

TRABAJO FIN DE MÁSTER

MICRODONACIONES EN LA BLOCKCHAIN

MÁSTER EN TECNOLOGÍA BLOCKCHAIN Y
CRIPTOECONOMÍA

Curso 2020/2021
- UPV/EHU -

- 4 de octubre del 2021 -

LIERNI ORTIZ ELORZA

DIRECTOR: ALEXANDER HERRANZ SANTAMARÍA

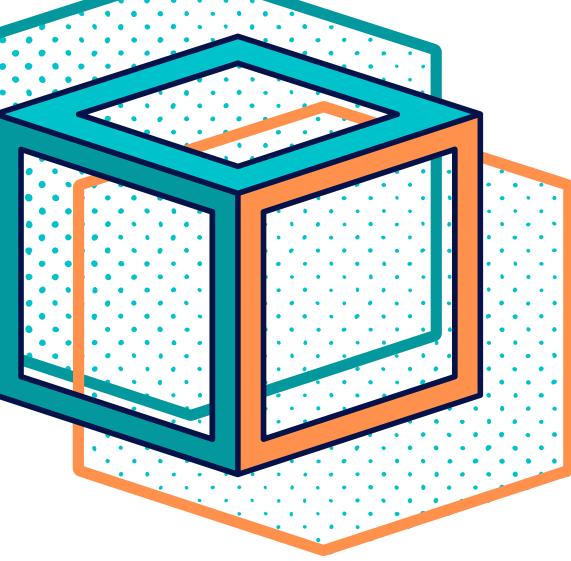
RESUMEN

En el presente trabajo se mostrará el proceso de creación de una plataforma que, de forma descentralizada, permitirá realizar microdonaciones. Éstas estarán orientadas a fomentar una sociedad más mimetizada con la cultura y el bienestar colectivo.

La plataforma se implementa sobre la red Ethereum y se le ha denominado *Kometa*.

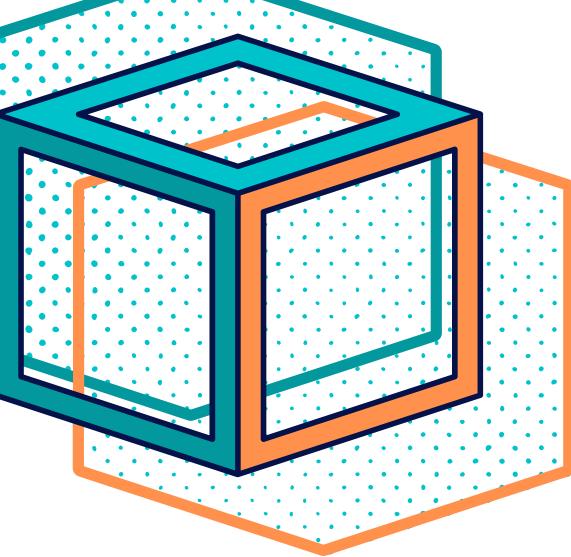
ÍNDICE

CONTENIDO



5	INTRODUCCIÓN
	Blockchain
	El proyecto
8	CONTEXTUALIZACIÓN
	ONG
	Crowdfunding
	Valores
11	OBJETIVOS
12	IDEA
14	CÓDIGO
	Estructura visual de la DApp
	Interfaz web
	Conexión entre la interfaz y el SM
	Cronología del desarrollo
32	CONCLUSIONES
35	REFERENCIAS





INTRODUCCIÓN

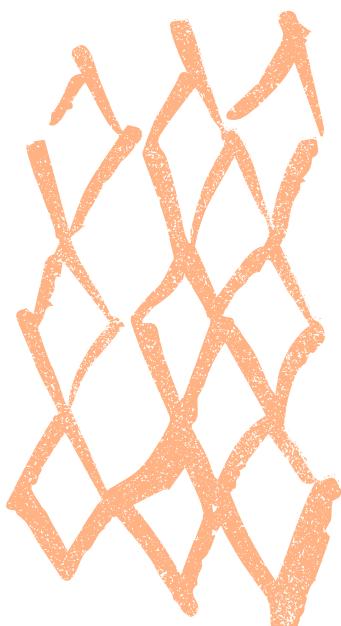
FILANTROPÍA: amor al género humano (RAE, 2021).

En el contexto de la donación de fondos, la filantropía se expresa mediante la dádiva económica. Es más, en el ámbito de la psicología y el *fundraising* se considera, en cierto sentido, un esfuerzo que será más valorado cuanto más desinteresado o abnegado sea. Sin embargo, surgen dos problemáticas a raíz de esto (AEFr, 2021):

- 1 La ‘abnegación’ no es un concepto moderno: donar es más eficaz si el donante se identifica con una organización o causa; pero, de este modo, la donación deja de ser desinteresada.
- 2 A pesar de considerar la donación como un esfuerzo, renunciamos a la oportunidad de sentirnos bien por ello. La Psicología de la Filantropía trata de contrarrestar esa tendencia apoyando a los *fundraisers* y donantes para que se enfoquen en su altruismo y no lo centren en el gesto en sí.

Centrándose en las estadísticas del estado Español se observa que el 37% de la población ha colaborado con alguna entidad no lucrativa durante el 2020 (AEFr, 2020).

El principal motivo (24%) de donar es por cercanía al colectivo. Por el contrario, analizando las razones por las que una persona no quiere donar, caben destacar no tener suficientes medios económicos en un 41% de los casos , y la desconfianza en un 34% (AEFr, 2020).



Dejando las donaciones a ONGs a un lado, el *crowdfunding* (denominación que se está extendiendo como forma de microdonaciones menos oficiales) está tomando más y más fuerza. Las microdonaciones *peer to peer* que ayudan a ciertos individuos o colectivos a alcanzar cierta cantidad prefijada para un proyecto o compra son más habituales día a día. Además son fáciles de difundir mediante plataformas digitales habilitadas para ello.

Este sistema evita, de alguna manera, la centralización económica que se produce al hacer donaciones a una ONG, ya que la organización será la encargada de gestionar el dinero recibido. Aún así, dicha donación depende de la plataforma y los dueños de la misma.

En España el *crowdfunding* es evidente que está también en alza, ya que recaudó 200.827.059€ en 2019, lo cual supone el 25,76% más que el año anterior (González, 2019).

Se pueden definir varios tipos de *crowdfunding* (Blanco, 2020):

- *Crowdfunding* de donación.
- *Crowdfunding* de recompensa.
- *Crowdfunding* de inversión.
- *Crowdlending*.
- *Crowdfactoring*.

Cada uno de ellos se analizará más adelante, ya que se verá cuál se puede implementar al proyecto Kometa teniendo en cuenta los objetivos.

Blockchain

Es una tecnología basada en la web que nos permite el traspaso de valor de forma descentralizada. Es decir, las transacciones realizadas en la misma no cuentan con ningún intermediario, como puede ser un banco, por ejemplo.

Hay diversas redes *blockchain*: públicas, permisionadas y privadas. A continuación, se repasará brevemente la red pública Ethereum, ya que será donde se desarrollará el proyecto.



Ethereum, nacida en 2013, dice ser una red que ofrece acceso abierto a dinero digital y servicios amigables con los datos accesibles para todos, sin importar su origen o ubicación. Es una tecnología construida por la comunidad detrás de la criptomoneda ether (ETH) para miles de aplicaciones que se pueden usar hoy en día (Ethereum, 2021).

Subraya tres casos de uso principales:

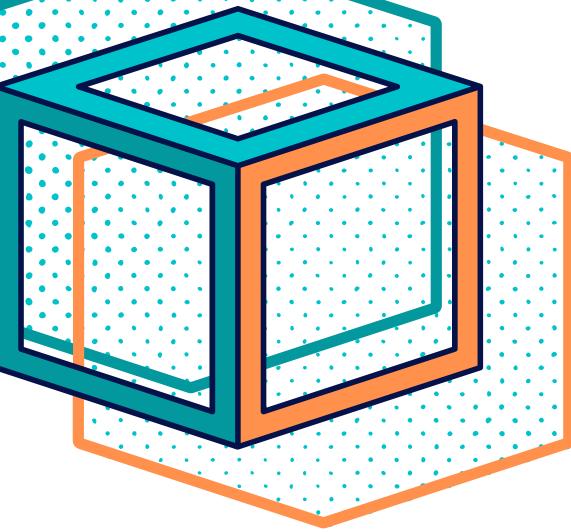
- Finanzas descentralizadas (DeFi).
- Tokens no fungibles (NFT).
- Organizaciones Descentralizadas Autónomas (DAOs)

El proyecto

En este proyecto se contemplarán los casos de uso que brinda Ethereum para canalizar las microdonaciones que hasta ahora se están haciendo a ONGs o mediante *crowdfunding*. De esta manera, se ofrecerá total transparencia al donante, ya que su transacción será traceable en la *blockchain*. Además se reducirán las tasas que un banco o la plataforma de *crowdfunding* pueda exigir.

Primero se analizará la situación actual de las donaciones para entender las necesidades que hay en la sociedad. Después se presentará la idea de la plataforma descentralizada que se creará y, por último, se explicará el código de implementación.

De esta forma se quiere lograr impulsar el sector de la donación.



CONTEXTUALIZACIÓN

Es evidente que la sociedad actual se debe observar desde la perspectiva analizada. Por una parte, está dispuesta a donar si se identifica con el proyecto que se va a llevar a cabo, pero al mismo tiempo no se fía del destino de dicha donación. Varias herramientas han sido creadas para subsanar este problema.

Organizaciones No Gubernamentales (ONG)

A nivel estatal, la "Fundación Lealtad" es la principal encargada de dar acreditaciones a las ONGs que cumplan las características de la figura 1:



Figura 1. 9 principios de la acreditación del Sello Lealtad (Fuente: fundacionlealtad.org)

Las organizaciones que cumplen los 9 principios, consiguen el Sello Lealtad, lo cual los afirma como organizaciones fiables. Aún así, hemos visto que el segundo mayor obstáculo a la hora de donar a ONGs es la poca fiabilidad que ofrecen.

Esto puede deberse a varias razones según la entrevista telefónica mantenida con Leire Agirreazkuenaga de la ONG Saluganda:

- **Desconocimiento del sello.**

No es comúnmente sabido que existe el Sello Lealtad que supone asegurar la fiabilidad de las donaciones.

- **No fiarse de la Fundación Lealtad.**

Puede ser que la gente desconfíe de la fundación misma que da el sello, o que piense que esta se mueve por intereses.

- **Caro para organizaciones pequeñas.**

La obtención del sello no es asequible para cualquier organización. Las que se mueven en presupuestos muy justos para sacar adelante sus proyectos no se pueden permitir el sello.

Crowdfunding

Como ya se ha mencionado en la introducción, se definen actualmente varios tipos de *crowdfunding* (Blanco, 2020):

- **Crowdfunding de donación.**

El usuario dona desinteresadamente a favor de un proyecto.

- **Crowdfunding de recompensa.**

El usuario consigue un producto o servicio a cambio de su donación.

- **Crowdfunding de inversión.**

El usuario recibe una participación en el capital social donde se lleva a cabo la inversión.

- **Crowdlending.**

Consiste en un sistema de financiación por medio del cual los ahorradores o inversores que aportan los fondos, reciben una contraprestación en forma de intereses con una rentabilidad atractiva más el retorno de la inversión inicial.

- **Crowdfactoring.**

Sirve a las empresas como forma alternativa de descontar pagarés.

Los tres primeros tipos de *crowdfunding* se consideran aplicables al proyecto, ya que concuerdan con los objetivos de la plataforma, los cuales se explicarán más adelante.

Valores

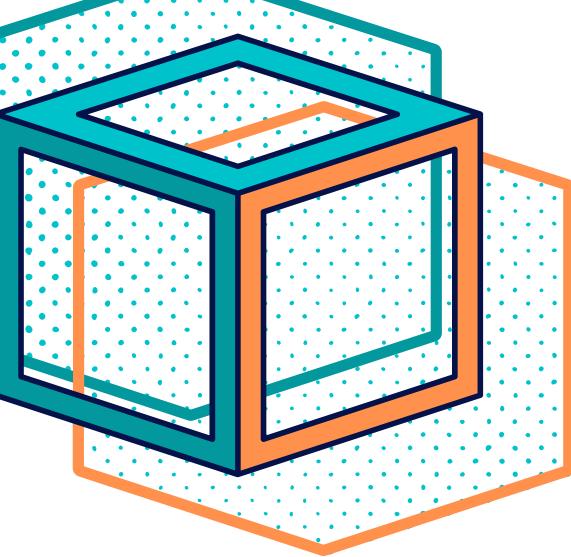
Se ha visto que el 24% de donaciones en España se realizan por cercanía al colectivo, pero hay más razones por las que sucede, y algunas de ellas no tienen conexión con la causa que el proyecto persigue. De hecho, es común que las empresas donen dinero a ONGs o a organizaciones cercanas para salir favorecidas legal o económicaamente.

Según Save the Children (2021), los 4 puntos que las empresas deben tener en cuenta al desgravar donaciones a ONGs para hacer sus cálculos son:

- Esta deducción tiene un límite de base liquidable del 10%.
- No aplica a País Vasco y Navarra, donde la deducción es del 20% y el 25%.
- Si se ha donado en los 2 años anteriores a la misma ONG, la cantidad de desgravación puede ser mayor.
- En el caso de los socios, la desgravación será total en el caso de que se lleve un año colaborando (12 cuotas). Si se llevan menos de 12 meses como socio, la desgravación será proporcional a los meses en los que se haya colaborado durante ese año fiscal.

Teniendo en cuenta la situación expuesta, se observa que hay una razón significativa para que las empresas grandes realicen donativos periódicos. Estos intereses implican que individuos particulares y pequeñas empresas desconfíen del sistema.

Es por esto que esta plataforma está orientada hacia estos últimos y a ONGs de corto presupuesto, puesto que estos grupos son quienes tienen más dificultades en un sistema centralizado. La razón principal de poner el foco en ellos es que se pretende que el donativo tenga un fundamento filantrópico y no de beneficio tributario.

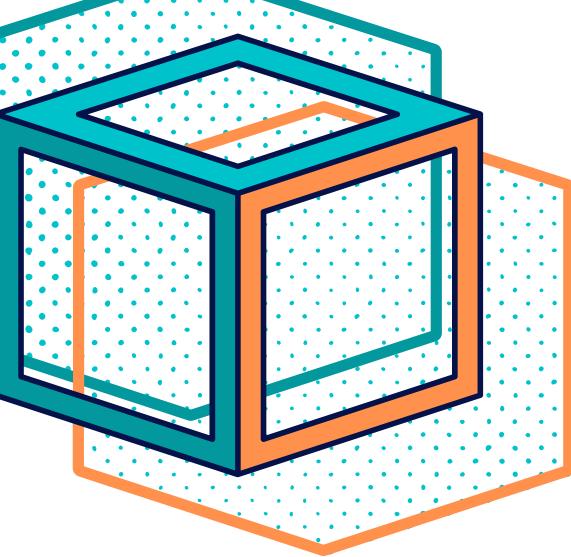


OBJETIVOS

El objetivo general de este proyecto es crear una aplicación descentralizada que permita donaciones *peer to peer* sobre la blockchain Ethereum.

Objetivos específicos :

- Permitir a las ONGs de presupuestos más reducidos obtener donaciones a tasas más bajas que las actuales.
- Facilitar a las ONGs el "transporte" del dinero al lugar de actuación.
- Permitir al usuario de forma transparente un seguimiento de su donación.
- Hacer que la plataforma sea accesible para los usuarios mediante una interfaz de fácil comprensión.
- Fomentar proyectos sociales y culturales.



IDEA



Kometa será una plataforma donde los usuarios puedan realizar donaciones a diferentes proyectos y, además, puedan después hacer seguimiento de dicho dinero.

Teniendo en cuenta los tipos de *crowdfunding*, Kometa diferenciará dos tipos de donaciones:

- **Desinteresada:** el usuario dona desinteresadamente sin esperar nada a cambio.
- **Con recompensa:** el usuario obtiene un NFT a cambio.

La plataforma tendrá dos tipos de usuarios: donantes y/o representantes de proyectos. Los representantes facilitarán obligatoriamente los siguientes datos: nombre del proyecto, contacto, descripción, donación global requerida, imágenes y si es de tipo desinteresado o con recompensa. Estos proyectos quedarán expuestos en la plataforma preparados para recibir donaciones por los usuarios.

Una vez que el usuario haya donado podrá rastrear dicha donación mediante un explorador de la red Ethereum.

Estrategias para el futuro

Se parte de la idea expuesta en la sección anterior para la implementación del código. A futuro, se valorarán los resultados de lo planteado, por lo que no será algo cerrado, sino que se irá adecuando, planificando y adaptando de manera secuencial en los diferentes momentos del proyecto en función de los datos generados y analizados.

Con todo ello se pretende conseguir lo siguiente:

- Incrementar la confianza del usuario.
- Facilitar el uso de la interfaz para el seguimiento de la donación.
- Crear redes de apoyo como sistema de funcionamiento relacional para el fomento social y cultural.

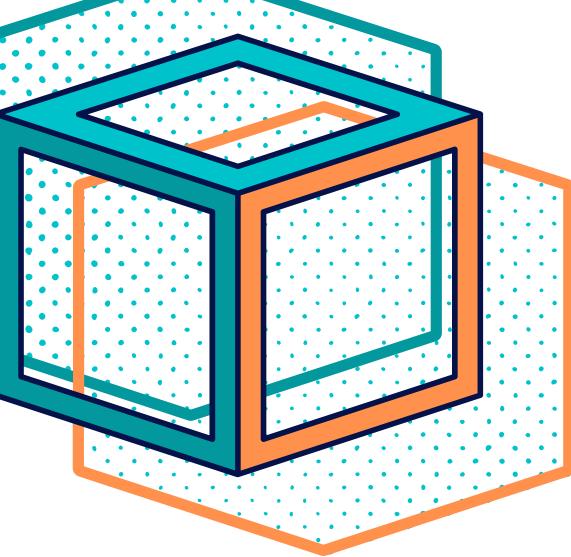
Con el fin de abordar los tres puntos anteriores se han planteado, de momento, varias opciones de las cuales no se presenta el código, quedan como prototipo.

Por un lado, se ha hecho el esquema que permite crear una red de confianza mediante un método de aceptación de proyectos en la plataforma. En él los representantes de proyectos solo podrán subir sus datos si son aceptados por alguien que ya ha participado anteriormente.

Por otro lado, también está presente la idea de verificar el destino de la donación mediante inteligencia artificial (IA). El dinero donado quedaría bloqueado en el *Smart Contract* correspondiente hasta que el representante del proyecto presente facturas acordes con el uso en la descripción facilitada. De esta manera, la factura sería corroborada por la IA y el dinero correspondiente a ella sería desbloqueado para su uso. Si se implementara este sistema no sería necesaria la aceptación de proyectos mencionada anteriormente.

Estos métodos harían frente al gran porcentaje de personas cuya razón principal para no donar es la desconfianza.

La opción a donar con recompensa fomenta la red de apoyo previamente comentada. Ya que esta opción está pensada sobre todo para proyectos cercanos, en los que el usuario pueda participar después de haber aportado económicoamente, sintiéndose así parte del proyecto.



CÓDIGO

En esta sección se explicará el proceso de construcción de la *DApp* *Kometa*. Ésta ha sido desarrollada principalmente utilizando Truffle, el cual es un entorno de desarrollo, un marco de prueba y una canalización de activos para las *blockchain* que utilizan la Máquina Virtual Ethereum (EVM).

Ganache ha sido el entorno *blockchain* de pruebas que se ha utilizado, una red *blockchain* instalada localmente, la cual evita la espera de un nuevo bloque y las tasas correspondientes a las transacciones.

El punto de partida ha sido un esquema el cual muestra la estructura general de la *DApp*. De esta forma se han podido entender los puntos necesarios a programar para un correcto funcionamiento. En el esquema (figura 2) queda explicado el proceso relacional entre el usuario que quiere subir un proyecto o donar a uno:

- El desarrollador migrará el contrato utilizando Truffle a la red *blockchain*. Este proceso se hará una sola vez y el *Smart Contract* (SM) quedará almacenado en una dirección.
- Una vez realizado el despliegue, el usuario podrá acceder a los datos de la *blockchain* teniendo como intermediario Metamask, que será la principal herramienta para realizar transacciones.
- Además de ello, se utilizará IPFS, utilizando el servicio de Pinata, para almacenar las imágenes y evitar demasiado peso en la *blockchain*. Se almacenará la imagen subida por el usuario e IPFS se encargará de devolver el hash de la misma.

Estructura visual de la DApp

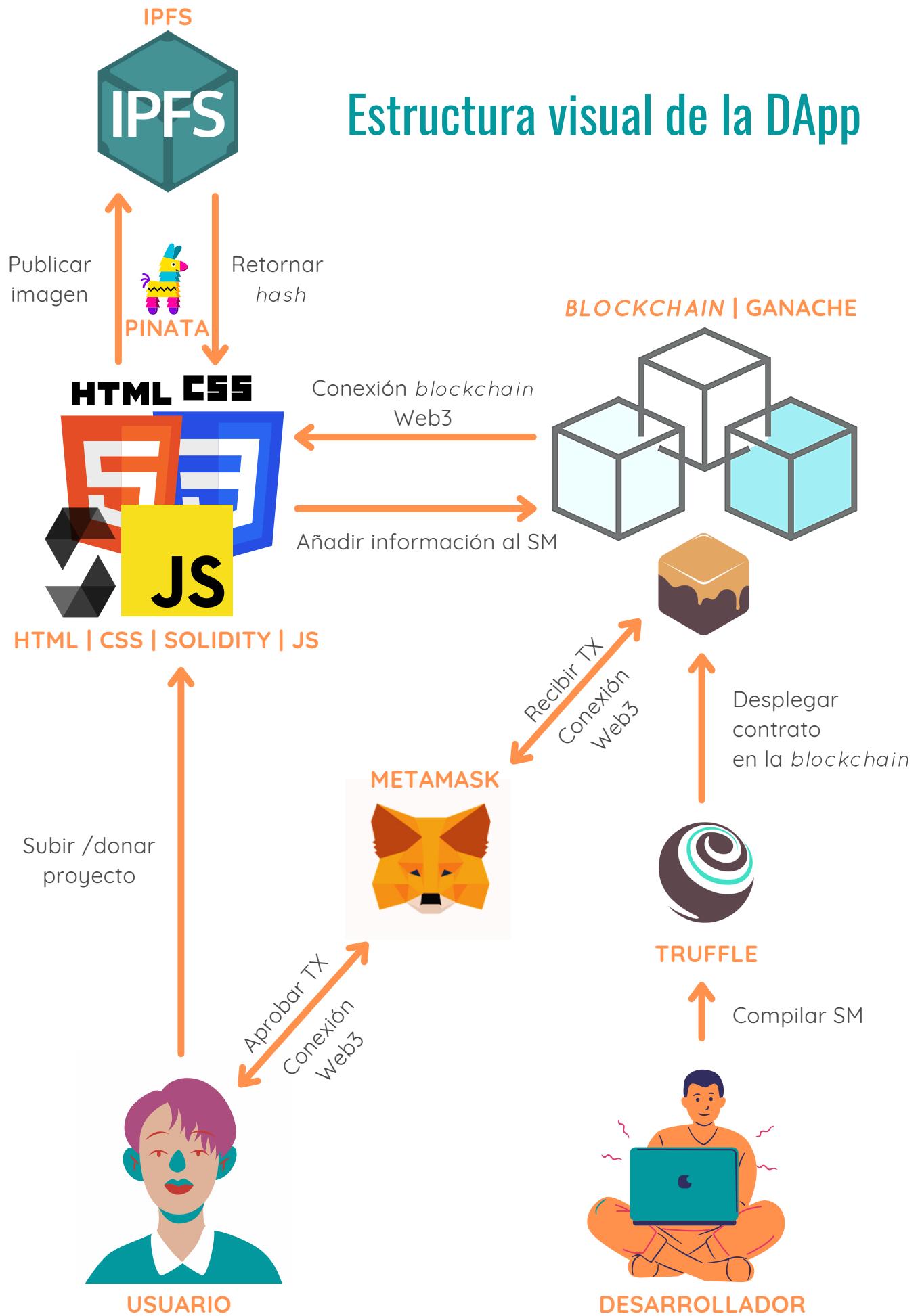


Figura 2. Estructura de Kometa

Interfaz web

Mediante los lenguajes HTML5, CSS y JavaScript se ofrecerá una interfaz al usuario tanto para subir proyectos, como para donar a los que ya están subidos. En la figura 3 están visibles las secciones de la plataforma.



Figura 3. Página inicial de Kometa

Los usuarios que quieran subir un proyecto tendrán que acceder a la sección "Subir un proyecto" y llegarán a un formulario donde indicarán los datos de su proyecto tal y como se muestra en la figura 4. En cambio, aquellos usuarios que quieran donar a un proyecto tendrán que acceder a la sección donar "Donar a un proyecto" y se les mostrará el listado de todos los proyectos subidos tal y como se muestra en la figura 5.

The image shows the 'SUBIR UN PROYECTO' (Upload Project) form. The title 'SUBIR UN PROYECTO' is at the top in a blue header. Below it, the word 'Datos' (Data) is centered. The form consists of several input fields: 'Nombre del proyecto' (Project Name), 'Organización' (Organization), 'Contacto' (Contact), 'Donación requerida (ETH)' (Required Donation (ETH)), and 'Descripción del proyecto' (Project Description). There is also a file upload section with a button 'Seleccionar archivo' (Select file) and a message 'ningún archivo seleccionado' (No file selected). A 'Subir' (Upload) button is at the bottom. The background features the same space-themed illustration as the homepage.

Figura 4. Formulario de datos para subir un proyecto

Conservación de la reserva de chimpancés en el Congo

Organización: ChimpancéCongo

Contacto: chimpancecongo@gmail.com

En una visita a Congo, hace más de dos décadas, la doctora Jane Goodall descubrió una pequeña cría de chimpancé, casi moribunda, atada en un puesto de venta del mercado. Su madre había sido asesinada por su carne y la cría podía valer como mascota. Tras acercarse y hacerle unas vocalizaciones aprendidas en Gombe, la cría respondió con luz en la mirada. Jane sabía que no podía dejar atrás a esa pequeña, y ese ha sido su lema de vida. Hizo intervenir a las autoridades y confiscar la cría, proveyéndole luego de refugio y cuidados veterinarios. Aquella cría fue la primera rescatada.

0.00 % de la donación completada.
Faltan 5 ETH para completar el proyecto.

Cantidad a donar (ETH)

Donar



Figura 5. Ejemplo de un proyecto subido

Además de ello se presentarán secciones para entender bien qué es Kometá o para ver cómo funciona.

No se entrará en los detalles de cómo es el código del HTML o CSS.

Conexión entre la interfaz y el SM

Este código es la parte principal de este proyecto, en el cual se han utilizado los lenguajes JavaScript y Solidity. Con ellos se han realizado las conexiones necesarias a la *blockchain* mediante la ayuda de la librería web3.js.

Para ello, se han creado 3 documentos principales (figura 6): *Donacion.sol*, *app.js* y *donar.js*. En *Donacion.sol* se ha escrito el *smart contract* *Donacion*, el cual será el encargado de escribir o leer de la *blockchain*. Los documentos de JavaScript, utilizando la librería web3.js, llamarán a los métodos de dicho contrato.

Cuando alguien suba un proyecto el proceso será el siguiente: la función *conexionWeb3* se conectará a un proveedor de web3 y las funciones *loadJson()* e *init()* instanciarán el contrato para que se puedan aplicar los métodos escritos en el mismo. La función *recogerDatos()* recogerá los datos del formulario y el método *subirProyecto()* del SM los agregará a la *blockchain*. Después, *escribirDatos()*, mediante el método *getProyecto(id)* se encargará de enseñar todos los proyectos. Por último, si alguien quiere donar, la función *donar(id)* relacionará un identificador con un proyecto y recogerá la cantidad que ha indicado el usuario. Mediante el uso del método *donar(id)* la donación quedará registrada en la *blockchain*.

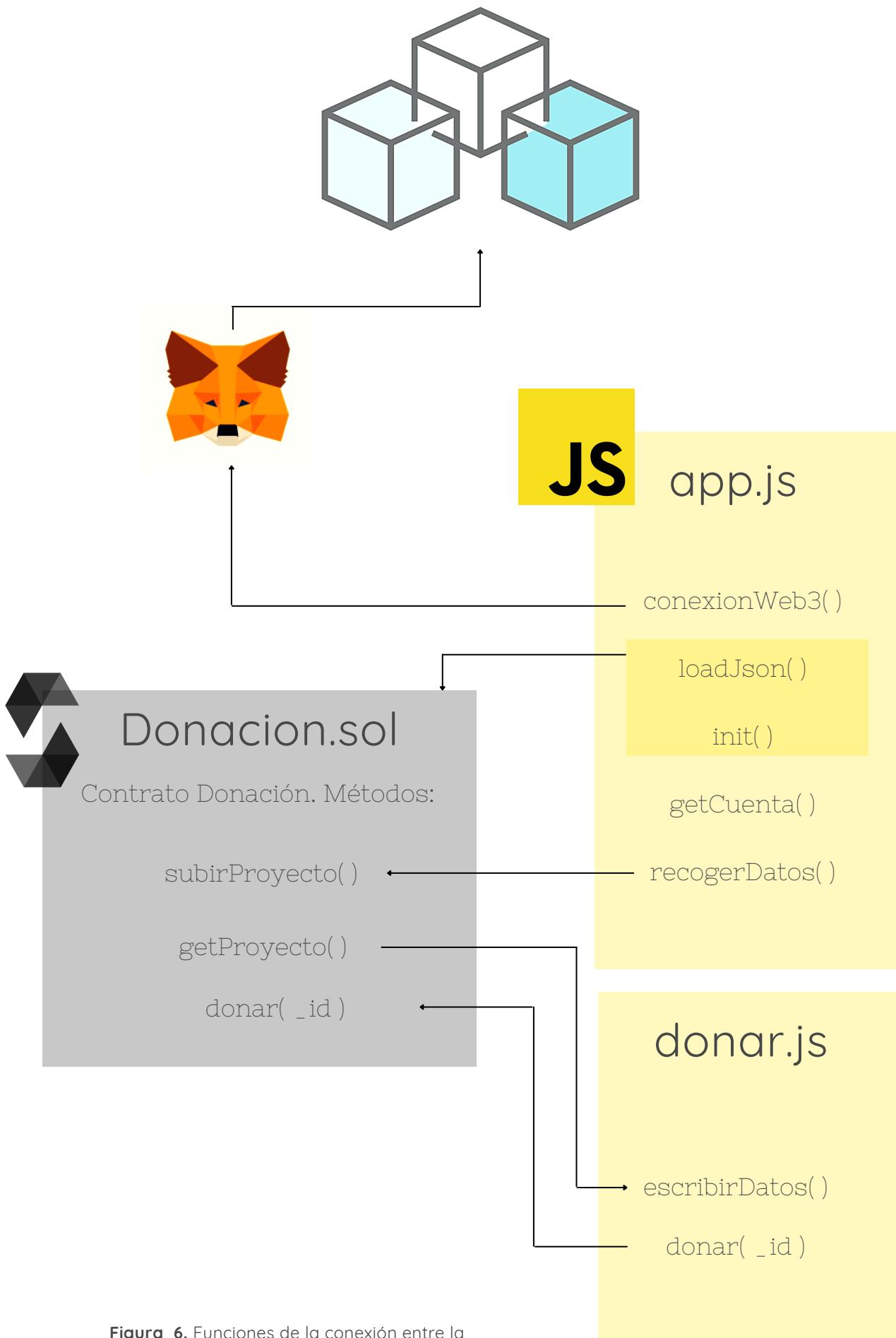


Figura 6. Funciones de la conexión entre la interfaz y el SM

Cronología del desarrollo

En la presente sección se describirá el proceso que se ha seguido para la construcción de *Kometa*. Hay que tener en cuenta que los test han sido con fines académicos y se han realizado siempre en la *blockchain* local Ganache. Además, también se han realizado algunas pruebas en la red Ropsten de Ethereum, por lo que cualquiera puede acceder al SM en dicha red.

1

PREPARACIÓN DEL ENTORNO

Instalar dependencias necesarias:

- node.js

<https://nodejs.org/es/>

- Truffle

```
nmp install --g truffle
```

- Ganache

<https://www.trufflesuite.com/ganache>

ADDRESS	BALANCE	TX COUNT	INDEX	⋮
0xB187c86A3788EB6265353b4e8CC838De67f20Ed	100.00 ETH	0	0	🔗
0xc208FB3b8dA5c8B05967F80125E109660c83e29a	100.00 ETH	0	1	🔗
0xb294850A3e28A396F097512146fB94221aD17C7E	100.00 ETH	0	2	🔗
0x41Ffdc370Faaa962C7FF7BC4E279D59201782769	100.00 ETH	0	3	🔗
0x2dea8Be8e2834e8CE46831c294577343DE4494F7	100.00 ETH	0	4	🔗
0x5e2D82ab71D0e9d9dbCcB22Ae7b50B49833DB763	100.00 ETH	0	5	🔗
0xA1d1e88848E494dd77f3F60E4123C052dFd45474	100.00 ETH	0	6	🔗
ADDRESS	BALANCE	TX COUNT	INDEX	⋮

Figura 7. La blockchain personal Ganache después de instalarla

- Metamask (extensión para el navegador)
<https://metamask.io/download.html>

2

EMPEZAR CON EL CÓDIGO

Truffle brinda la oportunidad de descargar Truffle boxes para la preparación del código. Este proyecto se comenzó por el box llamado "petshop" al cual se le han ido haciendo las modificaciones necesarias para el desarrollo de *Kometa*.

El primer paso es instalar el Node Package Manager. Éste será necesario para instalar módulos y administrar las dependencias a la hora de trabajar con JavaScript.

```
nmp install
```

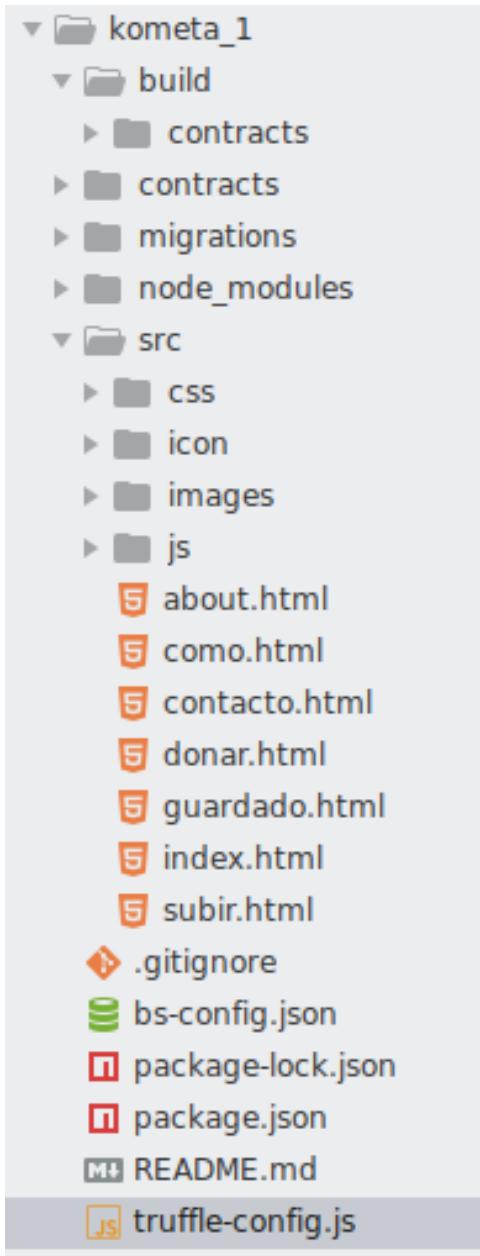


Figura 8. Estructura del proyecto

Una vez descargado el box de Truffle, se dispondrá del *package.json* y del *truffle-config.js* configurados. También se tendrá la carpeta "node_modules" al instalar npm. La estructura resultante se muestra en la figura 8.

Las carpetas "contracts" y "migrations" almacenarán los SM escritos en el lenguaje Solidity y los archivos para migrar los SM a la *blockchain* respectivamente.

La carpeta "build" se creará una vez realizada la migración conteniendo los ABI de los SM en formato *.json*.

La carpeta "src" contendrá todo lo que tenga que ver con la interfaz o los documentos de JavaScript. Este proyecto se comenzó desde una plantilla disponible en la siguiente dirección:

<https://www.free-css.com/free-css-templates/page267/intro>

Esta plantilla también ha sido modificada convenientemente tal como se explicará más adelante.

3 SMART CONTRACTS

El primer paso ha sido escribir el contrato necesario para después desplegarlo a la *blockchain*. Como ya ha sido mencionado se ha escrito en el lenguaje de programación Solidity.

A continuación se explicarán la estructura y las funciones del contrato.

```
1 pragma solidity ^0.5.0;
2
3 contract Donacion{
4     string public nombre = "Donacion";
5
6     uint public proyectoID = 0;
7     mapping(uint => Proyecto) public proyectos; //public para poder llamarlo por fuera del SM
8
9
10
11     struct Proyecto{
12         uint id; //identificador de cada proyecto
13         string hash; //(hash de IPFS)
14         string nombre;
15         string organizacion;
16         string contacto;
17         string descripcion;
18         address payable autor; //Para enviar donaciones al dueño del proyecto
19         uint donacionRecibida; //Donacion que ha recibido
20         uint donacionRequerida; //Donación que desearía recibir
21     }
}
```

Figura 9. Comienzo del SM Donacion.sol

- Se dispondrá de un solo contrato llamado `Donacion.sol` (figura 9).
- Se define una variable `proyectoID`, la cual estará destinada a identificar a cada proyecto con un número entero.
- Se define un *mapping* que será donde quedarán registrados los proyectos subidos identificados con un número. Es importante utilizar la palabra reservada `public` para poder acceder al *mapping* desde fuera del SM.
- Se define la estructura de cada proyecto, que contará con los atributos que se pueden ver en el código de la figura 9 a partir de la línea 11.

```
24     event proyectoSubido(
25         uint id,
26         string hash,
27         string nombre,
28         string organizacion,
29         string contacto,
30         string descripcion,
31         address payable autor,
32         uint donacionRecibida,
33         uint donacionRequerida
34     );
35
36     event proyectoDonado(
37         uint id,
38         string hash,
39         string nombre,
40         string organizacion,
41         string contacto,
42         string descripcion,
43         address payable autor,
44         uint donacionRecibida,
45         uint donacionRequerida
46     );

```

- Se definen dos eventos (figura 10). Uno se emitirá cuando alguien suba un proyecto y el otro cuando alguien haga una donación.

Figura 10. Eventos

```

49   function subirProyecto (string memory _proyHash, string memory _nombre, string memory _organizacion,
50     string memory _contacto, string memory _descripcion, uint _donacionRequerida) public {
51
52     //Asegurarse de que no son vacíos
53     require(bytes(_descripcion).length > 0);
54     require(msg.sender != address(0x0));
55
56     //Incrementar id de proyectos
57     proyectoID++;
58
59     //Añadir proyectos al contrato
60     proyectos[proyectoID] = Proyecto(proyectoID, _proyHash, _nombre, _organizacion, _contacto,
61       _descripcion, msg.sender, 0, _donacionRequerida);
62
63     //Emitir evento al subirse un proyecto nuevo
64     emit proyectoSubido(proyectoID, _proyHash, _nombre, _organizacion, _contacto,
65       _descripcion, msg.sender, 0, _donacionRequerida);
66 }
```

Figura 11. Función subirProyecto

- Esta función (figura 11) es la encargada de registrar los datos de los proyectos.
- Recibirá como parámetros el *hash* de la imagen asociada y subida en IPFS, el nombre del proyecto y de la organización, contacto, la descripción y la donación que les gustaría recibir a los representantes del proyecto.
- Para validar la información, se verificará que haya algo escrito en la descripción y que la función no la llame un *address* vacío.
- Incrementará en 1 la variable *projectoID*, para no coincidir en número de identificación con proyectos subidos anteriormente.
- Guardará el proyecto en el *mapping* definido con el identificador correcto.
- Emitirá el evento correspondiente.
- Es importante destacar que `msg.sender` es el *address* con el que se está transaccionando, por tanto, se dará por hecho que éste es el autor del proyecto y el que recibirá donaciones.

```

68 //Retorna el proyecto correspondiente a un id concreto que recibe como parametro
69 function getProyecto(uint _id) public view returns (uint id, string memory hs, string memory nm,
70   string memory org, string memory cont, string memory des, address add, uint donRec, uint donReq){
71   Proyecto memory _proyecto = proyectos[_id];
72   return(_proyecto.id, _proyecto.hash, _proyecto.nombre, _proyecto.organizacion,
73     _proyecto.contacto, _proyecto.descripcion, _proyecto.autor, _proyecto.donacionRecibida,
74     _proyecto.donacionRequerida);
75 }
```

Figura 12. Función getProyecto

- Esta función (figura 12) permitirá obtener datos de proyectos que ya han sido subidos.
- Recibirá como parámetro un entero, el cual se relacionará con el identificador de un proyecto y devolverá los datos del mismo.
- Es importante recalcar que Solidity no permite el `return` de una estructura entera, pero sí una tupla de sus atributos.

```

78 //DONAR A UN PROYECTO
79 function donar(uint _id) public payable{
80     require(_id > 0 && _id <= proyectoID);
81
82     //Identificar el proyecto al que donar
83     Proyecto memory _proyecto = proyectos[_id];
84
85     //Si la donación requerida ha sido cumplida no se podrá donar
86     require(_proyecto.donacionRecibida < _proyecto.donacionRequerida);
87
88     //Identificar el dueño del proyecto al que se quiere donar
89     address payable _autor = _proyecto.autor;
90
91     //Donar
92     address(_autor).transfer(msg.value);
93
94     //Aumentar el valor de la donación que ha recibido ese proyecto
95     _proyecto.donacionRecibida = _proyecto.donacionRecibida + msg.value;
96
97     //Volver a poner el proyecto en el mapping
98     proyectos[_id] = _proyecto;
99
100    //Emitir evento cuando se dona
101    emit proyectoDonado(_id, _proyecto.hash, _proyecto.nombre, _proyecto.organizacion,
102                         _proyecto.contacto, _proyecto.descripcion, _autor, _proyecto.donacionRecibida,
103                         _proyecto.donacionRequerida);
104
105 }
```

Figura 13. Función donar

- La función de la figura 13 permitirá donar a un proyecto que ya está subido (está en el *mapping*).
- Primero, se comprueba que el proyecto al que se quiere donar existe, y se identifica en el *mapping*.
- Despues se comprueba que el proyecto no haya recibido la donación completa, es decir, que la donación recibida hasta el momento sea menor a la que se requiere.
- Se identifica al autor del proyecto, es decir, el *address* que subió el proyecto, y se procede a realizar transferencia con el valor de la transacción especificada.
- Lo único que queda, es cambiar el parámetro *donacionRecibida* del proyecto y volver añadirlo al *mapping* con los nuevos datos, así quedará actualizado.
- Por último se emite el evento que comunica que ese proyecto ha recibido una donación.

4

DESPLEGAR EL CONTRATO

Una vez que el contrato ha sido escrito hay que compilarlo y desplegarlo en la *blockchain*. Para ello, se tendrá disponible el archivo *1_initial_migration.js*, pero se tendrá que construir el archivo *2_deploy_contracts.js* (figura 14).

```
1 var Donacion = artifacts.require("Donacion");
2 module.exports = function (deployer) {
3     deployer.deploy(Donacion);
4 };
```

Figura 14. *2_deploy_contracts.js*

De esta forma ya estará listo para ser desplegado en la *blockchain* local Ganache. Para ello utilizaremos los siguientes comandos:

Compilar

```
truffle compile
```

Desplegar

```
truffle migrate
```

Resultado [figura 15 y figura 15 (continuación)]

```
Starting migrations...
=====
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)

1_initial_migration.js
=====

Replacing 'Migrations'
-----
> transaction hash:  0xae91c81dffdef3b34191225c848c50699a17983b614f966d6fc35f58c082ec
> Blocks: 0           Seconds: 0
> contract address:  0xc288aa334E21eD52C1E53D070e375Ec9c3ce29F6
> block number:       1
> block timestamp:   1630658371
> account:            0x916C11182D62e014c654601b8CDcd6F848dd04Cf
> balance:             99.99616114
> gas used:            191943 (0x2edc7)
> gas price:            20 gwei
> value sent:          0 ETH
> total cost:           0.00383886 ETH      asegura

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:          0.00383886 ETH
```

Figura 15. Contrato desplegado

```

2_deploy_contracts.js
=====
Replacing 'Donacion'
-----
> transaction hash: 0x027c230706889a7da202571e2747fd75ffd4ff07db12b4119be294fc48265b4b
> Blocks: 0 Seconds: 0
> contract address: 0x3E60dB1031700bB397CF9A431605B081e9E42b52
> block number: 3
> block timestamp: 1630658371
> account: 0x916C11182D62e014c654601b8CDcd6F848dd04cf
> balance: 99.9635293
> gas used: 1589254 (0x184006)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.03178508 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.03178508 ETH

Summary
=====
> Total deployments: 2
> Final cost: 0.03562394 ETH

```

Figura 15 (continuación). Contrato desplegado

Realizado este proceso se podrá observar el cambio de la cuenta en Ganache (figura 16), ya que por los gastos que conlleva el despliegue tendremos menos ETH de los que se proporcionan inicialmente. También se pueden ver los datos referentes a cómo se ha realizado la transacción. Será imprescindible guardar la dirección del SM, para su posterior uso, ya que se tendrá que cargar el contrato en JavaScript para llamar a los métodos correspondientes cuando se lleve a cabo una acción u otra en la DApp.

The screenshot shows the Ganache interface with two tabs visible:

- ACCOUNTS**: Shows the mnemonic "silver tornado object promote mutual proud pelican vital hole mail blouse lift".
- BLOCKS**: Shows the current block (4), gas price (20000000000), gas limit (6721975), hardfork (MUIRGLEACIER), network ID (5777), RPC server (HTTP://127.0.0.1:7545), and mining status (AUTOMINING). It also shows a workspace section with "QUICKSTART", "SAVE", "SWITCH", and a gear icon.

Contract Details:

- Address:** 0x916C11182D62e014c654601b8CDcd6F848dd04cf
- Balance:** 99.96 ETH
- TX Count:** 4
- Index:** 0
- HD Path:** m/44'/60'/0'/0/account_index

Block 1 Details:

- Gas Used:** 191943
- Gas Limit:** 6721975
- Mined On:** 2021-09-03 10:39:31
- Block Hash:** 0xc7d1aef33f6503047da1755e12f60765fee20a55575dfcc39b443b97217f659a

Contract Creation:

- TX Hash:** 0xae91c81dffdef3b34191225c848c50699a17983b614f966d6fc35f58c082ec
- From Address:** 0x916C11182D62e014c654601b8CDcd6F848dd04cf
- Created Contract Address:** 0xc288aa334E21eD52C1E53D070e375Ec9c3ce29F6
- Gas Used:** 191943
- Value:** 0

Figura 16. Ganache después de haber desplegado el contrato

5 HTML y CSS

El código de la interfaz, tal como se ha comentado con anterioridad, se comenzó desde una plantilla y en la misma se han ido haciendo los cambios necesarios.

La DApp cuenta con dos secciones principales en la interfaz: una para subir proyectos y otra donde se pueden ver todos los proyectos en los que se puede donar. Además de ello cuenta con varias secciones ilustrativas donde se explica qué es *Kometa* y cómo funciona.

6 JavaScript

Teniendo la interfaz construida, hay que hacer el puente entre la misma y la *blockchain*. Para ello, se ha creado el archivo *app.js* y el primer paso ha sido construir una función de conexión a Web3 (figura 17).

```
4 function conexionWeb3() {
5     //PROTOCOLO: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1193.md
6     if (window.ethereum) {
7         web3Provider = window.ethereum;
8         try {
9             // Solicitar acceso a la cuenta
10            window.ethereum.request({ method: "eth_requestAccounts" });
11            alert("Conectado a Web3");
12        } catch (error) {
13            // Acceso denegado...
14            alert("Necesitas un proveedor de web3");
15        }
16    }
17    // Otros dapp browsers
18    else if (window.web3) {
19        web3Provider = window.web3.currentProvider;
20    }
21    // Si no se detecta ningún proveedor de web3, conectar de nuevo con Ganache
22    else {
23        web3Provider = new Web3.providers.HttpProvider("http://localhost:7545");
24    }
25
26    web3 = new Web3(window.ethereum);
27 }
```

Figura 17. Función de conexión a web3

- La función de la figura 17 sigue el protocolo especificado en la EIP-1193 de Ethereum para la conexión a web3.
- Se dan 3 opciones diferentes: en el primer `if` se ve si ya hay algún intermediario como Metamask instalado; si no es así se buscará otra vía. Por último, si ninguna de las dos opciones anteriores es posible se volverá a la conexión con la *blockchain* local Ganache.
- Si Metamask está instalado se abrirá automáticamente cuando se ejecute esta función.

El objetivo final del código es registrar las acciones del usuario en el SM. Para ello se necesita instanciar el contrato, de esta forma se podrán aplicar los métodos ya explicados del SM. Se han construido dos funciones para esto (figura 18): `loadJSON()` y `init()`.

```

38 //Cargar el Json
39 async function loadJSON() {
40   const file = "http://localhost:3000/Donacion.json";
41   const promise = await fetch(file);
42   return promise.json();
43 }
44
45 //Instanciar el smart contract
46 async function init(){
47   const contract = await loadJSON();
48   let contractAbi = contract.abi;
49   let contractInstance = new web3.eth.Contract(
50     contractAbi,
51     "0x6536170DD8205deDb2eaf01afe35eb8F5eDAlaB1"
52   );
53   return contractInstance;
54 }
```

Figura 18. Instancia del contrato

- En `loadJSON()` se pide el recurso `Donacion.json` al servidor en el que está el proyecto, en este caso, como lo estamos ejecutando localmente está en `localhost:3000`. Después se devuelve en formato `.json`.
- En `init()` convertimos el archivo JSON a la forma ABI y de esta manera creamos un nuevo objeto de tipo contrato mediante el método `eth.Contract` de `web3`. Para ello, se pasan dos parámetros: el ABI del contrato y la dirección donde se encuentra el contrato desplegado. Esta función nos devuelve el contrato con el atributo `methods` para su posterior uso.

Una vez construidas estas funciones se tendrá acceso a los métodos del contrato. Con ello, se puede escribir la función (figura 19) que recogerá los datos del formulario y subirá los proyectos.

```

64 //Recoge datos del formulario de inscripción de proyectos y los sube a la blockchain
65 async function recogerDatos() {
66   conexionWeb3();
67   const contractInstance = await init();
68
69   //Recoger datos del formulario
70   const nombreProyecto = document.getElementById("nombre").value;
71   const nombreOrganizacion = document.getElementById("organización").value;
72   const contacto = document.getElementById("contacto").value;
73   const donReq = document.getElementById("donReq").value;
74   const descripcion = document.getElementById("descripcion").value;
75   const img = document.getElementById("imagen").files[0];
76
77   const ac = await getCUENTA();
78
79   //Imagen a IPFS vía Pinata
80   const base64 = await toBase64(img);
81   const imgResponse = await uploadImage(base64);
82   const hs = imgResponse.cid;
83
84   //Subir proyecto al mapping del contrato
85   const donReqWei = web3.utils.toWei(donReq, "ether");
86   let tx = await contractInstance.methods.subirProyecto(hs, nombreProyecto, nombreOrganizacion,
87   contacto, descripcion, donReqWei).send({from: ac});
88   let txHash = tx.transactionHash;
89
90   //Dar a conocer al usuario su transacción
91   let container = document.querySelector("#datos");
92   container.innerHTML = `<div class="titlepage">
93     <h1>PROYECTO SUBIDO!</h1>
94   </div>
95   <div class="titlepageS">
96     <h2><i>${nombreProyecto}</i></h2><br>
97     <h3>Hash de tu transacción: </h3>
98     <div class="hs"><p> ${txHash} </p></div><br><br><br>
99     <p><a href="https://ropsten.etherscan.io/tx/${txHash}">
100       Accede al explorador de tu transacción aquí</a></p>
101   </div>`;
102
103   //Recoger evento emitido por el contrato
104   const eventos = await contractInstance.getPastEvents("proyectoSubido", {});
105   const ultimoEvento = eventos[0];
106
107 }
```

Figura 19. Función `recogerDatos()`

- El primer paso para recoger datos es la conexión a web3 realizada en la línea 66 de la figura 19 e instancia del contrato por las razones anteriormente mencionadas.
- Después se recogen los valores que el usuario ha introducido en el formulario con el método `getElementById`.
- Después se obtiene la cuenta que está subiendo el proyecto mediante la función de la figura 20 y el *hash* de la imagen que se ha subido a IPFS mediante la función de la figura 21, de la cual se explicará el funcionamiento a continuación.

```

31 //Devuelve la cuenta con la que está operando
32 async function getAccount() {
33   window.ethereum.enable();
34   let acc = await web3.eth.getAccounts();
35   return acc[0];
36 }

```

Figura 20. Función `getAccount()`

- Por último, se llama a `subirProyecto` del contrato y el proyecto se sube utilizando el método `send`. Se tiene que utilizar este método ya que se va a modificar el contrato, es decir, el *mapping* tendrá un elemento más.
- Una vez realizada la transacción se recoge el *hash* de la misma para enseñársela al usuario y redirigirlo al explorador de bloques.

```

97 async function uploadImage(base64) {
98   const config = {method: "POST", body: JSON.stringify({base64: base64}),
99   headers: {"Content-Type": "application/json"}};
100  const url = "http://app-a3ce6b6a-db3a-4579-9eb8-af58ba78ea82.cleverapps.io/upload";
101  const response = await fetch(url, config);
102  return response.json();
103 }

```

Figura 21. Función `uploadImage()`

Para subir la imagen a IPFS se hace uso de Pinata. De esta forma se hace un *pin* de la imagen desde Pinata a IPFS, sin la necesidad de un nodo de IPFS instalado. La arquitectura de este proceso se muestra en la figura 22.

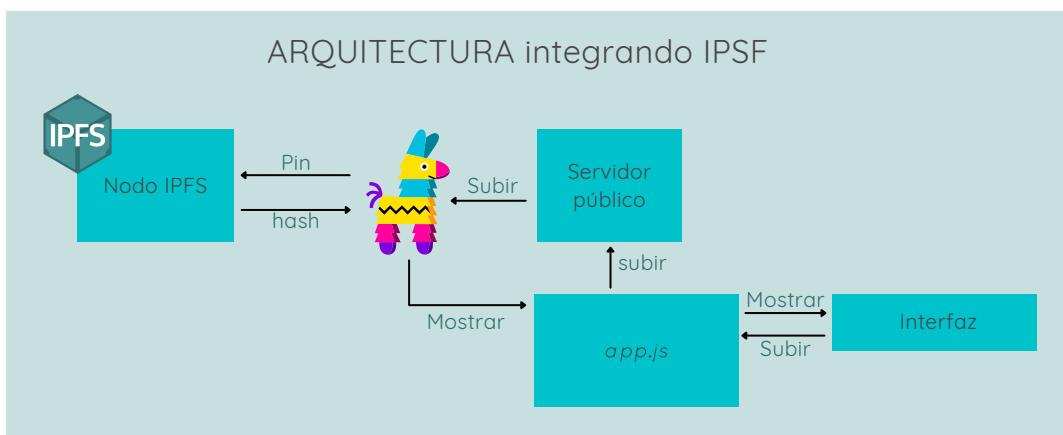


Figura 22. Arquitectura de la conexión a IPFS

Desde la interfaz se recoge la imagen (en base 64) tal y como se muestra en la línea 80 de la figura 19, y desde ahí se utiliza un servidor público creado con node.js para subirlo a Pinata. Allí, se hace el *pin* a IPFS obteniendo el *hash* donde está almacenado.

A continuación se explicarán las partes más relevantes del servidor creado, teniendo como referencia la figura 23. El código entero está disponible en: <https://github.com/lierniortiz/pinata>

```

1  require('dotenv').config()
2  const express = require('express')
3  const cors = require("cors")
4  const app = express()
5  const bodyParser = require('body-parser')
6  const fs = require('fs')
7  const Readable = require("stream").Readable
8
9  const pinataSDK = require('@pinata/sdk');
10 const yourPinataApiKey = process.env.API_KEY
11 const yourPinataSecretApiKey = process.env.API_SECRET
12
13 const pinata = pinataSDK(yourPinataApiKey, yourPinataSecretApiKey);
14
15 app.use(cors()) //esto nos devuelve la cabecera necesaria con Access-Control-Allow-Origin
16 // parse application/json
17 app.use(bodyParser.json({limit: "50mb"}))
18
19 app.post('/upload', async function (request, response) {
20   const imgBuffer = Buffer.from(request.body.base64, "base64") //Buffer es de node
21   var s = new Readable()
22   s.push(imgBuffer)
23   s.push(null) //para cerrar el readable
24   s.pipe(fs.createWriteStream("temp.png"));
25
26   const file = fs.createReadStream("temp.png")
27
28   const result = await pinata.pinFileToIPFS(file, {});
29   response.json({status: "ok", cid: result.IpfsHash})
30 })
31
32 app.get('/', function (request, response) {
33   response.json({status: "ok"})
34 })
35
36 const port = process.env.PORT
37 app.listen(port, () => {
38   console.log(`Example app listening at http://localhost:${port}`)
39 })

```

Figura 23. Servidor público para subir imágenes a Pinata

- Se observa en la línea 98 de la figura 21 que la imagen se recoge en una propiedad *body* que almacena un objeto llamado *base64*. Como se muestra en la línea 20 de la figura 23 se hace un *buffer* de este.
- En la línea 9 de la figura 23 se llama a *Pinata NodeJS SDK*, después de instalarlo mediante:

```
npm install --save @pinata/sdk
```

- De esta forma, se sube la imagen a IPFS a través de Pinata y se obtiene el *hash* mediante el atributo *IpfsHash*.
- Este dato es el que se devuelve en la función de la figura 21 mediante el atributo *cid*.

Este *hash* es el que quedará almacenado en el *mapping* del contrato, y la imagen quedará en una dirección de este tipo:

"<https://gateway.pinata.cloud/ipfs/hash>"

De esta forma, se llamará a la imagen a esa dirección.

Una vez habilitada la función de subir proyectos, se crea el documento `donar.js` para gestionar la donación a los proyectos. Se divide en dos partes: la que enseña los proyectos en la interfaz leyéndolos desde la `blockchain` (figura 24) y la que permite donar (figura 25).

```

1 window.addEventListener("load", function () {
2   window.conexionWeb3();
3   escribirDatos();
4 });
5
6 async function escribirDatos() {
7   const ac = await get cuenta();
8   if (ac != undefined)
9     document.getElementById("cuenta").innerHTML = "Cuenta: " + ac;
10
11 const contractInstance = await init();
12 let proyectos = [];
13 let cantidadProyectos = await contractInstance.methods.proyectoID().call();
14
15 for (i = 1; i <= cantidadProyectos; i++) {
16   proyectos.push(await contractInstance.methods.getProyecto(i).call());
17 }
18
19 let datosProyectos = "";
20 for (proyecto of proyectos) {
21   if (proyecto.donReq - proyecto.donRec > 0) {
22     id = "donacion" + proyecto.id;
23     datos =
24       <div id="donar" class="donar">
25         <div class="titulo" id="nombreProyecto">${proyecto.nm}</div>
26         <div class="row display_boxflex">
27           <div class="org">
28             <div class="box-text">Organización: ${proyecto.org}</div>
29           </div>
30           <div class="orgR">
31             <div class="box-text">Contacto: ${proyecto.cont}</div>
32           </div>
33         </div>
34         <div class="row display_boxflex">
35           <div class="col-xl-5 col-lg-5 col-md-5 col-sm-12">
36             <div class="box_text">
37               <p>
38                 ${proyecto.des}
39               </p>
40               <br>
41               <p>
42                 ${((proyecto.donRec * 100) / proyecto.donReq).toFixed(2)}
43                 % de la donación completada.
44               </p>
45               <p>
46                 Faltan ${(proyecto.donReq - proyecto.donRec) / 10 ** 18}
47                 ETH para completar el proyecto.
48               </p>
49               <form class="donar_py" type="POST">
50                 <div class="row">
51                   <div class="col-md-12">
52                     <div class="col-md-12">
53                       <input
54                         class="contactus"
55                         placeholder="Cantidad a donar (ETH)"
56                         type="number"
57                         id=${id}
58                         required
59                         />
60                     </div>
61                     <div class="col-xl-12 col-lg-12 col-md-12 col-sm-12">
62                       <button
63                         id="btnDonar"
64                         class="btnDonar"
65                         type="button"
66                         onclick="donar(${proyecto.id})"
67                         >
68                         Donar
69                     </div>
70                   ... html...
71                   <div class="col-xl-7 col-lg-7 col-md-7 col-sm-12 border_right">
72                     <div class="upcoming">
73                       <figure></figure>
74                     </div>
75                   </div>
76                   <div class="container">
77                     <div class="gitHub"></div>
78                   </div>
79                 </div>
80               </div>
81             </div>
82           </div>
83         </div>
84       </div>
85     </div>
86   </div>
87   ;
88
89   datosProyectos = datos + datosProyectos;
90 }
91
92 let container = document.querySelector("#proyectos_container");
93 container.innerHTML = datosProyectos;
94
95 }
```

Figura 24. Enseña los proyectos leyendo desde la `blockchain`

- En la línea 13 de la figura 24, mediante la propiedad `projectId` del contrato se sabe cuantos proyectos hay en el *mapping*. Con un bucle creamos un *array* que contiene todos los proyectos del *mapping*. Estos se leen con el método `getProyecto` del contrato. Se utiliza el método `call` porque no se va a modificar nada del contrato, simplemente se requiere información del mismo.
- Una vez hecho esto se va escribiendo todo en el HTML.
- Hay que tener en cuenta que la imagen mostrada está IPFS obteniéndola mediante el *hash* proporcionado por Pinata, como se puede observar en la línea 78 de la figura 24.

Por último la función que permite donar a los proyectos se muestra en la figura 25.

```

97 //Función que permite donar
98 async function donar(_id) {
99   const contractInstance = await init();
100  let donante = await getCuenta();
101  let proyecto = await contractInstance.methods.getProyecto(_id).call();
102  let idHtml = "donacion" + proyecto.id;
103  let cantidad = document.getElementById(idHtml).value;
104  //Realizar la transacción
105  let tx = await contractInstance.methods
106    .donar(_id)
107    .send({ from: donante, value: web3.utils.toWei(cantidad, "ether") });
108  let txHash = tx.transactionHash;
109  //Dar a conocer al usuario su transacción
110  let container = document.querySelector("#proyectos_container");
111  container.innerHTML = `<div class="fondo">
112    <div class="titlepage">
113      <h1>¡PROYECTO DONADO!</h1>
114    </div>
115    <div class="titlepageS">
116      <h2><i>${proyecto.nm}</i></h2><br>
117      <h3>Hash de tu transacción: </h3>
118      <div class="hs"><p> ${txHash} </p></div><br><br><br><br>
119      <p><a href="https://ropsten.etherscan.io/tx/${txHash}">
120        Accede al explorador de tu transacción aquí</a></p>
121      </div>
122    </div>`;
123  //Recoger eventos
124  const eventos = await contractInstance.getPastEvents("proyectoDonado", {});
125  const evento = eventos[0]
126 }
```

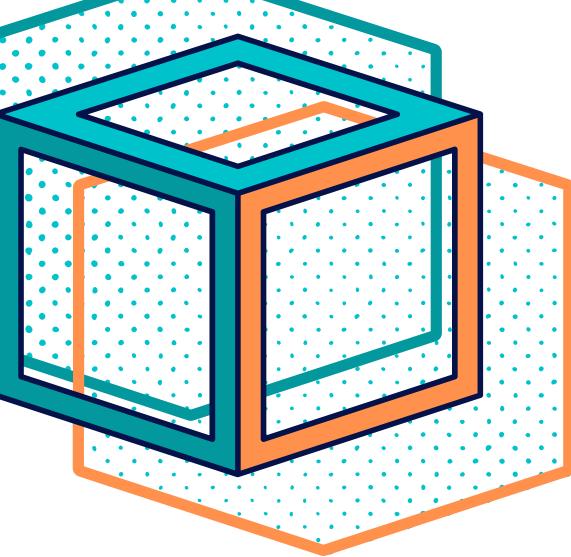
Figura 25. Funcion `donar(_id)`

- Como en todos los casos se comienza instanciando el contrato.
- En las líneas 100, 101 y 103 se obtienen la cuenta desde donde el donante está operando, el proyecto al que desea donar y la cantidad respectivamente.
- Después, en la línea 105, se llama al método `donar` del contrato para transferir el valor especificado.
- Por último, tal y como se ha realizado en la función de la figura 19, se le facilita al usuario el *hash* de su transacción y se recoge el evento emitido por el contrato.

El código entero está disponible en:
<https://github.com/lierniortiz/kometa>



Aunque la aplicación no esté en ningún servidor público, el contrato está en la red Ropsten, por lo que se pueden hacer pruebas con un servidor local sin utilizar Ganache.



CONCLUSIONES

Este Trabajo Fin de Máster es un prototipo de una plataforma digital descentralizada que permite subir proyectos y realizar donaciones *peer to peer* basadas en *Ethereum*.

Se considera que el trabajo tiene dos partes: por un lado, el prototipo diseñado, donde se explica el porqué de la necesidad de la plataforma y el diseño ideal pretendido. Y, por otro lado, el código implementado: un punto de partida para lograr convertir en algo real dicho prototipo. Por tanto, se analizarán las conclusiones en dos segmentos: el prototipo propuesto y el código realizado.

Prototipo

Esta plataforma permitiría a las ONGs recibir donaciones de forma sencilla sin pasar por ningún intermediario. Los usuarios estarían más dispuestos a donar puesto que podrían seguir su donación en la red mediante el *hash* de su transacción, superando con esto el obstáculo de la desconfianza observado en las estadísticas. Además, los sistemas de confianza propuestos -la aceptación de proyectos o las facturas leídas mediante IA- refuerzan esta confianza aún más, ya que sin estos sistemas se podría perder el rastro del dinero donado en la *blockchain*.

Sin embargo, hoy en día las altas tasas de Ethereum podrían ser un gran impedimento para las microdonaciones, ya que la tasa misma puede ser más alta que la cantidad de donación. Este obstáculo debería ser resuelto con la implantación de Ethereum 2.0.

Para ONGs que actúan en el extranjero, tener su dinero en la *blockchain* es una gran ventaja por el hecho de que con unas claves y conexión a internet pueden acceder a él en el lugar de actuación. Así, se evitarían las tasas de transferencias internacionales establecidas por los bancos, las cuales son muy altas para proyectos con bajo presupuesto. Asimismo, esto favorecerá, sobre todo, a ONGs con escasos recursos.

La interfaz de la plataforma no se ha considerado relevante para este Trabajo Fin de Máster, por lo que ha quedado sin especificar. Por tanto, como trabajo para futuro queda pendiente el diseño y la creación de una interfaz que facilite la interacción con el usuario mediante una pasarela de pago que permita la donación en euros con tarjeta de crédito, ya que no estamos acostumbrados al uso de criptomonedas. Además, se creará una sección, en conexión a un oráculo, para poder rastrear la donación fácilmente sin tener que acceder a un explorador.

Se observa que las páginas de *crowdfunding* están teniendo en los últimos años cada vez más relevancia, por lo que se debe suponer que la plataforma propuesta para este Trabajo Fin de Máster desempeñaría un papel importante en los proyectos sociales y culturales por la confianza que ofrece frente a las páginas actuales. Además, se ha diseñado la posibilidad de ofrecer NFT a cambio, lo cual hace participar al usuario en el proyecto en el que está donando. A modo de ejemplo, se pueden ofrecer entradas al evento cultural, algún recuerdo de lo que se quiere crear en el proyecto, un informe sobre cómo ha transcurrido el mismo...

Como reflexión final, cabe decir que habría que diseñar un sistema de ciberseguridad apropiado antes de transformar la plataforma en una realidad. De esta forma, se garantiza un funcionamiento correcto sin que nadie pueda modificar el código a su favor. Dicho código será abierto y público para todos y habría que diseñar un sistema de votación para poder cambiarlo.

Lo ideal, sería construir la plataforma sin un *backend*, ya que de esta forma se evita el *man in the middle attack*, aunque esto no siempre se puede llevar a la realidad por cuestiones de almacenamiento. En el momento en el que se introduce un servidor para algún servicio necesario, este servicio es centralizado, lo cual se desvía ligeramente de la filosofía de la DApp. Aun así, sabiendo que existe el riesgo de un *hackeo* del SM o del código, el usuario puede darse cuenta fácilmente de ello, por lo que podrá actuar con celeridad. Esto es una ventaja frente a las aplicaciones centralizadas actuales donde puede resultar difícil de detectar el *hackeo* del servidor.

Código

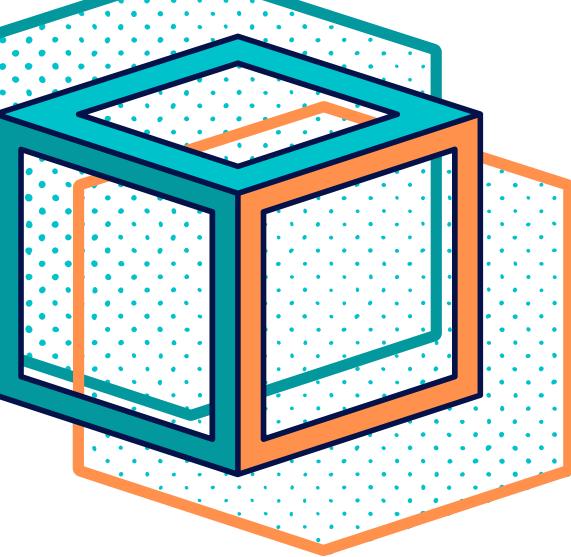
El código programado y presentado es un inicio básico del prototipo diseñado, el cual se ha programado para que la interfaz sea sencilla, fácilmente comprensible y manejable. Además, también es fácil obtener el *hash* después de haber realizado una transacción, lo cual es acorde con los objetivos propuestos, ya que acudiendo a un explorador será accesible rastrearla.

Se ha tenido en cuenta el orden de aparición de los proyectos de forma que el último subido sea el primero en aparecer en la lista. Sería conveniente programar un buscador para poder navegar más fácilmente por los proyectos expuestos.

Las pruebas realizadas han sido informales y en una blockchain local, lo cual asemeja la realidad. Funcionan correctamente, pero al desplegarlo oficialmente podrían surgir excepciones que se deben que observar. Para ello, habría que hacer más test que servirían para verificar el correcto funcionamiento de todos los casos posibles. También, habría que modificar las funciones de modo que paren la aplicación en caso de error.

En cuanto a vulnerabilidad de *hackeo* el momento más sensible que se identifica es en el que se suben las imágenes a Pinata, ya que pasamos por un servidor para ello. Esta es la única parte de la DApp que contiene un *backend*, puesto que todas las demás acciones se realizan conectándose directamente con la *blockchain*.

El trabajo, aunque aparentemente sea una plataforma sencilla, ha sido laborioso y ha resultado ser un valioso proceso y sistema de aprendizaje, tanto en lo referente a conocimientos de programación web y *blockchain* como en sistemas de donación actuales, así como el seguimiento de estos.



REFERENCIAS

AEFr. (2020). Asociación Española de Fundraising. Recuperado el día 2 de julio del 2021 desde: [Resultados del Perfil del Donante 2020: https://www.aefundraising.org/resultados-del-perfil-del-donante-2020/](https://www.aefundraising.org/resultados-del-perfil-del-donante-2020/)

AEFr. (2021). Asociación Española de Fundraising. Recuperado el día 2 de julio del 2021 desde: Psicología de la filantropía: <https://www.aefundraising.org/psicologia-de-la-filantropia/>

Blanco Martínez, M. (2020). Rankia. Recuperado el día 13 de julio del 2021 desde: [¿Qué tipos de crowdfunding existen?: https://www.rankia.com/blog/crowdfunding-crowdlending/3468321-que-tipos-crowdfunding-existen](https://www.rankia.com/blog/crowdfunding-crowdlending/3468321-que-tipos-crowdfunding-existen)

Ethereum. (2021). Recuperado el día 2 de julio del 2021 desde: <https://ethereum.org/es/>

Express. (2021). Expressjs-starter. Recuperado el día 28 de septiembre del 2021 desde:
<https://expressjs.com/es/starter/hello-world.html>

Fundación Lealtad. (2021). 9 Principios de Transparencia y Buenas Prácticas. Recuperado el día 13 de julio del 2021 desde: <https://www.fundacionlealtad.org/si-eres-ong-transparencia-y-buenas-practicas/conoce-los-9-principios/>

González Romero, A. (2019). Universo crowdfunding. Recuperado el día 7 de julio del 2021 desde: <https://www.universocrowdfunding.com/el-crowdfunding-recaudo-en-espana-mas-de-200-millones-de-euros-en-2019/>

IPFS. (2021). Documentation ipfs. Recuperado el día 2 de septiembre del 2021 desde: <https://docs.ipfs.io/>

IPFS. (2021). js-ipfs v0.52.0. Recuperado el día 2 de septiembre del 2021 desde: <https://js.ipfs.io/>

npm. (2021). Pinata-SDK. Recuperado el día 28 de septiembre del 2021 desde:

<https://www.npmjs.com/package/@pinata/sdk#pinFileToIPFS-anchor>

Save the Children. (2021). Recuperado el día 2 de septiembre del 2021 desde: CALCULADORA PARA DESGRAVAR DONACIONES A ONG EN LA DECLARACIÓN DE LA RENTA: <https://www.savethechildren.es/desgravar-donaciones-ong-calculadora-renta>

Solidity. (2021). Documentation v0.8.7. Recuperado el día 25 de agosto del 2021 desde:

<https://docs.soliditylang.org/en/v0.8.7/>

Trufflesuite. (2021). Recuperado el día 25 de agosto del 2021 desde:

<https://www.trufflesuite.com/tutorial>

web3. (2021). Documentation v1.5.2. Recuperado el día 25 de agosto del 2021 desde:

<https://web3js.readthedocs.io/en/v1.5.2/>