eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Desarrollo de un entorno virtual para el aprendizaje NoSQL: análisis de diferentes sistemas NoSQL y aplicación web de carácter didáctico con contenido NoSQL

Autora

Amaia Alfageme Palacio

2019

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Desarrollo de un entorno virtual para el aprendizaje NoSQL: análisis de diferentes sistemas NoSQL y aplicación web de carácter didáctico con contenido NoSQL

Autora

Amaia Alfageme Palacio

Directora

Arantza Illarramendi Echave

Resumen

Este Trabajo de Fin de Grado se ha desarrollado con el objetivo de crear un sistema real en torno a los sistemas de gestión de bases de datos *NoSQL* desde un enfoque didáctico. Para ello se ha realizado un análisis de rendimiento de diferentes sistemas y se ha generado una aplicación web real de carácter pedagógico.

Por un lado, se ha desarrollado un análisis de rendimiento de los sistemas *MongoDB*, *Apache Cassandra* y *Neo4j* como principales representantes *NoSQL* de los sistemas gestores orientados a documentos, de clave-valor y orientados a grafos respectivamente. A través de este análisis se ha evaluado la eficiencia de los tres sistemas en dos escenarios distintos, uno de ellos utilizando un conjunto de datos genérico y otro con un conjunto de datos en forma de serie temporal.

Por otro lado, se ha creado una aplicación web para usuarios que quieren iniciarse en el mundo de la tecnología *NoSQL* donde poder estar informado de novedades en torno a esta tecnología, acceder a contenido didáctico como vídeos o *blogs* educativos y poder evaluar los conocimientos adquiridos en torno a *NoSQL*.

Agradecimientos

Antes de comenzar, me gustaría agradecer especialmente a mi tutora en este Trabajo de Fin de Grado, Arantza Illarramendi, por la ayuda y supervisión que he recibido por su parte.

En primer lugar, quiero agradecer a los miembros del grupo de investigación *BDI* por la buena acogida que me han dado desde un primer momento. Especialmente agradecer a Borja Díez, Víctor Ramirez y Kevin Villalobos por la ayuda recibida en los meses que hemos compartido escritorio, y por la amabilidad con la que han sabido responder mis dudas.

En segundo lugar, me gustaría agradecer a mi familia, especialmente a mi madre y a mi padre, por confiar en mí y por apoyarme con este proyecto y con el resto de retos personales que tomo en mi día a día.

En tercer lugar, agradecer a mis amigos por el apoyo recibido durante el desarrollo de este proyecto, por acompañarme en los descansos y por estar conmigo cuando he necesitado desconectar.

Por último, agradecer a todas las personas que directa o indirectamente han formado parte de este proyecto.

Índice general

Anexos

Índice de figuras

Índice de tablas

Índice de código fuente

1. CAPÍTULO

Introducción

El ser humano lleva siglos recopilando información desde la existencia de la escritura cuneiforme (el sistema más antiguo de escritura conocido hasta la fecha). El crecimiento del sector tecnológico evidente en los últimos años ha supuesto un gran cambio en la concepción del almacenamiento de datos para la industria y las empresas. Esto ha provocado un aumento desmesurado del volumen de los datos, que no pueden ser tratados con los sistemas de gestión tradicionales. Surge así la necesidad de nuevas formas de almacenamiento y organización de los datos, dando lugar al concepto de *Big Data* [CampusBigData, 2017].

Los datos son una fuente de información muy enriquecedora de la cual se puede obtener mucho conocimiento más allá del propio dato. Las organizaciones se esfuerzan para trabajar los datos y obtener respuestas de éstos en forma de conocimiento valioso utilizando las tecnologías del *Big Data*.

Una de las tecnologías que permite gestionar el almacenamiento, la organización y la recuperación masiva de los datos que el *Big Data* implica es *NoSQL*. Los sistemas gestores de bases de datos *NoSQL* (*Not Only SQL*) proporcionan gran flexibilidad, facilitando la dinamización de los esquemas de las bases de datos, que se organizan en registros donde sus atributos se presentan de diferentes maneras en función de las necesidades existentes [Bendera et al., 2014].

Dada la repercusión que *NoSQL* está teniendo entre las tecnologías de la información, a través de este Trabajo de Fin de Grado se pretende crear un entorno relacionado con el contexto *NoSQL* que permita acercar esta tecnología a los usuarios que quieran familiari-

zarse con ella, tanto de manera práctica como desde una perspectiva analítica.

La memoria del proyecto se estructura en 9 capítulos y 5 anexos, organizados de la siguiente manera:

- En el **capítulo 1** se relata una breve introducción del proyecto, así como una presentación de la organización de la estructura de la memoria.
- En el **capítulo 2** se describe el contexto del proyecto a modo de explicación de los motivos por los que se desarrolla un proyecto como este.
- En el **capítulo 3** se describen los objetivos del proyecto extensamente y se realiza un análisis del alcance del proyecto y de la planificación a seguir en su desarrollo.
- En el **capítulo 4** se expone el trabajo realizado respecto al análisis de la comparativa de rendimiento de los sistemas de gestión de bases de datos utilizados en el proyecto.
- En el **capítulo 5** presenta la labor realizada en el desarrollo de la aplicación web, desde el diseño inicial hasta la finalización de ésta.
- En el **capítulo 6** se muestran las pruebas de funcionamiento realizadas sobre la aplicación web y el entorno de la comparativa de rendimiento.
- En el **capítulo 7** se exponen los datos de seguimiento y control analizados durante el desarrollo del proyecto.
- En el **capítulo 8** se exponen las conclusiones obtenidas tras la realización del proyecto, así como un análisis sobre posibles mejoras y líneas futuras del proyecto.
- En el **anexo A** se explican los procesos de preparación de la máquina virtual junto con los diferentes sistemas de gestión de bases de datos instalados.
- En el **anexo B** se incluyen las sentencias generadas para la preparación de las bases de datos y el análisis de rendimiento de los distintos sistemas *NoSQL*.
- En el **anexo C** se incluyen los diagramas de secuencia asociados a las diferentes funcionalidades que presenta la aplicación web.
- En el **anexo D** se explica el proceso de preparación del entorno de desarrollo de la aplicación web.
- En el **anexo E** se muestran las interfaces asociadas a cada caso de uso de la aplicación web.

2. CAPÍTULO

Contexto

En este capítulo se busca presentar unas bases sobre la tecnología *NoSQL* y algunos aspectos fundamentales que ocupan lugar en el ámbito de trabajo de este proyecto, con el fin de aclarar el contexto en el que se va a trabajar.

A través de los sistemas relacionales, los datos de las bases de datos se gestionan siguiendo un esquema estructurado, priorizando determinadas propiedades asociadas a la gestión de los datos, como son las propiedades **ACID** (*Atomicy, Consistency, Isolation, Durability*). Sin embargo, con motivo de abordar el concepto de *Big Data* surge la necesidad de dar con una tecnología que sea capaz de gestionar grandes volúmenes de datos, donde la prioridad no es cumplir las propiedades mencionadas, si no que surge otro tipo de necesidades [Martin, 2013].

Los sistemas de gestión de bases de datos *NoSQL* priman otras características más apropiadas al contexto en el que se utilizan [Oracle, 2016]:

- **Consitencia Eventual**: la consistencia global del sistema se garantiza a largo plazo, es decir, todos los cambios llegarán a todos los nodos del sistema con el tiempo. Esto da lugar al problema de la *lectura sucia*[1].
- **Flexibilidad en el esquema**: no es necesario establecer una estructura de almacenamiento con un esquema concreto para todos sus registros, si no que es posible dinamizar los esquemas de almacenamiento personalizando los elementos y atributos que pueden contener.

[1] Situación que se produce cuando una transacción lee un dato modificado por otra transacción concurrente que no ha finalizado.

- **Escalabilidad horizontal**: posibilidad de incrementar el rendimiento del sistema añadiendo más nodos servidores.
- **Estructura distribuida**: posibilidad de almacenar los datos en diferentes nodos servidores.
- **Tolerancia a fallos y disponibilidad**: es preciso que las bases de datos estén preparadas para continuar con su funcionamiento pese a los problemas y situaciones que puedan darse y que afecten al rendimiento de la base de datos y al propio sistema.

La utilización de Sistemas Gestores de Bases de Datos (SGBD) no relacionales como lo son los sistemas *NoSQL*, permiten tratar grandes volúmenes de datos de manera más eficiente que el modo en el que los datos son examinados a través de los sistemas relacionales. Existen diferentes tipos de sistemas *NoSQL*, basados en la forma en la que éstos almacenan los datos. Así, existen los SGBD orientados a documentos, de clave-valor, orientados a grafos, orientados a objetos o para series temporales. Este proyecto se va a focalizar en el uso de los tres primeros, concretamente en MongoDB como SGBD orientado a documentos, Apache Cassandra como SGBD de clave-valor y Neo4j como SGBD orientado a grafos, que son los principales representantes de cada tipo de sistema.

Los SGBD **orientados a documentos** guardan los datos en documentos individuales en formatos como JSON o XML con estructuras jerárquicas, permitiendo realizar búsquedas por clave-valor o consultas más avanzadas sobre el propio documento (Fig. 2.1).

Sánchez, J. (-). Fundamentos de los Sistemas Gestores de Bases de Datos [Figura]. Recuperado de https://jorgesanchez.net/manuales/abd/bases-sgbd.html.

Figura 2.1: Representación del modo de almacenamiento de datos de un SGBD orientado a documentos

Este tipo de sistemas permite almacenar los datos en documentos que no tienen por qué

seguir el mismo esquema, lo que proporciona cierta flexibilidad a la hora de almacenar los datos. Algunos usos reales de este tipo de sistemas son el almacenamiento de productos para el comercio electrónico (*GAP*), la gestión de blogs y contenido digital (*Google*), etc.

Los SGBD **clave-valor** son los más populares en el ámbito de las bases de datos *NoSQL*. Éstos guardan los datos en forma de pares de valores (clave y valor), es decir, a partir de una clave se obtiene su valor (Fig. 2.2).

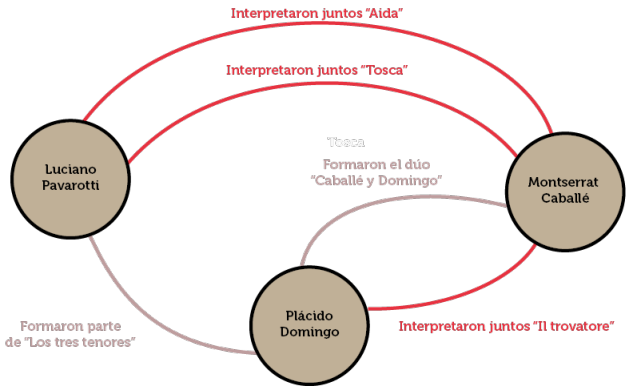
Fila	Clave	Datos personales		Otros	
1	1231	Nombre	Luciano	País	Italia
		Apellidos	Pavarotti		
2	9234	Nombre	Plácido	Ciudad	Madrid
		Apellidos	Domingo	País	España
		Fecha_n	21/1/1941		
3	9234	Nombre	Montserrat	Ciudad	Barcelona
		Apellidos	Caballé	email	montse@gmail.com
		Fecha_n	12/4/1933		

Sánchez, J. (-). Fundamentos de los Sistemas Gestores de Bases de Datos [Figura]. Recuperado de https://jorgesanchez.net/manuales/abd/bases-sgbd.html.

Figura 2.2: Representación del modo de almacenamiento de datos de un SGBD de tipo clave-valor

Los sistemas de tipo clave-valor proporcionan mecanismos que agilizan el acceso al contenido de la base de datos a través de una clave, lo que favorece las operaciones de lectura. Algunos usos reales de este tipo de sistemas son la mejora de la experiencia de usuario (*Netflix*), la monitorización de la actividad de usuarios (*Cern*), etc.

Los SGBD **orientados a grafos** guardan los datos en entidades representadas por nodos, que están unidos por aristas que representan las relaciones entre entidades, haciéndose más fácil recorrer la BD (Fig. 2.3).



Sánchez, J. (-). Fundamentos de los Sistemas Gestores de Bases de Datos [Figura]. Recuperado de https://jorgesanchez.net/manuales/abd/bases-sgbd.html.

Figura 2.3: Representación del modo de almacenamiento de datos de un SGBD orientado a grafo

Los sistemas orientados a grafos presentan gran utilidad en escenarios donde es clave la relación entre los elementos que componen la base de datos. Este sistema facilita el acceso a los datos recorriendo los nodos a través de sus relaciones, proporcionando una perspectiva más amplia del contenido. Algunos de los usos reales que este tipo de sistemas tiene en la actualidad es la detección de fraude en el ámbito de la banca, recomendaciones en tiempo real para el comercio online (*Ebay*), redes sociales (*LinkedIn*), etc.

La facilidad en la gestión de grandes cantidades de datos que ofrecen los sistemas *NoSQL* hacen que cada vez sean más organizaciones las interesadas en adoptarlos como sistema de base de datos. Esto conlleva un interés general por conocer estas tecnologías y adentrarse en el mundo de los sistemas de gestión de bases de datos *NoSQL*, para conocer aspectos como la estructura de almacenamiento y la sintaxis de las consultas de cada uno de ellos.

Cada tipo de SGBD *NoSQL* (refiriéndose a cada tipo con los modos de almacenamiento de datos previamente descritos) cumple una serie de requisitos y son más eficientes con un tipo de datos que con otro. Surge un interés por evaluar sus características y la manera en la que las empresas involucradas puedan obtener los mayores beneficios a modo de información valiosa.

Por motivos como los descritos se va a desarrollar este proyecto, con el fin de facilitar el acceso rápido a las personas interesadas en tomar contacto con las tecnologías *NoSQL*, que quieran aprender a utilizarlas y conocer su estructura para entender la utilidad que pueden tener en diferentes contextos.

3. CAPÍTULO

Planteamiento del proyecto

En este capítulo se va a presentar la planificación y la estructura que va a seguir el proyecto.

Por un lado, se realiza una descripción del proyecto describiendo de qué partes va a constar y la definición de cada una de las partes. Además, se detallan los objetivos específicos y generales del proyecto, definiendo también los límites del proyecto a través de su alcance.

Por otro lado, se describe la metodología de organización del trabajo y la planificación de las tareas que se van a desarrollar, así como el modo de gestión de riesgos que se va a implementar y la descripción de las tecnologías que se van a utilizar.

3.1. Descripción del proyecto

El propósito de este proyecto es proporcionar a usuarios interesados en conocer la tecnología *NoSQL* un entorno virtual con el que aprender a trabajar con esta tecnología. Esto va a ser posible tanto de manera práctica a través de un entorno virtual habilitado para trabajar con estas tecnologías, como desde una perspectiva más analítica trabajando con los diferentes sistemas, comprendiendo su modo de funcionamiento.

Por una lado, de cara a la preparación de un entorno de trabajo enfocado al conocimiento de diferentes SGBD *NoSQL* diferenciados por su tipo, en primer lugar se va a preparar una **máquina virtual con Windows 10**[1]**, con MongoDB, Apache Cassandra y Neo4j** insta-

[1]También se va a generar secundariamente una equivalente pero con el sistema operativo Linux Mint

lados (Fig. 3.1), así como un cliente para cada sistema servidor que facilite la interacción con la base de datos.

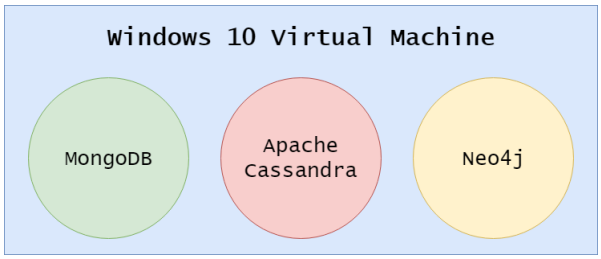


Figura 3.1: Representación de la máquina virtual

Además, se va a desarrollar una **aplicación web que sirva como herramienta pedagógica acerca de los sistemas de gestión de bases de datos *NoSQL*** a diferentes usuarios registrados, ofreciendo un acceso directo a la descarga de los SGBD mencionadas anteriormente, sus correspondientes clientes gráficos y fuentes de datos de ejemplo en formatos aceptados por estos sistemas para poder probarlas. La aplicación web también incluirá otras secciones de carácter educativo, como enlaces a vídeos didácticos, tutoriales con contenido sobre la sintaxis de cada sistema, control de acceso de usuarios, visualización del contenido consultado, etc (figura 3.2).

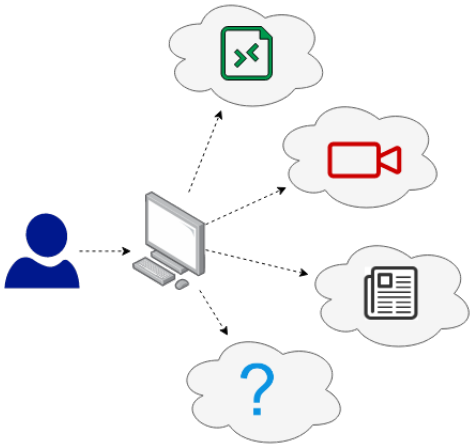


Figura 3.2: Representación simplificada de las funcionalidades de la aplicación web

Por último, se va a **diseñar un esquema de base de datos diferente para cada tipo de sistema gestor de base de datos** a partir de un análisis de requisitos previo en relación a un mismo conjunto de datos común, para poder **evaluar el rendimiento de cada base**

de datos ejecutando diferentes operaciones tanto de selección, inserción, actualización o borrado en cada una de ellas, comprobando la eficiencia de unas frente a las otras en cada tipo de operación y para cada conjunto de datos (Fig. 3.3).

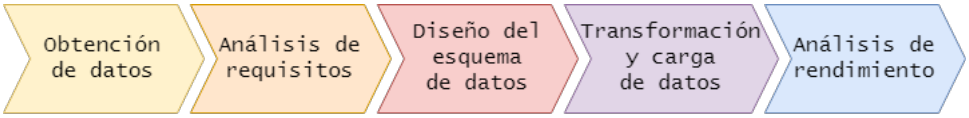


Figura 3.3: Fases de la actividad de análisis y diseño

3.2. Objetivos

Los objetivos que se van a desarrollar durante la ejecución de este proyecto se dividen en objetivos técnicos asociados a actividades concretas del proyecto, y en objetivos de formación ligados a propósitos de carácter personal.

Por un lado, en cuanto a los **objetivos técnicos**, el objetivo principal de este proyecto es crear un **sistema real en torno a los sistemas de gestión de bases de datos *NoSQL*** que proporcione ayuda y facilite la gestión e iniciación en éstos a aquellas personas que quieran conocerlos.

Como objetivo secundario se encuentra la realización de una **comparativa del rendimiento** entre los sistemas de bases de datos seleccionados. En esta comparativa se tendrá en cuenta tanto el tiempo de ejecución de diferentes operaciones de selección, actualización, inserción y borrado, como el espacio de almacenamiento que implica cada una de estas operaciones. Este proceso de comparación se va a realizar utilizando tanto bases de datos con series temporales como con datos de otra índole, con el fin de analizar y estudiar cuán preparado está cada sistema para trabajar con tipos de datos de diferente índole.

Por otro lado, en cuanto a los **objetivos de formación** el principal es el **aprendizaje de los diferentes tipos de sistemas de gestión de bases de datos *NoSQL*** existentes, profundizando sobre los tres tipos con los que se va a trabajar en este proyecto. Asociado a este aprendizaje se encuentra la toma de **contacto con la sintaxis de las consultas de los sistemas utilizados**, al igual que la comprensión de la estructura interna de almacenamiento con la que funciona cada uno de estos tipos. Estos métodos de almacenamiento influyen en la ejecución de las diferentes operaciones, que actúan de distinta manera en función del diseño del esquema de datos. También se va a **aprender a generar un diseño**

para cada base de datos asociado al análisis de requisitos a realizar previamente sobre el conjunto de datos con el que se está trabajando.

Otro objetivo secundario es la toma de contacto con herramientas facilitadoras del desarrollo web, como la plataforma de desarrollo de aplicaciones web Firebase, al igual que diferentes librerías y frameworks que ayuden a gestionar el diseño de una aplicación web con interfaz de usuario Material Design, como **Vue.js**, **MDBVue** para la parte *front-end* y **Node.js** y **Express** para el *back-end*.

Como último objetivo secundario pero no menos importante, con la elaboración de este proyecto se va a comprobar la **capacidad de gestión y desarrollo personal de un proyecto de esta envergadura** teniendo en cuenta todos los aspectos que afectan al inicio y finalización del proyecto. Se va a demostrar la habilidad de planificación y seguimiento sobre las tareas desempeñadas durante el desarrollo del mismo, desde la identificación de las tareas a realizar como a la gestión del alcance y tiempo sobre las mismas.

3.3. Alcance

Es importante definir los límites del contenido del proyecto, identificando qué tareas se van a hacer y qué aspectos no se van a incluir, para poder crear una planificación que describa de manera detallada los procesos que se van a desarrollar, así como la organización temporal de dichas tareas.

Este proyecto tiene como objetivo la creación de un sistema que proporcione ayuda a las personas que quieran iniciarse en las tecnologías *NoSQL*. Se han seleccionado los sistemas de gestión de bases de datos MongoDB, Apache Cassandra y Neo4j.

Para desarrollar este sistema se creará una aplicación web con gestión y autenticación de usuarios que ofrezca la posibilidad de descarga directa de las diferentes bases de datos y clientes asociados. Se añadirán diferentes apartados en la aplicación web que incluyan contenido educativo asociado al conocimiento de las tecnologías *NoSQL*.

Se realizará un análisis de requisitos sobre varios conjuntos de datos con el fin de establecer un diseño de base de datos apropiado para cada uno de los conjuntos de datos en cada SGBD. Además, se evaluará el rendimiento de los diferentes SGBD a través de las bases de datos generadas.

3.3.1. Exclusiones

El contenido educativo y noticias incluidas en el sistema a generar no es contenido creado por mí, si no que se trata de contenido extraído de diferentes fuentes en Internet, del cual se realizará una supervisión y se incluirá en la aplicación web cuando cumpla unos requisitos establecidos.

3.3.2. Paquetes de trabajo

Para realizar una buena planificación es necesario identificar desde el principio cuáles van a ser las tareas que se van a desempeñar, con el fin de saber organizarlas adecuadamente para lograr el mejor resultado posible en el proyecto.

A la hora de organizar las tareas, primeramente se han identificado 7 fases del proyecto. Cada fase agrupa una serie de paquetes de trabajo relacionados entre sí por el contenido de las tareas que cada uno ocupa y el objetivo común del conjunto de ellas.

A continuación se especifican las fases y los paquetes de trabajo identificados, así como las tareas que incluye cada paquete de trabajo con una breve descripción del objetivo de éstas.

Fase de Gestión del Proyecto

Planificación (P): en este paquete de trabajo se agrupan las tareas relacionadas con el planteamiento inicial del proyecto y las primeras decisiones que se toman sobre éste, así como las posibles alteraciones que pudieran surgir en el modo de organización.

- P.1: Tareas relacionadas con la identificación de requisitos, toma de decisiones iniciales, análisis de la información y resolución de dudas.
- P.2: Planificación inicial orientada a la preparación del entorno de desarrollo del proyecto y la generación del informe del TFG.
- P.3: Actualización, si fuera necesaria, de la planificación.

Seguimiento y Control (SyC): se incluyen las tareas asociadas a la verificación del correcto desarrollo del proyecto y de la comprobación continuada del cumplimiento de las expectativas marcadas sobre el alcance del proyecto.

- SyC.1: Recogida de la información relevante sobre el desarrollo del proyecto.
- SyC.2: Elaboración de una hoja de cálculo sobre Smartsheet en la que se lleve un control de los tiempos y fechas de desarrollo de las diferentes tareas planificadas, para facilitar la gestión del seguimiento y control.
- SyC.3: Confección de un documento de texto en el que se apunten diariamente los detalles de las tareas que se han desempeñado, complicaciones y nuevas tareas surgidas.
- SyC.4: Reuniones periódicas con la tutora del proyecto para compartir los aspectos en los que se ha avanzado y verificar el correcto progreso de la planificación.
- SyC.5: Contraste de la información de seguimiento con los planes, identificación de las desviaciones significativas y riesgos emergentes.
- SyC.6: Aseguramiento de las condiciones para el éxito del proyecto.
- SyC.7: Elaboración del contenido asociado al seguimiento y control del proyecto que debe ser incluido en el informe del TFG en su apartado asociado correspondiente.

Fase de Adquisición de Conocimiento

Elección de Herramientas (EHH): este paquete de trabajo incluye las tareas relacionadas con la búsqueda de información sobre diferentes herramientas que permitan el desarrollo del proyecto.

- EHH.1: Definir qué requisitos deben cumplir las herramientas con las que se va a trabajar e investigar cuáles pueden cumplirlas.
- EHH.2: Pruebas con las herramientas para ver qué tipo de desarrollo se podría realizar.

Adquisición de Competencias (ACC): en este paquete de trabajo se definen las tareas cuyo objetivo es la obtención de conocimiento acerca de las herramientas que van a ser utilizadas en el desarrollo del proyecto

- ACC.1: Conocer los sistemas de gestión de bases de datos MongoDB, Apache Cassandra y Neo4j, su estructura y modo de almacenamiento de datos, así como la sintaxis de consultas empleada por cada uno de ellos.

- ACC.2 : Conocer Robo 3T, DataStax DevCenter y Neo4j Desktop como clientes para los SGBD seleccionados, conociendo los aspectos de configuración y conexión con la parte servidora.
- ACC.3 : Conocer Draw.io y Smartsheet como herramientas de diseño gráfico tanto para la gestión del proyecto como para el diseño conceptual de los diagramas asociados al proyecto.
- ACC.4 : Conocer las librerías y frameworks que se van a utilizar para el desarrollo de la aplicación web tanto de la parte cliente como servidora. Estas son Vue.js, MDBVue, Node.js y Express.
- ACC.5 : Conocer Firebase como plataforma de alojamiento de la aplicación web, como origen de la base de datos de la aplicación web, como gestor de la autenticación de los usuarios y como delegado de la seguridad de la aplicación web.

Fase del Entorno de Virtualización

Virtualización(V): se engloban las tareas asociadas a la configuración y preparación de una máquina virtual con el software *NoSQL* necesario instalado para su posterior utilización en modo localhost.

- V.1: Descarga, instalación y configuración del software *NoSQL* asociado a las tecnologías seleccionadas, tanto la parte servidora como los clientes que se van a utilizar..
- V.2: Generación de una base de datos en cada uno de los SGBD seleccionados.

Fase de Aplicación Web

Diseño Conceptual (DC): en este paquete de trabajo se encuentran las tareas relacionadas con el análisis de la funcionalidad de la aplicación web, así como el posterior diseño y representación gráfica de los casos de uso a implementar y el esquema de base de datos a utilizar.

- DC.1: Descripción general de las funcionalidades que van a estar disponibles al usuario en la aplicación web a desarrollar.

- DC.2: Identificación y representación de los diferentes casos de uso asociados a las funcionalidades previamente planteadas a través de diagramas de casos de uso y diagramas de secuencia. Valoración de la usabilidad que proporciona al usuario cada caso de uso.
- DC.3: Identificación del contenido que se debe almacenar en la base de datos de la aplicación web. Análisis de requisitos y posterior diseño y representación gráfica del esquema de base de datos mediante los diagramas correspondientes.

Diseño Gráfico (DG): este paquete de trabajo engloba las tareas de diseño de la interfaz gráfica de la aplicación, desde los primeros bocetos hasta la implementación de los últimos detalles visuales con herramientas software de diseño de interfaces gráficas y las continuas mejoras.

- DG.1: Representación aproximada manual de las interfaces que va a disponer la aplicación web.
- DG.2: Representación concreta con software de diseño gráfico de las interfaces que va a disponer la aplicación web, digitalizando los bocetos a mano alzada previamente realizados.
- DG.3: Análisis de las características tanto visuales como de usabilidad que proporcionan las interfaces diseñadas, y sus consecuentes cambios si fuera necesario.

Preparación del Entorno (PE): en este paquete de trabajo se incluyen las tareas de configuración de las herramientas previas al inicio del desarrollo.

- PE.1: Instalación y configuración de Git y Github para crear un sistema de control de versiones que gestione el código de la aplicación web. Posterior sincronización con las herramientas de desarrollo.
- PE.2: Instalación y configuración de las Node.js, Express.js, Vue.js, MDBVue y Visual Studio Code como herramientas de desarrollo..
- PE.3: Añadir las funcionalidades de Firebase al código en desarrollo, desde las opciones de almacenamiento hasta la gestión de seguridad y usuarios.

Desarrollo (D): este paquete de trabajo hace referencia a las tareas de implementación de código sobre las herramientas de desarrollo previamente instaladas.

- D.1: Generación de las interfaces gráficas del lado cliente previamente diseñadas a través de las herramientas de desarrollo Vue.js y MDBVue, teniendo en cuenta los diagramas de casos de usos diseñados.
- D.2: Desarrollo de la parte servidora de la aplicación web con las herramientas de Node.js y Express, teniendo en cuenta los diagramas de secuencia asociados a las funcionalidades a implementar.
- D.3: Configurar el código en desarrollo para poder trabajar conjuntamente con Firebase.
- D.4: Enlazar el servicio de autenticación de usuarios de Firebase con la aplicación desarrollada, consiguiendo una gestión individual de la aplicación por parte de cada usuario.
- D.5: Enlazar el servicio de base de datos de Firebase con la aplicación desarrollada, configurando la parte común de la base de datos como el espacio asociado a cada usuario.
- D.6: Enlazar el servicio de almacenamiento de Firebase con la aplicación desarrollada, concretamente la parte del almacenamiento.
- D.7: Configurar el servicio de hosting de Surge y mantener una continua supervisión del alojamiento de la aplicación web hasta el final del proyecto.
- D.8: Continua sincronización de todos los servicios de Firebase con la aplicación web en desarrollo.

Fase de Pruebas

Pruebas del Desarrollo (PD): tareas relacionadas con el aseguramiento de la correcta implementación de las funcionalidades planteadas sobre la aplicación web en relación al código generado y las herramientas de desarrollo utilizadas.

- PD.1: Verificación de la correcta implementación de las interfaces gráficas, las características de usabilidad y la correlación entre los casos de uso y la experiencia de usuario.
- PD.2: Verificación de la conexión de la parte servidora con la base de datos y espacio de almacenamiento de Firebase.

- PD.3: Verificación de la conexión de la parte servidora con la gestión de la autenticación de usuarios de Firebase.
- PD.4: Verificación del correcto alojamiento de la aplicación web en Firebase.

Pruebas de Gestión (PG): en este paquete de trabajo se encuentran las tareas relacionadas con la comprobación del correcto funcionamiento de las herramientas de gestión utilizadas en el proyecto.

- PG.1: Verificación del funcionamiento de Git y Github como sistemas de control de versiones y copias de seguridad. Comprobar sincronización entre el código en local y el remoto.
- PG.2: Verificación de la sincronización de las copias del código y de la memoria tanto de manera local como en Google Drive y Overleaf de manera online.

Pruebas de Datos (PDD): incluye las tareas asociadas a la validación de los datos que van a ser utilizados en la evaluación del rendimiento de las diferentes bases de datos.

- PDD.1: Verificación de que los datos han sido transformados correctamente según el esquema de cada base de datos diseñado.
- PDD.2: Verificación de carga correcta de los datos con los que se va a trabajar en cada una de las tres bases de datos.

Fase de Comparativa de Rendimiento de las Bases de Datos

Adquisición de Datos (ADD): este paquete de trabajo cubre las tareas asociadas a la obtención de los conjuntos de datos con los que se van a realizar las pruebas de rendimiento entre los diferentes sistemas.

- ADD.1: Búsqueda de un formato de datos compatible a MongoDB, Apache Cassandra y Neo4j.
- ADD.2: Búsqueda de un conjunto de datos en forma de serie temporal y otro con estructura jerárquica. Los datos deben estar en un formato soportado por las tres bases de datos, de manera que no sea necesaria la conversión de un formato a otro y todas las bases de datos puedan trabajar con la misma fuente de datos.

- ADD.3: Carga de los datos en cada una de las tres bases de datos.

Diseño Base de Datos (DBD): este paquete de trabajo incluye la adaptación de los datos obtenidos a las diferentes bases de datos.

- DBD.1: Análisis de requisitos de cada conjunto de datos y su posterior diseño del esquema de base de datos para MongoDB, Apache Cassandra y Neo4j.
- DBD.2: Transformación de los datos al esquema de base de datos previamente diseñado para MongoDB, Apache Cassandra y Neo4j.

Evaluación del Rendimiento (ER): en este paquete de trabajo se desempeñan las tareas relacionadas con la valoración directa del rendimiento de cada base de datos.

- ER.1: Planteamiento general de sentencias de diferente índole y objetivo que vayan a ser ejecutadas sobre cada base de datos, y posterior adaptación de las sentencias seleccionadas a la sintaxis de consultas específica de cada SGBD.
- ER.2: Preparación y configuración de las herramientas para la posterior ejecución y análisis de las sentencias.
- ER.3: Ejecución de cada una de las sentencias sobre las tres bases de datos, analizando el rendimiento para cada una de ellas y posterior identificación de las conclusiones.

Fase de Documentación

Generación del Informe (INF): este paquete de trabajo incluye las tareas relacionadas con la elaboración del informe final del TFG.

- INF.1: Preparar el entorno de desarrollo del documento con LaTeX a través de la herramienta Overleaf. Adecuar una plantilla apropiada para el desarrollo de la memoria.
- INF.2: Generar documentos paralelamente con contenido asociado a la memoria en formato doc para posteriormente incluirlo en la memoria final.
- INF.3: Desarrollar la memoria del TFG en el entorno preparado.

Preparación Defensa TFG (PDT): se encuentran las tareas asociadas a la preparación de la defensa final de trabajo de fin de grado.

- PDT.1: Valorar los aspectos del trabajo realizado que deben ser mostrados en la presentación y evaluar la extensión que debe ocupar en la exposición.
- PDT.2: Generar un documento para la presentación a modo de soporte para la defensa del TFG incluyendo los aspectos que se han considerado previamente.
- PDT.3: Preparar la defensa del TFG planteando un orden en el contenido que va a ser expuesto, en concordancia con la presentación.
- PDT.4: Replantear tanto el documento de la presentación como el contenido de la ponencia, realizando mejoras si se ve necesario.

3.3.2.1. Estructura de descomposición del trabajo

La Estructura de Descomposición de Trabajo (E.D.T.) es un recurso gráfico indispensable en la gestión de un proyecto que ayuda a identificar visualmente de manera rápida la organización general de la planificación de un proyecto. Esta representación gráfica permite subdividir los objetivos del proyecto en tareas más pequeñas que faciliten la gestión del desarrollo del proyecto.

Una vez están definidas las fases del proyecto y los paquetes de trabajo, se define el E.D.T. estableciendo dos niveles en la estructura. El primer nivel se refiere a la fase del proyecto, mientras que el segundo nivel agrupa los paquetes de trabajo asociados a cada una de las fases (Fig. 3.4).

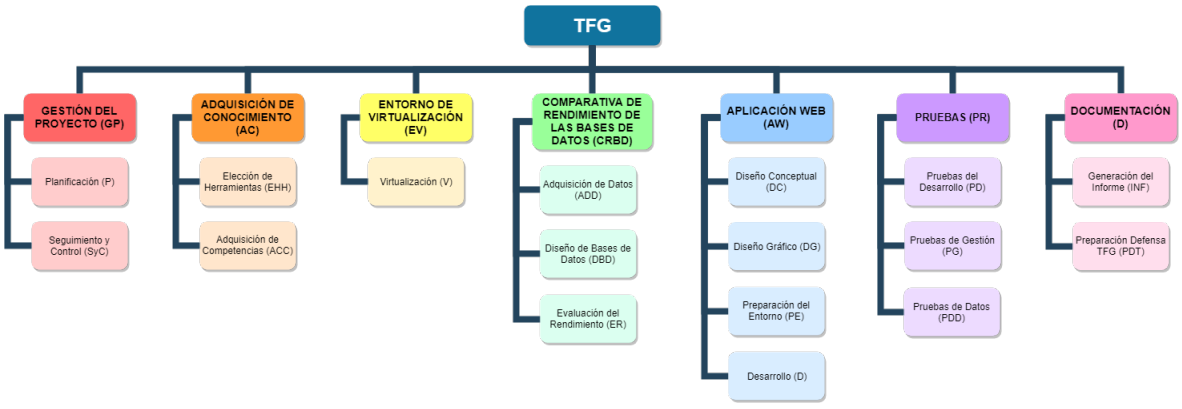


Figura 3.4: Estructura de descomposición del trabajo del proyecto

3.4. Metodología

A la hora de organizar el desarrollo del proyecto en base a una serie de fases, se ha estimado oportuno llevar una metodología tradicional con un ciclo de vida incremental (Fig. 3.5), es decir, se irá avanzando sobre las fases de una metodología tradicional para cada una de las funcionalidades a implementar.

De este modo, las fases de que costará cada uno de los ciclos serán Análisis, Diseño, Programación y Pruebas, que se desarrollarán con motivo de finalizar cada una de las funcionalidades, dando como resultado una versión mejorada y avanzada con una serie de funcionalidades completamente implementadas.

Este tipo de metodología permite gestionar los objetivos de cada una de las funcionalidades que se van a implementar de manera concreta y con toda la atención focalizada en el mismo punto del ciclo de vida del proyecto.

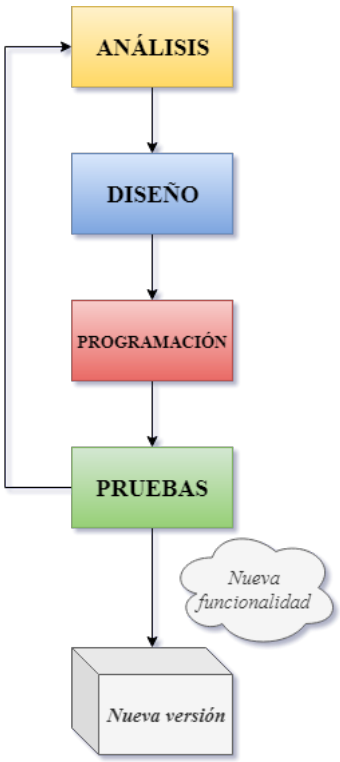


Figura 3.5: Ciclo de vida del proyecto

3.5. Planificación

3.5.1. Gestión del tiempo

Con las fases, paquetes de trabajo y tareas del proyecto identificadas, se procede a realizar una estimación inicial de la dedicación en horas a cada una de las tareas, así como consecuentemente a cada paquete de trabajo y por ende a cada fase. Todas las fases no requieren la misma dedicación y algunas difieren respecto a otras notablemente debido a que la caracterización de una fase es el contenido de las tareas que incluye y no la duración de éstas.

En la tabla 3.1 se indican las horas estimadas a cada tarea del proyecto.

Tabla 3.1: Estimación de la dedicación en horas para las tareas

Tareas	Estimación en horas
Trabajo de Fin de grado	**339**
Gestión del proyecto (GP)	**43**
Planificación (P)	16
P.1: Decisiones iniciales	2
P.2: Planificación inicial	12
P.3: Actualización de la planificación	2
Seguimiento y Control (SyC)	27
SyC.1: Recogida de información	3
SyC.2: Control tiempos Smartsheet	2
SyC.3: Diario de tareas desarrolladas	3
SyC.4: Reuniones con la tutora	4
SyC.5: Desviaciones y riesgos	2
SyC.6: Condiciones para el éxito del proyecto	5
SyC.7: Desarrollo contenido seguimiento y control	8
Adquisición de Conocimiento (AC)	**32**
Elección de Herramientas (EHH)	8
EHH.1: Investigar herramientas	4
EHH.2: Realizar pruebas con las herramientas	4
Adquisición de Competencias (ACC)	24
ACC.1: MongoDB, Apache Cassandra y Neo4j	10
ACC.2: Robo 3T, DataStax DevCenter y Neo4j Desktop	3
ACC.3: Draw.io, Smartsheet y Qlik Sense Desktop	2
ACC.4: Vue.js, MDBVue, Node.js y Express	5

Continuación de la tabla 3.1

Tareas	Estimación en horas
ACC.5: Firebase	4
Entorno de Virtualización (EV)	**8**
Virtualización (V)	8
V.1: Instalación y preparación software *NoSQL*	5
V.2: Carga de datos en bases de datos	3
Comparativa de Rendimiento de Bases de Datos (CRBD)	**48**
Adquisición de Datos (AD)	8
ADD.1: Búsqueda de formato compatible	1
ADD.2: Búsqueda de datos	4
ADD.3: Carga de datos en las bases de datos	3
Diseño de Base de datos (DBD)	18
DBD.1: Análisis de requisitos SGBDs	12
DBD.2: Transformación de datos	6
Evaluación del Rendimiento (ER)	22
ER.1: Sentencias a ejecutar y adaptación a SGBD	7
ER.2: Preparación herramientas análisis	3
ER.3: Ejecución de las sentencias y conclusiones	12
Aplicación Web (AW)	**149**
Diseño Conceptual (DC)	33
DC.1: Descripción funcionalidades aplicación	4
DC.2: Identificación y representación de los casos de uso	23
DC.3: Diseño modelo base de datos	6
Diseño Gráfico (DG)	20
DG.1: Representación mano alzada interfaces	7
DG.2: Representación concreta interfaces	10
DG.3: Análisis y mejora interfaces	3
Preparación del Entorno (PE)	10
PE.1: Git y Github y sincronización	4
PE.2: Node.js, Express.js, Vue.js, MDBVue y Visual Studio Code	3
PE.3: Añadir servicios de Firebase	3
Desarrollo (D)	86
D.1: Desarrollo front-end	30
D.2: Desarrollo back-end	30
D.3: Sincronización del código fuente con Firebase	2
D.4: Servicio de autenticación de Firebase y aplicación	4
D.5: Servicio de base de datos de Firebase y aplicación	5
D.6: Servicio de almacenamiento de Firebase y aplicación	4

Continuación de la tabla 3.1

Tareas	Estimación en horas
D.7: Servicio de hosting de Surge y aplicación	5
D.8: Sincronización de todos los servicios de Firebase	6
Pruebas (PR)	**18**
Pruebas del Desarrollo (PD)	9
PD.1: Verificación de las interfaces gráficas	2
PD.2: Verificación conexión servidor y base de datos	2
PD.3: Verificación conexión servidor y autenticación	2
PD.4: Verificación del alojamiento en Firebase	3
Pruebas de Gestión (PG)	4
PG.1: Verificación Git y Github	2
PG.2: Verificación sincronización código	2
Pruebas de Datos (PDD)	5
PDD.1: Verificación transformación de datos	3
PDD.2: Verificación carga correcta de datos	2
Documentación (D)	**41**
Generación del Informe (INF)	28
INF.1: Preparar entorno de desarrollo	2
INF.2: Generar documentos	6
INF.3: Desarrollar memoria TFG	20
Preparación Defensa TFG (PDT)	13
PDT.1: Elegir aspectos de la presentación	4
PDT.2: Generar documento de presentación	3
PDT.3: Preparar defensa del TFG	6

La distribución del total de horas en el desarrollo del proyecto no es equitativa respecto a cada fase del proyecto, haciéndose notable la diferencia de horas estimadas entre diferentes fases. No todas las fases requieren la misma implicación.

3.5.2. Dependencias

Para planificar los periodos de desarrollo de las diferentes tareas es conveniente identificar las dependencias existentes entre las diferentes tareas, con el fin de estimar las fechas de inicio y fin en función de dichas dependencias.

Respecto a la fase de *Gestión del Proyecto*, la tarea identificada como punto de partida es

P.1, por lo que el resto de tareas del proyecto dependen directa o indirectamente de esta tarea.

En los paquetes de trabajo de *Virtualización*, *Diseño Conceptual*, *Diseño Gráfico*, *Pruebas de gestión* y *Evaluación del Rendimiento* las tareas están secuenciadas en orden según sus dependencias internas, es decir, el número que secuencia la tarea del paquete de trabajo indica el orden el que tiene que ejecutarse dichas tareas dentro de ese paquete.

Las tareas incluidas en la fase de *Pruebas* deben ejecutarse después de las tareas que desarrollan el objeto que se está verificando en la fase de pruebas.

Las tareas incluidas en el paquete de trabajo de *Adquisición de Competencias* deben iniciarse antes que las tareas que requieren del uso de alguna de las competencias a estudiar.

Dentro de la fase de *Comparativa de Rendimiento de Bases de Datos*, en el paquete de trabajo de *Adquisición de Datos* la tarea de búsqueda de un formato compatible debe realizarse antes que las tareas asociadas a la búsqueda de conjuntos de datos. En el paquete de trabajo de *Diseño de Base de Datos* las tareas de análisis de requisitos de las bases de datos deben preceder a las tareas de la transformación de los datos. Además, la tarea de carga de datos incluida en el paquete de trabajo anterior debe realizarse tras la finalización de estas tareas. Las tareas del paquete de trabajo *Evaluación del Rendimiento* deben desarrollarse tras la obtención de los datos.

En la fase de *Aplicación Web*, los paquetes de trabajo son bastante dependientes en el tiempo. Para las tareas del paquete de trabajo *Desarrollo* es preciso que previamente se hayan desempeñado las tareas del resto de paquetes.

En la figura 3.6 se muestra la representación gráfica de dichas dependencias[2].

[2]Los puntos negros representan una bifurcación. Cada tarea se representa dentro de un rectángulo que diferencia entre inicio y fin de la tarea.

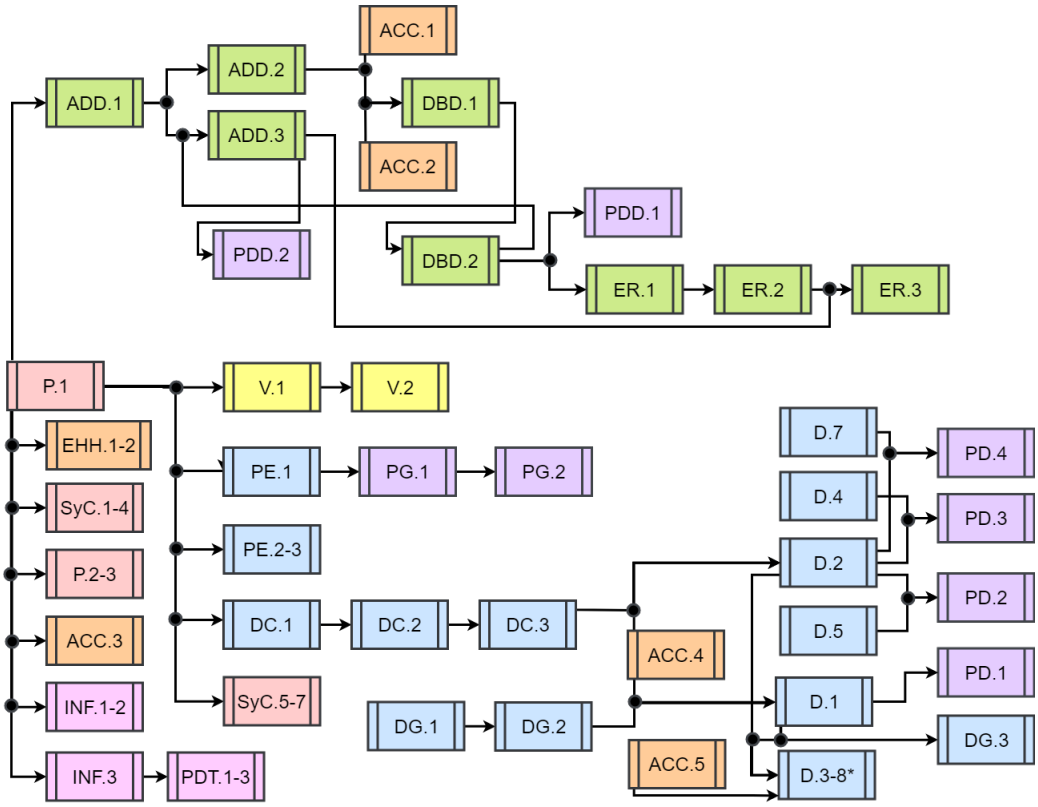


Figura 3.6: Diagrama de dependencias entre tareas

3.5.3. Calendario de paquetes de trabajo e hitos

Una vez están definidas las tareas, su duración y las dependencias entre ellas, se procede a desarrollar el diagrama de Gantt asociado al desarrollo del proyecto. Cada tarea tiene vinculada una fecha de inicio y una fecha de fin para su desarrollo. Además, sobre el mismo diagrama de Gantt se incluyen las fechas de los hitos más importantes que se van a dar en el proyecto.

En este diagrama están representadas todas las fases y todos los paquetes de trabajo identificados en el proyecto, determinando el momento de inicio y de fin de cada una de las fases y de los paquetes de trabajo, así como de las tareas que albergan. La repartición de dichas tareas se ha llevado a cabo teniendo en cuenta el 28 de enero de 2019 como día de inicio del proyecto, y el 1 de julio de 2019 (fecha aproximada) como día de cierre total del proyecto por ser la fecha de su defensa.

A parte de los dos hitos mencionados, destacan siete hitos más en la planificación del

proyecto. Se tratan de fechas remarcadas por ser días clave en el desarrollo del proyecto y en el cumplimiento de las tareas.

En la representación gráfica del diagrama de Gantt por meses (Fig. 3.7 y Fig. 3.8) se toma una perspectiva general de la duración y periodo de desarrollo de cada tarea.

	Nombre de la tarea	Inicio	Fin
1	Trabajo de Fin de Grado	28/01/19	01/07/19
2	*Hito.1: Inicio de proyecto*	*28/01/19*	*28/01/19*
3	*Hito.2: Matriculación TFG*	*14/06/19*	*14/06/19*
4	*Hito.3: Subir memoria a plataforma digital*	*23/06/19*	*23/06/19*
5	*Hito.4: Fin de proyecto*	*01/07/19*	*01/07/19*
6	Gestión del Proyecto (GP)	28/01/19	03/07/19
7	Planificación (P)	28/01/19	27/03/19
8	P.1: Decisiones iniciales	28/01/19	14/02/19
9	P.2: Planificación inicial	28/01/19	15/02/19
10	P.3: Actualización de la planificación	17/02/19	27/03/19
11	Seguimiento y Control (SyC)	28/01/19	03/07/19
12	SyC.1:Recogida de información	17/02/19	12/06/19
13	SyC.2: Control tiempos Smartsheet	17/02/19	03/07/19
14	SyC.3: Diario de tareas desarrolladas	17/02/19	11/06/19
15	SyC.4: Reuniones con la tutora	28/01/19	01/07/19
16	SyC.5: Desviaciones y riesgos	15/03/19	30/04/19
17	SyC.6: Condiciones para el éxito del proyecto	10/06/19	14/06/19
18	SyC.7: Desarrollo contenido seguimiento y control	13/06/19	17/06/19
19	Adquisición de Conocimiento (AC)	04/02/19	10/05/19
20	Elección de Herramientas (EHH)	04/02/19	18/02/19
21	EHH.1: Investigar herramientas	04/02/19	13/02/19
22	EHH.2: Realizar pruebas con las herramientas	14/02/19	18/02/19
23	Adquisición de Competencias (ACC)	04/02/19	10/05/19
24	ACC.1: MongoDB, Apache Cassandra y Neo4j	20/03/19	10/04/19
25	ACC.2: Robo 3T, DataStax DevCenter y Neo4j Desktop	20/03/19	01/04/19
26	ACC.3: Draw.io, Smartsheet y Qlik Sense Desktop	04/02/19	15/02/19
27	ACC.4: Vue.js, Vuetify, Node.js y Express	15/04/19	30/04/19
28	ACC.5: Firebase	15/04/19	10/05/19
29	Entorno de Virtualización (EV)	18/02/19	25/02/19
30	Virtualización (V)	18/02/19	25/02/19
31	V.1: Instalación y preparación software NoSQL	18/02/19	21/02/19
32	V.2: Carga de datos en bases de datos	22/02/19	25/02/19
33	*Hito.5: Finalización de entorno virtual NoSQL*	*25/02/19*	*25/02/19*
34	Comparativa de Rendimiento de Bases de Datos (CRBD)	25/02/19	17/04/19
35	Adquisición de Datos (AD)	25/02/19	25/03/19
36	ADD.1: Búsqueda de formato compatible	25/02/19	26/02/19
37	ADD.2: Búsqueda de datos	28/02/19	04/03/19
38	ADD.3: Carga de datos en las bases de datos	23/03/19	25/03/19
39	Diseño de Base de Datos (DBD)	06/03/19	21/03/19
40	DBD.1: Análisis de requisitos SGBDs	06/03/19	18/03/19
41	DBD.2: Transformación de datos	19/03/19	21/03/19
42	Evaluación del Rendimiento (ER)	03/04/19	16/04/19
43	ER.1: Sentencias a ejecutar y adaptación a SGBD	03/04/19	05/04/19
44	ER.2: Preparación herramientas análisis	08/04/19	10/04/19
45	ER.3: Ejecución de las sentencias y conclusiones	11/04/19	16/04/19
46	*Hito.6: Finalización de la comparativa de bases de datos*	*17/04/19*	*17/04/19*

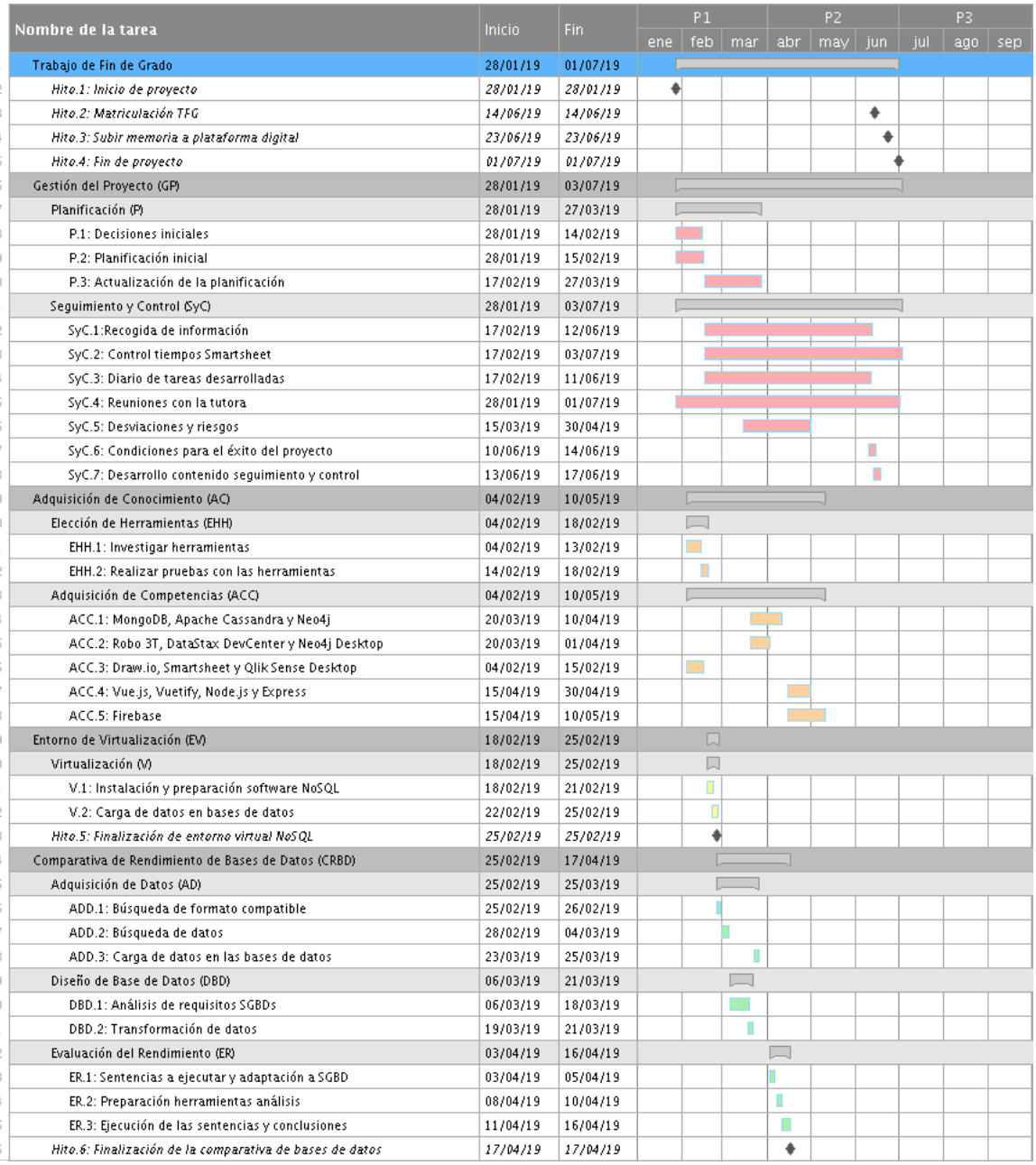
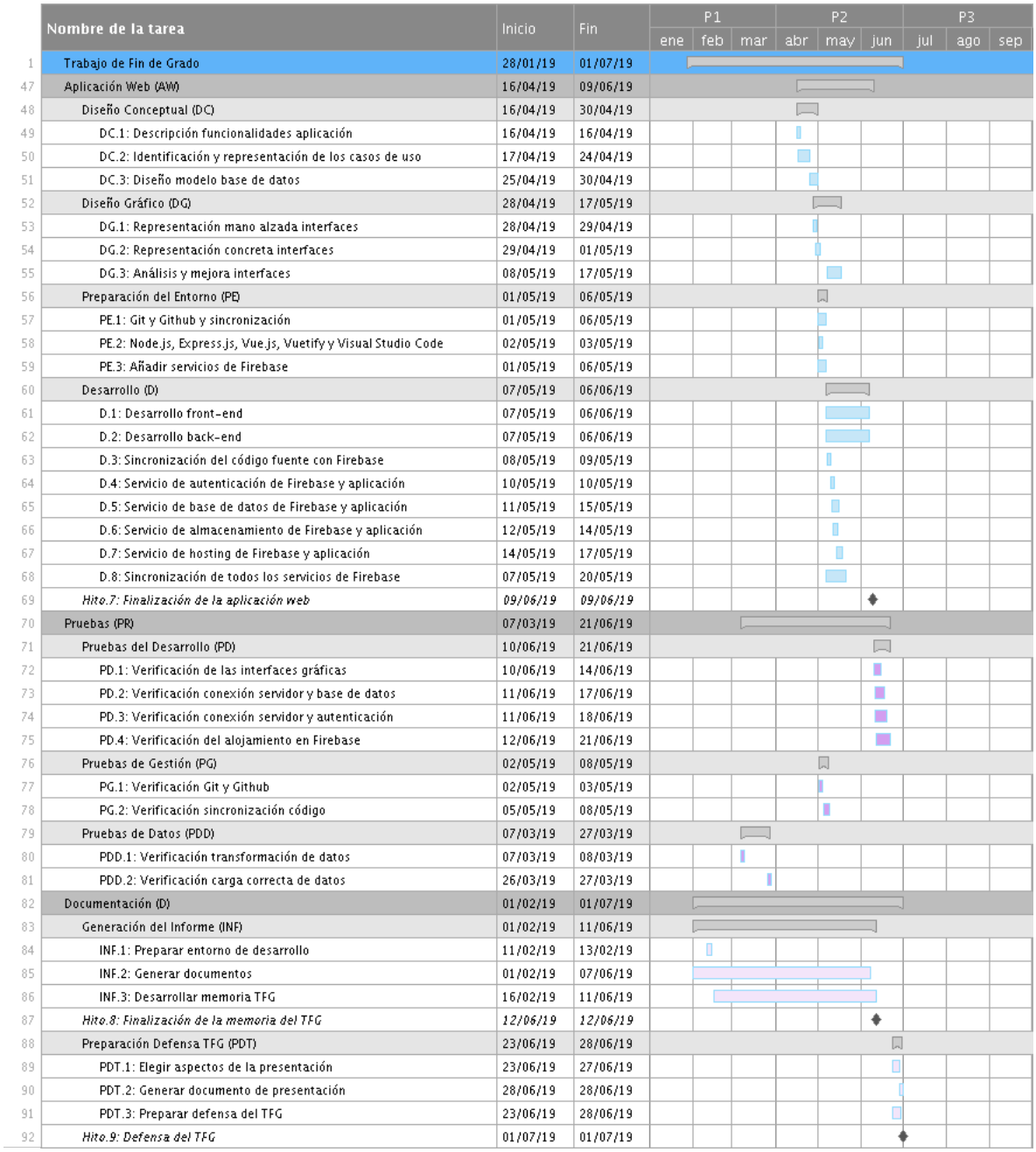


Figura 3.7: Diagrama de Gantt - I



	Nombre de la tarea	Inicio	Fin
1	Trabajo de Fin de Grado	28/01/19	01/07/19
47	Aplicación Web (AW)	16/04/19	09/06/19
48	Diseño Conceptual (DC)	16/04/19	30/04/19
49	DC.1: Descripción funcionalidades aplicación	16/04/19	16/04/19
50	DC.2: Identificación y representación de los casos de uso	17/04/19	24/04/19
51	DC.3: Diseño modelo base de datos	25/04/19	30/04/19
52	Diseño Gráfico (DG)	28/04/19	17/05/19
53	DG.1: Representación mano alzada interfaces	28/04/19	29/04/19
54	DG.2: Representación concreta interfaces	29/04/19	01/05/19
55	DG.3: Análisis y mejora interfaces	08/05/19	17/05/19
56	Preparación del Entorno (PE)	01/05/19	06/05/19
57	PE.1: Git y Github y sincronización	01/05/19	06/05/19
58	PE.2: Node.js, Express.js, Vue.js, Vuetify y Visual Studio Code	02/05/19	03/05/19
59	PE.3: Añadir servicios de Firebase	01/05/19	06/05/19
60	Desarrollo (D)	07/05/19	06/06/19
61	D.1: Desarrollo front-end	07/05/19	06/06/19
62	D.2: Desarrollo back-end	07/05/19	06/06/19
63	D.3: Sincronización del código fuente con Firebase	08/05/19	09/05/19
64	D.4: Servicio de autenticación de Firebase y aplicación	10/05/19	10/05/19
65	D.5: Servicio de base de datos de Firebase y aplicación	11/05/19	15/05/19
66	D.6: Servicio de almacenamiento de Firebase y aplicación	12/05/19	14/05/19
67	D.7: Servicio de hosting de Firebase y aplicación	14/05/19	17/05/19
68	D.8: Sincronización de todos los servicios de Firebase	07/05/19	20/05/19
69	*Hito.7: Finalización de la aplicación web*	*09/06/19*	*09/06/19*
70	Pruebas (PR)	07/03/19	21/06/19
71	Pruebas del Desarrollo (PD)	10/06/19	21/06/19
72	PD.1: Verificación de las interfaces gráficas	10/06/19	14/06/19
73	PD.2: Verificación conexión servidor y base de datos	11/06/19	17/06/19
74	PD.3: Verificación conexión servidor y autenticación	11/06/19	18/06/19
75	PD.4: Verificación del alojamiento en Firebase	12/06/19	21/06/19
76	Pruebas de Gestión (PG)	02/05/19	08/05/19
77	PG.1: Verificación Git y Github	02/05/19	03/05/19
78	PG.2: Verificación sincronización código	05/05/19	08/05/19
79	Pruebas de Datos (PDD)	07/03/19	27/03/19
80	PDD.1: Verificación transformación de datos	07/03/19	08/03/19
81	PDD.2: Verificación carga correcta de datos	26/03/19	27/03/19
82	Documentación (D)	01/02/19	01/07/19
83	Generación del Informe (INF)	01/02/19	11/06/19
84	INF.1: Preparar entorno de desarrollo	11/02/19	13/02/19
85	INF.2: Generar documentos	01/02/19	07/06/19
86	INF.3: Desarrollar memoria TFG	16/02/19	11/06/19
87	*Hito.8: Finalización de la memoria del TFG*	*12/06/19*	*12/06/19*
88	Preparación Defensa TFG (PDT)	23/06/19	28/06/19
89	PDT.1: Elegir aspectos de la presentación	23/06/19	27/06/19
90	PDT.2: Generar documento de presentación	28/06/19	28/06/19
91	PDT.3: Preparar defensa del TFG	23/06/19	28/06/19
92	*Hito.9: Defensa del TFG*	01/07/19	01/07/19

Figura 3.8: Diagrama de Gantt - II

La carga horaria se ha repartido en función a la dedicación personal que se puede dedicar a cada momento del ciclo de vida del proyecto. De este modo, queda reflejado (Fig. 3.9) que durante los meses de abril y mayo la dedicación a las tareas del proyecto es notablemente superior que en el resto de meses.

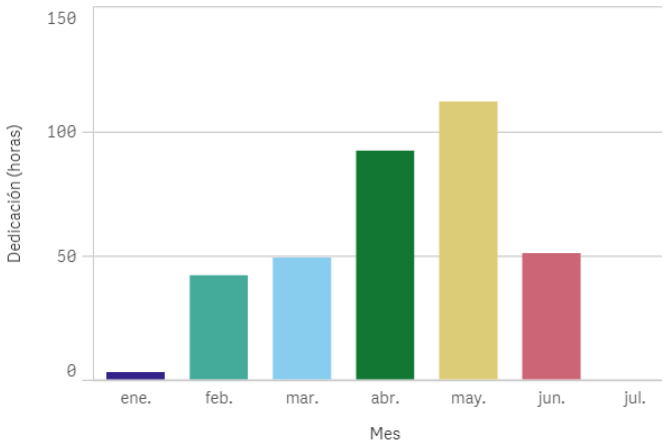


Figura 3.9: Carga en horas respecto a la dedicación estimada por mes en gráfico de barras

3.6. Gestión de riesgos

3.6.1. Problemas relacionados con la compaginación con otras asignaturas

La evolución del TFG se llevará a cabo de manera paralela al desarrollo de una asignatura del segundo cuatrimestre del grado en Ingeniería Informática. Esto pudiera conllevar algún tipo de desajuste en la planificación debido a contratiempos relacionados con la gestión de la asignatura.

- **Consecuencias**: menos tiempo del estimado disponible para realizar tareas asociadas al desarrollo del TFG.
- **Probabilidad**: alta.
- **Responsabilidad**: propia.
- **Impacto**: alto.
- **Plan de mitigación**: estudiar detalladamente la guía de la asignatura, observando fechas importantes (exámenes, presentaciones...), teniendo en cuenta la planificación de la asignatura para generar una planificación del proyecto que permita compaginar las actividades de la asignatura con las del proyecto.
- **Plan de contingencia**: replanificar las tareas afectadas tratando de que ésto conlleve el menor impacto posible sobre el desarrollo del proyecto.

3.6.2. Falta de recursos para una aplicación concreta sobre una herramienta

Aunque la elección de las herramientas que se van a utilizar para el desarrollo del proyecto se trata de una elección estudiada y analizada, es posible que mientras se esté utilizando una de las herramientas se compruebe que una de las opciones a implementar no está cubierta por las funcionalidades que ofrece la herramienta.

- **Consecuencias**: tiempo adicional en la búsqueda de otro método para implementar la funcionalidad que la herramienta no es capaz de suplir, o bien para la búsqueda de otra herramienta y posterior traslado del trabajo realizado en dicha herramienta sobre la nueva.
- **Probabilidad**: media.
- **Responsabilidad**: propia.
- **Impacto**: muy alto.
- **Plan de mitigación**: analizar con atención tanto las funcionalidades concretas que se deben poder implementar como las características propias de las herramientas, escogiendo aquellas que permitan contemplar todos los requisitos establecidos.
- **Plan de contingencia**: en función de la gravedad de la situación, se debe actuar bien replanteando la funcionalidad que no se puede implementar, o bien buscando una nueva herramienta que sea capaz de implementarlo.

3.6.3. Falta de disponibilidad del hardware utilizado

Si el hardware que se está empleando para el desarrollo del TFG sufre algún tipo problema y deja de funcionar, esto supondría un impacto en la planificación.

- **Consecuencias**: disminución en el tiempo destinado al desarrollo de las tareas del TFG.
- **Probabilidad**: baja.
- **Responsabilidad**: ajena.
- **Impacto**: medio.

- **Plan de mitigación**: evaluar y preparar el hardware que va a ser utilizado previamente a su utilización. Probar los componentes y valorar si sus capacidades son suficientes para el desarrollo del proyecto. Realizar un mantenimiento continuo del sistema.
- **Plan de contingencia**: evaluar la gravedad del estado del componente hardware afectado y proceder a repararlo o bien sustituirlo.

3.6.4. Falta de disponibilidad del software utilizado

La elección del software que se va a utilizar en el desarrollo del proyecto es previa a su utilización en éste. Pudiera ocurrir que la herramienta dejara de funcionar por actualizaciones ajenas o incompatibilidades con el sistema.

- **Consecuencias**: parón en el desarrollo del proyecto que conllevaría retrasos en la elaboración de las tareas que tengan relación con el software afectado.
- **Probabilidad**: baja.
- **Responsabilidad**: ajena.
- **Impacto**: alto.
- **Plan de mitigación**: evaluar para cada herramienta software los requisitos del sistema sobre el que puede ser instalado. Disponer de una copia del instalador de cada herramienta en la versión que sí es aceptada por el sistema. Desactivar las actualizaciones automáticas y, en caso de que las hubiere, no actualizar el software hasta comprobar la compatibilidad con el sistema utilizado.
- **Plan de contingencia**: búsqueda de una versión de la herramienta software compatible. Si no hay alternativas, búsqueda de otra herramienta que sea capaz de cubrir las funcionalidades de la anterior y cuya implantación en el proyecto suponga el menor impacto posible.

3.6.5. Planificación incorrecta

Aunque se haya trabajado para desarrollar una planificación acorde a los requisitos del proyecto, es posible la existencia de aspectos que no hayan sido los suficientemente evaluados a la hora de planificar.

- **Consecuencias**: tareas que no pueden ser cumplidas en el periodo de tiempo estimado, retraso en cadena de tareas en el proyecto, herramientas que no cumplen con las funcionalidades esperadas.
- **Probabilidad**: media.
- **Responsabilidad**: propia.
- **Impacto**: alto.
- **Plan de mitigación**: invertir el tiempo suficiente en la planificación con el fin de prevenir cualquier tipo de amenaza que pudiera tener un impacto en el desarrollo del proyecto. Identificar de manera precisa las tareas que se deben realizar, para así poder evaluar las dependencias entre ellas y las duraciones que cada una necesita.
- **Plan de contingencia**: replanificar las tareas que se ven afectadas por un mal planteamiento sobre un aspecto concreto de la planificación, intentando que el impacto sobre el proyecto sea el menor posible.

3.6.6. Motivos personales

Durante el desarrollo del proyecto se pueden dar situaciones de condición personal que afecten a las tareas y otros aspectos del proyecto y conlleven un impacto en su desarrollo.

- **Consecuencias**: tareas retrasadas respecto al periodo planificado.
- **Probabilidad**: baja.
- **Responsabilidad**: propia y/o ajena.
- **Impacto**: desde muy bajo hasta muy alto.
- **Plan de mitigación**: no se puede predecir los acontecimientos de índole personal que pudieran ocurrir y afectar al desarrollo del proyecto. Se debe intentar identificar acontecimientos conocidos y trasladarlos a la planificación, así como evaluar qué consecuencias pudieran tener estos acontecimientos que pudiesen afectar al desarrollo del proyecto.
- **Plan de contingencia**: replantear la planificación adaptándola a las consecuencias que ha provocado los acontecimientos personales sucedidos.

3.7. Tecnologías

Las herramientas que se van a utilizar durante la gestión, diseño y desarrollo del proyecto son las que se describen a continuación.

3.7.1. Gestión de la documentación y código

- **Git**[3] se va a utilizar como sistema de control de versiones que permita registrar los cambios que se dan en los ficheros del código, coordinando las modificaciones que se realicen.
- **Github**[4] junto con Git se utilizará para mantener una copia de seguridad del código en desarrollo almacenado en sus repositorios, a la vez que para llevar un control de versiones sobre los ficheros del código.
- **Google Drive**[5] se utilizará para almacenar las copias de seguridad del documento de la memoria y de los diferentes elementos de gestión que se van a ir generando, como diagramas, esquemas, tablas, etc.
- **LaTeX**[6] se utiliza como sistema de composición de textos para el desarrollo de este proyecto, empleando **Overleaf**[7] como editor de texto para LaTeX.

3.7.2. Diseño de la documentación y código

- **SmartSheet**[8] se va a utilizar en este proyecto para la implementación de los diagramas de Gantt, esquemas de dependencias entre tareas y otros aspectos relacionados con la gestión del proyecto.
- **Draw.io**[9] se utilizará para la generación de los diferentes diagramas que se van a incluir en la memoria del proyecto.

[3] Git: https://git-scm.com/
[4] Github: https://github.com/
[5] Google Drive: https://www.google.com/intl/es_ALL/drive/
[6] LaTeX: https://www.latex-project.org/
[7] Overleaf: https://www.overleaf.com/
[8] SmartSheet: https://es.smartsheet.com/
[9] Draw.io: https://www.draw.io/

- **Qlik Sense Desktop**[10] se va a utilizar en este proyecto para generar representaciones gráficas de los datos de rendimiento obtenidos de las diferentes bases de datos.

3.7.3. Herramientas para las bases de datos

- **MongoDB**[11] es un sistema gestor de bases de datos *NoSQL* orientado a documentos, que almacena los datos en estructuras en formato BSON (una representación binaria de JSON). Permite gestionar los datos con estructuras diferentes, ya que no es necesario que los datos de una colección sigan un esquema fijo. MongoDB es el SGBD orientado a documentos seleccionado para este proyecto ya que se trata del sistema *NoSQL* orientado a documentos que tiene una posición más alta en el ranking de SGBD de *DB-engines*[12] a fecha previa de la publicación de este proyecto.

- **Robo 3T**[13] es una interfaz gráfica de usuario que permite gestionar de manera visual e intuitiva las bases de datos de MongoDB, es decir, se trata de un cliente gráfico de MongoDB. Proporciona ayuda a la hora de gestionar la sintaxis de consultas de MongoDB, facilitando la escritura rápida de los comandos. También permite realizar diferentes tipos de operaciones de manera gráfica a través de los botones y menús de la interfaz. Se ha escogido Robo 3T (antiguo Robomongo) como cliente para gestionar MongoDB por ser el cliente de MongoDB más utilizado y de los que más comunidad online tiene generada.

- Apache Cassandra es un sistema gestor de bases de datos *NoSQL* basado en un modelo de almacenamiento clave-valor. Se ha escogido la **Distribución de Apache Cassandra de Datastax**[14], que parte de Apache Cassandra y tiene generada una comunidad y un soporte en torno a ella. Apache Cassandra es el SGBD de tipo clave-valor escogido para este proyecto por tratarse del sistema *NoSQL* de tipo clave-valor que tiene una posición más alta en el ranking de SGBD de *DB-engines*[12] a fecha previa de la publicación de este proyecto. El sistema *Redis* (clave-valor) ocupa una posición superior pero es una tecnología que no permite realizar consultas, por lo que queda descartada.

[10] Qlik Sense Desktop: https://www.qlik.com/es-es/products/qlik-sense/desktop
[11] MongoDB: https://www.mongodb.com/
[12] DB-engines: https://db-engines.com/en/ranking
[13] Robo 3T: https://robomongo.org/
[14] Distribución de Apache Cassandra de Datastax: https://academy.datastax.com/planet-cassandra/cassandra

- **Datastax DevCenter**[15] es una interfaz gráfica de usuario para Apache Cassandra que se instala por defecto con la instalación de la Distribución Datastax de Apache Cassandra. Se trata de un cliente para Apache Cassandra preparado específicamente para trabajar con Apache Cassandra como base de datos. Se ha elegido Datastax DevCenter como cliente para Apache Cassandra por la ayuda que ofrece como soporte Datastax, y por ser un cliente generalizado en el entorno de Apache Cassandra.
- **Neo4j**[16] es un sistema gestor de bases de datos orientado a grafos, que almacena los datos en estructuras de grafos en lugar de en tablas u otro tipo de registros. Este tipo de tecnología se focaliza en encontrar las relaciones entre los datos para extraer así su verdadero valor. Neo4j es el SGBD orientado a grafos seleccionado para utilizar en el proyecto ya que es el sistema *NoSQL* orientado a grafos que tiene una posición más alta en el ranking de SGBD de *DB-engines*[12] a fecha previa de la publicación de este proyecto.
- **Neo4j Desktop**[17] es una interfaz gráfica de usuario para Neo4j que puede ser instalada por defecto con la instalación del propio Neo4j. Permite gestionar la base de datos de manera visual, con la generación de grafos y realización de operaciones sobre la base de datos de manera gráfica e intuitiva.

3.7.4. Herramientas para la implementación del proyecto

- Con **Oracle VM VirtualBox**[18] se va a crear la máquina virtual para este proyecto.
- **Visual Studio Code**[19] es un editor de código fuente preparado para trabajar con Git. Está preparado para trabajar con varias de las tecnologías con las que se va a trabajar, como Node.js, HTML, JavaScript o CSS.
- **Firebase**[20] y **Google Cloud Platform**[21] son dos plataformas de Google que facilitan la creación de aplicaciones web y móviles (principalmente Firebase) y de proyectos más grandes (principalmente Google Cloud Platform). Para el desarrollo

[15] Datastax DevCenter: https://docs.datastax.com/en/developer/devcenter/doc/devcenter/dcAbout.html
[16] Neo4j: https://neo4j.com/
[17] Neo4j Desktop: https://neo4j.com/developer/neo4j-desktop/
[18] Oracle VM VirtualBox: https://www.virtualbox.org/
[19] Visual Studio Code: https://code.visualstudio.com/
[20] Firebase: https://firebase.google.com/?hl=es-419
[21] Google Cloud Platform: https://cloud.google.com/

de este proyecto se van a utilizar los servicios de almacenamiento, base de datos, autenticación y gestión de usuarios (Firebase), y de creación de máquina virtual (GCP).

- **Surge**[22] es una herramienta para la publicación de webs estáticas de manera gratuita. La aplicación web a desarrollar utilizará este servicio para alojar la parte del cliente, permitiendo un acceso *online* a su contenido.
- **Vue.js**[23] es un framework open-source de JavaScript para la generación de interfaces de usuario y aplicaciones de una sola página (Single-Page Application, SPA). Se trata de un framework progresivo, que permite generar aplicaciones sencillas y poder avanzar en la complejidad de éstas progresivamente. Vue.js se utilizará para generar todas las interfaces gráficas de la aplicación web.
- **MDBVue**[24] es una librería basada en Google ***Material Design***[25] específicamente diseñada para Vue.js en conjunto con Bootstrap, y **BootstrapVue**[26] se trata de una librería que facilita la gestión de Bootstrap con el framework Vue.js.
- **Node.js**[27] es una librería y entorno en tiempo de ejecución multiplataforma, que permite ejecutar el código JavaScript en el servidor y que está basado en eventos. Ofrece la posibilidad de crear herramientas del lado del servidor y aplicaciones con JavaScript. Se va a utilizar Node.js para implementar el código de la parte *back-end* utilizando JavaScript, que conjuntamente trabajará con otras librerías y frameworks para acabar desarrollando una aplicación web basada en JavaScript, HTML y CSS.
- **Express.js**[28] es una infraestructura de aplicaciones web con Node.js que proporciona un conjunto de características para las aplicaciones tanto web como móviles. Facilita el rápido desarrollo de las aplicaciones web basadas en Node.js. Se emplearán las funcionalidades que proporciona Express.js para el desarrollo de la aplicación web en el proyecto.

[22] Surge: https://surge.sh/
[23] Vue.js: https://vuejs.org/
[24] MDBVue: https://mdbootstrap.com/docs/vue/
[25] Material Design: https://material.io/
[26] BootstrapVue: https://bootstrap-vue.js.org/
[27] Node.js: https://nodejs.org/es/
[28] Express.js: https://expressjs.com/es/

4. CAPÍTULO

Rendimiento de las bases de datos *NoSQL*

En este capítulo se busca profundizar más sobre estas tecnologías y conocer la eficiencia que demuestra cada una y el tipo de contextos en las que son utilizadas. Se plantean dos escenarios diferentes propios del contexto *NoSQL*, uno de ellos orientado a la gestión de los datos en forma de serie temporal, y otro relacionado con la gestión de datos propios de una página web informativa. Estos escenarios son representados en tres bases de datos *NoSQL* con diferente tipo de motor de almacenamiento, como son el de MongoDB, Apache Cassandra y Neo4j, y cada uno de los escenarios es evaluado para cada uno de los tres contextos de cara analizar la eficiencia de cada SGBD con cada tipo de datos.

Durante este proceso y para cada escenario, primero se preparan los datos para obtener un origen de datos común a los diferentes SGBD, luego se diseña un esquema de datos para cada base de datos acorde a unos requisitos previamente analizados y, finalmente, los datos del origen de datos se transforman según el esquema diseñado y se cargan en la base de datos. Cuando la base de datos está preparada, se procede a realizar un análisis de rendimiento evaluando aspectos como el espacio de almacenamiento y el tiempo de ejecución de diferentes consultas. Por último, cuando se ha realizado este análisis sobre los tres contextos para cada escenario, se realiza una comparativa en función de los resultados obtenidos, identificando qué SGBD es más apropiado para cada uno de los dos escenarios planteados.

4.1. Preparación de los datos

4.1.1. Obtención y acondicionamiento de los datos

Para realizar el análisis de rendimiento de las bases de datos *NoSQL* seleccionadas (MongoDB, Apache Cassandra y Neo4j) se han identificado dos escenarios diferentes. Uno de ellos se corresponde a un conjunto de datos en forma de serie temporal, mientras que el otro es un conjunto de datos propio de una base de datos de una página web de carácter informativo.

4.1.1.1. Datos en forma de serie temporal

Los datos en forma de serie temporal utilizados son datos reales obtenidos del funcionamiento de máquinas industriales. La empresa *Urola Solutions* se encarga de la fabricación de máquinas industriales, y la empresa *Savvy Data Systems* digitaliza el funcionamiento de dichas máquinas industriales tranformando los datos captados en datos útiles y analizables. Estos datos son proporcionados para el proyecto por el grupo de investigación BDI[1] de la Facultad de Informática de Donostia-San Sebastián, quien posee un convenio de confidencialidad con ambas empresas que permite trabajar con los datos proporcionados en el ámbito académico y de la investigación.

El conjunto de datos se proporciona en ficheros con formato *csv* que representan valores relativos al consumo del motor, presión de fundido, revoluciones por minuto y temperatura en un periodo de tiempo desde el 10/12/2018 00:00:00 hasta el 16/12/2018 23:59:59. Los valores son captados por segundo y cada uno de estos instantes es almacenado en dos campos, donde uno de ellos es el *timestamp*[2] y el otro el valor obtenido en dicho *timestamp*. Cada fichero *csv* contiene únicamente información de un tipo de actividad recogida.

4.1.1.2. Datos con estructura propia de página web

Con intención de evaluar el rendimiento de las bases de datos con un conjunto de datos de diferente formato, se ha realizado una búsqueda en la web de diferentes fuentes de

[1] Grupo de investigación BDI: http://bdi.si.ehu.es/bdi/
[2] Timestamp: marca temporal

datos *open data* donde se ha dado con un *dataset* sobre películas, directores, actores, puntuaciones, etc.

Este conjunto de datos ha sido generado por un grupo de la Universidad Autónoma de Madrid y se trata de una extensión del *dataset MovieLens10M* publicado por el grupo de investigación *GroupLeans*[3]. El conjunto de datos está constituido por 11 ficheros en formato *dat* que han sido traducidos a ficheros en formato *csv*. Cada uno contiene valores sobre películas, directores, países, géneros de películas, etiquetas de películas o valoraciones. Los datos están relacionados entre sí a través de identificadores de objeto.

Los ficheros contienen datos de 2113 usuarios, 10197 películas, 20 géneros de película, 4060 directores y 95321 actores. El fichero de las películas está formado por una columna de id y varias columnas más con información relativa a la película, y de este fichero se obtiene el título, año e imagen de la película. Los ficheros de actores, directores, géneros, localizaciones y puntuaciones albergan una fila por cada valor que representan, que se asocia a la película a través de el id de la película. De estos ficheros se obtienen las listas de actores, directores, géneros y localizaciones, así como el valor de la puntuación de la película. Las etiquetas están representadas en dos ficheros, uno de ellos con dos columnas referentes al id de la película y al id de la etiqueta, y otro fichero con dos columnas referentes al id de la etiqueta y al valor de la etiqueta. Ambos ficheros se relacionan entre sí por el campo del id de la etiqueta. Se muestra en la figura 4.1 una representación esquematizada de la relación existente entre los ficheros, teniendo en cuenta que la tabla azul es representativa para los ficheros de actores, directores, géneros, localizaciones y puntuaciones en cuanto al modo en el que se relacionan con el fichero que almacena las películas.

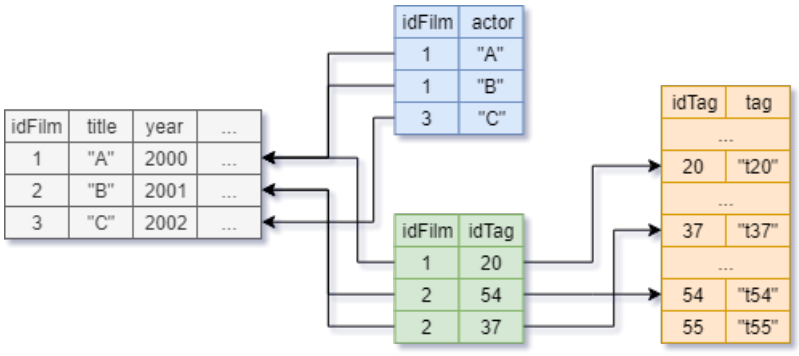


Figura 4.1: Representación esquematizada de la relación entre los ficheros del conjunto de datos sobre películas

[3]Grupo de investigación GroupLeans: http://www.grouplens.org

A través de un *script* se recogen todos los ficheros en formato *csv* con los datos de las películas y se genera un único fichero en formato *csv* que se va a tratar del origen de datos común para todos los SGBD. Se utiliza el símbolo "," como delimitador de los campos y el símbolo "|" como delimitador de las listas dentro de los campos. Para poder trabajar con una base de datos lo suficientemente grande que permita realizar un buen análisis, se han añadido datos sintéticos a los previamente mencionados llegando a un total de 249107 películas.

La cabecera del fichero generado se compone de los siguientes campos:

- **title**: Título de la película.
- **year**: Año de publicación de la película.
- **image**: Imagen de la portada de la película.
- **country**: País oficial de la película.
- **actors**: Lista de actores de la película.
- **directors**: Lista de directores de la película.
- **genres**: Lista de géneros asociados a al película.
- **tags**: Lista de etiquetas asociadas a la película.
- **locations**: Lista de localizaciones de grabación de la película.
- **score**: Puntuación media de la película.
- **num**: Número de puntuaciones disponibles de la película.
- **total**: Suma de las puntuaciones de la película.

GENERACIÓN DEL FICHERO *CSV*

A través del lenguaje de programación Java, utilizando Eclipse se confecciona un programa estructurado en dos partes, que ejecutadas secuencialmente han dado lugar al fichero en formato *csv* que va a ser utilizado como origen de datos para los tres SGBD.

Por una lado, la primera parte se encarga de leer y añadir a un único fichero en formato *csv* los datos de los diferentes ficheros que contienen los datos sobre películas, actores, directores, puntuaciones, localizaciones, géneros y etiquetas obtenidos del *dataset MovieLens10M*. El programa generado se encarga de leer cada fila del fichero de películas (cada película) y obtener de esa línea el campo del id, título, año, imagen y país. Utiliza

el id de la película para obtener del resto de ficheros los actores, directores, géneros, localizaciones, puntuación y etiquetas asociadas a esa película, y convierte cada una de estas listas de valores en un campo de texto donde cada elemento de la lista se separa a través del delimitador "|" dentro del campo.

Por otro lado, la segunda parte lee otro fichero adicional de datos sintéticos constituido por títulos de películas y los actores asociados, y se encarga de generar valores aleatorios que permitan rellenar todos los campos que se han constituido en la primera parte, añadiendo todos estos datos en el mismo fichero. Así, se termina de conformar el fichero *csv* que está constituido en aproximadamente un 96 % por datos sintéticos generados aleatoriamente. Este fichero será la fuente de datos a utilizar a la hora de cargar los datos en cada una de las diferentes bases de datos.

4.1.2. Análisis de requisitos

Para proceder a realizar el análisis de requisitos del conjunto de datos de cada escenario, es preciso identificar los tipos de perfiles que van a trabajar con cada tipo de conjunto, así como las tareas y operaciones que cada perfil podría realizar sobre el conjunto de datos. Es decir, se debe identificar qué tipo de consultas se realizan sobre la base de datos de cara a realizar un diseño que responda de manera rápida y eficiente a las consultas que se le plantean. Finalmente, se deben traducir estas consultas genéricas en consultas concretas que puedan ejecutarse sobre la base de datos para poder realizar el análisis.

4.1.2.1. Contexto y análisis de requisitos para las series temporales de maquinaria industrial

Los perfiles identificados como más probables para interactuar con el conjunto de datos de las series temporales y que más pueden diferir en el ámbito de búsqueda respecto al mismo conjunto de datos son el perfil del departamento técnico y el perfil analista.

El perfil del **departamento técnico** va a realizar diferentes preguntas a la base de datos que le permitan verificar el correcto funcionamiento de las máquinas y los sensores. Entre las consultas más frecuentes se encuentran las siguientes:

1.1 El cálculo de la media, mínimo y máximo (funciones agregadas) de los valores de los sensores de la máquina en un periodo de tiempo determinado.

1.2 La consulta de los valores obtenidos por encima o por debajo de un valor concreto dado para detectar desviaciones que fueran indicio de un mal funcionamiento.

1.3 Periodos de tiempo en los que los datos toman valores concretos.

1.4 Valores de un sensor entre dos fechas concretas.

El perfil **analista** va a necesitar realizar consultas generales a la base de datos que le permitan asociar los valores obtenidos al estado de los productos. Entre las consultas más frecuentes se encuentran las siguientes:

1.5 El cálculo de la media, mínimo y máximo (funciones agregadas) de los valores de los sensores de la máquina en un periodo de tiempo determinado para comprobar que los valores se corresponden con los óptimos.

1.6 Visualización de los valores de los sensores en los momentos en los que el producto físico producido por las máquinas presenta un valor concreto.

1.7 Comparar valores de un periodo de tiempo con los mismos en otro periodo de tiempo.

1.8 Comparar valores de un periodo de tiempo de un sensor concreto con los de otro sensor en el mismo periodo de tiempo.

1.9 Porcentaje de valores superiores, inferiores o iguales a unos valores dados en un periodo de tiempo concreto.

4.1.2.2. Contexto y análisis de requisitos para los datos sobre películas

Los perfiles que más pueden interactuar con los datos sobre películas en la base de datos son el perfil del *usuario común* y el perfil del *usuario cinéfilo*.

El perfil del **usuario poco frecuente o común** que quiere ver una película accede a la base de datos para analizar datos sobre las películas y escoger la que más se adecúe a sus gustos. Las operaciones que el usuario común realiza sobre la base de datos son:

2.1 Ordenar las películas por su puntuación media.

2.2 Filtrar las películas por una o varias etiquetas.

2.3 Filtrar las películas por uno o varios géneros.

2.4 Ordenar las películas filtradas por etiquetas por puntuación media.

2.5 Ordenar las películas filtradas por género por puntuación media.

2.6 Obtener la película con mejor puntuación media.

2.7 Obtener la película con mejor puntuación media para una etiqueta.

2.8 Obtener la película con mejor puntuación media para un género.

2.9 Obtener películas del género de una película concreta.

2.10 Obtener películas con la misma etiqueta que una película concreta.

El perfil del **usuario cinéfilo** comparte requisitos con el usuario frecuente y, además, realiza consultas sobre la base de datos que le permiten profundizar sobre contenido de la película, el reparto, la dirección, lugares donde ha sido filmada, etc. También realiza búsquedas directas sobre actores y películas en las que han actuado, así como directores y películas que han dirigido. Las operaciones más frecuentes que el usuario cinéfilo realiza sobre la base de datos son:

2.11 Obtener actores que han actuado en una película concreta.

2.12 Obtener directores que han dirigido una película concreta.

2.13 Obtener lugares donde se ha grabado una película concreta.

2.14 Filtrar películas por actor.

2.15 Filtrar películas por director.

2.16 Filtrar películas por país.

4.1.3. Diseño del esquema de datos

Una vez están analizadas las consultas y, por lo tanto, los requisitos que debe cumplir la base de datos para cada uno de los escenarios planteados, se procede a definir el diseño del esquema de datos para cada uno de los SGBD. Los esquemas diseñados son orientativos ya que no hay obligación de que todos los datos se adapten al 100 % al esquema en los casos en los que algunos de los campos no existan o tengan valores vacíos. Por lo tanto, los esquemas diseñados hacen referencia a los empleados en los casos en los que todos los campos tienen valores asociados.

4.1.3.1. Diseño del esquema de datos en forma de serie temporal

La base de datos que recoge los datos en forma de series temporales recibe el nombre ***TIME_SERIES_DATA***. A partir de este momento el conjunto de datos en forma de serie temporal sobre maquinaria industrial pasará a referenciarse como ***TIME_SERIES_DATA*** en este documento.

ESQUEMA DE DOCUMENTOS MONGODB (VERSIÓN 1):

Se plantea una única colección en la base de datos que almacena los valores de todos los sensores, llamada *SENSOR_VALUES*.

Colección *SENSOR_VALUES*

- **_id**: Identificador único del documento.
- **timestamp_hour**: Timestamp de la hora en formato ISODate.
    - **sec_values**: Un campo por cada minuto de una hora. Por cada minuto, un campo por cada segundo que tiene un minuto.
        - **engine**: Valor del consumo del motor.
        - **pressure**: Valor de la presión de fundido.
        - **rpm**: Valor de las revoluciones por minuto.
        - **temperature**: Valor de la temperatura.

En el fragmento de código 4.1 se muestra la representación de un ejemplo acortado de documento de MongoDB con la estructura jerárquica diseñada.

Esta estructura permite reducir el tiempo de búsqueda para aquellas preguntas en las que el campo de búsqueda es el tiempo, de modo que el coste se ve reducido a obtener el documento en base a la hora de una marca de tiempo concreta, más un coste adicional que en el peor de los casos es de 120 (lo que tarda en buscar el minuto dentro de una hora más lo que tarda en buscar el segundo dentro de ese minuto). El inconveniente de este diseño es que los documentos se deben transformar en *array*s para poder procesarlos y esto puede suponer un coste adicional en el tiempo de ejecución de las consultas.

Colocar los valores de los sensores dentro de una misma colección permite que las comparaciones entre los valores producidos por distintos sensores en un mismo rango de tiempo se reduzcan a la búsqueda de un solo documento.

```
{
  "_id": xxxxxxx,
  "timestamp_hour": ISODate("2018-12-11T00:00:00Z"),
  "sec_values": {
    "0": {
                "0": {
                        "engine":21367,
                        "pressure":202,
                        "rpm":23162,
                        "temperature":185.1,
                },
                ...,
                "59": {
                        "engine":21365,
                        "pressure":206,
                        "rpm":23171,
                        "temperature":185.7,
                }
        },
    "1": { ... },
    ...,
    "59": { ... }
  }
}
```

Código 4.1: Ejemplo reducido de documento MongoDB para la colección *SENSOR_VALUES* (versión 1)

Utilizando operaciones de paso de documentos a *array*s, todas las operaciones identificadas (de la 1.1 a la 1.9) pueden ser realizadas a través del esquema diseñado para la base de datos *TIME_SERIES_DATA* en la versión 1.

ESQUEMA DE DOCUMENTOS MONGODB (VERSIÓN 2):

Se plantea una única colección en la base de datos que almacena los valores de todos los sensores, llamada *SENSOR_VALUES*.

En el fragmento de código 4.2 se muestra un ejemplo de documento de la colección *SENSOR_VALUES*, donde cada documento almacena los valores de todos los sensores en un *timestamp* concreto.

El inconveniente de este diseño es que el espacio de almacenamiento ocupado por los datos será mayor que el utilizado por la versión 1. Sin embargo, los tiempos de ejecución medios de las consultas se verán agilizados gracias a la creación de un índice sobre el campo *timestamp*.

```
{
    "_id": xxxxxxx,
    "timestamp": ISODate("2018-12-11T00:00:59Z"),
    "values": {
        "engine":21365,
        "pressure":206,
        "rpm":23171,
        "temperature":185.7
    }
}
```

Código 4.2: Ejemplo de documento MongoDB para la colección *SENSOR_VALUES* (versión 2)

De este modo, todas las operaciones identificadas (de la 1.1 a la 1.9) pueden ser realizadas a través del esquema diseñado para la base de datos *TIME_SERIES_DATA* en la versión 2.

ESQUEMA DE TABLAS APACHE CASSANDRA:

Los datos obtenidos de todos los sensores se almacenan en una misma tabla llamada *SENSOR_VALUES*.

Tabla *SENSOR_VALUES*

- **cod_sensor**: Identificador del sensor.
- **date**: Día en el que se ha obtenido el valor.
- **timestamp**: Marca de tiempo en la que se ha obtenido el valor.
- **value**: Valor obtenido.
- **PRIMARY KEY**((cod_sensor, date), timestamp)

Esta implementación se puede ver reflejada en el fragmento de tabla representado en la figura 4.2, donde cada partición está distinguida por un color diferente.

Los valores obtenidos de un mismo sensor en un mismo día se almacenan en la misma partición debido a que la partición viene dada por los campos del *Partition Key* y, en este caso, son el nombre del sensor y el día. Cada partición es ordenada por el valor de su columna de marca de tiempo (columna de *clustering*). Este diseño evita que el espacio de columnas sea insuficiente, lo cual podría haber ocurrido si la clave de partición hubiera sido únicamente el identificador del sensor. En ese caso, todos los valores tomados por

Partition Key		*Clustering Key*	*Field*
cod_sensor	date	timestamp	value
54a25b	2018-12-11	2018-12-11T00:00:00Z	25
54a25b	2018-12-11	2018-12-11T00:00:01Z	27
54a25b	2018-12-11	2018-12-11T00:00:02Z	26
22d54a	2018-12-11	2018-12-11T00:00:00Z	700
22d54a	2018-12-11	2018-12-11T00:00:01Z	702
22d54a	2018-12-12	2018-12-12T00:00:00Z	698

Figura 4.2: Ejemplo de un fragmento de la tabla *SENSOR_VALUES*

dicho sensor formarían parte de la misma partición y ésta tiene un límite establecido en Apache Cassandra de 2 billones de columnas, es decir, se estaría estableciendo un límite de 2 billones de valores (o de marcas de tiempo) por sensor. Como los valores son tomados por segundo, cada día un único sensor debe almacenar 86400 valores. Con la estructura diseñada, cada partición (un sensor en un mismo día) almacena 86400 valores del sensor.

Las operaciones 1.1, 1.4, 1.5, 1.7 y 1.8 quedan cubiertas ya que los campos de filtrado forman parque de la *Partition Key*, por lo tanto no precisa de índices adicionales que permitan su filtrado. Debido a que el campo *value* no forma parte de la *Primary Key* no está permitido el filtrado por este campo, por lo tanto se genera un índice sobre este campo y se permiten consultas como la 1.2, 1.3, 1.6 y 1.9.

ESQUEMA DEL GRAFO NEO4J:

Para diseñar la estructura del grafo de *SENSOR_VALUES*, se genera un nodo de tipo *Value* con tres propiedades para cada uno de los valores. La propiedad *data* se trata del propio valor, la propiedad *date* es la fecha en formato *datetime* que facilita las operaciones con fechas en Neo4j, y la propiedad *sensor* indica el tipo del valor o el sensor con el que se ha obtenido (fig 4.3).

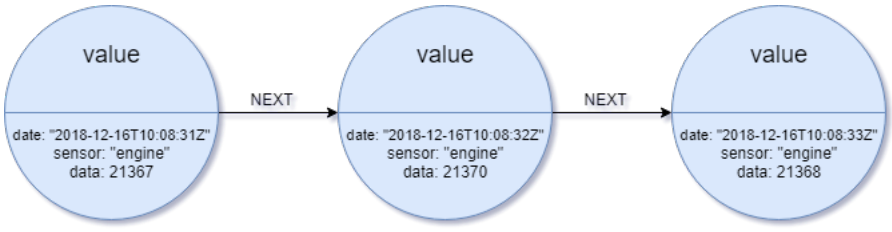


Figura 4.3: Ejemplo de la estructura del grafo *SENSOR_VALUES*

Todas las operaciones (de la 1.1 a la 1.9) pueden realizarse con al estructura del grafo diseñada. Además, creando un índice sobre la propiedad *date* del *label Value*, se aceleran los procesos de filtrado en base al *timestamp* (operaciones 1.1, 1.4, 1.5, 1.7, y 1.8).

4.1.3.2. Diseño del esquema de datos de página web

La base de datos que recoge los datos propios de una página web recibe el nombre ***FILMS_DATA***. A partir de este momento el conjunto de datos de la página web sobre películas pasará a referenciarse como ***FILMS_DATA*** en este documento.

ESQUEMA DE DOCUMENTOS MONGODB:

Se plantean 3 colecciones en la base de datos. Una de ellas almacena datos relativos a información sobre las películas (*FILMS*). Otra guarda datos referentes a información sobre géneros de películas (*GENRES_BY_FILM*). Y la última alberga datos relacionados con información sobre etiquetas de películas (*TAGS_BY_FILM*).

En el diseño de la base de datos sobre películas se ha empleado una técnica propia del diseño de bases de datos *NoSQL* como es la **redundancia**, es decir, "*repetir adrede los datos a conveniencia en varias partes de la BD (datos 'de-normalizados')*" [Mendoza, 2017] para así reducir el coste de procesamiento de las consultas que son más frecuentes, sustituyendo este coste por espacio de almacenamiento [Chodorow, 2013].

Colección *FILMS*

- **_id**: Identificador único del documento.
- **title**: Título de la película.
- **info**:
  - **year**: Año de lanzamiento de la película.
  - **image**: Dirección *url* de la imagen que proporciona IMDB sobre la película.
- **characteristics**:
  - **tags**: Lista de etiquetas asociadas.
  - **genres**: Lista de géneros asociados.
- **cast**:

- **actors**: Lista de actores de la película.
- **directors**: Lista de directores de la película.

- **location**:
  - **country**: País oficial de la película.
  - **places**: Lista de localizaciones concretas en las que ha sido grabada la película.

- **scores**:
  - **count**: Número de valoraciones registradas de la película.
  - **sum**: Suma total de las puntuaciones registradas de la película.
  - **avg**: Puntuación media sobre las valoraciones registradas de la película.

En el ejemplo de código 4.3 hay una muestra de un documento que cumple las características de un documento perteneciente a la colección *FILMS*.

```
    {
        "_id": xxxxxxx,
        "title":"Pirates of the Caribbean: At World's End",
        "info": {
                "year":"2007",
                "image":"http://ia.media-imdb.com/images/M/...",
        },
        "characteristics": {
                "tags":"pirates, caribbean, johnny deep",
                "genres":"adventure"
        },
        "cast" {
                "actors":"Johnny Deep, Keith Richards, Andy...",
                "directors":"Gore Verbinski"
        },
        "location" {
                "country":"USA",
                "places":"New York, Los Angeles",
        },
        "score" {
                "count":41523,
                "sum":200125,
                "avg":4.82
        }
}
```

Código 4.3: Ejemplo de documento MongoDB para la colección *FILMS*

Los documentos de esta colección unifican los datos generales asociados a una película de manera que la consulta de información relativa a una película concreta por su título se procesa consultando esta colección. Así, no existe la necesidad de realizar consultas adicionales para completar la información de una película, ni de relacionar un documento de la colección *FILMS* con un documento de otra colección, proceso que aumentaría significativamente el coste de procesamiento de la consulta.

Esta colección con un índice creado sobre el campo *title* facilita el procesamiento de las consultas cuyo campo de filtrado es el título de la película. Se cubren así las operaciones 2.11, 2.12, 2.13. Además, creando un índice compuesto con los campos *scores.avg* y *title*, se cubren las operaciones 2.1 y 2.6, haciendo uso únicamente de los índices para procesar la consulta, sin necesidad de que MongoDB deba acceder al documento correspondiente de la colección. Esto es posible ya que el índice compuesto está formado por el campo de filtrado (*score.avg*) y el campo consultado (*title*) y, por lo tanto, es posible responder a la consulta haciendo uso solamente del índice. Añadiendo un índice de tipo *text* sobre el campo *actors* la operación 2.14 queda cubierta.

Colección *GENRES_BY_FILM*

- **_id**: Identificador único del documento.
- **genre**: Nombre del género de la película.
- **title**: Título de la película.
- **score**: Puntuación media de la película.

En el ejemplo de código 4.4 hay una muestra de un documento acortado que cumple las características de un documento perteneciente a la colección *GENRES_BY_FILM*.

```
{
    "_id":xxxxxxx,
    "genre":"adventure",
    "title":"Pirates of the Caribbean: At World's End",
    "score":4.82
}
```

Código 4.4: Ejemplo reducido de documento MongoDB para la colección *GENRES_BY_FILM*

Esta colección permite, junto con un índice compuesto formado por los campos *genre* y *score*, obtener rápidamente las películas asociadas a un género o la película mejor puntuada de un género concreto. Se cubren las operaciones 2.3, 2.5, 2.8 y 2.9.

Colección *TAGS_BY_FILM*

- **_id**: Identificador único del documento.
- **tag**: Nombre de la etiqueta de la película.
- **title**: Título de la película.
- **score**: Puntuación media de la película.

En el ejemplo de código 4.5 hay una muestra de un documento acortado que cumple las características de un documento perteneciente a la colección *TAGS_BY_FILM*.

```
{
    "_id":xxxxxxx,
    "tag":"pirates",
    "title":"Pirates of the Caribbean: At World's End",
    "score":4.82
}
```

Código 4.5: Ejemplo reducido de documento MongoDB para la colección *TAGS_BY_FILM*

Esta colección de documentos acompañada de un índice compuesto sobre los campos *tag* y *score* facilita la consulta de películas por alguna de sus etiquetas y por su puntuación media. Se cubren las operaciones 2.2, 2.4, 2.7 y 2.10.

Esquema de tablas Apache Cassandra:

Los datos sobre las películas y su entorno se almacenan en 3 tablas diferentes. Una de ellas para almacenar información relativa a las películas, otra para guardar datos sobre géneros de películas y una última para almacenar datos sobre etiquetas asociadas a películas.

Tabla *FILMS*

- **title**: Título de la película.
- **year**: Año de lanzamiento de la película.
- **image**: Dirección *url* de la imagen que proporciona IMDB sobre la película.
- **audience**: Audiencia de la película.
- **tags**: Lista (set) de etiquetas asociadas.

- **genres**: Lista (set) de géneros asociados.
- **actors**: Lista (set) de actores de la película.
- **directors**: Lista (set) de directores de la película.
- **country**: País oficial de la película.
- **places**: Lista (set) de localizaciones concretas en las que ha sido grabada la película.
- **score**: Puntuación media sobre las valoraciones registradas de la película.
- **num**: Número de valoraciones registradas de la película.
- **total**: Suma total de las puntuaciones registradas de la película.
- **PRIMARY KEY**(title)

Esta implementación se puede ver reflejada en el fragmento de tabla representado en la figura 4.4, donde cada partición está distinguida por un color diferente.

Partition Key	*Fields*										
title	year	image	tags	genres	actors	directors	country	places	score	num	total
Pirates of the Caribbean	2007	http://i..	pirates,	advent..	Johnn..	Gore Ver..	EEUU	Hawa..	4.44	89653	398698
Jumanji	1995	http://i..	anima..	advent..	Robin ..	Joe Joh..	EEUU	Califor..	3.18	78500	250251
Nixon	1995	http://i..	presid..	biogra..	Antho..	Oliver St..	EEUU	Washi..	3.39	55896	189654

Figura 4.4: Ejemplo de un fragmento de la tabla *FILMS*

Con la tabla *FILMS* el acceso a datos sobre películas queda cubierto (operaciones 2.11, 2.12, 2.13), siendo cada película (identificada por su título único) una partición distinta. Añadiendo un índice sobre el campo *score*[4] se benefician las ordenaciones de películas según su nota media, cubriendo las operaciones 2.1 y 2.6. Añadiendo un índice para cada campo de tipo *set* (*actors, directors, locations*) y sobre el campo *country*, las operaciones 2.14, 2.15 y 2.16 quedan cubiertas.

Tabla *GENRES_BY_FILM*

- **genre**: Nombre del género de la película.
- **title**: Título de la película.
- **score**: Puntuación media de la película.

[4]CREATE INDEX score_idx ON films_data.films (score)

- **PRIMARY KEY**(genre, score)

Esta implementación se puede ver reflejada en el fragmento de tabla representado en la figura 4.5, donde cada partición está distinguida por un color diferente.

Partition Key	*Clustering Key*	
genre	score	title
adventure	4.82	Pirates of the Caribbean
adventure	3.75	Jumanji
biography	2.98	Nixon

Figura 4.5: Ejemplo de un fragmento de la tabla *GENRES_BY_FILM*

Cada género se almacena en una misma partición (quedan cubiertas las operaciones 2.3 y 2.9), y dentro de dicha partición el título de la película se ordena en función del valor del campo *score*, es decir, se ordenan las películas de un mismo género según su puntuación media, en este caso en orden descendente. Así, se cubren las operaciones 2.5 y 2.8.

Tabla *TAGS_BY_FILM*

- **tag**: Nombre de la etiqueta de la película.
- **title**: Título de la película.
- **score**: Puntuación media de la película.
- **PRIMARY KEY**(tag, score)

Esta implementación se puede ver reflejada en el fragmento de tabla representado en la figura 4.6, donde cada partición está distinguida por un color diferente.

Partition Key	*Clustering Key*	
tag	score	title
pirates	4.82	Pirates of the Caribbean
animals	3.75	Jumanji
president	2.98	Nixon

Figura 4.6: Ejemplo de un fragmento de la tabla *TAGS_BY_FILM*

Cada etiqueta se almacena en una misma partición (quedan cubiertas las operaciones 2.2 y 2.10), y dentro de dicha partición el título de la película se ordena en función del valor del campo *score*, es decir, se ordenan las películas con una misma etiqueta según su puntuación media. Así, se cubren las operaciones 2.4 y 2.7.

ESQUEMA DE GRAFOS NEO4J:

El diseño del grafo asociado al conjunto de datos sobre películas toma una estructura en la que cada elemento principal del conjunto de datos es un nodo, y todos los nodos toman como referencia el nodo *film*, es decir, tienen una relación directa con este nodo (Fig. 4.7).

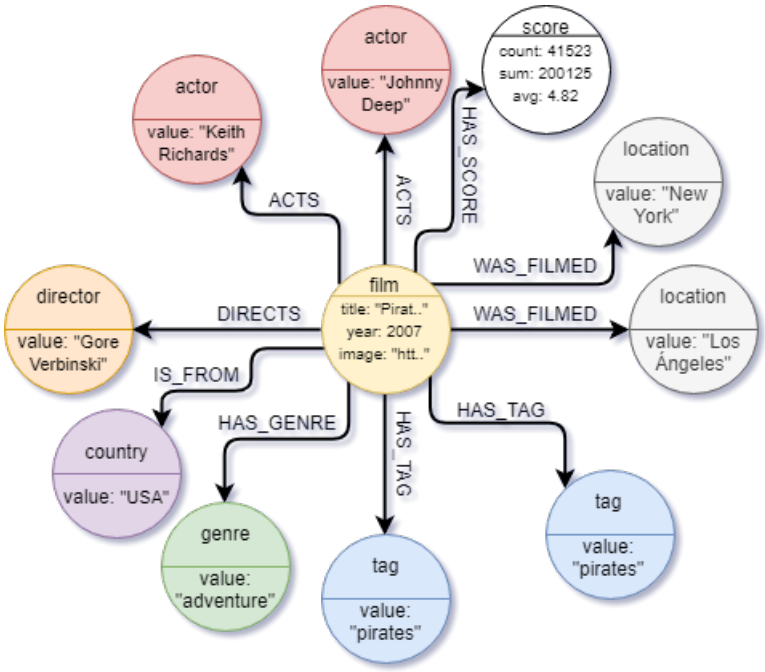


Figura 4.7: Ejemplo de la estructura del grafo para *FILMS_DATA*

Para acelerar las consultas sobre la base de datos, se genera un índice para cada uno de los *labels*[5] en función de su propiedad principal, es decir, para la propiedad *title* del tipo *film*, *name* del tipo *actor*, *name* del tipo *director*, *value* del tipo *genre*, *value* del tipo *tag*, *value* del tipo *location*, *value* del tipo *country* y *avg* del tipo *score*.

Los índices generados aceleran tanto las consultas sobre la base de datos como las sentencias de creación de nodos. Para beneficiarse de la gestión de relaciones entre nodos que proporciona Neo4j es necesario encontrar coincidencias en los nodos ya existentes a

[5]En el contexto de Neo4j, *label* es el nombre de los tipos de nodos existentes en el grafo.

la hora de insertar nuevos datos, por lo tanto, los índices toman un papel muy importante en este punto.

4.1.4. Transformación y carga de los datos

Utilizando Java[6] como lenguaje de programación sobre la plataforma Eclipse[7] se han desarrollado diferentes programas encargados de adaptar los datos al esquema asociado a cada conjunto de datos en cada una de las bases de datos. El proyecto ha sido gestionado y construido con la herramienta de software *Maven*[8] por la fácil gestión de dependencias que proporciona. Por lo tanto, las librerías utilizadas para el desarrollo del proyecto están mencionadas en el fichero de configuración *pom.xml* y el gestor de dependencias de *Maven* se encarga automáticamente de importar las bibliotecas desde su repositorio remoto.

4.1.4.1. Conector entre el programa Java y la base de datos

Se han utilizado los *drivers* de Java para MongoDB, DataStax Apache Cassandra y Neo4j como mecanismo de enlace entre el programa Java encargado de transformar los datos y las propias bases de datos, permitiendo cargar los datos con su correspondiente estructura directamente sobre la base de datos desde el propio programa Java. Las bibliotecas utilizadas para la gestión de los *drivers* de MongoDB, DataStax Apache Cassandra y Neo4j son *org.mongodb*, *com.datastax.cassandra* y *org.neo4j.driver* respectivamente, y se han añadido al árbol de dependencias manteniendo la estructura representada en el fragmento de código 4.6.

[6] Java: https://www.java.com/es/
[7] Eclipse: https://www.eclipse.org/
[8] Maven: https://maven.apache.org/

```
<dependencies>
    <dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-core</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongo-java-driver</artifactId>
        <version>3.10.1</version>
    </dependency>
    <dependency>
        <groupId>org.neo4j.driver</groupId>
        <artifactId>neo4j-java-driver</artifactId>
        <version>1.7.2</version>
    </dependency>
</dependencies>
```

Código 4.6: Dependencias de las bibliotecas incluidas en el fichero de configuración *pom.xml*

La estructura del programa encargado de transformar y cargar los datos sobre las diferentes bases de datos consta de los conectores a los diferentes SGBD, así como la parte de lectura y transformación directa del conjunto de datos. En el diagrama de la figura 4.8 se muestra la representación UML del programa encargado de la transformación y carga del conjunto de datos sobre cada base de datos.

Las clases encargadas de leer y transformar los datos del origen de datos (*CSVReaderMongo, CSVReaderCassandra, CSVReaderNeo4j*) comparten una serie de funcionalidades (métodos auxiliares) asociadas a la obtención y adaptación de los datos, que se exponen en una clase no instanciable de carácter abstracto que recoge dichas funcionalidades comunes. Los conectores (*MongoConnector, CassandraConnector, Neo4jConnector*) tienen implementados los métodos de conexión y de inserción en la base de datos. Este programa se encarga únicamente de enriquecer la base de datos con contenido, por lo que no es necesario implementar métodos adicionales relacionados con otras acciones sobre la base de datos como operaciones de selección, actualización o eliminación que, en su caso, deberían ser incluidas en este lugar.

El programa se encarga de leer y preparar los datos desde las clases *CSVReader**, utilizando los conectores **Connector* para insertar esos datos que van leyendo en cada una de las bases de datos.

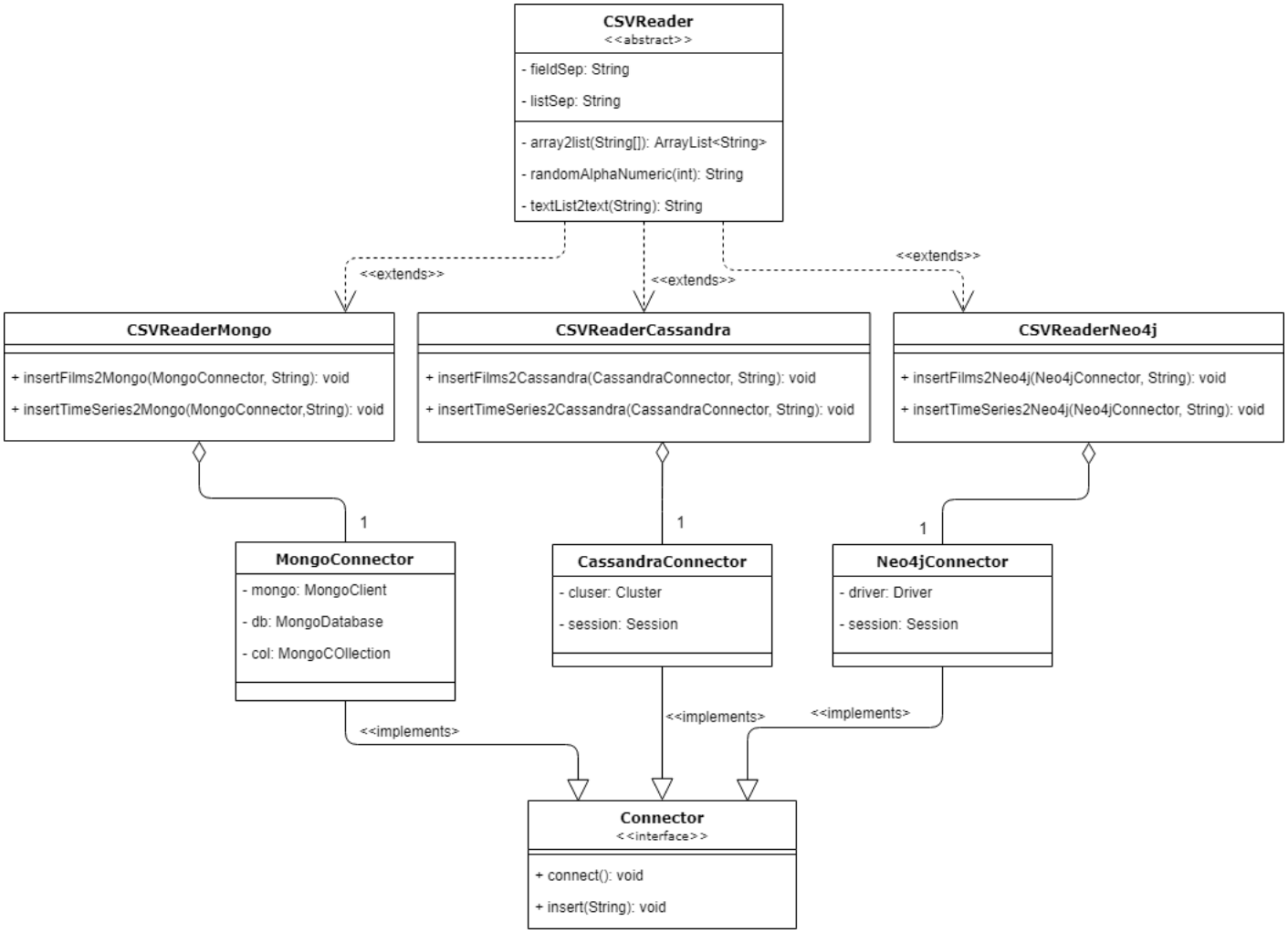


Figura 4.8: Diagrama UML de las clases del programa para la transformación y carga de datos

4.1.4.2. Transformación y carga del conjunto de datos *TIME_SERIES_DATA*

Tomando como origen de datos los ficheros en formato *csv* descritos en el apartado 4.1.1.1), se procede a adaptar el contenido al esquema diseñado para cada uno de los SGBD.

La manera en la que los datos se insertan en las bases de datos es similar, y se representa en pseudocódigo en el fragmento de código 4.7. Este método es utilizado para el caso de MongoDB y de Apache Cassandra, donde los datos son insertados desde el driver de Java. Sin embargo, para el caso de Neo4j se ha planteado un *script* en lenguaje *Cypher* que se ejecuta directamente en el cliente de Neo4j.

```
foreach (fichero de datos) {
    while (hay nueva línea del fichero para leer) {
        campos <- obtener campos de la línea
        adaptar campos para trabajar con ellos
        conector -> ejecutar sentencia de inserción de datos
    }
}
```

Código 4.7: Pseudocódigo del método de lectura e inserción en la base de datos *TIME_SERIES_DATA*

MONGODB:

Para la inserción de los datos de series temporales en el SGBD MongoDB se ha utilizado el paquete de java *org.bson.Document* que facilita la generación de documentos en formato *bson* y que se trata del tipo de objeto admitido como parámetro de entrada por los métodos de inserción en colecciones que proporciona el *driver* de MongoDB para Java.

Tras la generación del documento, la personalización de los objetos de tipo *org.bson.Document* tiene asociada una funcionalidad *append(fieldName, fieldValue)* que permite añadir campos al documento de manera sencilla. El valor del campo puede ser de tipo simple (*String, Integer, Boolean...)* u otro objeto de tipo *org.bson.Document*.

Por un lado, para la versión 1 del esquema diseñado (apartado 4.1.3.1), se genera un documento *bson* cada vez que la marca temporal coincida con una hora exacta, y se le añade el campo *timestamp_hour* con la fecha de tipo *date* como valor. Los siguientes valores hasta llegar a una hora exacta nueva se añaden al documento en el campo *sec_values*. Para ello, cada minuto exacto leído forma un nuevo campo dentro del subdocumento *sec_values*, y cada segundo exacto, de nuevo, otro campo dentro del subdocumento del minuto que le

corresponde. Finalmente, cada segundo es un subdocumento con 4 campos, uno por cada sensor asociado ese *timestamp*.

Por otro lado, para la versión 2 del esquema diseñado (apartado 4.1.3.1), se genera un nuevo documento para cada marca de tiempo, de modo que dicho valor se añade al campo *timestamp*, y en el campo *values* se añade un subdocumento con un campo por cada sensor con el valor asociado a esa marca de tiempo.

Tanto para la versión 1 como para la versión 2, en cada ciclo del bucle de lectura de ficheros se lee un valor de cada sensor en vez de insertar los datos de sensor en sensor, es decir, se trabaja con todos los tipos de sensores a la vez para facilitar la inserción de los valores en los documentos.

APACHE CASSANDRA:

Utilizando *CQL*[9] como lenguaje de consultas de Cassandra [Strauch, 2011], se genera una consulta de inserción en la tabla *SENSOR_VALUES* por cada fila leída del origen de datos.

Para conseguir el esquema de Cassandra diseñado en la sección 4.1.3.1, se obtiene de cada fila de los ficheros correspondientes el dato de la fecha y el del valor, que junto con el nombre del sensor asociado a ese valor, se utilizan para generar una sentencia en lenguaje *CQL* que se encarga de insertar esos datos en la tabla *SENSOR_VALUES* de la base de datos *TIME_SERIES_DATA*.

NEO4J:

Para insertar los datos sobre series temporales en la base de datos de Neo4j se ha generado un *script* escrito en *Cypher* que para cada fichero obtiene de cada línea la fecha y el valor del dato, y junto con el nombre del sensor se genera un nuevo nodo de tipo *Value* (Cod. 4.8).

```
LOAD CSV WITH HEADERS FROM 'fileName' AS row
WITH datetime(row.date) AS date,
(CASE row.value
    WHEN 'NA' THEN 0
    ELSE toInteger(row.value) END) AS value
CREATE (:Value {sensor: "sensorName", data: value, date: date})
```

Código 4.8: Script en *Cypher* de generación de nodos para *TIME_SERIES_DATA*

[9] CQL: Cassandra Query Language

Además, tras la generación de los nodos, se ejecuta un nuevo *script* encargado de relacionar los nodos entre ellos (Cod. 4.9). Este *script* obtiene los nodos de un tipo de sensor, los ordena por su fecha y seguidamente genera una relación *NEXT* desde el primero en tiempo al siguiente.

```
MATCH (v:Value {sensor: "sensorName"})
WITH v
ORDER BY v.date
WITH collect(v) AS values
FOREACH(i in RANGE(0, size(values)-2) |
FOREACH(v1 in [values[i]] |
    FOREACH(v2 in [values[i+1]] |
        CREATE (v1)-[:NEXT]->(v2))));
```

Código 4.9: Script en *Cypher* de generación de relaciones entre nodos para *TIME_SERIES_DATA*

El *script* de generación de nodos se ejecuta para cada fichero, modificando el nombre del sensor según corresponda en el momento de la inserción. El *script* de generación de relaciones entre nodos se ejecuta una vez para cada tipo de sensor, ya que se encarga de crear relaciones *NEXT* entre nodos de un mismo tipo de sensor.

4.1.4.3. Transformación y carga del conjunto de datos *FILMS_DATA*

Tomando como origen de datos el fichero en formato *csv* generado en el proceso de preparación de los datos (apartado 4.1.1.2), se procede a adaptar el contenido al esquema diseñado para cada uno de los SGBD.

La estructura del método de inserción de películas sigue un patrón común en cada uno de los tres casos cuya representación en pseudocódigo se muestra en el fragmento de código 4.10.

Cabe destacar la comprobación de los campos vacíos ya que en caso de serlo no se añaden a la base de datos, haciendo uso de la propiedad característica de las bases de datos *NoSQL* que permite que los elementos de almacenamiento que utilizan no deban presentar un esquema idéntico, aportando cierta flexibilidad y ahorrando espacio de almacenamiento.

```
while (hay nueva línea del fichero para leer) {
    campos <- obtener campos de la línea
    adaptar campos para trabajar con ellos
    foreach (campo) {
        if (campo no es vacío) {
            añadir campo a la sentencia de inserción de película } }
    conector -> ejecutar sentencia de inserción de película

    foreach (género) {
        conector -> ejecutar sentencia de inserción de género }

    foreach (etiqueta) {
        conector -> ejecutar sentencia de inserción de etiqueta }
```

Código 4.10: Pseudocódigo del método de lectura e inserción en la base de datos *FILMS_DATA*

MONGODB:

Para la inserción de los datos sobre películas en el SGBD MongoDB se ha utilizado el paquete de java *org.bson.Document* que facilita la generación de documentos en formato *bson* y que se trata del tipo de objeto admitido como parámetro de entrada por los métodos de inserción de colecciones que proporciona el *driver* de MongoDB para Java.

Tras la generación del documento, la personalización de los objetos de tipo *org.bson.Document* tiene asociada una funcionalidad *append(fieldName, fieldValue)* que permite añadir campos al documento de manera sencilla. El valor del campo puede ser de tipo simple (*String, Integer, Boolean...)* u otro objeto de tipo *org.bson.Document* (fig 4.9).

Siguiendo este esquema, el método de inserción de documentos a las colecciones de *FILMS*, *GENRES_BY_FILM* y *TAGS_BY_FILM* utiliza el objeto *org.bson.Document* para insertar documentos con el formato especificado en el apartado MongoDB de la sección 4.1.3.2.

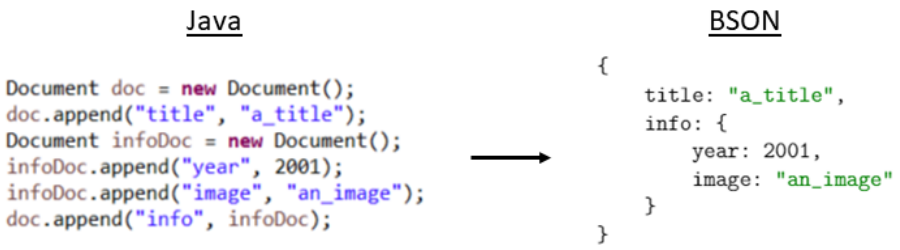


Figura 4.9: Equivalencia de un objeto Java de tipo *org.bson.Document* y un documento BSON

APACHE CASSANDRA:

Utilizando *CQL* como lenguaje de consultas de Cassandra, se genera una consulta de inserción por cada fila leída del origen de datos para cada una de las tres tablas existentes sobre películas (*FILMS*, *GENRES_BY_FILM* y *TAGS_BY_FILM*).

Para reducir el tiempo de inserción de cada una de las filas del origen de datos en la distintas tablas, se emplea la funcionalidad *batch*[10] que permite ejecutar varias sentencias conjuntamente evitando confirmar las inserciones en la base de datos una por una, reduciendo así el tiempo de confirmación teniendo lugar al final de la ejecución únicamente.

Tras ejecutar las sentencias de inserción para todo el origen de datos, la estructura de la base de datos adquiere el formato indicado en el apartado Apache Cassandra de la sección 4.1.3.2.

NEO4J:

Utilizando el lenguaje de consultas *Cypher* para Neo4j se prepara para cada línea del origen de datos una sentencia de inserción de datos. La sentencia en formato de texto se le pasa como parámetro al método de inserción del conector Neo4j preparado, junto con otro parámetro de tipo *Map<String, Object>* que hace referencia a los parámetros que se necesitan para ejecutar la sentencia. El *driver* de Neo4j para Java dispone de un método *run(String, Map<String,Object>)* sobre la clase *Session* que permite utilizar parámetros en la ejecución de la sentencia *Cypher* sobre Neo4j. Cada tupla del objeto de tipo *Map* es un parámetro, siendo su nombre el primer campo de la tupla y su valor el segundo. Para hacer referencia al valor del parámetro, en la sentencia debe indicarse su nombre entre llaves (fig 4.10) [Robinson et al., 2015].

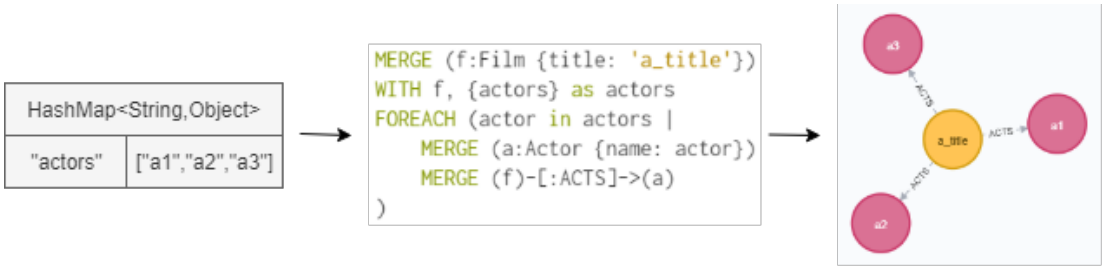


Figura 4.10: Paso de parámetros a sentencia *Cypher* y su representación en grafo

La funcionalidad del paso de parámetros junto con la sentencia *Cypher* se emplea para

[10]BEGIN BATCH *consultas* APPLY BATCH;

indicar las listas de actores, directores, géneros, etiquetas y localizaciones que deben ser relacionadas con la película.

4.2. Análisis de rendimiento de los SGBD

Una vez se han preparado las bases de datos y se ha insertado el contenido con su correspondiente formato, comienza el proceso de análisis del rendimiento de los SGBD a través de las bases de datos generadas. Para realizar esta evaluación se van a tener en cuenta dos factores: espacio de almacenamiento ocupado por el conjunto de datos en la base de datos y tiempo de ejecución medio de las sentencias.

Por una lado, se tiene en cuenta el modo en el que cada base de datos tiene almacenados los datos y el espacio de almacenamiento que ocupa en consecuencia. En este contexto se entiende espacio de almacenamiento ocupado como la cantidad de espacio que ocupan los datos contenidos en la base de datos, así como los índices que hayan podido ser generados. Se excluyen de este análisis otro tipo de almacenamiento como *logs*, ficheros de configuración u otros.

Por otro lado, para realizar un análisis del tiempo medio de ejecución de las sentencias sobre las bases de datos, se identifica una lista de consultas a realizar sobre las bases de datos que deben cubrir los diferentes tipos de operaciones que se pueden efectuar sobre las base de datos y que concretan las operaciones identificadas en el análisis de requisitos previo (apartado 4.1.2). Con las sentencias identificadas, se traduce cada una de ellas al lenguaje de consultas de cada uno de los SGBD y, finalmente, se ejecutan sobre la base de datos empleando funcionalidades de control de tiempo de ejecución.

4.2.1. Análisis de rendimiento base de datos *SENSOR_VALUES*

4.2.1.1. Ocupación del espacio de almacenamiento por SGBD

MONGODB:

Dado que se han diseñado dos esquemas (versión 1 y versión 2) para la base de datos *TIME_SERIES_DATA* en MongoDB, en este apartado se realiza una comparación entre ambos diseños para continuar el análisis de los diferentes SGBD con el más eficiente de los dos.

La expresión *db.collection.totalSize()*[11] devuelve el tamaño en *bytes* de los datos de la colección más el tamaño de cada índice de la colección. Ejecutando la expresión, se obtiene el tamaño en *bytes* de la única colección de la base de datos *TIME_SERIES_DATA*.

En la versión 1 este tamaño viene dado por los datos de la colección, y ocupan un total de 13'54 MB. En este caso el 100 % del espacio de almacenamiento está ocupado por los datos ya que no se ha generado ningún índice adicional.

En la versión 2 el tamaño es de 41'01 MB, donde 25'57 MB son datos y 15'44 MB son índices. En este caso el 62'35 % del espacio de almacenamiento está ocupado por los datos y el 37'65 % por los índices.

Teniendo en cuenta el espacio de almacenamiento ocupado por los datos la versión 2 ocupa aproximadamente el doble que la 1. Sin embargo, en su totalidad la base de datos que sigue el diseño de la versión 2 ocupa casi un 64 % más que la basada en la versión 1. Esto es porque la primera no tiene índices debido a que no mostraron mejoras en las ejecuciones.

APACHE CASSANDRA:

Utilizando la herramienta *nodetool cfstats*[12] que proporciona *DataStax Apache Cassandra*, se ejecuta la expresión *nodetool cfstats TIME_SERIES_DATA* y devuelve el tamaño en *bytes* de cada una de las tablas e índices que compone el *keyspace TIME_SERIES_DATA*. En este caso la tabla *SENSOR_VALUES* junto con los índices ocupan un espacio de 43'82 MB (26'1 MB en datos y 17'72 MB en índices).

Los índices utilizan un 40'42 % del espacio de almacenamiento que la base de datos *TIME_SERIES_DATA* ocupa en disco, mientras que los propios datos utilizan un 59'38 % del espacio.

NEO4J:

Para comprobar el espacio de almacenamiento que el grafo *SENSOR_VALUES* ocupa en disco se ejecuta la expresión *:sysinfo* y se obtiene el campo *Total Store Size*. El grafo *SENSOR_VALUES* ocupa 1'02 GB en el disco (849'57 MB en datos y 194'91 MB en índices).

[11] *db.collection.totalSize()*: https://docs.mongodb.com/manual/reference/method/db.collection.totalSize/

[12] *nodetool cfstats*: https://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsCfStatsRedirect.html

Los índices generados utilizan un 18'66 % del espacio de almacenamiento que el grafo *SENSOR_VALUES* ocupa en disco, mientras que los propios datos utilizan un 81'33 % del espacio.

4.2.1.2. Tiempo medio de ejecución por SGBD

Paso 1. ***Identificación de las sentencias a ejecutar***

- **Selección**:
    1. **Valores de un sensor entre dos fechas concretas**: obtener los valores del sensor *engine* entre las fechas 14/12/2018 00:00:00 y 15/12/2018 23:59:59.
    2. **Valores máximo, mínimo y media de un sensor entre dos fechas concretas**: obtener los valores máximo, mínimo y media de los valores del sensor *engine* entre las fechas 14/12/2018 00:00:00 y 15/12/2018 23:59:59.
    3. **Fecha de un valor máximo de un sensor entre dos fechas concretas**: obtener el momento en la que se produce el valor máximo del sensor *engine* entre las fechas 14/12/2018 00:00:00 y 15/12/2018 23:59:59.
    4. **Fechas en las que el valor es menor a un valor dado**: obtener las fechas en las que se produce un valor inferior a 20500 por el sensor *engine*.
    5. **Valores de más de un sensor entre dos fechas concretas**: obtener los valores del sensor *engine* y *pressure* entre las fechas 14/12/2018 00:00:00 y 15/12/2018 23:59:59.
- **Inserción**:
    6. **Inserción de un valor para un sensor en una fecha dada**: insertar el valor 23125 del sensor *engine* en la fecha 17/12/2018 00:00:00.

- **Eliminación**:

  7. **Eliminación de datos hasta una fecha**: eliminar los valores anteriores al 10/12/2018 19:59:59 para el sensor *engine*.

- **Actualización**:

  8. **Modificación del valor de un sensor en una fecha dada por un valor dado**: actualizar el valor del 15/12/2018 12:15:27 en el sensor *engine* por 21000.

Paso 2. ***Ejecución de las sentencias y resultados obtenidos***

Cada sentencia se ha ejecutado cinco veces y se determina el tiempo de ejecución medio de cada sentencia como la media entre los 5 valores obtenidos para cada SGBD.

MONGODB:

Los resultados de la versión 1 muestran que el tiempo de ejecución medio de las consultas es de 564'82 ms, siendo 901'84 ms para las operaciones se selección y 3,3 ms el empleado en las operaciones de inserción, eliminación y actualización (Fig. 4.11).

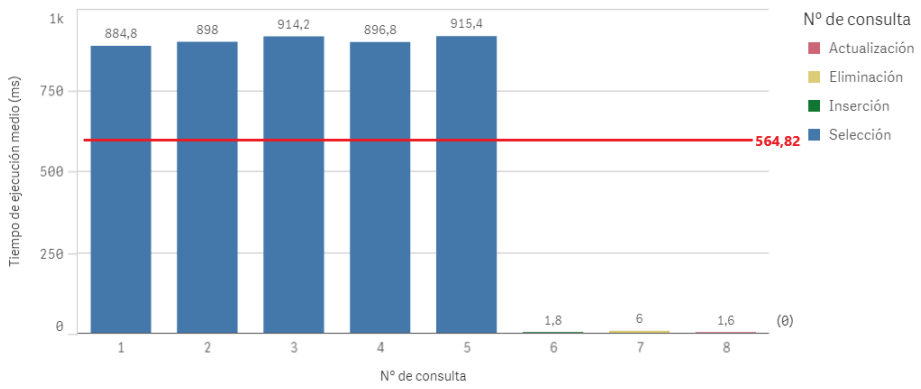


Figura 4.11: Gráfico del tiempo de ejecución medio por consulta de la base de datos *TIME_SERIES_DATA* de MongoDB (versión 1)

El tiempo se ve incrementado debido a la complejidad de las operaciones que el SGBD debe realizar sobre la base de datos.

En cambio, los resultados de la versión 2 reflejan que el tiempo de ejecución medio de las consultas es de 203'05 ms, siendo 282'2 ms para las operaciones se selección y 41'4 ms el empleado en las operaciones de inserción, eliminación y actualización (Fig. 4.12).

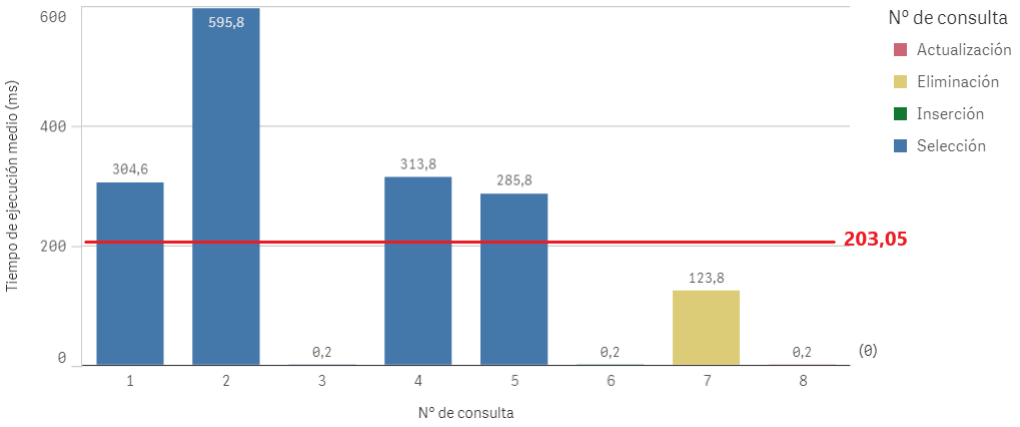


Figura 4.12: Gráfico del tiempo de ejecución medio por consulta de la base de datos *TIME_SERIES_DATA* de MongoDB (versión 2)

El tiempo de ejecución medio para las consultas es más de un 50 % inferior en la base de datos de la versión 2. Aunque el espacio de almacenamiento ocupado por la versión 2 sea un 64 % mayor, dada la gran diferencia en el tiempo de ejecución de las consultas y que el coste del espacio es poco significativo, se determina que la versión más eficiente es la número 2 y es la base de datos representativa de MongoDB a partir de este apartado en este documento.

APACHE CASSANDRA:

Para obtener el tiempo de ejecución medio de las consultas en Apache Cassandra se utiliza la funcionalidad *tracing on* disponible en la consola *cqlsh* de *DataStax Apache Cassandra*. Uno de los valores que devuelve esta opción es el tiempo de ejecución de la sentencia consultada, dado en microsegundos.

Los tiempos para las consultas de selección son, en media, muy similares. Lo mismo ocurre con las operaciones de escritura. Sin embargo, la diferencia en media de ambos tipos de operaciones es bastante significativa.

El tiempo de ejecución medio para las consultas es de 162’48 ms, siendo el de las de selección 257’64 ms, y 3’89 ms el de las operaciones de inserción, actualización y borrado.

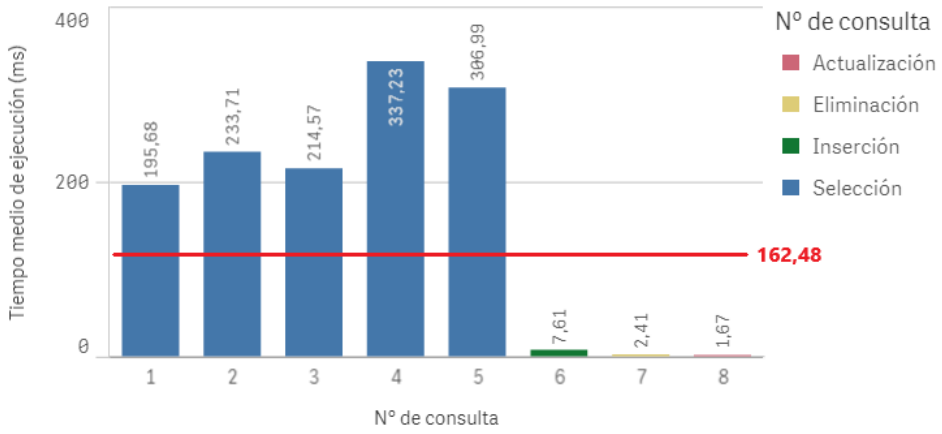


Figura 4.13: Gráfico del tiempo de ejecución medio por consulta de la base de datos *TIME_SERIES_DATA* de Apache Cassandra

NEO4J:

El tiempo de ejecución de las consultas en Neo4j se puede obtener añadiendo la expresión *PROFILE* antes de la consulta, que devuelve el valor en milisegundos (ms).

El tiempo de ejecución medio para las consultas es de 1195'86 ms, siendo el de las de selección de 1530'64 ms, y 638 ms para las de inserción, eliminación y actualización.

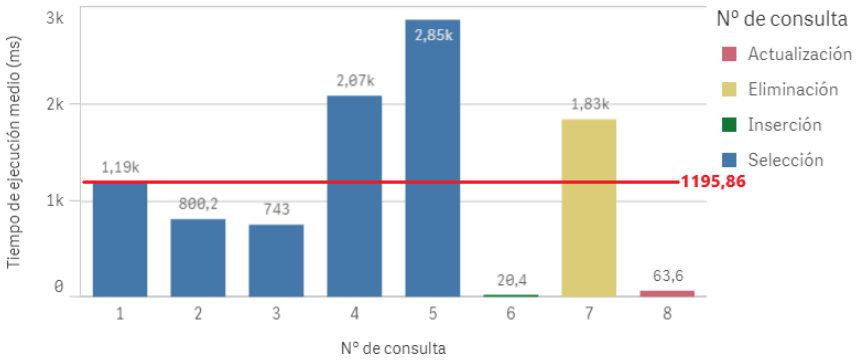


Figura 4.14: Gráfico del tiempo de ejecución medio por consulta del grafo *TIME_SERIES_DATA* de Neo4j

4.2.1.3. Comparativa global

El espacio de almacenamiento que el mismo conjunto de datos adaptado a un esquema específico ocupa en los diferentes SGBD es muy grande para Neo4j. Queda reflejado que

Neo4j necesita más del 99'95 % del espacio que necesitan MongoDB y Apache Cassandra para almacenar el mismo conjunto de datos (Fig. 4.15).

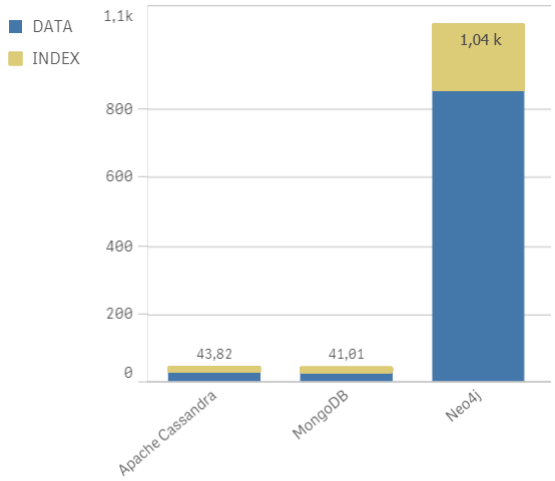


Figura 4.15: Distribución del espacio de almacenamiento de *TIME_SERIES_DATA* en cada SGBD

Este dato es muy significativo para este escenario ya que trabajando con series temporales se deben añadir nuevos valores a la base de datos en periodos muy cortos de tiempo de manera continua, por lo que supondría un incremento mayor en Neo4j que en el resto de SGBD. Esto se puede ver reflejado en la representación gráfica del espacio de almacenamiento que cada base de datos ha ocupado cada 100.000 filas (aprox.) insertadas (Fig. 4.16).

Las tres bases de datos se representan, aproximadamente, con una recta creciente donde la pendiente es considerablemente mayor en el caso de la base de datos de Neo4j, la cual indica que según se vayan insertando nuevos valores a la base de datos, el espacio de almacenamiento ocupado por la base de datos Neo4j se va a incrementar en mas de un 99 % de lo que lo harían las otras dos bases de datos.

El inconveniente que Neo4j presenta respecto al espacio de almacenamiento es un indicativo de que este SGBD se debe rechazar como gestor de la base de datos para las series temporales.. Aunque la base de datos de Apache Cassandra ocupe más espacio y la pendiente que refleja sea mayor que la de MongoDB, no se trata de una diferencia muy significativa, y podría ser compensada con los resultados obtenidos respecto a los tiempos de ejecución analizados.

Respecto al tiempo de ejecución medio de las diferentes consultas, el SGBD que menor

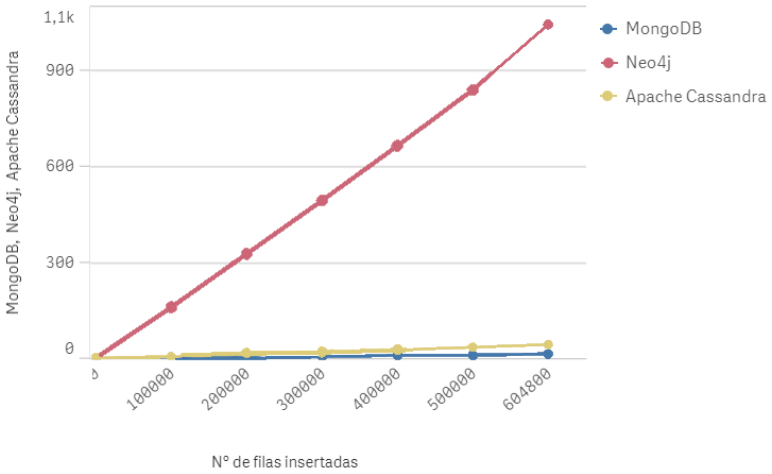


Figura 4.16: Representación gráfica del espacio de almacenamiento ocupado progresivamente cada 100.000 filas insertadas por SGBD para *TIME_SERIES_DATA*

coste de tiempo necesita es Apache Cassandra, con 162'48 ms de tiempo medio utilizado para todas las consultas. En cuanto a MongoDB, el tiempo medio de ejecución de todas las consultas es de 203'05 ms, y el de Neo4j es de 1195'9 ms.

A modo global, Apache Cassandra presenta unos valores de tiempo de ejecución más bajos. Sin embargo, como se puede apreciar en las operaciones de selección (Fig. 4.17) y las operaciones de inserción, actualización y borrado (Fig. 4.18), la diferencia entre los tiempos de ejecución medios entre Apache Cassandra y MongoDB no son muy distintos, y en algunas de las consultas MongoDB consigue resultados más positivos.

Para el escenario *TIME_SERIES_DATA* queda descartado como SGBD más eficiente Neo4j, ya que la base de datos generada en este entorno presenta los resultados en espacio de almacenamiento y tiempos de ejecución medios de las consultas más altos entre las tres bases de datos implementadas. Entre Apache Cassandra y MongoDB los resultados son similares, siendo la primera ligeramente más eficiente respecto al tiempo de ejecución y la segunda respecto al espacio de almacenamiento, aunque se debe valorar que son muy destacables los bajos costes de tiempo en las consultas de MongoDB en las que son utilizados los índices generados, por lo que en otros contextos que requieran de otro esquema de la base de datos, MongoDB podría mantenerse líder, también, en el coste de los tiempos de ejecución. Por lo tanto, un SGBD será más eficiente que el otro en función del aspecto que se quiera priorizar, teniendo en cuenta que los resultados obtenidos entre ambos son muy semejantes.

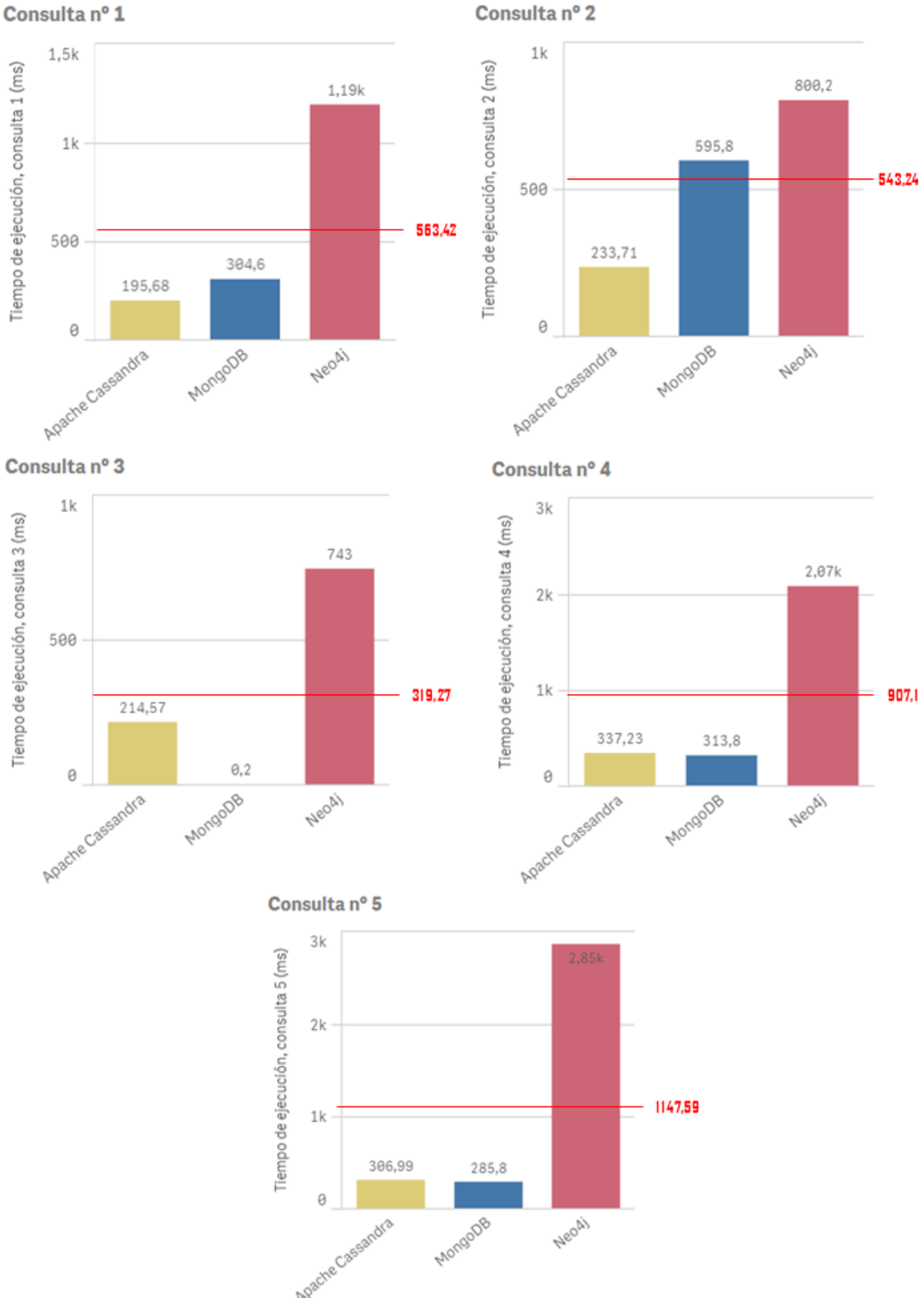


Figura 4.17: Gráficos del tiempo de ejecución medio de las consultas de selección por SGBD para *TIME_SERIES_DATA*

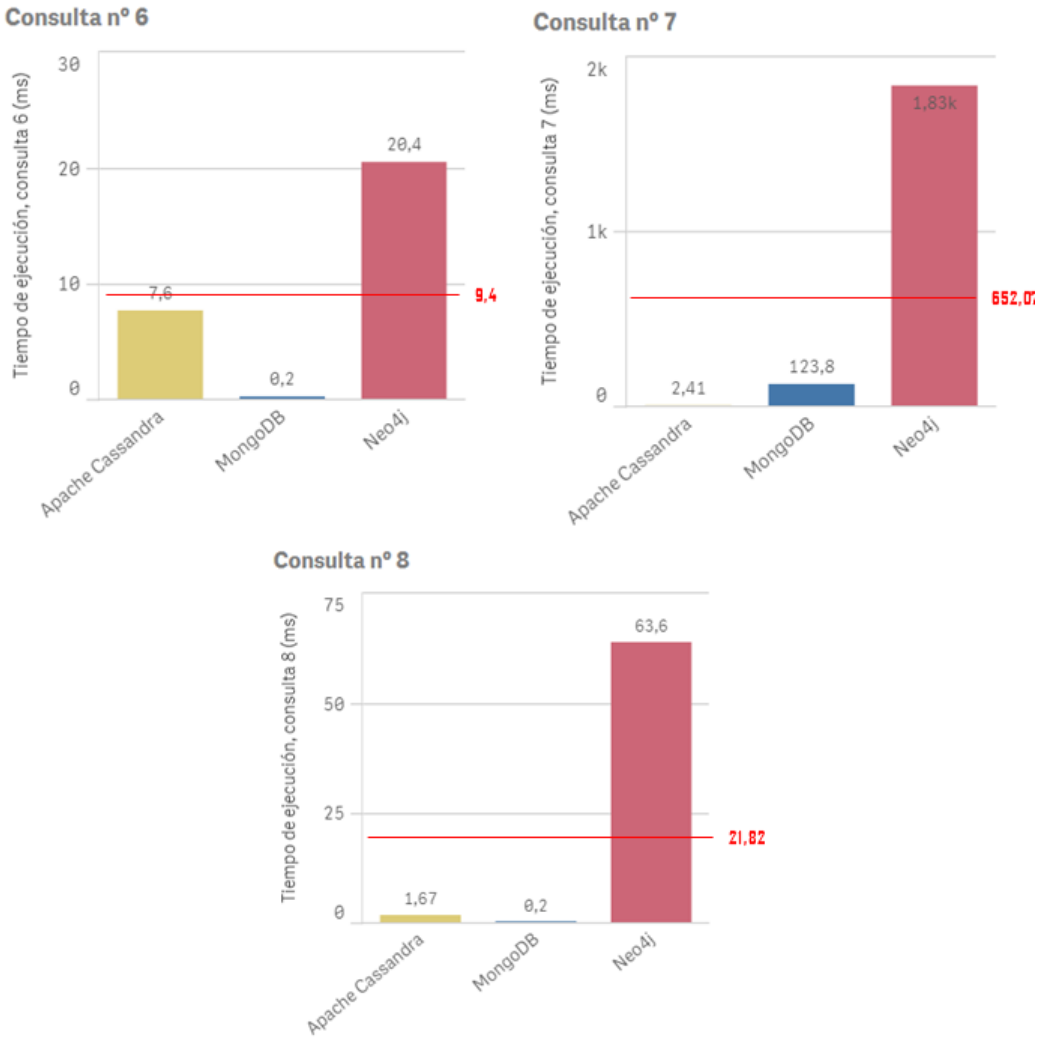


Figura 4.18: Gráficos del tiempo de ejecución medio de las consultas de inserción, eliminación y actualización por SGBD para *TIME_SERIES_DATA*

4.2.2. Análisis de rendimiento base de datos *FILMS_DATA*

4.2.2.1. Ocupación del espacio de almacenamiento por SGBD

MONGODB:

Utilizando la expresión *db.collection.totalSize()*, se obtiene que el tamaño total de la base de datos *FILMS_DATA* viene dado por la suma del tamaño total de las colecciones *FILMS* (303767552 *bytes*: 196800512 *bytes* en datos y 106967040 *bytes* en índices), *GENRES_BY_FILM* (28868608 *bytes*: 19968000 *bytes* en datos y 8900608 *bytes* en índices) y *TAGS_BY_FILM* (79597568 *bytes*: 50040832 *bytes* en datos y 29556736 *bytes* en índices), es decir, un total de 393'14 MB.

Los datos ocupan un 64'7 % del espacio de almacenamiento, mientras que los índices utilizan un 35'3 % del espacio ocupado por la base de datos *FILMS_DATA*.

APACHE CASSANDRA:

Utilizando la herramienta *nodetool cfstats* que proporciona *DataStax Apache Cassandra* se ejecuta la expresión *nodetool cfstats FILMS_DATA* y devuelve el tamaño en *bytes* de cada una de las tablas e índices que compone el *keyspace FILMS_DATA*. En este caso las tablas *FILMS*, *GENRES_BY_FILM* y *TAGS_BY_FILM* junto con los índices ocupan un espacio total de 306'24 MB (148'78 MB en datos y 157'46 MB en índices).

Los datos ocupan un 51'4 % del espacio de almacenamiento, mientras que los índices utilizan un 48'6 % del espacio ocupado por la base de datos *FILMS_DATA*.

NEO4J:

Para comprobar el espacio de almacenamiento que el grafo *FILMS_DATA* ocupa en disco se ejecuta la expresión *:sysinfo* y se obtienen los campos *Index Store* y *Total Store Size*. El grafo *SENSOR_VALUES* ocupa 3'43 GB en el disco de los cuales 137'23 MB están ocupados por los índices.

Los datos ocupan un 96'1 % del espacio de almacenamiento, mientras que los índices utilizan un 3'9 % del espacio ocupado por el grafo *FILMS_DATA*.

4.2.2.2. Tiempo medio de ejecución por SGBD

Paso 1. ***Identificación de las sentencias a ejecutar***

- **Selección**:
  1. **Valor máximo entre las puntuaciones**: película mejor puntuada.
  2. **Filtrar películas por un género dado**: películas del género *adventure*.
  3. **Valor máxima entre las puntuaciones por películas filtradas por un género dado**: película mejor puntuada para el género *adventure*.
  4. **Primeras *n* películas en función de la mayor puntuación**: 10 películas mejor puntuadas para el género *adventure*.
  5. **Películas con un actor dado**: películas en las que actúa *Johnny Depp*.
  6. **Actores de una película dada**: actores que han actuado en la película *Life of Brian*.
- **Inserción**:
  7. **Inserción de una nueva película**: añadir la película *AAA* del año *1234* en *USA*, con url *miURL*, actores *a1*, *a2* y *a3*, director *dir*, género *Adventure*, con *5* votos y una suma de *5* (*1* de media).
- **Eliminación**:
  8. **Eliminación de una película**: liminar la película titulada *AAA*.
- **Actualización**:
  9. **Modificación de un campo de una película**: modificar el año de la película *Life of Brian* a 2001.

Paso 2. ***Ejecución de las sentencias y resultados obtenidos***

Cada sentencia se ha ejecutado cinco veces y se determina el tiempo de ejecución medio de cada sentencia como la media entre los 5 valores obtenidos para cada SGBD.

MONGODB:

Para obtener el tiempo de ejecución de las consultas en MongoDB se utiliza la expresión *explain("executionStats")* y se anota el valor del campo *executionTimeInMillis*, que devuelve el valor del tiempo de ejecución de la sentencia que acompaña a la expresión en milisegundos.

Analizando los resultados obtenidos (Fig. 4.19), se observa que ninguna de las consultas supera los 5 *ms* en tiempo medio de ejecución. La media entre todas las consultas es de 1'7 ms, siendo 1'43 ms el tiempo medio para las consultas de selección y 2'27 ms el empleado en las consultas de inserción, eliminación y actualización.

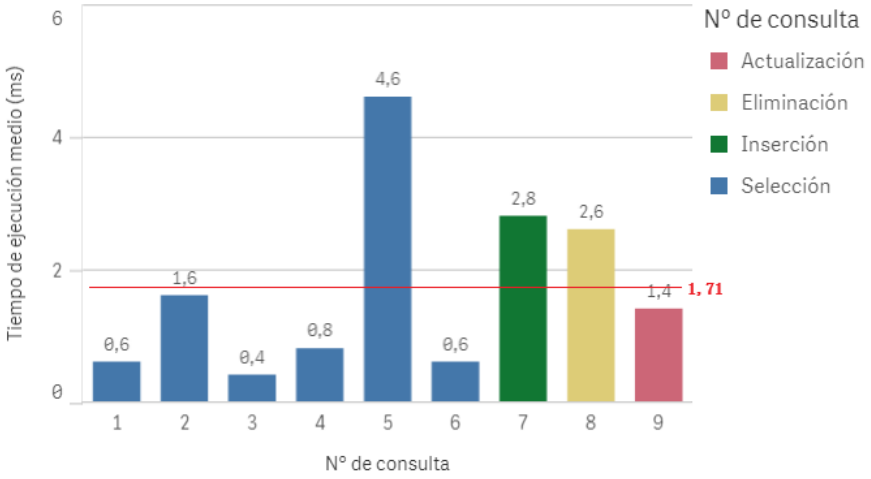


Figura 4.19: Gráfico del tiempo de ejecución medio por consulta de la base de datos *FILMS_DATA* de MongoDB

APACHE CASSANDRA:

Para obtener el tiempo de ejecución medio de las consultas en Apache Cassandra se utiliza la funcionalidad *tracing on* disponible en la consola *cqlsh* de *DataStax Apache Cassandra*. Uno de los valores que devuelve esta opción es el tiempo de ejecución de la sentencia consultada, dado en microsegundos.

Los resultados obtenidos en el análisis de tiempos de ejecución para la base de datos *FILMS_DATA* de Apache Cassandra (Fig. 4.20) son en media bastante más elevados que los obtenidos para MongoDB. La consulta nº1 presenta un tiempo de ejecución medio considerablemente más elevado que el resto de consultas. Sin embargo, esta diferencia es importante porque se trata de una de las consultas más frecuentes sobre la base de datos,

por lo que puede ser una desventaja que, incluso utilizando índices para los campos afectados, la consulta presente un tiempo de ejecución muy por encima de el del resto. Debido a la consulta nº1, la media de los tiempos de ejecución para las consultas ejecutadas se eleva (69,85 ms) y es muy superior al resto de consultas. El tiempo de ejecución medio en las consultas de selección es de 100'5 ms y el de las consultas de escritura es de 8'57 ms.

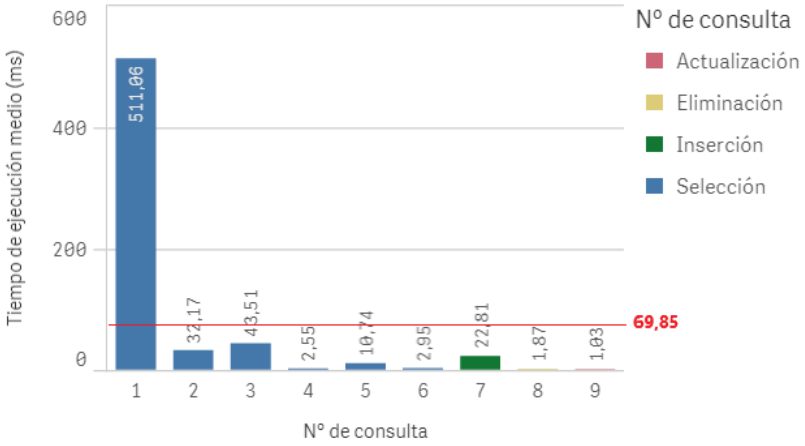


Figura 4.20: Gráfico del tiempo de ejecución medio por consulta de la base de datos *FILMS_DATA* de Apache Cassandra

NEO4J:

El tiempo de ejecución de las consultas en Neo4j se puede obtener añadiendo la expresión *PROFILE* antes de la consulta, devolviendo el valor en milisegundos (ms).

Los valores medios del tiempo de ejecución obtenidos para el grafo *FILMS_DATA* de Neo4j (fig 4.21) son muy dispares entre consultas. La media se queda en 175,44 ms, de manera que algunas consultas superan considerablemente la media y otras se quedan muy por debajo de ella. El tiempo de ejecución medio para las consultas de selección es de 193'73 ms y el de las consultas de escritura es 138'86 ms.

En general, las consultas que necesitan realizar operaciones de búsqueda entre diferentes nodos relacionados entre sí son los que presentan un tiempo de ejecución medio superior, es decir, analizando las sentencias consultadas se puede observar que el tiempo de ejecución medio es mayor cuando en una misma consulta intervienen varios nodos de diferente *label* relacionados entre sí.

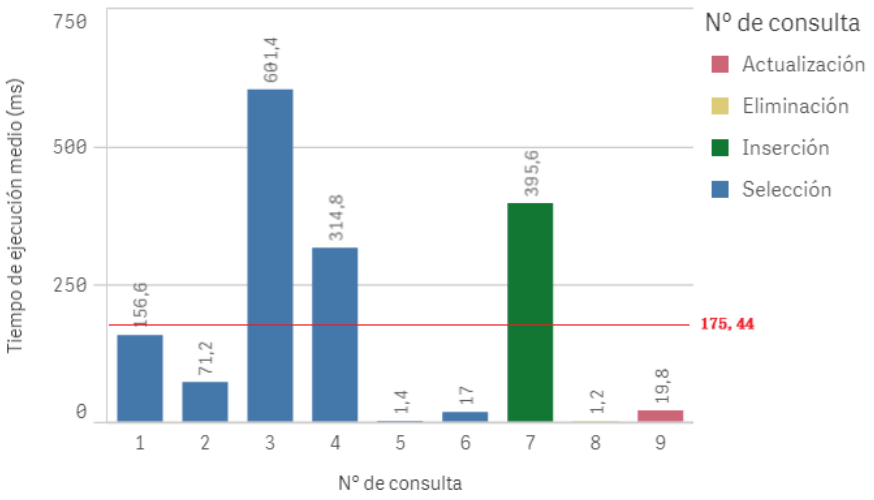


Figura 4.21: Gráfico del tiempo de ejecución medio por consulta del grafo *FILMS_DATA* de Neo4j

4.2.2.3. Comparativa global

El espacio de almacenamiento que las bases de datos ocupan son similares para MongoDB y Apache Cassandra, mientras que en la base de datos en Neo4j es más del 90 % mayor que cada una de las anteriores (Fig. 4.22). Por lo tanto, los mecanismos de gestión del almacenamiento en forma de grafo del SGBD Neo4j incrementan considerablemente el espacio de almacenamiento para el mismo conjunto de datos.

Es importante tener en cuenta la tendencia del almacenamiento ocupado por las tres bases de datos (Fig. 4.23). El espacio de almacenamiento que ocupa cada base de datos se representa, aproximadamente, con una recta creciente, donde la pendiente de la correspondiente a Neo4j es mayor que la de MongoDB y la de Apache Cassandra.

Esto implica que cuantos más datos se inserten en la base de datos, el tamaño de la correspondiente a Neo4j va a aumentar 9 veces más en proporción a la de MongoDB o la de Apache Cassandra. Si las dimensiones de la base de datos no son muy grandes (como es el caso del conjunto de datos sobre películas con el que se está trabajando), el tamaño no es un aspecto a tener en cuenta y la eficiencia de SGBD se evaluaría en función a los tiempos de ejecución medios de las consultas exclusivamente. Sin embargo, con conjuntos de datos grandes que van a verse incrementados en tamaño considerablemente conviene rechazar Neo4j como SGBD.

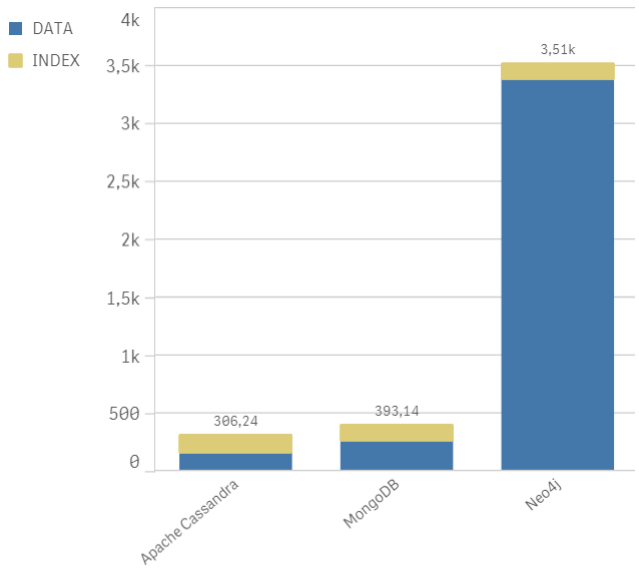


Figura 4.22: Distribución del espacio de almacenamiento de *FILMS_DATA* en cada SGBD

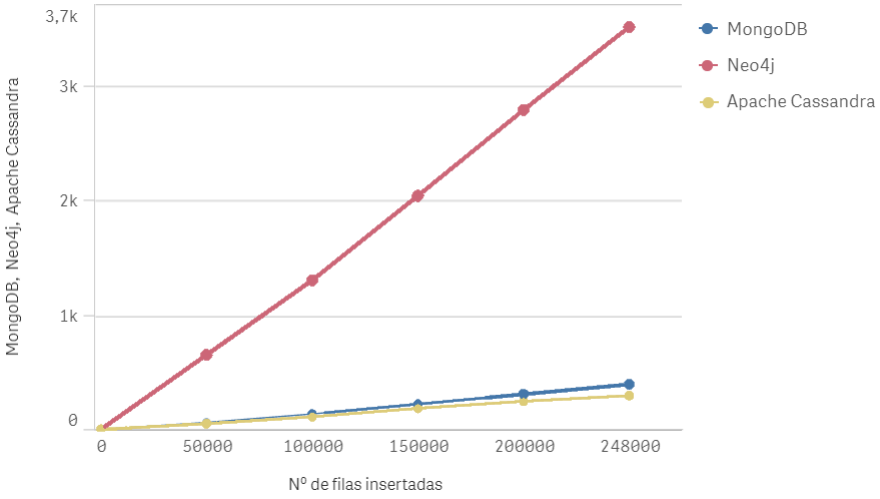


Figura 4.23: Representación gráfica del espacio de almacenamiento ocupado progresivamente cada 50.000 filas insertadas por SGBD para *FILMS_DATA*

Respecto al tiempo de ejecución medio de las diferentes consultas, el SGBD que menor coste de tiempo necesita es MongoDB, con 1'71 ms de tiempo medio utilizado para todas las consultas. En cuanto a Apache Cassandra, el tiempo medio de ejecución de todas las consultas es de 69'85 ms, y el de Neo4j es de 175'44 ms.

Por un lado, se muestran los valores del tiempo de ejecución medio por SGBD para cada consulta de selección (Fig. 4.24), donde se observa que el SGBD Neo4j obtiene tiempos

de ejecución muy elevados en todas las consultas de selección excepto en la nº 5.

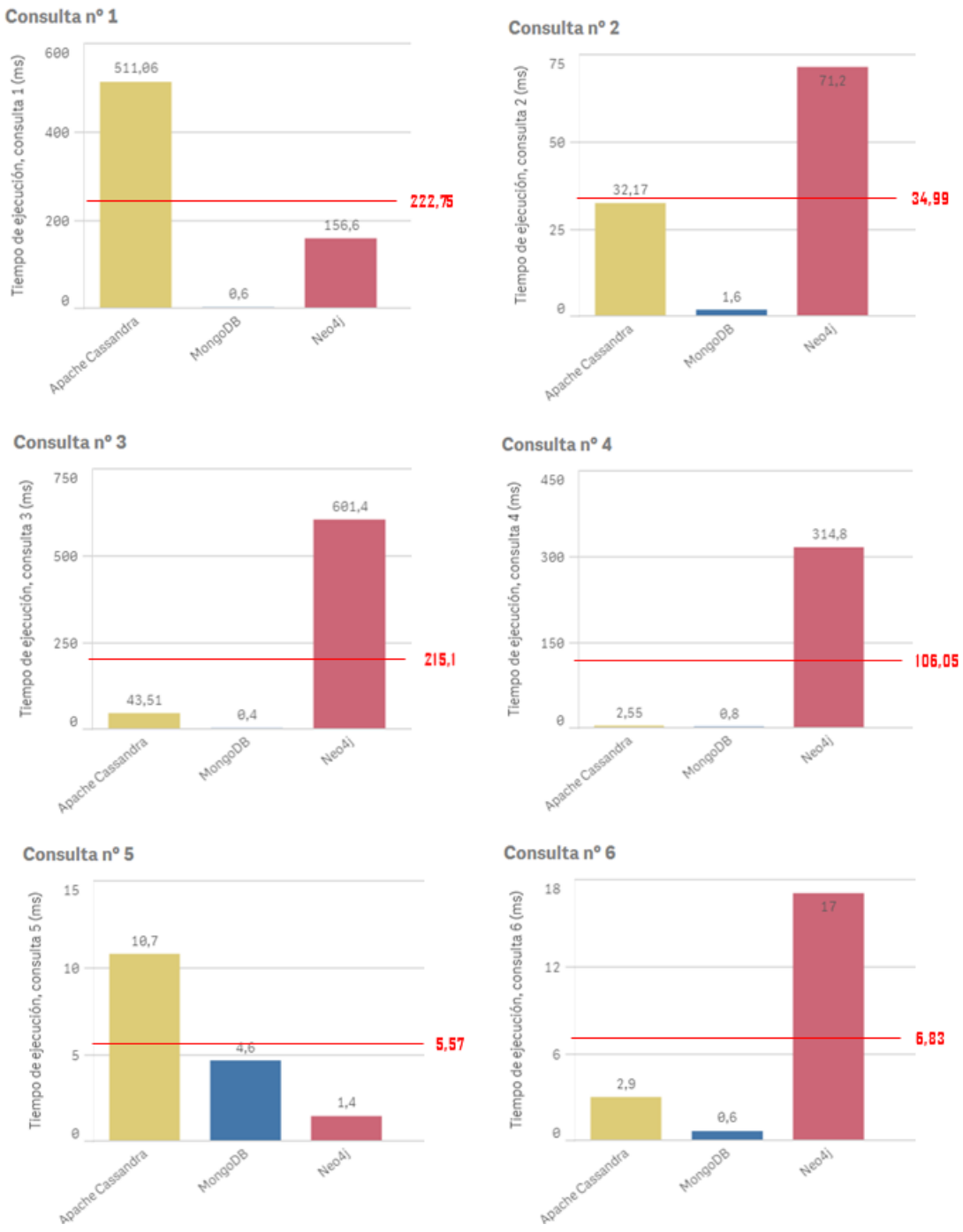


Figura 4.24: Gráficos del tiempo de ejecución medio de las consultas de selección por SGBD para *FILMS_DATA*

Por otro lado, se muestran los valores del tiempo de ejecución medio por SGBD para cada

consulta de inserción, eliminación y actualización (Fig. 4.25).

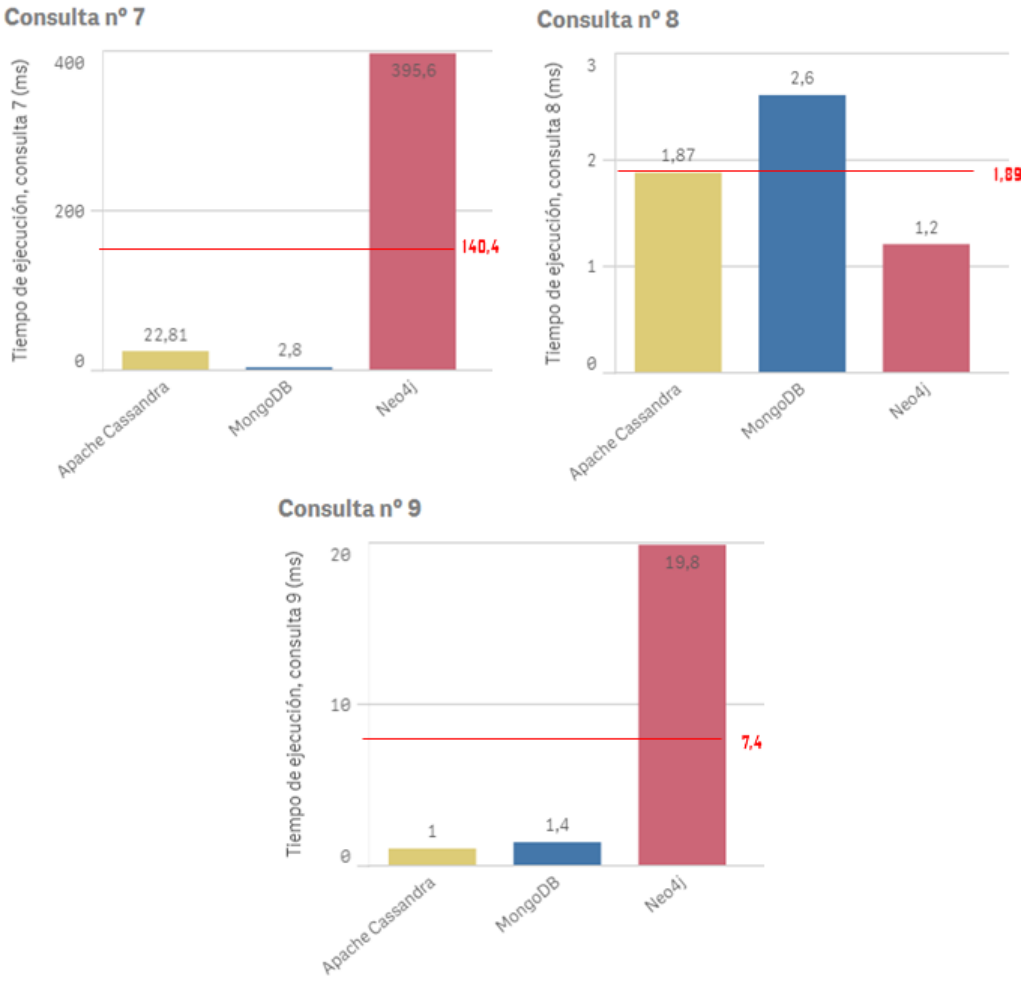


Figura 4.25: Gráficos del tiempo de ejecución medio de las consultas de inserción, eliminación y actualización por SGBD para *FILMS_DATA*

Para el escenario *FILMS_DATA* queda descartado Neo4j como SGBD más eficiente ya que es el que presenta peores resultados tanto en cuanto a espacio de almacenamiento ocupado por la base de datos como en tiempos de ejecución medios de las consultas. La base de datos de Apache Cassandra es la que menos espacio de almacenamiento en disco ocupa, pero no es indicativo de ser un SGBD más eficiente que MongoDB ya que la diferencia del espacio de almacenamiento entre ambas bases de datos es pequeña y poco significativa. Sin embargo, MongoDB destaca por los buenos resultados obtenidos en los tiempos de ejecución medios de las consultas planteadas. En la mayoría de las consultas es el SGBD que menores costes de tiempo ha reflejado, con una media muy por debajo del resto de SGBD.

Por lo tanto, el SGBD más eficiente para el conjunto de datos *FILMS_DATA* es MongoDB.

4.2.3. Conclusiones obtenidas de los diferentes SGBD

Hasta ahora se ha evaluado qué SGBD es más eficiente para cada uno de los dos escenarios planteados en base a los resultados obtenidos de las pruebas de ejecución. Por un lado, en el caso del escenario número 1 del conjunto de datos *TIME_SERIES_DATA* se ha determinado que tanto Apache Cassandra como MongoDB muestran unos resultados semejantes, siendo el primero ligeramente más eficiente en el aspecto del tiempo de ejecución medio de las consultas y el segundo respecto al espacio de almacenamiento. Por otro lado, para el escenario numero 2 del conjunto de datos *FILMS_DATA* se ha determinado que el SGBD MongoDB es el más apropiado por mostrar los resultados más positivos tanto en el aspecto del almacenamiento como en el del tiempo de ejecución.

La realidad de los diferentes SGBD analizados es que, en primer lugar, Neo4j es eficiente con conjuntos de datos donde hay expresa necesidad de relacionar diferentes elementos entre sí y donde la unión entre los elementos proporciona una visión mayor de los datos, priorizando así la eficiencia de las consultas con dicha estructura y no el espacio de almacenamiento que los datos ocupan. Algunos ejemplos de uso claros son la redes sociales y su forma de interconectar los elementos, la gestión de transacciones bancarias o el modo de generar recomendaciones en tiempo real a un tipo de perfil en contextos de grandes cantidades de datos. Estos tipos de escenarios en otros SGBD como Apache Cassandra o MongoDB supondrían una excesiva redundancia de datos, ya que estos SGBD no permiten realizar operaciones de *join* (disponen de alternativas que simulan el funcionamiento de *join* y similares, pero la recomendación general es utilizar este tipo de operaciones lo menos posible debido al coste computacional que requiere), lo cual conllevaría a un coste en almacenamiento elevado.

Por lo tanto, Neo4j como SGBD no es apropiado para almacenar un conjunto de datos de series temporales como con el que se está trabajando ni para un conjunto de datos como el de las películas, donde la relación entre los elementos no es un factor primordial y donde la redundancia de los datos, en este caso, no supone un coste excesivo.

En segundo lugar, Apache Cassandra es el SGBD más restrictivo en cuanto al formato en el que los datos son almacenados en las tablas, donde cada fila de datos debe presentar uno o varios campos comunes que en conjunto no se pueden repetir (formando la *Primary Key*), y que deben ser el elemento de filtrado principal de los datos en esa tabla. Para el

caso del conjunto de datos en forma de serie temporal es un SGBD apropiado cuando las consultas que se le van a plantear a la base de datos no presentan gran complejidad, es decir, cuando las consultas no requieren de relaciones entre diferentes tablas. Para el conjunto de datos sobre películas se ha visto reflejado que la redundancia de los datos ha perjudicado el rendimiento de la base de datos y, por lo tanto, no ha sido el SGBD más apropiado.

Y en tercer lugar, el SGBD MongoDB permite almacenar todo tipo de datos siguiendo cualquier esquema en sus documentos. Destaca por la eficiencia de la utilización de índices. De los tres SGBD analizados es el que mejores resultados ha mostrado frente al filtrado de documentos por un campo con un índice creado. La libertad en el diseño que proporciona MongoDB permite diseñar esquemas donde el campo de filtrado esté acompañado por un índice y así favorecer en gran medida los costes de tiempo de las consultas que se le plantean.

En el caso del conjunto de datos en forma de serie temporal es importante identificar las prioridades para diseñar el esquema más apropiado a los requisitos, ya que como se ha visto en este análisis es posible diseñar dos modelos distintos que presenten resultados muy diferentes. Si se genera un documento para cada marca de tiempo es posible generar un índice para ese campo y aprovechar así la eficiencia de los índices en MongoDB para agilizar el procesamiento de las consultas. MongoDB recomienda que en una consulta se vean implicados el menor número de documentos posibles, por lo tanto la generación de un documento por marca de tiempo en espacios temporales muy grandes podría ir en contra de las indicaciones de MongoDB. Sin embargo, existen otras alternativas como la agrupación de valores en base a una marca de tiempo "en punto", como puede ser un minuto exacto o una hora exacta (ejemplo representado en la versión 1 de *TIME_SERIES_DATA*). Esto conlleva la necesidad de realizar operaciones de paso de documentos a *array*s y su posterior evaluación que conlleva un coste adicional que debe ser evaluado en función de los requisitos y la cantidad de datos con la que se va a trabajar.

Respecto al conjunto de datos sobre películas en MongoDB, se ha planteado un diseño acorde al tipo de consultas que se presupone se van a realizar sobre la base de datos, que acompañadas de los índices oportunos ha permitido que MongoDB haya sido el SGBD más eficiente para este conjunto. En este caso ocurre algo similar que en el caso de las series temporales, donde es importante identificar requisitos y crear un buen diseño acompañado de índices.

5. CAPÍTULO

Desarrollo de la aplicación web

En este capítulo se explican las características y funcionalidades de la aplicación web desarrollada, así como los procesos llevados a cabo para la consecución de la aplicación final, desde aspectos relacionados con el diseño gráfico hasta características del servidor de la aplicación.

Por un lado, se especifican los requisitos de la aplicación que van a permitir identificar el alcance de ésta y, posteriormente, una serie de funcionalidades que la aplicación web debe tener implementadas. Por otro lado, una vez están los casos de uso identificados, se procede a desarrollar la parte *front-end* y *back-end* que permitan implementar las diferentes funcionalidades.

5.1. Análisis de requisitos

En primer lugar se describen las características de la aplicación web identificando los procesos que el usuario puede realizar y, en segundo lugar, se construyen los casos de uso asociados a dichas funcionalidades, detallando los flujos de eventos de cada uno de los casos de uso identificados.

5.1.1. Identificación y descripción de las características y funcionalidades de la web

La aplicación web tiene como objetivo el acercamiento de la tecnología *NoSQL* a los usuarios que accedan a ella, proporcionando tanto contenido didáctico como metodologías para la evaluación posterior del contenido aprendido.

Entre el contenido de la aplicación web destaca el acceso a noticias y contenido multimedia en torno a *NoSQL*, un *quizz* con preguntas sobre la misma temática, un ranking de puntuaciones para dicho *quizz*, la consulta de la actividad y progresos en la aplicación por parte del usuario, etc.

Se diferencian varios apartados accesibles por los usuarios que se restringen en función del tipo de usuario que accede a la aplicación web. De este modo, hay dos tipos de usuarios, por un lado está el usuario no autenticado y por otro el usuario autenticado que dispone de unas credenciales asociadas.

El usuario no autenticado tiene acceso al *home* de la aplicación, y puede acceder a las noticias publicadas y consultarlas en profundidad, observar el ranking de puntuaciones del *quizz* y ordenarlo en base a diferentes campos, ver la información de la web y mandar mensajes de contacto. Para pasar a ser un usuario autenticado, el usuario no autenticado puede realizar el proceso de *login* y pasar a ser un usuario autenticado si el sistema verifica correctamente las credenciales introducidas

El usuario autenticado (el que ha realizado la operación de inicio de sesión y ha sido verificado con sus credenciales por el sistema), además de realizar las operaciones que puede realizar el usuario no autenticado, puede realizar otra serie de operaciones asociadas a sus credenciales.

En primer lugar, puede consultar los datos de su perfil que están almacenados en la base de datos, pudiendo realizar cambios sobre ellos. Estos cambios son verificados antes de ser ingresados en la base de datos tanto en la parte del cliente como por el propio sistema tras la verificación en el primero. Con ambas verificaciones, el sistema actualiza la información proporcionada por el usuario. Un usuario autenticado debe poder guardar en su perfil información como su alias, correo electrónico, nombre y apellido, edad, número de teléfono y dirección.

En segundo lugar, el usuario autenticado puede consultar sus progresos en la aplicación web observando las acciones que ha desarrollado en ella y los avances que ha tenido en

las diferentes actividades.

En tercer lugar, puede observar los ficheros para descargar disponibles en la aplicación, así como descargarlos directamente desde el mismo lugar. El tipo de archivos que el usuario autenticado puede descargar son sistemas gestores de bases de datos, clientes gráficos para esos sistemas, *datasets* con datos para generar bases de datos, máquinas virtuales preparadas con un entorno *NoSQL* y otro tipo de contenido relacionado.

En cuarto lugar, puede acceder al *quizz* de la aplicación eligiendo una de las temáticas para participar respondiendo una serie de preguntas asociadas. Las preguntas son de 4 respuestas de las cuales solamente una es correcta. Además, las preguntas pueden ir acompañadas de una imagen asociada a la pregunta.

En quinto lugar, puede ver las acciones que ha desarrollado en la aplicación consultando los registros de su actividad de manera general. Estos registros se pueden filtrar en base a diferentes campos. El usuario debe saber qué acción ha realizado y la fecha en la que lo hizo.

Por último, el usuario autenticado puede consultar la información de su cuenta, pudiendo eliminarla de la base de datos. Además, como usuario autenticado puede cerrar su sesión para pasar a ser un usuario no autenticado.

5.1.2. Estructura de los casos de uso

Con una descripción consistente de las características y funcionalidades de la web, se procede a identificar los casos de uso concretos de la aplicación web. Los actores que toman parte en el flujo de actividad de la aplicación web son el usuario no autenticado, el usuario autenticado y el sistema, entendiendo como sistema la agrupación de procesos ajenos a las acciones directamente asociadas a un usuario, ya sea autenticado o no autenticado.

En el diagrama de casos de uso representado en la figura 5.1 se muestran los casos de uso asociados a los diferentes tipos de actores que pueden participar en la aplicación web, priorizando las actividades del usuario no autenticado y del usuario autenticado.

A continuación se describen los casos de uso identificados detallando en profundidad el caso de uso *Login*.

- **Login**: Al pulsar sobre el apartado de *login* desde el menú se accede a la página de inicio de sesión donde el usuario puede introducir sus credenciales para autenticarse (fig. 5.2).

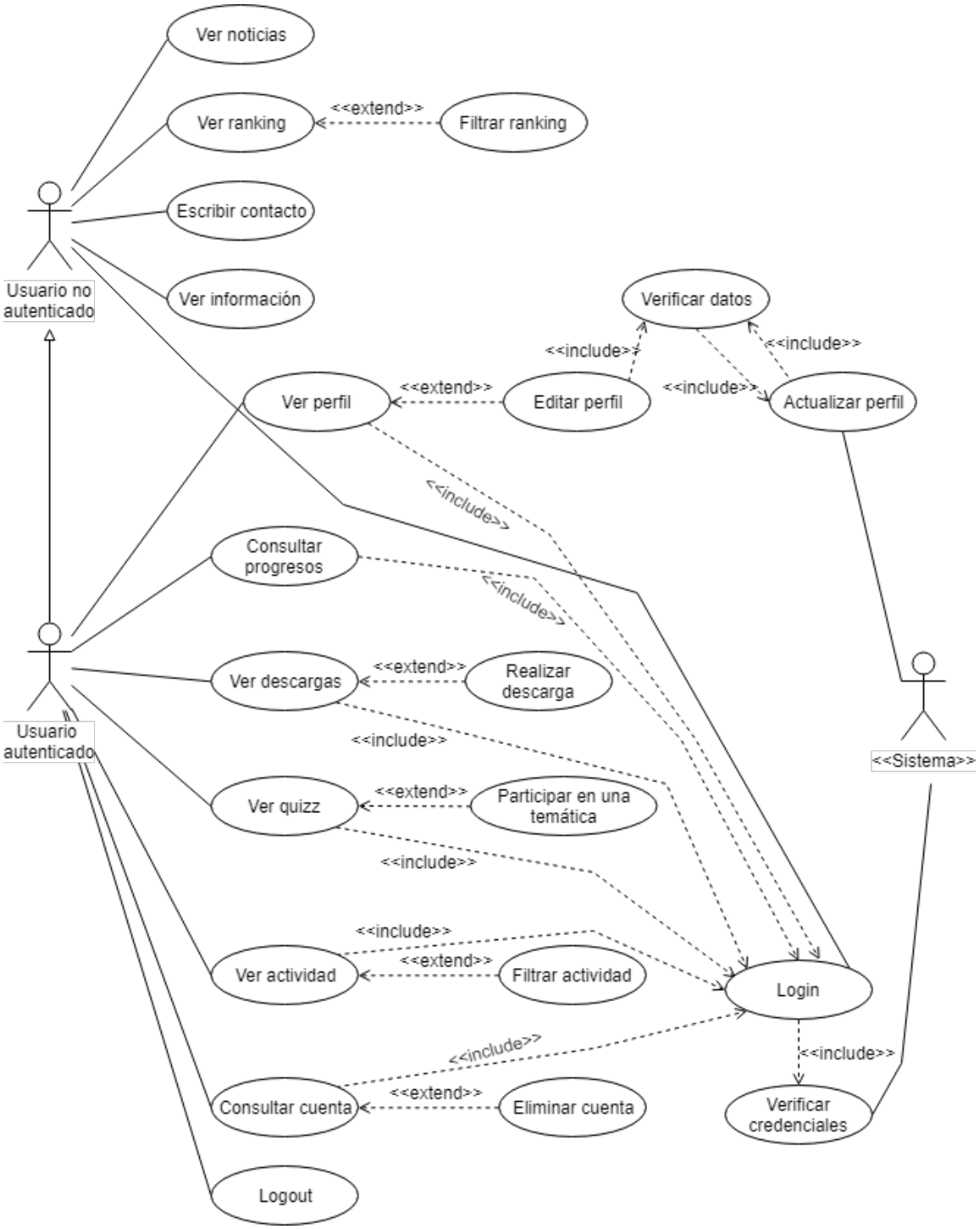


Figura 5.1: Diagrama de casos de uso de la aplicación web

Flujo de eventos:

1. El usuario accede a la página de *login*.
2. Si el usuario dispone de credenciales, las introduce en los campos de email y contraseña y pulsa en iniciar sesión para autenticarse. Si el usuario no dispone de credenciales, pulsa en registrarse para acceder a una interfaz de registro, donde el usuario introduce su correo electrónico y contraseña para registrarse.
3. En caso de iniciar sesión, el sistema verifica las credenciales y da acceso al usuario autenticado, o en su defecto, le muestra los errores obtenidos en el inicio de sesión. En caso de registrarse, el sistema verifica que los campos son correctos, registra al nuevo usuario y da acceso al usuario autenticado.

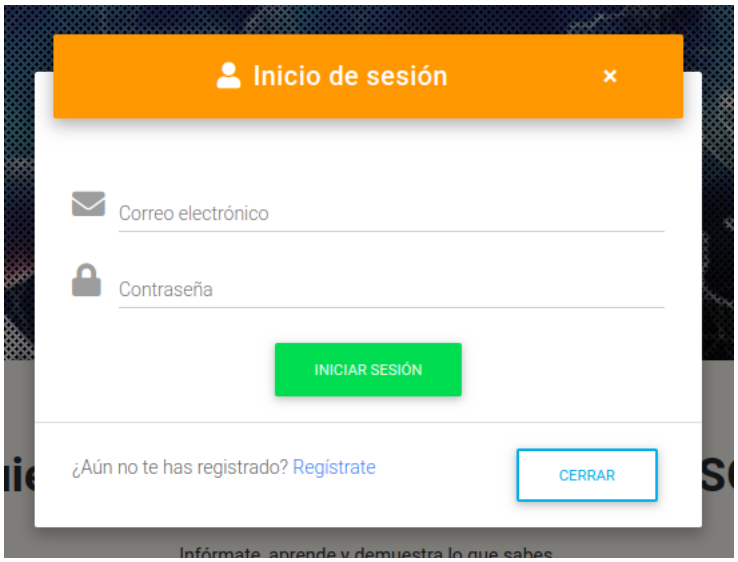


Figura 5.2: Captura del modal de *login* de la aplicación web

- **Ver noticias**: Al pulsar sobre el apartado de noticias desde la barra de navegación se accede a la interfaz de noticias donde se puede ver una lista de noticias seleccionables. El acceso a la página de noticias genera un nuevo evento de actividad si el usuario está autenticado, de modo que el sistema almacena dicho evento en la base de datos.

- **Ver ranking**: Al pulsar sobre el apartado de ranking desde la barra de navegación se accede a la interfaz del ranking donde aparece una tabla con la clasificación de las puntuaciones obtenidas en el *quizz* por todos los usuarios autenticados. El

acceso a la página de ranking genera un nuevo evento de actividad si el usuario está autenticado, de modo que el sistema almacena dicho evento en la base de datos.

- **Escribir contacto**: Al pulsar sobre el apartado de contacto desde la barra de navegación se accede a la página de contacto donde se puede rellenar un formulario y enviar un mensaje. El acceso a la página de contacto genera un nuevo evento de actividad si el usuario está autenticado, de modo que el sistema almacena dicho evento en la base de datos.

- **Ver información**: Al pulsar sobre el apartado de información desde la barra de navegación se accede a la página de información donde se muestra una fragmento de texto con información sobre la aplicación web. El acceso a la página de información genera un nuevo evento de actividad si el usuario está autenticado, de modo que el sistema almacena dicho evento en la base de datos.

- **Ver perfil**: Al pulsar sobre el apartado del perfil desde el menú se accede a la página de perfil donde se muestra los datos de perfil del usuario autenticado almacenados en la base de datos.

- **Consultar progresos**: Al pulsar sobre el apartado de progresos desde el menú se accede a la página de progresos donde se muestran los avances del usuario autenticado en la aplicación web en base a las acciones y los eventos que han sido registrados. El acceso a la página de progresos genera un nuevo evento de actividad, de modo que el sistema almacena dicho evento en la base de datos.

- **Ver descargas**: Al pulsar sobre el apartado de descargas desde el menú se accede a la página de descargas donde se muestran los archivos disponibles para descargar por el usuario autenticado. El usuario puede descargar dichos archivos. La descarga de un archivo genera un nuevo evento de descarga, de modo que el sistema almacena dicho evento en la base de datos.

- **Ver quizz**: Al pulsar sobre el apartado de *quizz* desde el menú se accede a la página de *quizz* donde se muestran los cuestionarios disponibles en base a diferentes temáticas. El acceso a uno de los cuestionarios genera un nuevo evento de *quizz*, de modo que el sistema almacena dicho evento en la base de datos.

- **Ver actividad**: Al pulsar sobre el apartado de actividad desde el menú se accede a la página de actividad donde se muestra una tabla con las acciones y eventos registrados por el usuario autenticado y la fecha del registro. El acceso a la página

de actividad genera un nuevo evento de actividad, de modo que el sistema almacena dicho evento en la base de datos.

- **Consultar cuenta**: Al pulsar sobre el apartado de cuenta desde la barra de navegación se accede a la página de cuenta donde se muestra información sobre el estado de la cuenta del usuario autenticado y se da la posibilidad de eliminar la cuenta definitivamente.
- **Logout**: Al pulsar sobre el apartado de *logout* desde la barra de navegación el sistema cierra la sesión del usuario, que pasa a ser un usuario no autenticado.

5.2. Arquitectura

En este apartado se describe la arquitectura que presenta la aplicación web generada, relacionando todas las partes que la componen.

En la figura 5.3 se muestra la representación gráfica de la arquitectura del sistema web a través de un diagrama con los diferentes módulos que lo constituyen.

Por una lado, resaltado en azul aparecen los servicios externos utilizados en la aplicación web y que han sido configurados en el desarrollo de este TFG para adaptarlo a las necesidades de la aplicación web. Por otro lado, en negro se muestra el contenido desarrollado completamente para este TFG.

WEB APP

Representa la aplicación web desarrollada en este TFG.

El módulo *User Interface* hace referencia a las diferentes interfaces gráficas que componen la aplicación web y a las que el usuario final puede acceder.

En el módulo *Business Logic* se encuentran implementadas las diferentes funcionalidades de la aplicación web a través de llamadas al servidor. Este módulo conecta con los servicios que abastecen a la aplicación y prepara los datos para ser utilizados directamente por la aplicación web. La principal función de este módulo es preparar los datos obtenidos de la aplicación web y enviarlos al servicio correspondiente, así como obtener los datos del servidor y adaptarlos a las características de la aplicación web, es decir, gestiona el acceso a los datos a través de inserciones y consultas de datos en el servidor.

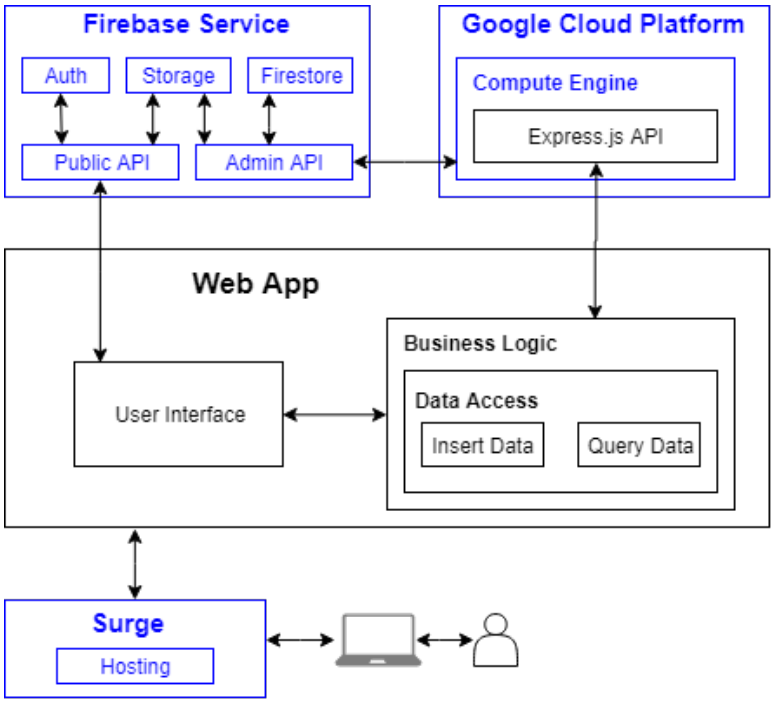


Figura 5.3: Diagrama de la arquitectura del entorno web

FIREBASE SERVICE

Representa el conjunto de servicios que la plataforma Firebase ofrece y que se utilizan para este proyecto. Los servicios utilizados son el de autenticación, almacenamiento y base de datos (*Firestore*). El acceso a estos servicios se proporciona de dos modos.

Por un lado, para el acceso desde la parte cliente de la aplicación, Firebase dispone de *Firebase JavaScript SDK*[1], el kit de desarrollo de Firebase con las librerías necesarias que la parte cliente de una aplicación necesita para utilizar sus servicios. *Public API* hace referencia a las diferentes API utilizadas en este proyecto a través de *Firebase JavaScript SDK*.

Por otro lado, con *Firebase Admin SDK*[2] se proporciona un kit de desarrollo software dedicado a servidores que soporta *Node.js*, *Java*, *Python* y *Go*. Este SDK ofrece funcionalidades más privilegiadas al servidor y es utilizado en el servidor *Express.js* generado en este TFG. *Admin API* hace referencia a las diferentes API utilizadas en este proyecto a través de *Firebase Admin SDK*.

[1] *Firebase JavaScript SDK*: https://firebase.google.com/docs/reference/js
[2] *Firebase Admin SDK*: https://firebase.google.com/docs/reference/admin

GOOGLE CLOUD PLATFORM

Representa el conjunto de servicios de la plataforma *Google Cloud* que se utilizan en este TFG. En este caso se utiliza la funcionalidad de *Compute Engine*, es decir, una máquina virtual alojada en esta plataforma. *Firebase* y *Google Cloud Platform* comparten muchos servicios por lo que no es del todo correcta la separación indicada. Sin embargo, se ha colocado cada servicio desde la plataforma sobre la que se está utilizando.

Compute Engine[3] es un tipo de infraestructura como servicio (*IaaS*) que permite crear máquinas virtuales escalables de alto rendimiento. En este TFG se ha generado una máquina virtual con el sistema Operativo Debian 9[4], donde se ha preparado un servicio a modo de API utilizando *Express.js* (explicado en profundidad en el apartado 5.3.2).

SURGE

Representa el servicio de *hosting* donde está alojada la parte cliente de la aplicación web. *Surge* proporciona un nombre de dominio gratuito que permite acceder a la aplicación web de manera *online*.

5.2.1. Diseño de la base de datos y almacenamiento

El sistema de bases de datos utilizado por la aplicación es el proporcionado por *Firebase*, que se trata de un gestor de bases de datos orientado a documentos, similar al que gestiona *MongoDB*. A través del diagrama de la figura 5.4 se muestra una representación esquemática de la estructura de las bases de datos, donde cada recuadro es una colección y cada fila un campo de un documento de dicha colección.

Por otra parte, el sistema de almacenamiento debe tener una estructura que permita almacenar las imágenes y contenido de cada usuario, los ficheros a descargar por los usuarios y contenido público adicional para la aplicación. En la figura 5.5 se muestra la representación jerárquica del diseño del sistema de almacenamiento utilizado en la aplicación.

Cada recuadro representa un directorio que almacena otros directorios, excepto los de más bajo nivel que son los directorios donde se almacenan los ficheros:

- **Default**: Almacena ficheros públicos que utiliza la aplicación web, como imágenes y logos.

[3] *Compute Engine*: https://cloud.google.com/compute/
[4] Debian: https://www.debian.org/

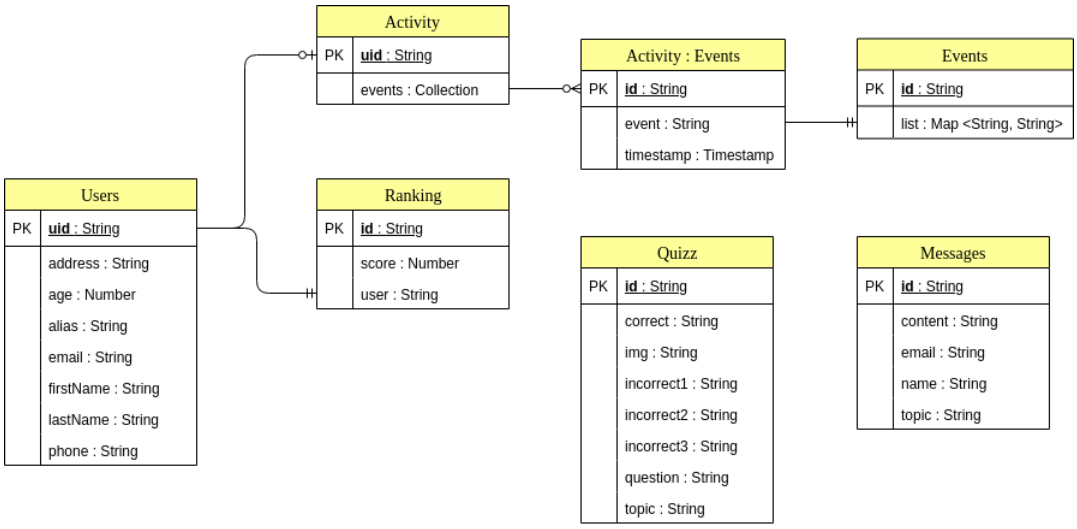


Figura 5.4: Diseño de la base de datos *Firebase* utilizada en la aplicación

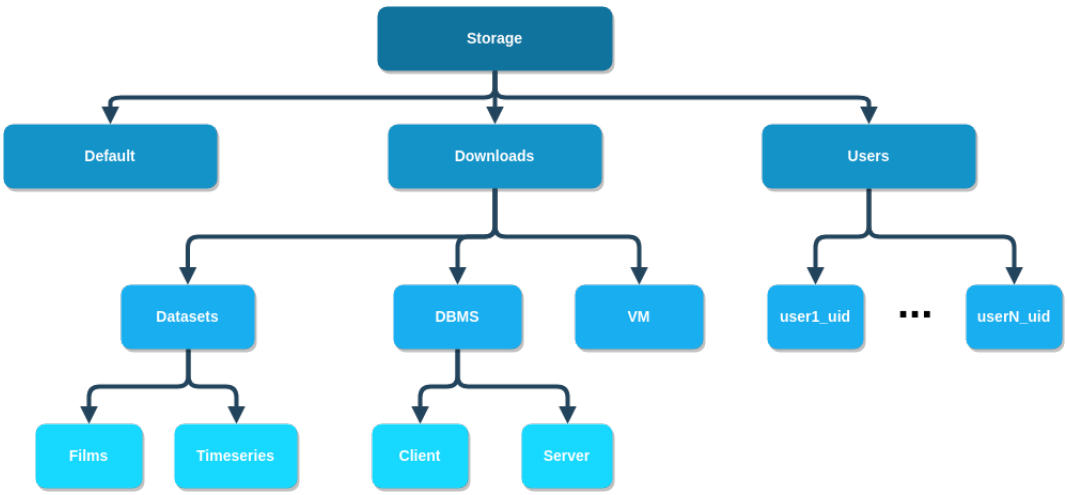


Figura 5.5: Diseño del sistema de almacenamiento *Storage* utilizado en la aplicación

- **Downloads**: Almacena los archivos descargables por el usuario autenticado. Estos archivos se agrupan en *datasets* (conjuntos de datos) que a su vez se dividen en *datasets* de películas y de series temporales, *DBMS* (sistemas de bases de datos) que a su vez se dividen en los clientes gráficos y los propios sistemas servidores, y *VM* (máquinas virtuales).
- **Users**: Almacena el contenido asociado a los usuarios autenticados. Hay un directorio por cada usuario, donde el nombre del directorio es el *uid* (identificador de usuario en la aplicación) del usuario. Estos directorios almacenan contenido como imágenes, fotos de perfil, etc.

5.3. Diseño e implementación de la aplicación

En este apartado se exponen las características del diseño y de la implementación del conjunto de la aplicación web, tanto del *front-end* como del *back-end*.

5.3.1. Creación de la aplicación con *Vue.js*

Una vista y un componente de *Vue.js* se estructuran en tres partes (código 5.1):

- **Plantilla**: Se incluyen todos los elementos *html* y componentes *Vue.js* de la vista.
- ***Script***: Se incluyen las propiedades, métodos y procesos de inicialización de la vista.
- **Estilo**: Se incluyen las propiedades CSS de la vista.

La vista principal se denomina *App.vue*, que se trata del primer componente que se inicializa y desde donde se cargan el resto de vistas y componentes. Para que esto sea así hay que indicarlo en el fichero *main.js*, donde se señala que será el contenedor con id igual a "app" donde se van a renderizar el resto de componentes de la aplicación (código 5.2).

Para acceder a los servicios de *Firebase* de autenticación y almacenamiento desde la aplicación se utilizan las APIs de JavaScript de *Firebase*. Por otro lado, para acceder al servicio generado en *Express.js* (apartado 5.3.2) se genera una API interna que gestiona las llamadas al servicio generado. Para implementar estas conexiones a los diferentes servicios se ha generado un esquema de ficheros JavaScript que facilitan el acceso a las

```
<template>
  <div ref="app" id="app">
    <elem1></elem1>
  </div>
</template>

<script>
    import * from '***';

    export default {
      name: "App", components: { },  data() { return { } },
      beforeCreate() { }, mounted() { }, methods: { }
    };
</script>

<style>
</style>
```

Código 5.1: Esquema simplificado del fichero *App.vue* de la aplicación

```
import Vue from 'vue'
import App from './App' //componente App.vue
import router from './router' //incluye enrutamiento

new Vue({
    el: '#app',
    router,
    components: { App },
    template: '<App/>'
});
```

Código 5.2: Inicialización del proyecto *Vue.js* a través del fichero *main.js*

diferentes funcionalidades. La configuración del fichero asociado a *Firebase* se explica detalladamente en el apartado 5.3.3. Para la conexión entre el módulo de *Express.js* y la aplicación se ha generado un fichero *Api.js* que se encarga de crear la conexión con la IP y puerto donde se encuentra el servicio utilizando el módulo de *axios* (código 5.3)

De este modo, los ficheros *DefaultServices.js* y *UserServices.js* utilizan la conexión creada para conectarse con el servidor remoto y obtener vía *POST* la información que éste le devuelve (código 5.4). Cada uno de estos dos ficheros contiene los métodos que el cliente de la aplicación va a utilizar para obtener la información contenida en el servidor remoto de *Express.js* y, además, se encargan de transformar los datos a un formato que tanto el cliente como el servidor puedan entender.

Vue.js proporciona una serie de directivas que se colocan delante de los atributos de los

```
import axios from 'axios'

export default () => {
    return axios.create({
        baseURL: 'http://remoteIp:5000'
    })
}
```

Código 5.3: Conexión vía *axios* con el servicio *Express.js* desde *Api.js*

```
import Api from '@/services/Api' //Api.js con la conexión axios

export default {
  async method1(data) {
    var result = await Api().post('remoteMethod', data)
    return result.data
  }
 }
```

Código 5.4: Estructura de los ficheros *DefaultServices.js* y *UserServices.js*

elementos que componen la plantilla (<*template*>) de un componente, y que permiten controlar eventos y dar funcionalidad a los componentes de una vista. Algunas de estas directivas son *v-on*, *v-bind*, *v-model*, *v-for*, *v-show* o *v-if*[5].

Es posible generar componentes externos personalizados y utilizarlos en una vista. Para personalizar los componentes dentro de una vista, éstos especifican una serie de propiedades que son personalizadas desde la propia vista a través de los atributos del componente. Los componentes externos pueden ser reutilizados en diferentes vistas y, gracias a las propiedades, puede personalizarse de diferentes formas en cada una de la vistas a pesar de tratarse del mismo componente. En el fragmento de código 5.5 se muestra un ejemplo concreto simplificado de un componente externo que se utiliza en la aplicación desarrollada.

[5]Directivas Vue.js: https://vuejs.org/v2/api/#Directives

```
<template>
  <header>
    <div class="mask pattern-8 flex-center page-title">
      <div class="white-text text-center wow fadeInUp"><h1>{{title}}</h1></div>
    </div>
  </header>
</template>

<script>
  export default {
    props:{
      title:{ type:String, default:"NoSQL" }
    }
  }
</script>

<style>
  header {  /* Estilos para una cabecera */   }
</style>
```

Código 5.5: Ejemplo de un componente externo

En el código 5.6 se muestra un ejemplo concreto de la renderización del componente creado en el fragmento de código 5.5 dentro de una vista, y de la utilización de diferentes directivas, así como el uso los servicios remotos.

Un componente importante en el desarrollo de esta aplicación web es el formulario, ya que se utiliza en el proceso de inicio de sesión, registro, contacto y de edición de perfil. Los campos utilizados en un formulario son proporcionados por la librería *MDBVue*. La manera en la que los datos introducidos en los campos del formulario son transmitidos a la aplicación es a través de la directiva *v-model*, que permite enlazar datos desde el componente a la interfaz de usuario y viceversa. Esta directiva referencia a un elemento dentro de la propiedad *data* del componente, por lo que cada vez que el usuario modifica dicho campo, la propiedad del componente adquiere el valor actualizado y es posible procesarlo.

```
<input type="text" id="phone" class="form-control" v-model="phone">
```

En el contexto de los formularios otra directiva importante es *v-on* acompañando al atributo *click* de la etiqueta *button* que se encarga de procesar el contenido introducido en el formulario. La directiva *v-on:click="checkForm"* permite ejecutar el método *checkForm()* al pulsar sobre el elemento que contiene el atributo. Éste método obtiene los valores introducidos por el usuario en la aplicación y los procesa en base a unas condiciones

```
<template>
    <div class="view_content">
        <headerComp v-bind:title="title"></headerComp>
        <mdb-datatable :data="data" striped bordered order="[3, 'dec']" />
    </div>
</template>
<script>
    import headerComp from '../element/Header' //componente nuevo
    import DefaultServices from '../../services/DefaultServices'
    export default {
      name: 'Raking',
      components: {
        headerComp
      },
      data() {
        return {
          title: 'Ranking', //renderizado en atributo title, directiva bind
          data: ''
        }
      },
      methods: {
        async getRanking() {
          var result = await DefaultServices.getRanking(); //servicio remoto
          this.data = result;
        }
      },
      beforeCreate() {
        this.getRanking();
      }
    }
</script>
```

Código 5.6: Ejemplo de una vista *Vue.js* con directivas, componentes renderizados y métodos

predefinidas. Se muestra un ejemplo reducido de un formulario de la aplicación en el fragmento de código 5.7.

```
<template>
  <div class="view-content">
    <form class="text-center border border-light p-5"
    enctype="multipart/form-data">
        <label for="alias" class="grey-text left">Alias</label>
        <input type="text" id="alias" class="form-control"
        v-model="profile.alias">
        <button class="btn btn-orange my-4 btn-block" type="button"
        v-on:click="checkForm">Guardar cambios</button>
    </form>
  </div>
</template>

<script>
    ...
  data() {
    return {
      profile: {
        alias: ''
      },
      errors: []
    }
  },
  methods: {
    async checkForm(e) {
      this.errors = [];
      if (!this.profile.alias) {
        this.errors.push('Debe introducir un alias.');
      } else if (this.profile.alias.length < 4) {
        this.errors.push('El alias debe contener al menos 4 caracteres.');
      }
    }
  }
</script>
```

Código 5.7: Ejemplo de un formulario simplificado concreto de la aplicación

5.3.2. Creación del servicio con Express.js

El servicio a generar debe tener la estructura de una API a la que un cliente realiza peticiones. Por lo tanto, el esquema del fichero *app.js* es una lista de registros de respuesta a peticiones *POST* donde se recogen los valores recibidos y se envían los valores solicitados, como en el siguiente fragmento:

```
app.post('/', function (req, res) {
    var data = req.body.data;
    res.send({
        message: "Your data is: " + data
        });
  });
```

Al comienzo del fichero *app.js* se importan todos los paquetes preinstalados (mencionados en el anexo D.3) y se crea una instancia de *Express.js*. Para incluir los módulos importados en la aplicación *express* generada hay que especificar que dicha aplicación debe usar esos módulos.

```
var app = express();
app.use(morgan('combine'));
app.use(bodyParser.json());
app.use(cors());
```

Se debe predefinir un puerto por el que el servidor va a estar escuchando las distintas peticiones que le lleguen. Para especificar esta propiedad se dice a la aplicación *Express.js* que escuche por el puerto por defecto o, en su defecto, por el 5000. De este modo el cliente debe acceder al puerto 5000 para acceder a los servicios del servidor.

```
app.listen(process.env.PORT || 5000);
```

5.3.3. Configuración de Firebase

Firebase ofrece varios servicios a través de distintas APIs asociadas a cada servicio. Además, diferencia entre las APIs de JavaScript o públicas y las APIs Admin o privadas. Estas últimas precisan de una clave de cuenta para poder utilizarlas y pueden ser utilizadas en servidores *Node.js*, *Java*, *Python* o *Go*.

Para utilizar los servicios de autenticación y almacenamiento desde la parte cliente de la aplicación se debe instalar el paquete de *Firebase*.

```
$ npm install firebase --save # Firebase
```

Es necesario añadir una nueva aplicación web al proyecto de *Firebase* desde la consola. Para la nueva aplicación creada se genera automáticamente unos parámetros de configuración que se deben incluir en la aplicación web de *Vue.js*. De este modo, se genera un

fichero *Firebase.js* con el contenido indicado en el fragmento de código 5.8. Este fichero incluye los parámetros de configuración previamente mencionados y la inicialización de la aplicación de *Firebase* utilizando dichos parámetros. Con la aplicación inicializada, es posible acceder a los servicios ofrecidos, que en este caso son los de autenticación (*auth()*) y almacenamiento (*storage()*), que son exportados para poder ser llamados y utilizados desde otras vistas de la aplicación.

```
import firebase from 'firebase'

const firebaseConfig = {
  apiKey: "****",
  authDomain: "***.firebaseapp.com",
  databaseURL: "https://***.firebaseio.com",
  projectId: "NoSQL",
  storageBucket: "***.appspot.com",
  messagingSenderId: "***",
  appId: ****"
};
firebase.initializeApp(firebaseConfig);

export const auth = firebase.auth();
export const storage = firebase.storage();
```

Código 5.8: Fichero *Firebase.js* con la configuración para la parte cliente de la aplicación

Cada vez que se quiera utilizar alguno de los servicios (a traves de su API) desde otro componente *vue*, solamente hay que importar dichos servicios en el componente:

```
import { auth, storage } from '../../services/Firebase' //Firebase.js
```

Por otro lado, para poder utilizar los servicios de administrador de *Firebase* hay que instalar el paquete de *Firebase-Admin* en el servidor:

```
$ npm install firebase-admin --save # Firebase-Admin
```

El servidor debe iniciar la aplicación de *Firebase* con unas credenciales proporcionadas por una clave privada de la cuenta de servicio del proyecto de *Firebase* (cod. 5.9). Esta clave privada se obtiene desde el apartado de *cuentas de servicio* desde la consola de *Firebase*.

Para iniciar la aplicación de *Firebase*, se genera un nuevo fichero *Firebase.js* similar al generado para la parte cliente de la aplicación. Este fichero inicializa la aplicación de

```
{
  "type": "service_account",
  "project_id": "***",
  "private_key_id": "***",
  "private_key": "-----BEGIN PRIVATE KEY-----*****-----END PRIVATE KEY-----",
  "client_email": "****.gserviceaccount.com",
  "client_id": "***",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/
    firebase-adminsdk-tn6cs***serviceaccount.com"
}
```

Código 5.9: Fichero *ServiceAccountKey.json* con la clave privada para el uso de la *Admin API* de *Firebase*

Firebase con la clave privada previamente obtenida y exporta los servicios de base de datos (*Firestore*) y almacenamiento que va a utilizar el servidor. En el fragmento de código 5.10 se muestra el contenido de este fichero.

```
const admin = require('firebase-admin');
var serviceAccount=require('./serviceAccountKey.json'); //serviceAccountKey.json
var path = require('path');

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  storageBucket: "***.appspot.com"
});

exports.db = admin.firestore();
exports.bucket = admin.storage().bucket();
```

Código 5.10: Fichero *Firebase.js* con la configuración para utilizar las APIs de *Firebase* en el servidor

Del mismo modo, la manera en la que el servidor utiliza los servicios (APIs) de *Firebase* como administrador es importando el fichero generando y accediendo a los servicios exportados:

```
import admin from './config//Firebase' //Firebase.js
var db = admin.db;
var storage = admin.storage;
```

Es importante configurar las reglas de seguridad para restringir el acceso tanto a la base de datos de *Firestore* como al servicio de almacenamiento. Desde la consola de *Firebase* es

posible generar reglas que referencian a rutas concretas dentro de la jerarquía del servicio y deniegan o permiten el acceso a dicha ruta si se cumplen una serie de requisitos preestablecidos. En el siguiente ejemplo se muestra una regla para el servicio de almacenamiento de *Firebase* utilizada en este proyecto en la que se permite el acceso a todo el contenido del directorio *default* para operaciones de lectura, y en la que se permite el acceso al resto de contenido del almacenamiento a todos los usuarios autenticados para operaciones de lectura y escritura:

```
service firebase.storage {
   match /b/{bucket}/o {
     match /default/{allPaths=**} {
         allow read;
     }
     match /{allPaths=**} {
       allow read, write: if request.auth != null;
     }
   }
 }
```

Generando reglas de restricciones se configura la base de datos para que las colecciones de *activity*, *quizz* y *users* solamente puedan ser accedidas con operaciones de lectura y escritura por usuarios autenticados. Además, la colección *messages* solo puede ser accedida por todos los usuarios para operaciones de escritura. Por otro lado, las reglas de restricciones en el servicio de almacenamiento garantizan que el directorio *default* es accesible por todos los usuarios para operaciones de lectura, el directorio *downloads* es accesible por los usuarios autenticados para operaciones de lectura, y el directorio *users* es accesible por los usuarios autenticados para operaciones de lectura y escritura, concretando que cada usuario puede acceder al subdirectorio que tenga el nombre de su identificador de usuario en el sistema, restringiendo el acceso a los directorios personales del resto de usuarios.

5.3.4. Configuración de Google Cloud Platform

La plataforma de *Google Cloud* se utiliza en este proyecto para generar una máquina virtual que albergue el servidor *Express.js* y a la que se pueda acceder desde un cliente externo.

Para generar una nueva instancia de una máquina virtual, desde la consola de GCP (*Google Cloud Platform*) se accede a *Compute Engine* pasando por el menú. En la nueva vista

aparece la opción de *crear nueva instancia*. En este punto hay que especificar una serie de parámetros de configuración de la máquina, como el nombre de la instancia, CPU o el sistema operativo con el que va a funcionar, que en este caso será *Debian GNU/Linux 9 (stretch)*. El resto de parámetros se mantienen por defecto.

Al iniciar la instancia vía *SSH* desde el panel de instancias generadas, se abre la consola arrancando la nueva máquina virtual. Hay que realizar el mismo proceso de configuración del entorno de desarrollo *Express.js* que el realizado en local (descrito en el anexo D.3). Una vez hecha la configuración, en la ventana de la consola hay un desplegable donde una de las opciones permite incluir ficheros externos a la máquina virtual, que es el proceso a seguir para añadir los ficheros del servidor local en la máquina.

Para poder acceder al servidor remotamente tal y como tiene que hacerlo el cliente de la aplicación construida, es necesario configurar las reglas de cortafuegos para permitir el acceso exterior a la máquina virtual por el puerto 5000, que es el puerto por el que está activo el servicio. La nueva regla debe indicar que admite la entrada desde cualquier rango de ip (0.0.0.0/0) por el puerto 5000 de TCP.

Para mantener el servicio activo aún terminando la conexión *SSH*, se debe ejecutar el servidor utilizando el comando *nohup* en segundo plano (&):

```
$ nohup npm start > /dev/null 2>&1 &
```

5.3.5. Publicación de la aplicación web

Con el servidor accesible en la web, hay que alojar la parte cliente de la aplicación en un host público. La herramienta *Surge* permite configurar a través de la consola el directorio estático de la aplicación y alojarlo bajo un nombre de su dominio de manera gratuita.

La instalación de *surge* se realiza de manera global en el sistema:

```
$ npm install surge --global
```

Desde el directorio *dist* (directorio con los ficheros estáticos públicos de la aplicación) se ejecuta el comando *surge*. En el proceso de configuración se solicitan algunos datos como correo electrónico, contraseña, directorio concreto de publicación y nombre de dominio. En ese momento los archivos del directorio *dist* están públicos bajo el dominio indicado.

Dado que la aplicación construida es de tipo SPA (*Single Page Aplication*), con la configuración realizada solamente es posible acceder a la raíz (*index.html*) de la aplicación o a una ruta de la que existe un fichero estático en el directorio raíz, en caso contrario (p. ej. */news*), se mostraría un error de página no encontrada. Por defecto, cuando una ruta no es encontrada se busca en el fichero *200.html*[6] y desde ahí se realiza el enrutamiento oportuno. De este modo, cada vez que se actualice el código de la aplicación y se quiera desplegar, se debe reconstruir la aplicación, cambiar el nombre del fichero *index.html* construido por defecto a *200.html* y finalmente desplegar el código. En el fichero *package.json* se añade una línea de comandos en al propiedad *scripts* que realice este proceso automáticamente.

```
"deploy": "npm run build && mv dist/index.html dist/200.html &&
cd dist && surge --domain aprendenosql.surge.sh --project /***/dist/"
```

De este modo, ejecutando el comando *npm run deploy* se actualiza el directorio público, se realizan las modificaciones oportunas y se despliega, pudiendo interactuar con el enrutamiento de la aplicación desplegada correctamente.

El nombre de dominio donde se aloja la aplicación web final es *http://aprendenosql.surge.sh*.

[6] *200.html*: https://surge.sh/help/adding-a-200-page-for-client-side-routing

6. CAPÍTULO

Verificación y pruebas

En este capítulo de detallan las principales pruebas realizadas a lo largo del desarrollo del proyecto. Estas pruebas han servido para identificar errores y debilidades en algunos puntos del proyecto, para después solventarlos adecuadamente.

Se recogen pruebas relacionadas con el apartado del análisis de rendimiento de los diferentes SGBD y el conjunto de datos con los que se ha trabajado, así como pruebas desarrolladas para la aplicación web generada, tanto pruebas intermedias como pruebas finales.

6.1. Pruebas del análisis de rendimiento

En la tabla 6.1 se identifican las pruebas realizadas en el desarrollo del análisis de rendimiento de diferentes sistemas *NoSQL* llevado a cabo en este proyecto.

Las pruebas realizadas son relacionadas con la generación de las diferentes bases de datos y la identificación de las consultas generadas con las que se ha realizado el propio análisis. La finalidad de estas pruebas es asegurar que se realiza un análisis correcto, demostrando que el conjunto de datos cumple con las condiciones requeridas, que las sentencias analizadas dan los resultados esperados, y que los *scripts* generados para adaptar los datos e introducirlos en las bases de datos funcionan correctamente.

Tabla 6.1: Conjunto de pruebas para la verificación del correcto análisis de rendimiento

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
Adquisición y adaptación de los datos	Evaluar la eficiencia del *script* de generación de fichero CSV de películas.	1.1	El fichero tiene 249175 líneas y 12 columnas, y el contenido es el correspondiente.	El fichero tiene 249175 líneas y 12 columnas, y el contenido es el correspondiente.	CORRECTO
	Evaluar la carga correcta de los datos sobre películas en MongoDB.	1.2	Se suben 249175 documentos en la colección *FILMS* y las etiquetas y géneros asociados aparecen en las colecciones apropiadas.	Se suben 249175 documentos en la colección *FILMS* y las etiquetas y géneros asociados aparecen en las colecciones apropiadas.	CORRECTO
	Evaluar la carga correcta de los datos sobre películas en Cassandra.	1.3	Se crean 249175 registros en la tabla *FILMS* y las etiquetas y géneros asociados aparecen en las tablas apropiadas.	Se crean 249175 registros en la tabla *FILMS* y las etiquetas y géneros asociados aparecen en las tablas apropiadas.	CORRECTO
	Evaluar la carga correcta de los datos sobre películas en Neo4j.	1.4	Se crean 249175 nodos con la etiqueta. *FILM*	Se crean 249175 nodos con la etiqueta. *FILM*	CORRECTO
	Evaluar la carga correcta de los datos de series temporales en MongoDB.	1.5	Se suben 604800 documentos en la colección *SENSOR_VALUES* y 10 documentos aleatorios están bien constituidos.	Se suben 604800 documentos en la colección *SENSOR_VALUES* y 10 documentos aleatorios están bien constituidos.	CORRECTO
	Evaluar la carga correcta de los datos de series temporales en Cassandra.	1.6	Se crean 2419200 registros en la tabla *SENSOR_VALUES* y 10 registros aleatorios están bien constituidos.	Se crean 2419200 registros en la tabla *SENSOR_VALUES* y 10 registros aleatorios están bien constituidos.	CORRECTO

Continuación de la tabla 6.1

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Evaluar la carga correcta de los datos de series temporales en Neo4j.	1.7	Se crean 604800 nodos con la etiqueta *VALUE* y hay 10 elementos consecutivos bien anidados desde un nodo aleatorio.	Se crean 604800 nodos con la etiqueta *VALUE* y los elementos no se unen consecutivamente.	INCORRECTO
Adecuación de las consultas	Correlación entre las consultas planteadas y el resultado obtenido para las consultas de películas en MongoDB.	2.1	El resultado de todas las consultas se corresponde con el esperado.	El resultado de todas las consultas se corresponde con el esperado.	MEJORABLE: devuelve todo el documento afectado
	Correlación entre las consultas planteadas y el resultado obtenido para las consultas de películas en Cassandra.	2.2	El resultado de todas las consultas se corresponde con el esperado.	El resultado de todas las consultas se corresponde con el esperado.	CORRECTO
	Correlación entre las consultas planteadas y el resultado obtenido para las consultas de películas en Neo4j.	2.3	El resultado de todas las consultas se corresponde con el esperado.	El resultado de todas las consultas se corresponde con el esperado.	CORRECTO
	Correlación entre las consultas planteadas y el resultado obtenido para las consultas de series temporales en MongoDB.	2.4	El resultado de todas las consultas se corresponde con el esperado.	El resultado de todas las consultas se corresponde con el esperado.	MEJORABLE: devuelve todo el documento afectado

Continuación de la tabla 6.1

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Correlación entre las consultas planteadas y el resultado obtenido para las consultas de series temporales en Cassandra.	2.5	El resultado de todas las consultas se corresponde con el esperado.	El resultado de todas las consultas se corresponde con el esperado.	CORRECTO
	Correlación entre las consultas planteadas y el resultado obtenido para las consultas de series temporales en Neo4j.	2.6	El resultado de todas las consultas se corresponde con el esperado.	El resultado de todas las consultas se corresponde con el esperado.	CORRECTO

Resultados incorrectos:

- **Prueba 1.7**: Tras la inserción de los datos de series temporales en el grafo de Neo4j las relaciones *NEXT* entre nodos consecutivos del mismo tipo de sensor no se habían constituido correctamente. Un fallo en el *script* de generación del grafo no creaba la relación correspondiente.

Resultados mejorables:

- **Prueba 2.1**: Las consultas de MongoDB para el conjunto de datos de películas devolvían los documentos que cumplen las condiciones, y no los campos correspondientes de dichos documentos. Se ha realizado una modificación en las consultas de manera que se responde a cada consulta con campos y no con documentos.
- **Prueba 2.4**: Las consultas de MongoDB para el conjunto de datos de series temporales devolvían los documentos que cumplen las condiciones, y no los campos correspondientes de dichos documentos. Se ha realizado una modificación en las consultas de manera que se responde a cada consulta con campos y no con documentos.

6.2. Pruebas de la aplicación web

Durante el desarrollo de la aplicación web se ha llevado a cabo la verificación de ciertas funcionalidades con las que se han identificado ciertos puntos débiles de la aplicación que posteriormente han sido solventados. En este proceso se han tenido en cuenta aspectos relacionados con el flujo de eventos de diferentes casos de uso, así como de la seguridad de la aplicación y la gestión de los datos.

6.2.1. Verificación de funcionalidades generales

En la tabla 6.2 se detallan las pruebas realizadas para la verificación del correcto funcionamiento de algunas de las funcionalidades que un usuario puede realizar sobre la aplicación web.

Se tratan de pruebas que han servido para identificar errores y mejoras durante y tras la finalización del desarrollo de dichas funcionalidades, y que han sido solucionadas antes de obtener la aplicación final.

Tabla 6.2: Conjunto de pruebas para la verificación de las funcionalidades generales de la aplicación web

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
Enrutamiento	Redirección a diferentes vistas desde los menús de la aplicación	1.1	Tanto desde el menú de la barra de navegación como del menú lateral izquierdo se redirige correctamente a todas las vistas.	Tanto desde el menú de la barra de navegación como del menú lateral izquierdo se redirige correctamente a todas las vistas.	MEJORABLE: etiqueta *href* utilizada
	Recarga de las vistas	1.2	Al recargar una vista de la aplicación que no sea la principal se puede acceder manteniendo la propiedad SPA.	Al recargar una vista de la aplicación que no sea la principal da un error de página no encontrada, solo accede a la raíz (/).	INCORRECTO

Continuación de la tabla 6.2

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Accesos al inicio bien redireccionados	1.3	Desde el logo y el botón de inicio se accede a la vista de inicio.	Desde el logo y el botón de inicio se accede a la vista de inicio.	CORRECTO
Gestión del inicio y cierre de sesión	Recarga de las vistas tras el inicio de sesión	2.1	Al iniciar sesión se carga el menú de usuario autenticado y se cambia la barra de menú con la foto del usuario.	Al iniciar sesión se carga el menú de usuario autenticado y se cambia la barra de menú con la foto del usuario.	MEJORABLE: el nombre del menú lateral no se modifica
	Recarga de las vistas tras el cierre de sesión	2.2	Al cerrar sesión desaparece el menú lateral de usuario autenticado y la barra de navegación es la de por defecto.	Al cerrar sesión desaparece el menú lateral de usuario autenticado y la barra de navegación es la de por defecto.	CORRECTO
	Funcionalidades del usuario autenticado	2.3	Tras iniciar sesión, las vistas de usuarios autenticados muestran datos asociados a la cuenta de dicho usuario.	Tras iniciar sesión, las vistas de usuarios autenticados muestran datos asociados a la cuenta de dicho usuario.	CORRECTO
Gestión de las tablas	Modo de inicialización de las tablas	3.1	Todas las tablas de la aplicación se inicializan ordenadas convenientemente y mostrando un número de registros inicial predefinido.	Todas las tablas de la aplicación se inicializan ordenadas convenientemente y mostrando un número de registros inicial predefinido.	CORRECTO

Continuación de la tabla 6.2

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Filtrado de las tablas	1.2	Todas las tablas disponen de un campo de texto para el filtrado y funciona obteniendo los registros de la tabla que contienen dicho texto.	Todas las tablas disponen de un campo de texto para el filtrado y funciona obteniendo los registros de la tabla que contienen dicho texto.	CORRECTO
	Ordenación de las tablas	1.3	Todas las tablas disponen de un icono de ordenación (ascendente y descendente) en cada campo y funciona según las especificaciones.	Todas las tablas disponen de un icono de ordenación (ascendente y descendente) en cada campo y funciona según las especificaciones.	CORRECTO

Resultados incorrectos:

- **Prueba 1.2**: En un principio, al acceder a una ruta que no se correspondía con la raíz, la aplicación daba un error de "página no encontrada". Esto es debido a que el servicio de *hosting* trabaja con el enrutamiento como si fuera una aplicación web convencional, y no una *Single Page Application*. La solución es duplicar el fichero principal *index.html* por *202.html*, que es el fichero que el servicio de *hosting* lee con las aplicaciones SPA.

Resultados mejorables:

- **Prueba 1.1**: La redirección entre vistas se produce pero se recarga la página al hacerlo debido a que se utiliza la etiqueta *href* y no la etiqueta *to* que proporciona el enrutamiento de *Vue.js*, dando una sensación de *Single Page Application*.
- **Prueba 2.1**: En un primer momento el nombre del usuario era una constante y no se modificaba en función del usuario autenticado actual. La solución es añadir la referencia al nombre de usuario en dicho campo.

6.2.2. Verificación de los datos

En la tabla 6.3 se detallan las pruebas realizadas para la comprobación de la correcta interacción entre las diferentes vistas y la base de datos, es decir, tanto la obtención y muestra de datos por la interfaz como el almacenamiento de nuevos datos introducidos por el usuario a través de las diferentes vistas de la aplicación.

Tabla 6.3: Conjunto de pruebas para la verificación de la interacción entre base de datos y aplicación

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
Inserción de datos	Creación de nuevos usuarios	1.1	Cuando un usuario se registra se generan los documentos asociados en las colecciones de *users*, *activity* y *ranking* y se añaden los campos asociados a la cuenta.	Cuando un usuario se registra se generan los documentos asociados en las colecciones de *users* y *activity* y se añaden los campos asociados a la cuenta.	INCORRECTO
	Inserción de nuevos eventos de un usuario	1.2	Cuando un usuario autenticado realiza una acción que tiene asociado un evento, el evento se almacena en base de datos.	Cuando un usuario autenticado realiza una acción que tiene asociado un evento, el evento se almacena en base de datos.	CORRECTO
	Funcionamiento de formularios	1.3	Cuando un usuario rellena un formulario y lo envía, si se verifican los datos, éstos se guardan correctamente en la base de datos.	Cuando un usuario rellena un formulario y lo envía, si se verifican los datos, éstos se guardan correctamente en la base de datos.	CORRECTO

Continuación de la tabla 6.3

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Modificación de las puntuaciones de un usuario	1.4	Cuando un usuario participa en el *quizz*, los puntos obtenidos se suman a su puntuación actual en la base de datos.	Cuando un usuario participa en el *quizz*, los puntos obtenidos no se suman a su puntuación actual en la base de datos.	INCORRECTO
Consulta de datos	Obtener datos de perfil de usuario	2.1	Las sentencias de consulta de la colección *users* desde el servidor son correctas y garantizan el acceso a los datos.	Las sentencias de consulta de la colección *users* desde el servidor son correctas y garantizan el acceso a los datos.	CORRECTO
	Obtener datos de actividad y progresos de usuario	2.2	Las sentencias de consulta de la colección *activity* desde el servidor son correctas y garantizan el acceso a los datos.	Las sentencias de consulta de la colección *activity* desde el servidor son correctas y garantizan el acceso a los datos.	CORRECTO
	Obtener ranking general	2.3	Las sentencias de consulta de la colección *ranking* desde el servidor son correctas y garantizan el acceso a los datos.	Las sentencias de consulta de la colección *ranking* desde el servidor son correctas y garantizan el acceso a los datos.	CORRECTO
	Obtener preguntas del *quizz*	2.4	Las sentencias de consulta de la colección *quizz* desde el servidor son correctas y garantizan el acceso a los datos.	Las sentencias de consulta de la colección *quizz* desde el servidor son correctas y garantizan el acceso a los datos.	CORRECTO

Continuación de la tabla 6.3

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Obtener archivos de descarga del almacenamiento	2.5	El acceso a los archivos almacenados en el servicio de almacenamiento está abierto.	El acceso a los archivos almacenados en el servicio de almacenamiento está cerrado.	INCORRECTO

Resultados incorrectos:

- **Prueba 1.1**: En un principio, con la creación de un nuevo usuario (el registro de un nuevo usuario) se generaba únicamente un documento asociado en la colección *users* y en la colección *activity*, pero faltaba por crearlo en la colección *ranking*.
- **Prueba 1.4**: Cuando un usuario suma puntos en el *quizz* los puntos deben sumarse a la colección *ranking*, en su registro personal. La sentencia de actualización del registro era errónea y no sumaba la puntuación adicional.
- **Prueba 2.5**: El servicio de almacenamiento restringe todos los accesos a su contenido por defecto, por lo que en un primer momento los archivos no se podían descargar. La solución es modificar las reglas de acceso al sistema dando permisos a los usuarios en función del archivo requerido.

6.2.3. Verificación de la seguridad

En la tabla 6.4 se detallan las pruebas realizadas para verificar la seguridad interna de la aplicación web en relación a los accesos permitidos a los usuarios y la verificación de formularios en el servidor.

Tabla 6.4: Conjunto de pruebas para la verificación de la seguridad interna de la aplicación

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
Restricción de accesos a vistas	Usuarios no autenticados no pueden acceder a vistas para usuarios autenticados.	1.1	Si un usuario no autenticado trata de acceder a las vistas de perfil, progresos, descargas, *quizz*, actividad o cuenta, se le redirige automáticamente al inicio.	Si un usuario no autenticado trata de acceder a las vistas de perfil, progresos, descargas, *quizz*, actividad o cuenta, se le redirige automáticamente al inicio.	CORRECTO
	Un usuario autenticado que cierra sesión no tiene acceso a vistas para usuarios autenticados.	1.2	Al cerrar sesión el usuario autenticado pasa a ser un usuario no autenticado y no tiene acceso a las vistas restringidas.	Al cerrar sesión el usuario autenticado pasa a ser un usuario no autenticado y no tiene acceso a las vistas restringidas.	CORRECTO
Restricción de accesos a base de datos y sistema de almacenamiento	Usuarios autenticados y no autenticados tienen acceso a la zona de la base de datos general.	2.1	Las colecciones de *ranking* y *events* son accesibles por todos los usuarios.	Las colecciones de *ranking* y *events* son accesibles por todos los usuarios	CORRECTO
	Usuarios no autenticados no tienen acceso a las zonas de la base de datos que requieren de autenticación.	2.2	Las colecciones de *users*, *activity* y *quizz* son accesibles solamente por los usuarios autenticados.	Las colecciones de *users*, *activity* y *quizz* son accesibles por todos los usuarios.	INCORRECTO
	Usuarios autenticados solo pueden acceder a las zonas de las bases de datos correspondientes a su usuario.	2.3	Las colecciones de *users* y *activity* son accesibles por los usuarios autenticados únicamente a través de su documento asociado.	Las colecciones de *users* y *activity* son accesibles totalmente por los usuarios autenticados.	INCORRECTO

Continuación de la tabla 6.4

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Usuarios autenticados y no autenticados tienen acceso a la zona del almacenamiento general.	2.4	Todos los usuarios pueden acceder al directorio *default* del servicio de almacenamiento.	Todos los usuarios pueden acceder al directorio *default* del servicio de almacenamiento.	CORRECTO
	Usuarios autenticados tienen acceso a las zonas restringidas de almacenamiento de su usuario.	2.5	Los usuarios autenticados pueden acceder solamente a su directorio personal dentro del directorio *users*.	Los usuarios autenticados pueden acceder solamente a su directorio personal dentro del directorio *users*.	CORRECTO
Gestión de formularios	Envío de contenido erróneo por formulario.	3.1	La verificación en cliente comprueba que los datos introducidos en un formulario no cumplen los requisitos y detiene el envío.	La verificación en cliente comprueba que los datos introducidos en un formulario no cumplen los requisitos y detiene el envío.	CORRECTO
	Envío de contenido correcto por formulario	3.2	La verificación en cliente comprueba que los datos introducidos en el formulario cumplen los requisitos y envía el contenido al servidor.	La verificación en cliente comprueba que los datos introducidos en el formulario cumplen los requisitos y envía el contenido al servidor.	CORRECTO
	Envío forzado de contenido erróneo por formulario directamente al servidor sin verificación en cliente	3.3	La verificación en servidor comprueba que los datos introducidos en un formulario no cumplen los requisitos y rechaza los datos.	La verificación en servidor comprueba que los datos introducidos en un formulario no cumplen los requisitos y rechaza los datos.	CORRECTO

Continuación de la tabla 6.4

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
	Envío forzado de contenido correcto por formulario directamente al servidor sin verificación en cliente	3.4	La verificación en servidor comprueba que los datos introducidos en el formulario cumplen los requisitos y acepta los datos.	La verificación en servidor comprueba que los datos introducidos en el formulario cumplen los requisitos y acepta los datos.	CORRECTO

Resultados incorrectos:

- **Prueba 2.2**: En un principio, las reglas de acceso a la base de datos permitían el acceso a todos los usuarios a todo el contenido. Se ha generado una regla en el servicio que solo permite acceder a los usuarios autenticados a los registros de la base de datos con necesidad de autenticación.
- **Prueba 2.3**: En un principio, las reglas de acceso a la base de datos permitían el acceso a todos los usuarios autenticados a todo el contenido. Se ha generado una regla en el servicio que permite acceder a los usuarios autenticados a su registro personal en los apartados con necesidad de autenticación.

6.2.4. Verificación de los productos descargables

En la tabla 6.5 se detallan las pruebas realizadas con el contenido descargable de la aplicación, tanto en relación a la disponibilidad de dichos archivos por el usuario como a la verificación de las condiciones apropiadas de cada uno de los archivos.

Tabla 6.5: Conjunto de pruebas para la verificación de las condiciones de los archivos descargables

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
Disponibilidad de archivos	Descarga continua del mismo archivo varias veces consecutivas	1.1	El sistema responde correctamente a todas las solicitudes.	El sistema responde correctamente a todas las solicitudes.	CORRECTO

Continuación de la tabla 6.5

Grupo	Descripción	Cód.	Salida esperada	Salida real	Observaciones
Evaluación del contenido	Pruebas de rendimiento con las máquinas virtuales	2.1	Ambas máquinas son instalables, tienen conexión a internet, están configuradas en una red y pueden utilizar los servicios y clientes de los sistemas *NoSQL*.	Ambas máquinas son instalables, tienen conexión a internet, están configuradas en una red y pueden utilizar los servicios y clientes de los sistemas *NoSQL*.	MEJORABLE: acceso a servicios en MV Linux complicado
	Subida de ficheros de datos a una base de datos para verificar su correcta estructura	2.2	Los ficheros de datos presentan una estructura correcta y se pueden subir a las distintas bases de datos correctamente.	Los ficheros de datos presentan una estructura correcta y se pueden subir a las distintas bases de datos correctamente.	CORRECTO
	Instalación de los diferentes ejecutables	2.3	Tanto los archivos de servidores como de clientes se instalan correctamente.	Tanto los archivos de servidores como de clientes se instalan correctamente.	CORRECTO

Resultados mejorables:

- **Prueba 2.1**: La máquina virtual con el sistema operativo Linux Mint puede mejorar si se facilita la inicialización de los servicios. En un principio había necesidad de ejecutarlos manualmente desde la consola, ahora se ha generado un *script* alojado en el escritorio que con un doble *click* ejecuta todos los procesos asociados a la inicialización de dicho servicio.

7. CAPÍTULO

Seguimiento y control

En este capítulo se presentan los resultados obtenidos del seguimiento y control de este TFG, que se ha ido llevando a cabo durante el desarrollo del proyecto. Estos resultados han servido para identificar vulnerabilidades del proyecto y encontrar soluciones a los diferentes conflictos que han podido surgir.

7.1. Gestión del alcance

El alcance del proyecto no ha sufrido grandes variaciones respecto al contenido del proyecto y los objetivos planteados.

7.1.1. Descripción de los objetivos alcanzados

En cuanto a los objetivos técnicos alcanzados con el desarrollo de este proyecto, se ha conseguido **desarrollar un sistema real en torno a los sistemas de gestión de bases de datos *NoSQL*** a través de una aplicación web cuyo objetivo principal es facilitar la iniciación en estos sistemas a aquellas personas que quieran hacerlo. El contenido de la aplicación web desarrollada es el especificado en un inicio y, además, se han ido añadiendo diferentes funcionalidades adicionales que han permitido generar un producto de mayor calidad y más atractivo para el usuario.

Además, se ha realizado una **comparativa de rendimiento** entre varios sistemas *NoSQL*

seleccionados y se han mostrado los resultados obtenidos de cara a proporcionar una visión detallada de las características del funcionamiento de cada uno de los sistemas. Estos resultados permiten valorar la eficiencia de cada sistema en distintos tipos de escenarios en función del tipo de datos con los que se está trabajando, así como entender el modo de funcionamiento de dichos sistemas.

Respecto a los objetivos de formación, se ha conseguido alcanzar un nivel de conocimiento alto de los tres sistemas *NoSQL* con los que se ha trabajado en este proyecto, tanto del funcionamiento del propio sistema como de la sintaxis de consultas con la que trabajan. Además, se ha aprendido a generar diseños de la base de datos para cada tipo de sistema y a comprender la relación entre el modo de almacenamiento de datos por parte de cada sistema y la influencia que ello tiene en los resultados finales de las consultas implementadas.

Asimismo, se ha conseguido alcanzar un nivel de conocimiento alto acerca de las tecnologías para el desarrollo de la aplicación web, tanto las destinadas al desarrollo del *front-end* como el *back-end*.

7.1.2. Evolución del EDT inicial

Los paquetes de trabajo identificados en la planificación del proyecto y representados a través de un diagrama EDT (fig 3.4) en el apartado 3.3.2.1 de este TFG, no han sufrido variaciones significativas. Sin embargo, se han añadido algunas tareas y subtareas a las identificadas para algunos de los paquetes de trabajo.

A continuación, se indican las tareas añadidas a los paquetes de trabajo del proyecto, especificando únicamente los paquetes de trabajo afectados por estas variaciones.

Paquete de trabajo **Virtualización (V)**:

- **V.3**: Configuración y preparación de la máquina virtual generada para dejarla en condiciones de ser utilizada por el usuario.

Paquete de trabajo **Desarrollo (D)**:

- **D.9**: Especificar las reglas se seguridad apropiadas tanto a los servicios de almacenamiento y base de datos como a la propia aplicación.

Paquete de trabajo **Pruebas del desarrollo (PD)**:

- **PD.5**: Verificación de la seguridad de la aplicación web generada respecto a los servicios que utiliza y a la gestión de las restricciones a los usuarios.

7.2. Gestión de las incidencias

Los tiempos estimados en la planificación del proyecto para las tareas de análisis de rendimiento y desarrollo de la aplicación web se plantearon mal desde un principio. Los colchones de tiempo planificados no han sido suficientes para garantizar un seguimiento correcto de las fechas estimadas para cada tarea, lo que ha implicado que la fecha de finalización del proyecto se haya atrasado en 2 meses. A finales del mes de mayo se llevó a cabo una reflexión sobre el estado del desarrollo del proyecto y se vio que no era viable que la finalización del proyecto pudiera darse a principios del mes de julio, por lo tanto se realizó una replanificación con las tareas pendientes y se acordó atrasar una convocatoria para la entrega de este TFG.

Los motivos principales de estas desviaciones han sido la realización de actividades de carácter personal durante el desarrollo del proyecto que han atrasado notablemente las fechas de inicio de ciertas tareas, así como problemas surgidos durante el desarrollo del proyecto. Los principales problemas identificados y que más han afectado a la desviación de tiempos de este proyecto son los siguientes:

- **Funcionalidades de software restringidas**: A la hora de analizar el software utilizado en el desarrollo del proyecto se identificaron las especificaciones de cada uno pero no se tuvo en cuenta el tipo de licencia del software. Algunas herramientas como *MDBVue* o *BootstrapVue* disponen de licencias gratuitas y no gratuitas. En un primer momento se identificaron algunas de las funcionalidades asociadas a cuentas no gratuitas para utilizarlas en la aplicación web a generar, pero dado que dentro del alcance de este proyecto no cabe la utilización de licencias de pago, se tuvieron que replantear dichas funcionalidades para tratar de implementarlas en el proyecto con herramientas de licencia gratuita. Esto supuso un tiempo adicional tanto para las tareas de conocimiento de las herramientas como para las de desarrollo de la parte *front-end* de la aplicación web.

- **Consultas para el sistema MongoDB mal generadas**: De cara a realizar el análisis de rendimiento del sistema *MongoDB* para el conjunto de datos de series temporales, en un principio se planteó un diseño de la base de datos que implicaba plantear

unas consultas con una sintaxis extensa y complicada, y finalmente se vio que no era el modo más apropiado de realizar el análisis. Este diseño de la base de datos, identificación de consultas y posterior análisis con malos resultados, supuso volver a realizar todo el proceso iniciando con otro diseño de la base de datos más apropiado. Finalmente se han incluido ambos diseños y resultados en el análisis de rendimiento de este TFG de cara a proporcionar dos alternativas, por lo tanto no se trata de trabajo "perdido". Sin embargo, ha supuesto un tiempo adicional al planificado que ha extendido notablemente el tiempo destinado al análisis de rendimiento de los diferentes sistemas *NoSQL*, sobretodo el concreto para el sistema *MongoDB*, y ha desviado las fechas de inicio y fin inicialmente identificadas de las tareas posteriores.

- **Problemas de red entre máquina virtual Windows 10 y software de inserción de datos**: Dado que el modo de insertar los datos en las diferentes bases de datos de algunos de los sistemas *NoSQL* utilizados es a través de un programa Java utilizando los *drivers* correspondientes de los sistemas, es necesario configurar un sistema de red en la máquina virtual que permita acceder desde un dispositivo externo al servicio para insertar directamente los datos en la base de datos. La máquina virtual generada con el sistema operativo Windows 10 planteó una serie de dificultades a la hora de realizar esta configuración ya que no se encontraba ninguna manera de permitir la conexión entre el sistema invitado y el anfitrión. Finalmente se optó por reconstruir la máquina virtual desde cero y el problema se acabó solucionando. Esto supuso una "pérdida" de tiempo respecto al estimado en la planificación para la realización de esta tarea, y un atraso en las fechas de inicio y fin de las tareas sucesivas.

- **Actividad adicional para el grupo de investigación *BDI***: A finales del mes de junio el desarrollo de este TFG se vio paralizado durante 5 días consecutivos debido a que se tuvieron que destinar varias horas para realizar un análisis de tiempos de ejecución del sistema *MongoDB* en torno a actividades relacionadas con el propio grupo de investigación. Estas actividades se desarrollaron durante el periodo estimado para la finalización de la aplicación web tras la replanificación, por lo tanto, se tuvo que atrasar una semana dicha finalización.

7.3. Gestión del tiempo y tareas

Debido a las incidencias previamente identificadas, el proyecto ha sufrido ciertas desviaciones tanto a nivel de los periodos de realización de las tareas como a la duración de las mismas.

7.3.1. Periodos de realización de las tareas y desviaciones

En la tabla 7.1 se muestra una comparativa entre la fecha de fin prevista en la planificación inicial para cada tarea y la fecha fin real de dicha tarea, indicando la desviación en días, semanas o meses que ha sufrido dicha tarea respecto a su periodo de realización. Para considerar la gravedad de la desviación, se muestran en verde aquellas desviaciones positivas o cuya diferencia es de 3 días o menos, en naranja aquellas cuya diferencia está entre 3 días y una semana, y en rojo aquellas cuya diferencia es superior a una semana.

Dado que algunas tareas son posteriores a la finalización de esta memoria, se indica la fecha aproximada de finalización y no la real.

Tabla 7.1: Comparativa de fecha fin prevista y fecha fin real de las diferentes tareas del proyecto

Tareas	Fecha fin prevista	Fecha fin real	Desviación
Trabajo de Fin de grado	**01/07/19**	**10/09/19 (aprox.)**	**+2 meses**
Gestión del proyecto (GP)	**03/07/19**	**16/09/19 (aprox.)**	**+2 meses y 1/2**
Planificación (P)	27/03/19	26/05/19	+2 meses
P.1: Decisiones iniciales	14/02/19	15/02/19	+1 día
P.2: Planificación inicial	15/02/19	20/02/19	+1 semana
P.3: Actualización de la planificación	27/03/19	26/05/19	+2 meses
Seguimiento y Control (SyC)	03/07/19	10/09/19 (aprox.)	+2 meses
SyC.1: Recogida de información	12/06/19	30/06/19	+2 semanas
SyC.2: Control tiempos Smartsheet	03/07/19	10/07/19	+1 semana
SyC.3: Diario de tareas desarrolladas	11/06/19	10/07/19	+1 mes
SyC.4: Reuniones con la tutora	01/07/19	16/09/19 (aprox.)	+2 meses y 1/2
SyC.5: Desviaciones y riesgos	30/04/19	26/05/19	+1 mes
SyC.6: Condiciones para el éxito del proyecto	14/06/19	14/06/19	
SyC.7: Desarrollo contenido seguimiento y control	17/06/19	10/09/19 (aprox.)	+3 meses

Continuación de la tabla 7.1

Tareas	Fecha fin prevista	Fecha fin real	Desviación
Adquisición de Conocimiento (AC)	**10/05/19**	**10/07/19**	**+2 meses**
Elección de Herramientas (EHH)	18/02/19	28/02/19	+1 semana
EHH.1: Investigar herramientas	13/02/19	16/02/19	+3 días
EHH.2: Realizar pruebas con las herramientas	18/02/19	28/02/19	+1 semana
Adquisición de Competencias (ACC)	10/05/19	10/07/19	+2 meses
ACC.1: MongoDB, Apache Cassandra y Neo4j	10/04/19	09/05/19	+1 mes
ACC.2: Robo 3T, DataStax DevCenter y Neo4j Desktop	01/04/19	05/05/19	+1 mes
ACC.3: Draw.io, Smartsheet y Qlik Sense Desktop	15/02/19	07/03/19	+3 semanas
ACC.4: Vue.js, MDBVue, Node.js y Express	30/04/19	10/07/19	+2 meses y 1/2
ACC.5: Firebase	10/05/19	14/05/19	+4 días
Entorno de Virtualización (EV)	**25/02/19**	**15/04/19**	**+1 mes y 1/2**
Virtualización (V)	25/02/19	15/04/19	+1 mes y 1/2
V.1: Instalación y preparación software *NoSQL*	21/02/19	17/03/19	+1 mes
V.2: Carga de datos en bases de datos	25/02/19	15/04/19	+1 mes y 1/2
V.3: Configuración final	–/–/–	20/04/19	
Comparativa de Rendimiento de Bases de Datos (CRBD)	**17/04/19**	**02/06/19**	**+1 mes y 1/2**
Adquisición de Datos (AD)	25/03/19	20/04/19	+1 mes
ADD.1: Búsqueda de formato compatible	26/02/19	26/02/19	
ADD.2: Búsqueda de datos	04/03/19	04/03/19	
ADD.3: Carga de datos en las bases de datos	25/03/19	20/04/19	+1 mes
Diseño de Base de datos (DBD)	21/03/19	30/03/19	+9 días
DBD.1: Análisis de requisitos SGBDs	18/03/19	23/03/19	+5 días
DBD.2: Transformación de datos	21/03/19	30/03/19	+9 días
Evaluación del Rendimiento (ER)	16/04/19	02/06/19	+1 mes y 1/2
ER.1: Sentencias a ejecutar y adaptación a SGBD	05/04/19	30/04/19	+3 semanas
ER.2: Preparación herramientas análisis	10/04/19	05/05/19	+1 mes
ER.3: Ejecución de las sentencias y conclusiones	16/04/19	02/06/19	+1 mes y 1/2
Aplicación Web (AW)	**09/06/19**	**10/07/19**	**+1 mes**

Continuación de la tabla 7.1

Tareas	Fecha fin prevista	Fecha fin real	Desviación
Diseño Conceptual (DC)	30/04/19	05/07/19	+2 meses y 1/4
DC.1: Descripción funcionalidades aplicación	16/04/19	15/06/19	+2 meses
DC.2: Identificación y representación de los casos de uso	24/04/19	30/06/19	+2 meses
DC.3: Diseño modelo base de datos	30/04/19	05/07/19	+2 meses y 1/4
Diseño Gráfico (DG)	17/05/19	05/07/19	+2 meses
DG.1: Representación mano alzada interfaces	29/04/19	20/06/19	+1 mes y 3/4
DG.2: Representación concreta interfaces	01/05/19	22/06/19	+1 mes y 3/4
DG.3: Análisis y mejora interfaces	17/05/19	05/07/19	+1 mes y 1/2
Preparación del Entorno (PE)	06/05/19	11/06/19	+1 mes y 1/4
PE.1: Git y Github y sincronización	06/05/19	06/06/19	+1 mes
PE.2: Node.js, Express.js, Vue.js, MDB-Vue y Visual Studio Code	03/05/19	10/06/19	+1 mes y 1/4
PE.3: Añadir servicios de Firebase	06/05/19	11/06/19	+1 mes y 1/4
Desarrollo (D)	06/06/19	10/07/19	+1 mes
D.1: Desarrollo front-end	06/06/19	10/07/19	+1 mes
D.2: Desarrollo back-end	06/06/19	11/04/19	+1 mes
D.3: Sincronización del código fuente con Firebase	09/05/19	10/06/19	+1 mes
D.4: Servicio de autenticación de Firebase y aplicación	10/05/19	11/06/19	+1 mes
D.5: Servicio de base de datos de Firebase y aplicación	15/05/19	11/06/19	+1 mes
D.6: Servicio de almacenamiento de Firebase y aplicación	14/05/19	12/06/19	+1 mes
D.7: Servicio de hosting de Surge y aplicación	17/05/19	15/07/19	+2 meses
D.8: Sincronización de todos los servicios de Firebase	20/05/19	13/06/19	+3 semanas
D.9: Seguridad de la aplicación	–/–/–	29/06/19	
Pruebas (PR)	**21/06/19**	**20/07/19**	**+1 mes**
Pruebas del Desarrollo (PD)	21/06/19	20/07/19	+1 mes
PD.1: Verificación de las interfaces gráficas	14/06/19	16/07/19	+1 mes
PD.2: Verificación conexión servidor y base de datos	17/06/19	16/07/19	+1 mes

Continuación de la tabla 7.1

Tareas	Fecha fin prevista	Fecha fin real	Desviación
PD.3: Verificación conexión servidor y autenticación	18/06/19	16/07/19	+1 mes
PD.4: Verificación del alojamiento en Surge	21/06/19	20/07/19	+1 mes
PD.5: Verificación de la seguridad	–/–/–	04/07/19	
Pruebas de Gestión (PG)	08/15/19	30/06/19	+1 mes y 1/2
PG.1: Verificación Git y Github	03/05/19	09/06/19	+1 mes y 1/4
PG.2: Verificación sincronización código	08/05/19	30/06/19	+1 mes y 3/4
Pruebas de Datos (PDD)	27/03/19	20/04/19	+3 semanas
PDD.1: Verificación transformación de datos	08/03/19	05/04/19	+1 mes
PDD.2: Verificación carga correcta de datos	27/03/19	20/04/19	+3 semanas
Documentación (D)	**01/07/19**	**10/09/19 (aprox.)**	**+1 mes y 1/2**
Generación del Informe (INF)	11/06/19	27/07/19	+1 mes y 1/2
INF.1: Preparar entorno de desarrollo	13/02/19	05/02/19	
INF.2: Generar documentos	07/06/19	07/06/19	
INF.3: Desarrollar memoria TFG	11/06/19	27/09/19	+1 mes y 1/2
Preparación Defensa TFG (PDT)	28/06/19	10/09/19 (aprox.))	+1 mes y 1/2
PDT.1: Elegir aspectos de la presentación	27/06/19	03/09/19 (aprox.)	+2 meses y 1/4
PDT.2: Generar documento de presentación	28/06/19	10/09/19 (aprox.)	+2 meses y 1/2
PDT.3: Preparar defensa del TFG	28/06/19	10/09/19 (aprox.)	+1 mes y 1/2

7.3.2. Dedicación de tiempo a los paquetes de trabajo y desviaciones

En la tabla 7.2 se muestra una comparativa entre el tiempo inicialmente estimado para el desarrollo de cada paquete de trabajo y el tiempo real utilizado.

La duración real se muestra en verde cuando es inferior o menos de un 25 % mayor a la duración estimada, en naranja cuando es igual o menos de un 50 % mayor que la estimada, y en rojo cuando es un 50 % o más mayor que la duración estimada.

Tabla 7.2: Comparativa de duración estimada y duración real en horas para cada paquete de trabajo del proyecto

Paquete de trabajo	Duración estimada	Duración real
Gestión del proyecto (GP)	**43**	30
Planificación (P)	16	10
Seguimiento y Control (SyC)	27	20
Adquisición de Conocimiento (AC)	**32**	32
Elección de Herramientas (EHH)	8	12
Adquisición de Competencias (ACC)	24	20
Entorno de Virtualización (EV)	**8**	9
Virtualización (V)	8	9
Comparativa de Rendimiento de Bases de Datos (CRBD)	**48**	72
Adquisición de Datos (AD)	8	7
Diseño de Base de datos (DBD)	18	30
Evaluación del Rendimiento (ER)	22	35
Aplicación Web (AW)	**149**	187
Diseño Conceptual (DC)	33	20
Diseño Gráfico (DG)	20	15
Preparación del Entorno (PE)	10	12
Desarrollo (D)	86	140
Pruebas (PR)	**18**	20
Pruebas del Desarrollo (PD)	9	10
Pruebas de Gestión (PG)	4	5
Pruebas de Datos (PDD)	5	5
Documentación (D)	**41**	85
Generación del Informe (INF)	28	75
Preparación Defensa TFG (PDT)	13	10
Proyecto	**339**	435

Las desviaciones más significativas se aprecian en los paquetes de desarrollo dentro de la fase de la aplicación web, y en el de generación del informe dentro de la fase de documentación. El desarrollo de la aplicación ha tomado más de un 60 % de duración adicional a la estimada en un inicio, mientras que el tiempo empleado para generar el informe ha duplicado el tiempo inicial estimado.

Esta diferencia se puede apreciar en el gráfico de la figura 7.1, donde se representan las estimaciones de tiempo y duraciones reales de cada fase.

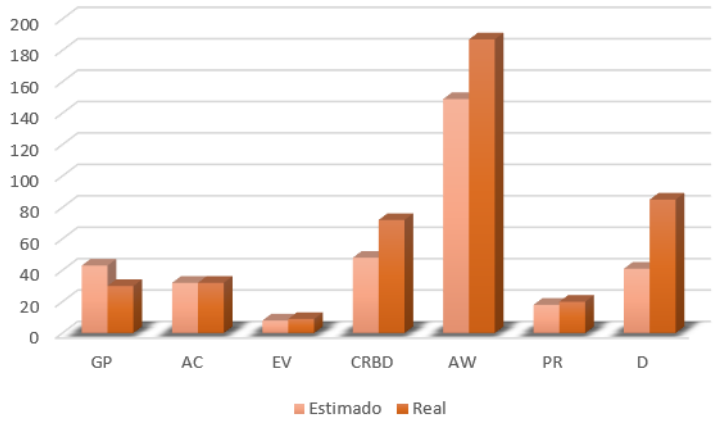


Figura 7.1: Gráfico con la distribución de horas estimadas y reales por cada fase del proyecto

8. CAPÍTULO

Conclusiones y trabajo futuro

En este capítulo de describen las conclusiones obtenidas tras el desarrollo del proyecto, tanto a nivel técnico como personal. Del mismo modo, se detallan una serie de mejoras y avances que podrían ser desarrollados tanto por la parte del análisis de rendimiento como de la aplicación web generada.

8.1. Conclusiones

El principal objetivo del desarrollo de este TFG era conseguir generar un entorno para el aprendizaje *NoSQL* empleando para ello técnicas de análisis de sistemas *NoSQL* y técnicas de desarrollo de aplicaciones web. Con el proyecto finalizado, se puede afirmar que las expectativas planteadas se han conseguido. Han surgido algunos imprevistos durante el desarrollo que han implicado un cambio respecto a la planificación generada en un principio, pero los objetivos planteados se han conseguido.

8.1.1. Conclusiones técnicas

La evaluación global respecto a los aspectos técnicos del proyecto es positiva. Se ha desarrollado un proyecto inmerso en numerosas competencias técnicas que ha permitido llevar a cabo un TFG de alta calidad. Los objetivos técnicos alcanzados tras la realización de este TFG son los siguientes:

- **Creación de un sistema real en torno a los sistemas *NoSQL***: Se ha implementado una aplicación web destinada al aprendizaje *NoSQL* utilizando *software* actual que está en pleno auge en entornos de desarrollo profesionales de aplicaciones web. Las funcionalidades que incluye la aplicación web permiten al usuario adquirir unas competencias básicas en torno a los sistemas *NoSQL* con los que se trabaja. Además, la aplicación web es completa, ya que permite al usuario pasar por varias fases, desde la fase de aprendizaje, pasando por una fase de consolidación de conocimientos, hasta la fase de evaluación de los conceptos aprendidos.

- **Comparativa de rendimiento de diferentes sistemas *NoSQL***: El análisis realizado a los diferentes sistemas permite obtener una visión concreta de los tres sistemas evaluados respecto a su eficiencia en entornos de trabajo similares a los que se han planteado en este proyecto. El trabajo desarrollado en este proyecto da una perspectiva general del modo en el que funcionan algunos sistemas *NoSQL* y permite entender cómo se almacenan los datos en una base de datos y la repercusión que ello tiene en el rendimiento de cada sistema.

- **Interoperabilidad de herramientas de desarrollo**: Se ha conseguido establecer un entorno de desarrollo utilizando diferentes herramientas capaces de interactuar conjuntamente para obtener una aplicación web final con características de cada *software* utilizado.

8.1.2. Conclusiones de carácter personal

La evaluación global personal tras la realización de este proyecto es positiva. Se han adquirido muchos conocimientos en torno a la tecnología *NoSQL* gracias a la práctica con los sistemas *MongoDB*, *Apache Cassandra* y *Neo4j*. Los objetivos personales alcanzados tras la realización de este TFG son los siguientes:

- **Aprendizaje de diferentes tipos de sistemas de gestión de bases de datos *NoSQL***: El continuo trabajo con *MongoDB*, *Apache Cassandra* y *Neo4j* a lo largo del desarrollo de este proyecto me ha permitido conocer el funcionamiento de estos sistemas. Por un lado, la manera en la que almacenan los datos y cómo ello influye en aspectos como el espacio de almacenamiento que ocupan o la facilidad con la que el sistema es capaz de consultar los datos. Además, he tomado contacto con la sintaxis de consultas propia de dichos sistemas y he podido compararlas con la sintaxis

propia de otro tipo de sistemas como los relacionales. El hecho de haber trabajado con más de un tipo de sistema diferenciando el modo de almacenamiento de los datos (orientado a documentos, clave-valor u orientado a grafos) me ha permitido conocer otras alternativas de almacenamiento estructurado de datos.

- **Generar un diseño para bases de datos de sistemas *NoSQL***: Vinculado al objetivo anterior, el trabajo continuado con los sistemas *NoSQL* me ha permitido conocer cómo se genera un diseño de base de datos para este tipo de sistemas. Estos diseños están vinculados al modo en el que los sistemas almacenan los datos en las bases de datos correspondientes. Salir del esquema relacional me ha posibilitado conocer otras opciones de diseño que tienen mucha repercusión en la actualidad.
- **Software para el desarrollo web**: La principal tecnología utilizada para el desarrollo de la aplicación web ha sido JavaScript, tecnología con la que ya había tenido contacto previamente. Sin embargo, los *frameworks* y librerías que he utilizado para desarrollar la aplicación han sido totalmente nuevas para mí. He podido aprender a utilizar estas herramientas (que tienen mucho peso en el ámbito del desarrollo *front-end* y *back-end* actual) y a desenvolverme en un entorno de desarrollo distinto al que estoy acostumbrada.
- **Capacidad de gestión y desarrollo personal de un proyecto de esta envergadura**: Hasta el momento no había desarrollado de manera individual un proyecto con la implicación requerida para este TFG. He podido comprobar de primera mano la dificultad de planificar un proyecto de esta envergadura y lo difícil que puede ser concretar las tareas asociadas a un proyecto y la duración de las mismas. Cualquier contratiempo puede alterar considerablemente la planificación y complicar el transcurso del proyecto. Sin embargo, considero que el proyecto ha finalizado con éxito, habiendo conseguido cumplir los principales objetivos planteados y obteniendo un producto final acorde a los requisitos inicialmente planteados.

8.2. Trabajo futuro

Este TFG ha finalizado habiéndose conseguido los requisitos inicialmente planteados tanto por parte del análisis de rendimiento de los sistemas *NoSQL* como de la aplicación web. Sin embargo, es posible realizar mejoras sobre el trabajo ya realizado para progresar con un análisis más exhaustivo y realizar una aplicación web con mayor número de funcionalidades.

8.2.1. Posibles mejoras en el análisis de rendimiento

De cara a realizar un análisis de rendimiento de sistemas *NoSQL* con más profundidad, hay una serie de aspectos fuera del alcance de este proyecto que se pueden realizar:

- **Pruebas de rendimiento de los sistemas con series temporales reducidas**: En este TFG se ha evaluado el rendimiento de tres sistemas *NoSQL* trabajando con conjuntos de datos genéricos y con series temporales. Las series temporales reducidas con alguna técnica de reducción permiten disminuir el espacio de almacenamiento ocupado por las series temporales. Una línea de análisis podría estar en realizar el mismo procedimiento utilizado en este TFG pero con series temporales reducidas, evaluando los tiempos de ejecución y de reconstrucción de series conjuntamente.

- **Análisis de otro tipo de sentencias**: Se ha realizado un análisis a partir de los resultados obtenidos de la ejecución de sentencias genéricas de consulta, inserción, actualización y eliminación de registros en la base de datos. Se podría ampliar el análisis evaluando otro tipo de sentencias adicionales como sentencias de eliminación en cascada, eliminación de registros de almacenamiento (colecciones, tablas o grafos), consultas, inserciones, actualizaciones o eliminaciones utilizando técnicas de agrupación de sentencias (p. ej.: *BATCH* en Apache Cassandra) en aquellos sistemas que presenten la funcionalidad, etc.

- **Evaluación de otro tipo de sistemas *NoSQL***: En este proyecto se ha optado por trabajar con los principales representantes de los sistemas *NoSQL* orientados a documentos, clave-valor y orientados a grafos. Sin embargo, existen otro tipo de sistemas caracterizados por el modo de almacenamiento de los datos, como orientados a objetos (p. ej.: *Db4o*), orientados a series temporales (p. ej.: *InfluxDB*), multimodelo (p. ej.: *IBM Db2*), etc. Este análisis podría enriquecerse evaluando algunos sistemas de diferente tipo como los mencionados.

- **Técnicas de compresión de datos de los sistemas**: Algunos sistemas presentan funcionalidades de comprensión de los datos que almacenan en la base de datos. Este análisis de rendimiento se ha realizado teniendo en cuenta los valores de almacenamiento por defecto. Sin embargo, sistemas como *MongoDB* utilizan técnicas para comprimir los datos en la base de datos de cara a beneficiar al usuario. Evaluar la eficiencia de estas técnicas podría enriquecer el contenido del análisis, mostrando más alternativas incluso dentro de un mismo sistema.

- **Evaluar la interoperabilidad entre sistemas *NoSQL***: Al igual que se priorizan unos sistemas frente a otros en función de las características de los datos que se quieren almacenar, una característica a tener en cuenta a la hora de seleccionar un sistema con el que trabajar puede ser la capacidad de compartir datos y posibilitar el intercambio de información entre diferentes sistemas que tiene un sistema concreto. Analizar esta característica entre los sistemas evaluados puede enriquecer el análisis realizado y proporcionar resultados más trabajados.

8.2.2. Posibles mejoras para la aplicación web

Los requisitos inicialmente especificados en torno a la aplicación web han sido implementados. Sin embargo, es posible añadir funcionalidades adicionales a la aplicación web con el fin de proporcionar al usuario una serie de operaciones más con las que interactuar con el sistema.

- **Interacción entre usuarios registrados**: Un sistema de mensajería instantánea en tiempo real para la comunicación directa entre usuarios conectados y/o un foro de consultas acerca de aspectos relacionados con *NoSQL*.
- **Usuarios *premium***: Añadir un escalón a la jerarquía de usuarios con los usuarios *premium*, que puede acceder a contenido adicional de la aplicación web, como vídeos o contenido didáctico restringido, descargas adicionales, etc. Esto supondría configurar las reglas de seguridad e implementar nuevas funcionalidades.
- **Suscripción a novedades de la aplicación**: Añadir una funcionalidad a través de la cuál los usuarios pueden recibir en su correo electrónico las novedades que se dan en la aplicación. Estos avisos podrían ser relacionados con noticias nuevas, nuevas descargas disponibles, movimiento en el *ranking* de posiciones del *quizz* y otros aspectos analizables.

Anexos

A. ANEXO

Preparación de la máquina virtual

En este anexo se indica el proceso de creación y preparación de la máquina virtual que es incluida en el contenido de este trabajo de fin de grado.

Con motivo de facilitar un entorno sobre el que trabajar con las tecnologías *NoSQL*, se genera un entorno virtual a través de una máquina virtual con el sistema operativo Windows 10. Sobre esta máquina instalan tres tipos de sistemas gestores de bases de datos *NoSQL*. Uno de ellos en representación de los SGBD orientados a documentos (MongoDB), otro de tipo clave-valor (Apache Cassandra, en su distribución de DataStax) y un último orientado a grafos (Neo4j).

Asimismo, se instala un cliente gráfico para cada uno de los sistemas, que facilita la interacción entre base de datos y usuario gracias a las ayudas visuales que éstos clientes proporcionan. Las herramientas gráficas instaladas son Robo 3T para MongoDB, DataStax DevCenter para Apache Cassandra y Neo4j Desktop para Neo4j.

Para que la persona interesada en conocer estos SGBD tenga disponible un entorno preparado para experimentar y trabajar con ellos, se crean diferentes bases de datos para cada uno de los SGBD con datos de diferente índole y con diferente formato en cada uno.

En este proyecto se parte desde una máquina virtual con el sistema operativo Windows 10 Education previamente configurado sobre Oracle VM VirtualBox v6.0, por lo que se documentan los procesos seguidos para la preparación de esta máquina hacia un entorno *NoSQL*. Además, la máquina virtual generada se utiliza como entorno de trabajo en el análisis de los diferentes SGBD que se realiza en este TFG.

A.1. Adquisición e instalación del software

Con la máquina virtual preparada en su entorno de Windows 10, se procede a descargar el software de cada sistema desde sus páginas oficiales, tanto del lado servidor como de los propios clientes gráficos.

A.1.1. Entorno de MongoDB

A.1.1.1. Instalación de MongoDB

Desde la página oficial de MongoDB se accede a la sección de *descargas*[1] y se descarga el servidor de MongoDB. En este caso, se ha seleccionado la versión 4.0.6 para Windows (64 bits) en formato MSI.

La descarga comienza tras pulsar el botón verde donde indica "Download". Cuando ya se ha descargado el archivo, se pulsa en él y se abre la ventana de instalación. El proceso se instalación es intuitivo y se basa en ir aceptando (y en ocasiones configurando) las diferentes ventanas que van apareciendo.

Se elige la versión completa en la instalación (Fig. A.1), que es la versión que más se adecúa a usuarios que han tenido poco o ningún contacto con MongoDB.

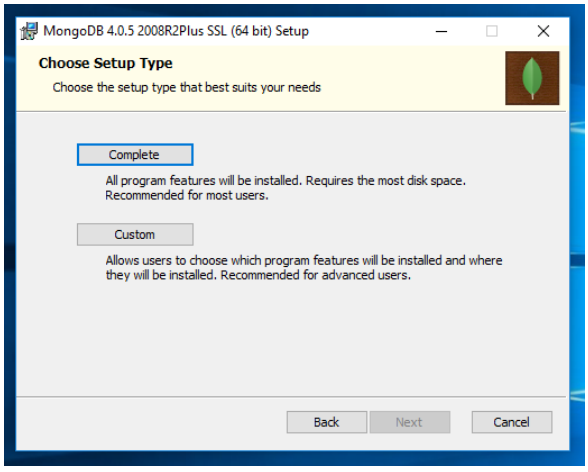


Figura A.1: Selección de instalación completa de MongoDB

Con la configuración por defecto que se muestra en la ventana, se procede a instalar

[1]MongoDB: https://www.mongodb.com/download-center/community?jmp=docs

MongoDB como un Servicio de Windows (Fig. A.2), con el nombre MongoDB y las direcciones de *data* y *log* asociadas por defecto en el directorio de MongoDB de Windows.

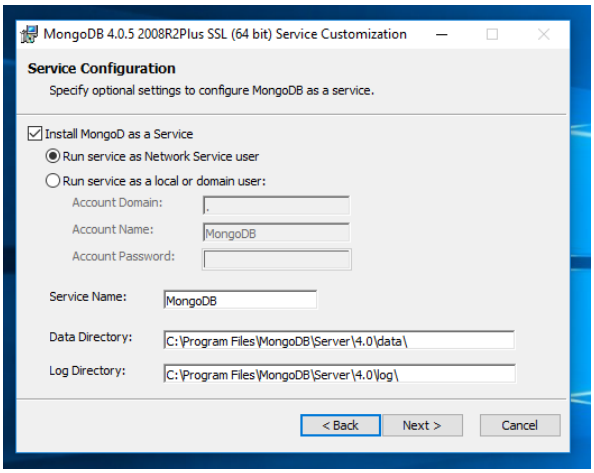


Figura A.2: Instalación de MongoDB como servicio de Windows

La opción de instalar MongoDB Compass se de-selecciona (Fig. A.3). Se trata de una GUI (*Graphical User Interface*) que permite gestionar de manera visual las consultas y operaciones que se pueden realizar sobre MongoDB. Se utilizará otra plataforma similar, pero algo más completa que se explicará posteriormente.

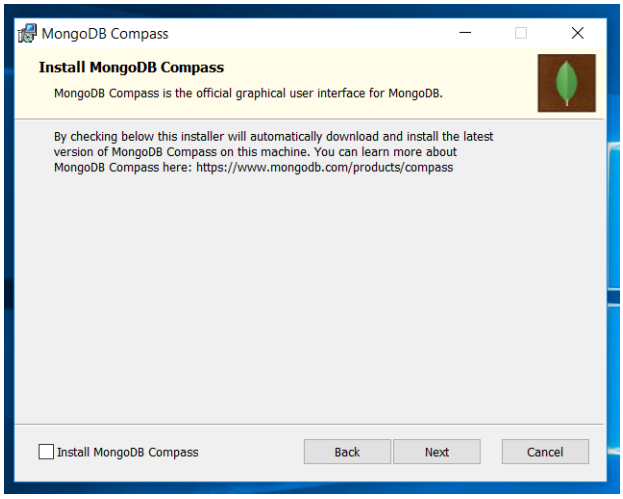


Figura A.3: De-selección de MongoDB Compass

Con toda la configuración previa correctamente realizada, se procede a permitir la instalación (con la cuenta de administrador) de MongoDB en el equipo Windows 10 (64 bit).

A.1.1.2. Instalación de Robo 3T

Para obtener la aplicación cliente que permita trabajar con el servidor MongoDB de manera más sencilla, se ha optado por hacerlo con Robo 3T (RoboMongo).

Se accede a la página oficial de *descargas*[2] de Robo 3T. Desde ésta, se pulsa en el botón verde que indica Download Robot 3T (Fig. A.4) para descargar la versión gratuita de Robo 3T.

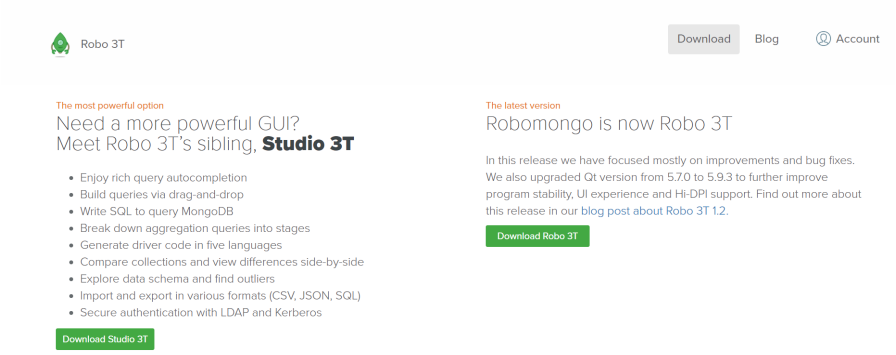


Figura A.4: Vista de la página oficial de descargas de Robo 3T

A continuación, se abre una ventana emergente perteneciente a la interfaz de la página de Robo 3T en la que por defecto se detecta el sistema operativo con el que se está trabajando y se da opción de descargar el instalador de Robo 3T versión 1.2 para ese sistema operativo (Fig A.5). Las opciones seleccionables de descarga son entre los sistemas operativos de Windows, Mac o Linux, en versión 64 bit.

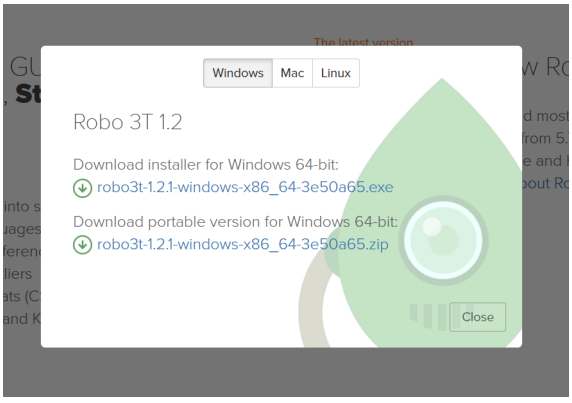


Figura A.5: Selección del ejecutable de Windows

[2]Robo 3T: https://robomongo.org/download

Para la versión de Windows está disponible tanto el instalador como el portable. Se procede a descargar el instalador para Windows. Cuando finaliza la descarga, se pulsa en el ejecutable y comienza el proceso de instalación, que se trata de una secuencia de ventanas del instalador común de Windows.

Tras la finalización de la instalación de Robo 3T la herramienta queda accesible en el equipo a través de un acceso directo en el Escritorio.

A.1.1.3. Configuración del entorno MongoDB

Tras la finalización de la instalación, el servicio MongoDB Server está activo en el SO. Para comprobar la actividad del servicio, desde la herramienta Ejecutar de Windows (Tecla de Windows+R) se accede a la lista de servicios de Windows tecleando *services.msc*.

El servicio con nombre MongoDB Server está en estado "En ejecución" (Fig. A.6), lo que significa que está iniciado. Si el estado de este servicio no se correspondiera con "En ejecución", pulsando con el botón secundario sobre dicho servicio aparece la opción de "Iniciar".

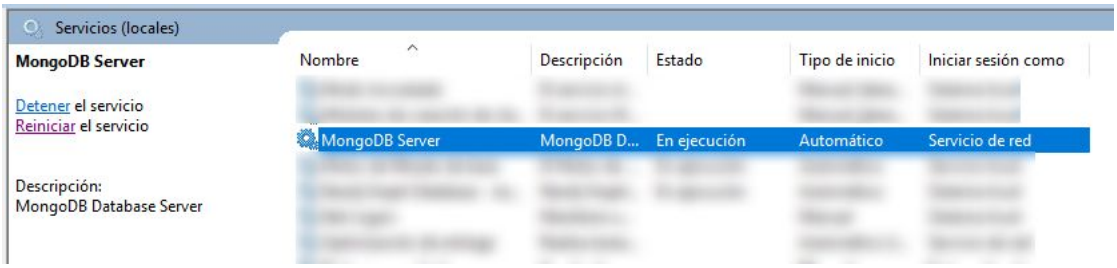


Figura A.6: Estado "En ejecución" del servicio MongoDB Server de Windows

Para establecer una conexión entre el cliente Robo 3T y el servidor MongoDB que está en ejecución en el sistema, hay que crear una conexión desde la interfaz de Robo 3T. Al abrir Robo 3T aparece una ventana con la lista de conexiones creadas, como todavía no se ha configurado ninguna, se pulsa en la opción de *Create* que aparece en la parte superior izquierda de la ventana. A la hora de crear una conexión con la base de datos, Robo 3T permite realizar algunas configuraciones de autenticación y seguridad avanzadas que en este caso no se van a implementar. Desde la pestaña de *Connection* se escoge el tipo por defecto de *Direct Connection* indicando que se quiere establecer una conexión directamente con una base de datos de MongoDB (con el tipo *Replica Set* se permite trabajar con réplicas de las bases de datos). Se anota un nombre para la conexión y finalmente se

indica la dirección del servidor MongoDB a través del *host* y del puerto correspondiente al servicio. En este caso el *host* es *localhost* y el puerto es el 27017, que es el puerto por defecto del servidor MongoDB (Fig. A.7). Este puerto es configurable con el parámetro *port* desde el fichero de configuración *mongod.cfg*.

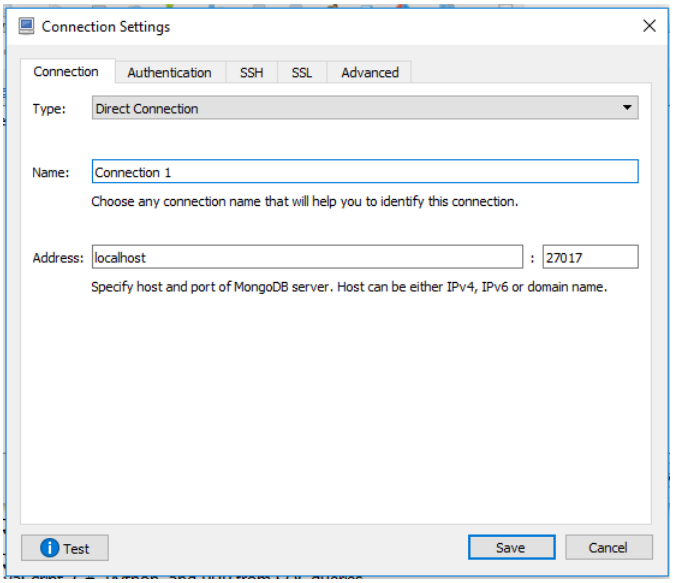


Figura A.7: Ventana de configuración de conexión entre Robo 3T y MongoDB Server

Antes de guardar la configuración es posible testear la conexión pulsando sobre *Test*. Tras guardar los cambios, la conexión aparece en la lista de conexiones y simplemente hay que pulsar sobre ella y después en *Connect*.

A.1.2. Entorno de Apache Cassandra

A.1.2.1. Instalación de la distribución de Apache Cassandra de DataStax

Se va a trabajar con Datastax Distribution Apache Cassandra (DDC), descargable desde la página oficial de Datastax en el apartado de *descargas*[3] de Apache Cassandra. Se descarga la versión 3.9.0 del archivo en formato Microsoft Installer (MSI Installer) para Windows (Fig. A.8).

[3]DataStax Distribution of Apache Cassandra: https://downloads.datastax.com/#ddac

The legacy DataStax Distribution of Apache Cassandra™ (DDC) is a free packaged distribution of Apache Cassandra™ made available by DataStax. Besides Apache Cassandra™ itself, it includes the DataStax DevCenter tool, a Window's installer, and DataStax's professional documentation.

To get started with your download, select a DataStax Distribution of Apache Cassandra™ installer from the table below.

	v3.9.0	v3.8.0	Instructions
All Linux distributions and Mac OS X 10.x	Tarball	Tarball	Tarball
Windows Server 2008/2012 or Windows 7/8 (64-Bit)	MSI Installer	MSI Installer	MSI Installer
Red Hat Enterprise Linux / CentOS 6.5 or later	See "Instructions"	See "Instructions"	RPM Using Yum
Latest Ubuntu LTS or Debian Stable	See "Instructions"	See "Instructions"	DEB Using Apt-Get

DataStax Community Edition for Apache Cassandra™ (v2.1.x – v3.0.x) Consists of Several Components:

- An "Archive Release" of Apache Cassandra™
- DataStax OpsCenter Monitoring Tool (Does not work with Apache Cassandra™ v3.0 and beyond; click here to download & setup OpsCenter for other operating systems and previous versions)
- Sample application and demo database

Figura A.8: Vista de la página oficial de descargas de DataStax Distribution of Apache Cassandra

Cuando el instalador está descargado, se procede a realizar la instalación de DDC a través de una secuencia de ventanas que se deben ir aceptando consecutivamente.

Entre estas ventanas cabe destacar la configuración de DataStax DDC como un servicio que se va a iniciar automáticamente en el arranque del SO (Fig. A.8).

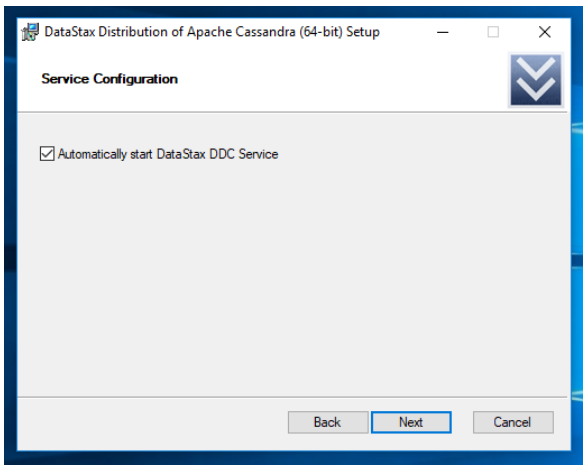


Figura A.9: Configuración de DataStax DCC como servicio de inicio automático en el SO

A.1.2.2. Instalación de DataStax DevCenter

DataStax DevCenter se instala automáticamente con la instalación previa de *DataStax Distribution of Apache Cassandra*. Accediendo desde el inicio a la opción *DataStax Distribution of Apache Cassandra* en el menú, aparece un desplegable con las herramientas instaladas y una de ellas es DataStax DevCenter.

A.1.2.3. Configuración de del entorno Apache Cassandra

Con el servicio DataStax DCC Server instalado, es posible comprobar y controlar la actividad de éste desde la lista de servicios de Windows.

El servicio con nombre DataStax DCC Server 3.9.0 está en estado "En ejecución" (Fig. A.10), lo que significa que está iniciado. Es posible detener y reiniciar el servicio desde esta misma ventana seleccionando el servicio y pulsando sobre detener o reiniciar.

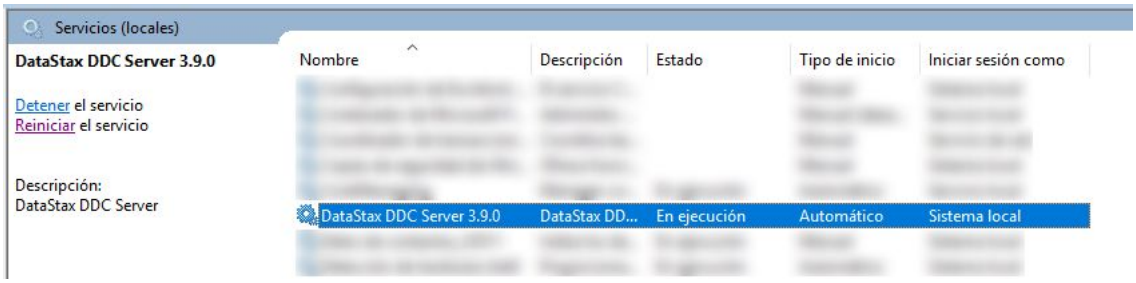


Figura A.10: Estado "En ejecución" del servicio DataStax DCC Server 3.9.0 de Windows

Para establecer una conexión entre el cliente DataStax DevCenter y el servidor DataStax DCC Server hay que abrir el cliente y crear una nueva conexión desde la ventana *Connections* de la interfaz pulsando en el icono "+" o bien con el atajo de teclado *Ctrl+Alt+Shift+N*.

Hay que añadir un nombre a la conexión y los *hosts* con los que se va a realizar la conexión. En este caso el *host* es *localhost* y el puerto es 9042 por ser el puerto por defecto (Fig. A.11). Este valor es configurable con el parámetro *native transport port* desde el fichero de configuración *cassandra.yaml*.

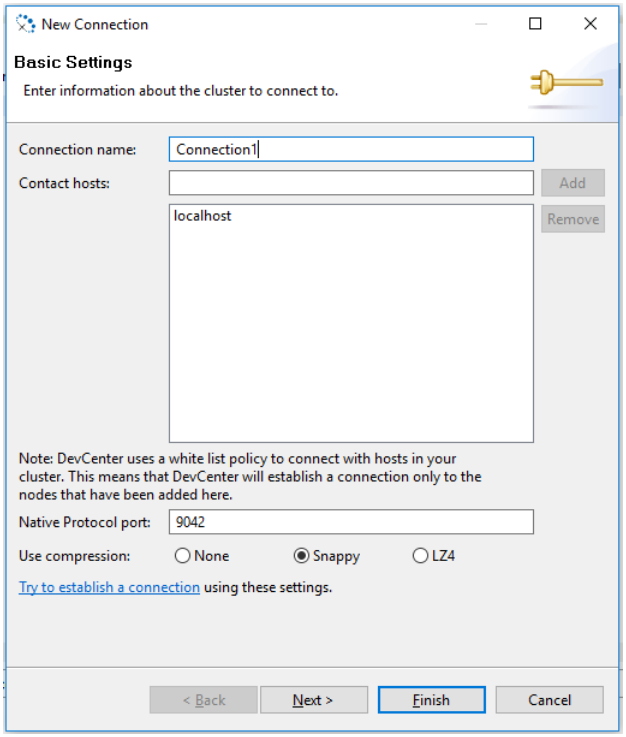


Figura A.11: Ventana de configuración de conexión entre DataStax DevCenter y DataStax DCC Server

Es posible añadir algunas configuraciones de autenticación y seguridad en la siguiente ventana, pero en este caso no se va a implementar. También se puede testear la conexión con el servidor en el host indicado pulsando en *Try to establish a connection*. Tras pulsar en *Finish*, la conexión se genera.

A.1.3. Entorno de Neo4j

A.1.3.1. Instalación de Neo4j

Como se va a trabajar con el cliente Neo4j Desktop no hay que instalar ningún tipo de servidor para Neo4j porque el propio cliente crea una instancia de servidor con cada grafo generado.

El servidor de Neo4j tiene como prerrequisito tener instalado el entorno de ejecución de Java. El *JRE*[4] se obtiene desde la página de *descargas*[5] de Oracle, en el apartado de

[4]Java Runtime Enviroment
[5]Oracle-Java *Downloads*: https://neo4j.com/download-center/

Java. Para descargar el ejecutable de Windows hay que aceptar el acuerdo de licencia y seguidamente pulsar sobre el enlace de descarga del instalador. El proceso de instalación de *JRE* es intuitivo y se basa en aceptar y continuar una serie de ventanas.

A.1.3.2. Instalación de Neo4j Desktop

Para proceder a descargar la interfaz gráfica de usuario oficial de Neo4j se accede a la página oficial de descargas[6] de Neo4j y desde ahí se debe acceder al apartado de Neo4j Desktop (Fig. A.12). Desde éste está accesible el enlace para descargar el ejecutable de Windows para instalar la herramienta. El proceso de instalación es intuitivo y consiste en avanzar en las sucesivas ventanas configurando las opciones que se muestran.

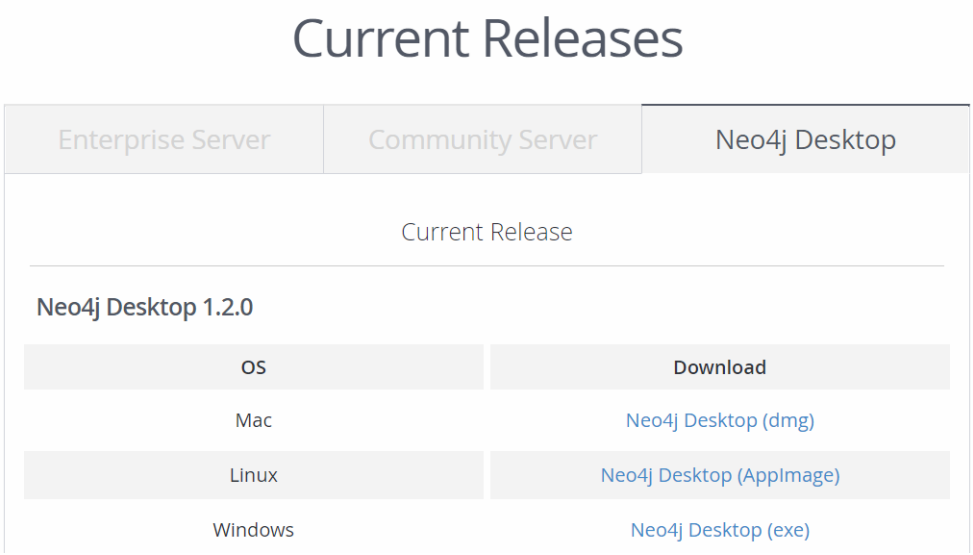


Figura A.12: Vista de la página oficial de descargas de Neo4j, en el apartado de Neo4j Desktop

A.1.3.3. Configuración del entorno Neo4j

Tras abrir Neo4j Desktop aparece un escritorio vacío para crear un nuevo proyecto pulsando en *New*. Sobre este proyecto se generan nuevos grafos (*Add graph*) que en este caso serán grafos locales con un nombre para la base de datos y una contraseña (Fig. A.13). Neo4j Desktop añade unos puertos por defecto a la base de datos creada en función de los puertos que ya ha asignado a otras bases de datos. Pulsando en el botón de *Start* de cada grafo se inicia la base de datos que tiene asociada.

[6]Neo4j *Downloads*: `https://neo4j.com/download-center/`

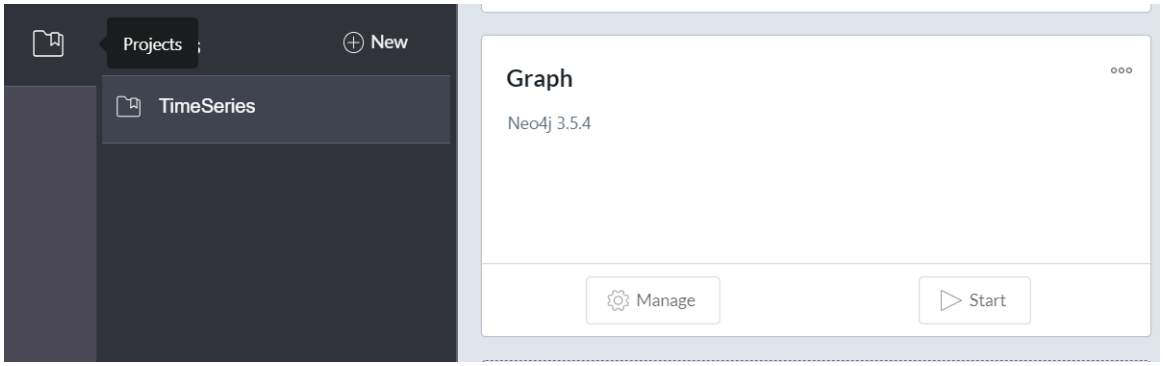


Figura A.13: Vista del escritorio de Neo4j con un proyecto y un grafo generados

A.2. Búsqueda y carga de datos

Dado que los datos que se van a añadir a las bases de datos de cada SGBD son los mismos que se utilizan para realizar el análisis de rendimiento de cada SGBD, el proceso de búsqueda, adaptación y carga de los datos en las diferentes bases de datos se detalla en el apartado 4.1 del siguiente capítulo.

B. ANEXO

Consultas por SGBD

En este anexo se presentan las sentencias de preparación de base de datos y de selección traducidas al lenguaje de consultas de cada sistema gestor de bases de datos utilizado.

Para la preparación se especifican las sentencias de generación de índices identificados en el apartado 4.1.3.

Las consultas de selección se muestran enumeradas en el orden en el que se identifican en los apartados 4.2.1.2 (series temporales) y 4.2.2.2 (datos sobre películas) del capítulo 4.

B.1. Preparación de *TIME_SERIES_DATA*

B.1.1. MongoDB

```
db.sensor_values.createIndex({timestamp: 1})
```

B.1.2. Apache Cassandra

```
CREATE INDEX value_idx ON time_series_data.sensor_values(value)
```

B.1.3. Neo4j

```
CREATE INDEX ON :Value(date)
```

B.2. Preparación de *FILMS_DATA*

B.2.1. MongoDB

```
db.films.createIndex({title:1})
db.films.createIndex({'score.avg':1, title:1})
db.films.createIndex({'cast.actors': "text"})
db.genres_by_film.createIndx({genre: 1, score:1})
db.tags_by_film.createIndex({tag: 1, score:1})
```

B.2.2. Apache Cassandra

```
CREATE INDEX actors_idx ON films_data.films (actors)
CREATE INDEX directors_idx ON films_data.films (directors)
CREATE INDEX locations_idx ON films_data.films (locations)
CREATE INDEX country_idx ON films_data.films (country)
```

B.2.3. Neo4j

```
CREATE INDEX ON :Films (title)
CREATE INDEX ON :Actor (name)
CREATE INDEX ON :Director (name)
CREATE INDEX ON :Genre (value)
CREATE INDEX ON :Tag (value)
CREATE INDEX ON :Location (value)
CREATE INDEX ON :Country (value)
CREATE INDEX ON :Score (avg)
```

B.3. Consultas para *TIME_SERIES_DATA*

B.3.1. MongoDB

B.3.1.1. Versión 1

Las sentencias generadas para la base de datos de la versión 1 presentan cierta complejidad debido a la estructura de los documentos de la colección *SENSOR_VALUES*. No existe un documento para cada valor dado en un *timestamp* concreto, si no que un mismo documento almacena los valores de datos en una franja de una hora. Para realizar consultas de cierta complejidad es necesario utilizar el *framework* de agregación de MongoDB, que permite expresar *pipelines*[1] funcionales que preparan, transforman y analizan los datos por medio de diferentes escenarios que agrupan, enlazan, ordenan y organizan los datos [Walters, 2018].

[1] *Pipeline*: conjunto de elementos de procesamiento de datos conectados en serie, donde la salida de un elemento es la entrada del siguiente

- **Selección**:

1. Consulta nº 1:

```
db.sensor_values.aggregate([
{ $match: {
        timestamp_hour: {$lt: ISODate("2018-12-16T00:00:00Z"),
        $gte: ISODate("2018-12-14T00:00:00Z")}
} },
{ $project: {
        timestamp_hour:1,
        minutes: {$objectToArray: "$sec_values"}
} },
{ $unwind: "$minutes"
},
{ $project: {
        timestamp_hour: 1,
        minute_index: "$minutes.k",
        seconds: {$objectToArray: "$minutes.v"}
} },
{ $unwind: "$seconds"
},
{ $project: {
        reconstructed_date: {$dateFromParts: {
            year: {$year: "$timestamp_hour"},
            month: {$month: "$timestamp_hour"},
            day: {$dayOfMonth: "$timestamp_hour"},
            hour: {$hour: "$timestamp_hour"},
            minute: {$toInt: "$minute_index"},
            second: {$toInt: "$seconds.k"}
        }},
        value: "$seconds.v"
} },
{ $match: {
        reconstructed_date: {$lt: ISODate("2018-12-15T00:20:00Z"),
        $gte: ISODate("2018-12-14T23:00:00Z")}
} },
{ $project: {
     timestamp: "$reconstructed_date",
     engine: "$value.engine"
} }
])
```

2. Consulta nº 2:

```
db.sensor_values.aggregate([
{ $match: {
        timestamp_hour: {$lt: ISODate("2018-12-16T00:00:00Z"),
        $gte: ISODate("2018-12-14T00:00:00Z")}
} },
{
    $project: {
        timestamp_hour:1,
        minutes: {$objectToArray: "$sec_values"}
    }
},
{ $unwind: "$minutes"
},
{ $project: {
        timestamp_hour: 1,
        minute_index: "$minutes.k",
        seconds: {$objectToArray: "$minutes.v"}
} },
{ $unwind: "$seconds"
},
{ $project: {
        reconstructed_date: {$dateFromParts: {
            year: {$year: "$timestamp_hour"},
            month: {$month: "$timestamp_hour"},
            day: {$dayOfMonth: "$timestamp_hour"},
            hour: {$hour: "$timestamp_hour"},
            minute: {$toInt: "$minute_index"},
            second: {$toInt: "$seconds.k"}
        }},
        value: "$seconds.v"
} },
{ $match: {
        reconstructed_date: {$lt: ISODate("2018-12-15T00:20:00Z"),
        $gte: ISODate("2018-12-14T23:00:00Z")}
} },
{ $project: {
     timestamp: "$reconstructed_date",
     engine: "$value.engine"
} },
{ $group: {
        _id: null,
        avg: { $avg: "$engine" },
        min: { $min: "$engine" },
        max: { $max: "$engine" }
} }
])
```

3. Consulta nº 3:

```
db.sensor_values.aggregate([
{ $match: {
        timestamp_hour: {$lt: ISODate("2018-12-16T00:00:00Z"),
        $gte: ISODate("2018-12-14T00:00:00Z")}
} },
{ $project: {
        timestamp_hour:1,
        minutes: {$objectToArray: "$sec_values"}
} },
{ $unwind: "$minutes"
},
{ project: {
        timestamp_hour: 1,
        minute_index: "$minutes.k",
        seconds: {$objectToArray: "$minutes.v"}
} },
{ $unwind: "$seconds"
},
{ $project: {
        reconstructed_date: {$dateFromParts: {
            year: {$year: "$timestamp_hour"},
            month: {$month: "$timestamp_hour"},
            day: {$dayOfMonth: "$timestamp_hour"},
            hour: {$hour: "$timestamp_hour"},
            minute: {$toInt: "$minute_index"},
            second: {$toInt: "$seconds.k"}
        }},
        value: "$seconds.v"
} },
{ $match: {
        reconstructed_date: {$lt: ISODate("2018-12-15T00:20:00Z"),
        $gte: ISODate("2018-12-14T23:00:00Z")}
} },
{ $project: {
     timestamp: "$reconstructed_date",
     engine: "$value.engine"
} },
{ $sort: {
        engine:-1
} },
{ $limit:1
},
{ $project: {
     _id: 0,
     timestamp: 1,
} }
])
```

4. Consulta nº 4:

```
db.sensor_values.aggregate([
{ $match: {
        timestamp_hour: {$lt: ISODate("2018-12-16T00:00:00Z"),
        $gte: ISODate("2018-12-14T00:00:00Z")}
} },
{ $project: {
        timestamp_hour:1,
        minutes: {$objectToArray: "$sec_values"}
} },
{ $unwind: "$minutes"
},
{ $project: {
        timestamp_hour: 1,
        minute_index: "$minutes.k",
        seconds: {$objectToArray: "$minutes.v"}
} },
{ $unwind: "$seconds"
},
{ $project: {
        reconstructed_date: {$dateFromParts: {
            year: {$year: "$timestamp_hour"},
            month: {$month: "$timestamp_hour"},
            day: {$dayOfMonth: "$timestamp_hour"},
            hour: {$hour: "$timestamp_hour"},
            minute: {$toInt: "$minute_index"},
            second: {$toInt: "$seconds.k"}
        }},
        value: "$seconds.v"
} },
{ $match: {
        reconstructed_date: {$lt: ISODate("2018-12-15T00:20:00Z"),
        $gte: ISODate("2018-12-14T23:00:00Z")}
} },
{ $project: {
     timestamp: "$reconstructed_date",
     engine: "$value.engine"
} },
{ $match: {
        engine: {$lt: 20500}
} },
{ $project: {
        _id:0,
        timestamp: 1
} }
])
```

5. Consulta nº 5:

```
db.sensor_values.aggregate([
{ $match: {
        timestamp_hour: {$lt: ISODate("2018-12-16T00:00:00Z"),
        $gte: ISODate("2018-12-14T00:00:00Z")}
  },
{ $project: {
        timestamp_hour:1,
        minutes: {$objectToArray: "$sec_values"}
} },
{ $unwind: "$minutes"
},
{ $project: {
        timestamp_hour: 1,
        minute_index: "$minutes.k",
        seconds: {$objectToArray: "$minutes.v"}
} },
{ $unwind: "$seconds"
},
{ $project: {
        reconstructed_date: {$dateFromParts: {
            year: {$year: "$timestamp_hour"},
            month: {$month: "$timestamp_hour"},
            day: {$dayOfMonth: "$timestamp_hour"},
            hour: {$hour: "$timestamp_hour"},
            minute: {$toInt: "$minute_index"},
            second: {$toInt: "$seconds.k"}
        }},
        value: "$seconds.v"
} },
{ $match: {
        reconstructed_date: {$lt: ISODate("2018-12-15T00:20:00Z"),
        $gte: ISODate("2018-12-14T23:00:00Z")}
} },
{ $project: {
     timestamp: "$reconstructed_date",
     engine: "$value.engine",
     pressure: "$value.pressure"
} }
])
```

- **Inserción**:

6. Consulta nº 6:

```
db.sensor_values.update(
{timestamp_hour: ISODate("2018-12-17T00:00:00Z") },
{ $set: {'sec_values.0.0.engine': 23125} }, { upsert : true } });
```

- **Eliminación**:

  7. Consulta nº 7:

```
db.sensor_values.remove({timestamp_hour:
$lt {ISODate("2018-12-10T20:00:00Z")}})
```

- **Actualización**:

  8. Consulta nº 8:

```
db.sensor_values.update({timestamp_hour:
ISODate("2018-12-15T12:00:00Z")},
{$set: {'sec_values.15.27.engine': 21000}})
```

B.3.1.2. Versión 2

- **Selección**:

  1. Consulta nº 1:

```
db.sensor_values.find( {timestamp:
{$lt:ISODate("2018-12-16T00:00:00Z"),
$gte:ISODate("2018-12-14T00:00:00Z")}}, {_id:0, 'values.engine':1})
```

  2. Consulta nº 2:

```
db.sensor_values.aggregate([{$sort:{'values.engine':-1}},
{$limit:1}, {$project: {_id:0, 'values.engine':1}}]);
db.sensor_values.aggregate([{$sort:{'values.engine':1}},
{$limit:1}, {$project: {_id:0, 'values.engine':1}}]);
db.sensor_values.aggregate([{$group: {
        _id: null,
        avg: { $avg: "$values.engine" }
    }
}]);
```

3. Consulta nº 3:

```
db.sensor_values.aggregate([{$sort:{'values.engine':-1}},
{$limit:1}, {$project: {_id: 0,timestamp: -1}}])
```

4. Consulta nº 4:

```
db.sensor_values.find({'values.engine': {$lte: 20500}},
{_id: 0,timestamp: 1})
```

5. Consulta nº 5:

```
 db.sensor_values.find( {timestamp:
{$lt:ISODate("2018-12-16T00:00:00Z"),
$gte:ISODate("2018-12-14T00:00:00Z")}},
{_id:0, 'values.engine':1, 'values.pressure':1})
```

- **Inserción**:

6. Consulta nº 6:

```
db.sensor_values.update(
{timestamp: ISODate("2018-12-17T00:00:00Z") },
{ $set: {'values.engine': 23125} }, { upsert : true });
```

- **Eliminación**:

7. Consulta nº 7:

```
db.sensor_values.remove({timestamp:
$lt {ISODate("2018-12-10T20:00:00Z")}})
```

- **Actualización**:

8. Consulta nº 8:

```
db.sensor_values.update({timestamp:
ISODate("2018-12-15T12:15:27Z")},
{$set: {'values.engine': 21000}})
```

B.3.2. Apache Cassandra

- **Selección**:

1. Consulta nº 1:

```
SELECT value FROM time_series_data.sensor_values
WHERE cod_sensor = 'engine'
AND timestamp > '2018-12-14 00:00:00'
AND timestamp < '2018-12-15 23:59:59';
```

2. Consulta nº 2:

```
SELECT max(value), min(value), avg(value)
FROM time_series_data.sensor_values
WHERE cod_sensor = 'engine'
AND timestamp > '2018-12-14 00:00:00'
AND timestamp < '2018-12-15 23:59:59';
```

3. Consulta nº 3:

```
SELECT timestamp, max(value)
FROM  time_series_data.sensor_values
WHERE cod_sensor = 'engine'
AND timestamp > '2018-12-14 00:00:00'
AND timestamp < '2018-12-15 23:59:59';
```

4. Consulta nº 4:

```
SELECT timestamp FROM  time_series_data.sensor_values
WHERE cod_sensor = 'engine'
AND value < 20500
ALLOW FILTERING;
```

5. Consulta nº 5:

```
SELECT value FROM time_series_data.sensor_values
WHERE cod_sensor IN ('engine', 'pressure')
AND timestamp > '2018-12-14 00:00:00'
AND timestamp < '2018-12-15 23:59:59';
```

- **Inserción**:

    6. Consulta nº 6:

```
INSERT INTO time_series_data.sensor_values
(cod_sensor, timestamp, value)
VALUES ('engine', '2018-12-17T00:00:00Z', 23152);
```

- **Eliminación**:

    7. Consulta nº 7:

```
DELETE FROM time_series_data.sensor_values
WHERE cod_sensor = 'engine'
AND timestamp < '2018-12-10 20:00:00';
```

- **Actualización**:

    8. Consulta nº 8:

```
UPDATE time_series_data.sensor_values
SET value = 21000
WHERE cod_sensor = 'engine'
AND timestamp = '2018-12-15 12:15:27';
```

B.3.3. Neo4j

- **Selección**:

    1. Consulta nº 1:

```
MATCH (v:Value {sensor: 'engine'})
WHERE v.date > datetime('2018-12-14T00:00:00Z')
AND v.date < datetime('2018-12-15T23:59:59Z')
RETURN v.data;
```

2. Consulta nº 2:

```
MATCH (v:Value {sensor: 'engine'})
WHERE v.date > datetime('2018-12-14T00:00:00Z')
AND v.date < datetime('2018-12-15T23:59:59Z')
RETURN max(v.data), min(v.data), avg(v.data);
```

3. Consulta nº 3:

```
MATCH (v:Value {sensor: 'engine'})
WHERE v.date > datetime('2018-12-14T00:00:00Z')
AND v.date < datetime('2018-12-15T23:59:59Z')
WITH max(v.data) AS max_value
MATCH (b:Value {data: max_value})
RETURN b.date;
```

4. Consulta nº 4:

```
MATCH (v:Value {sensor: 'engine'})
WHERE v.data < 20500
RETURN v.date;
```

5. Consulta nº 5:

```
MATCH (v:Value)
WHERE v.date > datetime('2018-12-14T00:00:00Z')
AND v.date < datetime('2018-12-15T23:59:59Z')
AND  (v.sensor='engine' OR v.sensor='pressure')
RETURN v.sensor, v.data;
```

- **Inserción**:

6. Consulta nº 6:

```
MATCH (a:Value {sensor: 'engine'})
WHERE a.date = datetime('2018-12-16T23:59:59Z')
CREATE (b:Value {sensor: 'engine', data: 23152})
SET b.date = datetime('2018-12-17T00:00:00Z')
CREATE (a)-[:NEXT]->(b)
```

- **Eliminación**:

    7. Consulta nº 7:

```
MATCH (v:Value {sensor: 'engine'})
WHERE v.date < datetime('2018-12-10T20:00:00Z')
DETACH DELETE v;
```

- **Actualización**:

    8. Consulta nº 8:

```
MATCH (v:Value {sensor: 'engine'})
WHERE v.date = datetime('2018-12-12T12:15:27Z')
SET v.data = 21000;
```

B.4. Consultas para *FILMS_DATA*

B.4.1. MongoDB

- **Selección**:

    1. Consulta nº 1:

```
db.films.find({},{title: 1, _id: 0}).sort({'score.avg':-1}).limit(1);
```

    2. Consulta nº 2:

```
db.genres_by_film.find({genre:'Adventure'},{title:1,_id:0});
```

    3. Consulta nº 3:

```
db.genres_by_film.find({genre:'Adventure'},{title:1,_id:0})
.sort({score:-1}).limit(1);
```

4. Consulta nº 4:

```
db.genres_by_film.find({genre:'Adventure'},{title:1,_id:0})
.sort({score:-1}).limit(10);
```

5. Consulta nº 5:

```
db.films.find({$text: {$search: "Johnny Depp"}}, {title: 1, _id: 0});
```

6. Consulta nº 6:

```
db.films.find({title: 'Life of Brian'},{'cast.actors': 1, _id: 0});
```

- **Inserción**:

7. Consulta nº 7:

```
db.films.insert({title:"AAA", info: {year: "1234", image: "miURL"},
characteristics: {genres: "Adventure"}, cast: {actors: "a1, a2,
a3", directors: "dir"}, location: {country: "USA"}, score: {count:
5, sum: 5, avg: 1}});

db.genres_by_film.insert({genre: "Adventure", score: 1
, title: "AAA"});
```

- **Eliminación**:

8. Consulta nº 8:

```
db.films.remove({title: "AAA"});
db.genres_by_film.remove({genre: "Adventure", score: 1, title: "AAA"});
```

- **Actualización**:

9. Consulta nº 9:

```
db.films.update({title: 'Life of Brian'},{year: 2001});
```

B.4.2. Apache Cassandra

- **Selección**:

1. Consulta nº 1:

```
SELECT title, max(score) FROM films_data.genres_by_film;
```

2. Consulta nº 2:

```
SELECT title FROM films_data.genres_by_film
WHERE genre = 'Adventure';
```

3. Consulta nº 3:

```
SELECT title, max(score) FROM films_data.genres_by_film
WHERE genre='Adventure';
```

4. Consulta nº 4:

```
SELECT title FROM films_data.genres_by_film
WHERE genre = 'Adventure'
ORDER BY score DESC LIMIT 10;
```

5. Consulta nº 5:

```
SELECT title FROM films_data.films
WHERE actors CONTAINS 'Johnny Depp' ALLOW FILTERING;
```

6. Consulta nº 6:

```
SELECT actors FROM  films_data.films
WHERE title = 'Life of Brian';
```

- **Inserción**:

  7. Consulta nº 7:

```
INSERT INTO films_data.films
(title, year, image, genres, actors, directors, country, score,
num, total)
VALUES ('AAA', '1234', 'miURL', {'Adventure'}, {'a1', 'a2', 'a3'},
{'dir'},'USA',1,5,5);

INSERT INTO films_data.genres_by_film
(genre, score, title)
VALUES ('Adventure', 1, 'AAA');
```

- **Eliminación**:

  8. Consulta nº 8:

```
DELETE FROM films_data.films
WHERE title = 'AAA';

DELETE FROM films_data.genres_by_film
WHERE genre='Adventure'
AND score=1 AND title = 'AAA';
```

- **Actualización**:

  9. Consulta nº 9:

```
UPDATE films_data.films
SET year = '2001' WHERE title = 'Life of Brian';
```

B.4.3. Neo4j

- **Selección**:

  1. Consulta nº 1:

```
MATCH (s:Score)
WITH max(s.avg) AS max_score
MATCH (f:Film)-[]->(b:Score) WHERE b.avg = max_score
RETURN f.title LIMIT 1;
```

2. Consulta nº 2:

```
MATCH (f:Film)-[]-(g:Genre {value: 'Adventure'})
```

3. Consulta nº 3:

```
MATCH (g:Genre {value: 'Adventure'})-[]-(f:Film)-[]->(s:Score)
WITH max(s.avg) AS max_score
MATCH (r:Film)-[]->(b:Score)
WHERE b.avg = max_score
RETURN r.title;
```

4. Consulta nº 4:

```
MATCH (g:Genre {value: 'Adventure'})<-[]-(f:Film)-[]->(s:Score)
RETURN f.title ORDER BY s.avg DESC LIMIT 10;
```

5. Consulta nº 5:

```
MATCH (f:Film)-[]->(a:Actor {name: 'Johnny Depp'})
RETURN f.title;
```

6. Consulta nº 6:

```
MATCH (a:Actor)-[]-(f:Film {title: 'Life of Brian'})
RETURN a.name;
```

- **Inserción**:

  7. Consulta nº 7:

```
PROFILE CREATE (f:Film {title: 'AAA', year: 1234, image: 'miURL'})
MERGE(c:Country {value: 'USA'})
MERGE (f)-[:IS_FROM]->(c)
MERGE(s:Score {sum: 5, count: 5, avg: 5})
MERGE (f)-[:HAS_SCORE]->(s)
WITH f, 'a1|a2|a3' AS actors, 'dir' AS directors,
'Adventure' AS genres
FOREACH (a in split(actors,'|') |
MERGE(y:Actor {name: a})
MERGE (f)-[:ACTS]-(y))
WITH f, directors, genres
FOREACH (d in split(directors,'|') |
MERGE(z:Director {name: d})
MERGE (f)-[:DIRECTED]->(z))
WITH f, genres
FOREACH (g in split(genres, '|') |
MERGE(x:Genre {value:g})
MERGE (f)-[:HAS_GENRE]->(x))
```

- **Eliminación**:

  8. Consulta nº 8:

```
MATCH (f:Film {title: 'AAA'}) DETACH DELETE f
```

- **Actualización**:

  9. Consulta nº 9:

```
MATCH (f:Film {title: 'Life of Brian'}) SET f.year = 2001;
```

C. ANEXO

Diagramas de secuencia de las funcionalidades de la aplicación web

En este punto se anexan los diferentes diagramas de secuencia asociados a las diferentes funcionalidades que ofrece la aplicación web desarrollada.

- **Ver noticias**: figura C.1
- **Ver ranking**: figura C.2
- **Escribir contacto**: figura C.3
- **Ver información**: figura C.4
- **Ver perfil**: figura C.5
- **Consultar progresos**: figura C.6
- **Ver descargas**: figura C.7
- **Ver quizz**: figura C.8
- **Ver actividad**: figura C.9
- **Consultar cuenta**: figura C.10
- **Iniciar sesión**: figura C.11
- **Registrarse**: figura C.12
- **Cerrar sesión**: figura C.13

En los diagramas cuya funcionalidad representada puede ser hecha tanto por usuarios autenticados como no autenticados, el color rojo en las diferentes flechas indica acciones que se realizan en el caso de los usuarios autenticados únicamente.

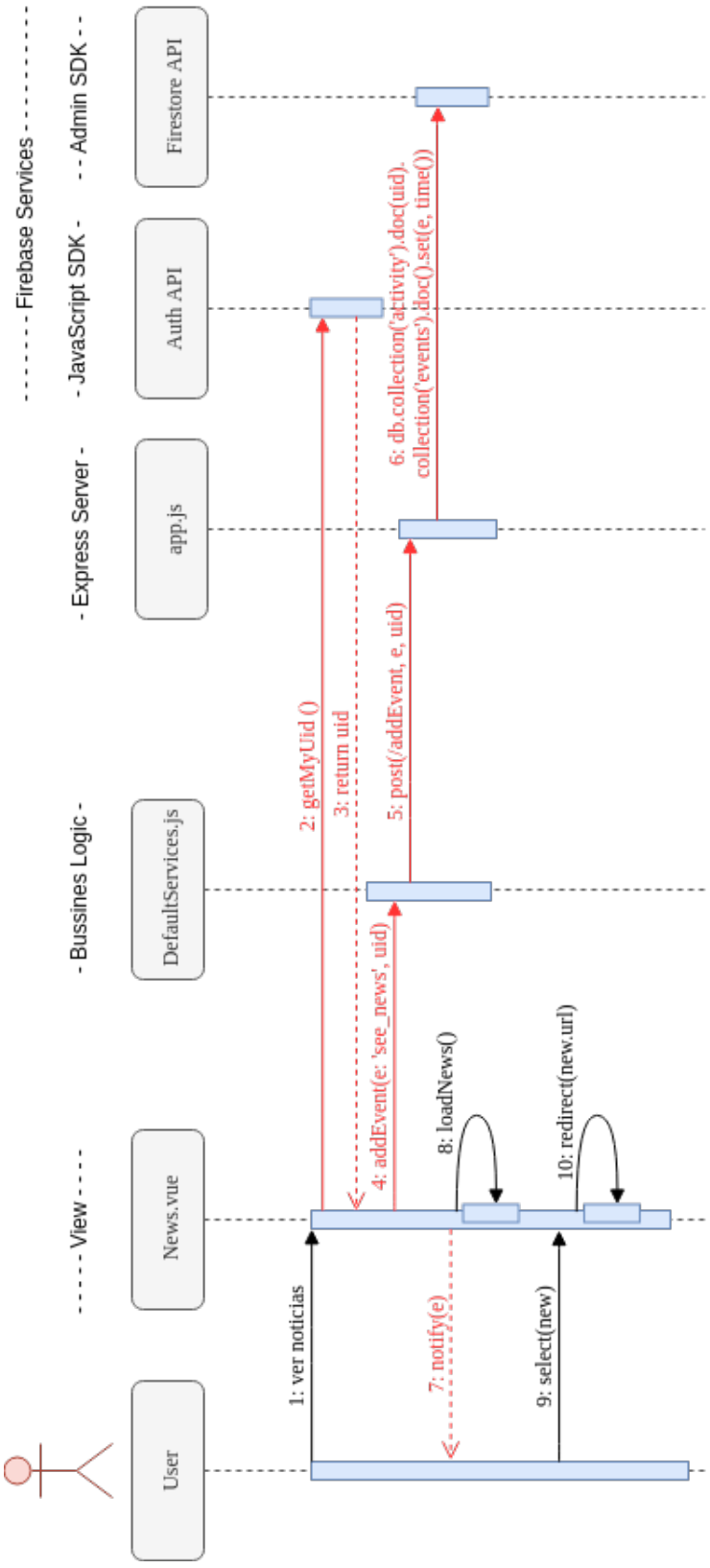


Figura C.1: Diagrama de secuencia de la funcionalidad *Ver noticias*

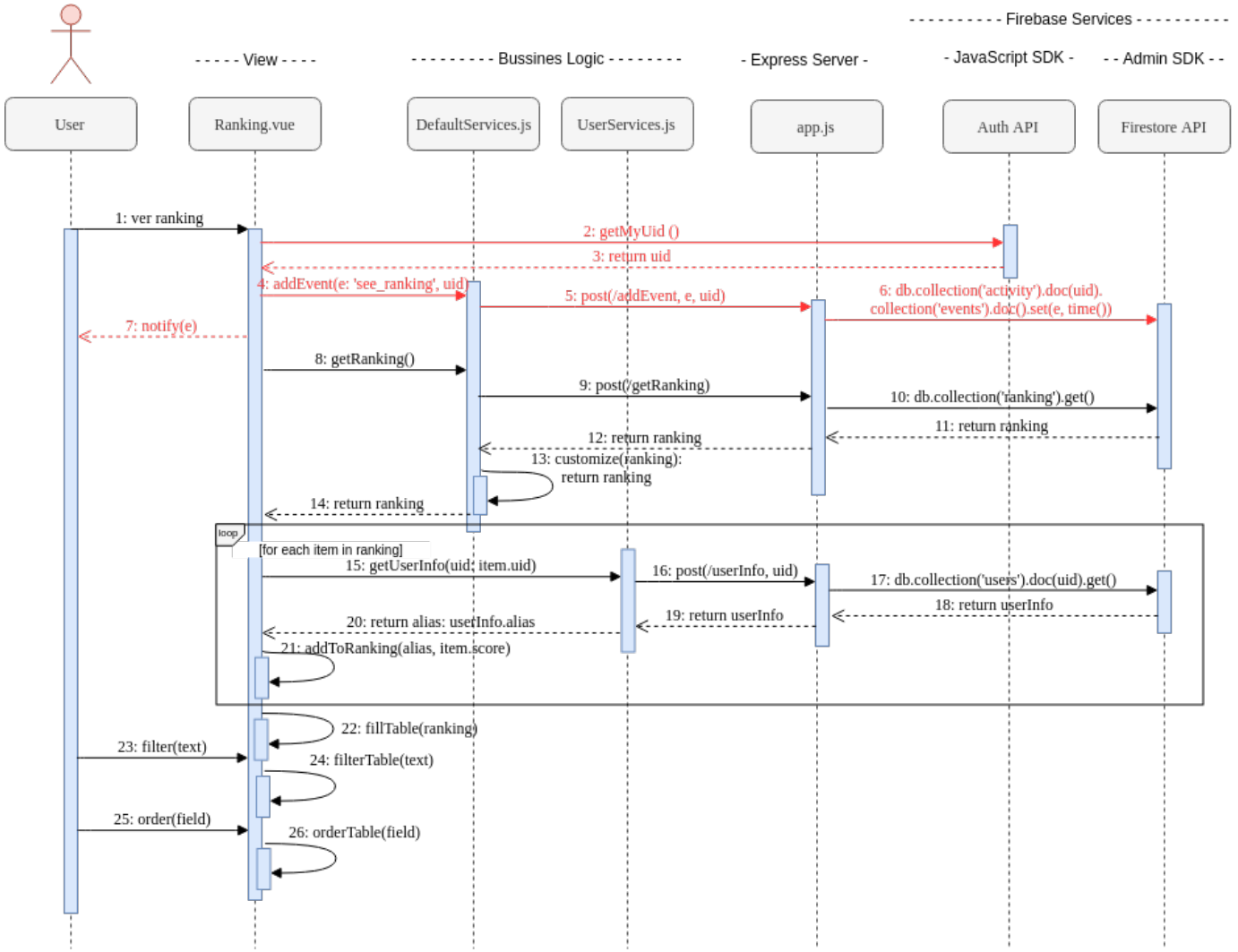


Figura C.2: Diagrama de secuencia de la funcionalidad *Ver ranking*

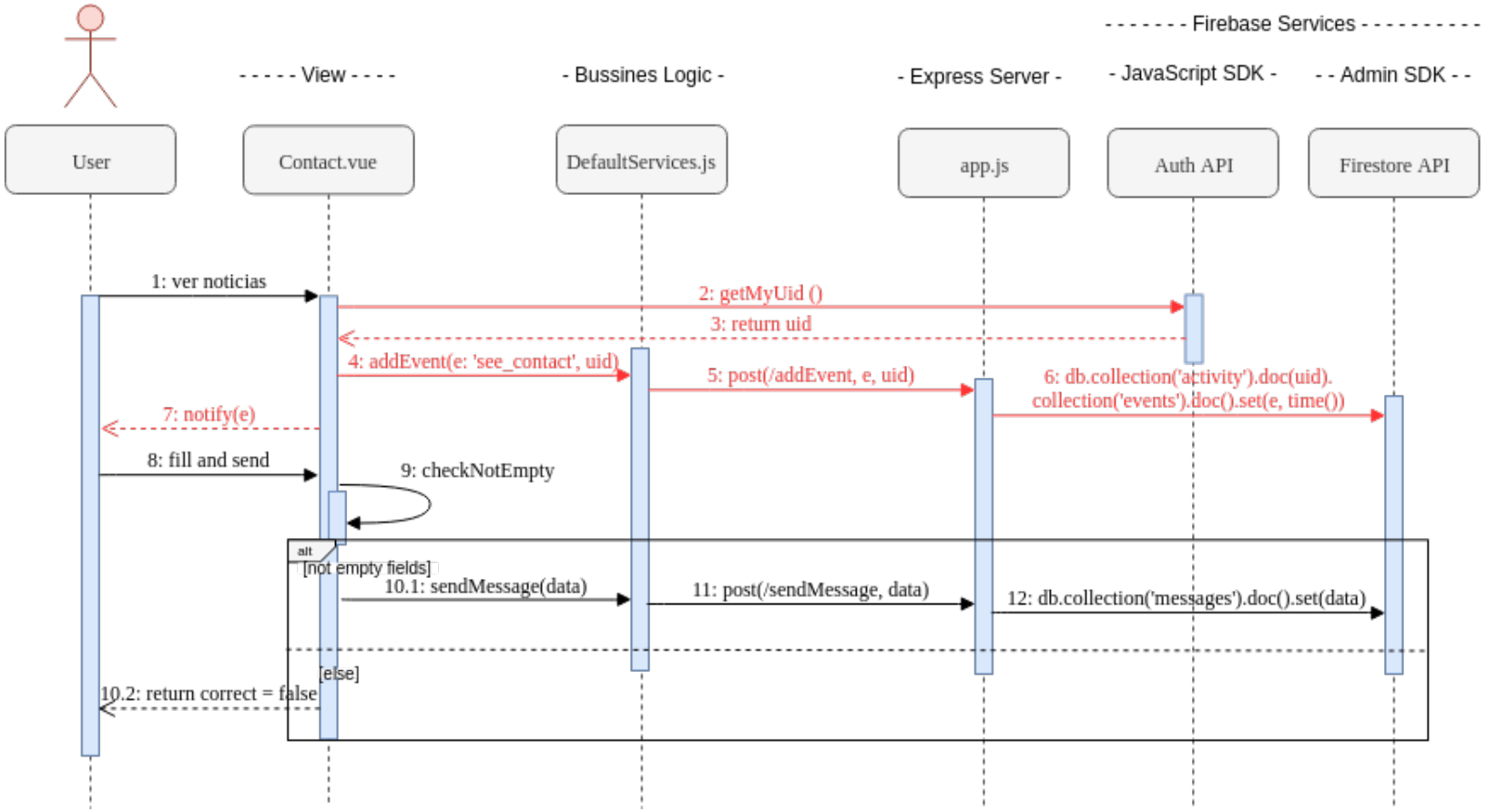


Figura C.3: Diagrama de secuencia de la funcionalidad *Escribir contacto*

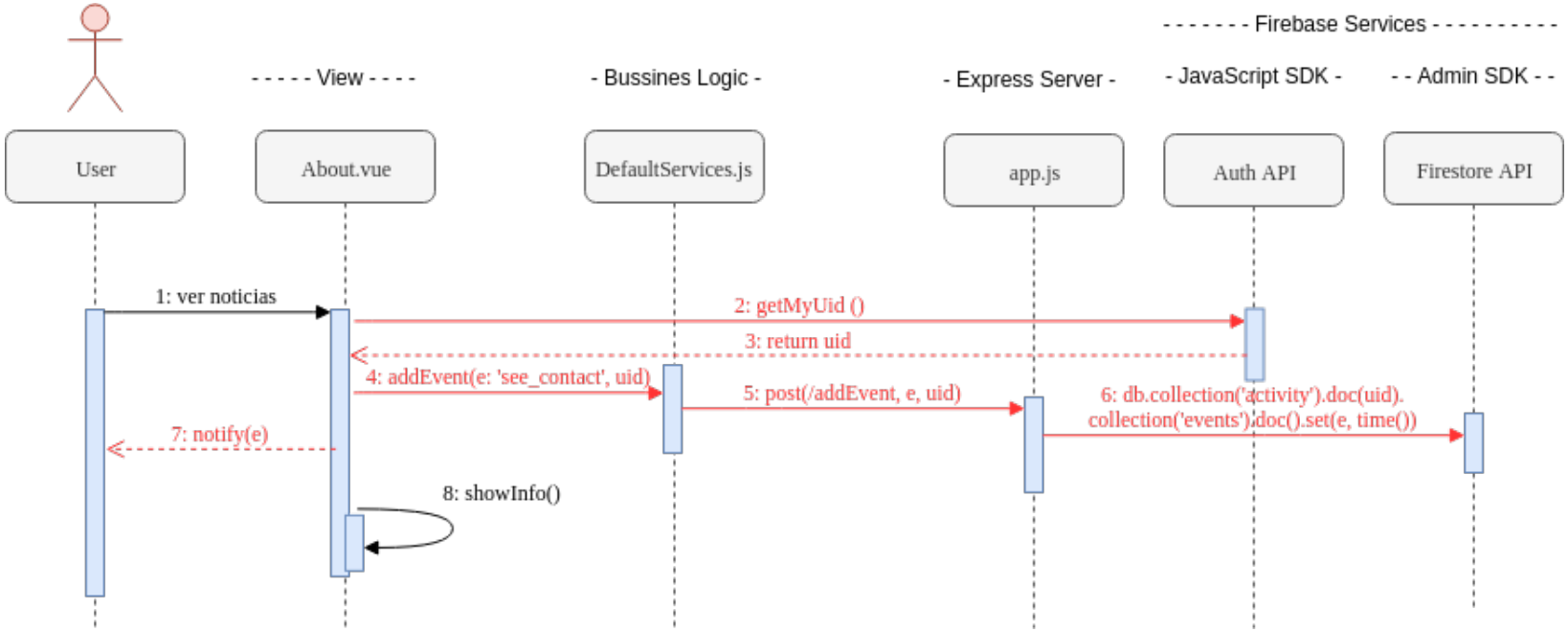


Figura C.4: Diagrama de secuencia de la funcionalidad *Ver información*

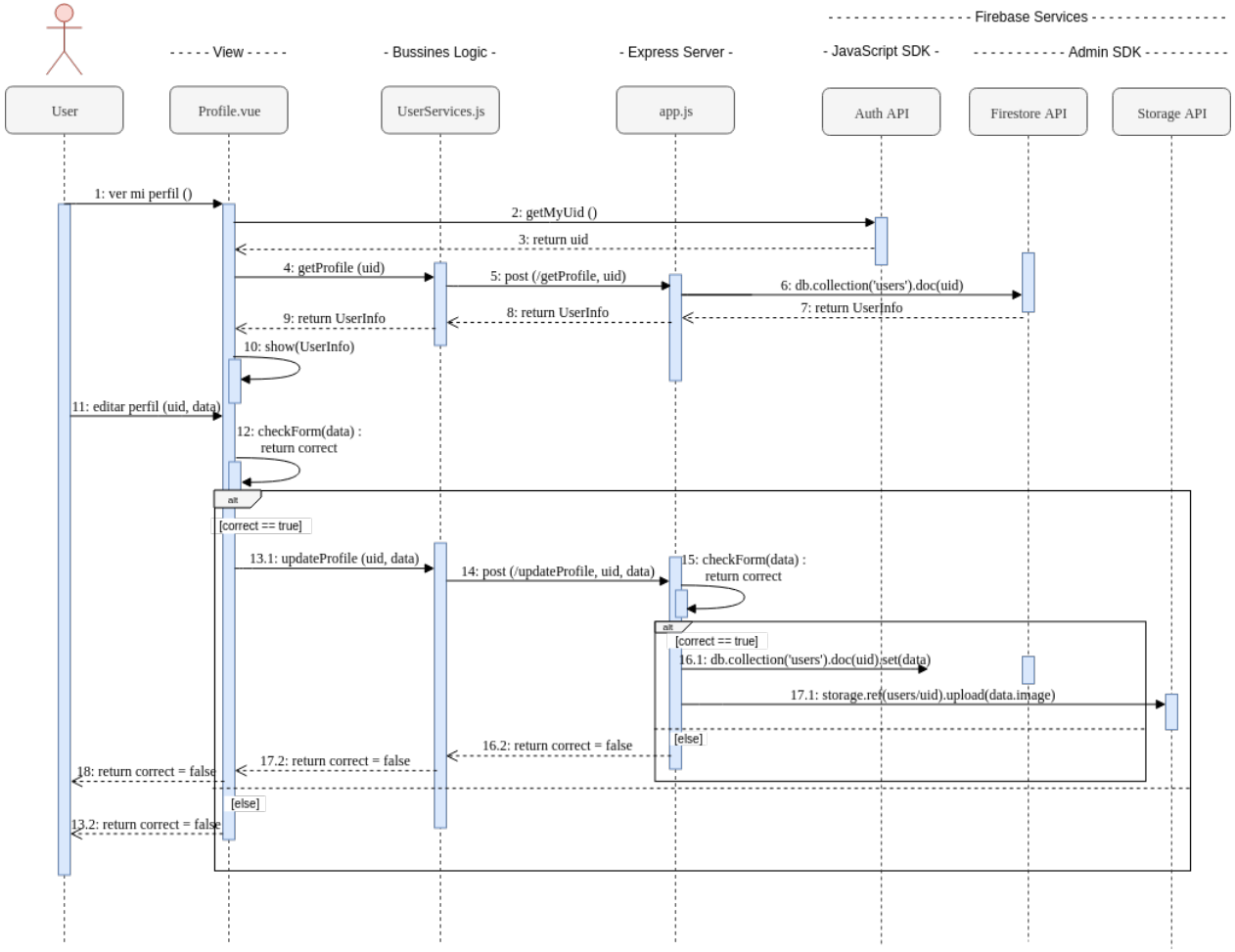


Figura C.5: Diagrama de secuencia de la funcionalidad *Ver perfil*

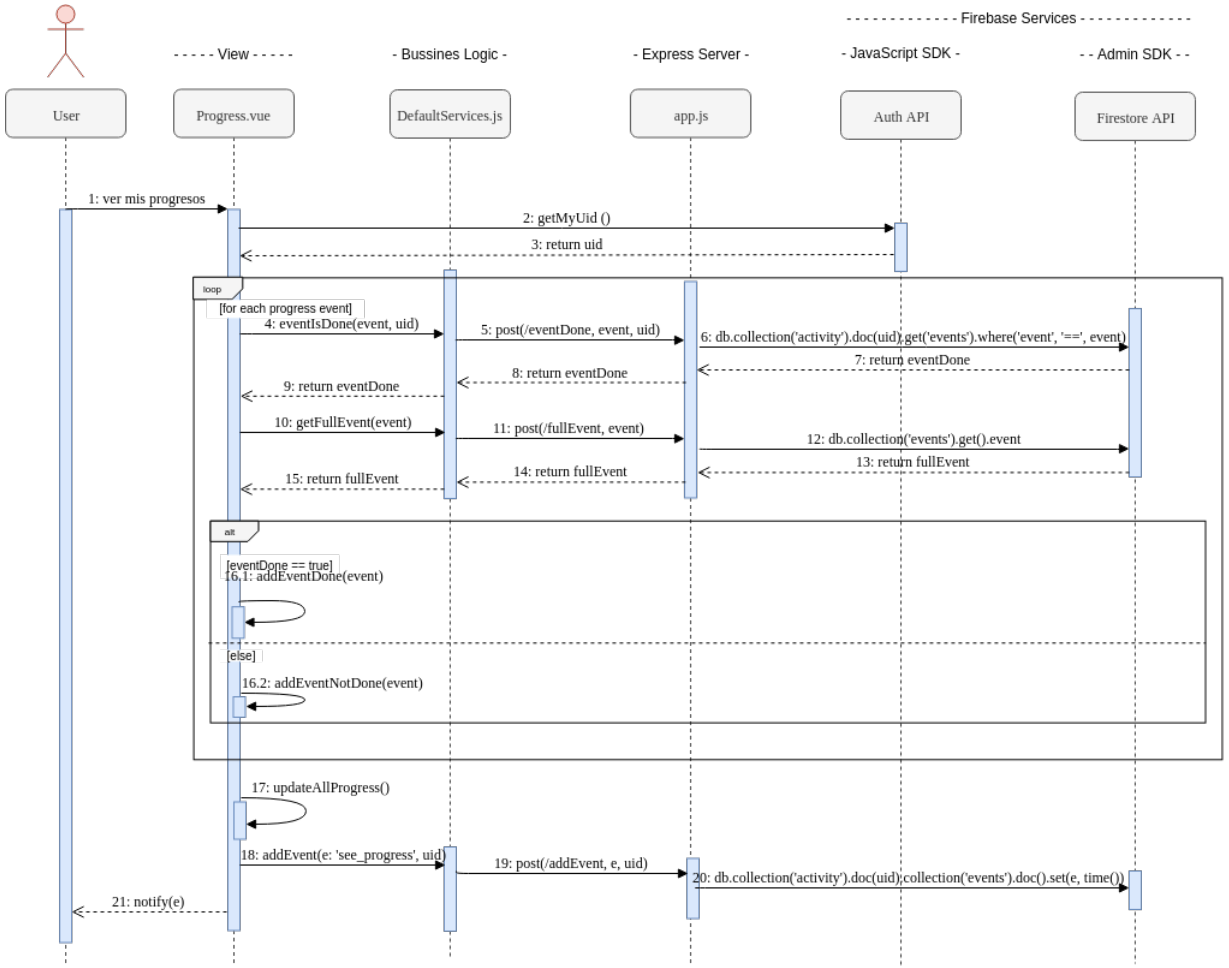


Figura C.6: Diagrama de secuencia de la funcionalidad *Consultar progresos*

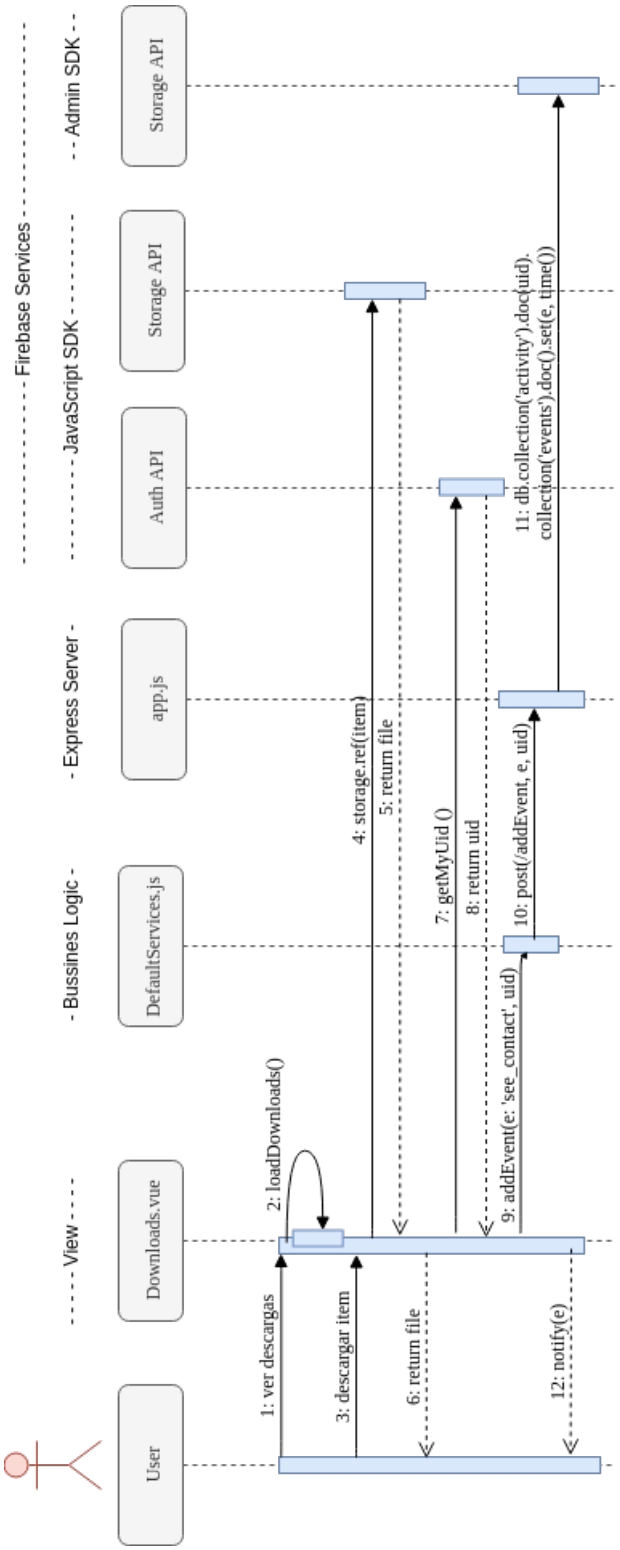


Figura C.7: Diagrama de secuencia de la funcionalidad *Ver descargas*

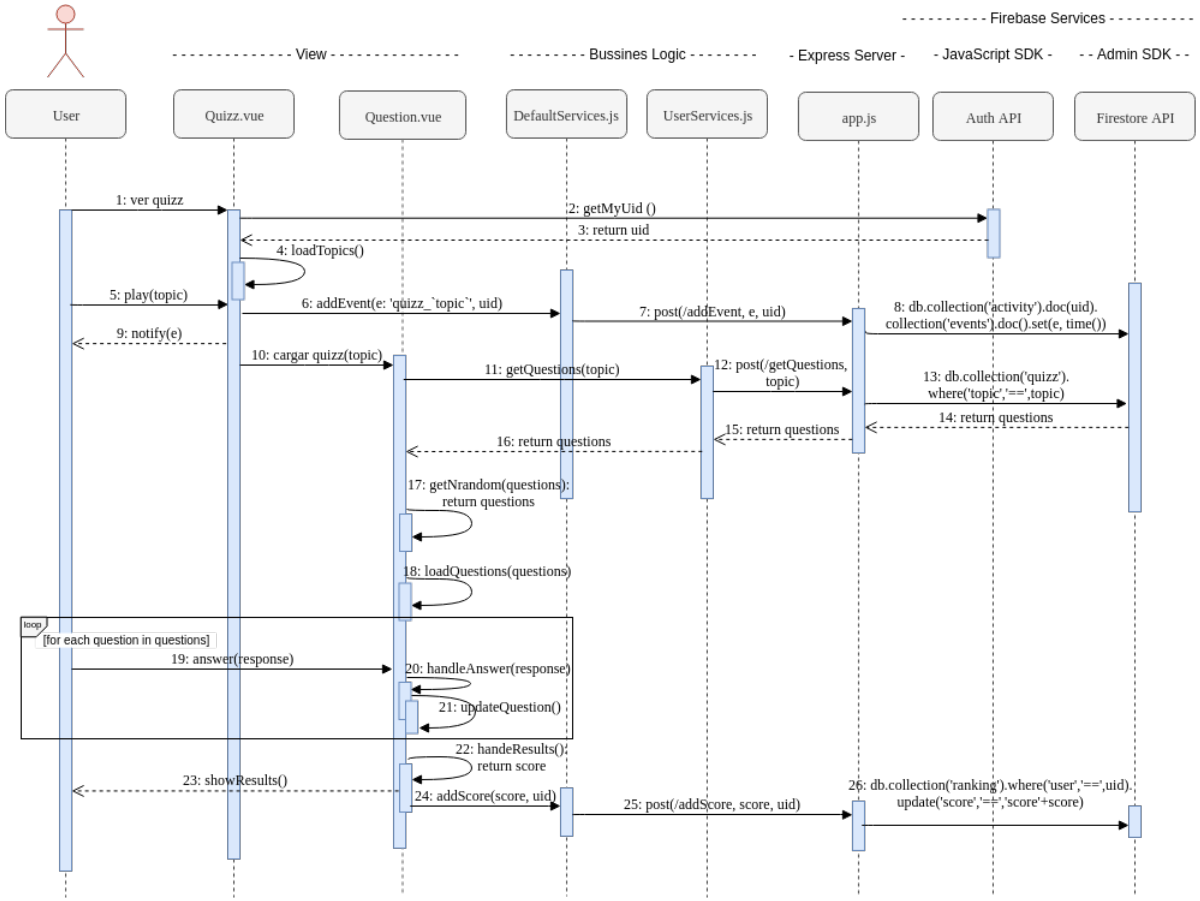


Figura C.8: Diagrama de secuencia de la funcionalidad *Ver quizz*

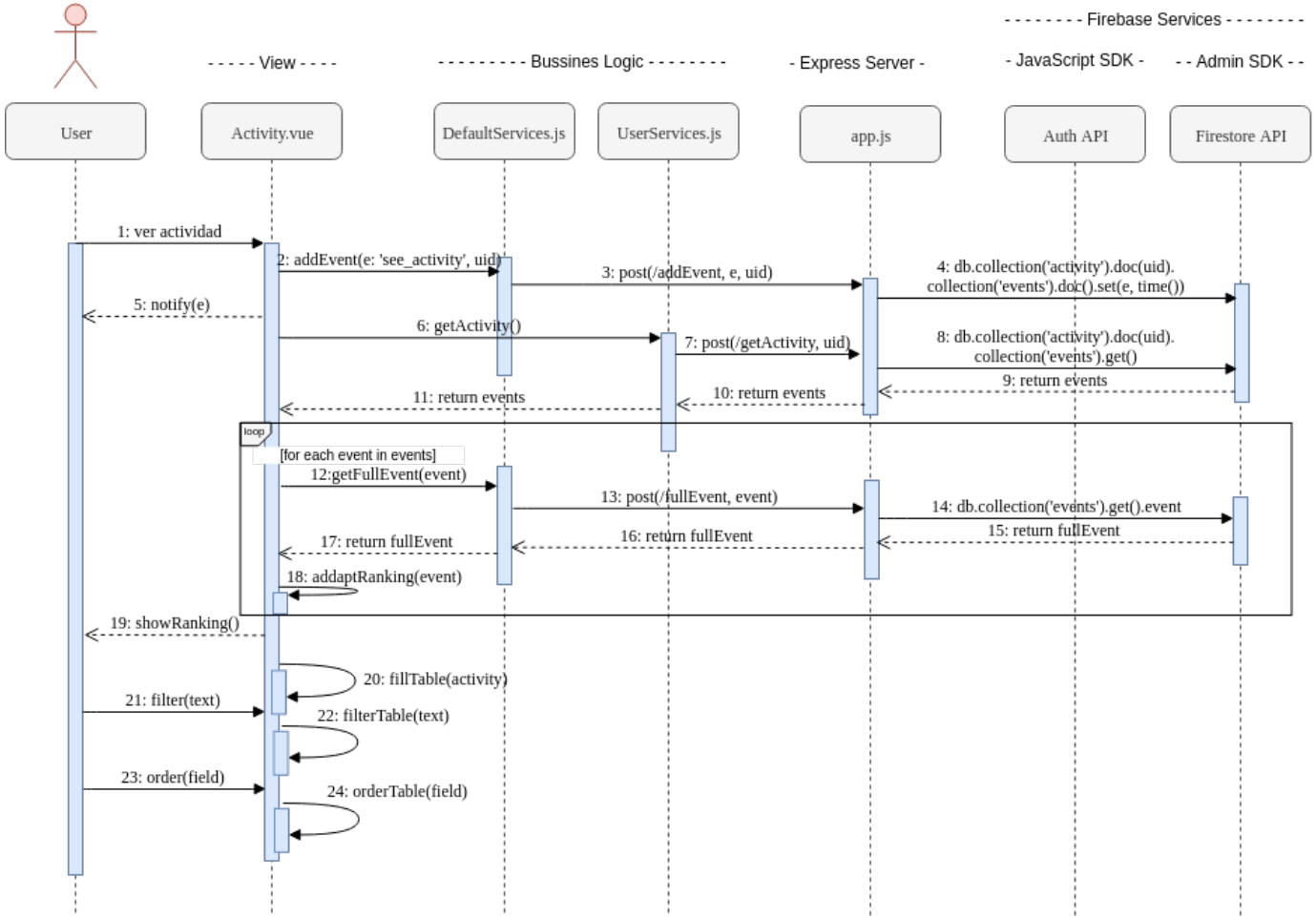


Figura C.9: Diagrama de secuencia de la funcionalidad *Ver actividad*

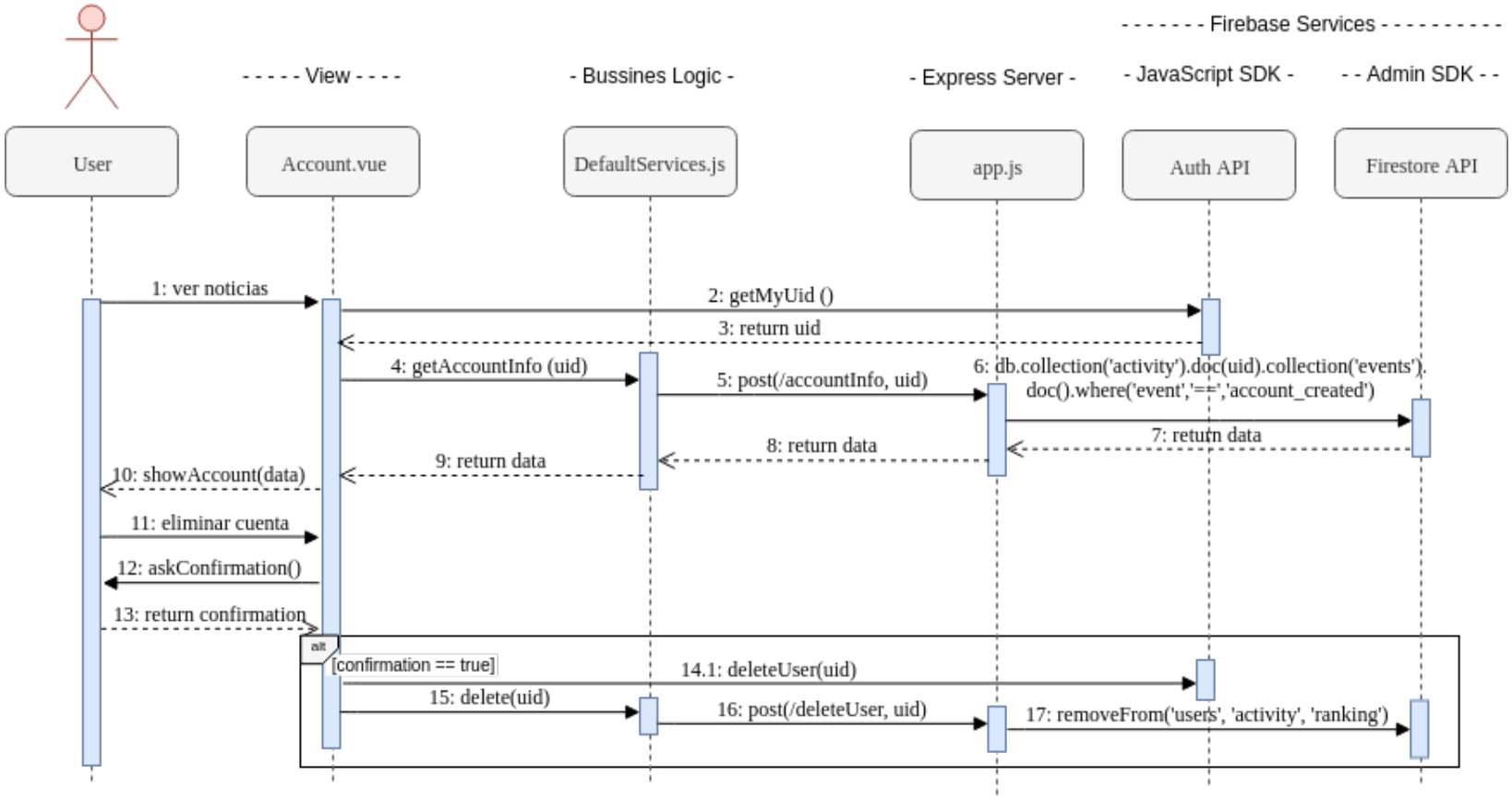


Figura C.10: Diagrama de secuencia de la funcionalidad *Consultar cuenta*

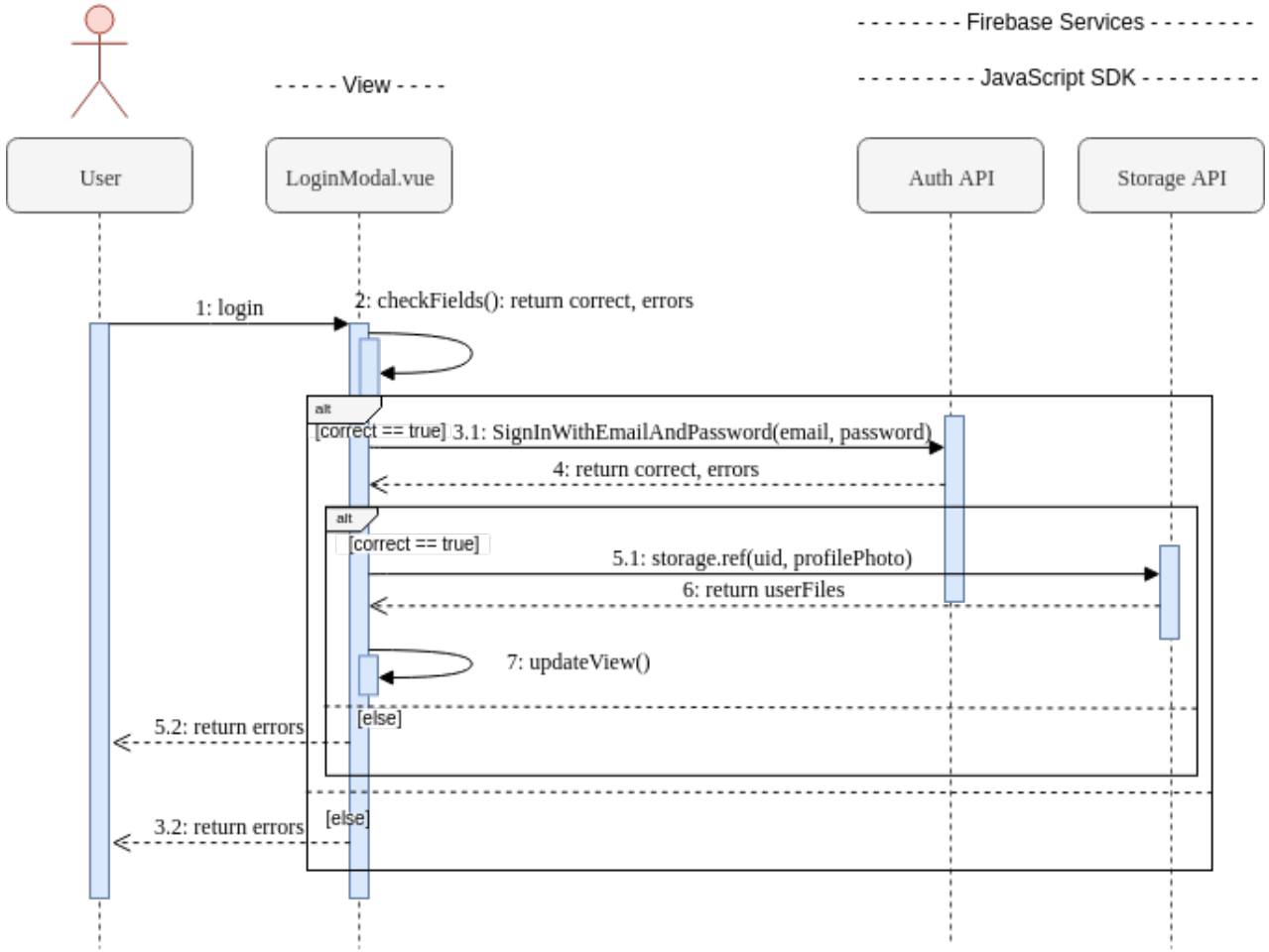


Figura C.11: Diagrama de secuencia de la funcionalidad *Iniciar sesión*

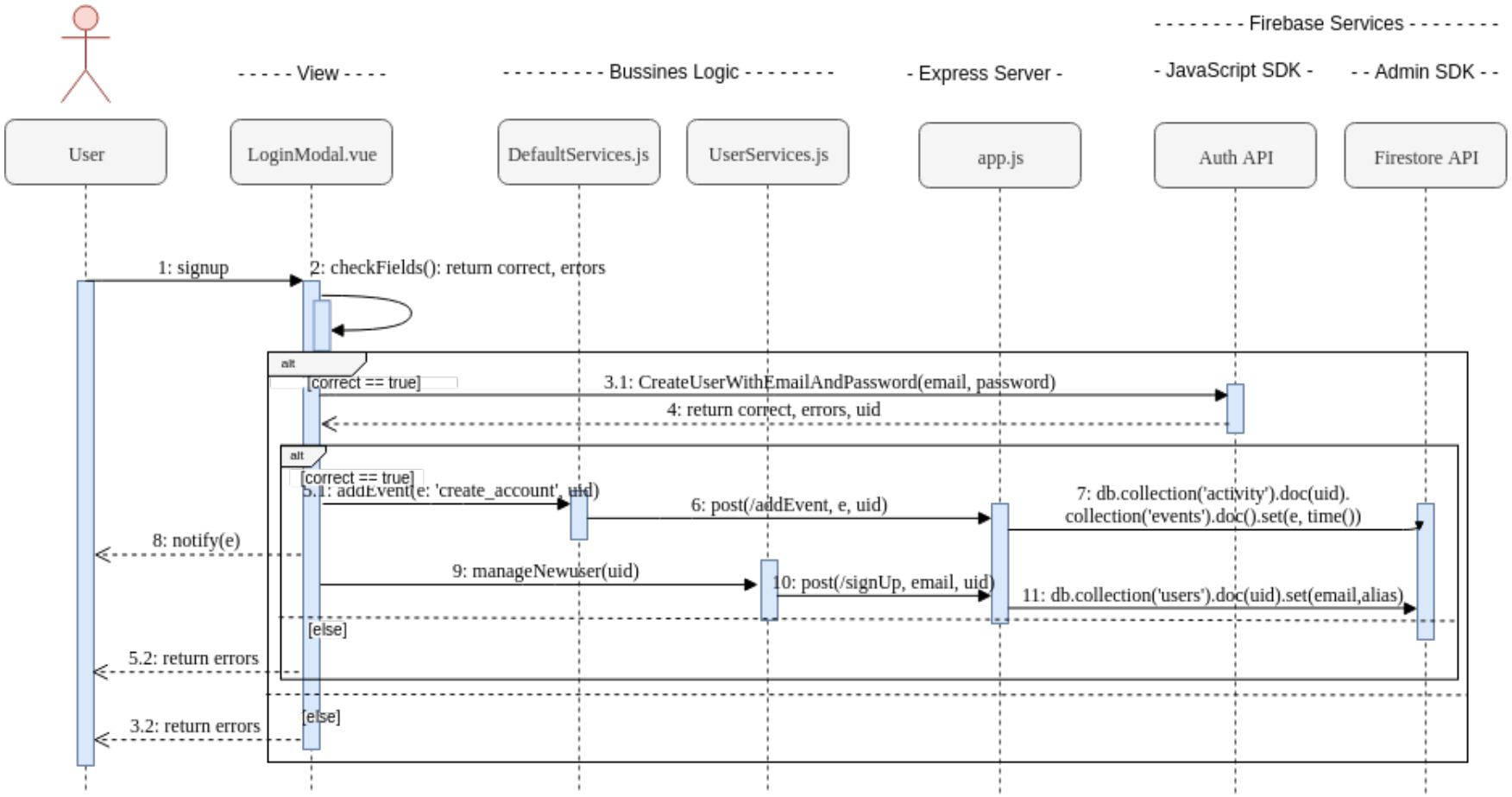


Figura C.12: Diagrama de secuencia de la funcionalidad *Registrarse*

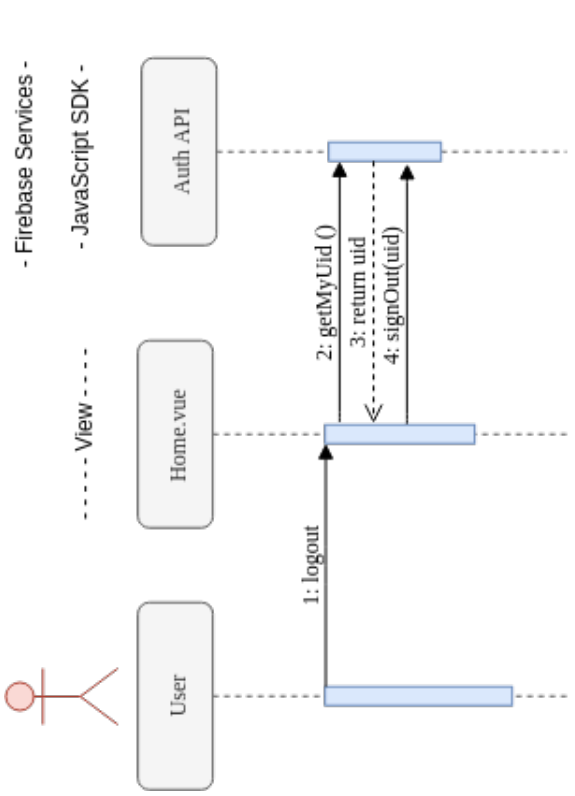


Figura C.13: Diagrama de secuencia de la funcionalidad *Cerrar sesión*

D. ANEXO

Preparación del entorno de desarrollo

En este anexo se detallan los procesos de configuración del entorno de desarrollo sobre el que se va a generar la aplicación web. Este desarrollo se lleva a cabo previamente a la generación de la aplicación, proceso descrito en el apartado 5 de este trabajo de fin de grado.

Dado que el desarrollo del proyecto se realiza en local, se genera un directorio donde se va a alojar todo el contenido de la aplicación web. Este directorio se añade al *workspace* del editor de código utilizado (*Visual Studio Code*) para empezar a trabajar con él sobre el proyecto. Para mantener un control de versiones y una copia de seguridad del proyecto, se genera un nuevo repositorio privado en *GitHub* y a través de la consola del editor de código se vincula el directorio generado con el nuevo repositorio (Cód. D.1).

```
$ apt-get install git
$ git init
$ git commit -m "first commit"
$ git remote add origin https://github.com/******/NoSQL.git
$ git push -u origin master
```

Código D.1: Configuración del repositorio en GitHub

El entorno de ejecución con el que se va a trabajar es *Node.js*, un entorno diseñado para trabajar con JavaScript. La instalación de *Node.js* (Cód. D.2) se realiza a través del repositorio oficial de *Node.js* con el fin de obtener una versión más reciente. La instalación se realiza instalando un *PPA* (*Personal Package Archive*) con la versión que se quiere instalar.

Como sistema de gestión de paquetes se utiliza *npm*, que trabaja por defecto con *Node.js*. El paquete de *Node.js* instala también el binario de *npm*, por lo que no hace falta realizar ninguna instalación adicional.

```
$ curl -sL https://deb.nodesource.com/setup_10.x -o nodesource_setup.sh
$ sudo bash nodesource_setup.sh
$ sudo apt install nodejs
$ node --version # v10.16.0
$ npm --version # 6.9.0
$ npm init
```

Código D.2: Instalación de *Node.js* y *npm*, y creación de proyecto

Con el comando *npm init* se genera automáticamente el fichero *package.json* que contiene registros con las dependencias del proyecto y otras características asociadas al proyecto como el nombre, versión, repositorio, autores, licencia, etc.

D.1. Configuración del proyecto *Vue.js*

La principal tecnología utilizada para generar la parte cliente de la aplicación es *Vue.js*, un *framework* progresivo de JavaScript. La instalación de *Vue.js* se realiza a través del gestor de paquetes *npm*. Además, se debe instalar *vue-router*[1], que se trata del enrutador oficial de *Vue.js*, permitiendo realizar transiciones entre vistas dentro de la misma página, dado que las aplicaciones de *Vue.js* son SPA (*Single Page Applications*).

Para que el navegador pueda entender el código *Vue.js* hay que convertir el código con una herramienta de compilación (*builder*) como es *webpack*. *Webpack* coloca en un grafo de dependencias todos los elementos que forman parte del proyecto, y se encarga de procesar los archivos de entrada convirtiéndolos en otros de salida utilizando *loaders*[2] [Alarcón, 2017]. Los *loaders* utilizados en este proyecto son *vue-loader*[3] (permite generar componentes de *Vue.js*), *vue-template-compiler*[4] (funciona conjuntamente con *vue-loader* y precompila los componentes con extensión *vue*), *vue-style-loader* y *css-loader*[5] (precompilan y cargan los ficheros de estilo *css*).

[1] *vue-router*: https://router.vuejs.org/

[2] *Loader*: en el contexto de *webpack* se tratan de elementos que preprocesan y cargan diferentes tipos de archivos

[3] *vue-loader*: https://vue-loader.vuejs.org/

[4] *vue-template-compiler*: https://www.npmjs.com/package/vue-template-compiler

[5] *vue-style-loader* y *css-loader*: https://vue-loader.vuejs.org/guide/css-modules.html

Otra dependencia importante a instalar es *babel* (*babel-loader* y *babel-core*). *Babel* es una dependencia que traduce el código JavaScript a una versión compatible por la mayoría de los navegadores.

Dado que la aplicación va a actuar como un cliente HTTP, se instala una librería que se encargue de gestionar las conexiones con el servidor web de destino. *Axios* es una librería de JavaScript que facilita la comunicación con un servidor web a través de promesas[6].

A la hora de instalar estos paquetes conviene identificar de qué tipo de módulos se tratan, es decir, si son módulos directamente relacionados con la aplicación final y son necesarios para producirla, o si por el contrario se tratan de módulos que sirven para el desarrollo del proyecto. *Vue*, *vue-router* y *axios* son módulos que la aplicación siempre va a necesitar para su correcto funcionamiento. Sin embargo, *webpack*, los *loaders* y *babel* son módulos directamente relacionados con el desarrollo de la aplicación. Esta diferencia debe reflejarse en el modo de instalación de los paquetes, donde se debe añadir *-dev* en aquellos paquetes destinados al desarrollo del proyecto (Cód. D.3).

```
$ npm install vue vue-router --save # vue
$ npm install webpack webpack-cli --save-dev # webpack
$ npm install vue-loader vue-template-compiler vue-style-loader
  css-loader --save-dev # loaders
$ npm install babel-core babel-loader --save-dev # babel
$ npm install axios --save # axios
```

Código D.3: Instalación de *Vue.js* y configuración inicial

El principio de desarrollo de la aplicación comienza con la creación de un proyecto *Vue.js* en el directorio principal (*vue create .*), de manera que se generan una serie de ficheros y directorios en el proyecto. Dado que el proyecto se ha configurado con *webpack*, en el fichero de configuración *package.json* se indican los *scripts* de construcción de la aplicación (Cód. D.2) y se modifica el esquema del directorio para adecuarse a la configuración de *webpack* (figura D.1):

- **build**: En este directorio se encuentran los ficheros de configuración de *webpack* y los asociados a los diferentes *loaders* con los que trabaja.
- **dist**: Directorio donde se aloja la aplicación web producida después de compilarse, es decir, destino de los archivos estáticos de la aplicación. En este directorio se encuentra la aplicación web que se va a publicar.

[6]Promesa: en el contexto de JavaScript (https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise), objeto asíncrono que puede estar disponible ahora, en el futuro o nunca

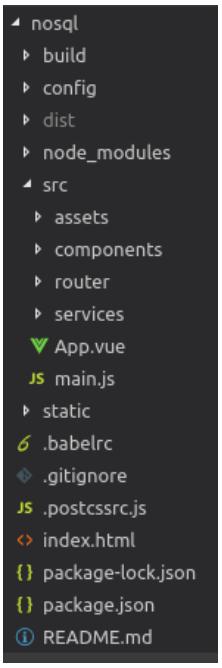


Figura D.1: Esquema de la organización inicial del proyecto *Vue.js*

```
...
"scripts": {
    "dev": "webpack-dev-server --inline
    --progress --config build/webpack.dev.conf.js",
    "start": "npm run dev",
    "build": "node build/build.js",
},
...
```

Figura D.2: Fragmento del fichero *package.json*, parámetro *scripts*

- **node_modules**: Se encuentran los diferentes módulos y librerías de *Node.js* con las que la aplicación puede trabajar como dependencias.
- **src**: Este directorio alberga todo el código fuente implementado. *Assets* almacena las imágenes utilizadas, *components* tiene los distintos ficheros con extensión *vue*, es decir, las vistas y los diferentes componentes del proyecto, *router* incluye un fichero *index.js* con las propiedades de enrutamiento de la aplicación, *services* se trata del módulo de lógica de negocio con los ficheros JavaScript que conectan directamente con el servidor, *App.vue* es la vista principal de la aplicación desde donde se cargan el resto de componentes y *main.js* es el fichero principal de inicialización de la aplicación.

La configuración de *webpack* se realiza a través de un fichero de configuración denominado *webpack.base.conf.js* donde se indican diferentes parámetros de configuración. El parámetro encargado de indicar cómo procesar los diferentes tipos de ficheros es *modules.rules*, que se trata de una lista patrones asociados a un *loader*. En el fragmento de código D.4 se puede ver de manera simplificada un ejemplo del funcionamiento de este fichero de configuración.

De este modo los distintos archivos del proyecto son redirigidos al *loader* correspondiente

```
...
 module: {
   rules: [
     {
       test: /\.vue$/,
       loader: 'vue-loader',
       options: vueLoaderConfig
     },
     {
       test: /\.js$/,
       loader: 'babel-loader',
       exclude: 'node-modules'
     },
     ...
   ]
 }
 ...
```

Código D.4: Fragmento del fichero de configuración de *webpack*, parámetro *rules*

a través del filtrado indicado en el fichero de configuración de *webpack* y es ahí donde se realiza el procesado de dichos ficheros.

D.2. Configuración de los paquetes de diseño

Las dependencias necesarias para el desarrollo de un proyecto *Vue.js* se encuentran instaladas. Sin embargo, existen otro tipo de dependencias secundarias que añaden funcionalidades a la aplicación y permiten realizar configuraciones adicionales tanto en el comportamiento como en el estilo de la aplicación.

MDBVue y *BootstrapVue* son dos librerías que proporcionan una serie de componentes gráficos prediseñados que pueden ser personalizados y añadidos a la aplicación. Algunos de los componentes de los que disponen son filas y columnas (*bootstrap*), tablas, *cards*, barras de navegación, menús, formularios, paneles, etc. Ambas librerías se utilizan importándolas en el fichero con extensión *vue* donde se quieren utilizar o bien añadiéndolas de manera global a todo el proyecto.

```
import {elem1, elem2}
from 'bootstrap-vue';

import {elem1, elem4}
from 'mdbvue';
```

Opción 1

```
import BootstrapVue
from 'bootstrap-vue';
Vue.use(BootstrapVue);

import MDBVue
from 'mdbvue';
Vue.use(MDBVue);
```

Opción 2

Además, es posible instalar componentes prediseñados concretos en el proyecto. *Vue-avatar*[7] es un componente que permite configurar de manera sencilla a través de parámetros predefinidos un avatar en la aplicación. *Vue-notification*[8] es un componente que permite gestionar notificaciones emergentes en la aplicación de manera gráfica.

La instalación de estas librerías y paquetes (Cód. D.5) se realiza a través del gestor de paquetes *npm*.

```
$ npm install mdbvue bootstrap-vue --save # MDBVue, BootstrapVue
$ npm install vue-avatar vue-notification --save # vue-avatar, vue-notification
```

Código D.5: Instalación de librerías y componentes gráficos

D.3. Configuración del servidor *Express.js*

Con la instalación de los paquetes necesarios para la parte cliente de la aplicación, se procede a configurar el entorno para el desarrollo del servicio. Esta configuración se realiza en local ya que la generación del servicio se hace localmente, pero con el servicio finalizado hay que realizar esta misma configuración en la máquina virtual que va a alojar el servidor. El *framework* de *Node.js* utilizado es *Express.js*. Para evitar reiniciar el servicio cada vez que se realiza un cambio en el código se instala *nodemon*, un módulo que refresca el servidor cada vez se actualiza un fichero. Otro módulo que facilita el desarrollo del servidor es *morgan*, que permite mostrar por consola información acerca de los accesos al servidor.

Se necesita un módulo que permita obtener el cuerpo de una petición *POST* si fuera necesario. El módulo *body-parser* permite realizar esta operación trasladando el cuerpo

[7] *vue-avatar*: https://www.npmjs.com/package/vue-avatar
[8] *vue-notification*: https://www.npmjs.com/package/vue-notification

de la petición al parámetro *req.body*. Además, se necesita un módulo que permita trabajar al servidor con imágenes. El módulo *multer* posibilita el procesamiento de ficheros de tipo *multipart/form-data*, por lo que es preciso instalarlo como dependencia.

La instalación de *Express.js* y el resto de dependencias se realiza tal y como se indica en el fragmento de código D.6.

```
$ npm install express --save # express.js
$ npm install nodemon --save-dev # nodemon
$ npm install body-parser --save # body-parser
$ npm install morgan --save # morgan
$ npm install multer --save # multer
```

Código D.6: Instalación de *Express.js* y configuración inicial

Para crear el servicio, desde un nuevo directorio se inicia un nuevo proyecto (*npm init*) para generar el fichero *package.json* asociado, y seguidamente se genera un archivo *app.js* que será el punto de partida del servidor, por donde pasarán todas las peticiones.

E. ANEXO

Interfaces de la aplicación

En este anexo se incluyen las principales interfaces de la aplicación web desarrollada, asociadas a los casos de uso identificados en el apartado 5.1.2 de este documento.

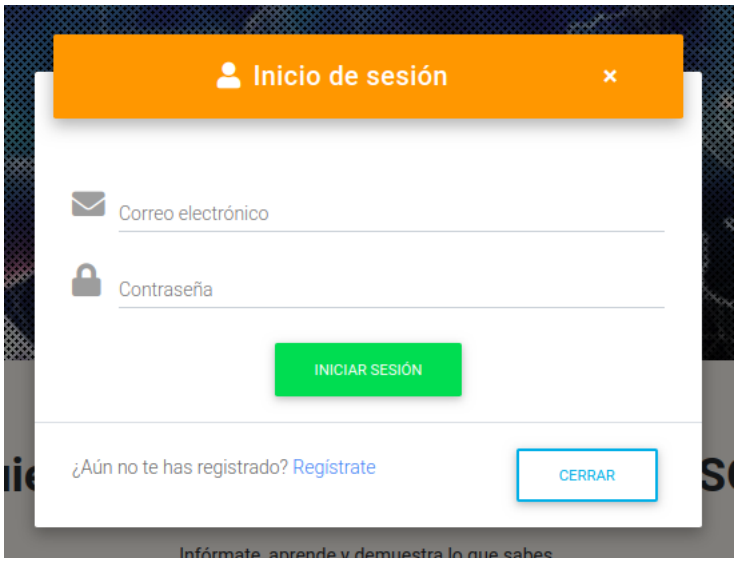


Figura E.1: Caso de uso *Login*

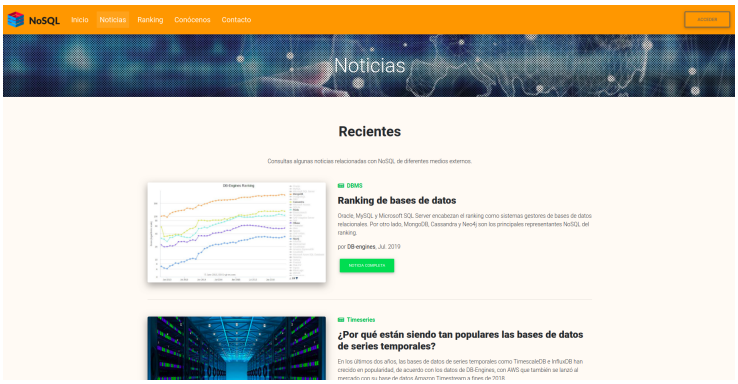


Figura E.2: Caso de uso *Ver noticias*

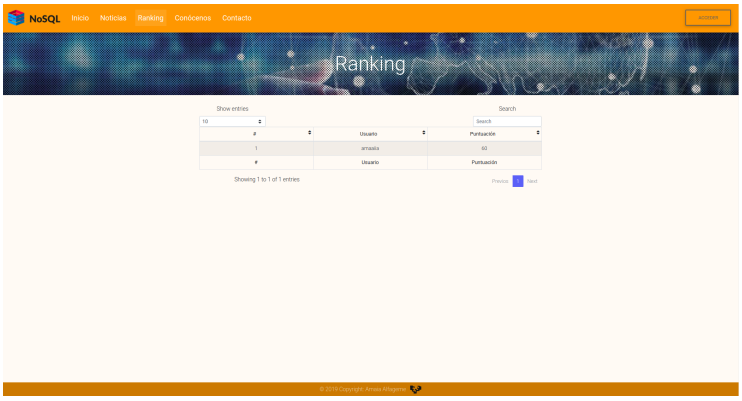


Figura E.3: Caso de uso *Ver raking*

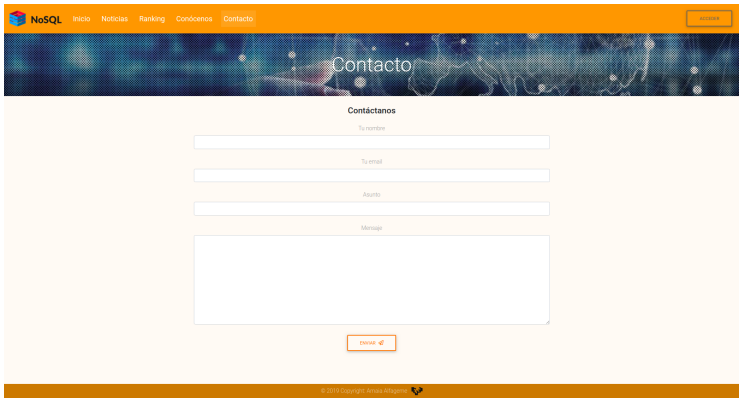


Figura E.4: Caso de uso *Escribir contacto*

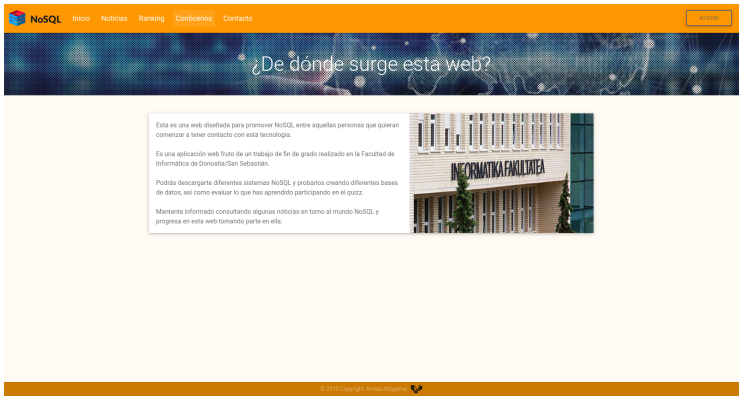


Figura E.5: Caso de uso *Ver información*

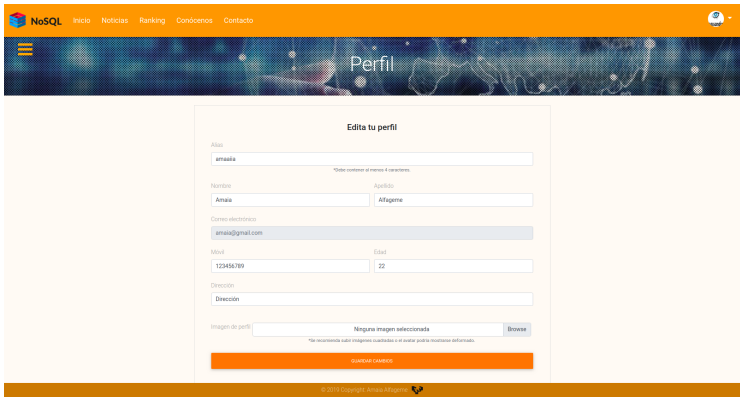


Figura E.6: Caso de uso *Ver perfil*

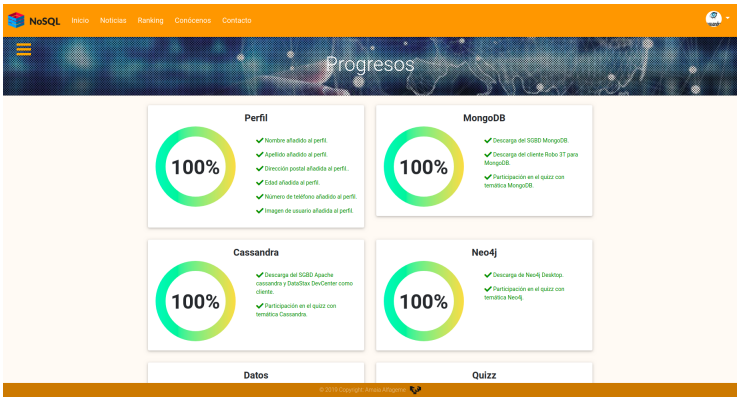


Figura E.7: Caso de uso *Consultar progresos*

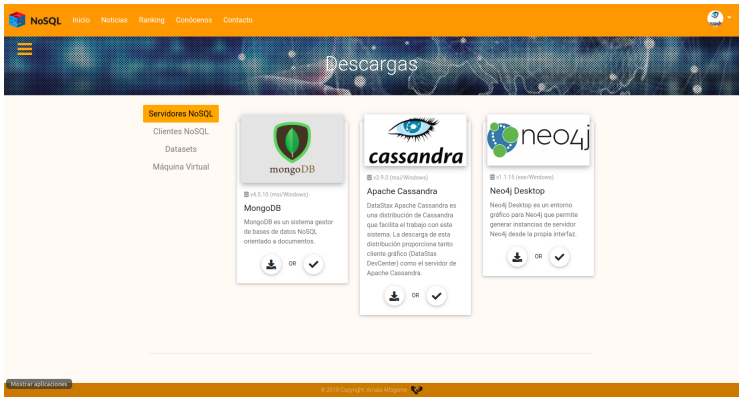


Figura E.8: Caso de uso *Ver descargas*

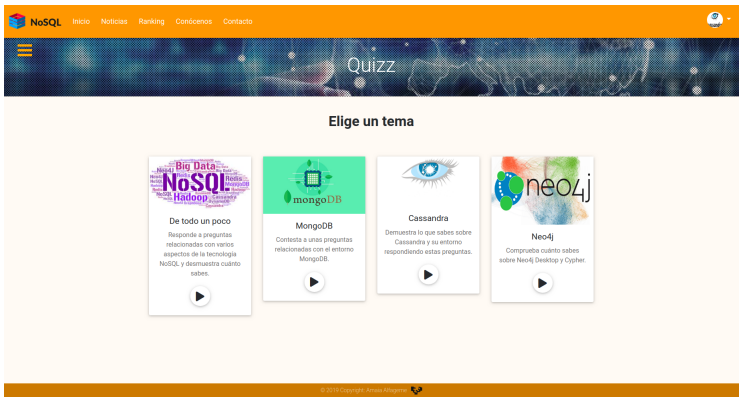


Figura E.9: Caso de uso *Ver quizz* (1)

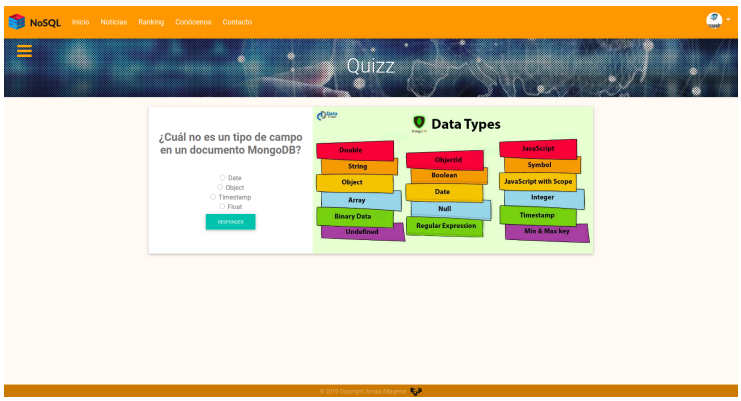


Figura E.10: Caso de uso *Ver quizz* (2)

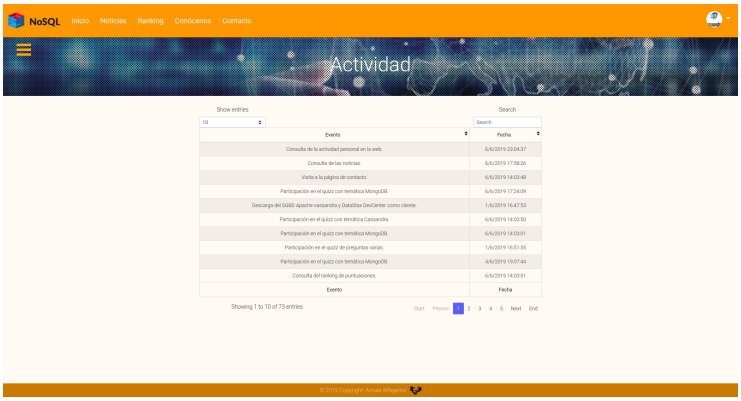


Figura E.11: Caso de uso *Ver actividad*

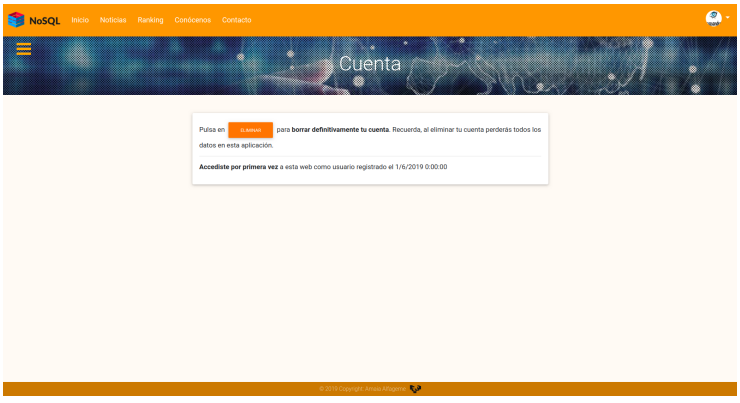


Figura E.12: Caso de uso *Consultar cuenta*

Bibliografía

[Alarcón, 2017] Alarcón, J. M. (2017). Webpack: qué es, para qué sirve y sus ventajas e inconvenientes. https://www.campusmvp.es/recursos/post/webpack-que-es-para-que-sirve-y-sus-ventajas-e-inconvenientes.aspx. [Online; accedido el 17-julio-2019].

[Bendera et al., 2014] Bendera, C. M., Deco, C., Sanabria, J. S. G., Hallo, M., and Gallego, J. C. P. (2014). *Tópicos avanzados de Bases de datos*, pages 103–106. Proyecto Latín.

[CampusBigData, 2017] CampusBigData (2017). El origen del Big Data. https://www.campusbigdata.com/big-data-blog/item/106-origen-big-data. [Online; accedido el 03-junio-2019].

[Chodorow, 2013] Chodorow, K. (2013). *MongoDB, the definitive guide*, pages 29–48. O'Reilly.

[Martin, 2013] Martin, L. (2013). Principales tecnologías Big Data: NoSQL. https://www.brainsins.com/es/blog/principales-tecnologias-big-data-nosql/107943. [Online; accedido el 27-marzo-2019].

[Mendoza, 2017] Mendoza, E. (2017). ¿Cómo saber si necesitas una Base de Datos NoSQL? https://medium.com/@eugeniomendoza/c%C3%B3mo-saber-si-necesitas-una-base-de-datos-nosql-b6cfd5bb7d9b. [Online; accedido el 10-abril-2019].

[Oracle, 2016] Oracle (2016). ¿Qué es una Base de Datos NoSQL? https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql. [Online; accedido el 27-marzo-2019].

[Robinson et al., 2015] Robinson, I., Webber, J., and Eifrém, E. (2015). *Graph databases*, pages 27–31. O'Reilly.

[Strauch, 2011] Strauch, C. (2011). *NoSQL databases*. PhD dissertation, Stuttgart Media University.

[Walters, 2018] Walters, R. (2018). MongoDB: Querying, Analyzing, and Presenting Time-Series Data. https://www.mongodb.com/blog/post/time-series-data-and-mongodb-part-3--querying-analyzing-and-presenting-timeseries-data. [Online; accedido el 21-mayo-2019].