INGENIERÍA INFORMÁTICA

Comparación de rendimiento entre bases de datos Relacionales, NoSQL y Blockchain.

Comparación de rendimiento entre MySQL, Cassandra y Ethereum.

Performance comparison among Relational, NoSQL databases and Blockchain.

Performance comparison among MySQL, Cassandra and Ethereum.

Realizado por
Rafael Ordóñez Alba

Tutorizado por
Enrique Soler Castillo

Departamento
Lenguajes y sistemas informáticos

UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO 2020

UNIVERSIDAD
DE MÁLAGA

E.T.S. INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA INFORMÁTICA

Comparación de rendimiento entre bases de datos Relacionales, NoSQL y Blockchain.

Comparación de rendimiento entre MySQL, Cassandra y Ethereum.

Performance comparison among Relational, NoSQL databases and Blockchain.

Performance comparison among MySQL, Cassandra and Ethereum.

Realizado por
Rafael Ordóñez Alba

Tutorizado por
Enrique Soler Castillo

Departamento
Lenguajes y ciencias de la computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, FEBRERO DE 2020

UNIVERSIDAD
DE MÁLAGA

E.T.S. INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Fecha defensa:

Resumen

Debido a la aparición de una nueva tecnología como es el Blockchain y a su capacidad de almacenar datos de forma segura y descentralizada se nos plantean numerosas dudas sobre su utilidad y rendimiento con respecto a las tradicionales formas de almacenamientos de datos como son las bases de datos relacionales y no relaciones.

Por ello, en este TFG pretendemos estudiar su viabilidad a la hora de usarlo en un proyecto real comparándolo con las formas de almacenamiento tradicionales pudiendo así comparar su rendimiento, facilidad de aprendizaje, versatilidad, capacidad de adaptación, coste de implementación…

Para ello, crearemos 3 páginas web completamente funcionales, la primera de ellas en MySQL (base de datos relacional), la segunda en Cassandra (base de datos no relacional) y, por último, utilizando Ethereum (tecnología Blockchain).

La página web creada simulará un caso de uso en concreto, se trata del ciclo de vida que siguen los animales de la cadena alimentaria desde que nacen hasta que terminan en el supermercado. Se almacenarán todos los registros sobre alimentación, inspecciones de sanidad, transportes… que se lleven a cabo en este proceso.

Una vez creadas las páginas web realizaremos diferentes pruebas de rendimiento tanto en inserciones como en consultas de datos en cada una de las páginas web, midiendo el tiempo que tardan en realizar estas operaciones.

Palabras clave: Blockchain, bases de datos, rendimiento, MySQL, Cassandra, Ethereum, base de datos relacional, base de dato no relacional.

Abstract

Due to the emergence of a new technology such as Blockchain and to its capacity to store data in a secure and decentralized way, we have doubts about its usefulness and performance with respect to traditional forms of data storage such as relational and non-relational databases.

Therefore, in this project we intend to study its viability using it in a real project of comparison with traditional storage methods, comparing its performance, ease of learning, versatility, adaptability, cost of implementation ...

For this, we will create 3 fully functional web pages, the first one in MySQL (relational database), the second one in Cassandra (non-relational database) and, finally, using Ethereum (Blockchain technology).

The website created will simulate a specific use case, it is the life cycle that the animals of the food chain follow from birth until they end up in the supermarket. All records on food, health inspections, transports ... that are carried out in this process will be recorded.

Once you create the web pages, we will execute different performance tests on insertions such as data queries on each of the web pages, measuring the time it takes to realize these operations.

Keywords: Blockchain, databases, performance, MySQL, Cassandra, Ethereum, relational databases, non-relational databases

Índice

1

Introducción

1.1 Importancia de las bases de datos

Hoy en día, cualquier aplicación de escritorio, de móvil o web gira en torno a la inserción o a la consulta de datos, por ello ser realiza una gran cantidad de operaciones en la base de datos. Esto es un factor determinante ya que una base de datos con un rendimiento pobre puede influir notablemente en la lentitud de la aplicación, afectando así a la experiencia de usuario negativamente.

En algunos casos puede incluso ser decisivo ya que se necesita que estas operaciones se realicen casi instantáneamente ya que el hecho de retrasarse conllevaría el mal funcionamiento de la aplicación o la invalidez de la misma. Esto sobre todo puede ser determinante en aplicaciones de uso empresarial.

Sin embargo, la velocidad no es el único factor determinante, encontramos otros factores que pueden ser incluso más relevantes dependiendo del uso de la aplicación. Es por ello, por lo que la elección de un tipo de base de datos debe de estar condicionada por las necesidades de la aplicación.

La seguridad de los datos es un factor fundamental ya que hay datos que necesitan ser protegidos de posibles filtraciones ya que pueden contener información personal o ser de suma importancia, a su vez, también es necesario garantizar la integridad de los mismos ya que de nada sirve encontrar los datos modificados.

Es por ello, por lo que pretendemos comparar las ventajas y desventajas que nos ofrecen las bases de datos más tradicionales con respecto al uso de la tecnología blockchain para el almacenamiento de datos.

1.2 Uso en empresas

Hoy en día, la tecnología blockchain apenas ha sido empleada para el uso empresarial debido a su reciente aparición. Sin embargo, algunas empresas como Carrefour sí que han empezado a utilizarla.

El caso en concreto para el cual lo utiliza Carrefour es muy interesante, ya que utiliza una red blockchain para garantizar la calidad del pollo. Esta red fue lanzada en el año 2018 y resulta muy interesante ya que se pueden ver todos los datos del pollo del cual ha salido el producto, desde la fecha de nacimiento y de muerte hasta la alimentación que ha llevado. En la figura 1.1 podemos ver un ejemplo de cómo Carrefour muestra la información.

Figura 1.1: Información del pollo Carrefour.

Al ser una red blockchain garantiza la veracidad e integridad de los datos ya que esto es una de las propiedades que tiene como ya veremos más adelante. Carrefour plantea desplegar una red similar para todos sus productos alimenticios para el año 2022 [1].

Como hemos mencionado antes, la redes blockchain apenas son utilizadas, la mayoría de las empresas opta por bases de datos más convencionales. Como podemos ver en la figura 1.2 las bases de datos más utilizadas son las relacionales, en concreto la que más MySQL.

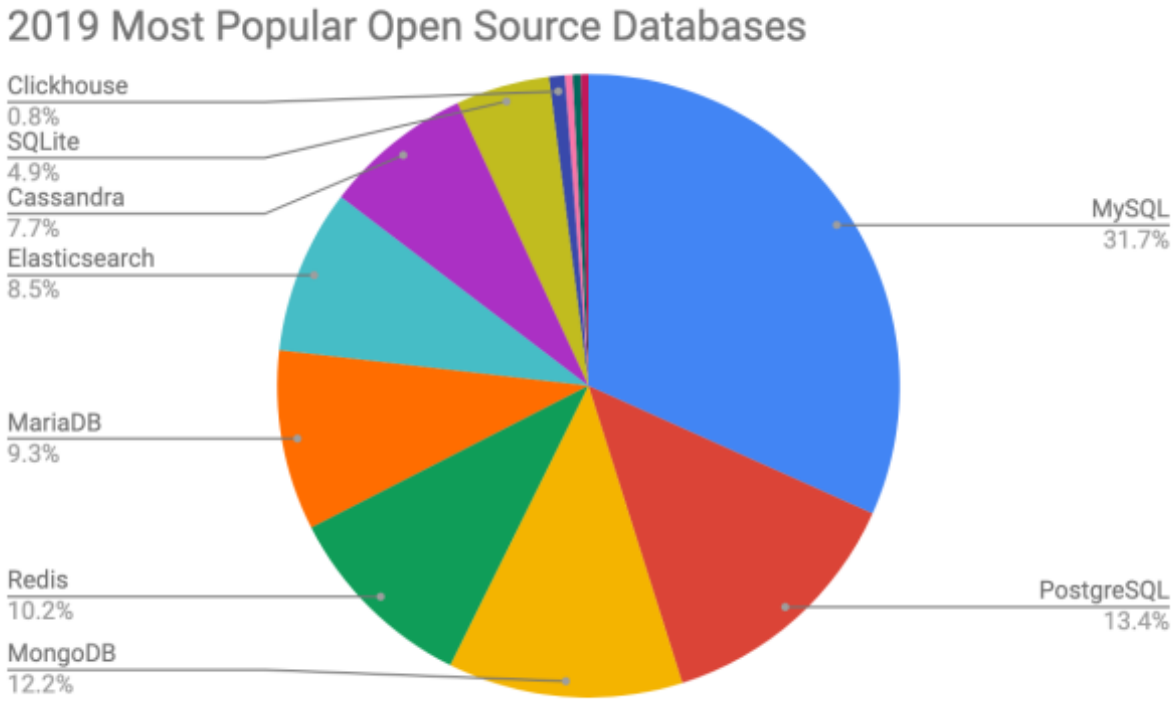

Figura 1.2: Bases de datos de código abierto más usadas.

Recuperado el 16/01/2020 de https://dzone.com/articles/2019-open-source-database-report-top-databases-pub .

Como podemos observar las más utilizadas son las bases de datos tradicionales. Esto puede deberse a facilidad de encontrar gente formada en estas bases de datos y a la negativa de querer realizar un amplio desembolso en sustituirlas por otra base de datos. Aunque bien es cierto, que cada base de datos ofrece sus ventajas e inconvenientes como ya analizaremos en profundidad.

1.3 Desarrollo y objetivos

El objetivo de este TFG no es simplemente basarnos en puras pruebas de rendimiento, sino que pretendemos obtener al final unas conclusiones que nos ayuden a seleccionar el tipo de base de datos que más nos convenga dependiendo del uso que le vayamos a dar a la aplicación y de las condiciones que se encuentre.

Para ello, documentaremos todo tipo de problemas que no hayan surgido durante el desarrollo e implementación de cada una de las 3 páginas web, ya sean problemas de aprendizaje, incompatibilidades de software, librerías o frameworks y otro tipo de problemas que puedan surgir.

También, tendremos en cuenta cada una de las ventajas e inconvenientes que cada una de ellas tiene, ya sean capacidad de estructurar los datos, seguridad e integridad de los datos, escalabilidad, versatilidad… Evidentemente, todo ello será complementado con las respectivas pruebas de rendimiento ya sea a la hora de insertar datos o a la hora de consultar datos.

La idea es poder obtener una guía lo más objetiva posible la cual nos permita poder escoger una base de datos en concreto de acorde con nuestras necesidades.

2

Conceptos previos

2.1 Bases de datos

El concepto de base de datos surgió con la informática, se trata de un conjunto de información organizada de forma que un programa de ordenador pueda seleccionar de forma rápida el conjunto de datos que necesite [2]. Es por ello por lo que podemos resumir que una base de datos se trata de un sistema de archivos electrónicos.

Los programas interactúan con las bases de datos ya sea pudiendo consultar información o pudiéndola almacenar. A la hora de la historia han surgido diversos tipos de bases de datos atendiendo a su modelo, sin embargo, la que van a ser motivo de estudio serán las relacionales, las no relacionales y las "bases de datos" usando la tecnología blockchain que, aunque no sean bases de datos como tal pueden actuar realizando la misma función.

Aunque esta no es el único tipo de clasificación de bases de datos, ya que existen otros criterios, como ser puede ver en la figura 2.1.

Figura 2.1: Otra clasificación bd
Recuperado el 16/01/2020 de http://e-ducativa.catedu.es/44700165/aula/archivos/repositorio/1000/1080/html/12_tipos.html .

Dependiendo de la variabilidad de los datos hay bases de datos pensadas para almacenar y registrar datos (estáticas) mientras que hay otras enfocadas a la actualización, edición y eliminación de datos (dinámicas).

Por otro lado, podemos dividirlas según el contenido que almacenan encontramos tres tipos, las bibliográficas, los directorios y las bibliotecas científicas.

En las bibliográficas almacenamos información en forma de cifras, números y datos rápidos de una fuente primaria. Pueden contener un resumen de la publicación, pero no textos grandes. Suelen tener información sobre resultados de laboratorio, fichas de clientes o registro de libros [3].

En los directorios se almacenan índices donde encontrar la información de una forma rápida como si se tratase de una guía telefónica. Es posible diferenciar las de directorios empresariales y los directorios personales.

Las de artículos científicos se caracterizan por almacenar datos procedentes de disciplinas científicas ya sean biológicas, químicas, médicas, matemáticas… Su uso es muy importante cuando se requiere acceso a información muy específica y concreta.
En último lugar, hay bases de datos que podemos dividir según su forma de almacenamiento, por un lado, se encuentra las centralizadas en las cuales se encuentra toda la información en un único nodo y por otro lado tenemos las distribuidas, en las cuales la información se encuentra distribuida entre todos los nodos.

2.2 Bases de datos relacionales

A primera hora cada aplicación almacenaba sus datos en una estructura propia creada para tal aplicación, entonces cuando otra aplicación u otro desarrollador quería usar esos datos necesitaba conocer exactamente cómo funcionaba esa estructura que en el caso de cada aplicación era diferente. Estás estructuras eran ineficientes, es por ello, por lo que surgieron las bases de datos relacionales.

Estas bases de datos proporcionaron un modelo estándar de representar y consultar los datos, el cual, pudiese usar cualquier aplicación sin necesidad de aprender a usar una nueva estructura.

Las bases de datos relacionales son un tipo de bases de datos en las cuales se almacenan datos de forma estructurada y relacionados entre sí. Están basadas en el modelo relacional, poseen tablas y cada fila es un registro que tiene un único ID llamado clave. La tabla posee atributos en cada columna y cada registro es un valor para cada uno de estos atributos.

Al ser un modelo relacional la información de las estructuras de datos lógicas se encuentra separadas mediante estructuras de almacenamiento físicas. De esta forma es posible modificar la estructura de almacenamiento física sin que afecta a la estructura lógica. Por ejemplo, podemos cambiar en lo nombre de un archivo en la base de datos que no se cambia el nombre de las tablas almacenadas en él [4].

Estas bases de datos usan el lenguaje de consulta estructurado (SQL) tanto para escribir como para consultar datos. SQL está basado en el algebra relacional, proporciona un lenguaje matemático coherente el cual ayuda a mejorar el rendimiento de todas las consultas.

El modelo relacional es ideal para mantener la coherencia de datos entre aplicaciones ya que garantiza que varias instancias de la base de datos tengan los mismos datos al mismo tiempo. Es decir, si se actualiza un dado desde una instancia, este debe de verse instantáneamente actualizado si se consulta desde otra instancia.

Las bases de datos relacionales utilizan reglas y políticas de empresa a un nivel muy detallado, una de sus propiedades es la atomicidad miente la cual se define un conjunto de elementos que forman parte de una transacción.

Las bases de datos relacionales más usadas son MS SQL Server, MySQL, Mariadb, Oracle, SQLite y PostgresSQL [5].

2.3 Bases de datos no relacionales

Las bases de datos no relacionales surgen con la llegada de la web 2.0 ya que cada vez las aplicaciones necesitan almacenar más información y recuperarla más rápido. Un ejemplo de esto es el crecimiento de las redes sociales provocando un aumento exponencial de la cantidad de datos que necesitan manejar.

Aparecen como solución para las bases de datos relacionales en las situaciones que estas tienen problemas de escalabilidad y rendimiento. Así, permiten la cita de miles de usuarios concurrentemente con millones de consultas diarias sin que el rendimiento se vea altamente perjudicado.

Estas bases de datos no siguen el modelo entidad-relación, ni utilizando una estructura de datos en forma de tabla, sino que utilizan otros formatos.

Las bases de datos con formato clave/valor utilizan una colección de pares clave/valor para almacenar los datos como se puede ver en la figura 2.2, también se le puede denominar diccionario o hash. La clave es un identificador único que permite localizar el valor asociado. Se puede almacenar cualquier tipo de datos desde documentos, imágenes, archivos… Ejemplos: Cassandra, BigTable o Hbase.

Clave	Valor
Juan	(123) 456-7890
Pedro	(234) 567-8901
Maria	(345) 678-9012
Francisca	(456) 789-0123

Figura 2.2: Colección clave/valor

Recuperado el 18/01/2020 de https://www.tecnologias-informacion.com/clave-valor.html.

Otro formato son las documentales en las cuales la estructura es un simple JSON o XML donde se utiliza una clave única para cada registro. Por lo tanto, aparte de realizar búsquedas similares a las de por clave-valor, también se realizan búsquedas más avanzadas. Ejemplos: MongoDB o Couch DB.

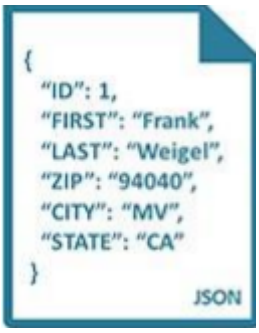

Figura 2.3: Formato JSON bd

Recuperado el 18/01/2020 de https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf.

Las bases de datos con formato en grafo la información se representa mediante nodos y las relaciones entre los datos como aristas entre los nodos. Se puede utilizar algoritmos de la teoría de grafos para recorrerla. Ejemplos: Neo4j, InfoGrid o Virtuoso.

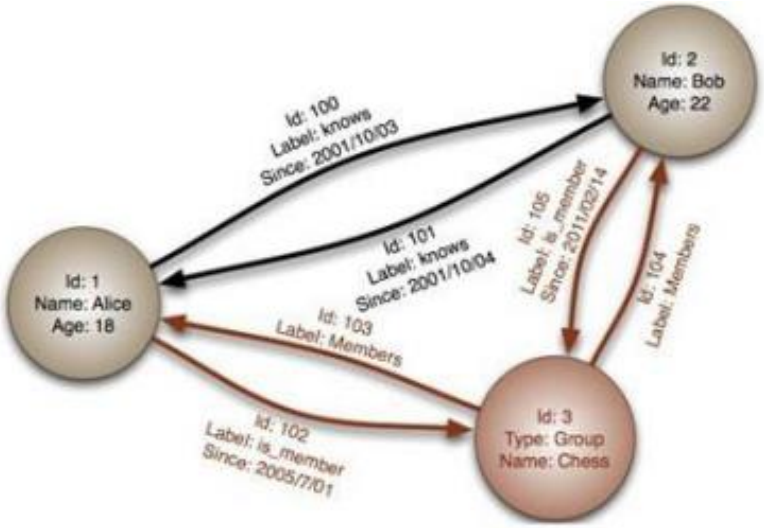

Figura 2.4: Formato grafo bd

Recuperado el 18/01/2020 de https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf .

Por último, encontramos las bases de datos con formato orientado a objetos, en las cuales la información se representa mediante objetos de forma similar a como se realiza en los lenguajes de programación orientados a objetos (Java, C# …). Ejemplos: Zope, Gemstone o DB4o.

En las bases de datos no relaciones no se usa el lenguaje de consultas SQL y se puede utilizar una arquitectura distribuida a diferencia de las relacionales que por lo general era una arquitectura centralizada. También nos permiten ejecutarse en una máquina con pocos recursos pudiendo manejar así una mayor cantidad de datos sin generar cuello de botella. A su vez nos proporcionan una mayor escalabilidad horizontal pudiendo añadir más nodos en cualquier momento a la base de datos.

Las principales bases de datos no relacionales más utilizadas son Cassandra, Redis, MongoDB y CouchDB.

2.4 Tecnología Blockchain

Blockchain surgió en 2008, dentro del proyecto Bitcoin. Apareció de la combinación de la tecnología de redes existente (P2P) con técnicas criptográficas avanzas. De ahí el término criptomoneda [6]. Su principal objetivo era el tráfico de criptomonedas garantizando seguridad, transparencia y privacidad entre los usuarios.

Las bases de datos con blockchain son un tipo de bases de datos distribuidas por todos los nodos que se encuentran en la red. A continuación, dejamos una cita en la cual se explica hasta qué punto es descentralizada la tecnología blockchain: “Las blockchains están políticamente descentralizadas (nadie las controla) y arquitectónicamente descentralizadas (no hay un punto de fallo central infraestructural) pero están lógicamente centralizadas (hay un estado comúnmente acordado y el sistema se comporta como una sola computadora).” (Vitalik Buterin, Fundador y desarrollador Ethereum.).

Como podemos ver al tratarse de un sistema descentralizado no nos encontramos con los típicos problemas de escalabilidad que tienen los sistemas centralizados, por lo tanto, tiene un potencial de escalabilidad mayor. A su vez, tanto el control de acceso a la escritura como el de lectura también se encuentran descentralizados, es decir, se pueden realizar transacciones seguras sin la necesidad de un tercero de confianza. Un ejemplo de esto sería cuando una persona (A) desea transferirle x dinero a una persona (B), normalmente la lógica dice que se necesita un intermediario para la gestión (el banco), sin embargo, con la tecnología blockchain no haría falta este intermediario.

Ante el caso anterior, se nos plantea la duda de cómo es esto posible, debido a que, si ahora una de las dos partes niega el haber realizado la operación, ¿cómo podemos saber si esto es cierto? Esto es posible gracias a que la tecnología blockchain posee un carácter incorruptible, es decir, cada transacción es inmutable y no puede ser eliminada o modificada. A su vez, aunque no pueda ser modificado u eliminado por nadie, sí que es transparente para ambas partes de las entidades que han realizado la transacción, cualquiera de ambas partes puede verla públicamente.

Al no necesitar intermediarios este proceso se puede realizar en cualquier momento las 24 horas del día, ya que no se necesita de ninguna entidad que lo verifique. La información se transmite y se guarda de forma automática, evitando tener que esperar al intermediario. Los datos que se encuentran en esta red carecen de error, ya que se encuentran en continua verificación por todas las personas que forman la red.

Como hemos podido ver nos encontramos ante un red privada, segura, transparente y descentralizada. Pero ¿cómo es esto posible? A continuación, vamos a hablar sobre cómo funcionan estas redes.

Vamos a seguir analizando el ejemplo anterior en el cual una persona A le quiere enviar X dinero a una persona B. Ahora vamos a suponer que estas personas forman parte de una red blockchain, en la cual como hemos visto anteriormente el resto de usuario se encargan de verificar cualquier operación que se realiza. Cuando A quiere enviar X dinero a B se lo comunica al resto de usuarios de la red, sin que estos sepan las identidades ni de A ni de B, estos usuarios comprueban que A tiene suficiente dinero y anotan la transacción. Con el tiempo tras muchas transacciones el bloque se llena, entonces llega el momento de la validación o sellado definitivo. Ahí es donde entran en juego los mineros, que realizan una serie de cálculos con el objetivo de que los bloques queden registrados permanentemente en la cadena de bloques. Una vez registrados quedan enlazados y no pueden modificarse sin alterar a los demás bloques que están enlazados [10].

2.4.1 Funcionamiento cadena de bloques

Una cadena de bloques es una secuencia de bloques enlazados entre sí. Dentro de cada uno de estos bloques se almacenan una cantidad de registros (realizándoles el hash), información referente al bloque y el bloque anterior al cual está vinculado,

haciendo referencia a través del hash del bloque. En la figura 2.5 podemos ver un ejemplo de cadena de bloques.

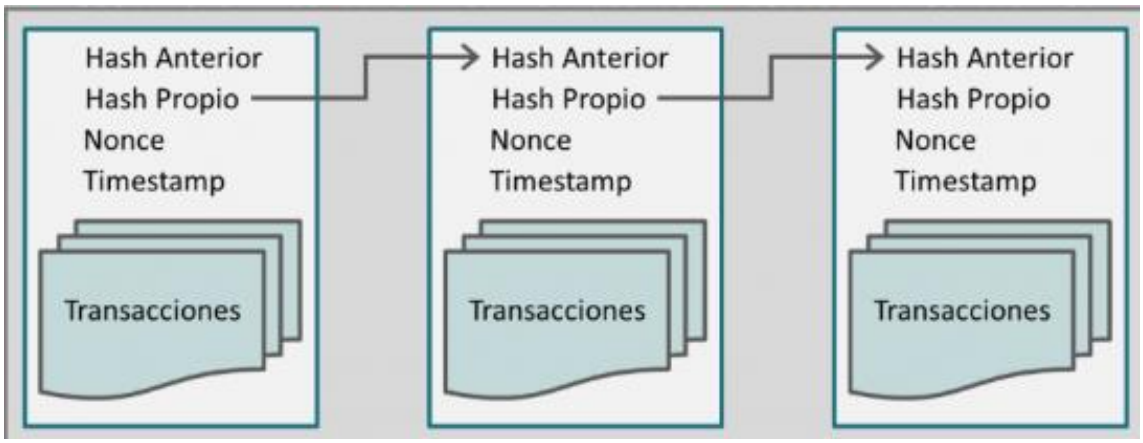

Figura 2.5: Cadena de bloques

Recuperado el 19/01/2020 de https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/ .

Un hash es una especie de identificador único de un bloque de datos, calculado mediante una función criptográfica. Vendría a ser por así decirlo como la huella digital de un bloque da datos. Esta función condensa los datos reduciendo en gran medida su longitud. Al tratarse de un identificador único, dos entradas distintas no pueden dar la misma salida, de hecho, un pequeño cambio en la entrada provoca una salida completamente distinta.

La función de los mineros es encontrar el Hash que le corresponde a un bloque para así poderlo añadir a la cadena, este es un proceso costoso y relativamente lento, de ahí que para el uso de redes públicas blockchain se requiera pagar por cada operación. Una vez es obtenido el hash, el resto de los miembros de la red verifican que sea correcto [11].

Para poder romper la seguridad de una cadena de bloques sería necesario modificar los bloques en al menos un 51% de los usuarios de la red, debido a que se encuentra distribuida por todos los miembros de la red. Por lo tanto, a más grande sea la red, mayor será la seguridad [12].

2.4.2 Usos del blockchain

Esta tecnología puede ser aplicada en numerosos campos debido a su alta seguridad y capacidad de mantener la información de forma intacta. Pudiendo suprimirse el uso de muchos intermediarios, suponiendo así cierta ventaja económica. Aunque aún apenas ha tenido aplicaciones reales debido a su reciente surgimiento.

Puede tener varios usos en el ámbito de la salud, protegiendo así el historial de los pacientes, pudiendo así consultarlo cualquier médico independientemente del centro de saludo donde se encuentre (público o privado). También se podría usar para verificar medicamentes en las farmacias y así evitar posibles falsificaciones.

Por otro lado, como ya vimos que estaba utilizando Carrefour, se puede utilizar en el ámbito alimenticio garantizando así la calidad y proveniencia de los productos, evitando así cualquier posible estafa.

Uno de los usos más importantes puede ser la supresión de los bancos y el poder realizar compras y ventas sin la supervisión de los mismos a través de internet o la firma de documentos, acuerdos… sin la necesidad de un notario.

Como vemos hay múltiples usos que probablemente veremos a lo largo de los próximos años y que en un futuro serán parte de nuestro día a día.

3

Análisis

3.1 Introducción

Las 3 páginas web que vamos a crear en este TFG con los distintos gestores de almacenamiento giran en torno a un caso de uso en común. Con caso de uso, nos referimos a la situación que hemos escogido para analizar.

El caso de uso que vamos a desarrollar se trata del ciclo de vida de los animales que se encuentran en una granja, similar al sistema implementado en Carrefour con los pollos, donde se pretende dejar constancia de todo el proceso que vayan recibiendo los distintos animales que se encuentren en la granja.

Analizaremos la trayectoria del animal de la granja desde que llega hasta que es sacrificado en el matadero, almacenando cada paso que influya en el mismo. Se almacenará información acerca de la alimentación del animal, de su transporte, de las inspecciones que este reciba y de su sacrificio. También tendremos almacenado a los distintos usuarios que intervienen en el proceso, ya que contamos con un sistema de roles, que dependiendo del que tengas asignado, podrás realizar ciertas operaciones, es decir, si eres granjero, no vas a dejar constancia sobre transportes, ya que tu rol no es transportista.

También almacenaremos información sobre los tipos de animales y los animales, denominados como tipo de objeto y objeto, e información sobre las 4 operaciones habladas anteriormente, además de las distintas características que puedan tener los distintos animales de cada especie con sus valores respectivos.

3.2 Casos de uso

Caso de uso	Registrarte en la página web
Descripción	El usuario podrá registrarse en la página web
Precondición	
Postcondición	El usuario quedará registrado en MySQL

Caso de uso	Iniciar sesión
Descripción	El usuario podrá iniciar sesión en la página web
Precondición	El usuario debe de estar registrado anteriormente
Postcondición	Acceder a ventana home y marcar en un campo que el usuario ha iniciado sesión en MySQL

Caso de uso	Insertar un animal
Descripción	Se insertará en MySQL un nuevo animal con sus características
Precondición	Ser un usuario administrador
Postcondición	El nuevo animal aparecerá en MySQL

Caso de uso	Insertar alguna operación sobre un animal
Descripción	Se insertará en MySQL algún tipo de operación llevada a cabo sobre alguno de los animales de la base de datos
Precondición	Ser un usuario registrado y que haya animales en MySQL
Postcondición	La operación aparecerá en MySQL

Caso de uso	Consultar operaciones realizas sobre los animales
Descripción	Poder realizar consultas sobre las operaciones que ha realizado un usuario sobre los animales
Precondición	Ser un usuario registrado y que haya operaciones en MySQL
Postcondición	Se mostraran las operaciones en una tabla

Caso de uso	Ver animales transportados
Descripción	Poder ver que animales han sido transportados en una operación de tipo transporte

Precondición	Ser un usuario transportista o administrador y que haya transportes en MySQL
Postcondición	Se mostraran en una tabla los animales que han sido transportados

Caso de uso	Ver características de un animal
Descripción	Poder ver las características de algún animal sobre el que se hay realizado una operación
Precondición	Ser un usuario registrado y que haya operaciones en MySQL
Postcondición	Se mostrarán las características de un animal en una tabla

Caso de uso	Realizar una simulación
Descripción	Realizar una simulación generando un número establecido de inserciones y consultas de forma aleatoria
Precondición	Ser un usuario administrador
Postcondición	Se mostrará el tiempo que ha tardado la ejecución de las inserciones y consultas

3.3 Diagramas de casos de uso

A continuación, vamos a mostrar los diagramas de casos de uso de los diferentes actores dentro de la página web una vez iniciada sesión. Vamos a separarlo según el tipo de usuarios que sean y la cantidad de funciones que puedan realizar.

En primer lugar, vamos a enseñar el del administrador, el cual es el que puede realizar un mayor número de funciones. Puede ver la vista de inicio, insertar cualquier tipo de operación, consultar todas las operaciones realizadas por cualquier usuario, incluidos los transportes y las características de los animales sobre las que se realizan. También podrá realizar dos funciones exclusivas de este rol como son el poder insertar animales con sus respectivas características según el tipo animal que sean y el poder realizar una simulación con los parámetros del número de inserciones y consultas deseado.

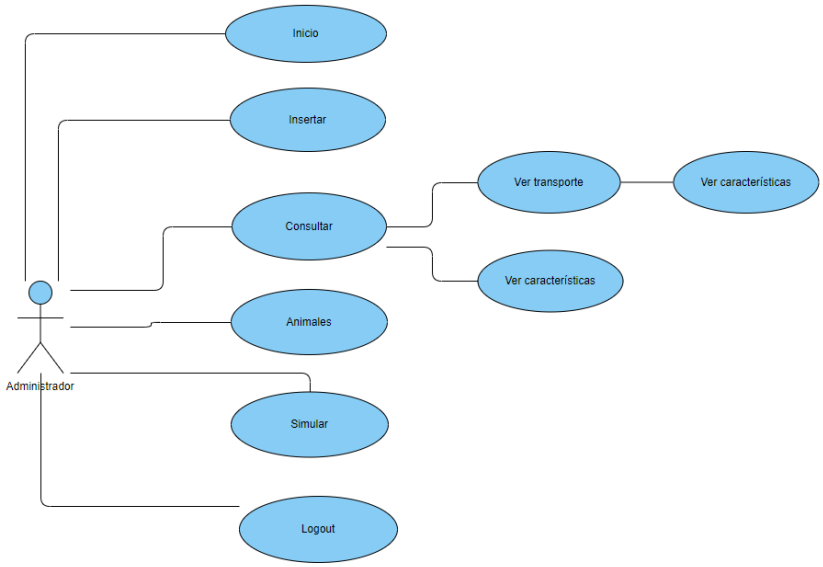

Figura 3.13: Diagrama caso de uso administrador.

En segundo lugar, vamos a mostrar el diagrama de caso de uso del usuario transportista, el cual, puede ver la vista de inicio, insertar transportes realizados por él y consultar sus propios transportes, viendo así los animales que ha transportado y sus características.

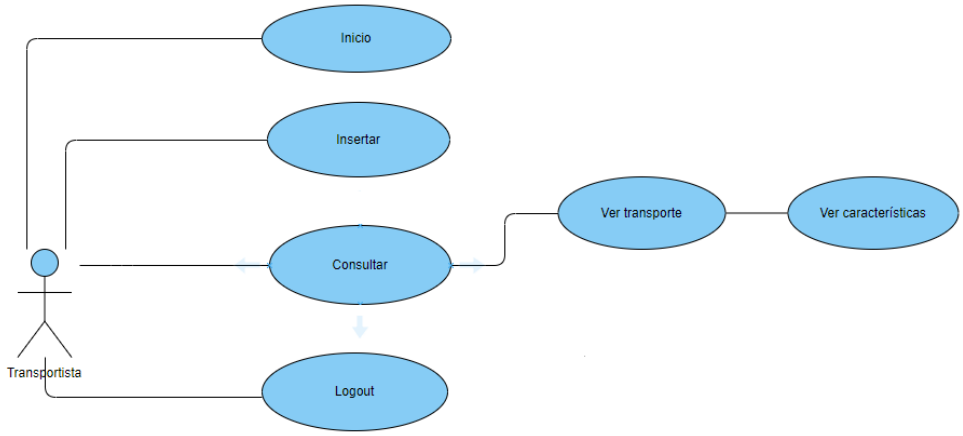

Figura 3.14: Diagrama caso de uso transportista.

Por último, encontramos el diagrama de caso de uso del resto de usuarios (matarife, alimentador e inspector), estos son muy similares al del transportista con la única diferencia de que cada uno podrá insertar y consultar datos sobre la operación de la que esté encargado y que pueden acceder directamente desde la vista consultar a las características del animal al cual le han realizado la operación.

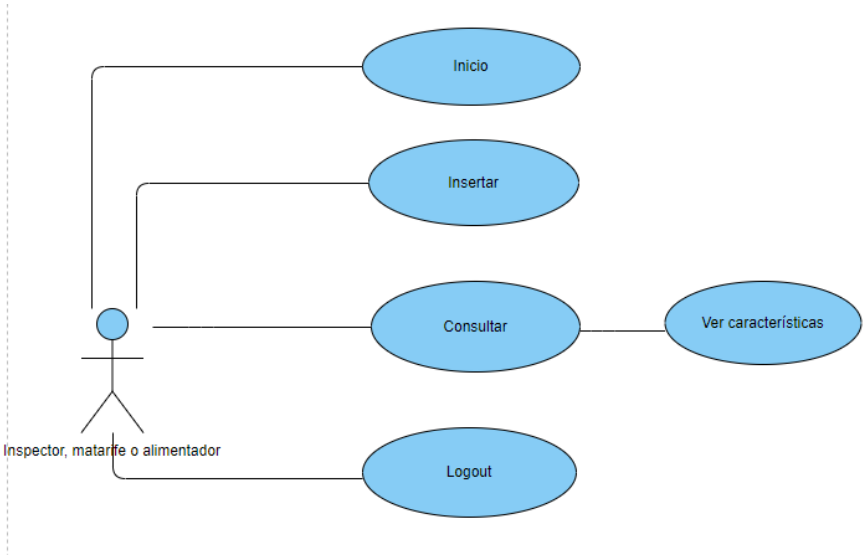

Figura 3.14: Diagrama caso de uso inspector, matarife y alimentador.

3.4 Requisitos funcionales

Identificador	Nombre	Descripción
RF0	Iniciar sesión en la aplicación	La aplicación tendrá una vista para poder realizar un inicio de sesión introduciendo el DNI y la contraseña
RF1	Realizar un registro en la aplicación	La aplicación tendrá una vista en la cual poder registrarse introduciendo una serie de datos por el tipo de usuario
RF2	Insertar operaciones sobre animales	La aplicación tendrá una vista para poder realizar inserciones sobre cualquier tipo de operación sobre animales
RF3	Consultar operaciones sobre animales	La aplicación tendrá una vista para poder realizar consultas sobre cualquier tipo de operación sobre animales
RF4	Insertar animales	La aplicación tendrá una vista en la cual insertar nuevos animales y sus características según el tipo
RF5	Realizar una simulación	La aplicación tendrá una vista para generar simulaciones, poniendo un número de inserciones y consultas, mostrará el tiempo empleado en realizarlas y serán generadas de forma aleatoria.
RF6	Consultar datos de las operaciones	Se podrá ver tanto datos de las operación en sí como sobre los animales sobre los que se ha realizado

3.5 Requisitos no funcionales

Identificador	Nombre	Descripción
RNF01	Implementación con diferentes bases de datos	La aplicación debe de ser implementada con las siguientes bases de datos: MySQL, Cassandra y Ethereum
RNF02	Capacidad de almacenamiento	La aplicación debe de ser capaz de almacenar todas las trazas sobre los animales
RNF03	Aplicación intuitiva	Debe de ser una aplicación intuitiva, es decir, fácil de utilizar por cualquier usuario.
RNF04	Concurrente	La aplicación debe de soportar múltiples usuarios concurrentemente

4

Diseño e implementación

4.1 Tecnologías utilizadas

Antes de comenzar vamos a definir algunas de las tecnologías utilizadas para el desarrollo de estas páginas web:

-JavaEE: Es una plataforma de programación para desarrollar y ejecutar software creado a través del lenguaje de programación Java [13].
-HyperText Markup Language (HTML): Es un lenguaje de marcado utilizado para estructurar páginas web. Se basa en el uso de etiquetas.
-Cascading Style Sheets (CSS): Es un lenguaje de diseño gráfico utilizado para describir cómo se deben de representar los elementos de una página web.
-JavaScript: Es un lenguaje de programación interpretado, proviene del estándar ECMAScript.
-JavaServer Faces (JSF): Es un frameworks de java utilizado en aplicaciones web para simplificar el desarrollo de interfaces de usuario en proyectos JavaEE [14].
-PrimeFaces: es una biblioteca de componentes para JavaServer Faces (JSF) de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web [8].
-GlasshFish: Es un servidor de aplicaciones de código libre.
-Netbeans: Es un entorno de desarrollo integrado libre. Permite desarrollar aplicaciones a partir de un conjunto de componentes software [15].
-XAMPP: Es un software libre, el cual se utiliza para ejecutar el sistema de gestión de base de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl [16].
- Apache TomEE: Es otro servidor de aplicaciones como Glassfish usado con JavaEE.
-Truffle Suite: Es un entorno de desarrollo que nos proporciona herramientas para ayudar a crear, probar e interpretar soluciones software de Blockchain de Ethereum [17].

-Smart Contract: Es un tipo especial de instrucciones almacenadas en Blockchain. Pueden autoejecutar acciones con una serie de parámetros programados. Todo ello lo realizan de una forma inmutable, transparente y segura [19].
-Ganache: Es un software que nos permite crear una red de pruebas local de forma sencilla para blockchain.
-Solidity: Es un lenguaje de alto nivel orientado a Smart Contracts. Su sintaxis es parecida a la de JavaScript y está pensado para usarse con Ethereum [20].
-Sublime text: Es un editor de código multiplataforma.
-Node.js: Es un entorno de ejecución multiplataforma de código abierto, basado en ECMAScript .
-Bootstrap: Es una biblioteca multiplataforma con herramientas de código abierto para el diseño de aplicaciones web.
-Web3.js: Es una colección de librerías que nos permiten interactuar con nuestro nodo local o remoto de Ethereum [21].
-JQuery: Es una biblioteca multiplataforma de JavaScript que nos ayuda a interactuar con documentos HTML, así como manejar eventos.

4.2 MySQL

4.2.1 Introducción

Es el sistema de base de datos open source más popular del mundo. Fue desarrollado por MySQL AB. Fue adquirida por Sun Microsystems en el año 2008 y posteriormente comprada por Oracle Corportation en el 2010. MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código. La base de datos se distribuye en dos tipos de versiones, una Community, bajo la licencia pública general de GNU y varias versiones Enterprise, para que las usen las empresas en sus productos privados. Estas versiones Enterprise se diferencian principalmente en el soporte que se les da a las empresas [7]. El lenguaje de programación utilizado por MySQL es Structured Query Language (SQL).

4.2.2 Modelo entidad/relación

En la figura 4.1 podemos ver el modelo que hemos creado, con las entidades y sus respectivas relaciones. Este modelo lo utilizaremos posteriormente para crear la base de datos.

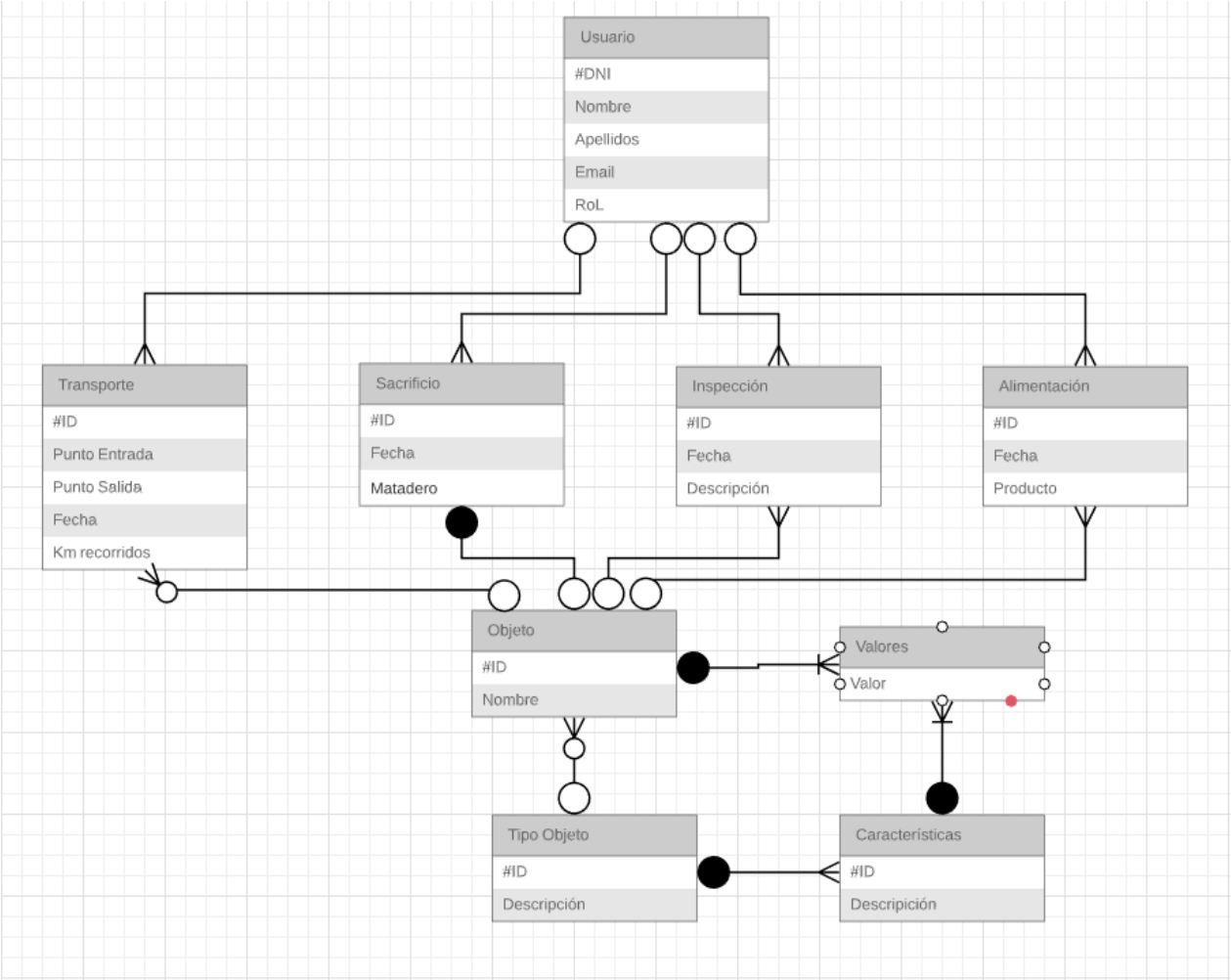

Figura 4.1: Modelo Entidad/Relación.

4.2.3 Desarrollo

Para el desarrollo de esta página web hemos utilizado el programa XAMPP para ejecutar el servidor MySQL y el servidor Apache como podemos ver en la figura 4.2.

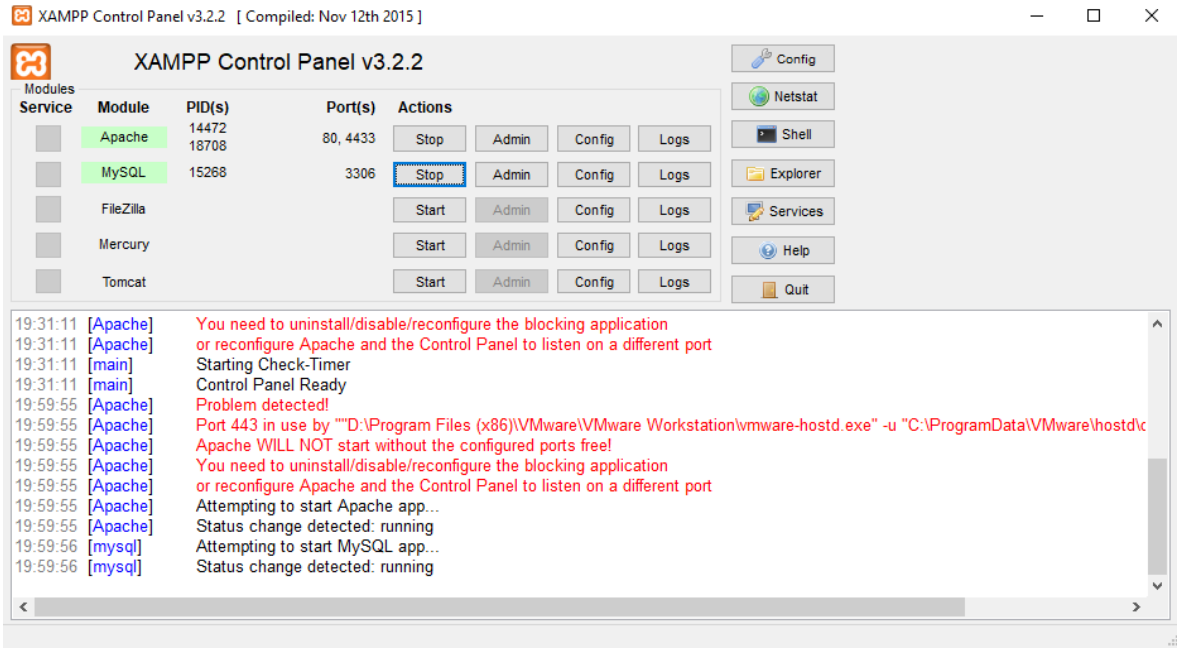

Figura 4.2: Programa XAMPP con Apache y MySQL en ejecución.

Hemos tenido que crear en la página de administración de MySQL una base de datos llamada TFG, representando el modelo entidad relación creado, para ello simplemente hemos usado un script generado automáticamente a través del modelo Entidad/Relación con sentencias DDL (Data Definicion Language) y lo hemos importado en la página de administración de MySQL, con el resultado que aparece en la figura 4.3.

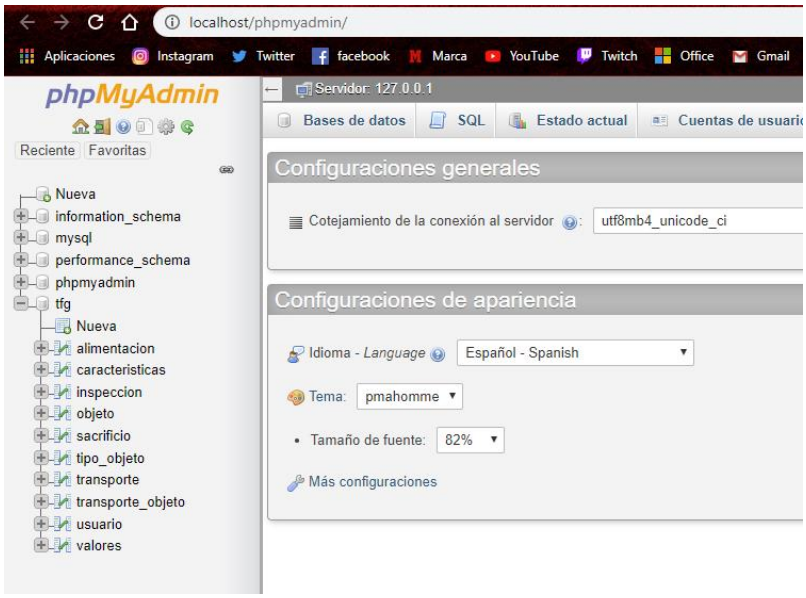

Figura 4.3: Menú de administración MySQL.

Una vez configurada y creada la base de datos, el desarrollo de la aplicación lo vamos a realizar mediante el IDE Netbeans y Github para tener en todo momento backups del proyecto en todo momento. Como servidor de aplicaciones hemos utilizado GlassFish.

Para realizar el proyecto hemos utilizado el patrón de diseño modelo-vista-controlador (MVC), es decir en primer lugar tenemos el modelo, el cual sería una serie de clases que representan la estructura de las tablas en la base de datos, las cuales nos ayudan a interactuar con MySQL. En segundo lugar, tenemos la vista, la cual representa la información y mediante la que el usuario interactúa. Estas peticiones enviadas por el usuario pasarían al controlador que se encarga de responder a eventos y realizar peticiones al modelo. Por lo general cada controlador está asociada a una única vista y realizaría la función de intermediario entre la vista y el modelo. Podemos ver la estructura del proyecto en NetBeans en la figura 4.4.

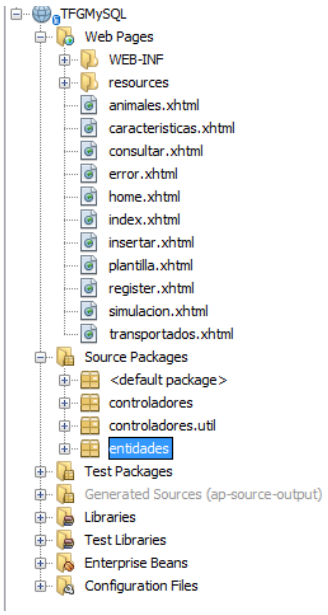

Figura 4.4: Estructura del proyecto en NetBeans.

Estas vistas han sido creadas mediante el uso del lenguaje de etiquetas HTML y dándole estilo gracias al lenguaje de estilos CSS. También hemos usado números componentes de la librería PrimeFaces que junto con JavaServer Faces nos facilitan en gran medida la estructura e interacción con los controladores. En la figura 4.5 podemos ver un ejemplo de un pedazo de código de la vista insertar donde aparece un componente de PrimeFaces que nos permite interactuar con nuestro controlador.

```
<c:if test="#{usuarioController.isTransportista()}">
    <p:tab title="Transporte">
        #{transporteController.prepareCreate()}
        <h:form>
            <h:panelGrid columns="2" cellpadding="5">
                <h:outputLabel for="menu" value="Transportados: " />
                <p:selectCheckboxMenu id="menu" value="#{transporteController.selectedObjetos}" label="Objetos" required="true" multiple="true"
                                      filter="true" filterMatchMode="startsWith" panelStyle="width:250px">
                    <f:selectItems value="#{objetoController.getObjetoListFormat(objetoController.itemsTodos)}" />
                </p:selectCheckboxMenu>

                <h:outputLabel value="Origen: " for="puntoEntrada" />
                <p:inputText id="puntoEntrada" value="#{transporteController.selected.puntoEntrada}" required="true" />

                <h:outputLabel value="Destino: " for="puntoSalida" />
                <p:inputText id="puntoSalida" value="#{transporteController.selected.puntoSalida}" required="true" />

                <h:outputLabel value="Fecha: " for="fecha" />
                <p:calendar id="fecha" value="#{transporteController.selected.fecha}" pattern="dd/MM/yyyy" required="true" />

                <h:outputLabel value="Km Recorridos: " for="kmRecorridos" />
                <p:inputText id="kmRecorridos" value="#{transporteController.selected.kmRecorridos}" required="true" />
            </h:panelGrid>

            <p:commandButton ajax="false"  action="#{transporteController.creaTransporte(usuarioController.getSelected(), objetoController.selecte
        </h:form>
        <br />
    </p:tab>
</c:if>
```

Figura 4.5: Tabla para insertar un transporte.

Como se puede observar aparecen numerosos elementos con el prefijo "p:", todos estos elementos son de las librerías de PrimeFaces, mientras que los elementos con el prefijo "h:" y "c:" son de las librerías de JavaServer Faces. Podemos ver por ejemplo el elemento "c:if" junto con "test" nos permite básicamente crear como si se tratase de un if, dependiendo de si la función "usuarioControles.isTransportista()" devuelve true o false, se mostraría el contenido que hay dentro de la etiqueta o no.

Hemos visto que desde una vista podemos llamar directamente a los controladores, estos se ocupan de realizar las operaciones necesarias para devolver el resultado esperado. A continuación, en la figura 4.6 vamos a mostrar el controlador de usuario, el cual es el que llamamos en el ejemplo anterior.

```
@Named("usuarioController")
@SessionScoped
public class UsuarioController implements Serializable {

    private Usuario selected = new Usuario();
    private DataModel items = null;
    @EJB
    private controladores.UsuarioFacade ejbFacade;
    private PaginationHelper pagination;
    private int selectedItemIndex;
    private String repitePass;

    public String getRepitePass() {
        return repitePass;
    }

    public void setRepitePass(String repitePass) {
        this.repitePass = repitePass;
    }

    public UsuarioController() {
    }

    public Usuario getSelected() {
        if (selected == null) {
            selected = new Usuario();
            selectedItemIndex = -1;
        }
        return selected;
    }

    private UsuarioFacade getFacade() {
        return ejbFacade;
    }

    public PaginationHelper getPagination() {
        if (pagination == null) {
            pagination = new PaginationHelper(10) {
```

Figura 4.6: Controlador de usuario.

Este controlador es lo que es denominado un "Bean", dependiendo del tipo de Bean que deseemos, podemos guardar los datos entre distintas peticiones de un usuario, por ejemplo, en nuestro caso tenemos puesto "@SessionScoped", esto significa que los datos almacenados en las variables de este controlador no se pierden hasta que se cierra la sesión. Por lo tanto, la aplicación puede usar datos de peticiones anteriores del usuario. Los beans pueden tener diferentes ámbitos, si por ejemplo no queremos guardar la información entre diferentes peticiones podemos usar "@RequestScoped", en la cual se pierde al realizar la petición.

Este controlador se comunica tanto con la vista como con el modelo para mantener los datos estructurados, nuestro modelo no es otra cosa más que un conjunto de variables estructuradas de forma similar a como están en las tablas de la base de datos para así poder interactuar más fácilmente con MySQL como se puede ver en la figura 4.7.

```java
@Entity
@Table(name = "usuario")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Usuario.findAll", query = "SELECT u FROM Usuario u")
    , @NamedQuery(name = "Usuario.findByDni", query = "SELECT u FROM Usuario u WHERE u.dni = :dni")
    , @NamedQuery(name = "Usuario.findByNombre", query = "SELECT u FROM Usuario u WHERE u.nombre = :nombre")
    , @NamedQuery(name = "Usuario.findByApellidos", query = "SELECT u FROM Usuario u WHERE u.apellidos = :apellidos")
    , @NamedQuery(name = "Usuario.findByEmail", query = "SELECT u FROM Usuario u WHERE u.email = :email")
    , @NamedQuery(name = "Usuario.findByRol", query = "SELECT u FROM Usuario u WHERE u.rol = :rol")
    , @NamedQuery(name = "Usuario.findByPassword", query = "SELECT u FROM Usuario u WHERE u.password = :password")})
public class Usuario implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 10)
    @Column(name = "dni")
    private String dni;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 30)
    @Column(name = "nombre")
    private String nombre;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 60)
    @Column(name = "apellidos")
    private String apellidos;
```

Figura 4.7: Modelo de usuario.

Podemos observar como en el modelo aparecen incluso el tamaño de variables y diferentes propiedades que tienen en la base de datos, así como una serie de consultas para poder realizarlas de forma rápida.

Como hemos dicho antes el controlador interactúa con la vista y el modelo para poder realizar operaciones en la base de datos. Cuando desde el controlador se desea realizar alguna operación en concreto sobre la base de datos utilizamos las clases ejb. Hay una clase para cada controlador, desde ellas se realizan operaciones sobre la base de datos. En la figura 4.8 podemos ver como en el ejb tenemos creada una clase para insertar usuarios creados en la simulación en la base de datos.

```java
@Stateless
public class UsuarioFacade extends AbstractFacade<Usuario> {

    @PersistenceContext(unitName = "TFGMySQLPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public void crearUsuario(String simulacion) {
        String cqlUsuario = "INSERT INTO usuario (dni, nombre, apellidos, email, rol, password) VALUES ('"
                + simulacion + "','"
                + "simulacion" + "','"
                + "simulacion" + "','"
                + "simulacion" + "','"
                + "Admin" + "','"
                + "simulacion" + "')";

        em.createNativeQuery(cqlUsuario).executeUpdate();
    }

    public UsuarioFacade() {
        super(Usuario.class);
    }

}
```

Figura 4.8: Ejb del usuario.

4.3 Cassandra

4.3.1 Introducción

Cassandra es un tipo de base de datos NoSQL, es distribuida y masivamente escalable, posee la capacidad de escalar linealmente. Cassandra surgió en Facebook, diseñada para potenciar la funcionalidad de búsqueda en el inbox. A partir del año 2008 paso a ser un proyecto open source y en 2010 paso a ser un proyecto top-level de la fundación Apache.

Al tratarse de una base de datos distribuida la información se distribuye entre todos los nodos que componen el cluster. Esto es una gran ventaja ya que si alguno nodo no está disponible no hay ningún problema, se puede recuperar la información. Al poseer un escalado lineal quiere decir que, si con 6 nodos soporta 100000 operaciones por segundo, con 12 nodos soporta 2000000. A su vez, este escalado es de forma horizontal por lo tanto basta con añadir nuevos nodos basados en hardware commodity de bajo coste.

Cassandra utiliza el lenguaje de acceso a datos Cassandra Query Language (CQL), muy parecido al tradicional lenguaje SQL [9].

4.3.2 Desarrollo

Lo primero que hemos hecho ha sido crear las "tablas" en Cassandra, una vez instalado Apache Cassandra en el ordenador. Cassandra nos trae una consola llamada CQLSH que utiliza el lenguaje CQL (Cassandra Query Language) para interactuar con la base de datos. En este caso como se trata de una base de datos no relacional, creamos un keyspace que puede ser visto como un tablespace en un sistema relacional, solo que dentro en vez de tener tablas, tenemos familias de columnas. En la figura 4.9 podemos ver el script usado para crear el keyspace.

```
create keyspace tfg
with replication = {'class':'SimpleStrategy','replication_factor':3};

use tfg;

create table operacion(id uuid primary key, fecha date, tipo varchar, objetos list<varchar>, dni varchar, puntoEntrada varchar, puntoSalida varchar, kmRecorridos
varchar, matadero varchar, descripcion varchar, producto varchar);
create table objeto(id uuid primary key, nombre varchar, tipo varchar, caracteristicas map<varchar,varchar>);
create table usuario(dni varchar primary key, nombre varchar, apellidos varchar, email varchar, password varchar, rol varchar);
```

Figura 4.9: Script utilizado para crear el keyspace

Al no tratarse de una base de datos relacional, hemos dividido la estructura en 3 familias de columnas a diferencia de en las relacionales. Encontramos una familia de columnas llamada objeto, en la cual hemos añadido una columna con el tipo de animal y otra con las características del tipo de animal. Por otro lado, la familia de columnas llamada usuario sería similar a la tabla usuario en la base de datos relacional. En último lugar, a diferencia de en MySQL que teníamos una tabla para cada tipo de operación, aquí simplemente hemos creado una familia de columnas para todas las operaciones, donde hemos añadido una columna que sea el tipo de operación y varias columnas con todas las características de todas las operaciones, simplemente rellenaremos las columnas adecuadas al tipo de operación que se crear, todo esto lo controlaremos a través del backend.

Igual que en la página de web anterior utilizaremos de nuevo NetBeans ya que nos proporciona una gran cantidad de herramientas y utilidades. A su vez, también hemos utilizado GitHub para los backups del proyecto. A diferencia de MySQL para este proyecto hemos utilizado de servidor de aplicaciones Apache TomEE ya que tuvimos algún que otro problema con GlassFish que comentaremos posteriormente. También, hemos incorporado al proyecto Maven por motivos de problemas a la hora de importar las librerías de Cassandra.

Hemo utilizado el mismo patrón de diseño que en el proyecto de MySQL, es decir el proyecto está estructurado en Modelo-Vista-Controlador (MVC), esto lo podemos ver en la figura 4.10.

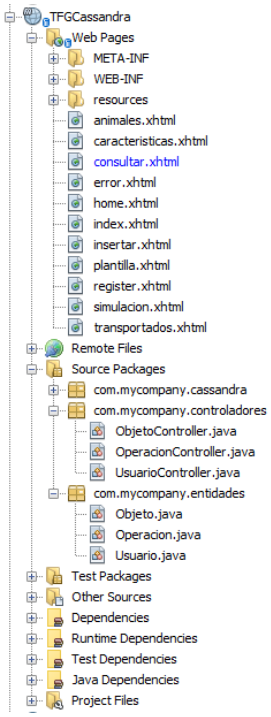

Figura 4.10: Estructura del proyecto de Cassandra.

Las vistas son similares a las de MySQL, utilizando también HTML, CSS, PrimeFaces y JSF. La única diferencia es que, en vez de llamar a las funciones de los controladores anteriores, se ha actualizado y llaman a nuevas funciones de los controladores actuales. En la figura 4.11 podemos ver como llaman a los controladores para crear un transporte, sin embargo, todas estas operaciones las gestiona el controlador "operaciónController" y no "matarifeController" como en la anterior página.

```
<c:if test="#{usuarioController.isMatarife()}">
        <p:tab title="Sacrificio">
            <h:form>
                <h:panelGrid columns="2" cellpadding="5">
                    <h:outputLabel for="menu" value="Sacrificado: " />
                    <p:selectOneMenu id="menu" value="#{operacionController.selectedObjeto}" required="true"
                                     panelStyle="width:180px" filter="true" filterMatchMode="startsWith">
                        <f:selectItems value="#{objetoController.getObjetoListFormat(objetoController.items)}" />
                    </p:selectOneMenu>

                    <h:outputLabel value="Matadero: " for="matadero" />
                    <p:inputText id="matadero" value="#{operacionController.selected.matadero}" required="true" />

                    <h:outputLabel value="Fecha: " for="fecha" />
                    <p:calendar id="fecha" value="#{operacionController.selected.fecha}" pattern="dd/MM/yyyy" required="true" />
                </h:panelGrid>

                <p:commandButton ajax="false"  action="#{operacionController.creaSacrificio(usuarioController.getSelected(),objetoController

            </h:form>
        </p:tab>
        <br />
    </c:if>
</div>
```

Figura 4.11: Tabla para insertar un sacrificio

En Cassandra tenemos una clase dedicada a la conexión de la base de datos, la cual después usamos en los controladores para realizar inserciones, la clase se llama Cliente y la podemos ver en la figura 4.11.

```
package com.mycompany.cassandra;

import com.datastax.driver.core.*;

public class Client {

    private Cluster cluster;
    private Session session;

    public Client() {
        connect("localhost");
    }

    private void connect(String node) {
        cluster = Cluster.builder()
                .addContactPoint(node)
                .build();

        session = cluster.connect("tfg");
    }

    public ResultSet execute(String query) {
        return session.execute(query);
    }

    public void close() {
        cluster.close();
    }

}
```

Figura 4.11: Clase para crear la conexión a Cassandra

Los controladores tienen la misma estructura y funcionalidad que en el proyecto de MySQL, la única diferencia es que para realizar operaciones en Cassandra, utilizan una instancia de la clase Client. Esto lo podemos observar en la figura 4.12.

```java
public void insertarSacrificio() {

    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");

    // Insertar Sacrificio
    String cqlStatement = "INSERT INTO operacion (id, fecha, tipo, objetos, dni, matadero) VALUES (uuid(), '"
            + formatter.format(selected.getFecha())+ "','"
            + selected.getTipo() + "','"
            + selected.getObjetos() + ",'"
            + selected.getDni() + "','"
            + selected.getMatadero() + "')";

    cliente.execute(cqlStatement);
    System.out.println("Documento insertado OK");

}
```

Figura 4.12: Función para crear un sacrificio en el controlador "operacionController"

En cuanto a los modelos, en este caso solo tenemos 3 entidades, las cuales nos ayudan a mapear la información.

```java
public class Operacion {
    private UUID id;              //ID
    private Date fecha;
    private String tipo;            // Tipo de Operacion realizada
    private List<UUID> objetos;   // Id de los animales con los que se opera
    private String dni;             // Dni del usuario que hace la operacion

    /* Campos en función del tipo de operacion */
    // TRANSPORTE
    private String puntoEntrada;
    private String puntoSalida;
    private String kmRecorridos;

    // SACRIFICIO
    private String matadero;

    // INSPECCION
    private String descripcion;

    // ALIMENTACION
    private String producto;

    public Operacion(){
    }
```

Figura 4.13: Entidad de operación

4.4 Ethereum

4.3.1 Introducción

Ethereum es una plataforma open source, al igual que Cassandra es descentralizada. Ethereum sigue el modelo de cadenas de bloques y permite la creación de contratos inteligentes entre pares. A su vez Ethereum se usa para intercambiar su propia moneda entre cuentas, esta criptomoneda se llama ether (ETH). A día de hoy, su precio equivale a unos 150 euros por unidad.

Ethereum fue anunciando oficialmente en la foto de Bitcointalk en 2014. La fundación Ethereum fue lanzada en julio de 2014, vendiendo 60 millones de fichas y creando 12 millones de fichas (ETH). Su precio se multiplico en un factor de 1000 en los 2 primeros meses.
A parte de su uso para intercambiar su propia moneda, Ethereum se puede usar para crear un sistema distribuido, donde no necesitemos el uso de una entidad

centralizada la cual gestione las operaciones. Este es el uso que le vamos a dar a continuación, el objetivo es simular una base de datos [18].

4.3.2 Desarrollo

En el caso de Ethereum no tenemos ningún tablespace o keyspace, ya que aquí se almacena la información en la cadena de bloques. Sin embargo, Ethereum nos proporciona la capacidad de utilizar Smart Contracts, gracias a ellos podemos simplificar en gran medida la estructura del proyecto.

Para esta página web vamos a usar una red local para desplegarla, para ello hemos utilizado Ganache, el cual también nos genera 10 cuentas con 100 de ETH para poder realizar las transacciones, ya que cada una supone un gasto de ETH. Como podemos ver en la figura 4.14 en nuestro caso hemos utilizado la primera cuenta para realizar las transacciones y como se puede ver hemos realizado un total de 7667, a pesar de eso aún nos quedan 80.22 ETH ya que el gasto por transacción es diminuto.

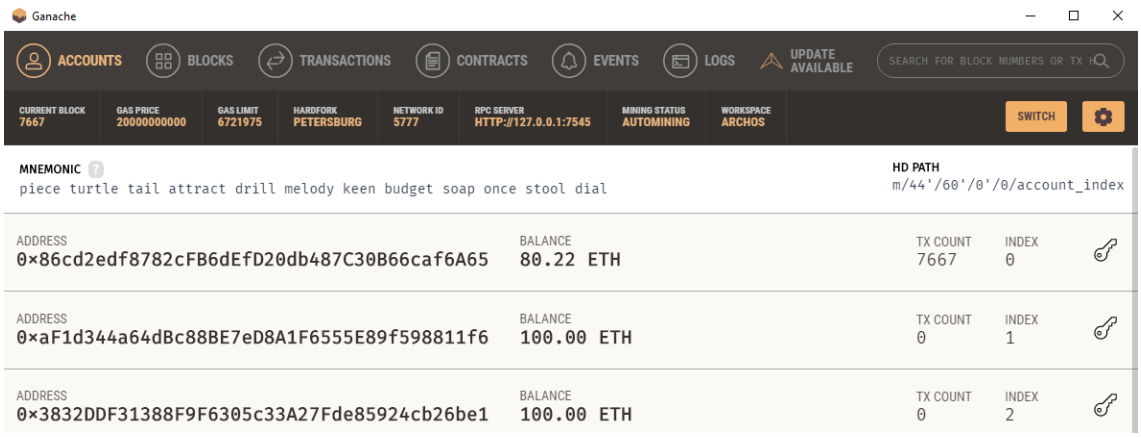

Figura 4.14: Programa Ganache con la red local y el ETH que disponemos por cuenta.

Para el desarrollo de esta página web no hemos usado NetBeans, hemos utilizado Truffle Suite como entorno de desarrollo, el cual nos ha proporciona una plantilla con la estructura del proyecto. Come editor de código hemos utilizado Sublime Text. La estructura del proyecto la podemos ver en la figura 4.15 que como se puede observar no se parece en nada a la mostrada en los proyectos anteriores.

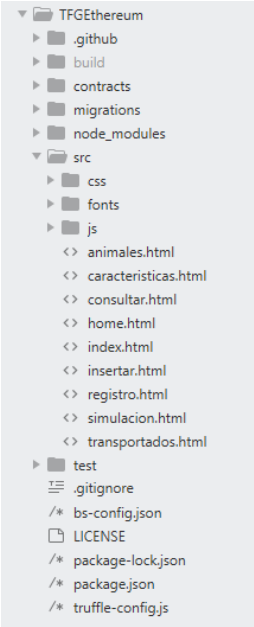

Figura 4.15: Estructura del proyecto de Ethereum.

El principal archivo de configuración de proyecto sería "truffle-config.js" donde establecemos las diferentes redes Ethereum sobre las que podemos desplegar la aplicación y los contratos. En el caso de la figura 4.16 podemos ver que tenemos 2 creadas, la red development y la red prueba, sin embargo, la red que vamos a usar es la development.

```
const HDWalletProvider = require("truffle-hdwallet-provider");

module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // for more about customizing your Truffle configuration!
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*" // Match any network id
    },

    prueba: {
      provider: function() {
        return new HDWalletProvider("dynamic valid better whale unf
      },
      network_id: 5777
    }
  }
};
```

Figura 4.16: Archivo de configuración de las redes del proyecto.

Para las vistas de este proyecto también hemos utilizado HTML y CSS para darle estilo. Sin embargo, no hemos utilizado ni PrimeFaces ni JSF por problemas de compatibilidad, debido a esto hemos usado el utilizado el framework Bootstrap para el uso de ciertos componentes que nos facilitasen el desarrollo. Estas vistas ahora no se comunican con los controladores como en los anteriores casos, sino que interaccionan con JavaScript. En la figura 4.17 podemos ver un ejemplo.

```
<div id="Alimentacion">
    <legend><span class="number">-</span>Alimentacion
    </legend>
    <form>
        <div class="form-group row">
            <label for="menu-alimentacion" class="
            col-sm-3 col-form-label">Alimentado</
            label>
            <div class="col-sm-9">
                <select class='form-control'  id='
                menu-alimentacion' name='
                menu-alimentacion'>
                </select>
            </div>

            <br />
            <br />
            <br />

            <label for="producto" class="col-sm-3
            col-form-label">Producto</label>
            <div class="col-sm-9">
                <input type="text" class="
                form-control-plaintext" id="producto"
                 />
            </div>

            <label for="fecha" class="col-sm-3
            col-form-label">Fecha</label>
            <div class="col-sm-9">
                <input type="date" id="
                fecha-alimentacion" min="2019-01-01"
                max="2019-12-31" />
            </div>
        </div>

        <button id="insertar-alimentacion" type="
        submit">Guardar</button>
    </form>
```

Figura 4.17: Tabla para insertar una alimentación.

Hemos utilizado los JavaScript para interactuar con los archivos HTML e invocar sus peticiones a la red Blockchain a través de los Smart Contracts. También cabe mencionar, que en algunos casos hemos utilizado las librerías de JQuery para interactuar de forma más sencilla con los HTML. Utilizamos las librerías de Web3 para establecer nuestro nodo local de Ethereum, así como la cuenta que deseamos utilizar.

```
loadWeb3: async () => {
  web3 =  new Web3(new Web3.providers.HttpProvider('http://localhost:7545'));

  App.web3Provider = web3.currentProvider;
  window.web3 = new Web3(web3.currentProvider)

  // Set the current blockchain account
  App.account = web3.eth.accounts[0]

  web3.eth.defaultAccount = web3.eth.accounts[0];
},
```

Figura 4.18: Creando conexión a nodo Ethereum y estableciendo la cuenta que seamos utilizar en JavaScript.

A su vez, también necesitamos cargar los diferentes contratos que hemos creado para poder interactuar con ellos y así realizar operaciones de manera más fácil sobre la red blockchain. También le asignamos el nodo Ethereum que estamos utilizando. En la figura 4.19 realizamos la carga los contratos de usuario, operación y objeto.

```
loadContract: async () => {
  // Create a JavaScript version of the smart contract
  const usuario = await $.getJSON('Usuario.json')
  App.contracts.Usuario = TruffleContract(usuario)
  App.contracts.Usuario.setProvider(App.web3Provider)

  // Hydrate the smart contract with values from the blockchain
  App.usuario = await App.contracts.Usuario.deployed()

  // Create a JavaScript version of the smart contract
  const objeto = await $.getJSON('Objeto.json')
  App.contracts.Objeto = TruffleContract(objeto)
  App.contracts.Objeto.setProvider(App.web3Provider)

  // Hydrate the smart contract with values from the blockchain
  App.objeto = await App.contracts.Objeto.deployed()


  // Create a JavaScript version of the smart contract
  const operacion = await $.getJSON('Operacion.json')
  App.contracts.Operacion = TruffleContract(operacion)
  App.contracts.Operacion.setProvider(App.web3Provider)

  // Hydrate the smart contract with values from the blockchain
  App.operacion = await App.contracts.Operacion.deployed()
},
```

Figura 4.19: Carga de los distintos contratos en JavaScript.

Una vez disponibles los contratos para su uso desde JavaScript es cuando podemos interactuar con ellos, a continuación, en la figura 4.20 vamos a ver un ejemplo de cómo llamamos a los contratos para crear una alimentación. Para ello, también utilizamos algunas funcionalidades de Node.js para esperar a que los contratos hayan sido desplegados. Como se puede ver necesitamos poner desde la cuenta que vamos a realizar la operación y podemos introducir el coste (gas) de llamar a esa función, que al tratarse de una red privada podríamos establecerlo como quisiésemos.

```
App.contracts.Operacion.deployed().then(async function(instance) {
  operacionInstance = instance;

  App.contracts.Usuario.deployed().then(async function(instance2) {
    usuarioInstance = instance2;

    const tam = await usuarioInstance.getNumAgentes();
    var dni;

    for(var i = 0; i < tam; i++){
        const ag = await usuarioInstance.agentes(i);
        if(ag[6] == account){
            await operacionInstance.crearAlimentacion(solidityDate, ag[0], producto, objeto, {from: account, gas:3000000});
            window.location.href = "http://localhost:3000/home.html";
        }
    }
```

Figura 4.20: Llamada a Smart Contract para crear una alimentación desde JavaScript.

Por último, ya solo nos queda por ver los Smart Contract, aunque dentro del proyecto es necesario un archivo que se encarga de desplegar los contractos en la red blockchain una vez que se hayan migrado.

```
2_deploy_contracts.js
1  var Usuario = artifacts.require("./Usuario.sol");
2  var Operacion = artifacts.require("./Operacion.sol");
3  var Objeto = artifacts.require("./Objeto.sol");
4
5  module.exports = function(deployer) {
6    deployer.deploy(Usuario);
7    deployer.deploy(Operacion);
8    deployer.deploy(Objeto);
9  };
10
```

Figura 4.21: Archivo para desplegar los contratos.

En cuanto a los contratos, hemos utilizado el lenguaje de programación Solidity para crearlos. Hemos creado 3, uno para usuarios, otro para operaciones y otro para objetos. En ellos establecemos una serie de variables similares a las utilizadas en la familia de columnas de mongo y también una serie de funciones que se encargan de realizar las operaciones deseadas desde las vistas. En las figuras 4.22 podemos ver un ejemplo de las variables creadas para los usuarios y una función para realizar el registro.

```
Usuario.sol
1  pragma solidity ^0.5.0;
2
3  contract Usuario {
4
5      struct Agente{
6          string dni;
7          string nombre;
8          string apellidos;
9          string email;
10         string password;
11         string rol;
12         address sesion;
13     }
14
15     uint public numAgentes;
16     mapping (uint => Agente) public agentes;
17
```

Figura 4.22: Variables del Smart Contract Usuario.

Para realizar una inserción en la cadena de bloques basta con hacer uso de alguna de estas funciones y establecer el valor de las variables, posteriormente automáticamente se realizaría el proceso de descrito en el capítulo 2.4.1. Un ejemplo de una función de un Smart Contract lo podemos ver en la figura 4.23 donde se hace uso del "array" de agentes para insertar un usuario nuevo.

```
function registro(string memory d, string memory p, string memory n,
    string memory a, string memory e, string memory r) public{

    // Comprobar que no este registrao (js)
    // Seteamos valores

    agentes[numAgentes] = Agente(d,n,a,e,p,r,msg.sender);

    numAgentes++;
}
```

Figura 4.23: Función de registro de un usuario en el Smart Contract Usuario.

5

Manual de usuario

La estructura de las páginas web será similar en cada una de ellas, teniendo los mismo apartados, distribución y organización en todas, con ligeramente algunos cambios de diseño y estilo.
Nada más acceder a la web, encontramos la ventana de inicio de sesión como podemos ver en la figura 5.1, en la cual podremos acceder con cualquiera de los tipos de usuarios presentes en la web (matarife, transportista, inspector, granjero y administrador). Para ello solo necesitaremos el DNI y la contraseña.

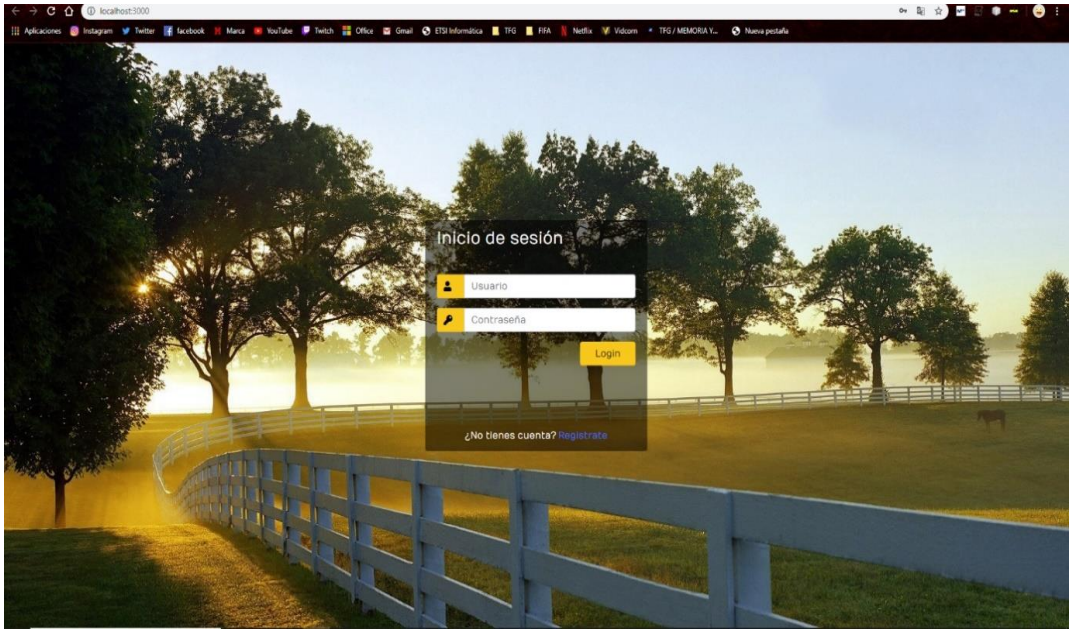

Figura 5.1: Vista de inicio de sesión

En la misma vista de inicio de sesión, encontramos un enlace para registrarnos si no tenemos cuenta, en la vista de registro deberemos de introducir diferentes datos personales y seleccionar el tipo de usuario que deseamos ser como podemos observar en la figura 5.2, sin embargo, no podremos seleccionar el usuario

administrador, este perfil ya estará creado en la base de datos y no se podrán crear más. La elección de un tipo de usuario o de otro determinará las funciones que podrás realizar en la página web.

Sin embargo, el usuario de tipo admin podrá realizar todo lo que realizan los otros tipos de usuarios a parte de añadir animales y usar la pestaña de simulación.

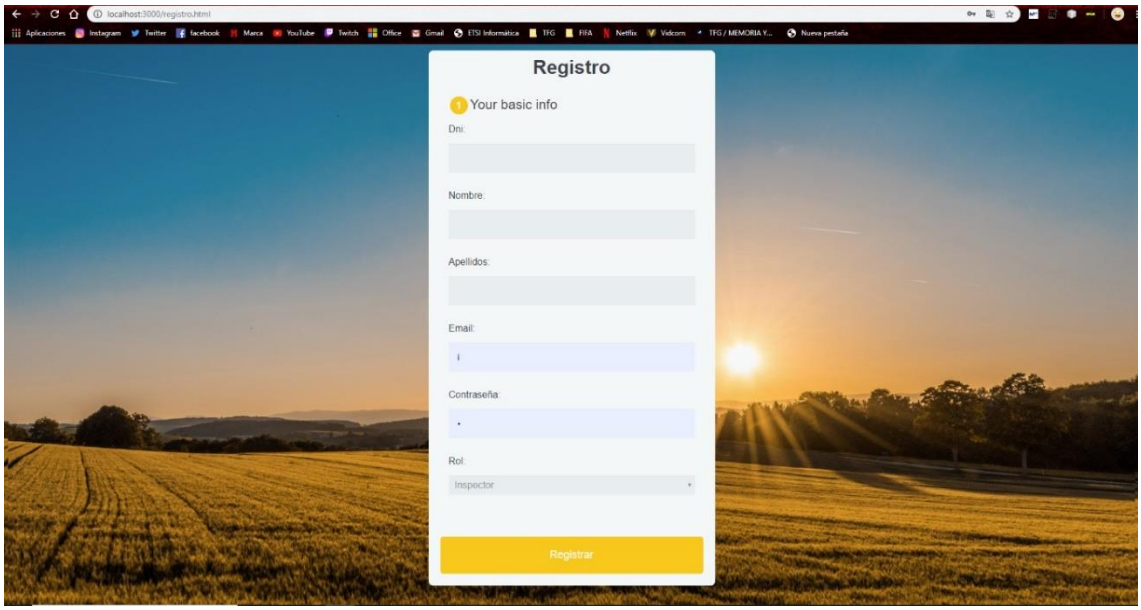

Figura 5.2: Vista de registro

Una vez realizado el registro o iniciada sesión nos situaremos en la vista de home, la cual es una pequeña presentación de la página web. Allí encontraremos una descripción de los distintos perfiles que tenemos en la página web.

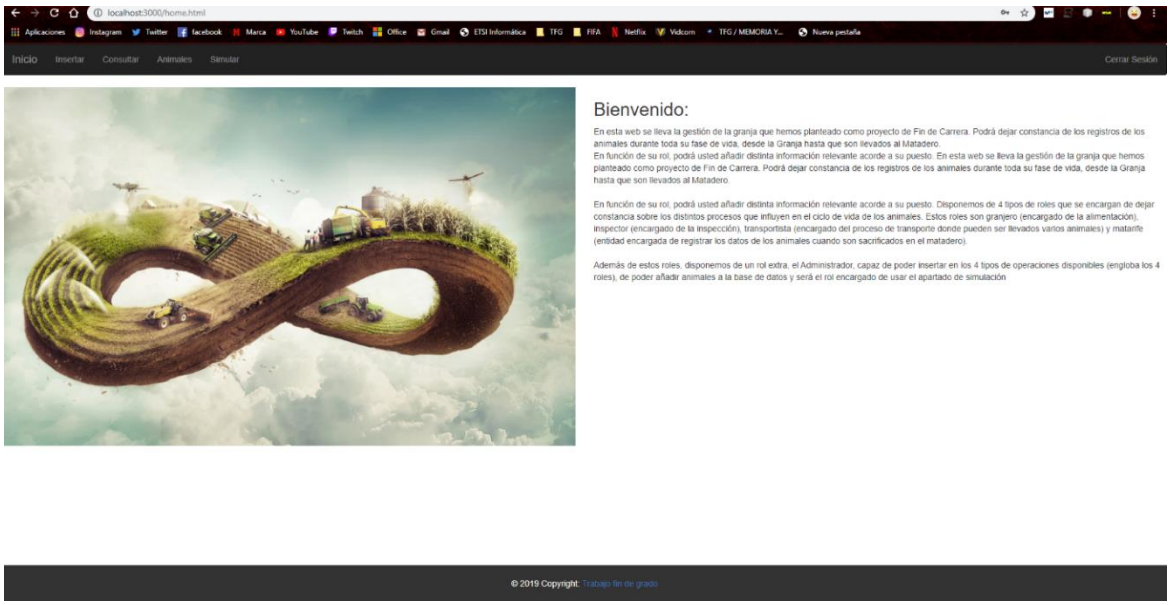

Figura 5.3: Vista de home

Si accedemos a la pestaña de consultar, encontraremos diferentes vistas, dependiendo del tipo de rol que seamos, en nuestro caso vamos a mostrar desde el rol de administrador que puede ver todo lo que ven los otros roles, sin embargo, si accediésemos con el perfil por ejemplo de transportistas, solo podríamos ver la tabla para insertar un transporte. A continuación, en las figuras 5.4, 5.5, 5.6 y 5.7, mostraremos los diferentes campos que hay que insertar para cada una de las operaciones (transporte, alimentación, inspección y sacrificio).

Para insertar un transporte necesitamos seleccionarlos animales que queremos transportar, el punto de origen y de destino del transporte, la fecha en la que se va a realizar y el número de km que se van a recorrer.

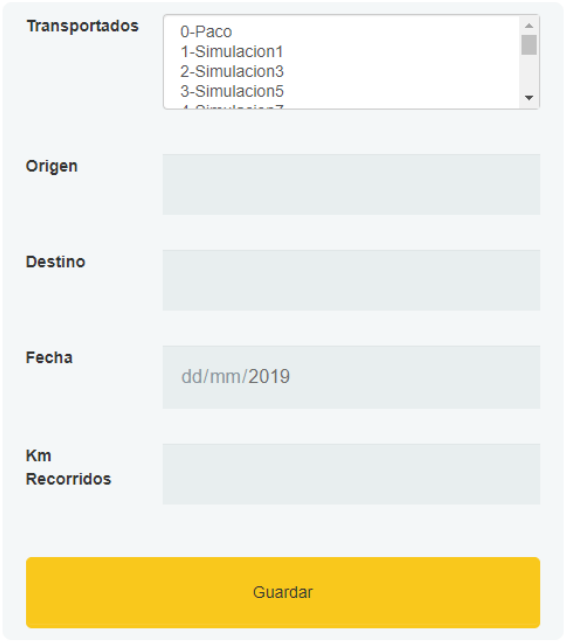

Figura 5.4: Vista para insertar un transporte

Las inspecciones en nuestro caso en concreto las tenemos puestas individuales, por ello se tiene que seleccionar el animal al que se le va a realizar la inspección, en el campo descripción se explica lo que se ha realizado y los resultados obtenidos y por último la fecha en la que ha tenido lugar.

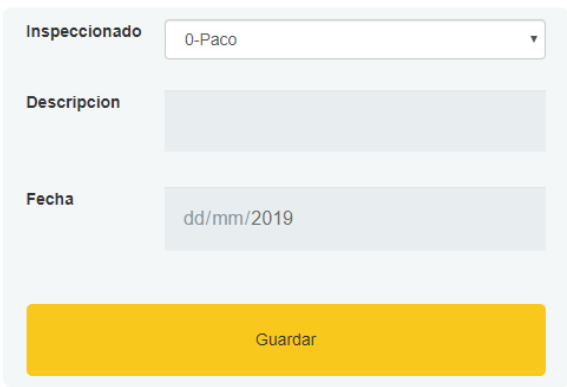

Figura 5.5: Vista para insertar una inspección.

Para insertar una alimentación, simplemente seleccionamos al animal al cual se le ha dado comida, el producto que se le ha dado y la fecha en la cual se le ha dado.

Figura 5.6: Vista para insertar una alimentación

Sacrificios solo se puede realizar uno por animal, ya que un animal no puede morir varias veces, entonces en esta vista se comprueba que el animal no haya sido sacrificado anteriormente. Para realizar el sacrificio simplemente seleccionamos el animal, ponemos en el matadero que se ha realizado y la fecha.

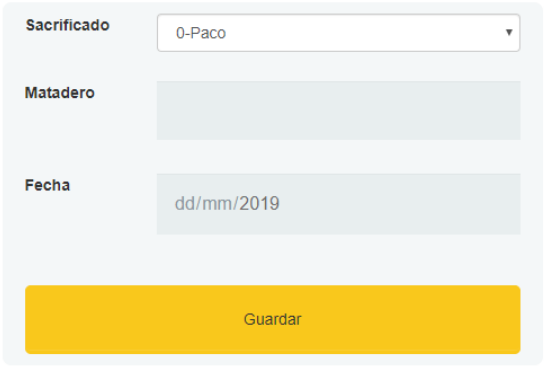

Figura 5.7: Vista para insertar un sacrificio

A continuación, vamos a mostrar la vista de consultas, donde en el caso de cada rol aparecerá simplemente una tabla con las operaciones que el mismo ha realizado. Sin embargo, a continuación, mostraremos desde la vista de administrador mediante la cual se pueden ver todas las operaciones realizadas por cualquier usuario.

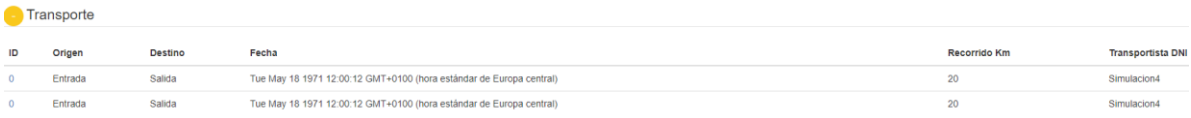

Figura 5.8: Vista de consultas de transportes

Podemos ver todos los campos que hemos insertado anterior mente en transporte, incluyendo el ID del mismo, el cual se genera automáticamente y donde podemos hacer click y nos lleva a una vista donde encontramos todos los animales que ha sido transportados como aparece en la figura 5.9. A su vez, al hacer clic en la ID de estos animales nos lleva a otra vista donde nos muestra las características de ese animal. Esto lo podemos ver en la figura 5.10.

Figura 5.9: Vista de los animales transportados

Figura 5.10: Vista de las características del animal

El usuario administrado es el único que puede añadir animales, esto realiza en la pestaña animales. Como podemos ver en la figura 5.11. para añadir un animal bastaría con seleccionar el tipo de animal, poner su nombre y añadir las características propias de ese tipo de animal, ya que no tienen las mismas características un cerdo que un pollo.

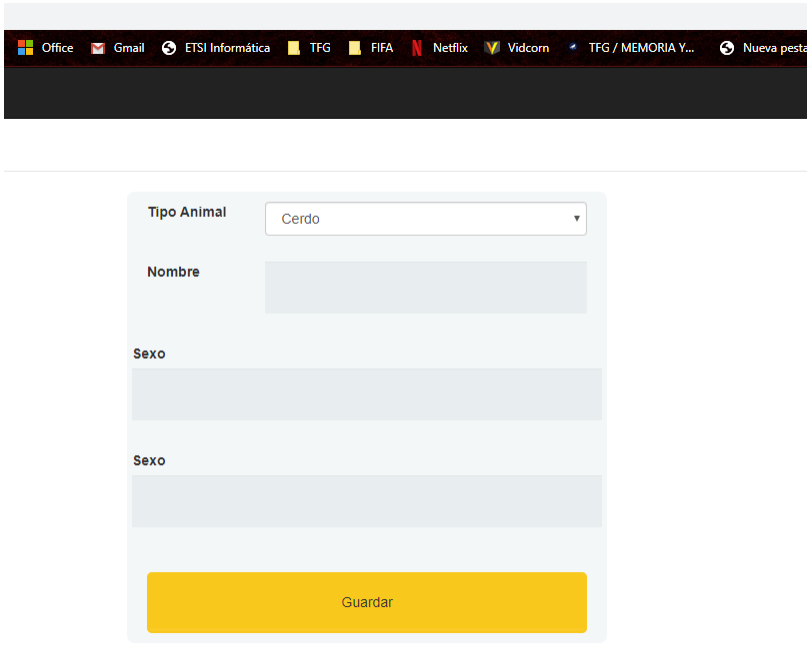

Figura 5.11: Vista de insertar animales

En último lugar encontramos la pestaña de simulación, la cual solo puede ser usada por el administrador. Para realizar la simulación necesitamos dos parámetros, el número de consultas que vamos a realizar y el número de inserciones. Una vez ejecutada la simulación en función de los parámetros, se crean consultas e inserciones de forma completamente aleatoria, pudiendo relazarse sobre usuarios, animales, transportes, sacrificios, inspecciones y alimentaciones. Al terminar la simulación aparece el tiempo que ha llevado realizarla en pantalla.

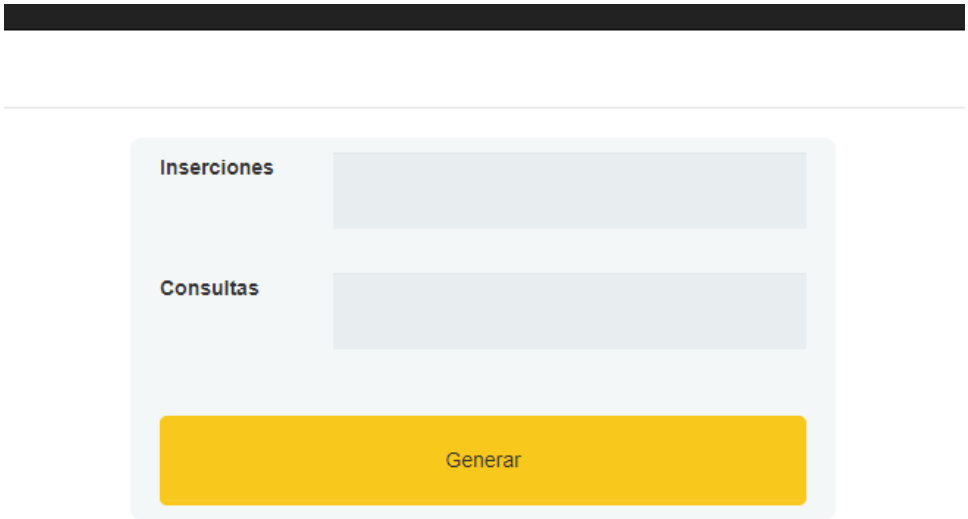

Figura 5.12: Vista de realizar simulación

6

Resultados

6.1 Pruebas de rendimiento

6.1.1 Introducción

Finalmente, las pruebas de rendimiento las hemos realizado en uno de nuestro ordenador de sobremesa con las siguientes especificaciones:
-Procesador: Intel Core i7-6700 3.4GHz
-Gráfica: NVIDIA GTX 1060 6 GB
-Memoria ram: 24 GG DDR4 2400Mhz

Todas las pruebas las hemos realizado utilizando las pestañas que hemos creado de simulación, desde la cual se crean inserciones y consultas de forma aleatoria. Hemos realizado pruebas para los valores de 100, 1000 y 10000 operaciones, en primer lugar, solo de inserciones, en segundo lugar, solo de consultas y en último lugar, tanto de consultas como de inserciones, realizando de ambas el mismo número de operaciones ya sea 100, 1000 o 10000. Todos los tiempos mostrados en todas las gráficas son en milisegundos.

6.1.2 MySQL

MySQL nos ha dado un rendimiento medio en cuanto a tema de inserciones, que era lo que nos esperábamos, sin embargo, en cuanto a consultas nos ha dado un rendimiento muy bueno, obteniendo unos tiempos casi nulos.

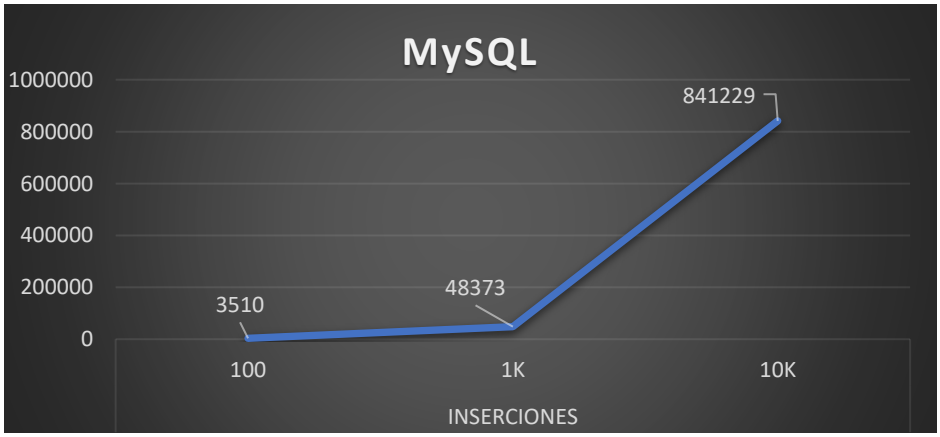

Figura 6.1: Gráfica de tiempo empleado en realizar inserciones en MySQL.

Figura 6.2: Gráfica de tiempo empleado en realizar consultas en MySQL.

Figura 6.3: Gráfica de tiempo empleado en realizar consultas e inserciones en MySQL.

6.1.3 Cassandra

Cassandra nos ha dado unos rendimientos algo peor de los esperado, de hecho, en un primer momento tuvimos problemas de rendimiento ya que al abrir la conexión a la base de datos tardaba bastante pero finalmente lo conseguimos solucionar, sin embargo, los rendimientos que nos ha dado en cuanto a las consultas son inferiores a los que esperábamos.

Figura 6.4: Gráfica de tiempo empleado (ms) en realizar consultas en Cassandra.

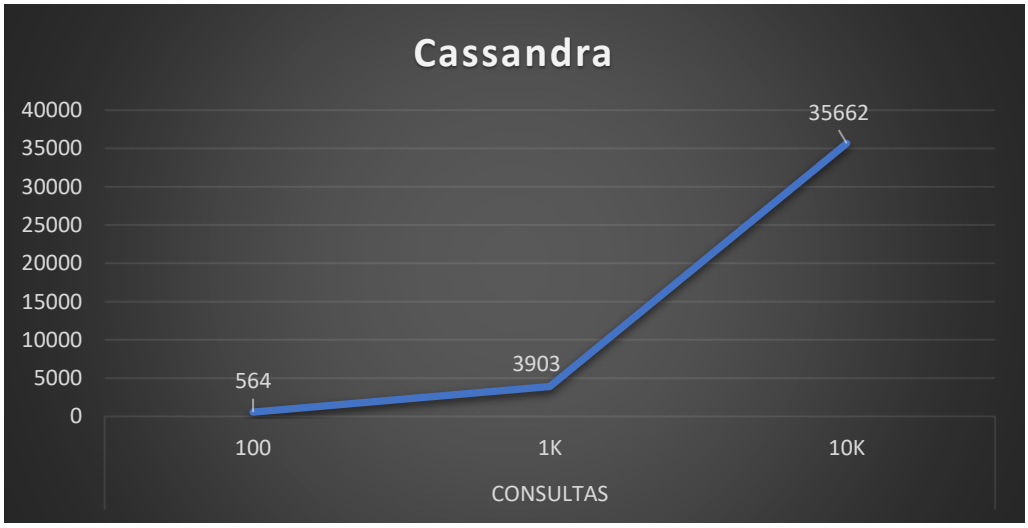

Figura 6.5: Gráfica de tiempo empleado (ms) en realizar consultas en Cassandra.

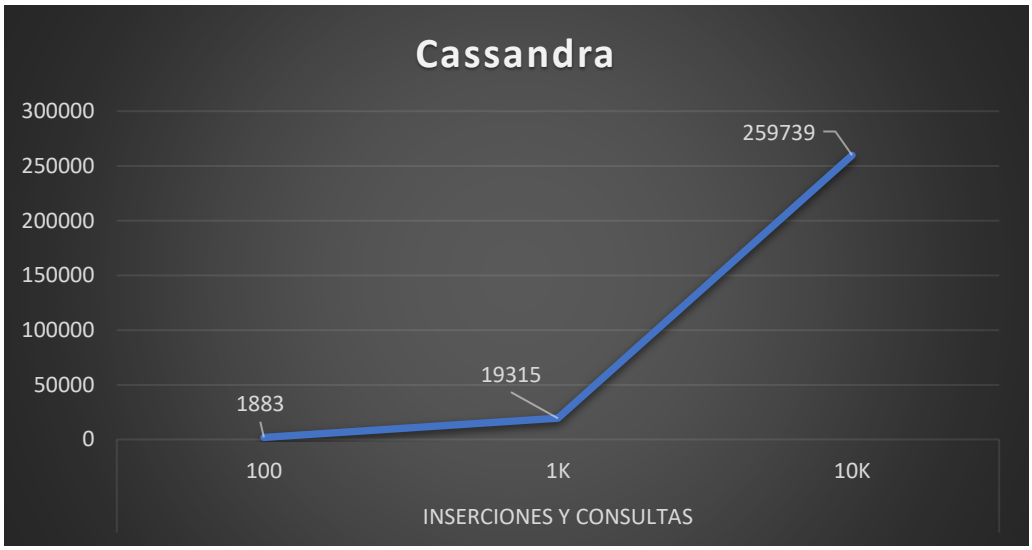

Figura 6.6: Gráfica de tiempo empleado (ms) en realizar consultas e inserciones en Cassandra.

6.1.4 Ethereum

El rendimiento de Ethereum es bastante malo como ya esperábamos, ya que al ser una tecnología nueva aún no está demasiado optimizada y si a eso le sumamos lo que implica el procedimiento de insertar y consultar datos pues nos encontramos con unos tiempos bastante elevados, llegando incluso a tener que dejar el ordenador encendido más de 17 horas seguidas para que completase el proceso.

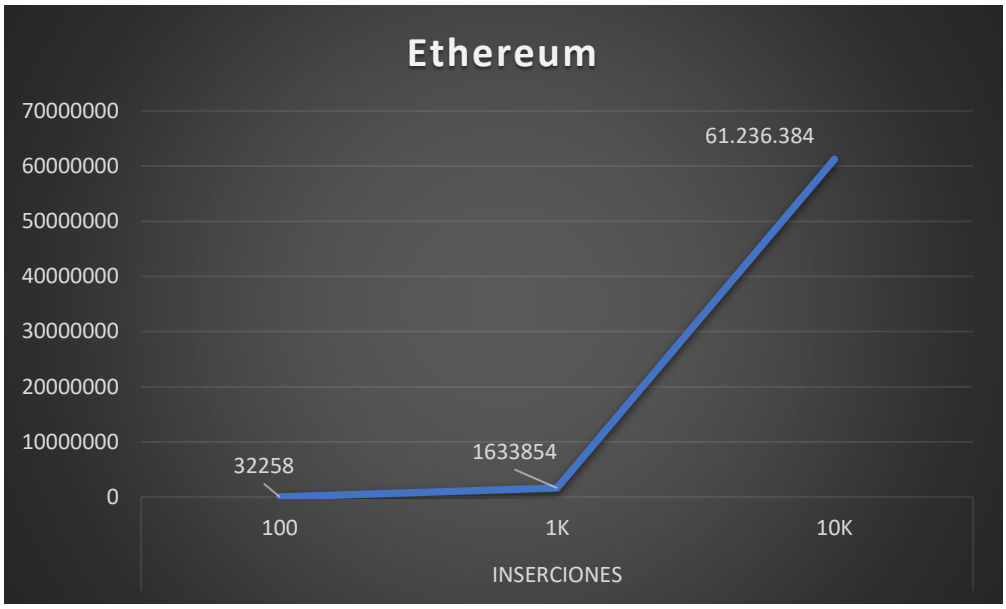

Figura 6.7: Gráfica de tiempo empleado (ms) en realizar inserciones en Ethereum.

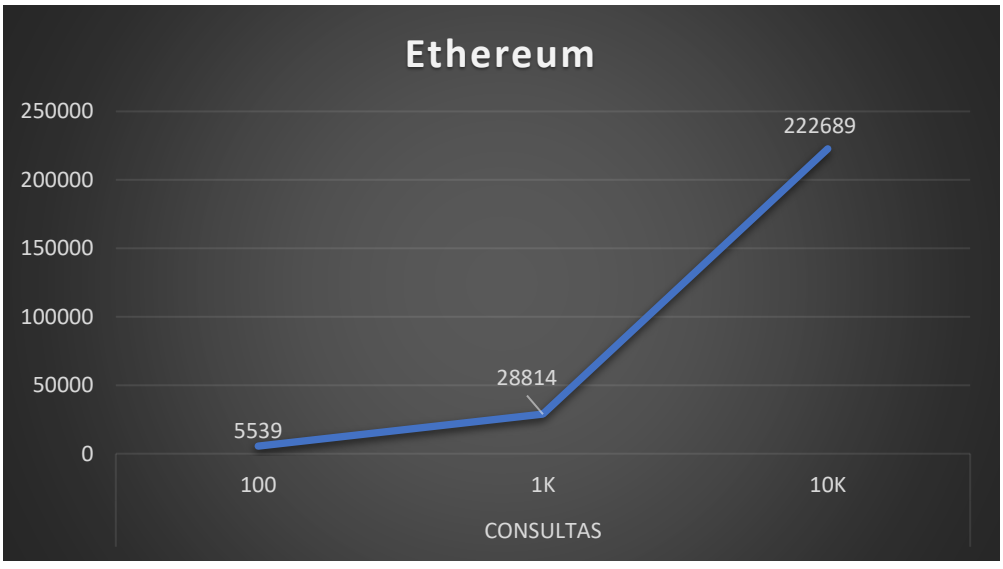

Figura 6.8: Gráfica de tiempo empleado (ms) en realizar consultas en Ethereum.

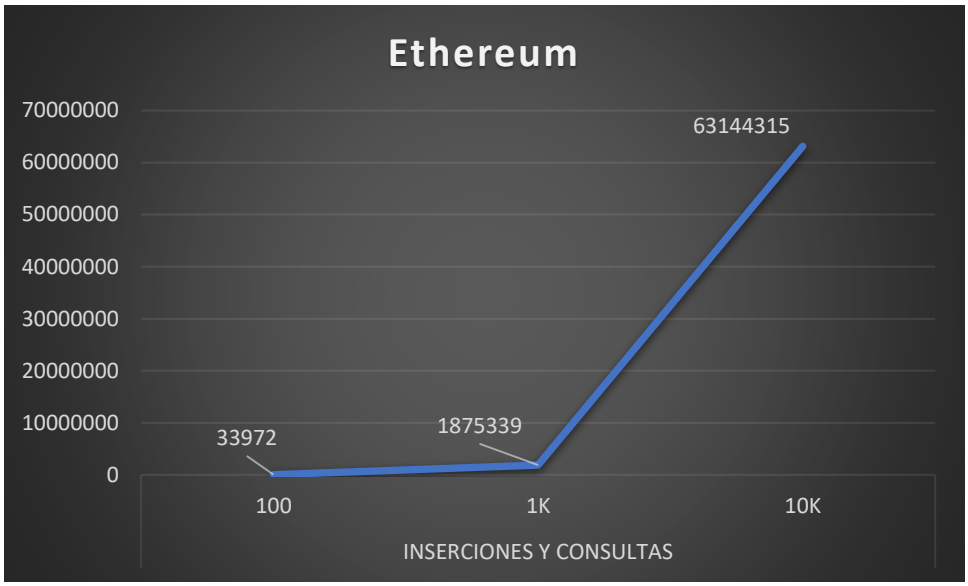

Figura 6.9: Gráfica de tiempo empleado (ms) en realizar consultas e inserciones en Ethereum.

6.2 Comparativas de rendimiento

Podemos ver como las más rápida a la hora de realizar inserciones es Cassandra, como cabría esperar y en último lugar encontramos Ethereum. Hemos aplicado la escala logarítmica en base 10 al siguiente gráfico, ya que había demasiada diferencia entre los valores y no se apreciaba bien las líneas de MySQL y Cassandra ya que aparecían pegadas al eje x.

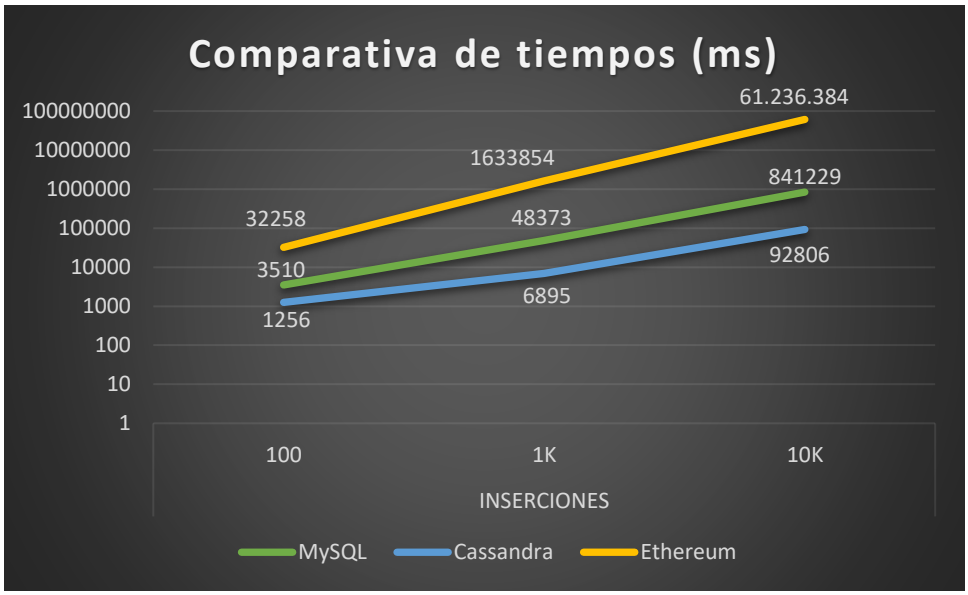

Figura 6.10: Gráfica comparativa de tiempos al realizar inserciones entre las diferentes bases de datos

En cuanto a consultas, encontramos que la más rápida es MySQL, aunque nosotros esperábamos que la más rápida fuese Cassandra, sin embargo, nos han salido unos tiempos bastante superiores, aunque lejanos a los ofrecidos por Ethereum.

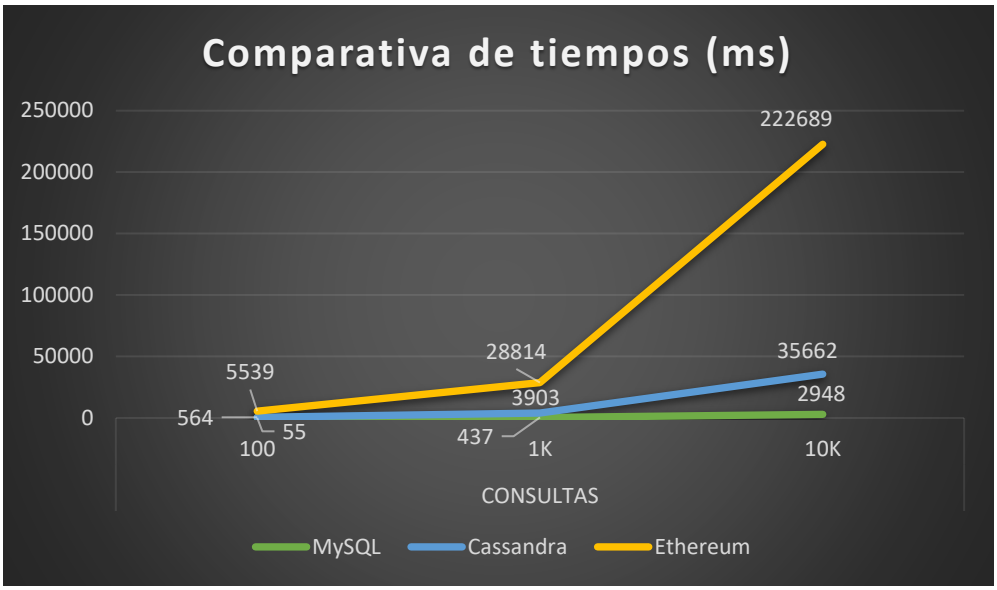

Figura 6.11: Gráfica comparativa de tiempos al realizar consultas entre las diferentes bases de datos

Por último, al mezclar inserciones y consultas, como las inserciones es lo que más tiempo lleva ambas bases de datos pues la gráfica es muy parecida a la que tenemos en cuanto a inserciones, es por ello por lo que también le hemos aplicado la escala logarítmica en base 10.

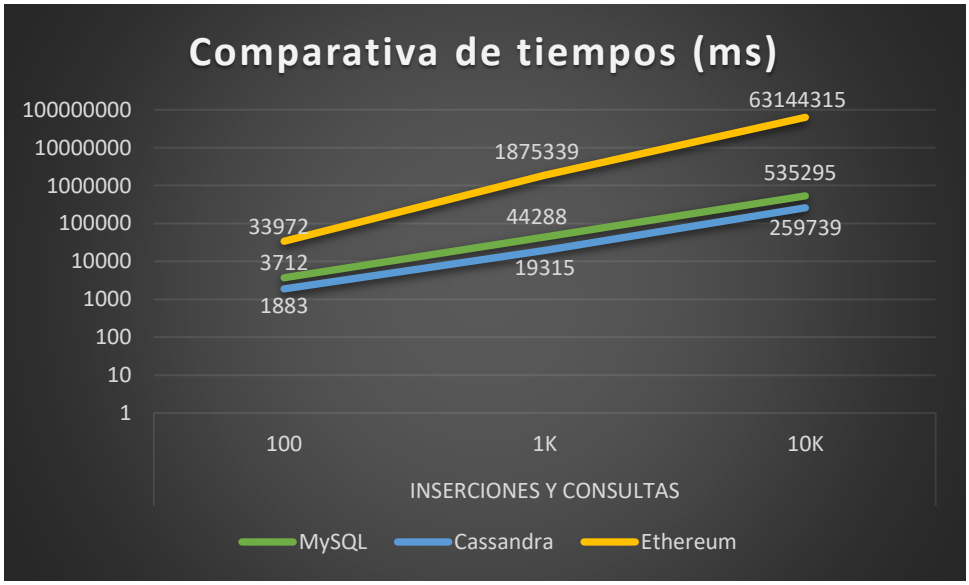

Figura 6.12: Gráfica comparativa de tiempos al realizar consultas e inserciones entre las diferentes bases de datos

6.3 Dificultades

A lo largo del desarrollo de las diferentes páginas web nos hemos encontrado con numerosos inconvenientes, en unos casos más y en otros casos menos, a continuación, desarrollaremos caso por caso lo que nos hemos encontrado.

En cuanto al desarrollo de MySQL fue bastante asequible ya que las tecnologías usadas ya las habíamos aplicado en algunas asignaturas de la carrera. Sin embargo, a la hora de instalar XAMPP no dio varios errores relacionados con algunos puertos que utiliza, que tuvimos que solucionar cambiándolos en algunos archivos de configuración.

Al desarrollar con Cassandra tuvimos algunos problemas, el primero de ellos fue con respecto al servidor de aplicaciones ya que nosotros usábamos GlassFish y no hubo manera de que la aplicación arrancase usándolo, finalmente cambiamos a Apache TomEE. Otro error fue a la hora de importar las librerías de Cassandra que no hubo otra manera de hacerlo que usando Maven. Una vez configurado y arrancado el proyecto, nos resultó bastante sencillo su desarrollo ya que el lenguaje CQL es muy parecido al SQL, así que nuestro desarrollo se basó en indagar por su API.

En cuanto a Ethereum fue sin duda el que más trabajo nos costó su desarrollo, todo fue debido a la poca información que hemos encontrado y la cantidad de nuevas tecnologías que hemos tenido que aprender como Node.js, Solidity, Truffle y Web3. A su vez, también nos llevó mucho más tiempo entender su estructura y funcionamiento ya que resulta mucho más complejo y se escapa de la lógica seguida por otras bases de datos.

7

Conclusiones

Durante la realización de este proyecto de fin de carrera, hemos podido aplicar numerosos conocimientos adquiridos durante la carrera, ya sea en algunos casos por aplicar algunas tecnologías aprendidas en clase como en otros casos a aplicar capacidad de análisis y síntesis que hemos ido desarrollando, así como la capacidad de autoaprendizaje a la hora de enfrentarnos a nuevas tecnologías. Es por ello, por lo que tras las realizaciones de esto proyecto hemos adquirido numerosos conocimientos que probamente a la hora de volvernos a enfrentar antes las mismas situaciones aplicaríamos de una forma más efectiva y exitosa.

A partir de todo el trabajo realizado, vamos a intentar exponer la ventajas y desventajas que hemos sacado en claro de lo que nos puede ofrecer cada tecnología y así poderlas aplicar dependiendo de las necesidades del proyecto que deseemos desarrollar.

En cuanto a términos de puro rendimiento probablemente la mejor tecnología sería Cassandra, ya que es la que nos ofrece unos tiempos en realizar las operaciones inferiores, a su vez, no es tan estricta ni tiene las restricciones que tiene una base de datos relacional, también al ser distribuida no tiene un único punto de fallo y puede escalar de manera lineal. Es por ello por lo que esta tecnología sería ideal para bases de datos muy grande que requieran de un buen rendimiento y que no necesiten un control muy estricto sobre lo datos.

MySQL puede ser una opción muy interesante en algunos casos, ya que por norma general es la tecnología de la que encuentra más personal formado y por lo tanto la empresa no necesita gastar dinero en formación. Es por ello por lo que probablemente sea la que se necesitas una menor inversión en su desarrollo y mantenimiento. También nos proporcionan consistencia y evitan la redundancia de datos, esto puede ser determinante en aplicaciones que dependan de que alguna información almacenada cumpla ciertas condiciones.

Por último, lugar, Ethereum puede que a priori no se vea como una buena opción debido a su rendimiento y a la necesidad y dificultad de su aprendizaje. Sin embargo, ofrece ciertas propiedades que otras bases de datos no pueden ofrecer. Es por ello,

por lo que hay aplicaciones que necesitan de su uso sí o sí y nos abre un nuevo abanico de posibilidades de desarrollo. Ethereum es la mejor opción tanto en aplicaciones que se necesite eliminar un intermediario que gestione todas las operaciones como en el caso de los bancos, como en aplicaciones que necesiten eliminar la capacidad de poder alterar la información de la base de datos y a mayor número de nodos se convierte en una tarea casi imposible.

Estas son las ventajas y desventajas que hemos sacado en claro al realizar el proyecto y los casos en los que creemos que sería interesante el uso de una u otra base de datos. Como podemos observar, hemos necesitado la aplicación y el aprendizaje de numerosas tecnologías que hasta la fecha nos veíamos incapaces de utilizar. Es por ello, por lo que consideramos que gracias a la realización de este TFG hemos aprendido una gran cantidad de cosas y nos ha ayudado a demostrarnos a nosotros mismos que somos capaces de enfrentarnos a proyecto tan ambicioso como este.

Bibliografía

[1] https://actforfood.carrefour.es/Por-que-actuar/BLOCKCHAIN-ALIMENTARIO
[2] https://www.masadelante.com/faqs/base-de-datos
[3] https://www.universidadviu.es/los-principales-tipos-base-datos/
[4] https://www.oracle.com/es/database/what-is-a-relational-database/
[5] https://es.quora.com/Cu%C3%A1les-son-las-5-bases-de-datos-SQL-y-Nosql-m%C3%A1s-populares-m%C3%A1s-utilizadas
[6] https://visualeo.com/blockchian-por-que-y-como-surge/
[7] https://basededatosutp26.wordpress.com/mysql/
[8] https://es.wikipedia.org/wiki/PrimeFaces
[9] https://www.paradigmadigital.com/dev/cassandra-la-dama-de-las-bases-de-datos-nosql/
[10] https://itrenting.com/que-es-blockchain/
[11] https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/
[12] https://www.cysae.com/el-minado-en-blockchain/
[13] https://es.wikipedia.org/wiki/Java_EE
[14] https://es.wikipedia.org/wiki/JavaServer_Faces
[15] https://es.wikipedia.org/wiki/NetBeans
[16] https://es.wikipedia.org/wiki/XAMPP
[17] https://www.trufflesuite.com/
[18] https://www.tokens24.com/es/cryptopedia/basics/quien-creo-ethereum
[19] https://academy.bit2me.com/que-son-los-smart-contracts/
[20] https://solidity-es.readthedocs.io/es/latest/
[21] https://web3js.readthedocs.io/en/v1.2.6/

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA