

|  |
| --- |
| **Universidad Internacional de La Rioja (UNIR)**  **Escuela Superior de Ingeniería y Tecnología**  **Grado en Ingeniería Informática** |
|  |
| Petchain: DApp de gestión de mascotas |

**Ubicación del código fuente:**<https://github.com/cescari/PetChain_TFG>

**Trabajo Fin de Grado**

**Presentado por:** Escario Bajo, Carlos

**Director/a:** Martínez Muñoz, Miriam.

Ciudad: San Lorenzo de El Escorial (Madrid)

Fecha: 05/06/2019

*Por y para Carolina.*

Resumen

El presente trabajo es una muestra de cómo es posible la realización de proyectos de gestión con Smartcontracts dentro de un ámbito corporativo o administrativo. La aplicación, bautizada como **Petchain**, pretende mostrar cómo es posible salvar la falta de comunicación existente en la actualidad, debido a la ausencia de interconexión entre los sistemas informáticos dentro del sector veterinario nacional, aprovechando las capacidades de confiabilidad e inmutabilidad que ofrece una red Blockchain y el sistema de ficheros IPFS.

Petchain es una aplicación “web clásica”, que permite la gestión de identidades de mascotas domésticas. Desarrollada como una Blockchain bajo una red *Ethereum*, funcionando en un entorno local en un servidor *Ganache* y con un frontal HTML, permite la gestión y administración de la información relacionada con las mascotas, los datos de sus dueños así como la distribución de los mismos, entre los profesionales veterinarios.

**Palabras clave**: Blockchain, web, Ethereum, Ganaché, Truffle, Vue.js token, nodo, IPFS



Ilustración 1: Petchain logo

Summary

The current project is an example of how it is possible to carry out management projects with Smartcontracts within a corporate or administrative environment. The application, called **Petchain**, is inteded to show how it is possible to overcome the today’s lack of communication, due to the absence of interconnection among systems within the national veterinary sector, taking advantage of the capabilities, reliability and immutability offered by a Blockchain network and the IPFS file system.

Petchain is a "classic web" project, which allows the management of domestic pet identities. Developed as a Blockchain under an Ethereum network, working in a local environment on a *Ganache* server and with an HTML front end, it allows the management and administration of information regarding to pets, their owners' data as well as their distribution among veterinary professionals.

**Keywords**: Blockchain, web, Ethereum, Ganaché, Truffle, Vue.js, token, node, IPFS.

Indice

[Resumen 1](#_Toc17930947)

[Summary 1](#_Toc17930948)

[Índice de ilustraciones 5](#_Toc17930949)

[Índice de Tablas 6](#_Toc17930950)

[1 Introducción 8](#_Toc17930951)

[1.1 Presentación del problema 8](#_Toc17930952)

[1.2 La solución. 9](#_Toc17930953)

[1.2.1 Sistema centralizado 10](#_Toc17930954)

[1.2.2 Sistema distribuido 11](#_Toc17930955)

[1.2.3 API de servicios 12](#_Toc17930956)

[1.3 Objetivos 14](#_Toc17930957)

[1.3.1 Objetivo general 14](#_Toc17930958)

[1.3.2 Objetivos específicos 14](#_Toc17930959)

[2 Estado del arte. 16](#_Toc17930960)

[2.1 La Blockchain 16](#_Toc17930961)

[2.1.1 Blockchain públicas 16](#_Toc17930962)

[2.1.2 Blockchain privadas 17](#_Toc17930963)

[2.1.3 Blockchain de consorcio 17](#_Toc17930964)

[2.2 Características 18](#_Toc17930965)

[2.2.1 ¿Qué es un bloque? 18](#_Toc17930966)

[2.2.2 Proof of work 21](#_Toc17930967)

[2.2.3 Proof of stake 22](#_Toc17930968)

[2.3 IPFS 22](#_Toc17930969)

[2.4 DApp. ¿Qué es? 23](#_Toc17930970)

[2.5 Smartcontracts 23](#_Toc17930971)

[3 Petchain. Especificación de requisitos. 24](#_Toc17930972)

[3.1 Propósito del documento 24](#_Toc17930973)

[3.2 Alcance del documento 24](#_Toc17930974)

[3.3 Participantes en el proyecto 24](#_Toc17930975)

[3.3.1 Presentación de participantes 24](#_Toc17930976)

[3.4 Organizaciones involucradas 26](#_Toc17930977)

[3.5 Descripción del sistema 26](#_Toc17930978)

[3.5.1 Perspectiva del producto 26](#_Toc17930979)

[3.5.2 Funciones del producto 26](#_Toc17930980)

[3.5.3 Características del usuario 27](#_Toc17930981)

[3.5.4 Restricciones 27](#_Toc17930982)

[3.6 Descripción técnica del sistema 27](#_Toc17930983)

[3.6.1 Front-End 27](#_Toc17930984)

[3.6.2 Seguridad 28](#_Toc17930985)

[3.6.3 Back-End 28](#_Toc17930986)

[3.6.4 Entorno de ejecución 28](#_Toc17930987)

[3.7 Objetivos del sistema 29](#_Toc17930988)

[3.8 Catálogo de requisitos del sistema. 33](#_Toc17930989)

[3.8.1 Requisitos de información 33](#_Toc17930990)

[3.8.2 Requisitos funcionales 36](#_Toc17930991)

[3.8.3 Requisitos no funcionales 41](#_Toc17930992)

[3.9 Matriz de rastreabilidad 45](#_Toc17930993)

[3.9.1 Objetivos del sist. – Requisitos funcionales / Requisitos no funcionales 45](#_Toc17930994)

[3.10 Casos de uso 46](#_Toc17930995)

[3.10.1 Gestión de mascotas 46](#_Toc17930996)

[3.10.2 Gestión de propietarios 46](#_Toc17930997)

[3.10.3 Consulta de saldo 47](#_Toc17930998)

[3.10.4 Gestión de usuarios 47](#_Toc17930999)

[4 Metodología y Plan de trabajo 48](#_Toc17931000)

[4.1 Seguimiento del proyecto 48](#_Toc17931002)

[5 Descripción técnica. 49](#_Toc17931003)

[5.1 Arquitectura 50](#_Toc17931004)

[5.1.1 Modelo 51](#_Toc17931005)

[5.1.2 Vista 51](#_Toc17931006)

[5.1.3 Controlador 52](#_Toc17931007)

[5.1.4 Análisis de elementos: Componentes y Vistas. 52](#_Toc17931008)

[5.2 Arquitectura técnica. 55](#_Toc17931009)

[5.3 Node.js 57](#_Toc17931011)

[5.3.1 Dependencias 57](#_Toc17931012)

[5.4 Vue.js 58](#_Toc17931013)

[5.4.1 Requerimientos 59](#_Toc17931014)

[5.5 Truffle 59](#_Toc17931015)

[5.5.1 Instalación de Truffle 59](#_Toc17931016)

[5.5.2 Generación del proyecto 59](#_Toc17931017)

[5.6 Ganache 60](#_Toc17931018)

[5.6.1 Instalación de Ganache 60](#_Toc17931019)

[5.7 Web3.js 61](#_Toc17931020)

[5.7.1 Instalación de Web3.js 61](#_Toc17931021)

[5.8 Bootstrap 62](#_Toc17931022)

[5.9 Herramientas utilizadas 62](#_Toc17931023)

[5.9.1 Argo UML 62](#_Toc17931024)

[5.9.2 REM 63](#_Toc17931025)

[5.9.3 GitHub Desktop 63](#_Toc17931026)

[5.9.4 Trello 63](#_Toc17931027)

[5.9.5 Visual Studio Code 63](#_Toc17931028)

[6 Petchain. Descripción funcional 64](#_Toc17931029)

[6.1 Login 64](#_Toc17931030)

[6.2 Pantalla principal 65](#_Toc17931031)

[6.3 Alta de mascotas 65](#_Toc17931032)

[6.4 Modificación de mascotas 65](#_Toc17931033)

[6.5 Baja de mascotas 65](#_Toc17931034)

[6.6 Transferencia de ETHERS 65](#_Toc17931035)

[6.7 Pantallas de información 65](#_Toc17931036)

[7 Test de código. 66](#_Toc17931037)

[7.1 Validación de código 66](#_Toc17931040)

[7.1.1 Validación inicial (\*.html) 66](#_Toc17931041)

[7.1.2 Validación inicial (\*.css) 68](#_Toc17931042)

[7.2 Test Smartcontracts 70](#_Toc17931043)

[7.3 Validación de objetos JSON 70](#_Toc17931044)

[7.4 Visualización en diferentes dispositivos 70](#_Toc17931045)

[8 Conclusiones y trabajo futuro. 71](#_Toc17931046)

[9 Bibliografía 72](#_Toc17931047)

[10 Glosario de términos 73](#_Toc17931048)

[11 Referencias 76](#_Toc17931049)

Índice de ilustraciones

[Ilustración 1: Petchain logo 1](#_Toc17931050)

[Ilustración 2: Modelo de sistema centralizado. 10](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931051)

[Ilustración 3: Modelo de sistema distribuido. 11](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931052)

[Ilustración 4: Logotipos de Bitcoin, Ether y Monero 17](#_Toc17931053)

[Ilustración 5: Representación de la cadena de bloques. 19](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931054)

[Ilustración 6: Ejemplo de bloque genesis.json 19](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931055)

[Ilustración 7: Bloque Genesis 20](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931056)

[Ilustración 8: mempool 21](#_Toc17931057)

[Ilustración 9: Esquema figurado DApp 23](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931058)

[Ilustración 10: Gestión de mascotas 46](#_Toc17931059)

[Ilustración 11: Gestión de propietarios 47](#_Toc17931060)

[Ilustración 12: Consulta del saldo en Ethers 47](#_Toc17931061)

[Ilustración 13: Tablero principal de Petchain en Trello 48](#_Toc17931062)

[Ilustración 14: Esquema gráfico de herramientas de Petchain (Fuente: diseño propio) 49](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931063)

[Ilustración 15: Patrón MVC (Fuente: Sommerville, 2009, pag. 156) 50](#_Toc17931064)

[Ilustración 16: Estructura de los ficheros del Modelo 51](#_Toc17931065)

[Ilustración 17: Estructura de ficheros de la Vista 51](#_Toc17931066)

[Ilustración 18: Esquema de la capa controladora 52](#_Toc17931067)

[Ilustración 19: Elementos de un componente Vue.js 53](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931068)

[Ilustración 20: Ejemplo de componente simple Hello World 54](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931069)

[Ilustración 21: Flujo de composición de la vista.Contactoview.vue 54](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931070)

[Ilustración 22: Esquema de componentes. 55](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931071)

[Ilustración 23: Comando de instalación de Truffle 59](#_Toc17931072)

[Ilustración 24: Comando de inicialización de un proyecto Truffle 60](#_Toc17931073)

[Ilustración 25: Web de Truffle framework. 61](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931074)

[Ilustración 26: Login PetChain 64](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931075)

[Ilustración 27: Login incorrecto. 64](#_Toc17931076)

[Ilustración 28: Resultados W3C de la validación de index.html 67](#_Toc17931077)

[Ilustración 29: Validación W3C index.html 68](#_Toc17931078)

[Ilustración 30: Resultados W3C de main.css 69](#_Toc17931079)

[Ilustración 31: Resultado final del W3C de la validación de main.css 70](file:///C:\Users\Carolina\Google%20Drive\UNIR\GRADO%20INGENIERIA%20INFORMATICA\TFG\docs\TFG_Carlos_Escario_Bajo.docx#_Toc17931080)

Índice de Tablas

[Tabla 1: Tipos de blockchain 15](#_Toc17931081)

[Tabla 2: Dirección del proyecto 24](#_Toc17931082)

[Tabla 3: Analista y desarrollador del proyecto. 24](#_Toc17931083)

[Tabla 4: Colaborador del proyecto. 24](#_Toc17931084)

[Tabla 5: Dirección facultativa y Veterinaria 24](#_Toc17931085)

[Tabla 6: Organización 1. 25](#_Toc17931086)

[Tabla 7: Organización 2 25](#_Toc17931087)

[Tabla 8: Objetivo de Alta de mascotas 28](#_Toc17931088)

[Tabla 9: Objetivo Alta de dueños. 28](#_Toc17931089)

[Tabla 10: Objetivo de Modificación de datos 29](#_Toc17931090)

[Tabla 11: Obetivo de Baja de mascotas 29](#_Toc17931091)

[Tabla 12: Objetivo de Consulta de saldo de ETH 30](#_Toc17931092)

[Tabla 13: Objetivo de control de acceso al sistema 30](#_Toc17931093)

[Tabla 14: Objetivo de visulaización en múltiples disposivos. 31](#_Toc17931094)

[Tabla 15: Objetivo de traspaso de ETH 31](#_Toc17931095)

[Tabla 16: Objetivo de la navegación de la aplicación 31](#_Toc17931096)

[Tabla 17: Objetivo de Gestión de sesiones 32](#_Toc17931097)

[Tabla 18: Requisito de Informacion del veterinario 33](#_Toc17931098)

[Tabla 19: Requisitos de información de mascotas 33](#_Toc17931099)

[Tabla 20: Requisitos de información de propietarios. 34](#_Toc17931100)

[Tabla 21: Requisitos de información del saldo en ETH 35](#_Toc17931101)

[Tabla 22: Requisito funcional de acceso al sistema 35](#_Toc17931102)

[Tabla 23: rquisito funcional de identificación de verinarios. 36](#_Toc17931103)

[Tabla 24: Requisito funcional de traspaso de weis 36](#_Toc17931104)

[Tabla 25: Requisito funcional de navegación en el sistema. 37](#_Toc17931105)

[Tabla 26: Requisito funcional de modificación de datos de una mascota 37](#_Toc17931106)

[Tabla 27: Requisito funcional para la baja de una mascota 38](#_Toc17931107)

[Tabla 28: Requisito funcional para la visualización de weis disponibles 38](#_Toc17931108)

[Tabla 29: Requisito funcional para el alta de una mascota. 39](#_Toc17931109)

[Tabla 30: Requisito funcional para el alta de un propietario. 39](#_Toc17931110)

[Tabla 31: Requisito funcional para la modificación de un propietario. 40](#_Toc17931111)

[Tabla 32: Requisito no funcional de visualización de Petchain. 41](#_Toc17931112)

[Tabla 33: Requisito no funcional del pago en weis 41](#_Toc17931113)

[Tabla 34: Requisito no funcional del tiempo de sesión. 41](#_Toc17931114)

[Tabla 35: Requisito no funcional de usuarios autorizados 42](#_Toc17931115)

[Tabla 36: Requisito no funcional del nº de cuentas en Ganache 42](#_Toc17931116)

[Tabla 37: Requisito no funcional para la autenticación JWT 43](#_Toc17931117)

[Tabla 38: Requisito no funcional de campos obligatorios 43](#_Toc17931118)

[Tabla 39: Requisito no funcional de validación de código 43](#_Toc17931119)

[Tabla 40. Matriz de dependencias de requisitos 44](#_Toc17931120)

[Tabla 41: Relación Arquitectura - Tecnología 49](#_Toc17931121)

[Tabla 42: Comando de instalación de Vue-CLI 58](#_Toc17931122)

[Tabla 43: Instalación de web3.js 60](#_Toc17931123)

[Tabla 44: Resultados de la validación inicial de los ficheros html 65](#_Toc17931124)

[Tabla 45: Resultados del código html validado 67](#_Toc17931125)

[Tabla 46: Resultados de la validación de main.css 68](#_Toc17931126)

[Tabla 47: Resultado de la validación de main.css 68](#_Toc17931127)

[Tabla 48: Resultados validación de ficheros JSON 69](#_Toc17931128)

# Introducción

¿Por qué Blockchain? ¿Si ya tiene más de 20 años, qué trae nuevo? ¿Por qué he de fiarme de algo que se califica a sí mismo como inmutable? ¿Pero es seguro? En la era de las APIs, ¿por qué debo de fiarme de un SmartContract? Pero y eso de un SmartContract, ¿qué es? ¿Necesito cambiar las competencias del equipo de desarrollo? Y además, ¿debo invertir en criptomonedas para poder hacer mi aplicación empresarial? Etc…

Estas cuestiones y otras muchas, son las que cualquier analista, jefe de proyecto, responsable de TI o incluso de negocio que se enfrenta a un nuevo desarrollo, podría llegar a hacerse si le planteasen la necesidad de diseñar e implementar un proyecto en una blockchain corporativa.

En el presente trabajo, se pretende ofrecer la visión de cómo es posible desarrollar una aplicación de gestión basado en un nuevo paradigma de desarrollo: **la Blockchain**. Observemos que se ha dicho “nuevo”, pues no es el objetivo de este proyecto, bajo ningún concepto, querer transmitir el concepto de “mejor”. Por otro lado, se quiere dejar claro que no es necesario saber invertir en criptomonedas, o cual es la cotización actual del Bitcoin o el Ether para poder desarrollar una aplicación basada en una Blockchain.

## Presentación del problema

En la actualidad, la movilidad de las personas y las cosas han forzado que estemos viviendo en la era de la interconexión a todos los niveles, ya sea personal, empresarial o administrativa. La mayoría de los sectores, para realizar sus labores de negocio o de gestión, se ven obligados a consultar datos de otras fuentes. Las empresas y administraciones del territorio español, necesitan exponer y compartir sus datos para poder tener un control y ofrecer los servicios que los ciudadanos requieran. Es este flujo de información el que sirve de motor para gran parte sectores económicos en el mundo actual, y cuando éste se rompe, o simplemente no existe, notamos que todo se vuelve más lento e ineficiente.

Pues bien, este escenario, el de la falta de comunicación global, es el que en la actualidad se está produciendo en el sector veterinario español. Y este problema, además de los profesionales veterinarios, quien lo sufre son los usuarios del servicio, las personas y sus mascotas. ¿Pero y por qué? Pues debido a la movilidad de las personas y las familias que, junto con sus mascotas, se trasladan a otras residencias constantemente en cuestión de horas, por razones personales, de trabajo o de ocio.

En España, según datos de la web de profesionales del sector veterinario Petshopmagazine[[1]](#footnote-1), en el año 2018 había censados 5.147.000 perros y 2.265.000 gatos. Pues bien, es esta movilidad y la problemática administrativa que ella conlleva, en la que se quiere centrar el desarrollo del presente trabajo. Estos traslados familiares, ya sean ocasionales o permanentes, a nivel de mascotas también requieren de un registro y control administrativo por parte de los profesionales del sector veterinario. Éstas necesitan un control sanitario para su vacunación periódica, una atención puntual por algún tipo de enfermedad o el registro en sus nuevos domicilios. Este tipo de atención tiene como elemento angular de entrada el **Código de Identificación**, o lo que comúnmente se conoce como “**el chip**”. Este es un dispositivo electrónico, de un tamaño mayor al de un grano de arroz y compuesto por una electrónica miniaturizada encapsulada en plástico, que almacena un código alfanumérico de 15 posiciones único por cada dispositivo. Es el DNI del animal, el cual permite asociarlo al dueño y dotarle de un historial de domicilio, teléfono de contacto, historial veterinario, movimientos, etc.

La problemática surge debido a la ausencia de interconexión entre los diferentes servicios sanitarios veterinarios de las comunidades autónomas en España. Según el sistema SIACYL[[2]](#footnote-2), Sistema de Identificación Animal de Castilla y León, esta comunidad sólo tiene conexión con Castilla y La Mancha (SIACAM), Aragón (RIACA), Murcia (SIAMU) y Melilla (SIAMEL). Por ejemplo, se producen con relativa asiduidad casos como los que ocurren en alguno de los servicios veterinarios de la provincia de Ávila que, a pesar de encontrarse a escasos kilómetros de la Comunidad de Madrid, no tienen medios para localizar a los dueños de una mascota extraviada procedente de esta Comunidad mediante la consulta del chip identificativo. En estas ocasiones, es necesario realizar una llamada telefónica a los servicios veterinarios de Madrid, en los que sólo existe atención telefónica en horario laboral y de Lunes a Viernes, con lo que se dan circunstancias en las que si un animal se extravía el sábado por la mañana y se intenta localizar al dueño, esto no sea posible hasta el Lunes siguiente y sólo mediante una llamada de telefónica.

Es decir, se antoja necesario la búsqueda de una solución que permita la interconexión de los diversos sistemas sanitarios animales de las diferentes Comunidades Autónomas, con el objetivo de dar servicio tanto a las mascotas como a sus propietarios.

## La solución.

Entonces, ¿qué se puede hacer? Evidentemente, conectar los sistemas de cada una de las 17 comunidades autónomas. ¿Cómo? La respuesta a esta pregunta es, que de múltiples maneras, vamos a proponer alguna de ellas.

### Sistema centralizado

Se concentraría en único punto central los datos de veterinarios, mascotas y propietarios, con un modelo de datos común, pero con una o múltiples lógicas de negocio, en función de las necesidades de cada CCAA.



Ilustración 2: Modelo de sistema centralizado.

**Ventajas**:

* Arquitectura de fácil mantenimiento.
* Datos securizados en un único punto de control.
* Desarrollo único de front-end y backend.
* Modelo de mantenimiento y soporte sencillo para la gestión de actualizaciones y cambios.
* Sistemas locales ligeros: “*Thin client*”.
* No se hace necesario el desarrollo de ninguna clase de middleware para comunicarse entre las diferentes CCAA.

**Inconvenientes:**

* Si el sistema central falla, toda la red deja de funcionar.
* Si el sistema central no está correctamente dimensionado, se puede provocar el efecto de “cuello de botella”.
* Número elevado de “puntos débiles” de acceso al sistema para usuarios malintencionados.
* En el contexto actual, es necesario una lógica de negocio diferente para cada CCAA, alojada en el sistema central, con el consiguiente aumento en la complejidad de mantenimiento.
* Sistemas con alta obsolescencia.

Con esta arquitectura se propone que las clínicas veterinarias se conecten a través de un cliente remoto a un servidor central, el cual serviría tanto la interface, la lógica de negocio y los datos. La dificultad de este modelo, además de las señaladas en las desventajas, es el riesgo que conllevaría una caída del nodo central, para todo el sistema a nivel nacional, y su coste asociado.

### Sistema distribuido

En la web [*https://sistemasdistribuidos.mforos.com*](https://sistemasdistribuidos.mforos.com/) expresa la correcta definición de los que es un sistema distribuido:

*Un sistema distribuido se define como una colección de computadoras autónomas conectadas por una red. Los sistemas distribuidos comparten un estado y ofrecen una visión de sistema único, que muestran los recursos de manera homogénea, ocultando su distribución.*



Ilustración 3: Modelo de sistema distribuido.

Los sistemas distribuidos poseen una serie de características que conviene señalar previamente:

* **Concurrencia**: permita a los equipos participantes en la red el hacer uso de los recursos de manera transparente y simultánea, independientemente de la localización.
* **Tolerancia a fallos**: el fallo en uno de los componentes del sistema no afecta al funcionamiento de la red en su conjunto, pues su funcionalidad puede ser suplida por otro componente.
* **Escalabilidad**: capacidad de aumentar el número de componentes de la red sin que ello vaya en detrimento de su rendimiento.

**Ventajas**

* Mayor capacidad de procesamiento en los equipos cliente.
* Interfaces de usuario más vistosas y amigables.
* Mayor seguridad en el sistema.
* Permite el funcionamiento de diferentes Sistemas Operativos y aplicaciones diferentes en cada uno de los equipos cliente.
* Alta tolerancia a fallos.

**Inconvenientes**

* Sistemas más complejos.
* Mantenimiento y soporte menos ágil y más caro.
* Proceso de actualización del software más arduo.
* Aumento de costes, por la necesidad de crear un middleware que permita la interconexión de diferentes sistemas.

En el contexto actual este es el modelo que se está empleando dentro del sector en estudio. El problema radica que el mapa representado en la Ilustración 3, no es exacto ni refleja la situación actual, dado que la interconexión global entre las diferentes CCAA no es algo que en este momento se esté produciendo, con lo que la percepción de un sistema único no es real.

¿Esta problemática podría resolverse? La respuesta es SI. ¿Cómo? Por ejemplo, con la creación de una, o muchas capas intermedias (middleware) que permitiesen la conexión de cada uno de los sistemas, y con ello el acceso a la información. Un ejemplo de ello podría ser, como se describe a continuación, a través de una capa o API de servicios.

### API de servicios

Ésta nos va a permitir el intercambio de información entre diferentes servicios web y una aplicación, usando para ello el protocolo http o https y empleando los dos formatos más extendidos: XML o JSON.

Normalmente entre los desarrolladores vamos a encontrar cuatro tipos de API para servicios web:

* **XML-RPC** (*Extensible Markup Language – Remote Procedure Call*): protocolo que permite la invocación de procedimientos o funciones alojadas en un sistema remoto. Con este mecanismo los desarrolladores pueden definir interfaces que pueden ser invocadas a través de la red.
  + Emplea protocolo http para la transmisión de información desde el cliente hasta el equipo servidor.
  + Especifica las llamadas a las funciones o procedimientos a través de XML.
  + Emplea un “vocabulario XML” muy limitado para realizar las llamadas (*requests*) y recibir las respuestas (*responses*).
  + El cliente XML-RPC especifica unos parámetros en la llamada y recibe del servidor una respuesta en formato XML o un error, en caso de producirse.
  + Las primeras definiciones del protocolo proceden del año 1998.
* **SOAP** (*Simple Object Access Protocol*): protocolo estándar para el intercambio de mensajes en formato XML entre dos objetos y diseñado para las comunicaciones vía Internet.
  + Es una aplicación de la especificación XML.
  + Permite el transporte de datos por los servicios web.
  + No está acoplado a ninguna plataforma o lenguaje de desarrollo.
  + Permite a las aplicaciones cliente conectarse de manera sencilla a servicios remotos e invocar métodos remotos.
* **JSON-RPC** (*Javascript Object Notation- Remote Procedure Call)*: concepto similar al XML-RPC, para la llamada a procedimientos o funciones remotas, pero realizados en formato JSON.
  + Emplea el protocolo http para la transmisión de información.
  + Paso de nombre del procedimiento o función y de los parámetros necesarios desde el cliente en un objeto con formato JSON y respuesta desde el servidor en un objeto con el mismo formato.
* **RESTful** (*Representational State Transfer*): Son servicios web basados en arquitectura REST.
  + Los servicios web RESTful se utilizan para la creación de APIs de aplicaciones basadas en web.
  + Se implementan sobre protocolo http.
  + Un servicio web RESTful define un URI, Uniform Resource Identifier un servicio, proporciona representación de recursos como JSON y un conjunto de métodos http (GET, POST, PUT, DELETE).
  + Independiente de la plataforma, sistema operativo y lenguaje de programación.

Con estos cuatro modelos de APIs podremos desarrollar diferentes capas de abstracción que permitan la interconexión de cada uno de los sistemas pertenecientes a cada comunidad autónoma. Los inconvenientes que conllevaría el desarrollo de una capa intermedia serían:

* Tiempos de desarrollo elevados, debido a la gran variedad de sistemas existentes y que son necesarios interconectar.
* Altos costes de implementación, motivados por el razonamiento anterior.

¿Cuál es entonces la propuesta que se pretende ofrecer? Se quiere mostrar como la BLOCKCHAIN funcionando bajo el rol de backend, interactuando con un frontal web y almacenando la información en la IPFS, se convierte en una alternativa viable y de bajo coste para el desarrollo de aplicaciones descentralizadas. El desarrollo de este modelo es lo que nos va a ocupar en el presente trabajo y sobre el cual vamos a desgranar sus características en capítulos posteriores.

## Objetivos

Se proponen una serie de objetivos que se pretenden conseguir al finalizar el desarrollo del presente trabajo, que se enumeran a continuación y que están ordenados en base a su importancia.

### Objetivo general

Como se ha expuesto con anterioridad, la problemática actual en el caso de la localización de las mascotas en el ecosistema de las Comunidades Autónomas, por la falta de conexión entre los diferentes entornos, se antoja complicada. Se pretende entonces mostrar cómo el modelo que se va a desarrollar permite salvar esta “trinchera”, unificando en un solo sistema toda la gestión de las identidades de las mascotas.

### Objetivos específicos

1. **Objetivo 1**: Demostrar que la Blockchain y la IPFS son entornos viables al 100%, para el desarrollo de aplicaciones de gestión. No es objeto del presente trabajo, ni del proyecto asociado, el querer demostrar el concepto de “mejor”, si no el de “nuevo”, es decir, que es posible una nueva alternativa al sistema de APIs, que aporte las características de descentralización e inmutabilidad.
2. **Objetivo 2:** Web usable, accesible y atractiva. Estos tres criterios, a juicio del autor, deberían ser parte de cualquier desarrollo web. Para los dos primeros se van a establecer criterios dentro de los requisitos no funcionales que, obviamente, deberán estar cumplidos al finalizar el desarrollo. Para el tercero se seguirá el principio de “menos es más” a la hora de abordar el diseño para poder lograrlo.
3. **Objetivo 3**: Software coste cero. Se pretende hacer un desarrollo con software de uso libre, que no implique desembolso alguno a las personas o entidades que lo aborden. Esto incluye IDEs de desarrollo, frameworks, iconos, entornos de alojamiento, herramientas de validación, servidores y entornos de despliegue.
4. **Objetivo 5**: No hace falta ser un experto en criptodivisas. Demostrar que para el desarrollo de aplicaciones, y más en concreto de una DApp, no es necesario ser un experto conocedor del mundo Bitcoin o Ether, ni dominar la cotización de ambas en el mercado de las criptodivisas.
5. **Objetivo 6**:Aumentar las competencias del autoren la programación de lenguajes como Javascript en entornos Node.js, Solidity para el desarrollo de Smartcontracts y el manejo del framework Web3JS para la combinación de todas estas tecnologías.

# Estado del arte.

Para comprender el entorno en el que nos pretendemos desenvolver, es necesario comprender de manera general los elementos la componen, haciendo hincapié en aquellos que resultan “novedosos” en el tipo de desarrollo que se pretende mostrar. Por lo tanto es necesario tener una visión general de:

* Blockchain.
* IPFS (Inter Planetary File System).
* DApp.
* Smartcontracts.

## La Blockchain

¿Pero qué es Blockchain? La respuesta es, una red P2P (*peer-to-peer*) en la que se crean transacciones digitales, y que funciona con el mismo concepto que el de un libro mayor de contabilidad o *ledger*. En esta red la información se almacena en bloques, en donde cada uno de ellos contiene información relativa al bloque anterior, lo que permite ir formado una cadena con información accesible pero inmutable.

En la actualidad la blockchain ha extendido su uso a muchos sectores económicos, por lo que ahora estamos en disposición de realizar una clasificación, según su uso:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tipos de Blockchain** | | | | | |
| **Publicas** | | **Privadas** | | **De consorcio** | |
| Bitcoin | Sin permisos.  Identidades anónimas | Hyperledger | Con permisos.  Identidades conocidas | B3i (seguros) | Con permisos.  Identidades conocidas |
| Ethereum | Fabric | Alastria |
| Monero | Ripple |

Tabla 1: Tipos de blockchain

### Blockchain públicas

Este modelo es el primero que surgió e Internet ha sido el entorno en donde se ha desarrollado hasta alcanzar la popularidad con la que ahora cuenta. Los tipos que podemos destacar por su popularidad serían Bitcoin, Ethereum y Monero. Cualquier persona, sin necesidad de permisos de acceso puede formar parte de cualquiera de ellas, tanto para realizar transacciones de carácter privado como la compra de bienes y servicios que admitan como moneda de pago bitcoins, ethers o moneros o también para el “minado” de bloques con el que conseguir un rédito económico.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Ilustración 4: Logotipos de Bitcoin, Ether y Monero

### Blockchain privadas

Tanto este modelo como el de consorcio se podrían considerar como la segunda generación de blockchain. El modelo privado consiste en iniciativas dirigidas por una organización, en donde el acceso a la red si está controlado por una entidad, y las identidades de los participantes si son conocidas. En la actualidad el entorno más extendido para la generación de una blockchain de carácter privado sería el Hyperledger® [[3]](#footnote-3). Este proyecto está liderado por la Fundación Linux, de la cual son miembros compañías como IBM®, SAP®, Oracle®, Google® o Intel®, las cuales aportan al mismo su apoyo económico y técnico.

En este entorno las compañías que lo deseen pueden desarrollar su proyecto basado en una blockchain privada con sus reglas de negocio y objetivos propios, sin la necesidad de realizar de manera obligatoria transacciones con criptomonedas.

### Blockchain de consorcio

Este modelo se puede definir como una combinación de los dos anteriores, pero con la característica de que está dirigido a sectores económicos específicos o a grupos de compañías con intereses comunes.

Los ejemplos que se pueden dar serían:

* Consorcio B3i[[4]](#footnote-4): Esta es una iniciativa creada por un consorcio de empresas de seguros, y que tiene como objetivo la creación de estándares y protocolos entre ellas con el objeto de disminuir los costes operativos derivados de los necesarios intercambios de información entre ellas.
* Alastria[[5]](#footnote-5): La definición que indica en su web nos da un idea muy clara de su objetivo: “Alastria es una asociación sin ánimo de lucro que fomenta la Economía Digital mediante el desarrollo de Blockchain.” (Alastria-Presentación corporativa.pdf, p. 1).

Este es un proyecto abierto a cualquier empresa, independientemente de su tamaño, que desee desarrollar proyectos dentro de un modelo de economía digital descentralizada. Para participar en el proyecto es necesario ser miembro del consorcio, y en la actualidad sólo está abierto a personas jurídicas.

## Características

Esta “cadena de bloques” posee las siguientes características:

* **Descentralizada** (*Trustless*): Todos los nodos son responsables de la gestión y modificación de la información al mismo tiempo. Esto significa que cualquier servicio o aplicación seguirá funcionando si un nodo o grupo de nodos deja de estar disponible, en este caso la información seguirá fluyendo al persistir en el resto de nodos de la red.
* **Distribuida** (*Many to many*)**:** Cualquier nodo perteneciente a la blockchain está conectado al resto de nodos de la red. Con la aplicación estricta de este concepto, en una Blockchain se hace casi imposible manipular los datos almacenados en ella, dado que todos los nodos tiene una copia de los mismos y los cambios no permitidos podrían ser rechazados al ser dados como *no válidos*.
* **Consistencia de datos** (*Proof of work*)**:** Tanto en **Bitcoin** como en **Ethereum** los nodos pertenecientes a la red verifican la validez de cada transacción realizada en ella, a través de un **algoritmo de consenso**. En el caso de estas dos redes, este algoritmo se denomina ***Proof of Work***, aunque en la actualidad la red Ethereum está comenzando a migrar al algoritmo ***Proof of Stake***. En capítulos posteriores profundizaremos en estos dos conceptos.
* **Inmutable** (*Ledger*)**:** Los datos almacenados en la red no pueden ser eliminados. Podrán ser modificados en posteriores operaciones, pero siempre existirá un registro histórico con todos los datos a través de la cadena de bloques.

### ¿Qué es un bloque?

Según se define en <https://www.sinergiablockchain.org/_/Doc/FAQ.pdf>:

*“Un bloque es un conjunto de transacciones confirmadas e información adicional que se ha incluido en la cadena de bloques. Cada bloque que forma parte de la cadena (excepto el bloque generatriz, que inicia la cadena) está formado por:*

* *Un código alfanumérico que enlaza con el bloque anterior.*
* *El “paquete” de transacciones que se incluye.*
* *Otro código alfanumérico que enlazará con el siguiente bloque.”*

La imagen siguiente ilustración muestra el concepto de una “cadena de bloques”.

Ilustración 5: Representación de la cadena de bloques.

**Block Hash: 0000078ecba1500**

**Previous Block Hash: 0000096effa4600**

**Block Hash: 0000011aee0534e**

**Previous Block Hash: 0000078ecba1500**

**Block Hash: 000056abff1189a**

**Previous Block Hash: 0000011aee0534e**

**Transaction Hash: 000aa76effa4133**

**Transaction Hash: aaf4000faee3a00**

**Bloque 76**

**Bloque 77**

**Bloque 78**

Pero como en todo siempre hay un principio, y la blockchain no es una excepción, existe un bloque generatriz denominado **Génesis**, a partir del cual se encadenan el resto.

Ilustración 6: Ejemplo de bloque genesis.json

|  |
| --- |
| "config": { |

|  |
| --- |
| "chainId": 0, |

|  |
| --- |
| "homesteadBlock": 0, |

|  |
| --- |
| "eip155Block": 0, |

|  |
| --- |
| "eip158Block": 0 |

|  |
| --- |
| }, |

|  |
| --- |
| "alloc"   : {}, |

|  |
| --- |
| "coinbase"   : "0x0000000000000000000000000000000000000000", |

|  |
| --- |
| "difficulty" : "0x20000", |

|  |
| --- |
| "extraData"  : "", |

|  |
| --- |
| "gasLimit"   : "0x2fefd8", |

|  |
| --- |
| "nonce"   : "0x0000000000000042", |

|  |
| --- |
| "mixhash" : "0x0000000000000000000000000000000000000000000000000000000000000000", |

|  |
| --- |
| "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000", |

|  |
| --- |
| "timestamp"  : "0x00" |

|  |
| --- |
| } |

Este bloque se forma en el momento de la creación de la Blockchain, y su estructura está definida en un fichero en formato JSON que recibe el nombre de *genesis.json*.

**Block Hash: 00000aa11ba1500**

**Previous Block Hash: 0x0000**

**Block Hash: 0000011aee0534e**

**Previous Block Hash: 00000aa11ba1500**

**Transaction Hash: 000aa76effa4133**

**Transaction Hash: aaf4000faee3a00**

**Bloque 0**

**Bloque 1**

…

Ilustración 7: Bloque Genesis

Como se observa en las imágenes anteriores, los identificadores de los bloques son códigos generados por funciones resumen o *funciones hash*. Estos identificadores son los elementos que sirven de enlaces entre los bloques. En cada uno de ellos hay un “Previous Block HASH” que permiten enlazar un bloque con el anterior. Además se almacena el “Transaction HASH” que contiene la transacción realizada y validada.

Los bloques se sitúan en los NODOS, y estos no son más que cada una de las máquinas que componen la red, y en donde existe una copia de todas las transacciones realizadas hasta el momento. Cada bloque se añade a la blockchain de manera cronológica y con una marca de tiempo o *timestamp*. Cada vez que una transacción es validada, esta se almacena de manera permanente y sin posibilidad de ser modificada o eliminada.

Ahora bien, ¿quién o que valida cada transacción? Cuando a la blockchain le llega una solicitud para realizar una transacción, realiza un proceso de *consenso,* éste es un acuerdo entre los nodos participantes en la red, en el que se decide que la información que se ha almacenado es cierta, y por lo tanto puede pasar a formar parte de la cadena de bloques. Estos acuerdos entre bloques se alcanzan a través de los *algoritmos de consenso,* éstos obligan a que el protocolo de la blockchain sea respetado, y por consiguiente dan fiabilidad a cualquier transacción que se realice.

Los encargados de ejecutar estos algoritmos, son los nodos mineros o *miners*, y ellos gastan energía y tiempo de cómputo para validar la transacción.

¿Qué algoritmos de consenso existen? La respuesta es que bastantes, pero centrándonos en la red Bitcoin y Ethereum, éstas dos emplean los denominados *Prueba de trabajo* (Proof of work) y *Prueba de participación* (Proof of stake). En la actualidad *Proof of work* es el algoritmo más utilizado, pero la red Ethereum está migrando a *Proof of stake* debido al alto consumo de energía al que obliga el primero.

### Proof of work

Fue el primer algoritmo de consenso creado para la red Bitcoin, y aquí los mineros realizan la labor de validación resolviendo un problema matemático. Es en este punto donde entra la potencia de cómputo de cada minero participante. Se establece una competición para lograr resolver el problema antes que ninguno, pues aquel que lo logre será el que consiga la recompensa en forma de criptomoneda, *bitcoins* para la red Bitcoin o *ethers* para la red Ethereum.

Como se ha dicho anteriormente, este algoritmo trata de resolver un problema matemático, que permita la validación del bloque. El proceso que sigue este algoritmo es el siguiente:

1. Existe un área de espera denominada *mempool*, en donde se alojan todas las transacciones pendientes de ser validadas por los mineros. Cada transacción lleva asociado un HASH y un *fee* o tasa, que el creador de la transacción debe pagar para que ésta sea reflejada en la cadena de bloques y que será cobrada por el minero que resuelva el problema.

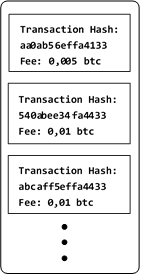


Ilustración 8: mempool

1. Cada minero agrupa transacciones, según un criterio propio, dentro de lo que se denomina un bloque candidato.
2. A cada bloque candidato se le asigna una dificultad, en nuestro caso podemos suponer que es cuatro. Esto significará que el cifrado final correspondiente al bloque debe de contener al menos cuatro ceros al inicio de su secuencia.
3. Para realizar la labor de minado es necesario utilizar un valor aleatorio, que se usa una sola vez, denominado *nonce* y que se añade al final del bloque. Con este valor y el algoritmo SHA256, se cifra el bloque y se halla un valor alfanumérico hexadecimal, por ejemplo: 454aa1487cc8b3137859531f98e999289269490579d495d5cb17680145ce0d52.
4. Como la dificultad del bloque está establecida en cuatro, si el valor obtenido no tiene cuatro ceros por delante, se considera inválido, por lo que es necesario emplear otro nonce y realizar de nuevo el cifrado.
5. Es decir, hay que realizar este proceso hasta encontrar un valor válido, por ejemplo: 0000e011db58748cfb2e44b007ec91bb1a0f0073dab58e5ddb4c013a8b5932cc.
6. Entonces, el objetivo del minero será el de encontrar el valor del nonce que proporcionará en HASH correcto antes que el resto, y con ello llevarse la recompensa y las tasas pagadas por el emisor de cada transacción. Como es obvio, la dificultad de encontrar un valor que satisfaga esta condición es muy alta, con lo que en este punto es donde entra la potencia de cómputo de cada minero, para encontrar la solución en primera posición.
7. El bloque hallado será añadido a la blockchain y el proceso se reiniciará por parte de todos los mineros a partir de las transacciones almacenadas en la mempool.

### Proof of stake

Este es el otro algoritmo de consenso, que a diferencia del proof of work, lo que pretende es premiar la participación en la red, en lugar de la potencia de cálculo. Es de reciente creación, de hecho surgió en el año 2011 y su primera implementación fue para la criptomoneda Peercoin en el año 2012.

En este sistema, la probabilidad de ser elegido para validar un bloque, y por lo tanto para recibir la recompensa, es un valor aleatorio pero que en cierta medida viene determinado por el porcentaje de monedas que cada minero posee dentro de la red, por lo tanto si un minero tiene el 0,05% de monedas, tendrá 0,05% posibilidades de ser elegido para validar el bloque. La prueba de validación pretende evitar el alto consumo de energía de PoW, así como la centralización de recursos de hardware por parte de un número reducido de mineros.

## IPFS

La definición más ortodoxa que podemos encontrar sobre este sistema de almacenamiento es la que se ofrece en el Libro Blanco de la web <https://ipfs.io/>:

“The Inter-Planetary File System (IPFS) is a peer-to-peer distributed file system, that seeks to connect all computing devices with the same system of files.” (Juan Benet, 2014, p.1).

Se define como un protocolo en el que cada fichero almacenado se identifica con un hash único, denominado *hash criptográfico*, esto impide la duplicación de los ficheros en la red. Cuando se busca un fichero lo que se está localizando es el nodo que se identifica detrás de este hash.

## DApp. ¿Qué es?

DApp es el acrónimo en inglés de Decentralized Application La definición más consensuada, que podemos encontrar es que una DApp es una aplicación que se ejecuta en una red peer-to-peer (P2P) y que aprovecha la capacidad de los Smartcontracts para realizar transacciones de carácter inmutable y confiable.

Esta capacidad de aprovechar las funcionalidades de los “contratos inteligentes” es la que ha permitido su extensión en el ámbito de la red Ethereum, pues en ésta blockchain es donde han alcanzado su máxima implantación.

Poseen una serie de características de carácter general que se enumeran a continuación:

* Son proyectos de código abierto.
* Al ser aplicaciones descentralizadas, no hay un gestor al cargo.
* Utilizan la estructura física y lógica de una blockchain pública o privada para poder funcionar.
* Todas las transacciones son inmutables y no es posible su falsificación.
* Son un formato de muy reciente generación, con lo que su estructura y formato están en constante evolución.
* No hay un límite de usuarios.



Ilustración 9: Esquema figurado DApp

## Smartcontracts

# Petchain. Especificación de requisitos.

Petchain es una DApp destinada al sector veterinario que permite la gestión de las identidades de mascotas domésticas, así como la de sus propietarios. La aplicación se basa en un desarrollo web tradicional, pero con la característica fundamental de estar implementada en una Blockchain bajo una red Ethereum y almacenado las identidades de las mascotas bajo ficheros JSON que son enviados al sistema IPFS.

Petchain gestiona de las identidades de las mascotas domésticas dando de alta, modificando los datos y gestionando la baja de los datos de los animales, bajo la ejecución de Smartcontracts creados en lenguaje Solidity.

## Propósito del documento

El presente documento pretende realizar una descripción de los requisitos informativos, funcionales y no funcionales de Petchain, con el fin de comprender y describir su funcionamiento, así como la enumeración de los participantes en el desarrollo, de cada una de sus partes y conseguir al final del mismo, una relación entre las partes y los creadores de las mismas.

Pretende ser una descripción de los requisitos informativos, funcionales, no funcionales del proyecto, con el objetivo de ofrecer a los lectores del documento del TFG una visión clara de la aplicación y las partes que lo componen.

## Alcance del documento

El presente documento tiene como alcance especificar:

* La declaración de los participantes en el proyecto.
* Definir de los objetivos de la aplicación.
* Proveer una descripción funcional del sistema que permita la mejor comprensión del mismo al equipo de evaluadores del proyecto TFG.
* Especificar de los requisitos informativos, funcionales y no funcionales que deberá cumplir la aplicación
* Establecer los límites de funcionales de Petchain.

## Participantes en el proyecto

### Presentación de participantes

|  |  |
| --- | --- |
| **Participante** | **Miriam Martínez Muñoz** |
| **Organización** | Universidad Internacional de la Rioja |
| **Rol** | Dirección de proyecto |
| **Es desarrollador** | Sí |
| **Es cliente** | No |
| **Es usuario** | No |
| **Comentarios** | Directora del proyecto de fin de grado. |

Tabla 2: Dirección del proyecto

|  |  |
| --- | --- |
| **Participante** | Carlos Escario Bajo |
| **Organización** | Universidad Internacional de la Rioja |
| **Rol** | Analista-Desarrollador |
| **Es desarrollador** | Sí |
| **Es cliente** | No |
| **Es usuario** | No |
| **Comentarios** | Ninguno |

Tabla 3: Analista y desarrollador del proyecto.

|  |  |
| --- | --- |
| **Participante** | **Pedro Escario Bajo** |
| **Organización** | Veterinaria García-Ochoa |
| **Rol** | Usuario |
| **Es desarrollador** | No |
| **Es cliente** | No |
| **Es usuario** | Sí |
| **Comentarios** | Trabajador en la Veterinaria García-Ochoa |

Tabla 4: Colaborador del proyecto.

|  |  |
| --- | --- |
| **Participante** | **Iciar García-Ochoa** |
| **Organización** | Veterinaria García-Ochoa |
| **Rol** | Usuario |
| **Es desarrollador** | No |
| **Es cliente** | No |
| **Es usuario** | Sí |
| **Comentarios** | Propietaria de la Veterinaria García-Ochoa |

Tabla 5: Dirección facultativa y Veterinaria

## Organizaciones involucradas

|  |  |
| --- | --- |
| **Organización** | **Universidad Internacional de la Rioja** |
| **Dirección** | Av. de la Paz, 137, 26006 Logroño, La Rioja |
| **Teléfono** | N/A |
| **Fax** | N/A |
| **Comentarios** | [www.unir.net](http://www.unir.net/) |

Tabla 6: Organización 1.

|  |  |
| --- | --- |
| **Organización** | **Veterinaria García-Ochoa** |
| **Dirección** | Carretera de Casillas 6 local 3, 05420 Sotillo de la Adrada, Ávila |
| **Teléfono** | N/A |
| **Fax** | N/A |
| **Comentarios** | Ninguno |

Tabla 7: Organización 2

## Descripción del sistema

### Perspectiva del producto

Petchain tiene como objeto mostrar las capacidades que posee la Blockchain para el desarrollo de aplicaciones de gestión tradicionales. No es objeto de la misma ofrecer la visión de que el modelo que se presenta "es mejor" que cualquier desarrollo web tradicional, sino que como nuevo paradigma de desarrollo, es necesario tenerlo en cuenta como un futuro modelo para la generación de aplicaciones de gestión corporativas.

Su ejecución se realiza íntegramente en un entorno local, permitiendo este modelo, la visualización de cada una de las partes que componen el desarrollo. Esto es posible gracias al uso del framework Truffle, que con el servidor Ganache que lleva incorporado, ofrece la posiblidad de crear una red Ethereum y con ello poder desarrollar, desplegar y testear Smartcontracts desarrollados con lenguaje Solidity. Por último, el almacenamiento de los objetos JSON generados se realiza bajo el protocolo IPFS.

### Funciones del producto

Petchain es una aplicación que debe de permitir a los veterinarios:

* Identificar a las mascotas domesticas mediante un código alfanumérico único, así como asociarlas a una persona que actuará como dueño, y que estará identificado mediante DNI o Pasaporte y los datos personales (nombre, apellidos, domicilio, y teléfono de contacto).
* Modificar los datos sanitarios de vacunaciones,
* Modificar los datos personales del dueño de la mascota.
* Dar de baja a las mascotas del sistema.
* Mostrar un listado de mascotas.
* Visualizar el saldo en ethers (ETH) y weis de cada uno de los veterinarios correspondientes a las transacciones realizadas.

### Características del usuario

Los usuarios de Petchain serán los veterinarios, que a nivel nacional, deben de realizar acciones de tipo administrativo a la hora de gestionar las identificaciones de animales de compañía.

Como perfil tecnológico, deben tener competencias a la hora de manejar formularios web, sin necesitar unos conocimientos técnicos profundos, pues el modelo visual no difiere de cualquier página web existente en el mercado.

Por otro lado el tipo de información que es necesaria cumplimentar en la web, está dentro del dominio del sector veterinario, con lo que su cumplimentación no debería presentar problemas para el colectivo que usará la aplicación.

### Restricciones

La visualización de Petchain debe ser óptima en toda clase de dispositivos, por lo tanto la estructura de las pantallas se debe de adaptar a la resolución de la pantalla en donde se ejecute, siendo el rango de estas desde dispositivos Smartphone, Tablets y PCs.

A nivel de seguridad, es necesario implementar características que permitan la autenticación y la autorización de los usuarios que intenten acceder a la aplicación, pero al ser una ejecución en un entorno local no es obligatorio el diseño de una base de datos SQL o NoSQL, que albergue sus credenciales de acceso.

## Descripción técnica del sistema

### Front-End

La web se desarrollará bajo el modelo de visualización responsive, por lo que su diseño se deberá adaptar al tipo de resolución de la pantalla desde la que se esté accediendo. Para conseguir este objetivo se empleará en su diseño el framework open source Twitter Bootstrap, ver. 4.3.1.

El desarrollo de las interfaces de usuario se realizará con HTML5 estándar, CSS3 y la programación de los scripts de cliente se realizará con JavaScript (ECMA Script 6) y el framework de cliente jQuery versión 3.1.1., siendo uno de los objetivos del desarrollo el de aprovechar al máximo el paradigma de programación MVC (Model View Controller).

### Seguridad

Como antes se indicaba, es necesario implementar un sistema que verifique la autenticación de los usuarios al sistema, así como la autorización para la realización de acciones asociadas al rol que tenga determinado. El sistema de credenciales empleado para el proyecto actual debe quedar preparado para poder implementar una conexión a una base de datos NoSQL en el caso de que Petchain se instale en un entono productivo. En el modelo de ejecución actual, sólo se exigirá que los usuarios estén definidos en un fichero JSON almacenado dentro de la estructura local de proyecto.

Aunque no se exija el almacenamiento de credenciales en una base de datos, si es necesario que la aplicación tenga implementados sistemas de protección de URL mediante JWT, así como la validación de usuarios mediante nombre de usuario y contraseña.

### Back-End

La aplicación se ejecutará en una Blockchain desplegada en un entorno local. La red estará funcionando desplegada en un servidor Ganache, el cual es un componente del framework de desarrollo y ejecución Truffle.

La conexión entre la capa de Back-End y el Front-End es posible mediante el uso del framework Web3.js. Este entorno, desarrollado íntegramente en Javascript, permite la interacción de Smartcontracts con el JavaScript alojado en el cliente y la ejecución de la lógica en ellos definida.

Por último la interacción de ambas capas permitirá la creación y modificación de uno o varios ficheros JSON, los cuales harán como soporte No SQL, el cual se implementará como un fichero físico en IPFS.

### Entorno de ejecución

La aplicación debe de ejecutarse en cualquier navegador estándar (IE11, Chrome o Firefox) en un servidor local o localhost.

Para la ejecución de funcionalidades como la seguridad, lectura y escritura de ficheros y paso de mensajes entre pantallas, es necesario la realización de solicitudes POST y GET, por lo la aplicación se va a desplegar con un servidor ExpressJS y bajo un entorno de ejecución Node.js.

## Objetivos del sistema

|  |  |
| --- | --- |
| **OBJ-0001** | **Alta de mascotas** |
| **Versión** | 1.0 (04/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo |
| **Descripción** | El sistema deberá permitir el alta de una mascota en el sistema, asociándola a un código de chip. Este es un identificador con longitud fija de quince (15) caracteres alfanumérico. |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 8: Objetivo de Alta de mascotas

|  |  |
| --- | --- |
| **OBJ-0002** | **Alta de dueños de mascotas** |
| **Versión** | 1.0 (04/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo |
| **Descripción** | El sistema deberá permitir el dar de alta a una persona, asociándola a una mascota. |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 9: Objetivo Alta de dueños.

|  |  |
| --- | --- |
| **OBJ-0005** | **Modificación de datos** |
| **Versión** | 1.0 (04/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo |
| **Descripción** | El sistema deberá permitir modificar los datos asociados a las mascotas y a sus dueños. el veterinario modificador deberá pagar un canon fijo en weis al veterinario identificador. |
| **Sub-objetivos** | * [OBJ-0006] Modificación de datos de mascotas: El sistema deberá permitir la modificación de los datos sanitarios de las mascotas, así como la inclusión de las vacunaciones periódicas obligatorias. * [OBJ-0007] Modificación de datos de los dueños: El sistema deberá permitir cambiar los datos correspondientes al dueño de la mascota, incluido el cambio a un nuevo propietario. |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 10: Objetivo de Modificación de datos

|  |  |
| --- | --- |
| **OBJ-0008** | **Baja de mascotas** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa |
| **Descripción** | El sistema deberá permitir dar de baja a una mascota del sistema, por motivos de defunción. |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 11: Obetivo de Baja de mascotas

|  |  |
| --- | --- |
| **OBJ-0009** | **Consulta de saldo** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Descripción** | El sistema deberá permitir consultar el saldo en ethers o weis que cada veterinario participante en el sistema, tiene por realizar acciones dentro del mismo. |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Por definir |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 12: Objetivo de Consulta de saldo de ETH

|  |  |
| --- | --- |
| **OBJ-00011** | **Control de acceso** |
| **Versión** | 1.0 (05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Descripción** | El sistema deberá permitir el acceso al sistema a aquellos usuarios que estén incluidos en el fichero de usuarios creado |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 13: Objetivo de control de acceso al sistema

|  |  |
| --- | --- |
| **OBJ-00012** | **Visualización de múltiples dispositivos** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Descripción** | El sistema deberá estar diseñado para poder visualizarse en dispositivos del tipo Smartphone, Tablet y PC |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Importante |
| **Urgencia** | Puede esperar |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 14: Objetivo de visulaización en múltiples disposivos.

|  |  |
| --- | --- |
| **OBJ-00013** | **Traspasar ETH** |
| **Versión** | 1.0 ( 05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Descripción** | El sistema deberá traspasar ETH desde el veterinario modificador hasta el veterinario identificador, siempre y cuando el modificador realice un cambio en los datos de una mascota que ya exista en el sistema. |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 15: Objetivo de traspaso de ETH

|  |  |
| --- | --- |
| **OBJ-00014** | **Navegación por la aplicación** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Descripción** | El sistema deberá permitir la navegación entre las opciones de alta, modificación y baja de mascotas, así como la consulta del saldo de weis. |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 16: Objetivo de la navegación de la aplicación

|  |  |
| --- | --- |
| **OBJ-00015** | **Gestión de sesiones** |
| **Versión** | 1.0 (05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Descripción** | El sistema deberá ser capaz de manejar las sesiones de usuario y el correcto funcionamiento de los navegadores. |
| **Sub-objetivos** | Ninguno |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Estado |
| **Comentarios** | Ninguno |

Tabla 17: Objetivo de Gestión de sesiones

## Catálogo de requisitos del sistema.

### Requisitos de información

Los requisitos de información son formas especializadas de requisitos que van a permitir la introducción de datos relativos a los actores que van a participar o componer el sistema.

|  |  |  |
| --- | --- | --- |
| **IRQ-0001** | **Datos de usuario veterinario** | |
| **Versión** | 1.0 (05/05/2019) | |
| **Autores** | * Carlos Escario Bajo | |
| **Fuentes** | * Iciar García-Ochoa | |
| **Dependencias** | * [FRQ-0001] Acceso al sistema | |
| **Descripción** | El sistema deberá almacenar la información correspondiente a los usuarios veterinarios que usen el sistema. En concreto: | |
| **Datos específicos** | * Nombre de usuario * Contraseña del usuario * Rol * Número de colegiado * Colegio | |
| **Tiempo de vida** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Ocurrencias simultáneas** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Importancia** | Vital | |
| **Urgencia** | Inmediatamente | |
| **Estado** | Finalizado | |
| **Estabilidad** | Alta | |
| **Comentarios** | Ninguno | |

Tabla 18: Requisito de Informacion del veterinario

|  |  |  |
| --- | --- | --- |
| **IRQ-0002** | **Datos de mascotas** | |
| **Versión** | 1.0 (05/05/2019) | |
| **Autores** | * Carlos Escario Bajo | |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo | |
| **Dependencias** | Ninguno | |
| **Descripción** | El sistema deberá almacenar la información correspondiente a las mascotas que se añaden al sistema. En concreto: | |
| **Datos específicos** | * Nº de identificación * Fecha de implantación * Nombre * Raza Peligrosa * Fecha de Nacimiento * Capa * Pelo * Sexo * Pasaporte animal * Aptitud * Última vacunación antirrábica * Fecha de la última revisión * Número de colegiado última revisión * Cicatrices * Numero de certificado | |
| **Tiempo de vida** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Ocurrencias simultáneas** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Importancia** | Importante | |
| **Urgencia** | Inmediatamente | |
| **Estado** | En construcción | |
| **Estabilidad** | Por definir | |
| **Comentarios** | Ninguno | |

Tabla 19: Requisitos de información de mascotas

|  |  |  |
| --- | --- | --- |
| **IRQ-0003** | **Datos del dueño** | |
| **Versión** | 1.0 (05/05/2019) | |
| **Autores** | * Carlos Escario Bajo | |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo | |
| **Dependencias** | Ninguno | |
| **Descripción** | El sistema deberá almacenar la información correspondiente al dueño de la mascota. En concreto: | |
| **Datos específicos** | * Nombre * Apellidos * Número de documento * Teléfono * Provincia * Municipio * Código Postal * País | |
| **Tiempo de vida** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Ocurrencias simultáneas** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Importancia** | Por definir | |
| **Urgencia** | Por definir | |
| **Estado** | Por definir | |
| **Estabilidad** | Por definir | |
| **Comentarios** | Ninguno | |

Tabla 20: Requisitos de información de propietarios.

|  |  |  |
| --- | --- | --- |
| **IRQ-0004** | **Saldo** | |
| **Versión** | 1.0 (05/05/2019) | |
| **Autores** | * Carlos Escario Bajo | |
| **Fuentes** | * Carlos Escario Bajo | |
| **Dependencias** | Ninguno | |
| **Descripción** | El sistema deberá almacenar la información correspondiente a al saldo en ethers o weis correspondiente a los usuarios veterinarios que usen el sistema. En concreto: | |
| **Datos específicos** | * Cuenta Ethereum * Saldo en weis * Número de colegiado | |
| **Tiempo de vida** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Ocurrencias simultáneas** | **Medio** | **Máximo** |
| Por definir | Por definir |
| **Importancia** | Importante | |
| **Urgencia** | Por definir | |
| **Estado** | Por definir | |
| **Estabilidad** | Por definir | |
| **Comentarios** | Ninguno | |

Tabla 21: Requisitos de información del saldo en ETH

### Requisitos funcionales

Ian Sommerville en su libro “Ingeniería de Software 9” define los requisitos funcionales como: “*servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas. En algunos casos, los requerimientos funcionales también explican lo que no debe hacer el sistema*” (Pearson Education, 2011, pags 84 - 85).

Estos requisitos nos darán una visión concreta de cuáles son los objetivos de funcionamiento de Petchain, y que además tienen su correspondencia con los Casos de uso descritos en la sección 5.10.

|  |  |
| --- | --- |
| **FRQ-0001** | **Acceso al sistema** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá permitir el acceso a los usuarios a través de un formulario en el que se debe introducir un nombre de usuario y una contraseña. Si los datos introducidos no son correctos, el sistema deberá mostrar en la pantalla un mensaje de error. |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 22: Requisito funcional de acceso al sistema

|  |  |
| --- | --- |
| **FRQ-0002** | **Identificación de veterinarios** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá mostrar el nombre del veterinario y el colegio en donde está inscrito, con solo introducir el número de Colegiado |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 23: rquisito funcional de identificación de verinarios.

|  |  |
| --- | --- |
| **FRQ-0003** | **Traspaso de weis** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá traspasar weis cuando se haga una modificación de los datos de una mascota, desde el veterinario modificador hasta el veterinario que dio de alta a la mascota, que estarán almacenados en el balance del SmartContract. |
| **Importancia** | Importante |
| **Urgencia** | Puede esperar |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 24: Requisito funcional de traspaso de weis

|  |  |
| --- | --- |
| **FRQ-0004** | **Navegación** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Pedro Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá permitir en todo momento la navegación por el sistema a través de las opciones que se muestren en la pantalla de inicio o en un menú situado en la cabecera de la aplicación. |
| **Importancia** | Vital |
| **Urgencia** | Imediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 25: Requisito funcional de navegación en el sistema.

|  |  |
| --- | --- |
| **FRQ-0005** | **Modificación de datos de una mascota** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa |
| **Dependencias** | Ninguno |
| **Descripción** | Como veterinario quiero poder modificar los datos sanitarios de una mascota, en concreto:   * Peligrosidad. * Nº de pasaporte. * Aptitud. * Fecha de revisión. * Estado. * Cicatrices. * Vacunas. * Observaciones |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 26: Requisito funcional de modificación de datos de una mascota

|  |  |
| --- | --- |
| **FRQ-0006** | **Baja de una mascota** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá Como veterinario quiero poder marcar una mascota como fallecida. En concreto indicar:   * Causa del fallecimiento. * Número del veterinario. * Nombre del veterinario. * Apellidos del veterinario. |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 27: Requisito funcional para la baja de una mascota

|  |  |
| --- | --- |
| **FRQ-0007** | **Consultar ETH disponibles** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá mostrar un listado con las transacciones recibidas de otros veterinarios por haber modificado los datos de una mascota dada de alta por él y el saldo total en ETH de su cuenta. , |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 28: Requisito funcional para la visualización de weis disponibles

|  |  |
| --- | --- |
| **FRQ-0009** | **Alta de una mascota** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá Como veterinario quisiera dar de alta una mascota en el sistema, mediante un formulario en la pantalla de "Alta de Mascotas", que contenga los campos imprescindibles para identificar a un animal, en concreto sería necesarios:   * Nombre del animal. * Fecha de implantación del chip. * Alta en la base de datos. * Número de identificación. * Raza. * Peligrosidad. * Fecha de nacimiento. * Capa. * Pelo. * Nº de pasaporte. * Aptitud. |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 29: Requisito funcional para el alta de una mascota.

|  |  |
| --- | --- |
| **FRQ-0010** | **Alta de propietario** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá permitir al veterinario dar de alta al dueño de una mascota, en la página de "Alta de Mascotas", que contenga los campos imprescindibles para identificar al dueño de un animal, en concreto sería necesarios:   * Colegio del veterinario. |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 30: Requisito funcional para el alta de un propietario.

|  |  |
| --- | --- |
| **FRQ-0011** | **Modificación de los datos de un propietario** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Iciar García-Ochoa * Pedro Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | Como veterinario quisiera poder modificar los datos del dueño de una mascota, en la página de "Modificación de Mascotas", que contenga los campos imprescindibles para identificar al dueño de un animal, en concreto sería necesarios:   * Nombre. * Apellidos. * Dirección: Calle, Ciudad, CP y País. * Id del documento de identificación. * Teléfono. |
| **Importancia** | Importante |
| **Urgencia** | Inmediatamente |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 31: Requisito funcional para la modificación de un propietario.

### Requisitos no funcionales

La descripción más exacta que podemos obtener de los requisitos no funcionales es la expresada en el libro Ingeniería de Software, Ian Sommerville (2009):

*Son requerimientos que no se relacionan directamente con los servicios específicos que el sistema entrega a sus usuarios. Pueden relacionarse con propiedades emergentes del sistema, como fiabilidad, tiempo de respuesta y uso de almacenamiento. De forma alternativa, pueden definir restricciones sobre la implementación del sistema, como las capacidades de los dispositivos I/O o las representaciones de datos usados en las interfaces con otros sistemas* (pag. 87).

|  |  |
| --- | --- |
| **NFR-0001** | **Visualización** |
| **Versión** | 1.0 ( 05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá ajustar la visualización de las interfaces gráficas en función del modelo de desarrollo responsive. Es decir será el cliente quien detecte el tipo de dispositivo desde el que se está accediendo a la web. |
| **Importancia** | Quedaría bien |
| **Urgencia** | Puede esperar |
| **Estado** | Finalizado |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

Tabla 32: Requisito no funcional de visualización de Petchain.

|  |  |
| --- | --- |
| **NFR-0002** | **Moneda de pago** |
| **Versión** | 1.0 ( 05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá permitir el pago en weis, siendo la conversión en ethers de 1 ether = 1 e-18 wei. |
| **Importancia** | Importante |
| **Urgencia** | Por definir |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 33: Requisito no funcional del pago en weis

|  |  |
| --- | --- |
| **NFR-0003** | **Tiempo de sesión** |
| **Versión** | 1.0 ( 05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá controlar que el tiempo que puede estar una sesión inactiva, será como máximo de 15 minutos. |
| **Importancia** | Importante |
| **Urgencia** | Puede esperar |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 34: Requisito no funcional del tiempo de sesión.

|  |  |
| --- | --- |
| **NFR-0004** | **Usuarios autorizados** |
| **Versión** | 1.0 ( 05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá tener almacenados los usuarios de la aplicación en un fichero en formato JSON, en donde se indique su número de colegiado, nombre de usuario, contraseña y número de colegiado. |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | Pendiente de validación |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 35: Requisito no funcional de usuarios autorizados

|  |  |
| --- | --- |
| **NFR-0005** | **Número de cuentas** |
| **Versión** | 1.0 ( 05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá tener definidas por defecto al menos 100 cuentas en el servidor Ganaché. |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 36: Requisito no funcional del nº de cuentas en Ganache

|  |  |
| --- | --- |
| **NFR-0006** | **Autenticación JWT** |
| **Versión** | 1.0 ( 05/05/2019 ) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá tener un control de autenticación mediante JWT, que se almacena como variable en la SessionStorage. |
| **Importancia** | Importante |
| **Urgencia** | Puede esperar |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 37: Requisito no funcional para la autenticación JWT

|  |  |
| --- | --- |
| **NFR-0007** | **Campos obligatorios** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El sistema deberá mostrar en pantalla un mensaje que informando de aquellos campos que son obligatorios y no han sido rellenados en el momento de enviar el formulario y señalándolos en color rojo. |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 38: Requisito no funcional de campos obligatorios

|  |  |
| --- | --- |
| **NFR-0008** | **Validación de código** |
| **Versión** | 1.0 (05/05/2019) |
| **Autores** | * Carlos Escario Bajo |
| **Fuentes** | * Carlos Escario Bajo |
| **Dependencias** | Ninguno |
| **Descripción** | El código desarrollado para la aplicación no deberá contener errores de codificación, por lo que todos los ficheros estáticos html y css deberán superar las pruebas de validación que establece el consorcio W3C en <https://validator.w3.org/nu/#file> |
| **Importancia** | Vital |
| **Urgencia** | Inmediatamente |
| **Estado** | En construcción |
| **Estabilidad** | Por definir |
| **Comentarios** | Ninguno |

Tabla 39: Requisito no funcional de validación de código

## Matriz de rastreabilidad

### Objetivos del sist. – Requisitos funcionales / Requisitos no funcionales

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TRM0001** | **FRQ0001** | **FRQ0002** | **FRQ0003** | **FRQ0004** | **FRQ0005** | **FRQ0006** | **FRQ0007** | **FRQ0009** | **FRQ0010** | **FRQ0011** | **NFR0001** | **NFR0002** | **NFR0003** | **NFR0004** | **NFR0005** | **NFR0006** |
| **OBJ0001** | - | - | - | - | - | - | - |  | - | - | - | - | - | - | - | - |
| **OBJ0002** | - | - | - | - | - | - | - | - |  | - | - | - | - | - | - | - |
| **OBJ0005** | - | - |  | - | - | - |  | - | - | - | - | - | - | - | - | - |
| **OBJ0006** | - | - | - | - |  | - | - | - | - | - | - | - | - | - | - | - |
| **OBJ0007** | - | - | - | - | - | - | - | - | - |  | - | - | - | - | - | - |
| **OBJ0008** | - | - | - | - | - |  | - | - | - | - | - | - | - | - | - | - |
| **OBJ0009** | - | - | - | - | - | - |  | - | - | - | - | - | - | - | - | - |
| **OBJ0011** |  | - | - | - | - | - | - | - | - | - | - | - | - |  | - |  |
| **OBJ0012** | - | - | - | - | - | - | - | - | - | - |  | - | - | - | - | - |
| **OBJ0013** | - | - |  | - | - | - | - | - | - | - | - |  | - | - | - | - |
| **OBJ0014** | - | - | - |  | - | - | - | - | - | - | - | - | - | - | - | - |
| **OBJ0015** | - | - | - | - | - | - | - | - | - | - | - | - |  | - | - | - |

Tabla 40. Matriz de dependencias de requisitos

## Casos de uso

Esta sección mostrará, a través de diagramas UML, los diferentes requerimientos del sistema, permitiendo tener una visión más clara de los aspectos funcionales de la aplicación, los actores que intervienen y los sistemas externos a los que están conectados.

### Gestión de mascotas

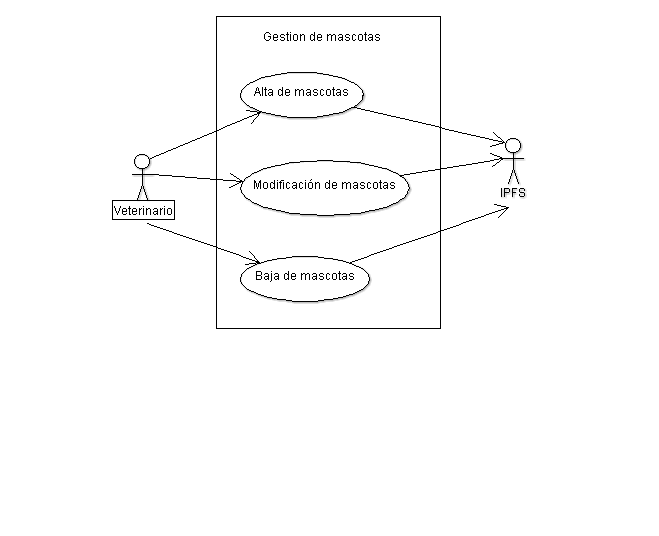


Ilustración 10: Gestión de mascotas

Aquí se muestra como el veterinario podrá realizar las acciones de alta, modificación y baja sobre los datos de una mascota, y su posterior almacenamiento en la red IPFS. Tanto el alta de mascotas como la modificación, son procesos que a nivel de gráfico están integrados dentro del mismo proceso que la Gestión de propietarios, pero en la presente implementación se consideran Casos de Uso separados, pues su gestión de almacenamiento se llevará a cabo de manera separada dentro de la IPFS.

### Gestión de propietarios

En este Caso de Uso se expone la gestión de los propietarios de las mascotas, y que como se observa sólo es posible realizar el alta y la modificación. Esto se debe a que un propietario siempre estará relacionado con una mascota, y las acciones siempre se realizarán sobre ésta, no sobre él, por lo que el sistema nunca permitirá el darlo de baja.

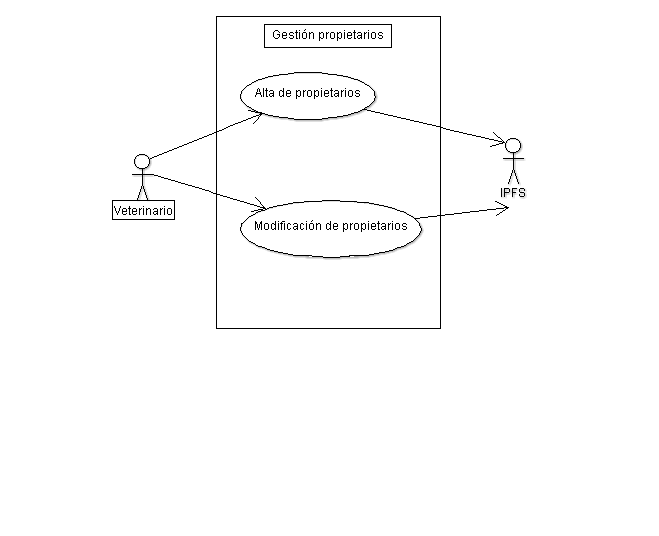


Ilustración 11: Gestión de propietarios

### Consulta de saldo

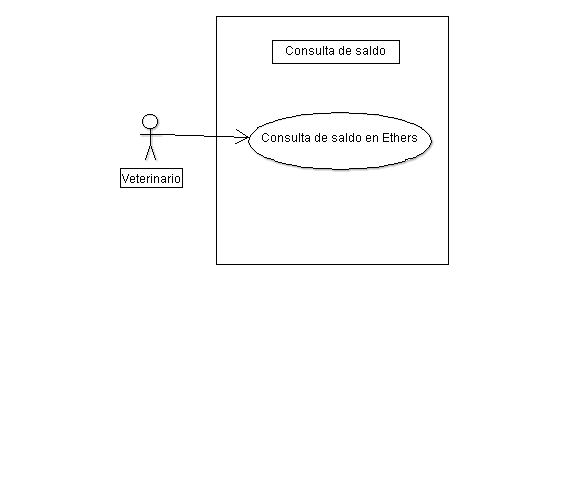


Ilustración 12: Consulta del saldo en Ethers

Este caso de uso muestra la consulta del saldo en moneda Ethers. Cada veterinario recibirá la cantidad de 0,001 ETH de otro veterianario dado de alta en el sistema por las acciones de modificación y baja que este último realice sobre mascotas que el primero introdujo en el sistema.

### Gestión de usuarios

# Metodología y Plan de trabajo

La metodología utilizada para el desarrollo de proyecto es el modelo evolutivo - iterativo..POR REDACTAR



## Seguimiento del proyecto

El seguimiento del proyecto ha sido realizado con la herramienta Trello[[6]](#footnote-6).

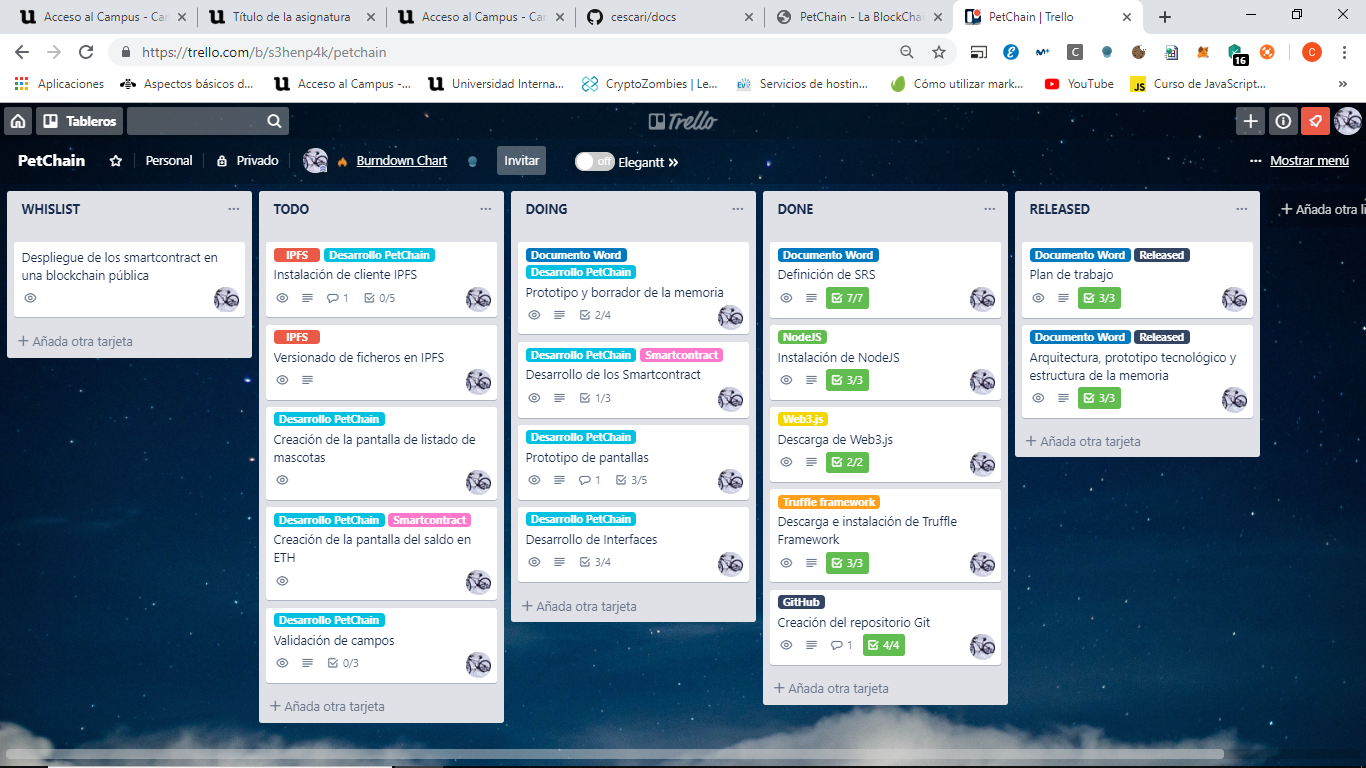


Ilustración 13: Tablero principal de Petchain en Trello

# Descripción técnica.

Petchain es una aplicación web que emplea para su ejecución una combinación de tecnologías, con el objetivo de conectar un frontal en HTML con un backend desarrollado con Smartcontracts e IPFS.

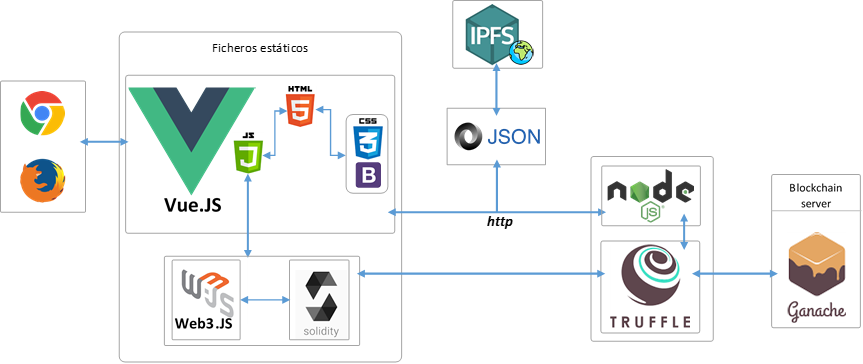


Ilustración 14: Esquema gráfico de herramientas de Petchain (Fuente: diseño propio)

## Arquitectura

Como ya se ha indicado, PetChain es una DApp, diseñada en forma de aplicación web, que tiene entre sus objetivos el mostrar a los usuarios una interfaz con un manejo sencillo y usable, basada en el patrón MVC (*Model View Controller*). “Separa presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí.” (Sommerville, 2009, pag. 145).

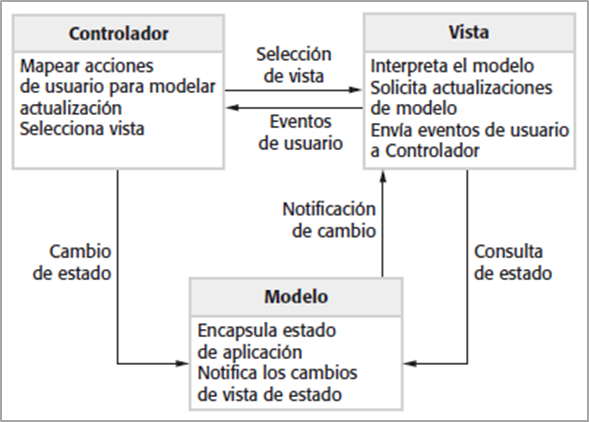


Ilustración 15: Patrón MVC (Fuente: Sommerville, 2009, pag. 156)

El objetivo que se pretende al usar este patrón, es el de lograr una independencia de los elementos que componen la aplicación, con el foco puesto en la simplificación de los procesos evolutivos y correctivos. Para ofrecer una visión más específica de que elementos de la tecnología utilizada forman parte de cada uno de los componentes del patrón, se muestra en la tabla siguiente una correspondencia entre ellos, y en las secciones siguientes se hace una descripción más pormenorizada de los ficheros involucrados dentro de cada una de las capas del patrón MVC.

|  |  |
| --- | --- |
| **Relación Arquitectura- tecnologías utilizadas** | |
| **Vista** | Vue (html) |
| css |
| JavaScript |
| **Controlador** | Javascript |
| **Modelo** | Solidity |
| JSON |

Tabla 41: Relación Arquitectura - Tecnología

### Modelo

Es la capa que se comunica con los datos, bien con la base de datos o con una lógica del negocio, y cuyo objetivo es desacoplar la parte gráfica de los datos que maneja la aplicación. En Petchain la capa que conforma el modelo está compuesta, por el conjunto de ficheros *\*.sol* (solidity), los ficheros con estructuras JSON que se emplean para poblar elementos del frontal y los archivos Javascript que componen los servicios, así como los necesarios para la compilación de los ficheros solidity que se localizan en la carpeta /truffle/migrations.

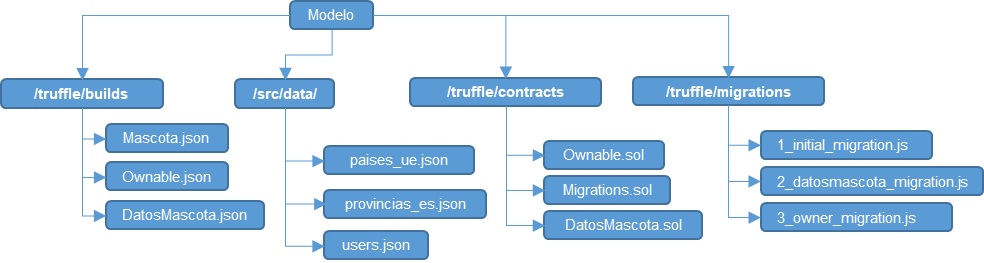


Ilustración 16: Estructura de los ficheros del Modelo

### Vista

La vista es el conjunto de ficheros que componen la parte gráfica y que determinan como debe de visualizarse la aplicación. El conjunto de ficheros que componen la estructura de la vista son los reflejados en la imagen siguiente.

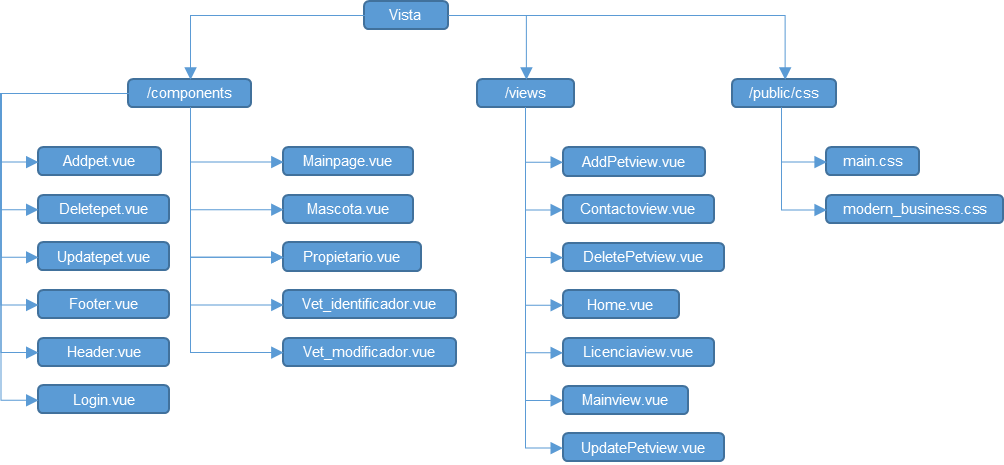


Ilustración 17: Estructura de ficheros de la Vista

Obsérvese la particularidad de la no existencia de ficheros \*.html en la estructura mostrada. En su lugar están los ficheros con extensión \*.vue, tanto para los componentes como para las vistas que componen la aplicación. Esta característica se comentará en secciones posteriores y en profundidad, pues está íntimamente ligada al framework Vue.js que ha utilizado para el desarrollo del proyecto.

### Controlador

Esta es la capa encargada de realizar las modificaciones realizadas por el modelo y/o la vista en respuesta a las peticiones realizas desde el frontal o la base de datos. Su objetivo es servir como elemento de abstracción entre el modelo y la vista. En la aplicación se ha buscado optimizar la codificación, es decir hacerla lo más estándar posible.

La labor de esta capa es de suma importancia, pues en ella se genera y se envía o se recibe un objeto con notación javascript (JSON), de los datos procedentes de la interfaz de usuario, o bien de los procesados por la lógica almacenada en los ficheros solidity y que es enviada a la IPFS.

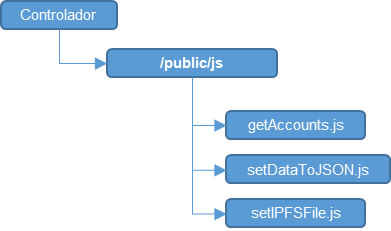


Ilustración 18: Esquema de la capa controladora

### Análisis de elementos: Componentes y Vistas.

Hasta el momento se ha realizado una exposición del patrón que se ha seguido para la arquitectura del proyecto. Hemos visto que elementos forman cada una de sus partes y se ha destacado en el apartado 5.1.2, la particularidad de la no existencia de ficheros \*.html dentro de su estructura. En su lugar hay ficheros con la extensión \*.vue alojados en dos carpetas: /*components* y /*views*. Los nombres de cada una de ellas son suficientemente descriptivos, en la primera se alojan los componentes, que serían los ladrillos básicos que forman las pantallas, y en la segunda se alojan las vistas, que son agrupaciones de 1 o n componentes y que tienen como finalidad su presentación en pantalla.

Por otro lado, la agrupación de componentes en vistas, facilita su comunicación a la hora de realizar flujos de datos entre ellos o realizar algún tipo de cambio de estado debido a que la vista actúa como “padre” y los componentes como elementos “hijo”, permitiendo una comunicación bidireccional. Esta característica reduce el número de ficheros empleados así como las líneas de código necesarias dentro de la aplicación.

En las subsecciones siguientes se realiza una descripción más pormenorizada de estos dos elementos.

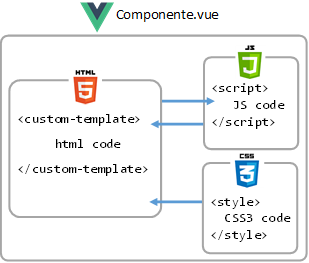
**Componentes**

Ilustración 19: Elementos de un componente Vue.js

Como se ha introducido en la sección anterior, los componentes son los ladrillos que forman las pantallas de nuestra aplicación.

En la imagen anterior se observa de manera gráfica la estructura de un componente. En ella se distinguen tres elementos:

* <template>: En esta sección se aloja el código html que forma el componente y que se visualizará en el navegador.
* <script>: Sección que contiene el código javascript necesario para el funcionamiento del componente. Aquí es donde se importan ficheros externo, se definen propiedades, los valores que toman estas y que se representan posteriormente en el template gracias a la comunicación bidireccional existente entre ellos.
* <style>: Sección que contendrá los estilos css que manejarán el formato de presentación del componente en pantalla. A diferencia de los dos anteriores, su definición no es obligatoria, pues los estilos pueden ser establecidos en ficheros externos a nivel general de la aplicación pero si es conveniente, en caso de necesitarlos, para favorecer la encapsulación de comportamientos.

El uso de componentes tiene unos beneficios directos en relación al código y al diseño:

* **Reutilización**: Un componente sólo es necesario definirlo una vez y puede ser utilizado en tantas vistas como sea necesario.
* **Encapsulación**: El comportamiento de un componente está restringido a su ámbito de actuación o “*scope*”, sin que interfiera, si así se requiere, con el resto de código de una vista.

Hasta el momento de la creación del presente capítulo, en PetChain se han definido catorce componentes dentro de la ruta /src/components/.

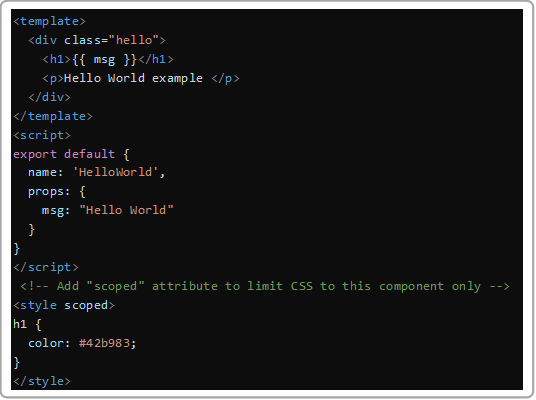
**Vistas**

Ilustración 20: Ejemplo de componente simple Hello World

Las vistas son agrupaciones de 1 o n componentes que combinándolos e interactuando entre ellos, permiten la definición de las pantallas. A nivel de código tienen la misma estructura que éstos, pero incorporan una característica muy importante que los hacen diferentes, y es que importan en su interior los ficheros necesarios para la formación de las pantallas.

En la imagen siguiente se muestra el flujo de composición de la pantalla de Contacto *(/src/views/Contactoview.vue*).

Ilustración 21: Flujo de composición de la vista.Contactoview.vue

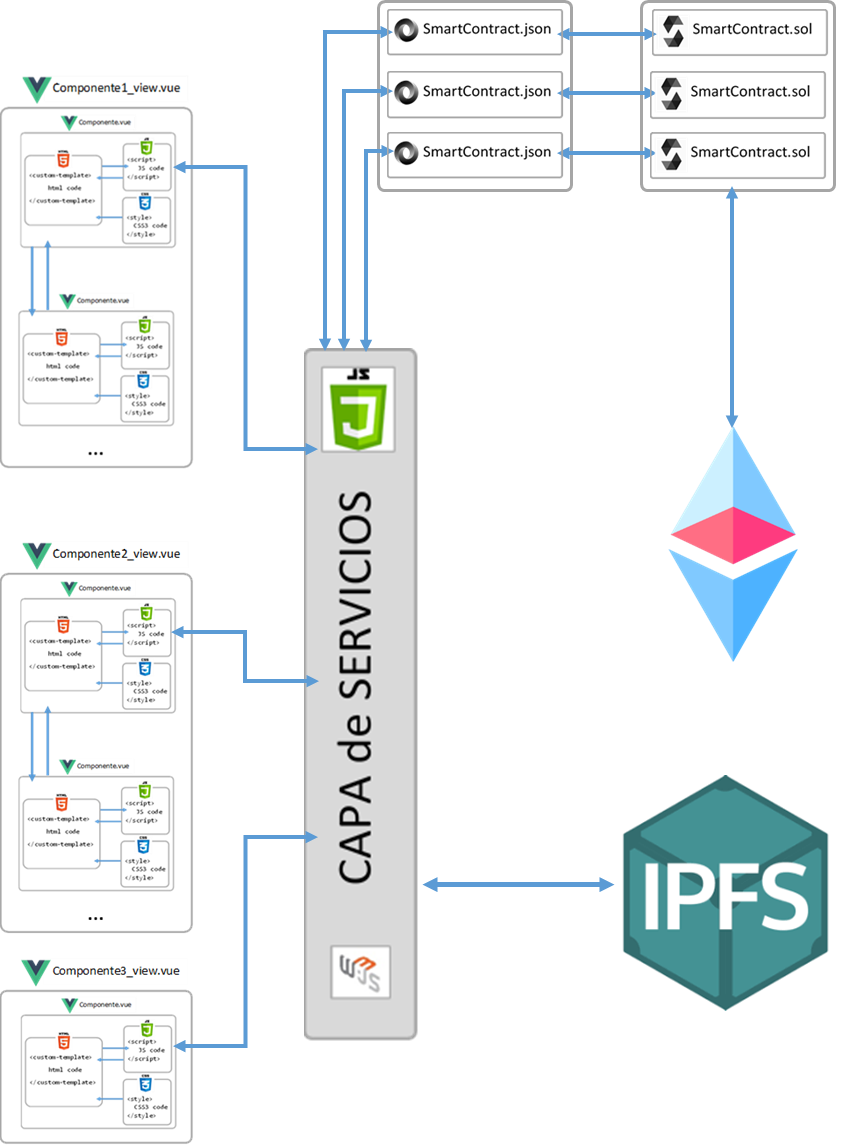
1. En primer lugar se importan los ficheros que van a componer la vista asociándolos a un “alias”. En nuestro ejemplo importamos *Menu.vue* y *Contacto.vue*, asociándolos a los alias “Menu” y “Contacto”, respectivamente.
2. A continuación indicamos que esos dos elementos los exportamos y que se van a comportar como componentes. Al hacer esto, le estamos indicado al framework que vamos a utilizar etiquetas personalizadas, con formato html y que en su interior van a contener todo el código de cada uno de los elementos importados.
3. En la sección del *template* ahora podemos definir tantas etiquetas personalizadas como sea necesario, con el objetivo de formatear la pantalla que necesitamos.
4. Por último, y tras la compilación de los ficheros, el navegador es capaz de representar las pantallas con el formato que hayamos definido, en nuestro ejemplo la pantalla de Contactos.

## Arquitectura técnica.

Hasta ahora se ha analizado el patrón arquitectónico de PetChain relacionándolo con los componentes de la aplicación, con el objetivo de tener una idea clara de la relación entre ellos. Ahora se pretende ahondar en la arquitectura técnica, definiendo y describiendo las capas que la componen así como la relación entre ellas.

Para comenzar, en la siguiente ilustración, se muestra una descripción detallada de los elementos, su posición en las capas de la aplicación y su forma de relacionarse.

Ilustración 22: Esquema de componentes.





1. **Capa de presentación**: Formada por los componentes, y estos a su vez agrupados en las vistas, estos elementos ya han sido descritos en secciones anteriores. Como se puede observar todos los componentes se relacionan con el exterior a través de la sección del *script*. Desde aquí se pueden hacer invocaciones a los elementos padres, hijos o hermanos, así como a otras vistas. En nuestra arquitectura, existe una comunicación entre componentes de la misma y de diferentes vistas, y también con la capa de servicios. Todas las comunicaciones con esta capa se realizan siempre invocando o recibiendo objetos de tipo *data()*, en formato JSON. De este modo se evita el acoplamiento y facilitando el mantenimiento evolutivo y correctivo.
2. **Capa de servicios**: Esla encargada de comunicar la capa de presentación con el backend y viceversa. Aquí se producen las comunicaciones bidireccionales entre los Smartcontracts para su ejecución en la red Ethereum, con el servidor Ganache y el almacenamiento y recuperación de ficheros almacenados en la IPFS. Está formada por los ficheros:
   1. */public/js/services/getAccounts.js:* recibe las cuentas del servidor Ganache, para asociarlas a los usuarios del sistema.
   2. */public/js/services/setIPFSFile.js:* encargado de subir a la IPFS el objeto JSON con los datos de las mascotas y propietarios, así como de recuperar el HASH generado en el almacenamiento.
   3. */truffle/builds/* : En esta carpeta se alojan los ficheros \*.json, que son el resultado de la compilación de los Smartcontracts codificados en lenguaje solidity (\*.sol). Éstos son con los que el frameworlk Web3.js es capaz de interactuar, y por lo tanto que se realice la comunicación con ellos desde la capa de front-end.
3. **Capa de backend**: Formada por:
   1. La red Ethereum, que en nuestro caso la ofrece el servidor local Ganache en la url http://127.0.0.1:7545.
   2. La IPFS servida desde el nodo público de INFURA en la url: https://ipfs.infura.io:5001

Todos los elementos anteriores necesitan estar orquestados por diversas tecnologías, que deben de trabajar en conjunto. En los próximos apartados se van a introducir y explicar con detalle cada una de las herramientas, frameworks y servidores necesarios para el correcto funcionamiento de la aplicación.

|  |
| --- |
| **ATENCION**: Recordamos el **Objetivo 3** del punto **1.3.2 Objetivos específicos**, de utilizar software de coste cero. |



## Node.js

Node.js es un entorno multiplataforma de código abierto, disponible para su descarga libre en [https://Node.js.org](https://nodejs.org/), y utilizado para la ejecución de JavaScript en el lado del servidor. Su funcionamiento se basa en la ejecución de eventos de manera asíncrona, esto significa que mientras Node.js gestiona cualquier solicitud de I/O que le haya llegado y hasta que la petición finalice, ésta se estará ejecutando en segundo plano, y mientras el sistema podrá seguir realizando cualquier otra operación en primer plano.

Gran parte de su éxito se basa en la enorme cantidad de módulos existentes que permiten extender las funcionalidades del entorno. Para la instalación de dichos módulos se emplea un gestor de paquetes incluido en Node.js llamado npm (Node Package Manager)[[7]](#footnote-7).

|  |
| --- |
| **ATENCION**: Todas las instalaciones hechas vía **npm**, son realizadas con el parámetro **--save-dev**. Este parámetro indica que todos los módulos se deben instalar en la lista de dependencias dentro del fichero **package.json**. |

En el proyecto estamos utilizando la versión 8.12.0, que aunque no es la más actualizada para su descarga (a la fecha de la redacción del presente trabajo la última versión disponible es la 10.16), cumple completamente con las necesidades del mismo, pues si es requisito técnico para el funcionamiento de los módulos instalados, que la versión de Node.js sea superior a la 6.0.0.

### Dependencias

Los módulos a los que nos hemos referido anteriormente, y que son necesarios para el funcionamiento de PetChain, son los que hay definidos en el fichero /*package.json* dentro del objeto *dependencies*.

* **core-js**: librería de utilidades javascript.
* **ipfs-mini**: módulo con un API que nos permitirá conectar con la IPFS.
* **jquery**: versión 3.4.1 de la conocida librería de javascript.
* **vue**: módulo que contiene las librerías necesarias del framework Vue.js.
* **vue-router**: utilidad para la gestión de url´s y navegación entre pantallas.
* **web3**: módulo que contiene las librerías necesarias del framework Web3.js. La versión instalada es la 0.20.6, pues para versiones posteriores hay reportadas incompatibilidades con el servidor de Truffle empleado para la compilación de los ficheros solidity y el servidor anache. Por lo que tras múltiples consultas en foros de internet se ha optado por instalar esta versión, que aunque no es la más actualizada si es la más estable hasta la fecha.

## Vue.js

Para la generación del proyecto se ha escogido el framework de Javascript Vue.js. Éste junto con Angular y React, forman la trilogía más utilizada a nivel global para el desarrollo de aplicaciones SPA bajo Node.js.

Se ha escogido Vue.js fundamentalmente por diversas razones:

* **Curva de aprendizaje**: Previamente al desarrollo del presente proyecto sólo se tenían conocimientos de Angular no así de Vue.js ni de React. La experiencia con Angular no había sido muy positiva, pues la exigencia de la curva de aprendizaje era muy alta. Tras la primera toma de contacto con Vue y React, se ha observado la sencillez de uso del primero, pudiendo comprobarlo con la creación de ejemplos sencillos de SPA`s, en contraposición a React, que de inicio exige el aprendizaje de una nueva sintaxis de etiquetas denominada JSX.
* **Documentación**: Vue.js contiene una muy extensa documentación para su aprendizaje, tanto en texto como en vídeo, dentro de su web: <https://vuejs.org/v2/guide/>. Asimismo, existe una muy amplia documentación en castellano que ayuda a un mejor aprendizaje.
* **API**: el API de Vue.js es realmente sencilla e intuitiva.
* **Creación de proyecto**: La creación de un proyecto Vue.js se realiza con un solo comando en la ventana del sistema, generando automáticamente el *scaffolding* del proyecto, sin necesidad de arduas configuraciones. PetChain ha sido desarrollado con la librería Vue-CLI, que está instalada como una dependencia en el entorno de Node.js, y permite la generación de un proyecto web en un entorno local.
* **Detección de errores en el puerto 8080**: Vue.js se ejecuta por defecto en modo desarrollo dentro de un servidor local en la url <http://localhost:8080>. El framework tiene la capacidad de detectar en tiempo de compilación si el puerto 8080 tiene algún problema y de manera automática redirecciona la url de ejecución al puerto 8081, e informando de ello al usuario vía consola para su ejecución en el navegador.

### Requerimientos

* Versión de Node.js +8.9.0
* Versión de npm +6.0

La instalación de Vue-CLI la realizamos con el siguiente comando:

|  |
| --- |
| **> npm install –g @vue/cli** |

Tabla 42: Comando de instalación de Vue-CLI

## Truffle

Truffle[[8]](#footnote-8) es un framework que permite el desarrollo, compilación, despliegue y pruebas de Smartcontracts sobre redes Ethereum. Es una herramienta que se instala como módulo de Node.js, v8.9.4 o superior, y está disponible para las plataformas Windows, Linux y Mac OS X. Las características que ofrece son:

* Scaffolding inicial del proyecto.
* Compilación de Smartcontracts y generación de archivos en formato JSON con la estructura del Smartcontract y su bytecode correspondiente.
* Despliegue sobre redes privadas y públicas.
* Generación y ejecución de test.

### Instalación de Truffle

Como se ha comentado anteriormente Truffle está disponible como un módulo del entorno de ejecución Node.js, por lo tanto para su instalación haremos uso de npm (Node Package Manager). Ejecutamos entonces la sentencia siguiente:

|  |
| --- |
| **> npm install –g truffle –save-dev** |

Ilustración 23: Comando de instalación de Truffle

Los parámetros de instalación son opcionales, pero permitirán una ejecución y actualización de las dependencias, en caso de actualizarse las versiones.

### Generación del proyecto

Dado que hemos realizado la instalación de Truffle con el parámetro –g, desde ahora vamos a tener disponible desde cualquier parte de la estructura de archivos de nuestra máquina los comandos ejecutables del framework, que permitirán su instalación y configuración desde la línea de comandos.

Para nuestro proyecto, ejecutaremos:

|  |
| --- |
| **> truffle init** |

Ilustración 24: Comando de inicialización de un proyecto Truffle

Este comando genera la estructura necesaria para el proyecto, que nos permitirá el almacenamiento de los ficheros solidity (\*.sol), su compilación y despliegue de binarios. Dicha estructura consta de los siguientes elementos.

* contracts/: directorio para el almacenamiento de ficheros solidity (\*.sol).
* test/: directorio para el almacenamiento de los test.
* migrations/: directorio para el almacenamiento de los scripts correspondientes a las migraciones de los archivos con extensión .sol a ficheros .js, y que contendrán la estructura del Smartcontract en formato JSON, así como en bytecode.
* truffle-config.js: fichero de configuración de Truffle para la definición de las redes de despliegue de los Smartcontracts.

|  |
| --- |
| **ATENCION**: Es necesario indicar que en la instalación de Truffle, se requiere **obligatoriamente** que la carpeta en donde se vaya a ejecutar el comando **truffle init**, esté completamente **vacía**. Si no se cumple esta premisa, el framework devolverá siempre un error indicando que no es posible su instalación para evitar sobre-escrituras a proyectos ya existentes. |

## Ganache

Ganache[[9]](#footnote-9) es un componente multiplataforma (Windows, Linux o Mac OS X), que forma parte del framework Truffle y su función es la servir como servidor de cuentas en una red Ethereum. Es el complemento ideal para el desarrollo de aplicaciones Blockchain en entornos cerrados, locales o corporativos, y está disponible para su ejecución como herramienta de escritorio o para invocarlo a través de línea de comandos (TestRPC).

### Instalación de Ganache

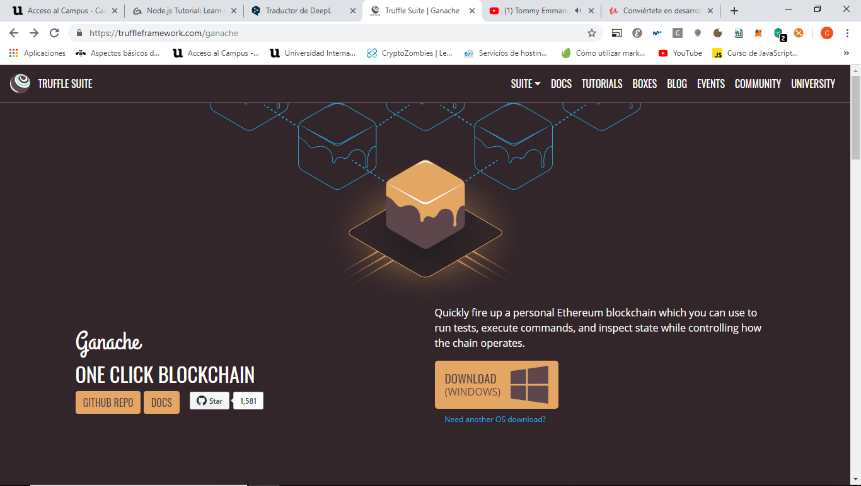
La instalación de Ganache es trivial y se realiza a través de un cásico “wizzard”. Para el presente proyecto se implementará como una herramienta de escritorio, tanto para un sistema Windows 10 y Linux Ubuntu 18.02, a través del instalador que se puede descargar desde la página web de Truffle.

Ilustración 25: Web de Truffle framework.

## Web3.js

Web3.js es la herramienta JavaScript, open source, multiplataforma e imprescindible para interactuar desde el navegador contra nuestro servidor Ethereum a través del protocolo http. Este servidor, será el que se denomine object provider, que en nuestro caso será el servidor de cuentas Ganache. Para que exista esta comunicación entre el frontal web y el object provider, web3.js expone un API[[10]](#footnote-10) con gran cantidad de métodos con los que comunicarse con su entorno.

### Instalación de Web3.js

Esta librería está disponible de diversos métodos:

* Es posible incorporarla a nuestro proyecto como un módulo de Node.js a través del comando npm en la carpeta del proyecto, y se instalará como una dependencia del proyecto dentro del package.json de Node.js. Como se ha indicado en el punto 5.2.1, con el fin de evitar incompatibilidades con Truffle la versión instalada es la 0.20.6.
* También es posible tener disponible la librería cuando se instala el framework de Truffle, dado que está incorporado como parte del paquete., la cual tiene bastantes problemas en cuanto a funcionalidades deprecadas.

|  |
| --- |
| **> npm install –g web3 –save-dev** |

Tabla 43: Instalación de web3.js

## Bootstrap

Bootstrap[[11]](#footnote-11), framework original de la empresa Twitter, es quizás una de las librerías de código abierto más conocidas en el sector del desarrollo web. Permite crear webs, webapps y DApps adaptables (*reponsive design*), multiplataforma, bajo el paradigma de “mobile first”. Con una amplísima documentación a lo largo de toda la Internet, la correcta aplicación de su extensa gama de plantillas gratuitas, estilos y scripts predefinidos, permiten al desarrollador el diseño de pantallas de diseño limpio y claro, con estilos modernos y con una extensa gama de componentes.

Desde contenedores, menús desplegables, formularios, validadores, componentes HTML, etc., Bootstrap ofrece un ahorro en tiempo y código a los diseñadores y programadores web, y, que bajo el criterio del autor, no hay duda en calificarla como la mejor herramienta existente en la actualidad, dentro de su dominio de aplicación.

La implementación dentro de Petchain se ha realizado descargando los ficheros estáticos de la web del framework, e incorporándolos a la estructura del proyecto dentro de la carpeta */src/vendor/bootstrap*. Para la aplicación se descarga la última versión disponible, la 4.3.1.

La librería de Bootstrap la incorporamos a través de un CDN de acceso público, lo que nos proporciona la ventaja de:

* Ahorro de espacio en el proyecto, pues nos evita la descarga de ninguna clase de fichero en el directorio local.

La invocación de los ficheros necesarios para tener accesible Bootstrap se realiza en */public/index.html* a través de las siguientes urls:

* [*https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css*](https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css): para el minificado de los estilos css.
* [*https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js*](https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js)*:* para el minificado con el javascript necesario para interactuar con los estilos Bootstrap.

## Herramientas utilizadas

### Argo UML

Herramienta Open Source, desarrollada en Java, bajo licencia EPL y con soporte para diagramas UML ver, 1.4. Ha sido el software utilizado para el diseño de los diagramas de Casos de Uso y los Diagramas de secuencia. Se encuentra disponible para su descarga gratuita en: <http://argouml.tigris.org/>.

### REM

Según se cita en la web del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla (2004)[[12]](#footnote-12):

*REM (REquirements Management) es una herramienta experimental gratuita de Gestión de Requisitos diseñada para soportar la fase de Ingeniería de Requisitos de un proyecto de desarrollo software de acuerdo con la metodología definida en la Tesis Doctoral "Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información", presentada por Amador Durán en septiembre de 2000.*

### GitHub Desktop

Cliente gráfico de GitHub, para la conexión al repositorio remoto de Git. Herramienta gratuita y disponible para su descarga en <https://desktop.github.com/>. Se ha utilizado como repositorio tanto para el código del proyecto Petchain, como para la presente memoria.

### Trello

Software para la gestión de tareas de un proyecto, basado de una interfaz web, y que permite de modo visual llevar el control de éstas en formato de tableros con tarjetas, en donde se podrán definir, listas, comentarios, fechas de entrega, miembros de la tarea, etc. Desde el año 2017 forma parte de la lista de herramientas de la empresa Atlassian. Disponible en <https://trello.com/>, dispone de licencias gratuitas y premium.

### Visual Studio Code

Editor de código fuente de la empresa Microsoft, con múltiples capacidades. Disponible para las plataformas Windows, Linux y Mac OS, permite además de la edición de código, conexión con repositorios Git, depuración de código, resaltado de errores de código y con una extensa lista de lenguajes soportados como Javascript, Solidity, C, C++, C#, Java, Python, PHP, Go, etc. Disponible para su descarga gratuita en: <https://code.visualstudio.com/download>.

# Petchain. Descripción funcional

En el capítulo anterior se introdujo de una manera detallada las características técnicas de la aplicación mostrando una visión a bajo nivel de la misma. En el presente capítulo el objetivo es que el lector obtenga una visión en profundidad de los aspectos funcionales de la aplicación con el objeto de que se familiarice con su manejo y funcionamiento.

## Login

Ilustración 26: Login PetChain

En correspondencia al FRQ001[[13]](#footnote-13), el acceso a la aplicación está securizado, mediante usuario y contraseña a través de la pantalla de login, en donde el usuario debe de estar registrado previamente en el fichero /src/data/user.json. En caso de no introducirse las credenciales correctamente el sistema lo indicará con un mensaje de “Usuario no encontrado”, en color rojo y situado la zona superior del formulario de entrada.

Ilustración 27: Login incorrecto.

Tal y como muestra la pantalla, en caso de no recordar la contraseña, el usuario tendrá la posibilidad de solicitar al sistema, a través del envío de un correo electrónico a la dirección que muestra, la contraseña que tiene registrada en el sistema.

## Pantalla principal

## Alta de mascotas

## Modificación de mascotas

## Baja de mascotas

## Transferencia de ETHERS

## Pantallas de información

# Test de código.

El objetivo de esta sección es comprobar como el código desarrollado cumple con las expectativas, requisitos y criterios técnicos definidos. En concreto se van a realizar los siguientes test para la validación de la aplicación:

* Validación de código.
* Test de Smartcontracts.
* Validación de objetos JSON.
* Visualización en diferentes dispositivos.



## Validación de código

En el capítulo 3 se hizo una enumeración de los requisitos necesarios para la implementación de Petchain, en concreto dentro de los requisitos no funcionales, se estableció la necesidad de la validación del código según los estándares del W3C (NFR0008).

Para la realización de estas pruebas se han cargado los ficheros estáticos (*\*.html y \*.css*) en la página web <https://validator.w3.org/nu/#file> con el objeto de conocer el nivel de cumplimiento del código desarrollado.

### Validación inicial (\*.html)

Basándonos en las pruebas realizadas, los ficheros \*.html de Petchain arrojan los siguientes resultados:

| **Pruebas de validación de código (\*.html)** | | |
| --- | --- | --- |
| Login.html | * **OK** |  |
| Contacto.html | * **Errores: 6** * **Warnings: 6** |  |
| Addpet.html | * **Errores: 9** * **Warnings: 6** |  |
| Deletepet.html | * **Errores: 9** * **Warnings: 4** |  |
| Updatepet.html | * **Errores: 12** * **Warnings: 6** |  |
| Licencia.html | * **Errores: 2** * **Warnings: 6** |  |
| index.html | * **Errores: 6** |  |

Tabla 44: Resultados de la validación inicial de los ficheros html

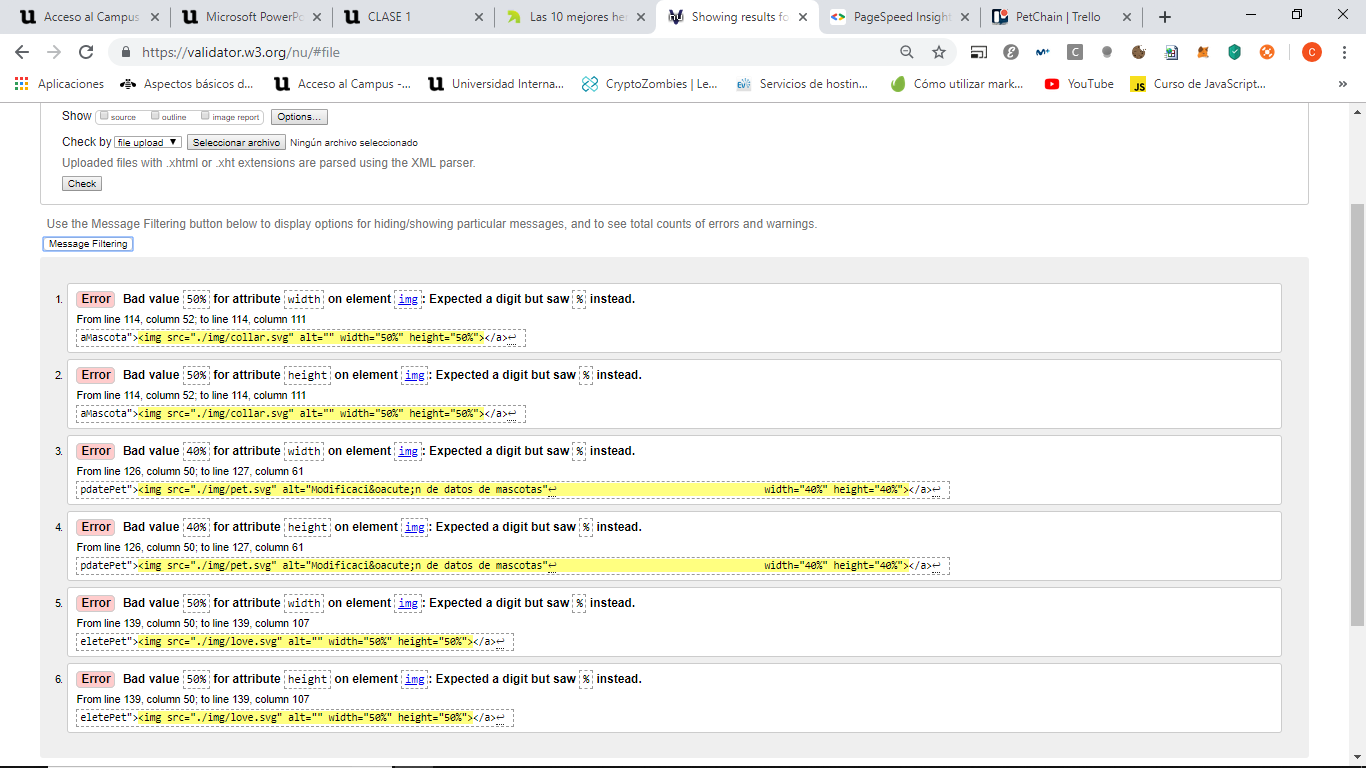


Ilustración 28: Resultados W3C de la validación de index.html

Los errores detectados en los ficheros html corresponden a:

* El uso deprecado o desaconsejado de atributos definidos en línea sin un valor numérico explícito, es decir usando “px” o “%” dentro de las etiquetas. Por ejemplo como se muestra en la imagen anterior, para la definición de atributos alto y ancho se aconseja el uso de estilos css (inline o en una clase) o bien la definición de un valor numérico sin modificadores en píxeles o porcentaje.
* En la línea del punto anterior, el uso de estos mismos atributos dentro de la definición del iframe, para determinar su alto y ancho.
* Definición de etiquetas <option value=””>, en donde el atributo “value” no puede contener un valor vacío aunque sea el elemento inicial que actúa a modo de etiqueta descriptiva. Se aconseja siempre la definición de un valor dentro de cualquier atributo html.

Por otro lado, el validador muestra advertencias o *warnings* sobre elementos que, a pesar de no considerarlos correctos, no invalidan el código en su totalidad. Como ejemplo de este tipo de avisos tenemos:

* Los generados por el uso innecesario del atributo type=”text/javascript”dentro de las etiquetas <script type=”text/javascript”>. El problema de este tipo de atributos, es que son autogenerados por el framework Vue.js en cada proceso de compilación y no es posible eliminarlo, con lo que se ha optado por no cambiar nada en la configuración de Vue y asumir este tipo de mensajes.
* Advertencias por el uso del atributo “date” dentro de las etiquetas “input” que se usan para la definición de los campos de los formularios: <input type=”date” …/>. En este caso el validador las desaconseja debido a que no están soportadas todavía por el 100% de los navegadores. En esta ocasión, también se asume esta advertencia debido a que se quiere mostrar en pantalla un control de tipo calendario que sea lo más sencillo posible, en lugar de usar algún plugin de terceros que necesitase la carga de una librería externa.

Tras la revisión del código y realizadas las correcciones que propone el W3C, el resultado es el siguiente:

| **Pruebas de validación de código (\*.html)** | | | | |
| --- | --- | --- | --- | --- |
| Login.html | * **OK** | |  |  |
| Contacto.html | * **Warnings: 1** | | |  |
| Addpet.html | * **Warnings: 4** | | |  |
| Deletepet.html | * **Warnings: 2** | | |  |
| Updatepet.html | * **Warnings: 5** | | |  |
| Licencia.html | * **Warnings: 5** | | |  |
| index.html | * **OK** |  | |  |

Tabla 45: Resultados del código html validado

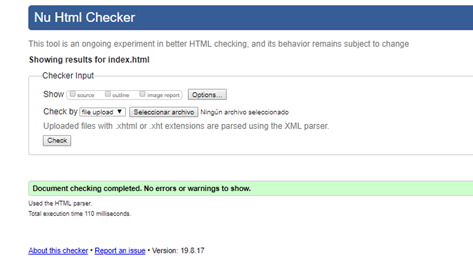


Ilustración 29: Validación W3C index.html

### Validación inicial (\*.css)

Los resultados de las pruebas realizadas a estos ficheros se muestran a continuación en el cuadro resumen.

|  |
| --- |
| **ATENCION**: No es objeto del presente análisis el realizar pruebas de validación a ficheros \*.css pertenecientes al framework Bootstrap 4. |

| **Pruebas de validación de código (\*.css)** | | |
| --- | --- | --- |
| main.css | * **Errores: 2** |  |

Tabla 46: Resultados de la validación de main.css

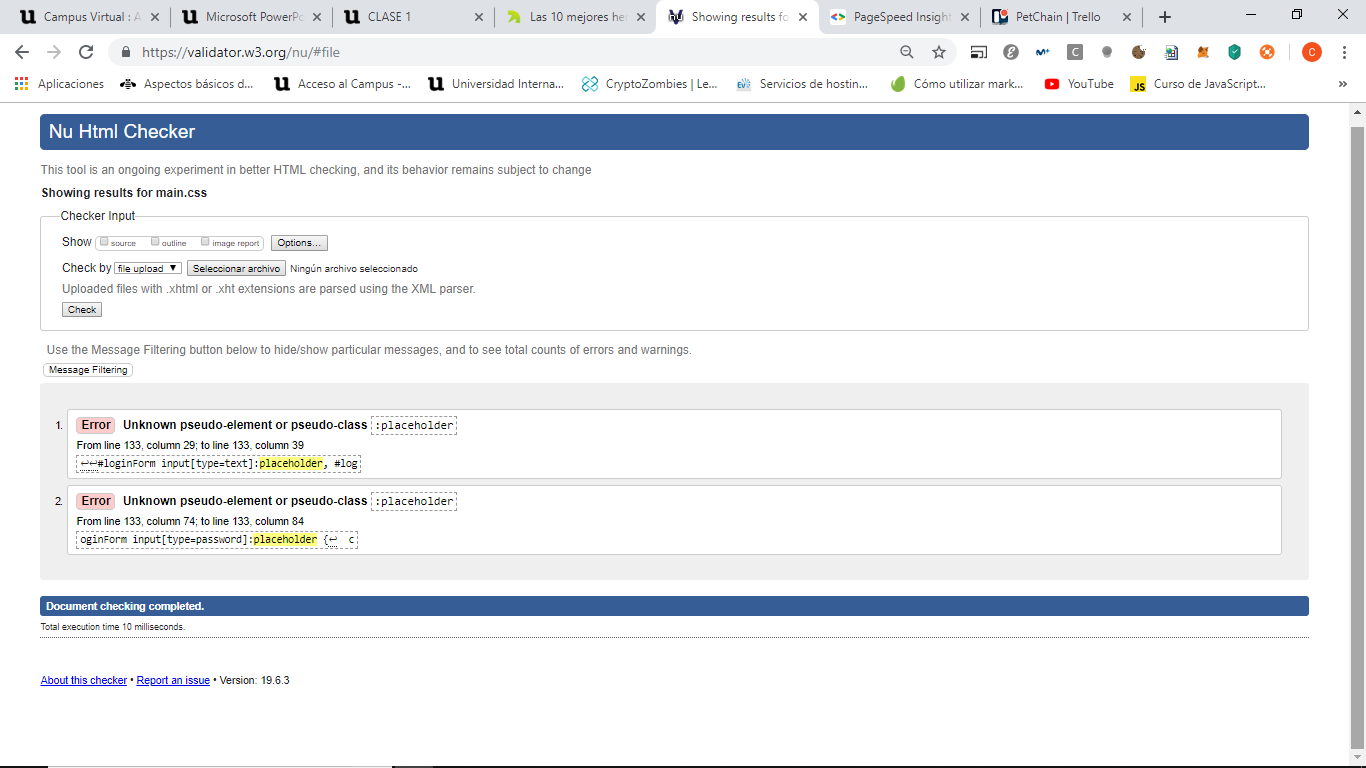


Ilustración 30: Resultados W3C de main.css

La pseudoclase “:placeholder”, empleada como modificador del comportamiento dentro de un css, no está definida en la especificación CSS3 del W3C. Sin embargo si está en fase de estudio para incorporarla a la definición que vendrá de CSS4, que en la actualidad está desarrollando el CSSWK[[14]](#footnote-14) , por este motivo, se permite incorporarla:

* Con el prefijo –ms, –moz o -webkit: para restringir su uso a navegadores de la familia Microsoft, Mozilla o Chrome, indicándole al validador que es una característica propia de ellos, y que por lo tanto la de por válida.
* Con el prefijo **::,** paradefinirla como un pseudoelemento en lugar de pseudoclase.

Modificando el estilo con la primera opción, obtenemos el resultado esperado.

| **Pruebas de validación de código (\*.css)** | | |
| --- | --- | --- |
| main.css | * **OK** |  |

Tabla 47: Resultado de la validación de main.css

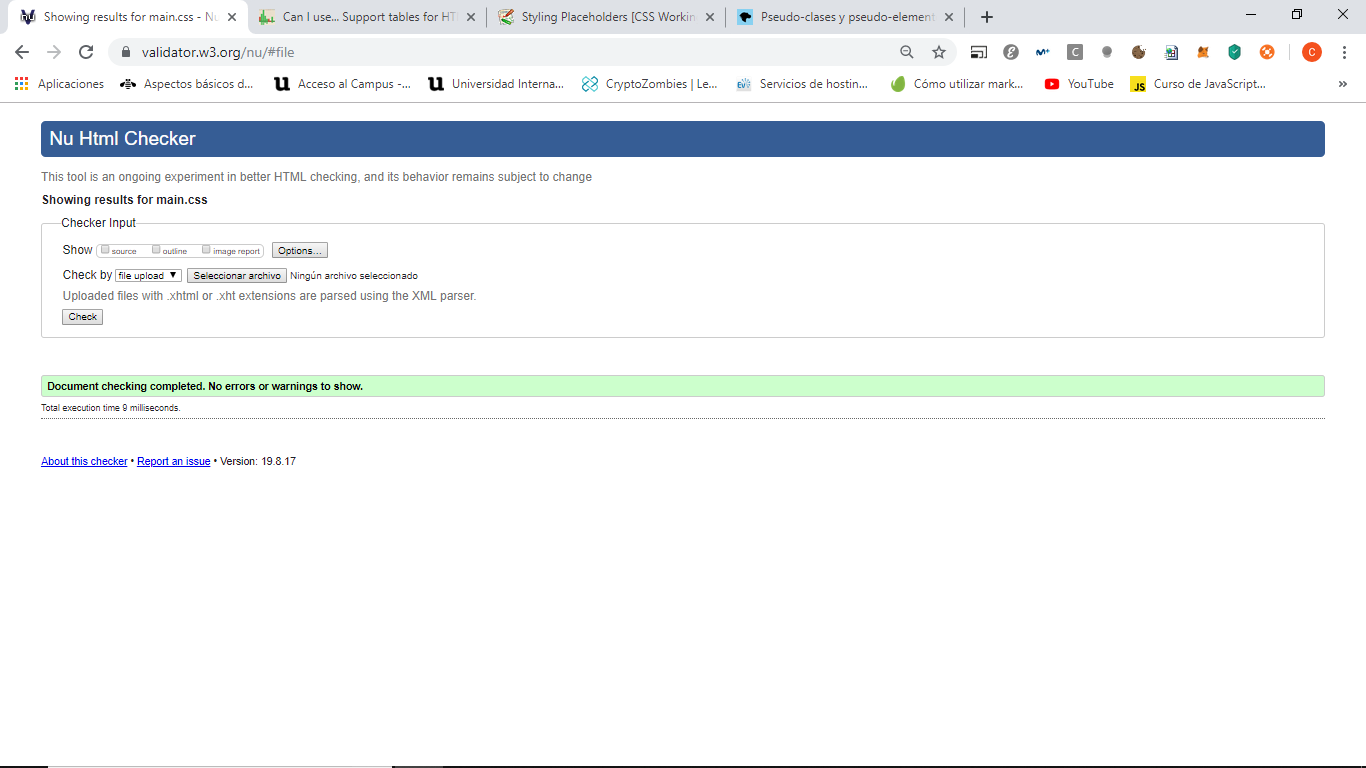


Ilustración 31: Resultado final del W3C de la validación de main.css

## Test Smartcontracts

POR DEFINIR Y REALIZAR: Tarjeta de Trello: https://trello.com/c/uyyB2IGM

## Validación de objetos JSON

Los objetos JSON empleados en PetChain deben de comportarse como elementos con una notación estándar estricta para favorecer su uso y su posible mantenimiento. Para validar que su estructura es correcta, se ha empleado una de las herramientas online más reputadas y que se ofrece desde la web [*https://jsonformatter.org/*](https://jsonformatter.org/).

La validación se ha realizado sobre los tres ficheros existentes en /src/data/, dando como resultado:

| **Pruebas de validación de objetos JSON (\*.json)** | | |
| --- | --- | --- |
| users.json | * **OK** |  |
| países\_ue.json | * **OK** |  |
| provincias\_es.json | * **OK** |  |

Tabla 48: Resultados validación de ficheros JSON

## Visualización en diferentes dispositivos

# Conclusiones y trabajo futuro.

POR REDACTAR: Tarjeta de Trello: https://trello.com/c/9Qk08d6Q

# Bibliografía

|  |
| --- |
| Bassagañas Galisteo, Jordi (2013). *MVC CodeIgniter para simpáticos newbies I (El gran curso de desarrollo de apps MVC nº 1)* España. Edición de Kindle. Amazon Media EU S.à r.l. |
| Benet, Juan (2014), *IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)* https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf |
| Cash, Mercury (2018). *Aplicaciones de la tecnología blockchain y su relación con los sistemas descentralizados*. https://blog.mercury.cash |
| Duran, Amador (2000). *Un Entorno Metodológico de Ingeniería de requisitos para Sistemas de la Información.* España: Universidad de Sevilla. |
| Gupta, Manav (2018). *Blockchain*. E.E.U.U: IBM |
| Modi, Ritesh (2018) *Solidity Programming Essentials. A beginner's guide to build Smartcontracts for Ethereum and blockchain*. UK: Packt Publishing. |
| Somerville, Ian (2009). *Ingeniería de Software*. México: Pearson educación. |
| Venter, Ryan (2016). *The Modern Ethereum*. UK: Amazon Media EU S.à r.l. |

# Glosario de términos

*A*

API

Application Program Interface 8, 9, 10

11, 45

*B*

*C*

CCAA

Comunidades Autónomas 10, 12

CDN

Content Delivery Network 63

CSSWG

CSS Working Group 75

CSS3

Cascade Style Sheet ver.3 17

*D*

DNI

Documento nacional de identidad 17

*E*

ECMA

European Computer Manufactures

Association 17

ETH

Ethers 6, 18, 19,

26, 30

*F*

FRQ

Functional Requirement 25, 28

29, 30, 31, 32

*G*

*H*

HTML5

Hiper Text Markup Language 17

http

hyper text transfer protocol 10, 11,

43, 47

https

hyper text transfer protocol

secure 10

*I*

IPFS

Inter Planetary File System 1, 2,14,

15, 19, 34,35

*J*

JSON

Javascript Object Notation 4, 7,8,

11, 17, 30, 35, 49

JSX

Javascript XML 61

JWT

Javascript Web Token 17, 30

*K, L*

*M*

MVC

Model View Controller 17

Mac OS

Macintosh Operating System 43, 44

*N*

NFR

No Functional Requirement 33, 34,

35, 36

NoSQL

No Structured Query Language 16,17

npm

Node package manager 41, 43, 45

*O*

*P*

P2P

Peer to peer 12

PC

Personal Computer 18

PHP

HyperText Preprocessor 48

*Q*

*R*

REM

Requirements Management 47

REST

Representational State Transfer 10

RIACA

Registro de Animales de Compañía de Aragón 9

RPC

Remote Procedure Call 10, 44

*S*

SIACAM

Sistema de Identificación Animal de Castilla la Mancha 9

SIACYL

Sistema de Identificación Animal de Castilla y León 9

SIAMEL

Sistema de Identificación Animal de la Ciudad de Melilla 9

SIAMU

Sistema de Identificación Animal de la Región de Murcia 9

SPA

Single Page Applicaction 59

SOAP

Simple Object Access Protocol 10

*T*

TFG

Trabajo de fin de grado 15

*U*

UNIR

Universidad Internacional de la Rioja 1

URL

Uniform Resource Locator 28, 57, 58

*V*

*W*

W3C

World Wide Web Consortium 35, 51

WAI

Web Accesibility Initiative 56

WCAG

Web Consortium Accesibility

Guidelines 48

*X*

XML

Extensible Markup Language 10

*Y, Z*

# Referencias

Iconos y elementos gráficos: Icon made by [Freepik](http://www.freepik.com/) from [www.flaticon.com](http://www.flaticon.com/)

https://lagisteria.com/2019/02/06/mapa-gratuito-espana-ccaa-descarga-gratis/

1. [*http://petshopsmagazine.com/mascotas/censo-mascotas-espana/*](http://petshopsmagazine.com/mascotas/censo-mascotas-espana/) [↑](#footnote-ref-1)
2. *www.siacyl.org* [↑](#footnote-ref-2)
3. [*https://www.hyperledger.org/*](https://www.hyperledger.org/) [↑](#footnote-ref-3)
4. [*https://b3i.tech/home.html*](https://b3i.tech/home.html) [↑](#footnote-ref-4)
5. [*https://alastria.io/*](https://alastria.io/) [↑](#footnote-ref-5)
6. *Trello web site*: [*https://trello.com*](https://trello.com/) [↑](#footnote-ref-6)
7. *npm web site:* [*https://www.npmjs.com/*](https://www.npmjs.com/) [↑](#footnote-ref-7)
8. *Truffle website: https://truffleframework.com* [↑](#footnote-ref-8)
9. *Ganache website:* [*https://truffleframework.com/ganache*](https://truffleframework.com/ganache) [↑](#footnote-ref-9)
10. [*https://github.com/ethereum/wiki/wiki/JavaScript-API#web3-javascript-app-api-for-02xx*](https://github.com/ethereum/wiki/wiki/JavaScript-API#web3-javascript-app-api-for-02xx) [↑](#footnote-ref-10)
11. [*https://getbootstrap.com*/](https://getbootstrap.com/) [↑](#footnote-ref-11)
12. [*http://www.lsi.us.es/descargas/descarga\_programas.php?id=3*](http://www.lsi.us.es/descargas/descarga_programas.php?id=3) [↑](#footnote-ref-12)
13. *Sección 3.8.2. Requisitos funcionales* [↑](#footnote-ref-13)
14. [*https://wiki.csswg.org/*](https://wiki.csswg.org/) [↑](#footnote-ref-14)