar el contrato y ejecutarlo introduciendo un array de nombres para que se pueda ejecutar el constructor del *smart contract*. En nuestro caso pondremos:

```
["Marcos","Andrea","Bahdon"]
```



DEPLOY & RUN TRANSACTIONS ✓ >

ENVIRONMENT 

Remix VM (Shanghai) ⓘ

VM

ACCOUNT +

0x5B3...eddC4 (99.9999999) ⓘ ✎

GAS LIMIT

3000000

VALUE

0 Ether

CONTRACT

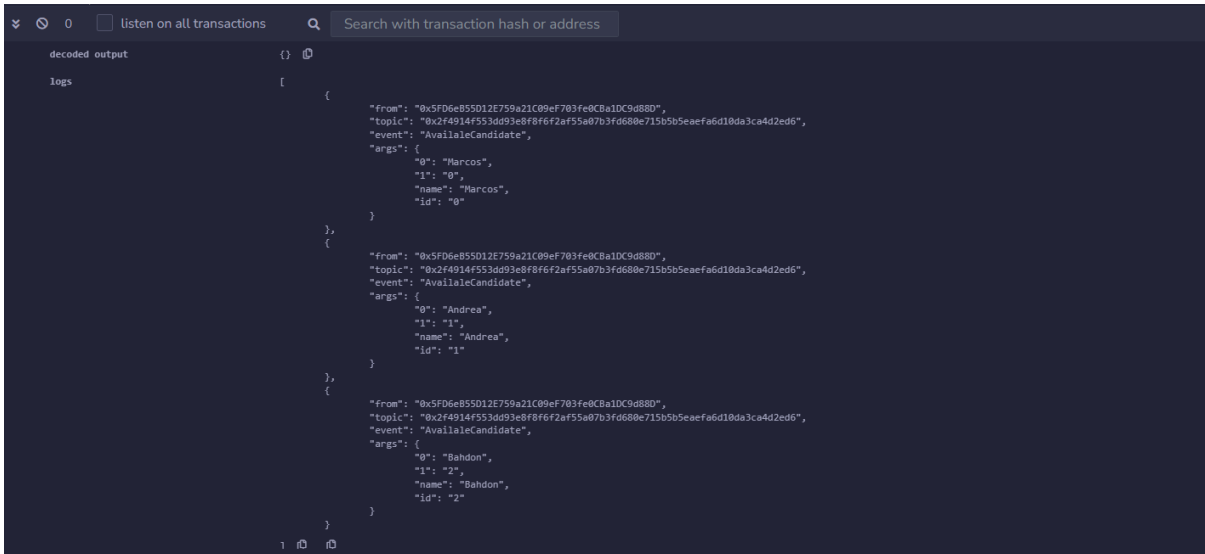
VotingSystemContract - contracts/BC ⓘ

evm version: shanghai

Deploy ["Marcos","Andrea","Bahdon"] ▼

☐ Publish to IPFS

A continuación vamos a ejecutar la función de `showAvailableCandidates()` para mostrar los candidatos que tenemos disponibles en nuestro sistema y sus respectivos identificadores.



```



decoded output {}
logs [
  {
    "from": "0x5FD6e855012E759a21C09eF703fe0C8a1DC9d880",
    "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eaeaf6d10da3ca4d2ed6",
    "event": "AvailableCandidate",
    "args": {
      "0": "Marcos",
      "1": "0",
      "name": "Marcos",
      "id": "0"
    }
  },
  {
    "from": "0x5FD6e855012E759a21C09eF703fe0C8a1DC9d880",
    "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eaeaf6d10da3ca4d2ed6",
    "event": "AvailableCandidate",
    "args": {
      "0": "Andrea",
      "1": "1",
      "name": "Andrea",
      "id": "1"
    }
  },
  {
    "from": "0x5FD6e855012E759a21C09eF703fe0C8a1DC9d880",
    "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eaeaf6d10da3ca4d2ed6",
    "event": "AvailableCandidate",
    "args": {
      "0": "Bahdon",
      "1": "2",
      "name": "Bahdon",
      "id": "2"
    }
  }
]

```

Una vez verificado que nuestros tres candidatos están insertados en el *smart Contract* vamos a intentar añadir otro candidato y ver si lo inserta correctamente. Digamos que queremos insertar una nueva candidata llamada Paula (obviamente solo el propietario del *smart Contract* puede insertar y eliminar candidatos)



**Deployed Contracts**

▼ VOTINGSYSTEMCONTRACT AT 0:  

Balance: 0 ETH

<b>addCandidate</b>	<input type="text" value="Paula"/>	▼
<b>tamperElections</b>	<input type="text" value="string candidateName"/>	▼
<b>vote</b>	<input type="text" value="uint256 _candidate_id"/>	▼

Tras insertar a Paula como una nueva candidata, podemos ejecutar la función `showAvailableCandidates()`

```

    },
    {
      "from": "0x5FD6e855D12E759a21C09ef703fe0C8a1DC9d88D",
      "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": "0xAb8483F64d9C6d1EcF9b849Ae677d03315835cb2"
  }
}
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

```

## 4.2 - Eliminar Candidatos

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 occured: 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:

```

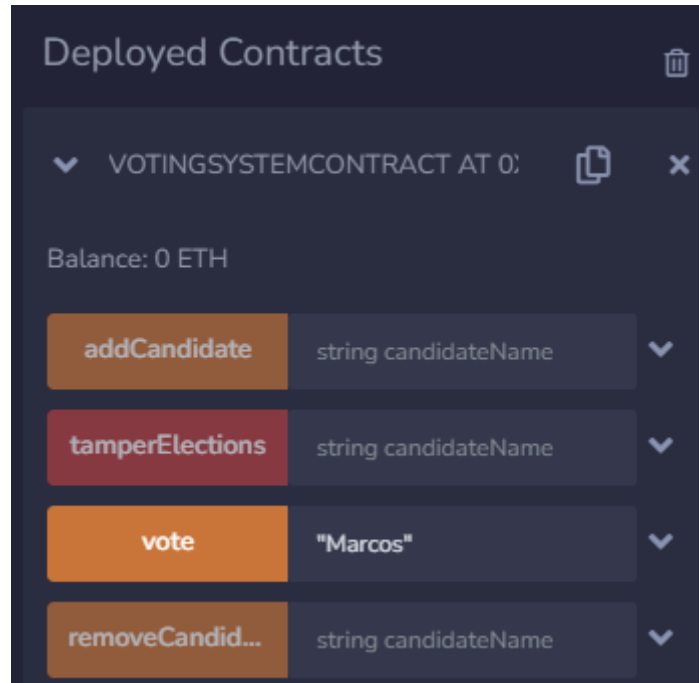
{
  "from": "0x5FD6eB55D12E759a21C09eF703fe0C8a1DC9d88D",
  "topic": "0x2f4914f553dd93e8f8f6f2af55a07b3fd680e715b5b5eaeafa6d10da3ca4d2ed6",
  "event": "AvailaleCandidate",
  "args": {
    "0": "Paula",
    "1": "0",
    "name": "Paula",
    "id": "0"
  }
},
{
  "from": "0x5FD6eB55D12E759a21C09eF703fe0C8a1DC9d88D",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

## 4.3 - 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

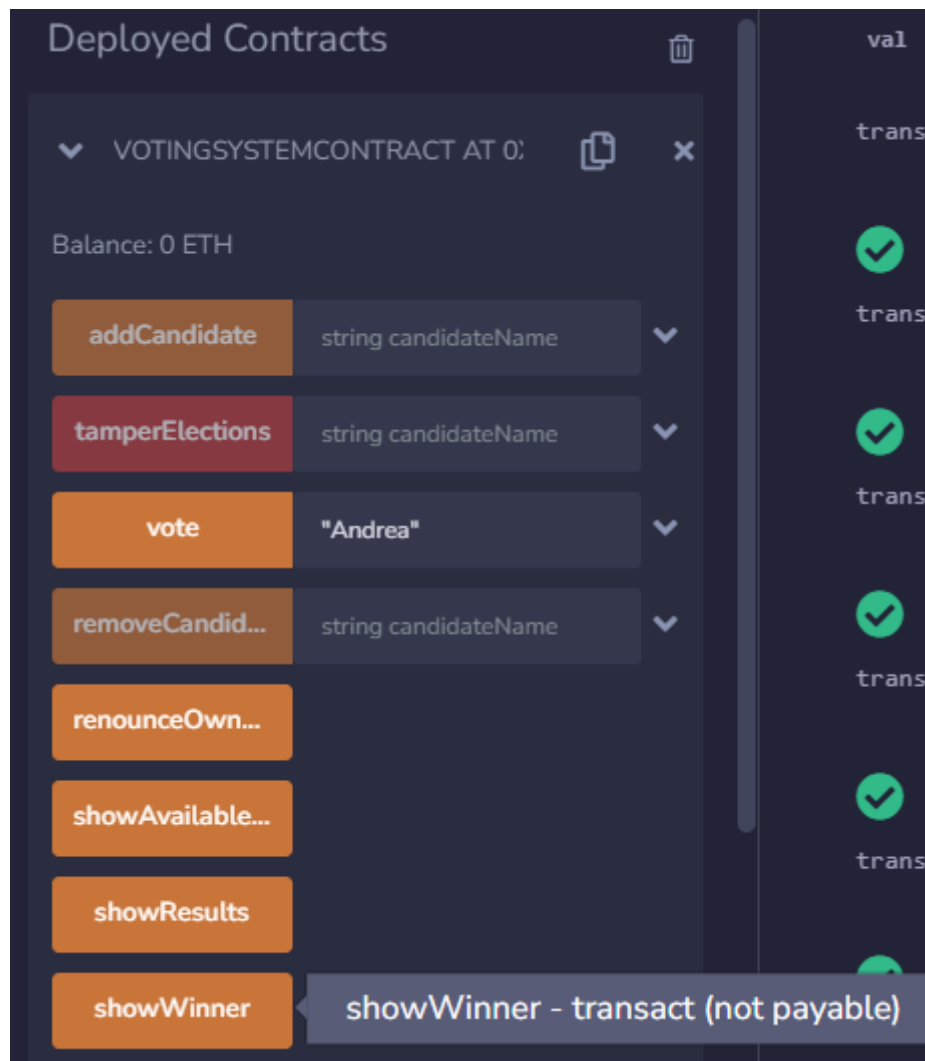
```
transact to VotingSystemContract.vote pending ...  
  
transact to VotingSystemContract.vote errored: Error occurred: revert.  
  
revert  
  The transaction has been reverted to the initial state.  
  Reason provided by the contract: "You have already voted".  
  Debug the transaction to get more 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
```

## 4.4 - Mostrar el 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": "0xec7b07f92bbe36adaa91eaf451da184b543d99b53d1d2f4826d9de4577750d2",
    "event": "Winner",
    "args": {
      "0": "Andrea",
      "1": "4",
      "name": "Andrea",
      "votes": "4"
    }
  }
]
```

## 4.5 - 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"
    }
  }
]
```



## 4.6 - Modificar los Resultados 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": "0xec7b07f92bbee36adaa91eaf451da184b543d99b53d1d2f4826d9de4577750d2",
              "event": "winner",
              "args": {
                "0": "Bahdon",
                "1": "6",
                "name": "Bahdon",
                "votes": "6"
              }
            }
          ]
```

## 5 - Problemas encontrados

Durante la realización de la práctica se han encontrado una serie de problemas dificultando el proceso de desarrollo.

El primero de ellos fue 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.

El segundo de ellos fue 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.

El último problema encontrado 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.