

---

# Trabajo Final de Experto

PLATAFORMA WEB DE GESTIÓN DE  
TOKENS CORPORATIVOS SOBRE ALASTRIA

Curso Experto en Desarrollo de Aplicaciones Blockchain  
2018-2019

María Salgado Iturrino

*Tutor: Fidel Panigua Diez*

# Índice

TFE	3
1. Objetivo e introducción	4
2. Estado del arte y tecnologías utilizadas	5
3. Descripción	9
4. Análisis y modelo del sistema propuesto	17
5. Despliegue	23
6. Pruebas	39
7. Conclusiones	46
8. Referencias	47

*A los compañeros de este curso experto,  
que nos volvamos a encontrar en este mundo blockchain.*

*A todos los compañeros de IECISA, por enseñarme tanto.*

*A mi familia, por todo.*

*A Juanan, por esos tulipanes naranjas.*

# 1. Objetivo e introducción

El objetivo de este Trabajo de Fin de Experto es realizar un proyecto en blockchain en el que se apliquen los conocimientos adquiridos a lo largo del curso. Como demostración de ello, se destaca que para conseguir poner en marcha la solución desarrollada es necesario disponer de:

- Conceptos básicos sobre tecnología blockchain, en concreto Quorum.
- Habilidades de programación y desarrollo de aplicaciones web (HTML, CSS y JavaScript) para el frontend.
- Experiencia con la librería web3 para integrar frontend y blockchain.
- Habilidades de programación de smart contracts (Solidity) y manejo del entorno (Remix) y de la herramienta de testing (Truffle).

El presente documento recoge una memoria de todos los pasos llevados a cabo hasta finalmente desarrollar una aplicación que interactúa con la red Alastria a través del nodo de la UNIR. Se comienza con un breve estado del arte teórico en el capítulo 2. A continuación, en el capítulo 3 se describe la solución implementada detalladamente ya que, de acuerdo con el tutor, se diseñó una solución distinta a la planteada en el enunciado del proyecto, consiste en una plataforma de gestión de tokens corporativos para empleados. En el capítulo 4 se presenta un análisis que justifica las decisiones tomadas en el diseño de los smart contracts. Después, en el capítulo 5 se exponen las tres fases de despliegue que han tenido lugar. En el capítulo 6 se detallan todas las pruebas unitarias realizadas y sus resultados. Finalmente, el capítulo 7 recoge las conclusiones del proyecto y el 8 las referencias.

## 2. Estado del arte y tecnologías utilizadas

### Ethereum

Tras el desarrollo de Bitcoin en 2009 por Satoshi Nakamoto, Vitalik Buterin crea Ethereum basándose en la tecnología de la cadena de bloque subyacente como herramienta de consenso distribuido y no solamente en el concepto de criptomoneda. Como visionario de la potencia que tiene que los activos digitales sean controlados por un código que implemente reglas arbitrarias (smart contract) o por organizaciones autónomas descentralizadas (DAO), desarrolló Ethereum. Ethereum proporciona una cadena de bloques con un lenguaje de programación Turing completo en el que se puedan codificar funciones. Esta descripción está basada en su white paper (Buterin, 2015) en el que se profundiza en todos sus detalles.



*Ilustración 1. Logo de Ethereum*

Los desarrollos sobre Ethereum se realizan en el entorno Remix, un IDE accesible desde el navegador en la url <http://remix.ethereum.org>. Además de la mainnet de Ethereum existen varias testnets como Rinkeby o Ropsten a las que es posible conectarse para desplegar smart contracts de prueba.



*Ilustración 2. Logo de Remix*

Otro entorno de desarrollo que se utiliza en todo el mundo que incorpora un framework para pruebas unitarias es Truffle<sup>1</sup>. Permite conectarse a cualquier red blockchain que utilice la Ethereum Virtual Machine (EVM).



*Ilustración 3. Logo de Truffle*

Para desplegar una red blockchain en local, la suite de Truffle proporciona una herramienta específica: Ganache<sup>2</sup>. Ganache está disponible en su versión escritorio aunque en este proyecto se ha utilizado la herramienta en línea de comandos.



*Ilustración 4. Logo de Ganache*

En una primera fase de desarrollo se recomienda arrancar rápidamente una red Ethereum en local con cuentas disponibles en la que realizar pruebas, ejecutar comando e inspeccionar el estado.

## Quorum

Quorum es una versión empresarial de Ethereum. Aporta alto rendimiento y permite realizar transacciones privadas entre un grupo autorizado de participantes conocidos. Es un proyecto open source impulsado por JP Morgan<sup>3</sup>.

---

<sup>1</sup> <https://truffleframework.com/>

<sup>2</sup> <https://truffleframework.com/ganache>

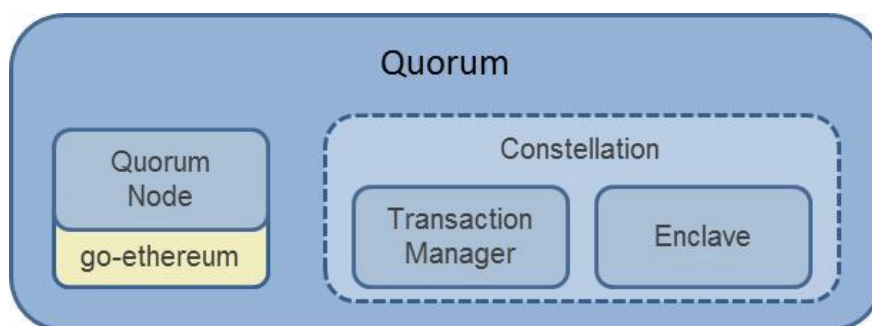
<sup>3</sup> <https://www.jpmorgan.com/global/Quorum>



*Ilustración 5. Logo de Quorum*

De forma muy resumida, la arquitectura de Quorum (JP Morgan, 2016) es lo que permite las diferencias mencionadas anteriormente respecto a Ethereum. Las transacciones privadas se consiguen mediante la separación entre el estado y una sección privada. Utiliza intercambios de mensajes cifrados entre claves (consultar Constellation<sup>4</sup> y Tessera<sup>5</sup>) para la transferencia de datos privados a los participantes de la red.

Quorum también soporta mecanismos de consenso alternativos sin necesidad de Proof of Work (PoW) o Proof of Stake (PoS) si se trata de una red permissionada.



*Ilustración 6. Arquitectura de Quorum*

## Alastria

Alastria<sup>6</sup> es el primer consorcio blockchain español. Su propósito es crear una infraestructura semipública común tanto para empresas como para instituciones públicas.

---

<sup>4</sup> <https://github.com/jpmorganchase/constellation>

<sup>5</sup> <https://github.com/jpmorganchase/tessera>

<sup>6</sup> <https://alastria.io/#1>

Se puede resumir Alastria como una “red Blockchain/DLT semipública, independiente, permitida y neutral, diseñada para ser conforme con la regulación existente, que permite a los asociados experimentar estas tecnologías en un entorno cooperativo.



*Ilustración 7. Logo de Alastria*

Alastria está organizado en comisiones y grupos de trabajo. Actualmente dispone de una primera testnet que se llama Arrakis y de una testnet definitiva que se llama Telsius. Se puede monitorizar el estado de ambas a través de netstats <sup>7</sup>. La colaboración en Alastria se realiza utilizando los repositorios de GitHub <sup>8</sup>



*Ilustración 8. Logo de GitHub*

---

<sup>7</sup> <http://netstats.telsius.alastria.io/> y <http://netstats.arrakis.alastria.io/>

<sup>8</sup> <https://github.com/alastria>



### 3. Descripción

La gamificación (Gaitán, 2013) es una técnica que traslada la mecánica de los juegos a otros ámbitos. En el medio profesional se está aplicando en muchas empresas a través de recompensas en acciones concretas. Debido a su carácter lúdico, genera una experiencia positiva en el usuario porque realmente consigue motivar a los empleados. “La idea de gamificar no es crear un juego, sino valernos de los sistemas de puntuación-recompensa-objetivo que normalmente componen a los mismos.”

Existen, fundamentalmente, dos tipos de técnicas. Por una parte, las técnicas mecánicas, como las que muestra la ilustración 9, que recompensan al usuario en función de los objetivos alcanzados. Y, por otra parte, las técnicas dinámicas como las que muestra la ilustración 10 en las que se hace referencia a la motivación del propio usuario para jugar.



*Ilustración 9. Técnicas mecánicas de la gamificación*



*Ilustración 10. Técnicas dinámicas de la gamificación*

En base a este pequeño análisis se justifica la idea de crear unos tokens corporativos que las empresas puedan utilizar para recompensar y motivar a sus empleados.

Desarrollar esta solución utilizando la blockchain es interesante porque:

- Garantiza la integridad de las transacciones con tokens.
- Las empresas no son “un banco” de tokens, no existe intermediario.

Utilizar la red de Alastria tiene mucho sentido en este caso porque los tokens, fuera del alcance de este proyecto, podrían ser interoperables entre empresas distintas. Esto genera múltiples casos de uso nuevos, por ejemplo, canjear los tokens que has obtenido en tu empresa en otra distinta o para promover la colaboración entre empleados de distintas empresas o subcontratas.

Los tokens, a los que en la mayor parte del proyecto se ha tratado de forma genérica se han denominado GLUPIs (GPI) y cumplen con el estándar ERC20 tal y como se explicará en detalle en el capítulo siguiente.

Aunque existen múltiples funcionalidades que podría realizar esta plataforma, ha sido necesario acotarlas a una primera fase en la que se recoge el alcance de este

TFE. En el presente los tokens no son interoperables, son privados para uso interno de los empleados de una misma empresa.

La plataforma consta de dos perfiles de usuarios: empresas y empleados.

Una empresa está definida en el sistema por:

- Una cuenta de Alastria (una dirección pública válida). Es su identificador único.
- El nombre de la empresa.
- El CIF de la empresa.

Un empleado se define en el sistema por:

- Una cuenta de Alastria (una dirección pública válida). Es su identificador único.
- El nombre del empleado.
- El número de empleado.
- La empresa a la que pertenece.

En el diseño se ha focalizado en que se preserve la privacidad tanto del empleado como de la empresa. Por este motivo, aunque se conozca la cuenta de Alastria de un empleado solo podrá accederse desde la cuenta de la empresa en la que trabaja o desde la suya propia. De igual manera, una empresa solo podrá listar los empleados que trabajan para ella.

Las operaciones que puede realizar una empresa son:

- Loguearse, cerrar sesión y visualizar su información.
- Registrarse en la plataforma introduciendo sus datos. Se añade al sistema y se le asignan automáticamente 100 GPIs.
- Registrar empleados en su empresa.
- Emitir tokens para esos empleados.
- Listar los empleados que tiene.

- Consultar el saldo de un empleado.
- Consultar el saldo de la empresa.

Las operaciones que puede realizar un empleado son:

- Loguearse, cerrar sesión y visualizar su información.
- Transferir a sus compañeros (que trabajen en la misma empresa) tokens.
- Canjear sus tokens por premios. Los tokens se transfieren de vuelta a la empresa.

Cada usuario (bien sea empresa o empleado) está asociada a una cuenta. En todo el proyecto, documentación y pruebas se ha utilizado la siguiente asignación de las mismas:

- accounts[0] es la cuenta administradora de la PlataformaTokens.
- accounts[1] es la empresa IECISA.
- accounts[2] es Maria Salgado, empleado 1 de IECISA.
- accounts[3] es Juan Perez, empleado 2 de IECISA.
- accounts[4] es la empresa UNIR.
- accounts[5] es Fidel Garcia, empleado 1 de UNIR.
- accounts[6] es Belen Sanchez, empleado 2 de UNIR.

La plataforma tiene una interfaz web de acceso que permite iniciar sesión a cualquiera de los perfiles tal y como se muestra en la ilustración 11. Mediante algunas capturas de pantalla, en las próximas paginas de este documento se pretende reproducir un customer journey.

Existe una cuenta administradora de la plataforma a la cual se le asignan todos los tokens en el momento de inicio. Esta cuenta asigna 100 GLUPIs a cada empresa en el momento en que se registra.

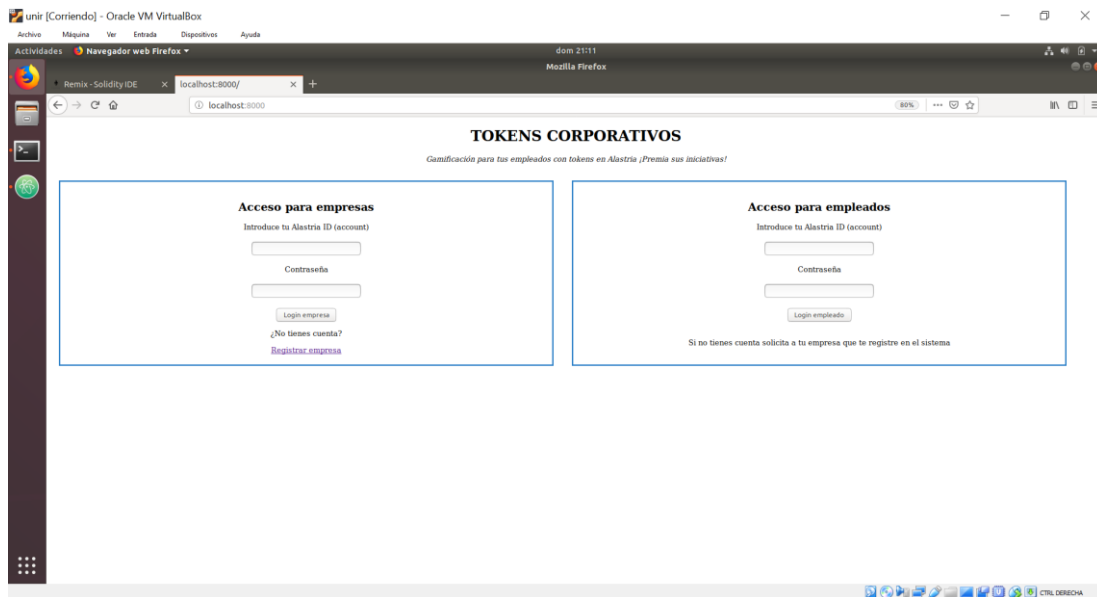


Ilustración 11. Interfaz de acceso a la plataforma

Comenzando desde el caso más inicial, lo primero que habría que hacer sería seleccionar “Registrar empresa” y rellenar los campos del formulario tal y como se muestra en la ilustración 12.

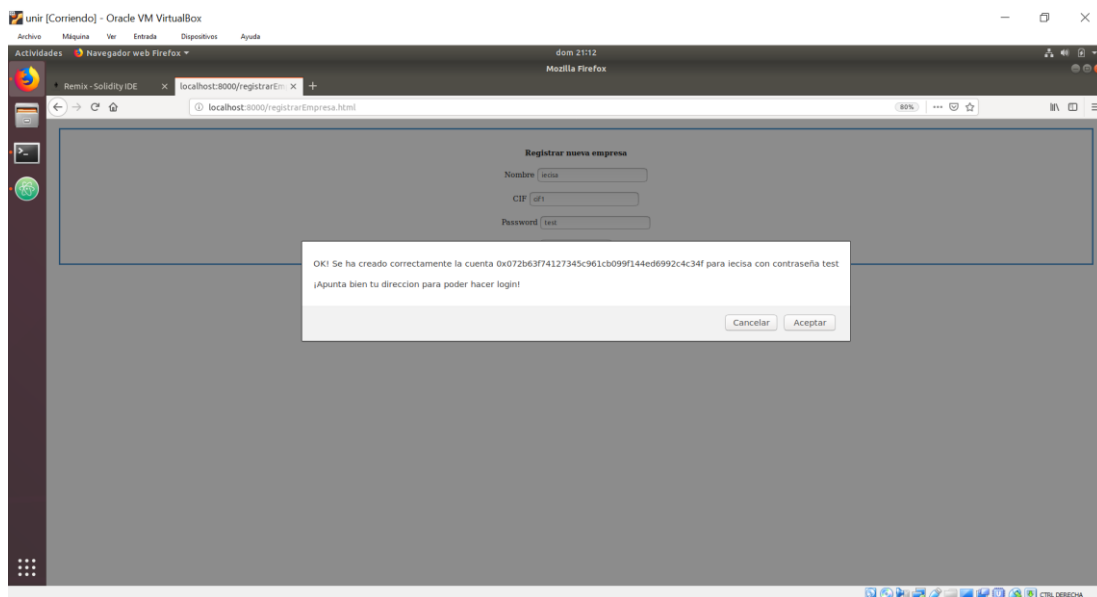


Ilustración 12. Interfaz de registro de una empresa

A continuación, ya podríamos iniciar sesión con esta cuenta para acceder a la interfaz web, que se muestra en la figura 13, con las distintas funcionalidades para la empresa. Los resultados de dichas acciones se muestran en el cuadro inferior titulado “Consola de resultados”.

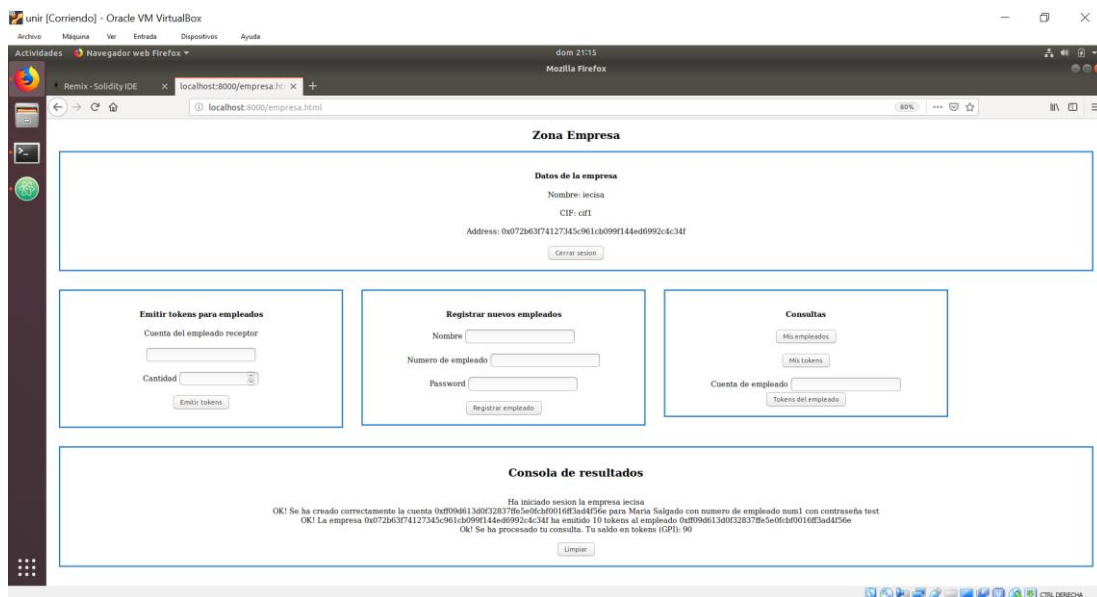


Ilustración 13. Interfaz para las empresas

Como se muestra en la consola de resultados, desde esta interfaz hemos creado un empleado por lo que si cerramos sesión y volvemos a la página inicial podemos hacer login con la nueva cuenta para acceder a la interfaz disponible para empleados que se muestra en la figura 14.

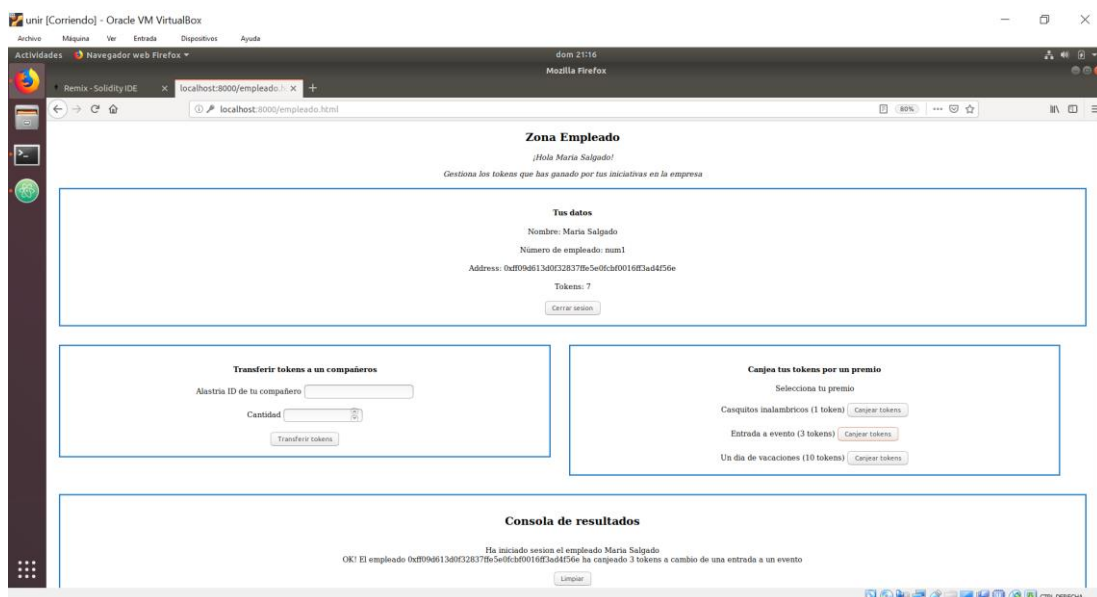
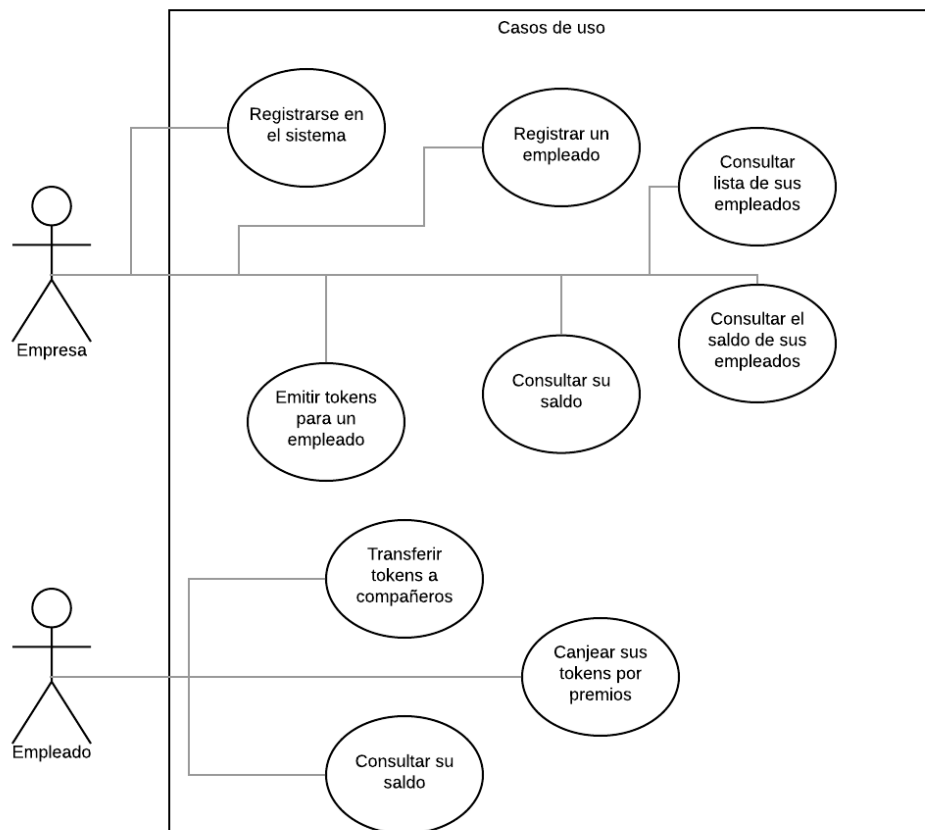


Ilustración 14. Interfaz para los empleados

En una fase posterior de este proyecto se podría desarrollar un panel para que la cuenta administradora que pudiera configurar con qué periodicidad se emiten más

tokens a las empresas o si, por alguna circunstancia, se quieren emitir manualmente.

Los casos de uso que cubre este proyecto se recogen en el diagrama de la ilustración 15.



*Ilustración 15. Diagrama de casos de uso*

A continuación, se muestran dos diagramas de secuencia de los casos que se han considerado más representativos. El primero, en la ilustración 16, es el caso de registrar una empresa en el sistema. El segundo, en la ilustración 17, es el caso de transferir tokens entre empleados. No se han recogido las llamadas a otros contratos.

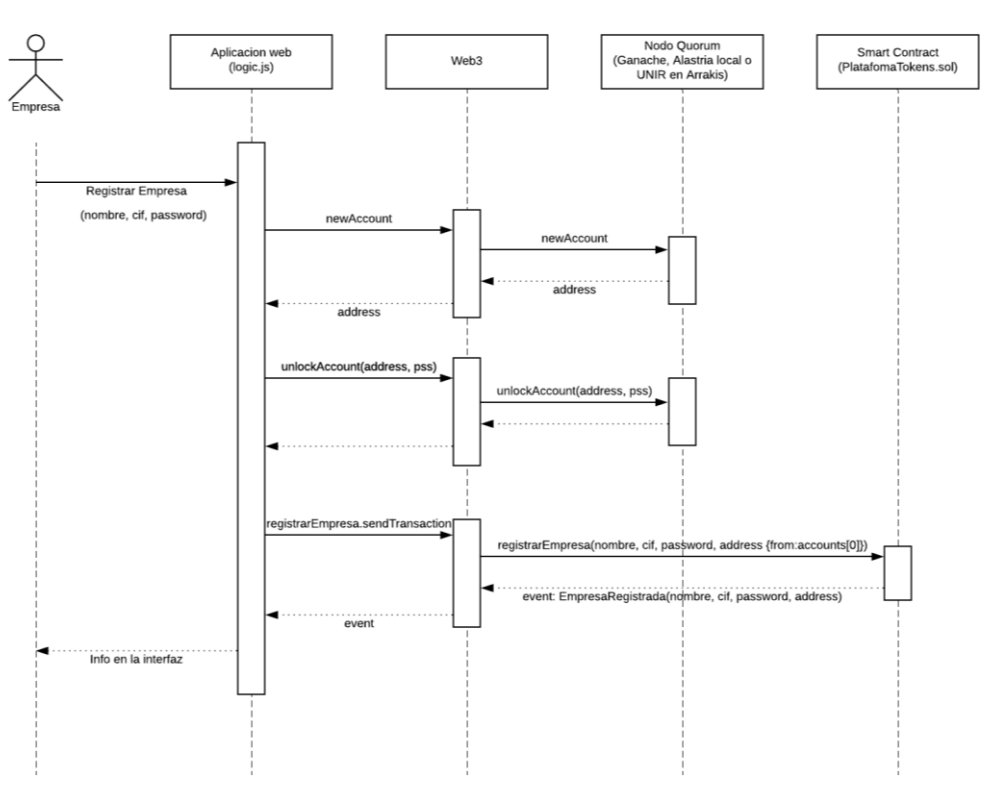


Ilustración 16. Diagrama de secuencia de registrar una empresa en el sistema.

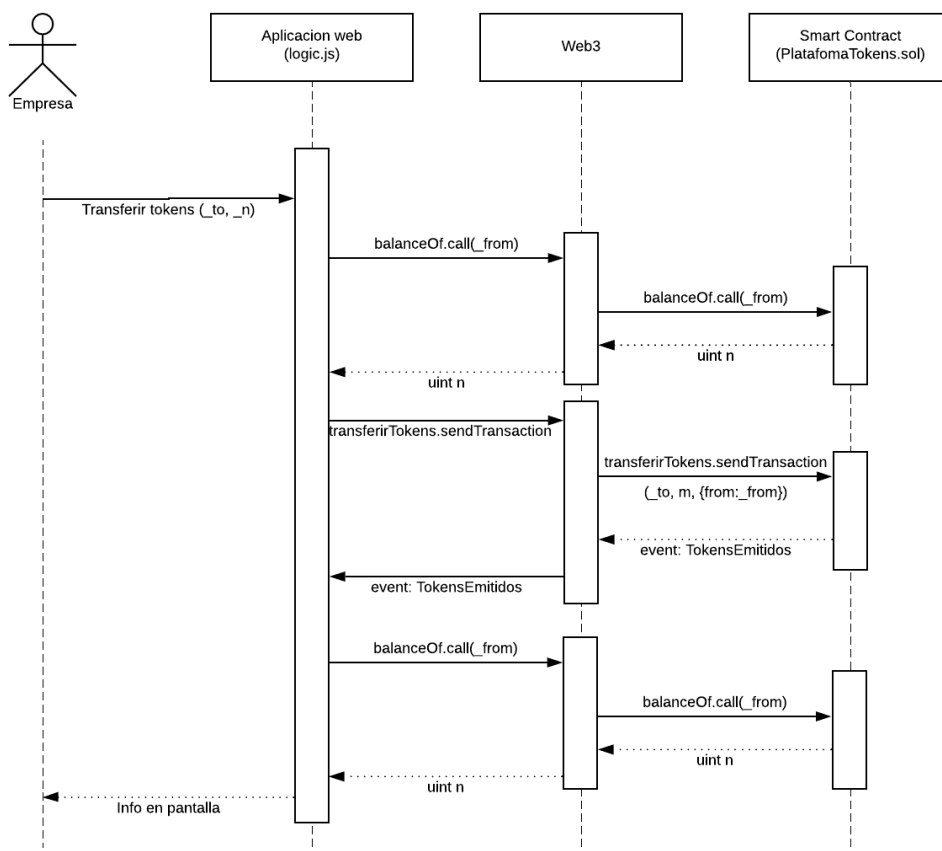


Ilustración 17. Diagrama de secuencia de transferir tokens entre compañeros



## 4. Análisis y modelo del sistema propuesto

En el diseño de la arquitectura de esta plataforma de tokens corporativos se ha querido mantener separada la vista, la lógica y el modelo. No estamos hablando de una arquitectura Modelo-Vista-Controlador tradicional, pero si se está buscando algo similar tratando todo lo que se almacena en la blockchain (incluidos los smart contracts) como modelo. Además, semanticamente también se distinguen 3 modulos: genérico, empresas y empleados. La ilustración 18 recoge el diagrama de esta arquitectuara.

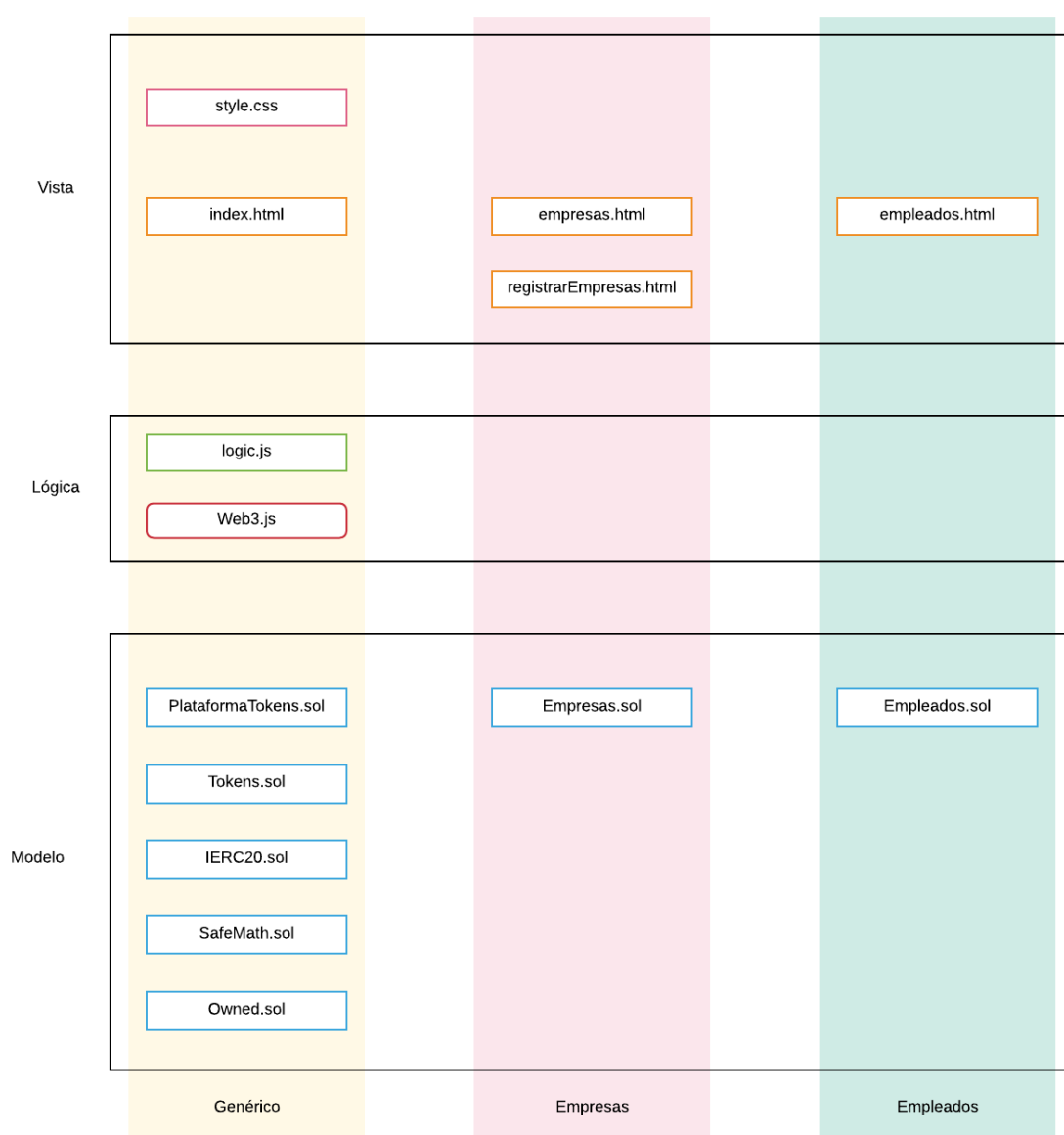


Ilustración 18. Arquitectura de la solución

## Arquitectura frontend

La interfaz web, que está desarrollada en HTML, CSS y JavaScript se comunica con la blockchain haciendo uso de la librería Web3.js en su versión 0.2.

Se muestra a continuación, en la ilustración 19, el diagrama de clases del frontend. Como se puede observar toda la lógica se encuentra en el mismo script “logic.js”. Esta decisión se ha tomado para poder reutilizar variables, para que no sea necesario instanciar más de una vez el contrato y para reducir la posibilidad de errores debido a las continuas modificaciones que es necesario hacer para probar (el ABI y la dirección de despliegue del contrato).

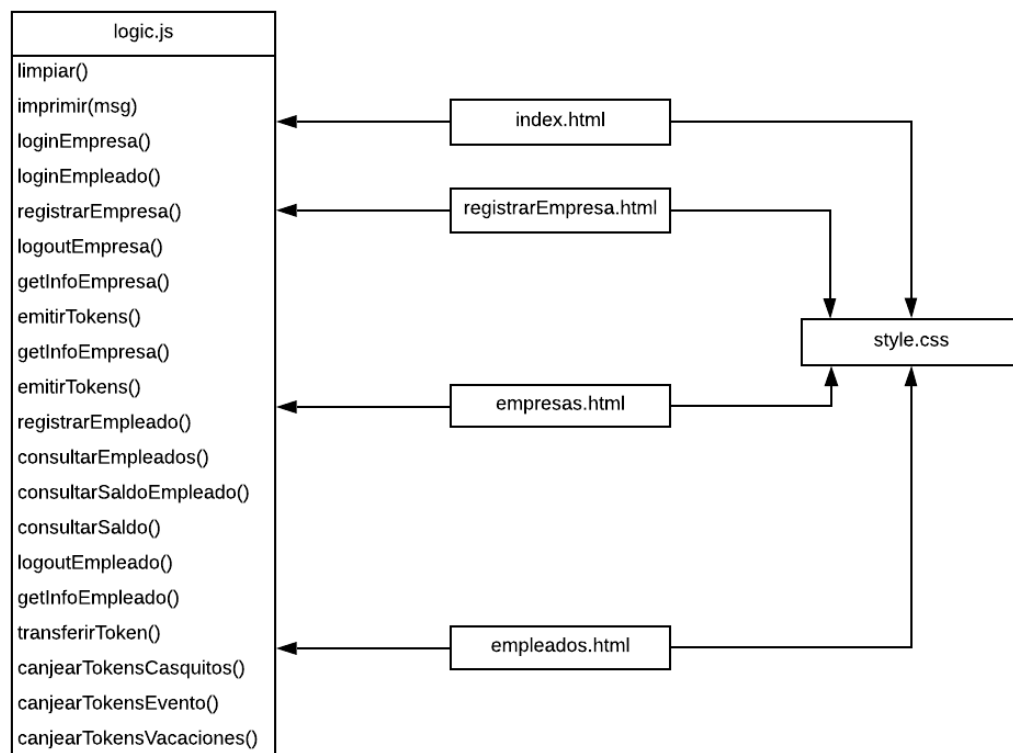


Ilustración 19. Diagrama de clases frontend

Para mayor seguridad y eficiencia se ha decidido realizar algunas comprobaciones desde el frontal de la aplicación para ahorrar llamadas innecesarias a la blockchain. Están relacionadas con que la cantidad de tokens que se intenta transferir sea inferior al saldo de la cuenta. Aparte de esto, lo único que realizan las funciones del fichero “logic.js” es llamadas al smart contract desplegado en la blockchain. Cuando

es necesario recoge los resultados o los eventos emitidos por esas funciones para mostrarlo en la vista.

La gestión de las cuentas se contará en mayor detalle en el capítulo 5 porque es ligeramente distinto dependiendo de la forma de despliegue.

## Arquitectura backend

Comenzaremos mostrando en la ilustración 20 el diagrama de clases.

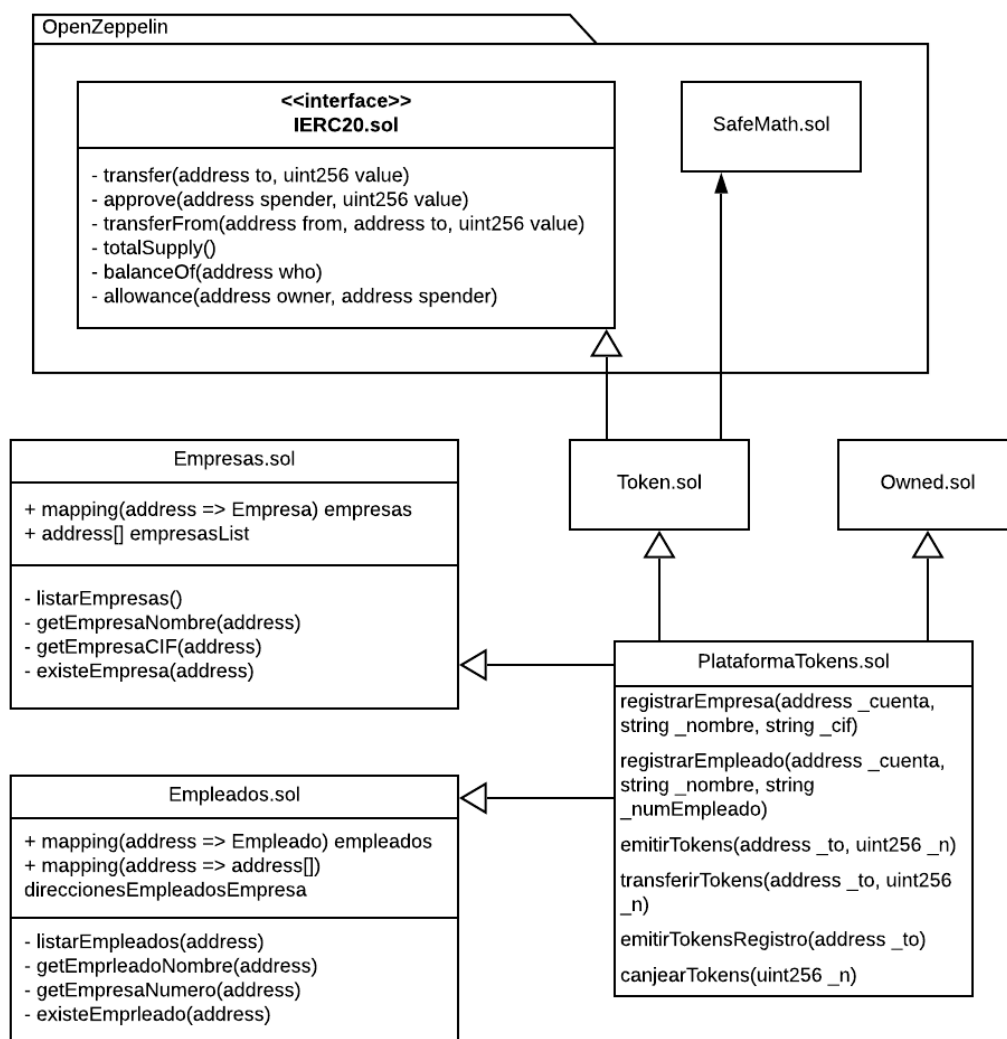


Ilustración 20. Diagrama de clases blockchain

Como se muestra, se ha utilizado la herencia entre contratos para que con solo desplegar “PlataformaTokens.sol” se pueda invocar a todas las funciones.

Como se ve en el diagrama, he utilizado dos contratos de OpenZeppelin:

- SafeMath.sol<sup>9</sup> para utilizar uint256 de forma segura evitando desbordamientos.
- IERC20.sol<sup>10</sup> que es la interfaz del estándar ERC20, el primero que se utilizó dentro de la plataforma Ethereum. Tiene ya solucionados múltiples problemas de seguridad y permite ejecutar la mayoría de las transacciones realizando llamadas a transfer().

Otros dos contratos son una versión minimamente adaptada para esta solución ya que me he basado en implementaciones muy parecidas:

- Owned.sol que permite aplicar el patrón *ownable* porque se asegura de que exista una cuenta propietaria del smart contract y mediante el modificador *onlyOwner* haya determinadas funciones restringidas para otras cuentas.
- Tokens.sol que es la adaptación del ERC20.sol de OpenZeppelin<sup>11</sup> pero cambiando el nombre del token a GLUPIs y su símbolo a GPI.

El resto de contratos son implementación propia de esta solución:

- Empresas.sol que define el struct empresa y las siguientes estructuras de datos:
  - mapping(address => Empresa) empresas;  
Para poder acceder rápidamente a la información de una empresa a partir de su dirección.
  - address[] empresasList;  
Para poder iterar con todas las direcciones de las empresas existentes.Esta estructura no se utiliza actualmente desde la plataforma web. Se ha usado para realizar testing invocando a esta función desde Remix. Se ha decidido mantenerla para casos de uso futuros.

---

<sup>9</sup> <https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol>

<sup>10</sup> <https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/IERC20.sol>

<sup>11</sup> <https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/ERC20.sol>

- Empleados.sol que define el struct empleado y las siguientes estructuras de datos:
  - mapping(address => Empleado) empleados;

Para poder acceder rapidamente a la información de un empleado a partir de su dirección independientemente de la empresa a la que pertenezca.

mapping(address => address[]) direccionesEmpleadosEmpresa;

Para poder listar a todos los empleados de una empresa, cada empresa debe mantener una lista con las direcciones de sus empleados.
- PlataformaTokens.sol que hereda los métodos de todas las anteriores e implementa algunas nuevas funciones en las que hay que utilizar datos de empresas y empleados.

La visibilidad de todas las funciones es pública para que puedan llamarse desde otros contratos e internamente. Además, las que no modifican la cadena se han declarado de tipo view.

Como se ha indicado previamente, esta solución incluye la seguridad por diseño por lo que dentro de los smart contracts se han declarado unos modificadores específicos para garantizar la privacidad de los empleados. A continuación, se justifica cada uno de ellos.

- Si la cuenta desde la que se invoca la función no es una empresa registrada en el sistema, no podrá llamar a ciertas funciones.

```

/*
 * Para comprobar que el msg.sender es una empresa existente en el sistema
 */
modifier esEmpresaValida(address _cuenta){
    if(empresas[_cuenta].isValue){
        _;
    }
}

```

- Las empresas solo pueden obtener información de sus propios empleados.

```

/*
 * Comprueba que el empleado actualmente este trabajando en la empresa que llama a la funcion (msg.sender)
 */
modifier empleadoTrabajaEnEstaEmpresa (address _empleado){
    if(empleados[_empleado].empresa == msg.sender){
        _;
    }
}

```

- Hay otras funciones que se pueden invocar desde la cuenta del empleado o desde la de la empresa en la que trabaja.

```

/*
 * Comprueba que el empleado es quien quiere acceder a su propia informacion o es su empresa
 */
modifier tienePermisosPrivacidad (address _empleado){
    if(_empleado == msg.sender || empleados[_empleado].empresa == msg.sender){
        _;
    }
}

```

- En esta versión, solo se pueden realizar transferencias de tokens entre compañeros de la misma empresa por lo que se ha desarrollado otro modifier que comprueba si dos empleados pertenecen a la misma empresa.

```

/*
 * Comprueba que un empleado actualmente este trabajando en la misma empresa que llama a la funcion (msg.sender)
 */
modifier esCompanero (address _to){
    if(empleados[_to].empresa == empleados[msg.sender].empresa){
        _;
    }
}

```

## 5. Despliegue

El despliegue de la solución se ha realizado utilizando siempre unos pasos comunes para todos los entornos que son los siguientes:

1. Se descarga el directorio desde el repositorio en GitHub en el que se ha publicado<sup>12</sup> con el comando.

```
git clone https://github.com/msalitu/Alastria-TokensCorporativos-TFE.git
```

2. Se conecta Remix con la carpeta local contracts/ utilizando el comando:

```
taskset -c 0 remixd -s /home/maria/Alastria-TokensCorporativos-TFE/contracts --remix-ide http://remix.ethereum.org
```

En caso de no tenerlo instalado se puede obtener con el comando:

```
sudo npm install -g remixd
```

3. Se inicia la red blockchain. Excepto en el caso de Alastria que nos conectamos a una red ya en marcha.
4. Se desbloquean la primera cuenta, que será la que despliega el smart contract. Excepto en el caso de Ganache que ya están desbloqueadas. Utilizando el comando:

```
personal.unlockAccount(address, password)
```

5. Desde Remix se compila el contrato con la versión 0.4.25 (0.4.25+commit.59dbf8f1) y se pega el ABI en logic.js. en la variable ABI.

```
var ABI = [];
```

6. Desde Remix, se selecciona el Environment “Web3 Provider” para conectarnos a la blockchain pertinente. Utilizando la primera cuenta se despliega el contrato y se copia la dirección del mismo en logic.js en la instanciación del contrato.

```
var plataforma =  
web3.eth.contract(ABI).at("0x4248ddc4e6cc68ceddc9a8b81622a6bfb8571dd7");
```

7. Desde el directorio web/ se lanza el servidor utilizando el comando:

```
python -m SimpleHTTPServer 8080
```

O, dependiendo de la versión de python instalada, también se puede usar el siguiente comando que lo levanta en el puerto 8000:

```
python -m http.server
```

---

<sup>12</sup> <https://github.com/msalitu/Alastria-TokensCorporativos-TFE>

8. Accediendo en un navegador a localhost:<puerto> se puede ver la interfaz e interactuar con la aplicación web.

Esta solución se ha desplegado en tres entornos distintos: Ganache, una testnet local de Alastria y, finalmente, en la red Arrakis de Alastria a través del nodo de la UNIR. Lo que se ha realizado en esta práctica ha sido, gracias a Oracle VirtualBox, crear una máquina virtual para disponer de un entorno independiente en cada despliegue. A continuación, se recoge el detalle de los pasos realizados en cada uno.

## Fase 1. Ganache

Tras haber programado por separado la interfaz y los smart contracts, el primer entorno de despliegue para probar se realizó utilizando Ganache, que se ha explicado en el capítulo 2 de este documento.

Realizados los dos primeros pasos para poder trabajar desde esta máquina, llega el momento de iniciar la blockchain. Para iniciar Ganache se utiliza el comando `ganache-cli` que muestra 10 cuentas disponibles que ya están directamente desbloqueadas e indica que la blockchain está escuchando en localhost a través del puerto 8545.



```

maria@galastris:~$ ganache-cli
Ganache CLI v6.4.1 (ganache-core: 2.5.3)

Available Accounts
=====
(0) 0x972ab89a16eaa1ba7331867652972d0ae02e9d (~100 ETH)
(1) 0x12a842b89d3f15b4e77886651eef82fa1f462a20 (~100 ETH)
(2) 0x46cf1ef4012667c2ef9f31bf4be596dc4bfc2022 (~100 ETH)
(3) 0x443afab66da0d53142d391b3f28ccbde41c822ec (~100 ETH)
(4) 0x3982ed74c2c807adccff0f040df9444b6725844b (~100 ETH)
(5) 0x2f4d58949c8fd0dd2b237d753f37901fea6c664b (~100 ETH)
(6) 0xfe9c0319872a2c4998d251844368cdc243a2b65 (~100 ETH)
(7) 0x66ed84b91c1ceb0a66697efd16b8758e25c0504f (~100 ETH)
(8) 0xc33df8b39724769bd3de3594aa24f10df7eb74f3 (~100 ETH)
(9) 0x84b5f7260b006e3cf1826905c0c081560aa513b1 (~100 ETH)

Private Keys
=====
(0) 0xb0ccf351fef6e3236d83d6dd90516e104023dd58e1876e8f8b9b694bdce28261
(1) 0xac3e8e2a7ab147b14b31485626f4749bac1a50d72013e4dd617ec40331d9bffe
(2) 0x249d37370627795c650514cf4f13af2e704d1aa4eba9bffe408d2387ab01d0c9
(3) 0x027b7bab33be2ebecadfb95e803bb0ef5ddc9be283a53848a73ee90135786ba5
(4) 0x3be39ccc9e8f8f35ebc6e46eae1c5384e11632c3f17114747d25b889c3efc19
(5) 0xe2c2683e41654b94712d6f408878eb4bc66a6ed5443068c39852cd32f65e1cb9
(6) 0xd2c12c1f5ffa0b969859d393e6ecda4a3c8af3738f973e7a5fe190eed93e85fb
(7) 0x55342a480b463ca6d21f14bd40db5920b0af17b6e7d02ac3365a3a5c519a8635
(8) 0x8d7a9b8305e46aa67de334f3571fa34789656d2fd59a6ce7097e59af694d0b99
(9) 0xa9782f0121ed0a48168a61c36885a4f96a31118d9acce7e0acfdc82c29e76d46

HD Wallet
=====
Mnemonic:      style method alarm suggest mutual magic powder decrease wreck color room jump
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
=====
20000000000

Gas Limit
=====
6721975

Listening on 127.0.0.1:8545

```

Ilustración 21. Iniciar Ganache

Para poder compilar el contrato desde Remix nos conectamos a nuestra máquina en ese puerto tal y como se muestra en la ilustración 22.

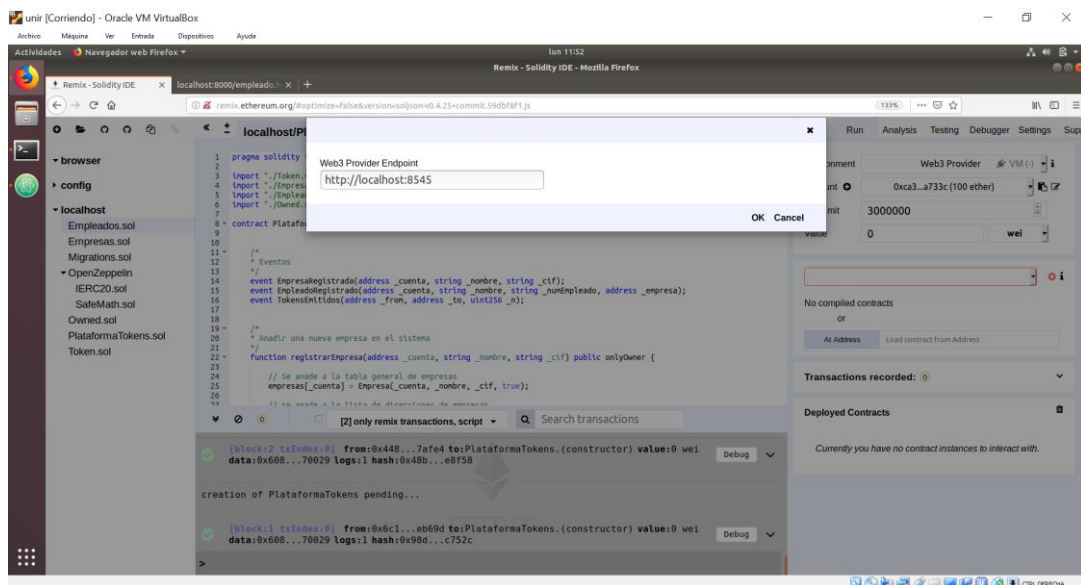


Ilustración 22. Conectar Remix al nodo de Ganache

A continuación, se compila el contrato PlataformaTokens.sol y se copia su ABI en el fichero logic.js. Después se despliega el contrato utilizando la primera de las cuentas. En Remix se puede ver que el contrato se ha desplegado correctamente (ilustración 23).

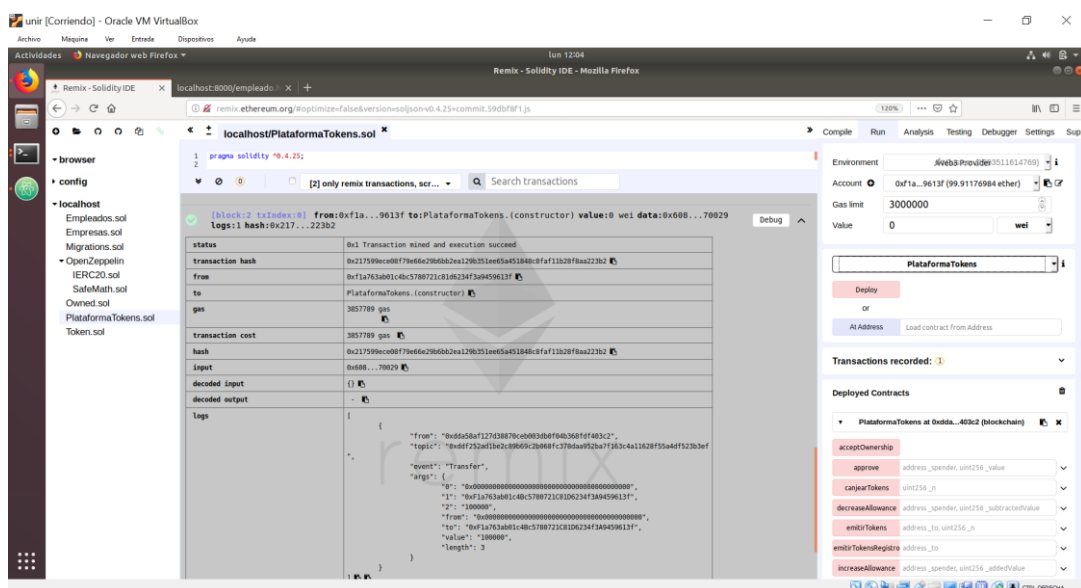


Ilustración 23. Desplegar contrato en Remix conectado a Ganache

Se copia la dirección en la que se ha desplegado el contrato y se pega en logic.js. A partir de este momento, tras lanzar el servidor HTTP, ya es posible interactuar con la interfaz web.

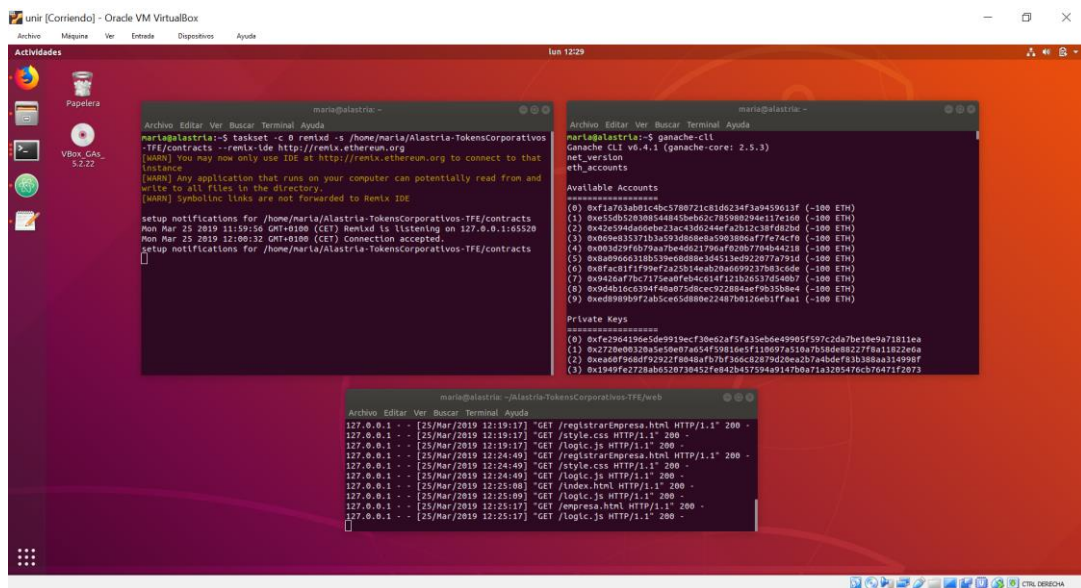


Ilustración 24. Despliegue de la aplicación web con Ganache

Para comprobar que todo está funcionando desde el principio, si se muestra la consola de JavaScript del navegador, se deberá ver el array de cuentas disponibles y el nombre de los tokens GLUPI tal y como se muestra en la ilustración 25.

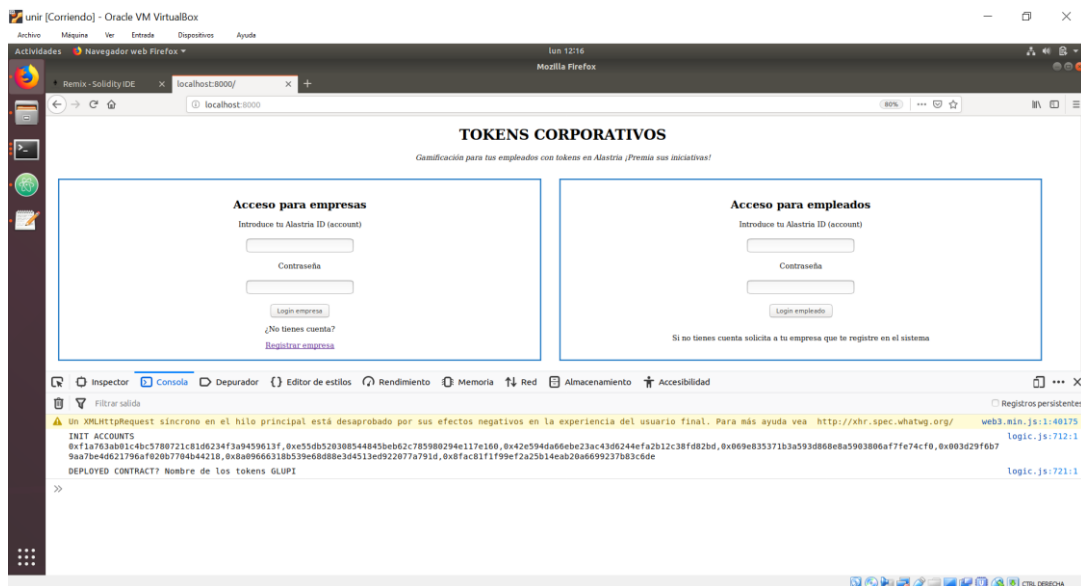


Ilustración 25. Interfaz web conectada al smart contract desplegado en Ganache

Los resultados de las diferentes acciones que se realizan desde la plataforma se van imprimiendo en la zona “Consola de resultados”. Lo que se hace es recibir el evento que se emite tras ejecutar la función de un smart contract y formar un mensaje que

se imprime por pantalla. Para ello se ha utilizado la sintaxis que muestra la ilustración 26 en las funciones.

```
// Registrar un nuevo empleado
function registrarEmpleado(){
  var nombre = document.getElementById("nombreEmpleado").value;
  var pss = document.getElementById("pssEmpleado").value;
  var nEmpleado = document.getElementById("numEmpleado").value;
  var cuentaEmpresa = localStorage.getItem("accountEmpresa");

  // En la testnet local de Alastria
  //var address = web3.personal.newAccount(pss);
  // En ganache y el nodo de la UNIR debe ponerse un indice para una cuenta disponible
  var address = accounts[2];

  web3.personal.unlockAccount(address, pss, 0);

  console.log("Cuenta empresa " + cuentaEmpresa);
  plataforma.registrarEmpleado.sendTransaction(address, nombre, nEmpleado, {from: cuentaEmpresa, gas:200000},
    function (error,result){
      if (!error){
        var event = plataforma.EmpleadoRegistrado({}, {fromBlock: 'latest', toBlock: 'latest'},
          function(error, result){
            if (!error){
              var msg = "OK! Se ha creado correctamente la cuenta " + result.args._cuenta + " para " +
                result.args._nombre + " con numero de empleado " + result.args._numEmpleado + " con contraseña " + pss;
              imprimir(msg);
            }else{
              console.log("Error" + error);
            }
          });
      } else {
        console.error("Error" + error);
      }
    });
  web3.personal.lockAccount(address, pss);
}
```

*Ilustración 26. Código ejemplo de captura de eventos por la interfaz*

Se ha detectado el problema de que la interfaz recibe todos los eventos emitidos por lo que se añadió el filtro para recoger el último bloque. Aún así, al realizar llamadas consecutivas a la misma función se muestran los mensajes de los eventos anteriores en la consola de resultados. En la ilustración 27 por ejemplo se puede observar que cuando se envía 1 token al empleado y luego otros 20, su saldo final, por tanto es 21. El mensaje de emisión de 20 tokens aparece repetido dando lugar a confusión. Este bug de la plataforma habría que arreglarlo cacheando bloques pero no se ha considerado dentro del alcance de este TFE.

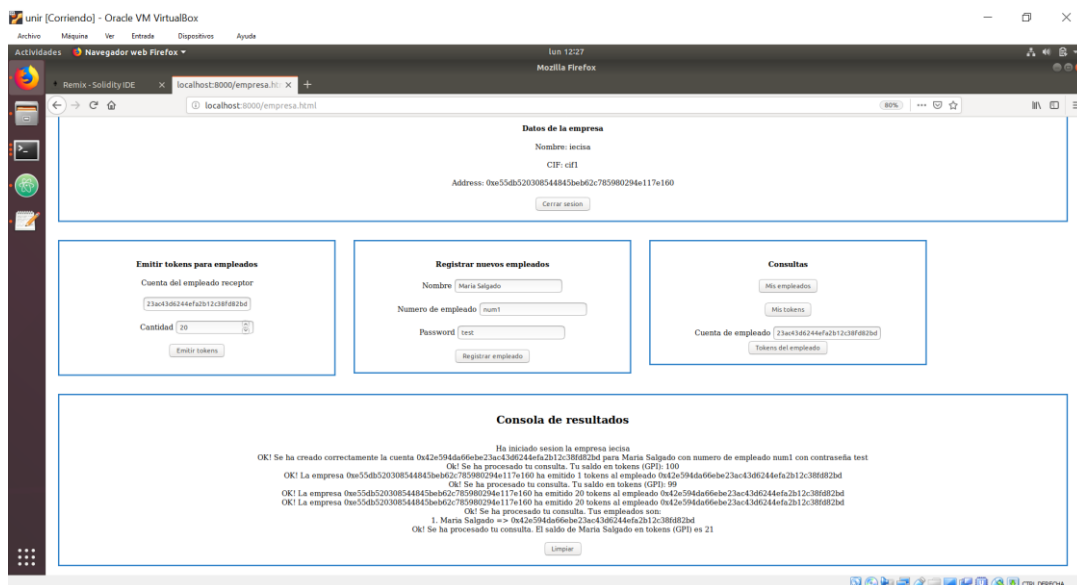


Ilustración 27. Plataforma web funcionando con Ganache

## Fase 2. Testnet Alastria local

Lo primero es crear una nueva máquina virtual a partir de una imagen de Ubuntu 18.04.1 nueva. A continuación se han seguido los pasos del manual “Despliegue de una red de pruebas local de Alastria” que se nos facilitó a los alumnos del curso experto. Para que este documento contenga toda la información, se reproducen a continuación:

- Descargar el paquete Libsodium18 desde [http://launchpadlibrarian.net/330507614/libsodium18\\_1.0.13-1\\_amd64.deb](http://launchpadlibrarian.net/330507614/libsodium18_1.0.13-1_amd64.deb) e instalarlo con el gestor de paquetes.
- Ir a la carpeta /home y clonar el repositorio:  
`git clone https://github.com/alastria/test-environment.git`

Si no se tiene, es necesario descargar previamente git con:

```
sudo apt install git
```

- Ejecutar el script bootstrap.sh como usuario root. Esto instalará todas las dependencias necesarias:

```
cd test-environment/infrastructure/testnet/
sudo ./bin/bootstrap.sh
```

A continuacion, es necesario acceder al script `start_node.sh` y en la variable `GLOBAL_ARGS` hay que añadir el atributo `rpccorsdomain` con las urls a las que nos vamos a conectar:

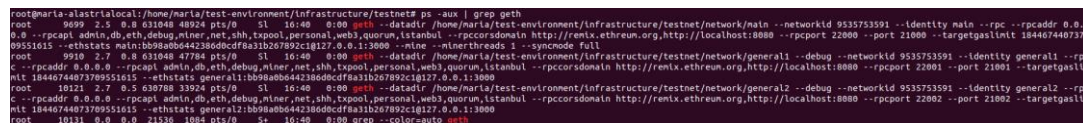
```
--rpccorsdomain http://remix.ethereum.org,http://localhost:8080
```

Ahora ya podemos iniciar la red. Para iniciar la red con un nodo validador y dos nodos regulares se debe usar el script `start_network.sh` de la siguiente manera:

```
./bin/start_network.sh clean 1 2
```

Una vez iniciada la red se puede comprobar qué procesos `geth` se han inicializado en el sistema con el comando:

```
ps -aux | grep geth
```



*Ilustración 28. Información de procesos geth*

Entonces, tal y como muestra la ilustración 28, se verá el nodo `general1` que será al que nos conectaremos. Comprobamos que aparecen los parámetros que acabamos de configurar en `rpccorsdomain`. En él se puede ver el puerto en el que escucha por `rpc`, en este caso el 22001.

Una vez desplegada la red, podemos descargar nuestro proyecto con un:

```
git clone https://github.com/msalitu/Alastria-TokensCorporativos-TFE.git
```

Y conectar Remix a la carpeta local con:

```
taskset -c 0 remixd -s /home/maria/Alastria-TokensCorporativos-TFE/contracts --  
remix-ide http://remix.ethereum.org
```

A continuación, podemos conectarnos a Remix de la misma forma que hacíamos con Ganache pero modificando el puerto (Ilustración 29).

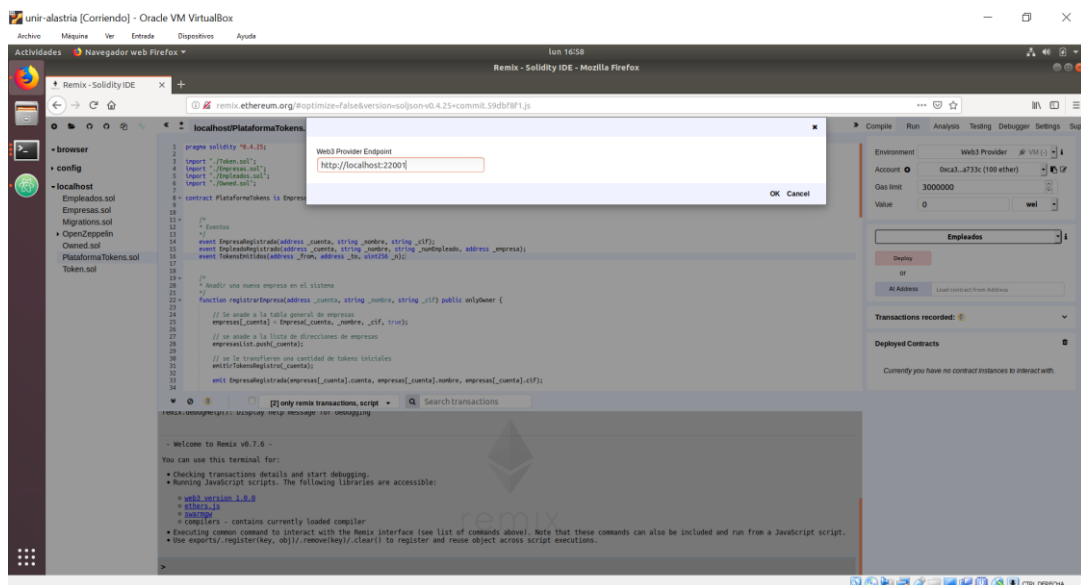


Ilustración 29. Conexión de Remix con la red local de pruebas de Alastria

Desde Remix, se compila el contrato (con la versión 4.25 del compilador) y se actualiza en el fichero logic.js:

- el ABI
- el puerto de la conexión (22001)

Finalmente, queda desplegar el contrato pero previamente debemos desbloquear la cuenta. Para ello nos conectamos al nodo general1 con el comando:

```
geth attach ./network/general1/geth.ipc
```

Y una vez en la consola de geth desbloqueamos la cuenta:

```
personal.unlockAccount(eth.accounts[0], "Passw0rd", 0)
```

Una vez que está el smart contract desplegado, copiamos la dirección en logic.js. De esta forma quedan actualizados todos los datos para que la aplicación web pueda interactuar con la blockchain.



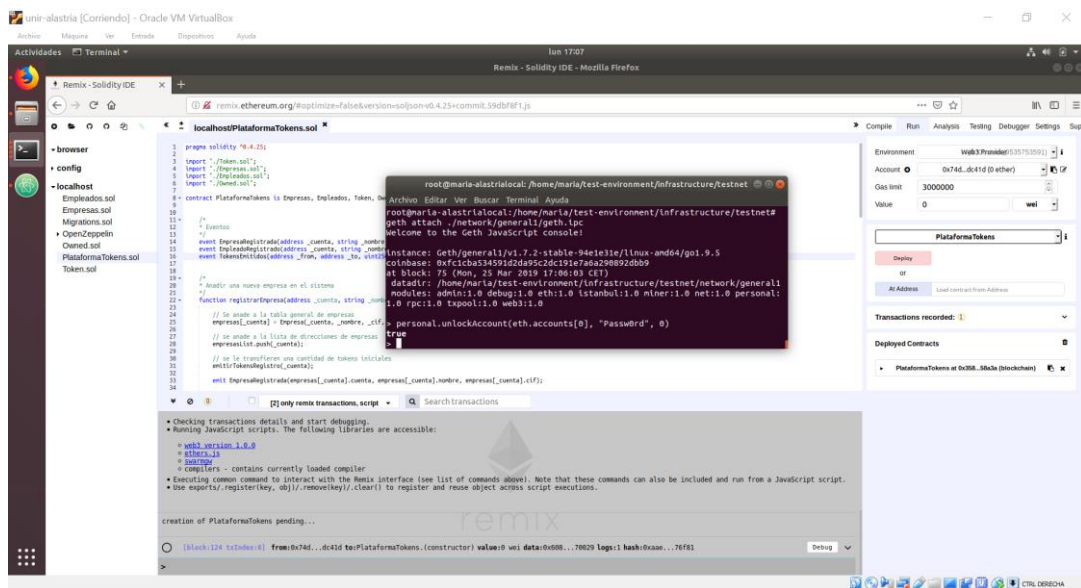


Ilustración 30. Desbloqueo de cuenta y despliegue de smart contract

Si se lanza el servidor y se accede a la interfaz web mostrando la consola de JavaScript del navegador podemos observar que ya está interactuando con el smart contract correctamente.

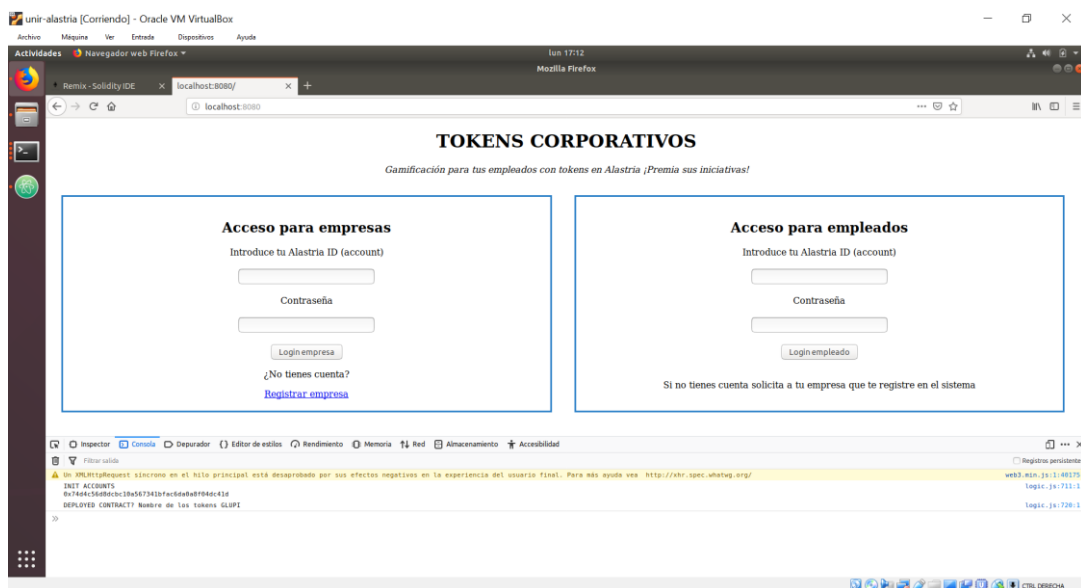


Ilustración 31. Interfaz web conectado al smart contract en la testnet local de Alastria

Las cuentas, dentro del fichero logic.js se crean con:

```
web3.personal.newAccount(pss);
```

Por este motivo, se pueden realizar todas las pruebas deseadas creando empresas y empleados.



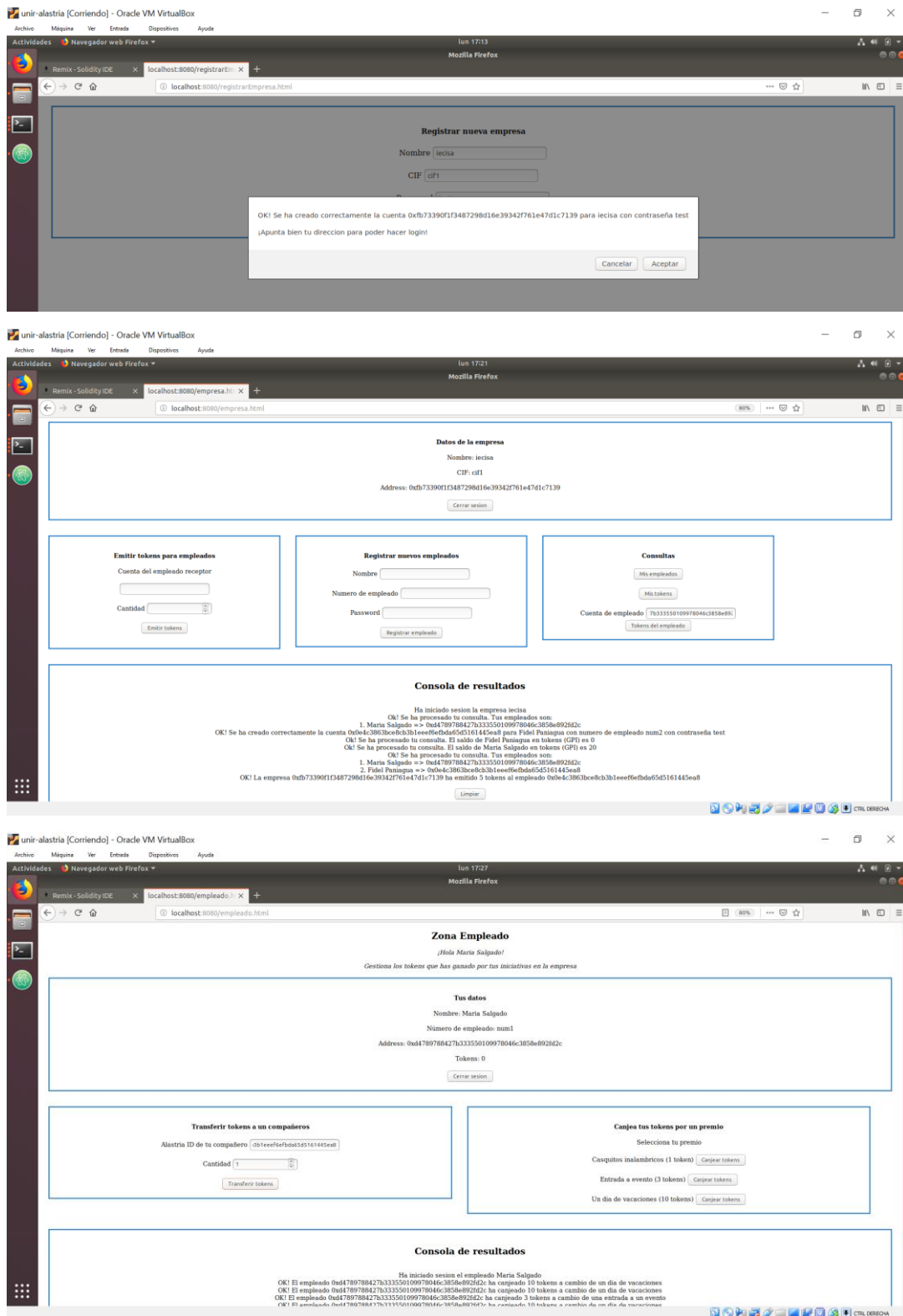


Ilustración 32. Pantallas de la aplicación web funcionando con la red de pruebas local de Alastria

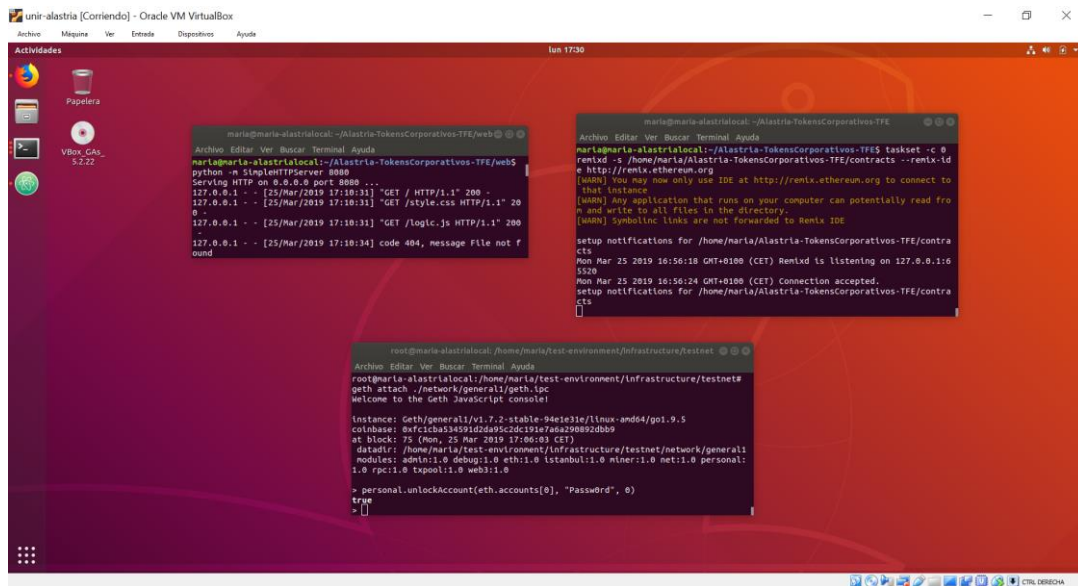


Ilustración 33. Despliegue de la aplicación web en la testnet local de Alastria

### Fase 3. Alastria nodo UNIR

Para desplegar en Alastria se nos ha facilitado la dirección y el puerto de escucha del nodo de UNIR en Arrakis, que es la primera versión de red de pruebas de Alastria. Actualmente, como se ha explicado en el capítulo 2, se está migrando a Telsius. La dirección es <http://138.4.143.82:8545>.

También se nos han proporcionado una serie de cuentas y su contraseña "Alumnos\_2018\_Q4\_IKx5srvT":

CUENTAS\_ALUMNO = [

"0x994319e1b1de09aac4aa5b225a7d5cade79d04ed",  
 "0x66c5a820d0e743fc7030f02aa873875c84a88f3f",  
 "0x34322a678b16ce26fc0e2bdde1e3c1b666a34a66",  
 "0xfc3b00c03b74ee1d94fa10e21aef4e6e9710e8a8",  
 "0xf76c62480a8a6a83451eeef40d331ed179da7f89",  
 "0x7b1b6d29cb425887d1bc4849d0708091bcbaf16b",  
 "0x12e3bb9f253bd233e03bd696b1c558a056179b87",  
 "0x59bedaa81edfd70b8e370a96cf29ee327e84e551",  
 "0x9a63729158a93f502935bc322af78e4f25a5cc02",

```
"0xab2c680816421e56ba3274a37c3df455fba32725"  
];
```

Tras realizar los dos primeros pasos comunes de descargar el proyecto y configurar Remix para que acceda a los smart contracts desde localhost, conectamos Remix a la dirección del nodo de la UNIR.

Web3 Provider Endpoint

<http://138.4.143.82:8545>

Lo primero que hay que hacer para poder desbloquear la cuenta con la que queremos desplegar el contrato es utilizar el comando:

```
geth attach http://138.4.143.82:8545
```

Y una vez en la consola de geth el comando:

```
personal.unlockAccount("0x994319e1b1de09aac4aa5b225a7d5cade79d04ed",  
"Alumnos_2018_Q4_IKx5srvT")
```

```
maria@maria-AlastriaTestnet:~/Alastria-TokensCorporativos-TFE$ geth attach http://138.4.143.82:8545  
Welcome to the Geth JavaScript console!  
  
instance: Geth/REG_UNIR0_2_4_0/v1.7.2-stable-94e1e31e/linux-amd64/go1.9.5  
coinbase: 0xab6db028cc07d0ce4282de5c59a7a63f474bcd81  
at block: 23608394 (Sun, 24 Mar 2019 16:37:14 CET)  
modules: admin:1.0 debug:1.0 eth:1.0 istanbul:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0  
> personal.unlockAccount("0x994319e1b1de09aac4aa5b225a7d5cade79d04ed", "Alumnos_2018_Q4_IKx5srvT")  
true
```

*Ilustración 34. Desbloqueo de la cuenta para interactuar con el nodo de UNIR en Arrakis*

El contrato se despliega con la primera de las cuentas que nos han pasado como disponibles, que es la que acabamos de desbloquear  
0x994319e1b1de09aac4aa5b225a7d5cade79d04ed

Environment: Web3 Provider (2584648528)

Account: 0x994...d04ed (0 ether)

Gas limit: 3000000

Value: 0 wei

PlataformaTokens

Deploy

or

At Address: Load contract from Address

Transactions recorded: 1

Deployed Contracts

PlataformaTokens at 0x424...71dd7 (blockchain)

Ilustración 35. Despliegue del smart contract en Arrakis con la cuenta desbloqueada

El contrato se ha desplegado en la dirección  
0x4248ddc4e6cc68ceddc9a8b81622a6bfb8571dd7

Utilizando ese hash se puede ver en explorador de Alastria <sup>13</sup>

También se puede ver la transacción de despliegue del contrato<sup>14</sup> cuyo hash es 0xacf3eb53d4c48eebb33920272a2549a8188689e3020a5b06a0fc4ec004af1fd2 y se puede comprobar que, efectivamente, se desplegó desde la cuenta que hemos indicado anteriormente si se consulta el campo from.

<sup>13</sup> <https://alastria-explorer.councilbox.com/account/0x4248DC4e6cC68CeDdC9a8b81622A6BFb8571DD7>

<sup>14</sup> <https://alastria-explorer.councilbox.com/transaction/0xacf3eb53d4c48eebb33920272a2549a8188689e3020a5b06a0fc4ec004af1fd2>

Aún así, desde Remix, nos encontramos un error. La creación del contrato no se termina de realizar correctamente. En la siguiente ilustración aparece que tiene estado “Status is not available at the moment”. Tras realizar las consultas pertinentes y diversos intentos, esto se debe a que Arrakis no soporta las llamadas entre contratos. PlataformaTokens.sol hereda de otros cuatro contratos por lo que este error es propio de la red, en Telsius ya está corregido.

The screenshot shows the Remix IDE interface. The console on the left displays the following messages:

```

creation of PlataformaTokens pending...
creation of PlataformaTokens error: authentication needed: password or unlock
creation of PlataformaTokens pending...

```

The transaction details table in the console shows the following information:

Field	Value
status	Status not available at the moment
transaction hash	0xacf3eb53d4c48ebbb33920272a2549a8188689e3029a5b06a0fc4ec004af1fd2
from	0x994319e1b1de09aac4aa5b25a7d5cade79d04ed
to	PlataformaTokens (constructor)
gas	3000000 gas
transaction cost	3000000 gas
hash	0xacf3eb53d4c48ebbb33920272a2549a8188689e3029a5b06a0fc4ec004af1fd2
input	0x608...70029
decoded input	{}
decoded output	-
logs	[ ]
value	0 wei

Ilustración 36. Información del despliegue del contrato. Error en status.

Aunque se intente invocar a alguna de las funciones de consulta que no modifican la cadena de bloques, no es posible. En la siguiente ilustración se muestra lo que ocurre cuando se invoca name(), que debería devolver GLUPI, el nombre del token.

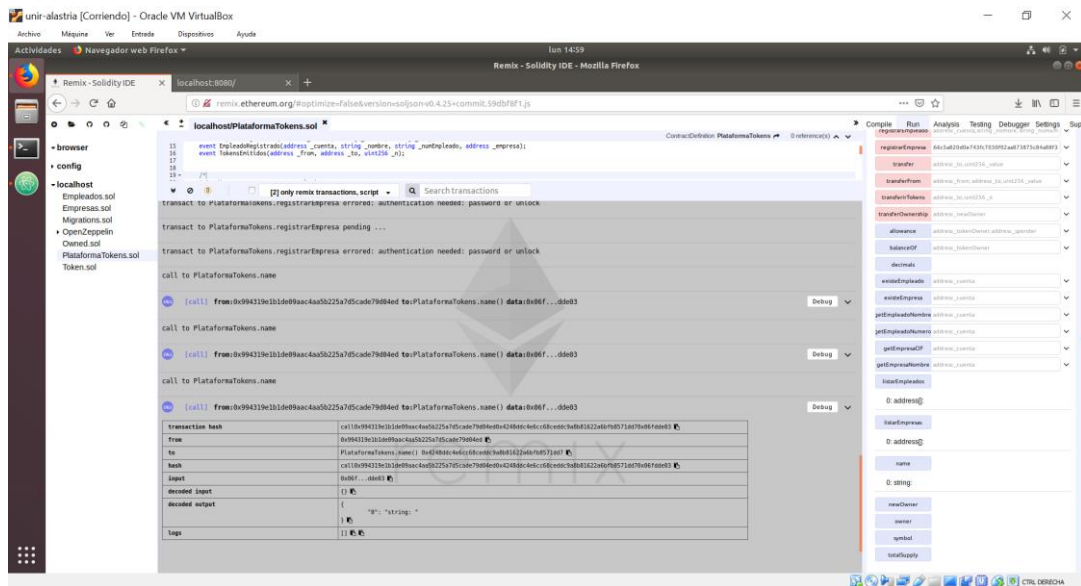


Ilustración 37. Información de la llamada a una función de consulta. Error.

Por este motivo, no ha sido posible probar el funcionamiento de la interfaz web en Arrakis.

## 6. Pruebas

Para cumplir con las buenas prácticas aprendidas en el curso y realizar testing e integración continúa, se han preparado unas pruebas unitarias para comprobar la funcionalidad y los casos extremos de la plataforma. Para ello se ha utilizado la herramienta Truffle, que se ha explicado en el capítulo 2.

Ha sido necesario instalar con npm tanto truffle como truffle-assertions.

```
sudo npm install -g truffle
```

Las pruebas unitarias se han dividido en dos conjuntos. En el primero se hacen comprobaciones a las funciones y en el segundo a los eventos.

Aunque los nombres de los tests son autoexplicativos, a continuación, se hace una breve descripción de cada uno de ellos.

0	La plataforma se ha desplegado correctamente y la cuenta administradora tiene el totalSupply de tokens
	Se comprueba que el saldo de accounts[0] sea igual a 100000.
1	Lista de empresas en el sistema vacia al principio
	Comprueba que el sistema está limpio y vacío.
2	Se impide registrar una empresa desde una cuenta no administradora
	Intenta registrar a una empresa desde una cuenta cualquiera pero no se permite. Se está testeando el modifier onlyOwner.
3	Registrar una empresa desde la cuenta administradora
	Comprueba que una empresa se da de alta en el sistema correctamente (si la llamada al smart contract viene de accounts[0]). Se está testeando el modifier onlyOwner.
4	Se impide registrar un empleado desde una cuenta que no es de una empresa registrada en el sistema
	Se hace la llamada al smart contract desde una cuenta cualquiera. Se está testeando el modifier esEmpresaValida.
5	Se impide registrar un empleado desde la cuenta de otro empleado
	Se hace la llamada al smart contract desde la cuenta de un empleado ya existentes en el sistema. Se está testeando el modifier esEmpresaValida.

6	Registrar un empleado desde una cuenta de empresa valida y consultar su info desde esa misma empresa
	Comprueba que se puede llamar a la función registrarEmpleado desde una cuenta de una empresa existente en el sistema y acceder a los datos del empleado desde la misma. Se está testeando el modifier esEmpresaValida.
7	Se impide consultar info de un empleado desde la cuenta de una empresa a la que no pertenece
	Se hace la llamada al smart contract desde la cuenta de una empresa distinta. Se está testeando el modifier tienePermisosPrivacidad.
8	Se impide consultar info de un empleado desde la cuenta de otro empleado
	Se hace la llamada al smart contract desde la cuenta de un compañero. Se está testeando el modifier tienePermisosPrivacidad.
9	Se impide que una empresa emita tokens para un empleado que no es suyo
	Se hace la llamada al smart contract desde la cuenta de una empresa distinta. Se está testeando el modifier tienePermisosPrivacidad.
10	Una empresa emite tokens para un empleado suyo
	Comprueba que la llamada a la función emitirTokens funciona correctamente. Se está testeando también el modifier tienePermisosPrivacidad y esEmpresaValida.
11	Se impide que un empleado transfiera tokens para otro empleado que no pertenece a su empresa
	Se hace la llamada al smart contract desde la cuenta de un empleado de una empresa distinta. Se está testeando el modifier esCompanero.
12	Un empleado transfiere tokens para un companero de su empresa
	Comprueba la llamada a la función transferirTokens. Se está testeando el modifier esCompanero.
13	Comprobar que existe un empleado en el sistema (TRUE)
	Comprueba la llamada a la función existeEmpleado.
14	Comprobar que no existe un empleado en el sistema (FALSE)
	Comprueba la llamada a la función existeEmpleado.
15	Comprobar que existe una empresa en el sistema (TRUE)
	Comprueba la llamada a la función existeEmpresa.
16	Comprobar que no existe una empresa en el sistema (FALSE)
	Comprueba la llamada a la función existeEmpresa.



En cuanto al conjunto de tests de eventos se han definido los siguientes:

0	Emite evento cuando se registra una empresa en el sistema
	Comprueba el evento EmpresaRegistrada.
1	Emite evento cuando se registra un empleado en el sistema
	Comprueba el evento EmpleadoRegistrado.
2	Emite evento cuando se transfieren tokens al registrar una empresa
	Comprueba el evento TokensEmitidos.
3	Emite evento cuando se emiten tokens para un empleado desde una empresa
	Comprueba el evento TokensEmitidos.
4	Emite evento cuando se transfieren tokens entre empleados
	Comprueba el evento TokensEmitidos.

La ilustración 38 muestra el fichero de configuración de truffle en el que se han planteado tres posibles redes a las que conectarse: ganache, localAlastria y Alastria.

```

module.exports = {

  networks: {

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

    localAlastria: {
      host: "127.0.0.1",      // Localhost (default: none)
      port: 22001,            // general 1
      network_id: "*",       // Any network (default: none)
      gasPrice: 0,
      gas: 4500000
    },

    alastria: {
      host: "138.4.143.82",
      port: 8545,
      network_id: "*",       // Any network (default: none)
      from: "0x994319e1b1de09aac4aa5b225a7d5cade79d04ed",
      gasPrice: 0,
      gas: 4500000
    }

  },

},

```

*Ilustración 38. Configuración de las distintas redes para desplegar los smart contracts en Truffle*

El comando para compilar y desplegar los smart contracts es:

```
truffle migrate --reset --all --network nombreDeLaRed
```

A continuación, para que se ejecuten los tests simplemente es necesario utilizar el comando:

```
truffle test
```

Se ha desplegado y se han realizado las pruebas con Ganache como muestra la ilustración 39.

```
maria@alastria: ~/Alastria-TokensCorporativos-TFE
Archivo Editar Ver Buscar Terminal Ayuda
maria@alastria:~/Alastria-TokensCorporativos-TFE$ truffle migrate --reset --all --network ganache
Compiling ./contracts/Empleados.sol...
Compiling ./contracts/Empresas.sol...
Compiling ./contracts/Migrations.sol...
Compiling ./contracts/OpenZeppelin/IERC20.sol...
Compiling ./contracts/OpenZeppelin/SafeMath.sol...
Compiling ./contracts/Owned.sol...
Compiling ./contracts/PlataformaTokens.sol...
Compiling ./contracts/Token.sol...
Writing artifacts to ./build/contracts

Using network 'ganache'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
... 0xc3fa06b09b45723b1594ca0da76dfec4f7063393c164e5121fcd324d2e76a8a2
  Migrations: 0xc96f73a8c4870263ea0590470f38f1abea29cd5d
  Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying PlataformaTokens...
... 0xb82fc150a2fb4cf98aaf3ea78a6abd72a6568ff4d97942859200d415ebb44b2d
  PlataformaTokens: 0x38febccce3d50d098c0e121545ab7e0bdade9550c
  Saving artifacts...
maria@alastria:~/Alastria-TokensCorporativos-TFE$

maria@alastria:~/Alastria-TokensCorporativos-TFE$ truffle test
Using network 'test'.

Contract: TestContratos
  ✓ Test0: La plataforma se ha desplegado correctamente y la cuenta administradora tiene el totalSupply de tokens (51ms)
  ✓ Test1: Lista de empresas en el sistema vacía al principio (51ms)
  ✓ Test2: Se impide registrar una empresa desde una cuenta no administradora
  ✓ Test3: Registrar una empresa desde la cuenta administradora (251ms)
  ✓ Test4: Se impide registrar un empleado desde una cuenta que no es de una empresa registrada en el sistema (94ms)
  ✓ Test5: Se impide registrar un empleado desde la cuenta de otro empleado (127ms)
  ✓ Test6: Registrar un empleado desde una cuenta de empresa válida y consultar su info desde esa misma empresa (151ms)
  ✓ Test7: Se impide consultar info de un empleado desde la cuenta de una empresa a la que no pertenece (97ms)
  ✓ Test8: Se impide consultar info de un empleado desde la cuenta de otro empleado (80ms)
  ✓ Test9: Se impide que una empresa emita tokens para un empleado que no es suyo (144ms)
  ✓ Test10: Una empresa emite tokens para un empleado suyo (104ms)
  ✓ Test11: Se impide que un empleado transfiera tokens para otro empleado que no pertenece a su empresa (94ms)
  ✓ Test12: Un empleado transfiere tokens para un compañero de su empresa (107ms)
  ✓ Test13: Comprobar que existe un empleado en el sistema (TRUE) (55ms)
  ✓ Test14: Comprobar que no existe un empleado en el sistema (FALSE)
  ✓ Test15: Comprobar que existe una empresa en el sistema (TRUE)
  ✓ Test16: Comprobar que no existe una empresa en el sistema (FALSE)

Contract: TestEventos
  ✓ Test0: Emite evento cuando se registra una empresa en el sistema (77ms)
  ✓ Test1: Emite evento cuando se registra un empleado en el sistema (105ms)
  ✓ Test2: Emite evento cuando se transfieren tokens al registrar una empresa (95ms)
  ✓ Test3: Emite evento cuando se emiten tokens para un empleado desde una empresa (59ms)

21 passing (2s)
```

Ilustración 39. Despliegue y tests en Ganache

Y también se ha desplegado y se han realizado las pruebas en la testnet local de Alastria como muestra la ilustración 40.

```
maria@maria-alastrialocal: ~/Alastria-TokensCorporativos-TFE
Archivo Editar Ver Buscar Terminal Ayuda
maria@maria-alastrialocal:~/Alastria-TokensCorporativos-TFE$ truffle migrate --reset --all --network localAlastria
Compiling ./contracts/Empleados.sol...
Compiling ./contracts/Empresas.sol...
Compiling ./contracts/Migrations.sol...
Compiling ./contracts/OpenZeppelin/IERC20.sol...
Compiling ./contracts/OpenZeppelin/SafeMath.sol...
Compiling ./contracts/Owned.sol...
Compiling ./contracts/PlataformaTokens.sol...
Compiling ./contracts/Token.sol...
Writing artifacts to ./build/contracts

Using network 'localAlastria'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
... 0x77eaf5967441344852a98108ae9ea59e64300f3f0ae5cab4fa743934a1b67e66
Migrations: 0x23ed61cf8da3b84ff5d0d31f285b0c07051d87b
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Replacing PlataformaTokens...
... 0xbe7374b070cdf5c87943abb32d9b661635dac28f2e9565d6c72380ee0e03647
PlataformaTokens: 0x7684fb450de19c57e164d588fe8ff399ba82e16d
Saving artifacts...
maria@maria-alastrialocal:~/Alastria-TokensCorporativos-TFE$

maria@maria-alastrialocal:~/Alastria-TokensCorporativos-TFE$ truffle test
Using network 'test'.

Contract: TestContratos
  ✓ Test0: La plataforma se ha desplegado correctamente y la cuenta administradora tiene el totalSupply de tokens (70ms)
  ✓ Test1: Lista de empresas en el sistema vacia al principio (77ms)
  ✓ Test2: Se impide registrar una empresa desde una cuenta no administradora
  ✓ Test3: Registrar una empresa desde la cuenta administradora (257ms)
  ✓ Test4: Se impide registrar un empleado desde una cuenta que no es de una empresa registrada en el sistema (142ms)
  ✓ Test5: Se impide registrar un empleado desde la cuenta de otro empleado (172ms)
  ✓ Test6: Registrar un empleado desde una cuenta de empresa valida y consultar su info desde esa misma empresa (167ms)
  ✓ Test7: Se impide consultar info de un empleado desde la cuenta de una empresa a la que no pertenece (118ms)
  ✓ Test8: Se impide consultar info de un empleado desde la cuenta de otro empleado (148ms)
  ✓ Test9: Se impide que una empresa emita tokens para un empleado que no es suyo (168ms)
  ✓ Test10: Una empresa emite tokens para un empleado suyo (158ms)
  ✓ Test11: Se impide que un empleado transfiera tokens para otro empleado que no pertenece a su empresa (124ms)
  ✓ Test12: Un empleado transfiere tokens para un companero de su empresa (134ms)
  ✓ Test13: Comprobar que existe un empleado en el sistema (TRUE) (63ms)
  ✓ Test14: Comprobar que no existe un empleado en el sistema (FALSE)
  ✓ Test15: Comprobar que existe una empresa en el sistema (TRUE)
  ✓ Test16: Comprobar que no existe una empresa en el sistema (FALSE)

Contract: TestEventos
  ✓ Test0: Emite evento cuando se registra una empresa en el sistema (102ms)
  ✓ Test1: Emite evento cuando se registra un empleado en el sistema (145ms)
  ✓ Test2: Emite evento cuando se transfieren tokens al registrar una empresa (146ms)
  ✓ Test3: Emite evento cuando se emiten tokens para un empleado desde una empresa (84ms)

21 passing (3s)
```

Ilustración 40. Despliegue y tests en la red local de pruebas de Alastria

También se ha podido desplegar sobre la red Alastria en el nodo de la UNIR pero no se han podido realizar los tests, como era de esperar.

```

maria@maria-alastrialocal:~/truffle-alastria$ truffle migrate --reset --all --network alastria
Compiling ./contracts/Empleados.sol...
Compiling ./contracts/Empresas.sol...
Compiling ./contracts/Migrations.sol...
Compiling ./contracts/OpenZeppelin/IERC20.sol...
Compiling ./contracts/OpenZeppelin/SafeMath.sol...
Compiling ./contracts/Owned.sol...
Compiling ./contracts/PlataformaTokens.sol...
Compiling ./contracts/Token.sol...
Writing artifacts to ./build/contracts

Using network 'alastria'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
... 0x88c146b2b6a317a6b056370a8c81dc37a88fece234faaa88b31e07c359cc1038
  Migrations: 0x360633513d3bfd295a402e5d5f1fbc91a0aa643e
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying PlataformaTokens...
... 0xaf2b594f87c6919f60fb3cd0ebf21240abf2ac1cd319a1d8c6fc3f5846e5d3f9
  PlataformaTokens: 0x21b2ad204f7c64e99437bfd9fb54657a1415c8c5
Saving artifacts...

```

*Ilustración 41. Despliegue con el nodo UNIR en Alastria*

Para poder desplegar los contratos ha sido necesario modificar ligeramente los ficheros JavaScript de migrations para desbloquear la cuenta que iba a desplegar los contratos. Esto ha quedado comentado en el fichero.

## 7. Conclusiones

Al final de este trabajo me gustaría realizar tres conclusiones sobre el desarrollo de aplicaciones blockchain.

La primera de ellas es que, debido a la frenética evolución de las tecnologías blockchain, hay que tener una enorme precaución con las versiones y los entornos. Es necesario trabajar sobre versiones estables y no tender a utilizar la última versión de las distintas herramientas que componen la solución, porque existen después graves problemas de integración. Ejemplos claros son las elecciones de versiones en este proyecto de web3.js, la 0.2, cuando la versión actual es la [1.0@32](#) y la del compilador de Solidity seleccionada, la 0.4.25 cuando ya existe la 0.6.

Mi segunda conclusión es que este proyecto es una prueba más de que blockchain no es solo teoría. Es una tecnología modelable y programable que permite crear soluciones a medida. Es una tecnología como otra cualquiera de backend que se vale de librerías o APIs para integrarse con aplicaciones web tradicionales. Ya existen entornos, herramientas y documentación que permiten a los desarrolladores aprenderla y utilizarla. La tecnología blockchain a día de hoy está mucho más madura de lo que comunmente se piensa.

Y finalmente, la tercera conclusión – más subjetiva - es que blockchain es un mundo apasionante que está en un momento de explosivo crecimiento. Cada tecnología es radicalmente distinta de las demás. Ha sido muy provechoso realizar este proyecto para aproximarme un poco más a Quorum y al increíble proyecto de Alastria, que en cuanto ponga a disposición de los desarrolladores una red estable en producción, será la infraestructura que utilicen múltiples aplicaciones blockchain.

## 8. Referencias

Buterin, V. (30 de 07 de 2015). *Ethereum - White Paper*. Obtenido de <https://github.com/ethereum/wiki/wiki/White-Paper>

Gaitán, V. (3 de 10 de 2013). *educativa*. Obtenido de <https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/>

JP Morgan. (01 de 09 de 2016). *Quorum - White paper*. Obtenido de <https://drive.google.com/file/d/0B8rVouOzG7cOeHo0M2ZBejZTdGs/view>