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

# Bases de Datos NoSQL para la aplicación TicketBai

Autor

*Gorka Álvarez Marlasca*

2021

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

# Bases de Datos NoSQL para la aplicación TicketBai

Autor

*Gorka Álvarez Marlasca*

Directora

Arantza Illarramendi Echave

# Resumen

Este Trabajo de Fin de Grado (TFG) se ha desarrollado dentro del marco de un proyecto real ubicado en el entorno del *Big Data*, en el que participa el grupo de investigación BDI[1] de la UPV/EHU. En dicho trabajo se ha desarrollado, a modo de prueba de concepto, dos sistemas de almacenamiento y consulta de facturas emitidas por las distintas entidades de Gipuzkoa (aplicación TicketBai).

En primer lugar se han diseñado e implementado dos Bases de Datos en dos Sistemas de Gestión de Bases de Datos NoSQL diferentes, MongoDB y Cassandra. Junto a este diseño e implementación, se han estudiado e integrado distintas técnicas de compresión de datos sin pérdida con el objetivo de reducir el volumen de datos almacenados sin perder demasiado rendimiento. El obteivo de realizar dos implementaciones es el de poder comparar el rendimiento de ambos sistemas de almacenamiento en un contexto real.

Por último, se ha diseñado una aplicación web, a modo de simulación de la aplicación web final, que facilite al usuario comprobar el rendimiento de cada uno de los sistemas implementados.

[1] https://bdigroup.web.app/

# Agradecimientos

En primer lugar me gustaría agradecer a mi tutora, Arantza Illaramendi, por brindarme la oportunidad de realizar este Trabajo de Fin de Grado y por acercarme lo máximo posible a trabajar en un proyecto de un entorno real. Agradecerle también la forma en la que me ha sabido guiar durante todo el transcurso del proyecto así como el apoyo proporcionado durante todo el curso del mismo.

En segundo lugar, quiero agradecer a Borja Díez y Victor Ramirez, ambos miembros del grupo de Investigación BDI, por haberme acogido tan bien desde el primer minuto y por haberme hecho sentir uno más en todo momento. Agradecerles también la ayuda que han mostrado siempre que he tenido algún problema o duda que no he sabido solucionar, parte de lo conseguido ha sido gracias a ellos.

Por último, agradecer a mi familia y amigos por todo el apoyo que me han mostrado siempre que lo he necesitado y haberme ayudado a despejar la mente siempre que lo he necesitado. En general, gracias por confiar en mí desde el principio.

# Índice general

# Índice de figuras

# Índice de tablas

# 1. CAPÍTULO

## Introducción

Los avances tecnológicos que han tenido lugar durante los últimos años han tenido un enorme impacto en la gestión de los datos que se realiza actualmente, también conocido como *Big Data*. Debido al desmesurado crecimiento del volumen de los datos, los métodos de tratamiento tradicionales se han visto obsoletos en la mayoría de los casos teniendo que buscar nuevos métodos de tratamiento y almacenamiento.

Relacionado con el incremento del volumen de los datos surge una nueva forma de almacenamiento, las bases de datos NoSQL. Este tipo de bases de datos permite mayor escalabilidad en el almacenamiento de los datos combinándolo con un diseño distribuido del sistema.

Considerando las oportunidades que se están generando alrededor del *Big Data*, el Departamento de Economía y Hacienda del Gobierno Vasco ha impulsado la iniciativa TicketBai.

TicketBai es un proyecto compartido entre las tres Diputaciones Forales y cuyo propósito es establecer un software común para todas las personas físicas y jurídicas que realicen una actividad económica [Gobierno Vasco, 2019]. Mediante este sistema, se permitirá a las haciendas forales controlar, en mayor medida, los ingresos de las personas contribuyentes, facilitando además, el cumplimiento de las obligaciones tributarias de los contribuyentes. Este control se pretende realizar a través del manejo de las facturas que emiten los comercios y entidades.

# 2. CAPÍTULO

# Contexto del proyecto

Durante este capítulo se presentará el contexto del proyecto. Este se ha dividido en dos partes, la primera, relativa al dominio en el que se desarrolla el proyecto, y la segunda, sobre las distintas tecnologías abordadas durante el transcurso del proyecto.

## 2.1. Contexto del dominio

El **Big Data** es el término utilizado para referirse a una colección de datos de un volumen inmenso y que se incrementa exponencialmente con el tiempo. Aunque no existe un origen claro del término Big Data la mayoría de investigadores coinciden en que su origen se debe al artículo *The Google File System* publicado en 2003 [Ghemawat et al., 2003].

El concepto de *Big Data* y las nuevas tecnologías relacionadas con ello tienen una repercusión especial en la industria, dando lugar a una nueva revolución industrial a la que se la denomina **Industria 4.0**. Muchas empresas y organizaciones han tenido que adaptar su forma de negocio considerando esta gran revolución.

En la misma línea, organizaciones gubernamentales como el Departamento de Economía y Hacienda del Gobierno Vasco, han visto la necesidad de impulsar la creación de la aplicación TicketBai, la cual, como ya se adelantaba, tiene el objetivo de garantizar que todas las personas físicas y jurídicas que realicen actividades económicas cumplan correctamente con sus obligaciones fiscales.

TicketBai es un proyecto común de las tres Haciendas Forales y del Gobierno Vasco cuyo

objetivo es la implantación de una serie de normas y software de facturación que permitan a la Administración tributaria controlar los ingresos de las personas contribuyentes en sus actividades económicas, especialmente las actividades relacionadas con el intercambio de bienes y servicios y cuyos ingresos son, generalmente, en efectivo [Gobierno Vasco, 2020]. Una vez el sistema este completamente implementado todas las personas y entidades que emitan facturas deberán hacerlo utilizando dispositivos, como TPVs[1] u ordenadores personales, que implementen las regulaciones correspondientes para garantizar el correcto funcionamiento de la aplicación.

## 2.2. Contexto tecnológico

Gracias a la gran expansión del Big Data surgieron nuevos métodos de almacenamiento de datos. A medida que aumentaba el número de datos a almacenar, las Bases de Datos relacionales perdían eficiencia y rendimiento, es por ello, que durante el año 2000, surgieron las Bases de Datos no relacionales o NoSQL.

El objetivo de este tipo de SGBDs era ofrecer la capacidad de crear bases de datos escalables, que no estuviesen sujetas a un esquema y que pudiesen manejar, de forma eficiente, un elevado volumen de datos [Hosting Data, 2020]. En función de la forma en la que se almacenan los datos, este tipo de SGBD se pueden dividir en cuatro grandes grupos: Documentos, Clave-Valor, Tabular y Orientadas a grafos, ver capítulo 5, Sistemas de Gestión de Bases de Datos.

Los Sistemas de Gestión de Bases de Datos, ya sean las SQL o las NoSQL, utilizan con frecuencia técnicas o algoritmos de compresión de datos. El origen de estas técnicas se remonta al año 1838, [Media Wiki, 2020] año en el que el código Morse fue inventado y el cual codificaba las distintas letras del alfabeto. No obstante, las técnicas de compresión como se conocen en la actualizad no adquirieron gran importancia hasta la década de 1970, cuando Internet se empezó a volver popular y el algoritmo de Lempel-Ziv fue inventado, ver capítulo 4, Descripción de las técnicas de compresión.

En general, existen dos tipos de algoritmos de compresión de datos, con pérdida y sin pérdida. Los primeros se utilizan frecuentemente para la compresión de texto y archivos, mientras que los segundos se utilizan para comprimir y descomprimir imágenes o vídeos en los que la pérdida de datos únicamente implica la pérdida de calidad en la imagen.

[1] Terminal Punto de Venta

# 3. CAPÍTULO

# Objetivos y planificación del proyecto

El siguiente capítulo recoge los objetivos y planificación completa del proyecto. Se detallarán los distintos paquetes de trabajo identificados junto con las etapas seguidas en la realización de los mismos.

## 3.1. Objetivo

El objetivo del proyecto consiste en **diseñar e implementar** dos Bases de Datos, utilizando distintos Sistemas de Gestión de Bases de Datos NoSQL, que permitan almacenar, de manera eficiente, las facturas emitidas por distintas empresas de Gipuzkoa con el fin de comparar su rendimiento.

## 3.2. Alcance

En todo proyecto es importante definir los límites del contenido del mismo, indicando las tareas a realizar junto con los aspectos que quedan excluidos.

El proyecto tiene como objetivo el desarrollo e implementación de diferentes Bases de Datos NoSQL que permitan seleccionar el sistema más adecuado para el posterior desarrollo operativo de la aplicación TicketBai. En concreto, se desarrollarán las Bases de Datos en los Sistemas de Gestión de Bases de Datos MongoDB y Cassandra.

Previamente al diseño e implementación de estas Bases de Datos será necesario analizar y estudiar distintas técnicas de compresión de datos sin pérdida que permitan almacenar, de forma eficiente, los datos recibidos por la aplicación TicketBai.

Así mismo, de cara a mostrar los resultados obtenidos se creará una aplicación web que permita insertar y obtener facturas, entre otras funcionalidades, de manera que simule el funcionamiento real de la aplicación TicketBai, mostrando el rendimiento, en tiempo real, de los sistemas implementados.

### 3.2.1. Objetivos

Tal como ya se ha mencionado, el objetivo principal del proyecto consiste en diseñar e implementar dos Bases de Datos NoSQL que permitan almacenar, de manera eficiente, las facturas emitidas mediante el software TicketBai en Gipuzkoa.

Entre los objetivos secundarios destacan el **análisis del dominio y diseño de esquemas conceptuales** en los que se reflejarán el conocimiento referente al dominio. Así mismo, también será necesario un correcto **manejo de tecnologías NoSQL y técnicas de compresión sin pérdida** para así crear un sistema capaz de almacenar los datos de manera eficiente. Ligado a este manejo, será necesario **diseñar e implementar las Bases de Datos correspondientes en Cassandra y MongoDB**, así como, **las pruebas correspondientes que permitan asegurar el adecuado rendimiento de las Bases de Datos diseñadas**. En estas pruebas se tendrán en cuenta aspectos como el tiempo de inserción y respuesta o los tiempos de compresión y descompresión de los datos almacenados. Por último, se **diseñará e implementará un entorno web** que permita comprobar, en tiempo real, el correcto desempeño de los sistemas desarrollados.

### 3.2.2. Exclusiones

En términos generales, no existen exclusiones destacables, salvo las relacionadas con los procesos de verificación de las facturas en los que se comprueban aspectos como los firma electrónica o la estructura del fichero.

No obstante, en lo que a la implementación del entorno web se refiere se excluyen aspectos como la seguridad de la misma, es decir, únicamente se realizará una implementación que derive en una página web funcional que permita ver las posibilidades que ofrece el sistema.

## 3.3. Estructura de Descomposición del Trabajo

Con el objetivo de complementar el diagrama de flujo del proyecto mostrado en la siguiente sección y comprender mejor el posterior Diagrama de Gantt, a continuación, se muestra la Estructura de Descomposición del Trabajo (E.D.T) del proyecto, figura 3.1.

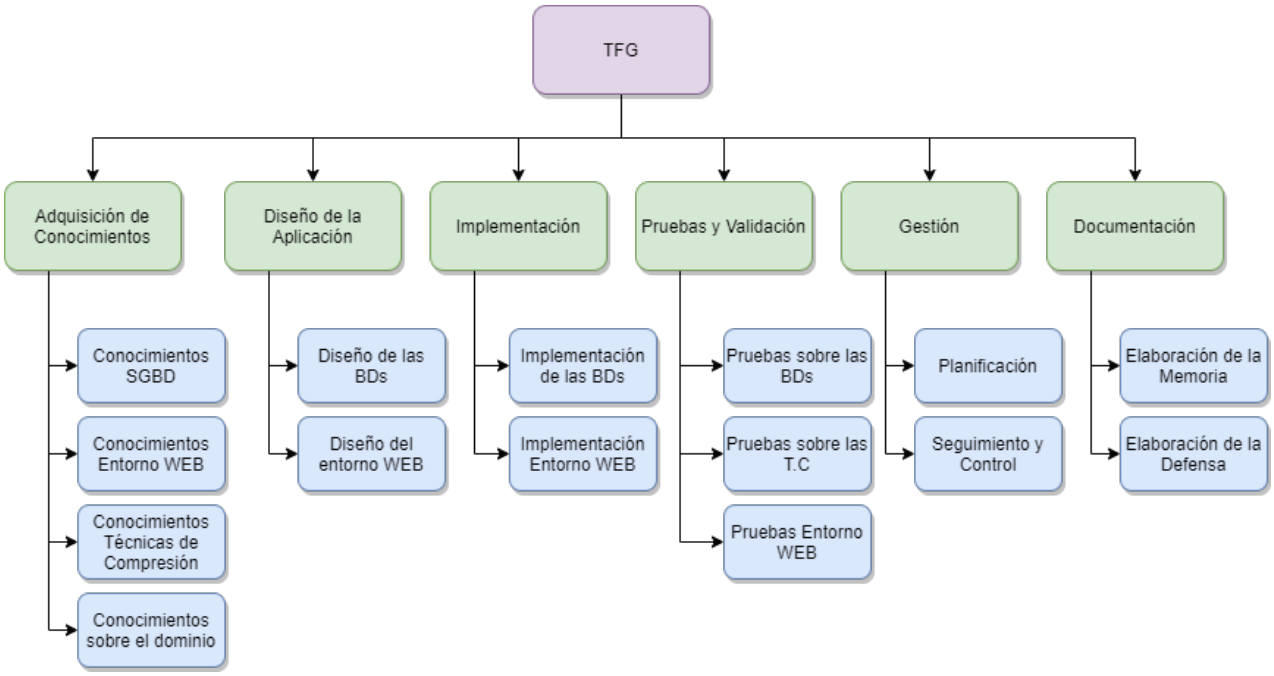



**Figura 3.1:** Estructura de Descomposición del Trabajo del Proyecto

#### Adquisición de Conocimientos

Esta fase de trabajo comprende todos los paquetes y tareas relacionados con la adquisición de competencias necesarias para alcanzar los objetivos definidos para el proyecto.

#### Diseño de la Aplicación

A lo largo de esta fase de trabajo se realizarán las tareas y paquetes relativos al diseño de los distintos sistemas, dentro de esta fase destacan tareas como el diseño de las Bases de Datos en MongoDB y Cassandra o el diseño del entorno web.

#### Implementación

Durante esta fase de trabajo se tratarán las tareas relacionadas con la implementación de los sistemas previamente diseñados.

#### Pruebas y Validación

Tras realizar la implementación de los sistemas anteriores, durante esta fase de trabajo se realizarán las pruebas necesarias para garantizar el correcto desempeño de los mismos. Así mismo, también se realizarán las pruebas necesarias que permitan seleccionar la técnica de compresión que mejor se adecue al sistema.

#### Gestión

Esta fase de trabajo agrupa todas las tareas relativas a la gestión y planificación del proyecto. Una correcta realización de las tareas de esta fase trabajo ayudarán a garantizar un proyecto que alcance los objetivos y requerimientos establecidos.

#### Documentación

Por último, esta fase de trabajo comprende las tareas relacionadas con la realización de los documentos y trámites relativos al TFG.

## 3.4. Diagrama de Flujo en Cascada

Para garantizar el correcto desarrollo del proyecto se ha decidido seguir un ciclo de vida en cascada. Este modelo de ciclo de vida es un proceso de desarrollo secuencial en el que las fases se ejecutan una detrás de otra, como una cascada. Además, en este ciclo de vida se realiza una verificación al final de cada una de las fases. El diagrama de flujo desarrollado para este proyecto es el mostrado en la figura 3.2. En dicho diagrama se ha presentado las etapas para cada uno de los objetivos considerados en este TFG.

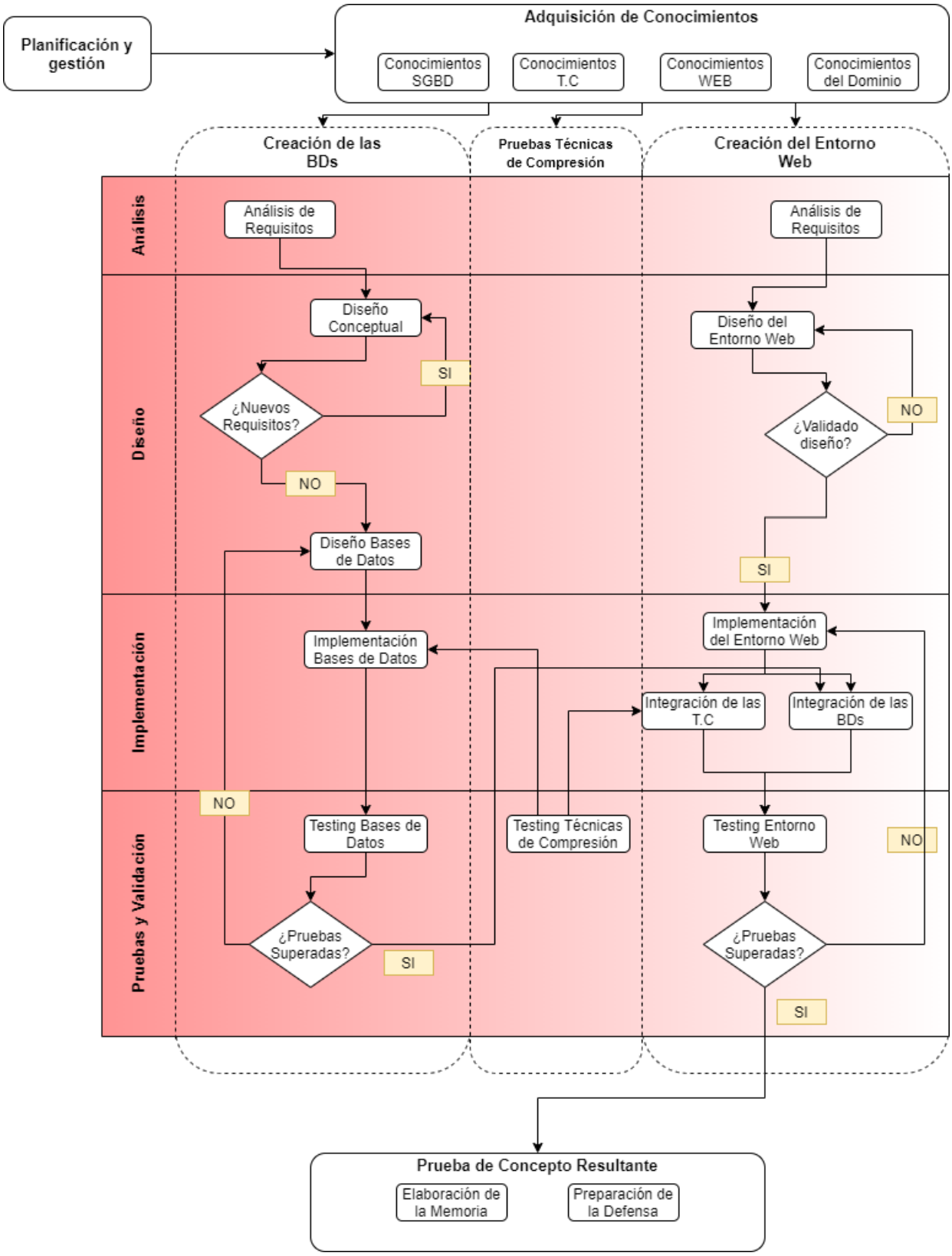
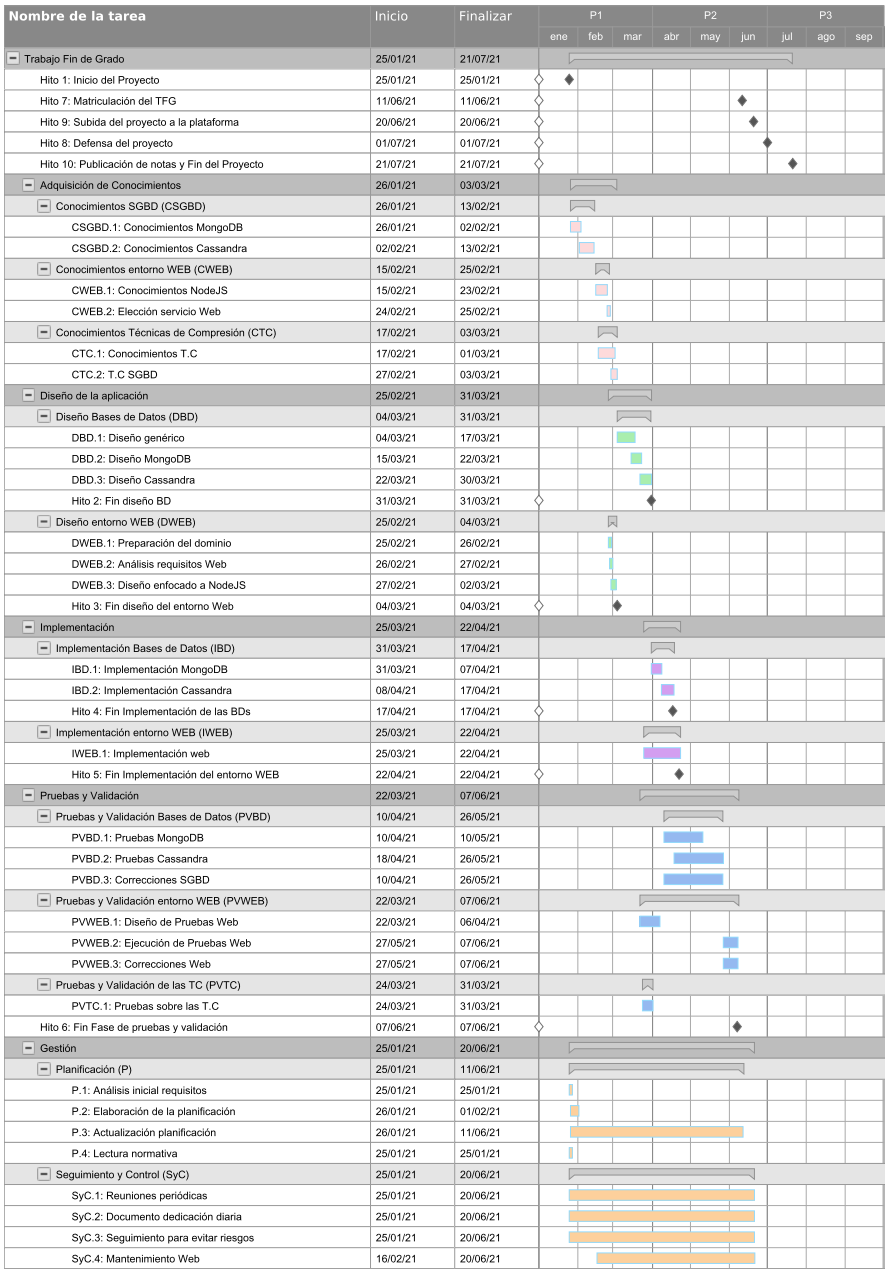



**Figura 3.2:** Diagrama de flujo en cascada del proyecto

## 3.5. Diagrama Gantt

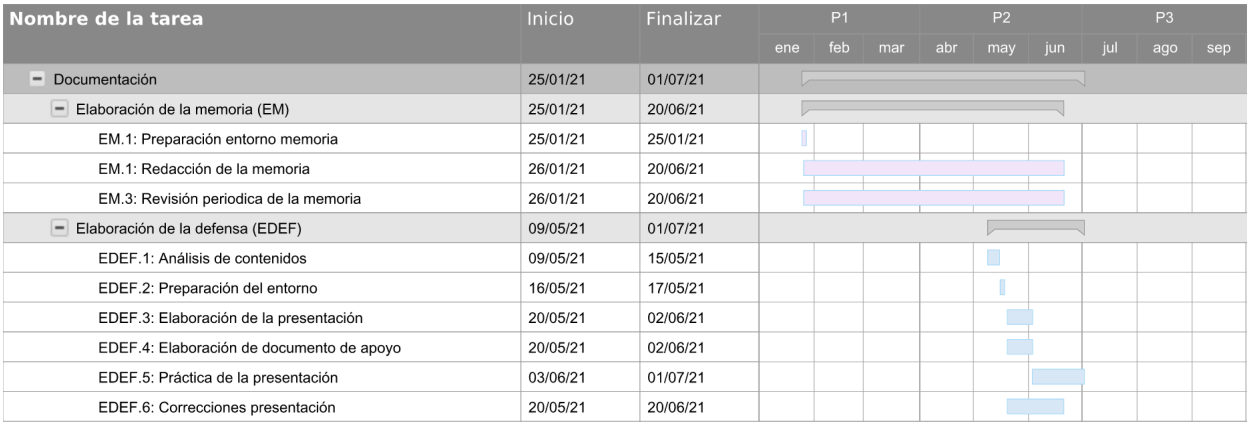
En esta sección se presenta el diagrama de Gantt del proyecto, el cual recoge las fechas de inicio y finalización previstas para cada una de las fases de trabajo antes mencionadas, junto con las tareas de cada una de ellas. Las figuras 3.3 y 3.4 muestran el diagrama Gantt del proyecto por meses. Para visualizar el diagrama de Gantt por semanas ver el anexo D.



| Nombre de la tarea | Inicio | Finalizar |
|---|---|---|
| Trabajo Fin de Grado | 25/01/21 | 21/07/21 |
| Hito 1: Inicio del Proyecto | 25/01/21 | 25/01/21 |
| Hito 7: Matriculación del TFG | 11/06/21 | 11/06/21 |
| Hito 9: Subida del proyecto a la plataforma | 20/06/21 | 20/06/21 |
| Hito 8: Defensa del proyecto | 01/07/21 | 01/07/21 |
| Hito 10: Publicación de notas y Fin del Proyecto | 21/07/21 | 21/07/21 |
| Adquisición de Conocimientos | 26/01/21 | 03/03/21 |
| Conocimientos SGBD (CSGBD) | 26/01/21 | 13/02/21 |
| CSGBD.1: Conocimientos MongoDB | 26/01/21 | 02/02/21 |
| CSGBD.2: Conocimientos Cassandra | 02/02/21 | 13/02/21 |
| Conocimientos entorno WEB (CWEB) | 15/02/21 | 25/02/21 |
| CWEB.1: Conocimientos NodeJS | 15/02/21 | 23/02/21 |
| CWEB.2: Elección servicio Web | 24/02/21 | 25/02/21 |
| Conocimientos Técnicas de Compresión (CTC) | 17/02/21 | 03/03/21 |
| CTC.1: Conocimientos T.C | 17/02/21 | 01/03/21 |
| CTC.2: T.C SGBD | 27/02/21 | 03/03/21 |
| Diseño de la aplicación | 25/02/21 | 31/03/21 |
| Diseño Bases de Datos (DBD) | 04/03/21 | 31/03/21 |
| DBD.1: Diseño genérico | 04/03/21 | 17/03/21 |
| DBD.2: Diseño MongoDB | 15/03/21 | 22/03/21 |
| DBD.3: Diseño Cassandra | 22/03/21 | 30/03/21 |
| Hito 2: Fin diseño BD | 31/03/21 | 31/03/21 |
| Diseño entorno WEB (DWEB) | 25/02/21 | 04/03/21 |
| DWEB.1: Preparación del dominio | 25/02/21 | 26/02/21 |
| DWEB.2: Análisis requisitos Web | 26/02/21 | 27/02/21 |
| DWEB.3: Diseño enfocado a NodeJS | 27/02/21 | 02/03/21 |
| Hito 3: Fin diseño del entorno Web | 04/03/21 | 04/03/21 |
| Implementación | 25/03/21 | 22/04/21 |
| Implementación Bases de Datos (IBD) | 31/03/21 | 17/04/21 |
| IBD.1: Implementación MongoDB | 31/03/21 | 07/04/21 |
| IBD.2: Implementación Cassandra | 08/04/21 | 17/04/21 |
| Hito 4: Fin Implementación de las BDs | 17/04/21 | 17/04/21 |
| Implementación entorno WEB (IWEB) | 25/03/21 | 22/04/21 |
| IWEB.1: Implementación web | 25/03/21 | 22/04/21 |
| Hito 5: Fin Implementación del entorno WEB | 22/04/21 | 22/04/21 |
| Pruebas y Validación | 22/03/21 | 07/06/21 |
| Pruebas y Validación Bases de Datos (PVBD) | 10/04/21 | 26/05/21 |
| PVBD.1: Pruebas MongoDB | 10/04/21 | 10/05/21 |
| PVBD.2: Pruebas Cassandra | 18/04/21 | 26/05/21 |
| PVBD.3: Correcciones SGBD | 10/04/21 | 26/05/21 |
| Pruebas y Validación entorno WEB (PVWEB) | 22/03/21 | 07/06/21 |
| PVWEB.1: Diseño de Pruebas Web | 22/03/21 | 06/04/21 |
| PVWEB.2: Ejecución de Pruebas Web | 27/05/21 | 07/06/21 |
| PVWEB.3: Correcciones Web | 27/05/21 | 07/06/21 |
| Pruebas y Validación de las TC (PVTC) | 24/03/21 | 31/03/21 |
| PVTC.1: Pruebas sobre las T.C | 24/03/21 | 31/03/21 |
| Hito 6: Fin Fase de pruebas y validación | 07/06/21 | 07/06/21 |
| Gestión | 25/01/21 | 20/06/21 |
| Planificación (P) | 25/01/21 | 11/06/21 |
| P.1: Análisis inicial requisitos | 25/01/21 | 25/01/21 |
| P.2: Elaboración de la planificación | 26/01/21 | 01/02/21 |
| P.3: Actualización planificación | 26/01/21 | 11/06/21 |
| P.4: Lectura normativa | 25/01/21 | 25/01/21 |
| Seguimiento y Control (SyC) | 25/01/21 | 20/06/21 |
| SyC.1: Reuniones periódicas | 25/01/21 | 20/06/21 |
| SyC.2: Documento dedicación diaria | 25/01/21 | 20/06/21 |
| SyC.3: Seguimiento para evitar riesgos | 25/01/21 | 20/06/21 |
| SyC.4: Mantenimiento Web | 16/02/21 | 20/06/21 |



**Figura 3.3:** Diagrama Gantt por meses - Primera parte



| Nombre de la tarea | Inicio | Finalizar |
|---|---|---|
| Documentación | 25/01/21 | 01/07/21 |
| Elaboración de la memoria (EM) | 25/01/21 | 20/06/21 |
| EM.1: Preparación entorno memoria | 25/01/21 | 25/01/21 |
| EM.1: Redacción de la memoria | 26/01/21 | 20/06/21 |
| EM.3: Revisión periodica de la memoria | 26/01/21 | 20/06/21 |
| Elaboración de la defensa (EDEF) | 09/05/21 | 01/07/21 |
| EDEF.1: Análisis de contenidos | 09/05/21 | 15/05/21 |
| EDEF.2: Preparación del entorno | 16/05/21 | 17/05/21 |
| EDEF.3: Elaboración de la presentación | 20/05/21 | 02/06/21 |
| EDEF.4: Elaboración de documento de apoyo | 20/05/21 | 02/06/21 |
| EDEF.5: Práctica de la presentación | 03/06/21 | 01/07/21 |
| EDEF.6: Correcciones presentación | 20/05/21 | 20/06/21 |



**Figura 3.4:** Diagrama Gantt por meses - Segunda parte

### 3.5.1. Hitos del Proyecto

En esta sección se presentan los hitos a alcanzar durante el desarrollo del proyecto.

- **Hito 1:** Inicio del proyecto. (25/01/2021)
- **Hito 2:** Fin de diseño de las BDs. (31/03/2021)
- **Hito 3:** Fin de diseño del entorno web. (04/03/2021)
- **Hito 4:** Fin de la implementación de las BD. (17/04/2021)
- **Hito 5:** Fin de la implementación del entorno web. (22/04/2021)
- **Hito 6:** Fin de la fase de pruebas y validación (07/06/2021)
- **Hito 7:** Matriculación del proyecto en G.A.U.R. (11/06/2021)
- **Hito 8:** Defensa del proyecto. (01/07/2021[1])
- **Hito 9:** Subida del proyecto a la plataforma. (20/06/2021)
- **Hito 10:** Publicación de notas y Fin del proyecto. (21/07/2021)

[1] Fecha estimada

## 3.6. Gestión de riesgos

En todo proyecto es necesario realizar un buen análisis y gestión de riesgos que permitan que el proyecto se ejecute sin ningún problema. Una buena mitigación de riesgos evitará costos extras en el proyecto.

Durante esta sección, se detallan los riesgos identificados que puedan afectar negativamente al desarrollo del proyecto, junto con las acciones a tomar en cada uno de ellos.

### 3.6.1. Problemas de compaginación con las asignaturas

Dado que las fechas de realización del TFG coinciden con el desarrollo del último cuatrimestre del cuarto curso del grado en Ingeniería Informática, gran parte del desarrollo del mismo deberá ser compaginado con las asignaturas a realizar durante ese cuatrimestre.

- **Probabilidad:** Baja
- **Impacto:** Alto
- **Responsabilidad:** Propia
- **Consecuencias:** Dificultad de cumplimiento de los plazos establecidos para el proyecto.
- **Prevención:** Realizar una planificación en la que se tengan en cuenta los hitos y dedicaciones estimadas de las asignaturas.
- **Plan de mitigación:** Replanificación de las tareas para tratar de compaginar ambas actividades de la mejor manera posible.

### 3.6.2. Errores en la planificación

En todo proyecto con una planificación de gran envergadura existe el riesgo de que dicha planificación contenga errores o que no contenga aspectos importantes no contemplados al comienzo

- **Probabilidad:** Media

- **Impacto:** Alto
- **Responsabilidad:** Propia
- **Consecuencias:** Retraso en el cumplimiento de los plazos establecidos.
- **Prevención:** Realizar un amplio estudio previo para evaluar los posibles errores e imprevistos. Así mismo, se ampliarán los plazos lo máximo posible dejando margen para posibles errores.
- **Plan de mitigación:** Replanificación de los plazos establecidos intentando garantizar el menor impacto posible a largo plazo.

### 3.6.3. Problemas con la captura de requisitos

Al el comienzo del desarrollo del proyecto, el grupo BDI no tenía información sobre las principales consultas que se realizarían sobre el sistema, por lo que, principalmente, el diseño de la BD usando el SGBD Cassandra se podría ver afectado.

- **Probabilidad:** Media
- **Impacto:** Alto
- **Responsabilidad:** Ajena
- **Consecuencias:** Retraso en el desarrollo de las BD, principalmente usando el SGBD Cassandra.
- **Prevención:** Para evitar que este riesgo afecte drásticamente al desarrollo del proyecto, si no se ha obtenido información suficiente que permita un correcto desarrollo de la BD antes del 1 de Abril, se descartará el desarrollo del BD usando el SGBD Cassandra.
- **Plan de mitigación:** Replanificación de las tareas sin tener en cuanta el diseño e implementación de la BD usando el SGBD Cassandra.

### 3.6.4. Problemas con el aprendizaje de las tecnologías

Dado que muchas de las tecnologías abordadas durante el proyecto son desconocidas o no han sido utilizadas nunca, pueden darse problemas en el aprendizaje de las mismas.

- **Probabilidad:** Baja
- **Impacto:** Alto
- **Responsabilidad:** Propia
- **Consecuencias:** Retraso en el desarrollo de algunas de las tareas del proyecto.
- **Prevención:** Realizar una planificación en la que se establezca un tiempo considerable a la adquisición de conocimientos, asegurando de esta manera la correcta consolidación de los mismos.
- **Plan de mitigación:** Adelantar la realización de algunas de las tareas no dependientes de la herramienta o tecnología en cuestión.

### 3.6.5. Problemas con el software

El uso de gran variedad de software externo en el proyecto, implica la existencia del riesgo de obtener problemas con dicho software.

- **Probabilidad:** Baja
- **Impacto:** Medio
- **Responsabilidad:** Ajena
- **Consecuencias:** Retraso en el desarrollo de algunas de las tareas del proyecto.
- **Prevención:** Realizar un mantenimiento continuado del sistema tratando de utilizar versiones estables del software utilizado.
- **Plan de mitigación:** Adelantar la realización de algunas de las tareas no dependientes del software en cuestión y replanificar en caso de que el problema no tuviera solución.

### 3.6.6. Problemas con el hardware

El proyecto depende del hardware en el que está siendo desarrollado, lo que supone la posible generación de problemas en dicho hardware.

- **Probabilidad:** Baja
- **Impacto:** Alto
- **Responsabilidad:** Ajena
- **Consecuencias:** Retraso en el desarrollo de algunas de las tareas del proyecto.
- **Prevención:** Realizar un mantenimiento continuado del hardware utilizado. Así mismo, se aprovecharán al máximo las herramientas de gestión de copias de seguridad para evitar la pérdida de datos críticos.
- **Plan de mitigación:** Replanificar las tareas y restaurar la copia de seguridad correspondiente tratando de minimizar el impacto en el desarrollo del proyecto.

### 3.6.7. Problemas personales

Durante el desarrollo del proyecto existe la posibilidad de que se den problemas personales que no se pueden prever.

- **Probabilidad:** Baja
- **Impacto:** Medio
- **Responsabilidad:** Propia
- **Consecuencias:** Retraso en el desarrollo y entrega de algunas de las tareas del proyecto.
- **Prevención:** Es imposible prever este tipo de problemas ya que habra variaciones en función del problema surgido.
- **Plan de mitigación:** Replanificación de las tareas con el objetivo de reducir las consecuencias a corto / medio plazo.

## 3.7. Herramientas y Tecnologías

En esta sección se presentan brevemente las principales tecnologías que se han utilizado durante el desarrollo del proyecto.

### 3.7.1. Overleaf y LaTeX

Overleaf[2] es un editor de Latex *online*. Se trata de un editor gratuito que no necesita instalación y que permite editar, de forma rápida y sencilla, documentos latex. Latex, es un lenguaje de composición de textos enfocado a crear documentos de alta calidad tipográfica. Tanto Overleaf como Latex, se han utilizado para redactar esta misma memoria.

### 3.7.2. Google Drive

Google Drive[3] es una de las más conocidas aplicaciones de almacenamiento en la nube. Permite acceder a los datos desde cualquier dispositivo y en cualquier momento. Se ha utilizado para almacenar copias de respaldo de los distintos documentos y archivos utilizados en el proyecto.

### 3.7.3. GitHub y GitKraken

GitHub[4] es una plataforma *online* que implementa el control de versiones Git[5] y que permite almacenar cualquier tipo de proyectos, principalmente proyectos *software*. En cuanto a GitKraken[6], se trata de un cliente Git multiplataforma que permite realizar todas las funciones de Git sin necesidad de conocer exhaustivamente los comandos. Tanto GitHub como GitKraken, se han utilizado para llevar un control de versiones del proyecto.

### 3.7.4. Smartsheet

Smartsheet[7] es un servicio enfocado a la organización y gestión de de equipos de trabajo. Smartsheet se ha utilizado para desarrollar el Diagrama Gantt que recoge los periodos de realización de las tareas del proyecto.

[2] https://www.overleaf.com/
[3] https://www.google.es/drive/apps.html
[4] https://github.com/
[5] https://git-scm.com/
[6] https://www.gitkraken.com/
[7] https://es.smartsheet.com/

### 3.7.5. Draw.io

Draw.io[8] es una aplicación online gratuita que permite la creación de todo tipo de diagramas de flujo, entidad relación, UML, etc.

Esta aplicación se ha utilizado para crear el diagrama de flujo y el E.D.T del proyecto.

### 3.7.6. MongoDB

MongoDB[9] es un Sistema de Gestión de Base de Datos No-SQL orientado a documentos. Estos documentos son almacenados en formato JSON, por lo que le ofrece una compatibilidad con multitud de lenguajes de forma nativa.

### 3.7.7. Cassandra

Cassandra [10] es un Sistema de Gestión de Base de Datos No-SQL orientado a tablas. Desarrollado por Apache en 2008 permite manejar grandes volúmenes de datos a través de varios servidores evitando de esta manera la existencia de un único punto de fallo.

### 3.7.8. Visual Studio Code

Visual Studio Code [11] es un editor de código abierto desarrollado por Microsoft[12]. Incluye características como finalización inteligente de código, *debugging* o resaltado de sintaxis. Estás características y otras muchas más le han hecho convertirse en uno de los editores de código más utilizado por los desarrolladores.

### 3.7.9. Node.JS

Node.JS [13] es un entorno de ejecución para la capa servidor, de código abierto, basado en el lenguaje de programación JavaScript. Este entorno se ha utilizado para crear todo el

[8] https://app.diagrams.net/
[9] https://mongodb.com/
[10] https://cassandra.apache.org/
[11] https://code.visualstudio.com/
[12] https://www.microsoft.com/
[13] https://nodejs.org/en/

sistema que simulará el envío y recepción de facturas a las Bases de Datos implementadas, además también se utilizará para crear el entorno web que simulará el cliente final de la aplicación.

### 3.7.10. PostMan

PostMan [14] es una plataforma enfocada para el desarrollo de APIs. Junto con Node.JS, se ha utilizado para realizar peticiones de testeo que permitan probar el correcto funcionamiento del sistema.

[14] https://www.postman.com/

# 4. CAPÍTULO

## Descripción de las técnicas de compresión

La RAE[1] [ASALE and RAE, 2021], define comprimir como la acción de oprimir, apretar, estrechar o reducir a menor volumen algo. Así, los algoritmos de compresión, se encargan de reducir el volumen que ocupa un fichero o archivo de datos.

A la hora de recuperar el formato original existen dos técnicas, las técnicas sin pérdida, que como su nombre indica permite recuperar el archivo en el estado que tenía originalmente, y las técnicas con pérdida, que al contrario que las anteriores, juegan con la "pérdida de datos" para restablecer su estado original.

Para poder medir la "efectividad" de los algoritmos de compresión, se utiliza el ratio de compresión, que se define como $1 - \frac{B_0}{B_1}$, donde $B_0$ representa el número de total de bits del archivo después de la compresión y $B_1$ representa el número total de bits del archivo antes de la compresión. Por tanto, cuanto más cercano a 1 sea el ratio de compresión mejor será el algoritmo, aunque más adelante veremos que hay más factores a tener en cuenta, como puede ser la velocidad de compresión.

[1] Real Academia Española

## 4.1. Técnicas de compresión sin pérdida

Los algoritmos de compresión sin pérdida (*lossless algorithms*) son capaces de reconstruir el formato original a partir del mensaje comprimido [Hosseini, 2012]. Por ello, estos algoritmos, se utilizan principalmente en tareas relacionadas con el almacenamiento de textos o programas.

A continuación, se describen 5 algoritmos de compresión sin pérdida.[Gupta et al., 2017] Estos algoritmos son los que el grupo BDI concluyó que se ajustaban mejor a la aplicación a desarrollar.

### 4.1.1. GZip

El compresor GZip [loup Gailly and Adler, 2018] es el compresor utilizado y desarrollado por GNU[2]. Apareció por primera vez en 1993 con el objetivo de sustituir el anterior programa de compresión de GNU, *compress*, y sigue en uso, siendo su última versión estable la 1.10, que se liberó en 2018.

GZip es capaz de reducir el tamaño de los archivos utilizando el algoritmo Lempel-Ziv (LZ77). Este algoritmo es la base de todos los algoritmos que se van a describir en los siguientes apartados, así como de prácticamente todos los algoritmos basados en el uso de un diccionario.

La idea básica del algoritmo es ir iterando sobre el texto de entrada y reemplazando los datos repetidos por la referencia a una única copia de esos datos los cuales se han tratado previamente. Cada una de estas coincidencias se codifica mediante el siguiente triple *(offset, length, character)*, donde el *offset* representa el número de pasos hacia atrás, es decir, hacia el flujo de datos sin comprimir, que se deben realizar para encontrar el comienzo de la cadena que representa. *Length* simplemente representa la longitud de la cadena. Y *character* representa el siguiente carácter después de la cadena.

Por ejemplo, si se quiere codificar la siguiente cadena.[Budhrani, 2019]

```
1   A B A B C B A B A B A A
```

El algoritmo, avanzará de la siguiente manera. Primero, leerá el carácter “A”, al no estar registrado en el diccionario se guardará la tripleta (0,0,A), ya que para obtener el primer

[2] https://gnu.org

carácter se tienen que ir 0 pasos hacia atrás con un *string* de longitud 0 y encontrando el carácter “A”. El próximo carácter a tratar se representa entre ‘[ ]’.

```
A [B] A B C B A B A B A A
(0,0,A)
```

A continuación, se leerá el carácter “B”, como su situación es la misma que en el carácter anterior la codificación es similar, (0,0,B).

```
A B [A] B C B A B A B A A
(0,0,A) (0,0,B)
```

Seguidamente, se leerá el carácter “A”, el cual ya esta en el diccionario, el primer carácter del flujo a representar, por ello, se leerá el conjunto “AB”, el cual también esta en el diccionario, los dos primeros caracteres del flujo. Sin embargo, no es ese el caso del conjunto “ABC”, por ello, para poder formar este conjunto, y tomando como referencia el carácter “A”, que era el carácter que tocaba tratar, se retrocederá dos posiciones, donde está la representación más cercana, hasta la primera “A”, se formará un *string* de longitud 2, “AB” y se añadirá la “C” al final. De esta manera, se formaría la tripleta (2,2,C).

```
A B A B C [B] A B A B A A
(0,0,A) (0,0,B) (2,2,C)
```

A continuación se tratará la “B”, la cual ya se ha tratado con anterioridad, lo mismo pasa con “BA” y “BAB”, pero no con “BABA”, así que para formar este conjunto se retrocederá, partiendo desde el carácter “B” que nos tocaba tratar en este paso, 4 posiciones hacía atrás, formar un *string* de longitud 3, “BAB", y añadirle la “A” al final. Por tanto, la tripleta que se forma será (4,3,A).

```
A B A B C B A B A [B] A A
(0,0,A) (0,0,B) (2,2,C) (4,3,A)
```

Es el turno de la última codificación, el carácter “B” ya se ha tratado, y el conjunto “BA” también, siendo la representación más cercana justo 2 caracteres hacia atrás. El conjunto “BAA” no esta representado, por lo que, partiendo desde “B”, se retrocederá 2 pasos y se formará, con longitud 2, el *string* “BA” y se añadirá el carácter “A” al final, formando la tripleta (2,2,A). Es decir, la codificación del flujo de caracteres final sería.

```
(0,0,A) (0,0,B) (2,2,C) (4,3,A) (2,2,A)
```

Teniendo la codificación, la descompresión es “sencilla”. Tomando como referencia la primera tripleta, (0,0,A), la cual nos indica que se deben dar 0 pasos hacia atrás, coger el conjunto de longitud 0, es decir, vacío, y añadir el carácter A.

```
A _
(0,0,B) (2,2,C) (4,3,A) (2,2,A)
```

Siendo el “_” la siguiente posición a tratar, se cogerá la siguiente tripleta (0,0,B). Al igual que en la tripleta anterior, simplemente será añadir el carácter que se indica en la posición correspondiente.

```
A B _
(2,2,C) (4,3,A) (2,2,A)
```

Se cogerá la siguiente tripleta, (2,2,C), la cual representa “El *string* de longitud 2 que comienza 2 posiciones hacia la izquierda y añadiendo el carácter C al final”.

```
A B (A B C) _
(4,3,A) (2,2,A)
```

La siguiente tripleta (4,3,A), nos indica que se debe desplazar 4 posiciones hacia la izquierda, leer el *string* de longitud 3, y añadirle una “A” al final. Por tanto, el resultado sería el siguiente.

```
A B A B C (B A B A) _
(2,2,A)
```

Por último, la siguiente tripleta, (2,2,A), indica un desplazamiento de 4 posiciones hacia la izquierda, lectura de dos caracteres (“BA”), y añadirle una “A” al final. Por tanto, el resultado final de la descompresión, que es igual al flujo original, es.

```
A B A B C B A B A (B A A)
```

### 4.1.2. Lzma

LZMA[3] es uno de los algoritmos de compresión que utiliza el conocido programa de (des)compresión de datos 7-zip[4]. LZMA es uno de los muchos algoritmos que usan un diccionario basado en el algoritmo LZ77 que implementa GZip. Las principales diferencias entre ambos algoritmos son, el superior radio de compresión y la implementación de un tamaño variable de diccionario, siendo el tamaño máximo hasta 4GB. [Wikipedia, 2021]

A pesar de estar basado en el algoritmo LZ77, LZMA no es un algoritmo cuyo diccionario se basa únicamente en la agrupación de bytes, sino que especifica bits que indican el contexto en el que se encuentran esos bytes. De esta manera, se obtiene un mejor rendimiento en la compresión, ya que los bytes no están todos juntos sin mucha relación entre sí.

Los algoritmos LZMA y LZMA2, posteriormente descrito, no tienen una documentación clara sobre su funcionamiento, es decir, la mayoría de las implementaciones de estos algoritmos se han deducido a partir de otras implementaciones, siendo la utilizada por 7-zip la original.

El principal problema con el que tiene que lidiar este algoritmo es también una de sus mayores virtudes, el tamaño de diccionario. Un tamaño de diccionario tan grande, hasta 4 GB, permite comprimir gran cantidad de archivos, sin embargo, la búsqueda dentro de este diccionario puede ser algo tediosa. Al ser de un tamaño tan grande una búsqueda secuencial sería muy ineficiente, ya que volvería el algoritmo muy lento y poco práctico para la mayoría de usos. Por ello, LZMA utiliza su propio codificador, el cual puede decidir libremente con que *string* emparejar el flujo leído o simplemente ignorarlo.

### 4.1.3. Lzma2

LZMA2 es simplemente, una versión mejorada del algoritmo Lzma antes descrito. El creador de ambos algoritmos de compresión, Igor Pavlov, describe las siguientes virtudes principales de este algoritmo frente a LZMA [Pavlov, 2011].

- Mejora en la velocidad de compresión de archivos de más de 256 MB al utilizar 4 hilos de CPU o más.
- Mejora en la compresión de datos previamente comprimidos.

[3] Lempel-Ziv-Markov chain algorithm
[4] https://www.7-zip.org/

- El algoritmo fue creado para funcionar con el formato XZ, por lo que incluye modificaciones que mejoran la compresión de datos en ese formato.

Cabe mencionar que estas mejoras del algoritmo LZMA2 se obtienen utilizando mayor cantidad de memoria.

### 4.1.4. Lzss

LZSS[5] es un algoritmo de compresión sin pérdida creado en 1982. Este algoritmo, como los dos descritos anteriormente, está basado en el algoritmo LZ77, por tanto, es también un algoritmo basado en el uso de un diccionario. [Dipperstein, 2019]

Para poder comprimir los archivos o cadenas de caracteres, LZSS, trata de reemplazar un *string* por la referencia en el diccionario de la representación de ese *string*, similar a LZ77. Sin embargo, una de las diferencias frente a este algoritmo es el "tamaño" de esta referencia, el cual siempre será inferior al tamaño de la cadena a representar, en el algoritmo LZ77, esa condición no siempre se daba.

Por tanto, las principales diferencias entre LZSS y LZ77 se dan en la forma de almacenamiento del diccionario que ambos algoritmos utilizan para comprimir y descomprimir archivos, siendo la característica más significativa, de este primer algoritmo, la forma en la que se guardan los caracteres individuales, es decir, los *string* de un único elemento. Así, el algoritmo LZ77, cuando tenía que guardar una cadena de un solo elemento, almacenaba, **(0,0,a)**, sin embargo, el algoritmo LZSS guardaría únicamente el carácter "a", de manera que ahorra el espacio ocupado por el *offset* y el *length*.

Así mismo, otras mejoras que ofrece este algoritmo respecto a su predecesor son, la adición de un bit de control que permite determinar si la entrada del diccionario esta codificada o no, es decir, si se almacenó en el diccionario simplemente un carácter o si se almacenó en texto plano. En el último, caso no tendría un coste computacional.

### 4.1.5. Brotli

Brotli es el algoritmo más moderno de los que se describen en este documento. Fue desarrollado en 2013 por Jyrki Alakuijala y Zoltán Szabadka y publicado en el estándar

[5]Lempel Ziv Storer Szymanski

RFC[6] de la IETF[7]en 2016 [Alakuijala and Szabadka, 2016].

Brotli, es un algoritmo de compresión sin pérdida desarrollado para uso genérico y que está basado en el algoritmo LZ77 y la codificación Huffman y promete una eficiencia de (des)compresión equiparable a la de los algoritmos antes vistos. Su gran eficiencia sobre archivos relacionados con la WEB (HTML[8], CSS[9]) lo hacen perfecto para usarlo con navegadores como Mozilla Firefox[Fir, 2016] o Google Chrome[Acc, 2016].

La codificación de *Huffman* se basa en la asignación de códigos a los caracteres de entrada basados en la frecuencia de aparición de los caracteres anteriores. Cuanto más frecuente sea el carácter más pequeño será el código que se le asigne a ese carácter [GeeksforGeeks, 2012]. Por tanto, el objetivo de este algoritmo es, a partir de la tabla de relaciones carácter-repeticiones, crear un árbol estructurado considerando el número de repeticiones de los caracteres, los más frecuentes serán hojas cerca de la raíz y los menos frecuentes serán las hojas de mayor profundidad.

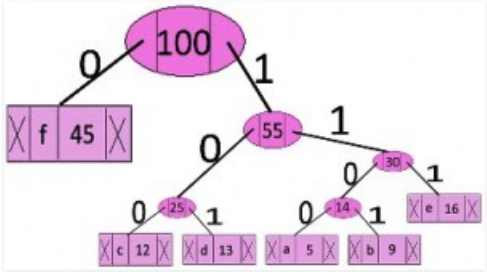



**Figura 4.1:** Árbol resultante de la codificación Huffman

| Carácter | Codificación |
|---|---|
| f | 0 |
| c | 100 |
| d | 101 |
| a | 1100 |
| b | 1101 |
| e | 111 |

**Tabla 4.1:** Codificación *Huffman* de los caracteres en base al árbol obtenido

[6]Request For Comments: https://www.ietf.org/standards/rfcs/
[7]Internet Engineering Task Force: https://www.ietf.org
[8]HyperText markup Language
[9]Cascading Style Sheet

En lo que al funcionamiento del algoritmo se refiere, los datos comprimidos se componen de una cabecera y una serie de meta-bloques. Cada uno de estos meta-bloques se descompone en una secuencia de 0 a 16MB sin comprimir. Los datos finales, sin comprimir, corresponden a la concatenación de las secuencias sin comprimir de cada meta-bloque. La cabecera, contiene los datos de la parte del *sliding window* utilizado.

Volviendo a los meta-bloques, cada uno de ellos esta comprimido mediante una combinación entre el algoritmo LZ77 y la codificación Huffman. Al resultado que se obtiene de la ultima codificación se le denomina "prefijo" y representa unívocamente a cada uno de los meta-bloques, semejante a la representación de clave primaria de una tabla de una BD. En cuanto al uso del algoritmo LZ77, se utiliza para codificar los contenidos de los meta-bloques, de manera que las cadenas sean referencias a otros meta-bloques.

## 4.2. Algoritmos de compresión con pérdida

A pesar de que para este proyecto se van a utilizar algoritmos de compresión sin pérdida, ya que las facturas a almacenar deben recuperarse en su formato original, de cara a comprender las distintas posibilidades de compresión que existen se ha decidido tratar algunos de los algoritmos de compresión con pérdida.

Por tanto, en esta sección se describen algunos de los algoritmos de compresión con pérdida (*lossy algorithms*) más utilizados. Estos algoritmos, generalmente utilizados en tareas relacionadas con la compresión de vídeos o imágenes, son capaces de obtener una aproximación parcial del archivo original.


**Figura 4.2:** Ejemplo de compresión de una imagen con pérdida

### 4.2.1. JPEG

JPEG (Joint Photographic Experts Group)[Wallace, 1992] es uno de los algoritmos más conocidos de compresión de imágenes, aunque muchos lo conocen por la extensión de archivo ".jpeg". Fue creado en 1992 por el grupo que lleva su nombre y se basa en la Transformada del Coseno Discreto (DCT) propuesta por Nasir Ahmed en 1972. El funcionamiento del algoritmo es el siguiente.

En primer lugar, se divide la imagen en bloques de 8x8 píxeles. Puesto que cada píxel está compuesto por los tres colores del modelo RGB[10], el siguiente proceso se aplica para cada uno de las 64 representaciones de color del RGB dentro del bloque de 8x8.

Una vez obtenido el bloque de intensidades del color rojo por ejemplo, se le aplica la Transformada del Coseno Discreto de dos dimensiones(TCD), antes mencionada, de manera que se obtengan los coeficientes que representan el bloque de 8x8 anterior. Con la tabla de coeficientes se pasa al proceso denominado "Cuantización", aquí, la tabla de coeficientes se "cuantiza" uniformemente con una tabla especificada por el usuario o aplicación. A esta tabla se la denomina "Tabla de Cuantización" y contiene 64 valores (8x8) entre 1 y 255. En función de los valores contenidos en esta tabla la imagen resultado tendrá una compresión diferente (Cuanto mayores los números mayor compresión y menor calidad.)


**Figura 4.3:** Niveles de compresión de JPEG

### 4.2.2. Algoritmos de Compresión de Texto

En lo que ha compresión de texto se refiere, en general, los esfuerzos se han centrado en utilizar técnicas de compresión sin pérdidas. Esto se debe a que generalmente, el texto

[10]Red Green Blue

suele ser parte de áreas críticas del sistema. Sin embargo, ¿Que pasa cuando el texto no pertenece a un área crítica?, ¿Es rentable utilizar algoritmos de compresión sin pérdida, los cuales pueden tener un mayor coste computacional, para estos campos? A continuación, se presentan tres sencillas técnicas capaces de comprimir un texto y cuya descompresión conlleva a una aproximación del texto original. [Palaniappan and Latifi, 2007]

La técnica **Letter Mapping (LMP)** consiste en el reemplazamiento de caracteres en todo el texto. El objetivo es reemplazar los caracteres con menos frecuencia de aparición, por otros con mayor frecuencia. Por ejemplo, en el alfabeto español, las consonantes con mayor porcentaje de aparición son *S, R,N, D* y *L*, mientras que las menos frecuentes son *J, Ñ, X, K* y *W*. La finalidad, por tanto, no es otra que sustituir los caracteres menos frecuentes por los más frecuentes, de manera que se reduzca el tamaño del alfabeto, en este caso por 5. De esta manera, si se aplica este método junto con la codificación de *Huffman*, ver 4.1.5, los caracteres que inicialmente eran menos probables, estarán ahora codificados con bits más pequeños. Tras descomprimir el texto, nos encontraremos con un gran número de palabras mal escritas, nada que no pueda solucionar un corrector ortográfico, aunque algunas palabras no serán las que se tenían originalmente.

La técnica **Dropped Vowels (DOV)** consiste en reemplazar las apariciones de todas las vocales que encontramos en un texto. En el alfabeto español las vocales son de las letras más frecuentes que nos encontramos, por lo que si eliminásemos las vocales se conseguiría una gran compresión, aunque el problema vendría al tratar de recuperar el texto original. Una posible solución es sustituir todas las vocales por una concreta, es decir, sustituir las vocales *e, i , o, u* por la vocal *a*, reduciendo de esta manera el tamaño de compresión y pudiendo recuperar la gran mayoría de palabras del texto utilizando un corrector ortográfico.

La última técnica **Replacement of Characters (ROC)**, consiste en sustituir grupos de caracteres por un único carácter que denote el mismo sonido o significado, exactamente igual que la taquigrafía. De esta manera, el texto descomprimido presentaría una gran variedad de errores, pero quedaría legible por cualquier persona.

## 4.3. Algoritmos de compresión utilizados por los SGBD

Conocer los algoritmos de compresión es importante, sin embargo, también hay que tener en cuenta que la gran mayoría de SGBD existentes tienen implementados algoritmos de compresión, pudiendo elegir el algoritmo a utilizar desde los parámetros de configuración de los SGBD.

En esta sección se explicarán los algoritmos que implementan los SGBD tratados en este proyecto (MongoDB y Cassandra).

### 4.3.1. MongoDB

El SGBD **MongoDB** utiliza tres librerías de compresión de datos, *Snappy*[11], *zlib*[12] y *zstd*[13], siendo la primera la utilizada por defecto.

#### Snappy

**Snappy** es un algoritmo de (des)compresión basado en el algoritmo LZ77 desarrollado por Google y liberado en 2011.[Google, 2021]. La idea de este algoritmo es conseguir una compresión aceptable en el menor tiempo posible. Es más, comparado con *zlib*, otro de los algoritmos que utiliza MongoDB, es mucho más rápido a la hora de comprimir aunque con un ratio de compresión menor.

Snappy, es también utilizado en otros SGBD, como puede ser Cassandra, el cual se verá a continuación, o Couchbase[14], un SGBD No-SQL enfocado a documentos y que utiliza formato JSON para almacenamiento.

#### ZLib

En cuanto a **zlib**, es una variación del algoritmo de (des)compresión *deflate* desarrollado por Phil Katz y basado en una combinación del algoritmo LZ77 y la codificación Huffman.

Hay una gran cantidad de aplicaciones conocidas que utilizan *zlib* como (des)compresor de datos[Gailly and Adler, 2002], entre ellas, el lenguaje de programación *Python* o la aplicación de comunicación *openSSH*.

En cuanto a su funcionamiento, básicamente es el mismo que el del algoritmo *deflate* antes mencionado y posteriormente explicado. En términos generales, el algoritmo divide el flujo de datos de entrada en bloques, y cada uno de estos bloques se comprime individualmente. Para realizar la compresión, *zlib* ofrece 3 formas distintas.[Gally and Adler, 2002]

[11] https://google.github.io/snappy/
[12] http://www.zlib.net/
[13] https://facebook.github.io/zstd/
[14] https://www.couchbase.com/

- **Sin comprimir.** Se utiliza esta opción cuando los datos de entrada ya están comprimidos de manera que no aumente mucho el tamaño del archivo.

- **Comprimido**. Primero se comprime con LZ77 y después se le aplica la codificación *Huffman*. En este caso, los árboles utilizados en la codificación *Huffman* son generados automáticamente por las especificaciones del algoritmo, por lo que no es necesario almacenarlos con los datos.

- **Comprimido**. Primero con LZ77 y luego se aplica la codificación *Huffman*. Los arboles son creados por el compresor, por tanto, son almacenados junto con los datos.

#### ZStd

En lo que a **zstd**[15] se refiere, es un algoritmo de compresión basado en el más que mencionado algoritmo LZ77, desarrollado por Yann Collet en 2016, trabajador de Facebook [Collet, 2016]. El objetivo, era crear un algoritmo que no fuese dependiente de la potencia de la CPU, del sistema operativo o de los caracteres utilizados.

Los datos comprimidos por el algoritmo se componen de lo que denominan *Zstandard Frame*, siendo su formato el que se describe en la siguiente tabla.

| Magic_Number | Frame_Header | Data_Block | Content_Checksum |
|---|---|---|---|
| 4 bytes | 2-14 bytes | n bytes | 0-4 bytes |

**Tabla 4.2:** Formato de los *frames* de ZSTD

El contenido de estos campos de los *frames* es el siguiente:

- **Magic_Number.** Contiene el identificador del bloque, por ejemplo, 0xFD2FB528, Sigue el formato *little-endian*[16] evitando además patrones triviales, como 0x00 y valores fuera del espacio de UTF-8, de esta manera se reduce considerablemente la posibilidad de que un archivo de texto represente este valor por accidente.

[15] Zstandard

[16] Forma de almacenamiento de los bits en memoria, en este formato los bits menos significativos se almacenan en las posiciones más bajas de la memoria.

- **Frame_Header.** Este conjunto de *bytes* contiene diferentes *flags* de información sobre el *frame*, así como otros campos opcionales, como la ID del diccionario a utilizar para ese *frame*.
- **Data_Block.** Todos los *frames* se componen de uno o más bloques de datos. Cada uno de ellos se compone de una subestructura concreta que contiene, los datos y otro tipo de información como el tamaño del bloque o el tipo de bloque (Información descomprimida, información comprimida, bloque reservado...).
- **Content_Checksum.** Contiene el resultado de aplicar la función hash **xxh64**[17] a los datos decodificados y con una semilla fijada en 0. Este campo es opcional y para utilizarlo se debe habilitar el *flag* correspondiente.

De estos campos, destaca el campo *Data_Block* cuya codificación de los bloques se realiza de la siguiente forma: se utiliza la denominada *Entropy Encoding* o Codificación de Entropía, en concreto dos tipos, la FSE[18] y la codificación *Huffman*. Esta última se utiliza para codificar literales, mientras que la FSE se utiliza para codificar todos los otros símbolos.

### 4.3.2. Cassandra

De los dos SGBD que se tratan en este proyecto, **Cassandra** es el que más opciones de compresión ofrece. [Apache, 2016] Algunas de ellas ya se han comentado en apartados anteriores, este es el caso de *zlib*, *Snappy* y *zstd*. Sin embargo, Cassandra también ofrece otras opciones de compresión, como son los algoritmos **Deflate**, **LZ4**[19] y **LZ4HC**, el cual es simplemente una versión de LZ4 con una mayor compresión (*High Compression*).

#### DEFLATE

**DEFLATE** es un algoritmo de compresión sin pérdida basado en el algoritmo LZ77. Fue creado por Phil Katz y publicado en el RFC en 1996 por Peter Deutsch [Deutsch and Katz, 1996].

El algoritmo comprime el flujo de datos entrante en una serie de bloques. Cada uno de los bloques se comprime mediante una combinación del algoritmo LZ77 y la codificación *Huffman*.

[17] http://cyan4973.github.io/xxHash/
[18] *Finite State Entropy* o Entropía de Estado Finito
[19] http://lz4.github.io/lz4/

Los arboles de *Huffman* de cada uno de los bloques son independientes entre sí, mientras que el algoritmo LZ77 puede hacer referencia a una cadena duplicada en un bloque anterior. De esta manera, se realiza una compresión eficiente de los datos de entrada.

#### LZ4

El algoritmo LZ4 tiene un gran parecido al algoritmo ZSTD antes descrito, habiendo sido ambos algoritmos desarrollados por Yann Collet. El algoritmo funciona con una estructura de *frames*, los cuales se representan de la siguiente forma.[Collet, 2020]

| Magic Number | Frame Descriptor | Block | End Mark | Content Checksum |
|---|---|---|---|---|
| 4 bytes | 3-15 bytes | n bytes | 4 bytes | 0-4 bytes |

**Tabla 4.3:** Estructura de los *frames* del algoritmo LZ4

- **Magic Number.** Representación, utilizando el formato *little-endian*, de los bytes de identificación de *frame*.
- **Frame Descriptor.** Contiene información relativa a la configuración del *frame*. Entre esta información están los *flags* que indican si los bloques son independientes entre sí o deben decodificarse de forma secuencial, o el *flag* de identificación del diccionario a utilizar para la descompresión.
- **Data Blocks.** Contenedor de los datos. Los 4 primeros bytes representan el tamaño del bloque, además a través del valor del bit más significativo se indica el estado de los datos (comprimidos o descomprimidos).
- **End Mark.** Indica el final del flujo de bloques mediante 32 bits fijados a 0 (0x00000000).
- **Content Checksum.** Contiene el resultado de aplicar la función hash, xxHash-32, a los datos iniciales.

## 4.4. Selección del algoritmo de compresión

Tras presentar la teoría detrás de cada uno de los algoritmos de compresión se ha seleccionado el algoritmo **GZIP** para comprimir las facturas. Tras evaluar los diferentes algoritmos se ha seleccionado este debido al gran rendimiento que ofrece a la hora de comprimir y

descomprimir las facturas. Además este gran rendimiento se combina con un elevado ratio de compresión, cercano al 70 %. Para ver las pruebas relacionadas con la selección del algoritmo ver la sección 7.2.1, Pruebas sobre el diseño.

En cuanto a los algoritmos de los SGBD seleccionados se ha decidido utilizar el mismo en ambos, *Snappy*. El motivo por el cual se ha seleccionado este algoritmo ha sido porque es el que mejor ratio de compresión ha ofrecido al comprimir los datos almacenados en las BDs. Por algún motivo, los otros algoritmos perdían rendimiento al tratar de comprimir datos ya comprimidos previamente.

# 5. CAPÍTULO

# Sistemas de Gestión de Bases de Datos

De cara a comprender mejor los esquemas de Bases de Datos definidos para cada uno de los SGBDs, durante este capítulo se realizará una breve descripción de las características y funcionamiento de los dos SGBDs NoSQL tratados en este proyecto, MongoDB y Cassandra.

## 5.1. SQL frente a NoSQL

En esta sección se describirán las principales características de los SGBDs relacionales y no relaciones. Así mismo, se detallarán algunas de las principales diferencias que existen entre ambos tipos de SGBDs

### 5.1.1. Bases de Datos SQL

SQL (*Structured Query Language*) [W3Schools, 2021] es un lenguaje definido para acceder y modificar Bases de Datos relacionales. Estas Bases de Datos, están basadas en el modelo relacional, propuesto originalmente por E.F. Codd en 1970. [Codd, 2002]

El modelo relacional, organiza los datos en tablas formadas por filas y columnas, a cada una de las filas se la denomina “tupla”. A una, o varias, de las columnas de las tuplas se la denomina clave primaria o *Primary Key* y contiene un identificador unívoco de esa tupla

dentro de la tabla. Las columnas restantes, representan todas las características a guardar de cada una de las tuplas de la tabla.

Algunos de los SGBD más conocidos de este tipo de Bases de Datos son Oracle, MySQL, PostgreSQL, etc.

### 5.1.2. Bases de Datos NoSQL

Las Bases de Datos NoSQL (*Non SQL* o *Not Only SQL*) surgieron durante el año 2000, con el objetivo de crear Bases de Datos escalables, que ofrecieran respuestas rápidas a las consultas formuladas y fáciles de adaptar a los constantes cambios en las aplicaciones [Schaefer, 2021].

En general hay diferentes tipos de SGBD NoSQL, siendo las más conocidas las siguientes.

- **Documentos.** Se enfocan a un uso genérico, la idea es almacenar los datos en documentos, por ejemplo de tipo *JSON*. Este es el caso del popular SGBD MongoDB.
- **Clave-Valor.** Tiene un funcionamiento similar a las tablas Hash o los diccionarios, en los que una clave única se asocia a una colección de objetos. Un SGBD que sigue esta implementación es DynamoDB.
- **Tabular.** En lo que al almacenamiento y visualización de datos se refiere, los SGBD de este tipo tienen una estructura parecida a las clásicas SQL. Almacenan los datos en tablas que tienen filas y columnas dinámicas. Este tipo de Bases de Datos se enfoca principalmente a aplicaciones con gran volumen de datos con patrones de consultas conocidos. En este tipo entra el SGBD Cassandra.
- **Orientadas a Grafos**. En general, son SGBD enfocados a tratar de manera eficiente relaciones entre los datos, por tanto, sirven para analizar datos conectados entre sí. En ese grupo de SGBD destaca Neo4J.

### 5.1.3. Diferencias entre ambas

Al comparar SGBD relacionales y SGBD NoSQL, la principal diferencia es la escalabilidad en los datos. A pesar de que los SGBD relacionales llevan ya mucho tiempo en el mercado, en la mayoría de los casos, cuando se habla del almacenamiento y gestión de un volumen

de datos muy grande, los SGBD NoSQL ofrecen prestaciones superiores a los SGBD relacionales. [Schaefer, 2021]

Otra de las diferencias principales entre los SGBD relaciones y los SGBD NoSQL es el no mantenimiento de los principios *ACID* por parte de los segundos. [Dave, 2012]. El principio *ACID* (**A**tomicidad, **C**onsistencia, a**I**slamiento y **D**urabilidad) es el principio en el que se basan las Bases de Datos Relacionales y que garantizan, que tras realizar una serie de transacciones, el estado de la Base de Datos siga siendo consistente.

Mientras que el principio *ACID* se refiere a los SGBD relacionales, los SGBD NoSQL deben cumplir el principio *BASE* (**B**asic **A**vailability, **S**oft State y **E**ventual Consistency). Todo SGBD NoSQL cumple estas tres propiedades que garantizan, la disponibilidad de los datos a pesar de la aparición de varios fallos, la consistencia de los datos debe ser labor del programador y no del sistema, y que la consistencia se establecerá en un punto del futuro.

Así mismo, existe el teorema *CAP* (**C**onsistencia, **A**vailability o Disponibilidad y **P**artition Tolerance o Tolerancia a Fallos), el cual defiende que todo sistema distribuido garantiza el cumplimiento de dos de las características del teorema. Cada sistema se diseñará en función de las características que quiera reforzar.

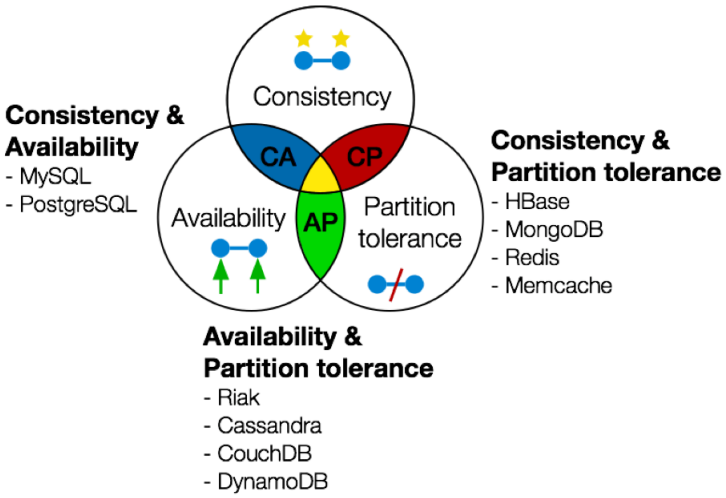



**Figura 5.1:** Esquema del teorema CAP

Por ejemplo, MongoDB se sitúa en la categoría **C+P**, por lo que en todo momento garantizará consistencia y comunicación entre todos los nodos, aunque los datos pueden no estar siempre disponibles.

## 5.2. MongoDB

MongoDB[1] es un sistema de gestión de bases de datos NoSQL enfocado a documentos y de código abierto. Al ser un SGBD enfocado a documentos, los datos se guardan en las Bases de Datos creadas con dicho sistema no en forma de tablas, como en las bases de datos relacionales, sino que se guardan en documentos de tipo JSON.

```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  },
  "hobbies": ["surfing", "coding"]
}
```

**Listado 5.1:** Documento de ejemplo JSON almacenado en MongoDB

Al guardar los documentos en este formato permite [MongoDB, 2021a], entre otras características.

- **Flexibilidad en los esquemas de los documentos**. El modelado enfocado a documentos de MongoDB permite a los desarrolladores modelar y manipular fácilmente los datos. Así mismo, al utilizar el formato JSON, permite tener diferentes campos en distintas entradas de una misma colección de documentos.

- **Acceso a datos de código nativo**. La gran mayoría de SGBD relacionales utilizan librerías de mapeo, las cuales convierten los datos de las tablas de la BD a objetos del propio lenguaje, como es el caso de *Hibernate*[2] para Java. Sin embargo, al ser una BD de documentos, MongoDB no necesita estas librerías, ya que sus estructuras de datos son accesibles mediante las estructuras propias de cada lenguaje de forma nativa.

- **Consultas y analíticas potentes**. MongoDB esta diseñada para construir una BD

[1] https://www.mongodb.com/
[2] https://hibernate.org/

en la que el acceso a datos sea fácil y sencillo y que rara vez se precise realizar operaciones como los costosos *joins* de SQL.

## 5.3. Cassandra

Cassandra[3] es un sistema de gestión de bases de datos NoSQL tabular, es decir, estructura los datos de forma similar a los SGBD relacionales, utilizando tablas, filas y columnas, aunque el formato de las columnas puede variar en cada una de las filas.

A continuación se listan algunas de las características que han hecho a Cassandra uno de los SGBD NoSQL más utilizados. [DataFlair, 2018]

- **Open Source.** Uno de los principales atractivos de este SGBD es que se trata de un proyecto de código abierto de Apache, gracias a ello, Cassandra cuenta con una enorme comunidad detrás.
- **Arquitectura *Peer-to-peer***. Cassandra almacena los datos en nodos, los cuales están distribuidos en forma de anillo. Los datos se replican en 1 a 3 nodos, de manera que no existe un nodo "maestro". Además, esto hace que no exista un único punto de fallo.
- **Gran escalabilidad.** Esta es una de las grandes características de Cassandra. Cada uno de los anillos permite mejorar la escalabilidad, ya que en cualquier momento se pueden añadir o eliminar cualquier número de nodos.
- **Alta disponibilidad y tolerancia a fallos.** Gracias a la replicación de los datos, Cassandra ofrece una alta disponibilidad y una fuerte tolerancia a fallos. Si un nodo falla los datos estarán disponibles en todo momento en otros nodos. Al número de copias de los datos dentro de un mismo anillo se le denomina *replication factor* o factor de replicación. Para un factor de replicación de 2, por ejemplo, existirán dos copias por cada una de las filas de datos, estando cada copia en un nodo diferente.
- **Consistencia modificable**. [DataStax, 2021] Cassandra ofrece distintos niveles de consistencia, aunque se pueden reducir a dos tipos generales. Un primer tipo denominado consistencia eventual, el cual, en cuanto el *cluster* acepta la escritura del dato, el sistema se asegura de recibir la aprobación del cliente. El segundo tipo,

[3] https://cassandra.apache.org/

consistencia fuerte, se asegura de que las escrituras en un nodo se notifican al resto de nodos en los que los datos están replicados. No obstante, no es necesario elegir un único modo, sino que se puede realizar una mezcla de ambos.

# 6. CAPÍTULO

# Diseño de la aplicación

En el siguiente capítulo se detalla por un lado, el contenido que debe ser almacenado en cada una de las bases de datos a diseñar, y por otro lado, un primer esquema del diseñado en ambos Sistemas de Gestión de Bases de Datos.

## 6.1. Formato de las facturas

Como ya se ha comentado, el objetivo del proyecto consiste en diseñar e implementar una base de datos, en diferentes SGBD, capaz de almacenar, de forma eficiente, los datos referentes a las facturas emitidas por diferentes entidades de Gipuzkoa (TicketBAI). Por ello, una buena forma de orientar el diseño de las Bases de Datos es conocer el formato en el que se reciben los datos. En el caso de TicketBai, los datos pertenecientes a una factura se reciben en formato *XML*[1].

Estas facturas *XML* siguen el formato, de un *XMLSchema*, proporcionado por la Diputación Foral. Cada factura, esta formada por cinco bloques, **Cabecera**, **Sujetos, Factura, HuellaTBAI** y ***Signature*** [Diputación Foral, 2021].

[1] eXtensible Markup Language

### 6.1.1. Cabecera

El bloque Cabecera de la factura contiene, simplemente, un número identificativo de la versión de la estructura del fichero TicketBAI utilizado.

### 6.1.2. Sujetos

Este bloque contiene los datos relativos al emisor y destinatario(s) de las factura. Del emisor, se guardan valores como el NIF o la denominación social del emisor.

Para cada uno de los posibles destinatarios de la factura, se guarda el NIF en caso de que sea un destinatario nacional, o el país al que corresponde el número de identificación extranjero.

### 6.1.3. Factura

El bloque Factura contiene los datos relacionados con la fecha de expedición, los productos adquiridos, etc. Es el bloque más extenso de toda la factura y se divide en tres partes.

#### Cabecera Factura

Este primer sub-bloque contiene los datos que identifican una factura concreta, estos campos son, el número de factura y la fecha y hora de expedición de la factura. Así mismo, también se almacenan en este bloque otros valores que indican si una factura es simplificada o si es una factura rectificativa.

#### Datos Factura

Este sub-bloque contiene, principalmente, los detalles de la factura, es decir, los bienes o servicios adquiridos y que quedan recogidos en la factura. Por cada uno de ellos, se almacena información relativa al importe con y sin IVA aplicado, o una descripción del producto.

Además, también se almacena el importe total de la factura o valores que indican el tipo de IVA aplicado a la factura.

#### Tipo Desglose

Cada una de las facturas tiene la opción de ofrecer un desglose del IVA de cada uno de los detalles que conforman la mencionada factura. El desglose que se ofrece puede ser de dos tipos diferentes, *Desglose de factura* o *Desglose por tipo de operación*. El primero se enfoca a un desglose más genérico, mientras que el segundo diferencia el desglose para servicios prestados y productos materiales.

Independientemente del tipo de desglose se diferencian dos casos, el primero, el caso menos común, aquel en el que el producto o productos no estén sujetos al impuesto por la causa posteriormente indicada, el segundo, el caso quizás más común, aquel en el que los productos de la factura estén sujetos al pago de IVA. Aún así, en este segundo caso existe la posibilidad de que algunos de los productos o servicios estén exentos del pago de IVA, indicando en su caso la causa y la base imponible exenta del pago de este impuesto. Para los productos o servicios que no están exentos del pago del IVA se agruparán por su tipo de operación y posteriormente por el tipo impositivo que se aplica a los productos.

### 6.1.4. HuellaTBAI

En este bloque se guarda la información relacionada con el *software* que ha emitido la factura, entre estos datos se encuentran campos como el número de alta-inscripción en el registro del Software TicketBAI o la versión del *software* utilizada.

### 6.1.5. Signature

El bloque *Signature* contiene la firma electrónica de las facturas TicketBAI. Dada la normativa vigente, las facturas XML deben ser firmadas en el el formato XAdES[2]. Este formato es el conjunto de extensiones a las recomendaciones definidas por *XML-DSig* de manera que sean adecuadas para firmar documentos XML electrónicamente según el reglamento Nº 910/2014 del Parlamento Europeo [Parlamento Europeo, 2014].

*XML-DSig* o *XML Digital Signature* define la sintaxis y reglas de procesamiento para firmar documentos XML digitalmente. Estas reglas y sintaxis están definidas en el documento correspondiente de la W3C [W3C, 2013]. La estructura básica del campo de firma de un documento es la siguiente.

[2] ***X**ML **Ad**vanced **E**lectronic **S**ignatures*

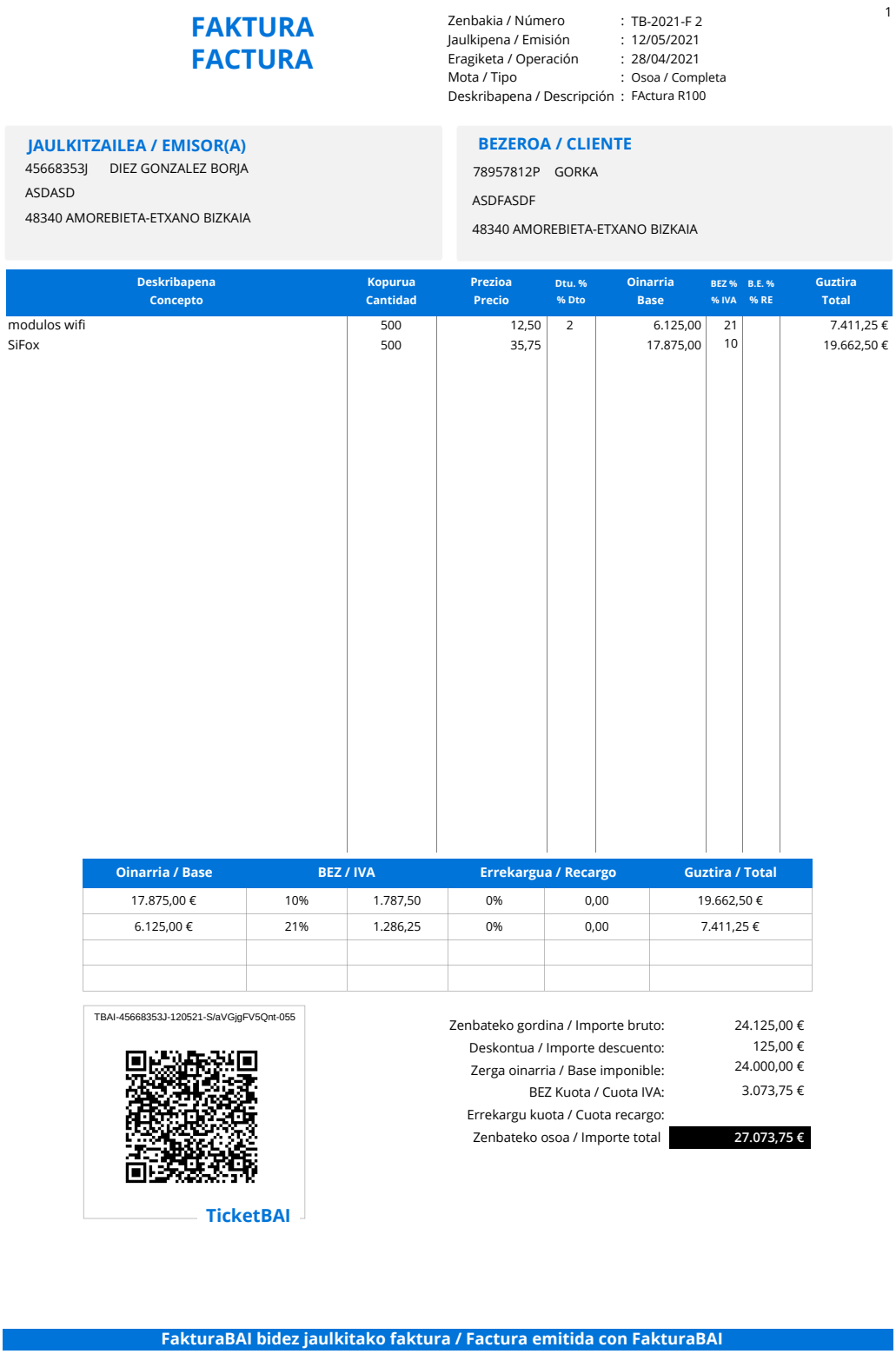
```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference>
            <Transforms />
            <DigestMethod />
            <DigestValue />
        </Reference>
        <Reference /> etc.
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

**Listado 6.1:** Estructura básica del campo *Signature* de un documento XML

El listado 6.1 contiene la estructura básica de la firma. No obstante, el tipo de firma XAdES ofrece diferentes tipos de formato de firma. Estos tres son, *Detached*, *Enveloping* y *Enveloped*. De estos tres, el formato solicitado por la Dipuatación Foral y a utilizar es el último, siendo las diferencias entre los tres las siguientes. [Administración Electrónica, 2021]

- **Detached.** Una firma XML en este formato permite tener la firma de forma separada e independiente del contenido firmado, de manera que la firma se relaciona con el contenido firmado mediante una referencia de URI. El uso más común de este tipo de firma es en la descarga de ficheros.
- **Enveloping.** En este formato de firma la única estructura existente es la de la propia firma. El contenido firmado se encuentra dentro de un elemento denominado *Object*.
- **Enveloped**. Este formato permite a un documento XML contener su propia firma digital. Por tanto, mientras que en los casos anteriores se podía firmar cualquier tipo de documento, mediante este formato de firma solo se podrán firmar documentos XML.

A continuación se muestra un ejemplo de factura sencilla emitida mediante la aplicación TicketBAI.



FAKTURA
FACTURA

1

Zenbakia / Número : TB-2021-F 2
Jaulkipena / Emisión : 12/05/2021
Eragiketa / Operación : 28/04/2021
Mota / Tipo : Osoa / Completa
Deskribapena / Descripción : FActura R100

JAULKITZAILEA / EMISOR(A)
45668353J DIEZ GONZALEZ BORJA
ASDASD
48340 AMOREBIETA-ETXANO BIZKAIA

BEZEROA / CLIENTE
78957812P GORKA
ASDFASDF
48340 AMOREBIETA-ETXANO BIZKAIA

| Deskribapena / Concepto | Kopurua / Cantidad | Prezioa / Precio | Dtu. % / % Dto | Oinarria / Base | BEZ % / % IVA | B.E. % / % RE | Guztira / Total |
|---|---|---|---|---|---|---|---|
| modulos wifi | 500 | 12,50 | 2 | 6.125,00 | 21 | | 7.411,25 € |
| SiFox | 500 | 35,75 | | 17.875,00 | 10 | | 19.662,50 € |

| Oinarria / Base | BEZ / IVA | | Errekargua / Recargo | | Guztira / Total |
|---|---|---|---|---|---|
| 17.875,00 € | 10% | 1.787,50 | 0% | 0,00 | 19.662,50 € |
| 6.125,00 € | 21% | 1.286,25 | 0% | 0,00 | 7.411,25 € |
| | | | | | |
| | | | | | |

TBAI-45668353J-120521-S/aVGjgFV5Qnt-055

TicketBAI

Zenbateko gordina / Importe bruto: 24.125,00 €
Deskontua / Importe descuento: 125,00 €
Zerga oinarria / Base imponible: 24.000,00 €
BEZ Kuota / Cuota IVA: 3.073,75 €
Errekargu kuota / Cuota recargo:
Zenbateko osoa / Importe total 27.073,75 €

FakturaBAI bidez jaulkitako faktura / Factura emitida con FakturaBAI

**Figura 6.1:** Ejemplo de factura emitida

## 6.2. Consultas más significativas

Un primer paso relevante para el diseño de los esquemas de Bases de Datos es el análisis de requisitos. Así, conocer con anterioridad algunas de las consultas, o al menos las más significativas, resulta necesario para realizar una definición adecuada del esquema en cada uno de los SGBD, MongoDB y Cassandra.

Por ello, tras una serie de reuniones con personas del departamento de Hacienda de la Diputación Foral, en las que se obtuvo la información necesaria para realizar el diseño, se seleccionaron las siguientes consultas.

#### Obtener una factura mediante el código QR

**Frecuencia de consulta estimada:** Muy frecuente. Cientos de miles de consultas diarias.

Todas las facturas emitidas mediante el software TicketBAI vienen con un código QR. Según las especificaciones establecidas, el código QR contiene el siguiente enlace `https://tbai.egoitza.gipuzkoa.eus/gr/`[3], junto con los siguientes parámetros.

- **id**. Identificador TicketBAI, este identificador determina unívocamente cada una de las facturas. El identificador se forma de la siguiente manera.
  - El texto **TBAI**.
  - Los nueve caracteres del NIF de la persona o entidad emisora de la factura.
  - La fecha de expedición de la factura. Esta se representa en formato DDMMAA.
  - Los trece primeros caracteres del campo *SignatureValue* del fichero XML, es decir, el campo que contiene la firma de la factura.
  - Tres caracteres que se corresponen con un código de detección de errores.

  Cada uno de los elementos listados del identificador va separado por el carácter "-", siendo el esquema genérico:
  TBAI-NNNNNNNNN-DDMMAA-FFFFFFFFFFFFF-CRC
- **s**. Representa la serie que sigue la factura. El contenido de este parámetro vendrá determinado por el campo *SerieFactura* del fichero XML que contiene los datos de la factura.

[3]Este es el enlace establecido para Gipuzkoa, en Bizkaia el enlace será `https://batuz.eus/QRTBAI/`, mientras que en Araba será `https://ticketbai.araba.eus/TBAI/QRTBAI`.

- **nf**. Representa el número de la factura emitida. Se corresponde con el campo *NumFactura* del fichero XML que contiene los datos de la factura.
- **i**. Contiene el importe total de la factura

El objetivo de esta consulta es permitir a los clientes comprobar que los datos de la factura que han recibido han sido anotadas correctamente en el sistema TicketBai.

#### Obtener las facturas emitidas por un NIF

**Frecuencia de consulta estimada:** Frecuente. Cientos de miles de consultas mensuales.

Como es de esperar, uno de los principales tipos de consultas es la obtención de las facturas por su NIF. Entre este tipo de consultas la que más destaca es la obtención de facturas emitidas por un NIF en unas determinadas fechas.

Así mismo, el objetivo de las consultas de este tipo es servir de "repositorio" a los usuarios de la aplicación. De esta manera, en todo momento podrán comprobar las facturas emitidas cada uno de los días.

#### Obtención de datos estadísticos

**Frecuencia de Consulta estimada:** Poco frecuente. Pocas consultas mensuales pero realizando cálculos con un gran volumen de datos.

Una de las características que deben permitir realizar los diseños, es la obtención de datos estadísticos. Uno de los propósitos de TicketBAI es permitir a los emisores de facturas ver datos estadísticos que permitan comparar su negocio con otros negocios del mismo. A pesar de que estas consultas sean poco frecuentes el diseño realizado debe permitir el cálculo de las diferentes estadísticas a realizar. Algunos ejemplos de este tipo de consultas serían, "*¿A qué países exportan principalmente las empresas de mi sector?*" o "*¿Cual es el importe medio de los tickets que he emitido esta semana?*.

#### Agrupación de facturas

**Frecuencia de consulta estimada:** Media. Miles de consultas mensuales.

A pesar de que no es una consulta como tal, de cara a obtener un mejor almacenamiento de las facturas se ha decido realizar una agrupación de la mismas. Es decir, pasado un

periodo de tiempo preestablecido, por ejemplo un año, las facturas almacenadas en la base de datos se moverán a una nueva tabla o colección en la que se agruparán todas las facturas emitidas en ese mismo año. De esta manera, tras comprimir el conjunto anual de facturas se reduce considerablemente el espacio ocupado en la base de datos. Así mismo, también se analizará el interés de realizar otros niveles de agrupación: mensual y trimestral.

## 6.3. Esquema de la aplicación en Cassandra

Durante este sección se presentan las distintas tablas definidas para la Base de Datos del SGBD Cassandra.

| **Partition Key** | | **Clustering Key** | **Data** | | |
|---|---|---|---|---|---|
| Nif Emisor | Fecha Exp | Ident-TBAI | SerieFact | Importe | Num Fact |

**Tabla 6.1:** Tabla 1 Cassandra

La tabla 6.1 muestra los datos que contendrá la tabla a responder a la consulta relacionada con el QR. Debido a la alta frecuencia estimada sobre estos datos se ha decidido crear una tabla que gestione la carga.

La clave de partición (*Partition Key*, la cual permite un acceso rápido a los datos) estará compuesta por el NIF del emisor de la factura y la fecha en la que está expedida. Dentro de esta agrupación, se ordenarán las facturas (*Cluestering key*) mediante el Identificador TicketBai. En cuanto a los datos guardados en esta tabla, son aquellos que se mostrarán al realizar la consulta sobre el QR.

| **Partition Key** | | **Clustering Key** | | **Data** | | |
|---|---|---|---|---|---|---|
| Nif Emisor | Fecha Exp | Hora Exp | Importe | Descripción | Detalles | xml |

**Tabla 6.2:** Tabla 2 Cassandra

Esta segunda tabla viene a responder a las otras dos consultas principales. La tabla permitirá, no solo obtener los datos más significativos de las facturas emitidas por un NIF en una fecha determinada, sino que también contiene los datos principales que permitirán realizar los estudios estadísticos deseados por el cliente. A destacar el campo *xml* el cual contiene la factura completa comprimida a la que corresponden los datos.

| Partition Key | | | Clustering Key | | Data |
|---|---|---|---|---|---|
| Nif Emisor | Fecha Inicio | Fecha Fin | Fecha Inicio | Fecha Fin | Agrupación |

**Tabla 6.3:** Tabla 3 Cassandra

Esta tercera y última tabla permitirá el almacenamiento de las facturas agrupadas. La tabla contendrá el NIF emisor de las facturas, la fecha en la comienza la agrupación, la fecha en la que finaliza y el conjunto de facturas emitidas en ese rango de fechas comprimidas. De esta manera, para obtener una factura "antigua", se accedería a la fila correspondiente de la tabla y se buscaría la factura en la agrupación de facturas correspondiente. Esta última búsqueda se realizará mediante una característica común de todos los ficheros XML, su cabecera.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

**Listado 6.2:** Cabecera común a todos los ficheros xml

## 6.4. Esquema de la aplicación en MongoDB

Mientras que el rendimiento que ofrece el SGBD Cassandra es muy dependiente de las consultas que se realizarán sobre la base de datos, MongoDB ofrece una mayor flexibilidad. Por ello, en base a las consultas más significativas definidas en la sección 6.2, se ha decidido crear una única colección que contiene los datos a los que se accederán con mayor frecuencia junto con la factura original comprimida. De esta manera, el esquema resultante sería el mostrado en el anexo B.1.

El primer campo, *_id*, es un identificador propio de MongoDB, el cual identifica unívocamente al objeto en cuestión, en este caso, el valor de este campo será el identificador TicketBai, el cual tenía el formato *TBAI-NNNNNNNNN-DDMMAA-FFFFFFFFFFFFF-CRC*. Además, las consultas que se realicen sobre este campo serán extremadamente rápidas, ya que MongoDB aplica un índice sobre este campo por defecto.

A continuación aparecen campos para los cuales se ofrece un acceso rápido (*NIF, FechaExpediciónFactura,* etc), a estos campos se les denominará campos en *RAW*. El objetivo de tener estos campos será responder rápidamente a consultas como la del QR.

El último campo, *FacturaComprimida*, contendrá el resultado de comprimir la factura

utilizando el algoritmo de compresión GZip. De esta manera, se tendrá la factura completa almacenada, de forma eficiente, preparada para ser utilizada en caso de tener que realizar consultas sobre algunos de los campos no establecidos como prioritarios.

Así mismo, para poder responder adecuadamente a la consulta correspondiente a la agrupación de las facturas, se ha decidido crear una nueva colección dedicada únicamente a ello. El esquema de esta colección es el mostrado en el anexo B.2.

Esta colección se compone de 3 campos. El primero, el campo identificador, que debe ser único, contiene el NIF emisor de las facturas, la fecha de inicio de la agrupación y la fecha fin. El formato, quedaría de la siguiente forma "NIF/DD-MM-AAAA/DD-MM-AAAA".

Para facilitar la búsqueda de una factura concreta dentro de la agrupación de facturas se utilizará el campo *idents*, el cual contiene los identificadores TicketBai, ordenados, de cada una de las facturas que se han agrupado.

## 6.5. Variaciones en los diseños

En esta sección se describen algunas de las variaciones en los diseños debido a posteriores reuniones con las personas del departamento de Hacienda de la Diputación Foral. En la sección 7.3 se describirán aquellas variaciones relacionadas con las pruebas ejecutadas.

### 6.5.1. Corrección en Cassandra

Como ya se adelantaba, debido a posteriores reuniones con los promotores del proyecto, se llegó a la conclusión de que separar las tablas 6.1 y 6.2 suponía una división innecesaria. Mientras que las consultas sobre el QR, junto con los datos correspondientes, se realizarán con alta frecuencia, las consultas sobre otros datos de la factura no serán tan frecuentes como para mantener una tabla dedicada a responder esa consulta. Así mismo, en el siguiente capítulo, se detallan pruebas que comparan los tiempos de obtención de datos de la factura comprimida a obtenerlos directamente de la base de datos. De esta forma, las tablas mencionadas anteriormente quedarían unidas de la siguiente forma.

| **Partition Key** | | **Clustering Key** | **Data** | | | |
|---|---|---|---|---|---|---|
| Nif Emisor | Fecha Exp | Ident-TBAI | SerieFact | Num Fact | Importe | xml |

**Tabla 6.4:** Nueva tabla que sustituye a las tablas no relacionadas con las agrupaciones de las facturas

De esta manera, el esquema en MongoDB y Cassandra sería muy similar, cada una de las Bases de Datos distribuirá sus datos en dos tablas o colecciones, una para las consultas sobre facturas unitarias recientes y otra para facturas guardadas en una agrupación de varias facturas.

# 7. CAPÍTULO

## Pruebas sobre el diseño

Durante este capitulo se detallarán las pruebas de rendimiento diseñadas de cara a probar el correcto funcionamiento del sistema. Así mismo, también se mostrarán los resultados obtenidos en la ejecución de las pruebas junto con los cambios realizados, en caso de que fuese necesario.

En cuanto al tipo de pruebas de rendimiento, únicamente se han realizado pruebas relacionadas con los tiempos de respuesta a la hora de recuperar los datos de diferentes formas, y la escalabilidad de estos datos. El motivo de esta elección ha sido que el sistema en el que se han hecho las pruebas no es el sistema final, por tanto, los resultados de tipo de pruebas relacionadas con la evaluación de la carga máxima que soporta el sistema o tests de resistencia no serían relevantes.

Tras analizar diferentes metodologías de *testing* de rendimiento de los sistemas, el esquema utilizado, el cual es el más extendido, ha sido el siguiente. [Guru99, 2021]

1. **Identificar del entorno de testeo.** Identificar las capacidades y limitaciones de la máquina en la que se ejecutarán las pruebas.

2. **Determinar los criterios de desempeño.** Identificar los criterios en los que se considerará válida la prueba o pruebas a realizar.

3. **Planificar y diseñar los tests de rendimiento.** Planificación del campo a probar y posterior diseño de los datos a insertar para probar el campo antes mencionado.

4. **Configurar el entorno de pruebas.** Preparación de las herramientas que se emplearán para ejecutar y analizar las pruebas.

5. **Implementar los tests.** Configuración del generador de facturas para que genere facturas con las características planificadas.

6. **Ejecutar los tests.** Ejecución de las pruebas, en este caso se realizarán las consultan e inserciones diseñadas a cada una de las bases de datos.

7. **Analizar, realizar cambios y retestear.** Análisis de los resultados obtenidos en los tests, en caso de no obtener un resultado satisfactorio se realizarán los cambios pertinentes y se ejecutarán las pruebas de nuevo.

## 7.1. Generación de datos sintéticos

Al existir problemas para trabajar con datos reales por un lado, por temas relacionados con la confidencialidad de la información manipulada, y por otro lado, porque todavía la institución cuenta con un número reducido de datos reales, antes de realizar las pruebas se tuvo que realizar la tarea de generación de datos sintéticos. Para ello se realizaron los siguientes pasos:

1. **Análisis y comprensión del esquema de las facturas XML.** A pesar de que muchos campos de la factura eran obligatorios, por ejemplo el Nif del emisor, otros muchos no solo eran opcionales sino que podían ser excluyentes o dependientes entre si. Es por ello, que de cara a realizar una correcta generación de los datos se tuvo que analizar cada una de las variaciones del esquema.

2. **Implementación de un programa de generación de datos.**[1] Una vez comprendido el esquema de la factura, se generó un programa que generaba los datos en función de una serie de parámetros previamente especificados. De esta forma, se podrían generar facturas con características diferentes.

3. **Implementación de un conversor a XML.** Los datos generados anteriormente aún no están en formato XML, formato final de la factura, por lo que se implementó un programa capaz de transformar los datos generados a un XML que siguiese el esquema proporcionado por el departamento de Hacienda de la Diputación Foral.[2]

[1] https://github.com/gorokotkd/tfg_nodejs/blob/main/functions/data_generator.js
[2] https://github.com/gorokotkd/tfg_nodejs/blob/main/functions/transformer.js

4. **Firma de la factura.** Una vez el xml con los datos es generado se debe realizar el proceso de firma de la factura. Para ello, utilizando la librería *xml-crypto*[3], y utilizando un par de claves público-privadas, se realizó el proceso de firma de la factura generada.

## 7.2. Descripción y ejecución de las Pruebas a realizar

Tal como se comentaba anteriormente, las pruebas definidas a continuación se han diseñado con el objetivo de evaluar el rendimiento de las bases de datos diseñadas y de evaluar posibles modificaciones del diseño que mejoren el rendimiento.

En términos generales, se han distinguido dos tipos de pruebas. Un primer grupo, encargado de probar el rendimiento en la obtención de facturas individuales, y datos de las mismas. Y un segundo grupo, encargado de probar la obtención de una factura, o un campo de la misma, de un conjunto de facturas agrupadas. Dentro de cada uno de estos grupos se han realizado dos tipos de pruebas principales, una con facturas "pequeñas", es decir, facturas con un reducido número de datos y otra facturas "grandes", con la mayor cantidad de datos posibles, siendo su tamaño máximo aproximado de 500KB.

Cabe destacar que todas las pruebas se han realizado sobre la misma máquina y con aproximadamente 30GB de datos ya insertados en las bases de datos, por lo que se han podido ver las diferencias y similitudes de rendimiento entre ambas Bases de Datos. En cuanto a los gráficos mostrados, la unidad de todos los tiempos que se muestran es *milisegundos*, en caso contrario, se indicará en la leyenda del gráfico.

### 7.2.1. Pruebas para seleccionar el algoritmo de compresión

Dado que una de las principales características del sistema es el almacenamiento de los datos comprimidos, el primer paso antes de realizar pruebas de inserción y consultas es elegir el algoritmo de compresión. En el capitulo 4, Descripción de las técnicas de compresión, se había definido la teoría detrás de cada una de las técnicas de compresión a probar en el proyecto.

A continuación, se muestran las pruebas a las que se han sometido los algoritmos y que han servido para decantarse por uno de ellos.

[3] https://www.npmjs.com/package/xml-crypto

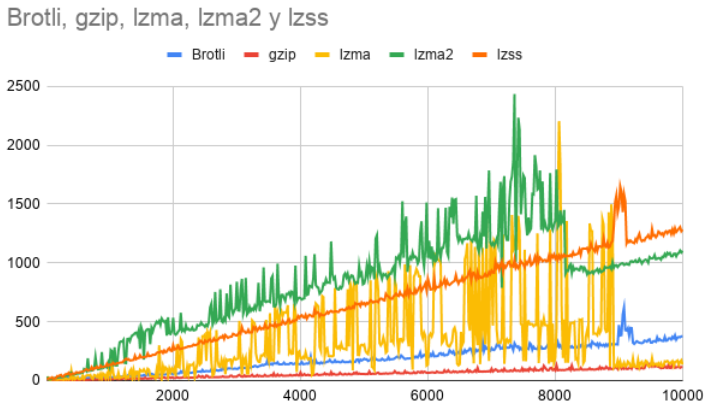



**Figura 7.1:** Comparación del tiempo de compresión entre algoritmos

La figura 7.1, contiene el gráfico resultante de comprimir agrupaciones de hasta 10000 facturas. El eje horizontal representa el número de facturas agrupadas, mientras que el eje vertical, representa el tiempo empleado, en milisegundos, por cada uno de los algoritmos a comprimir las agrupaciones de facturas. Tal como se puede observar en el gráfico, la técnica que ofrece mejores resultados es GZip, la cual mantiene un tiempo inferior a los 250 ms durante todos los casos

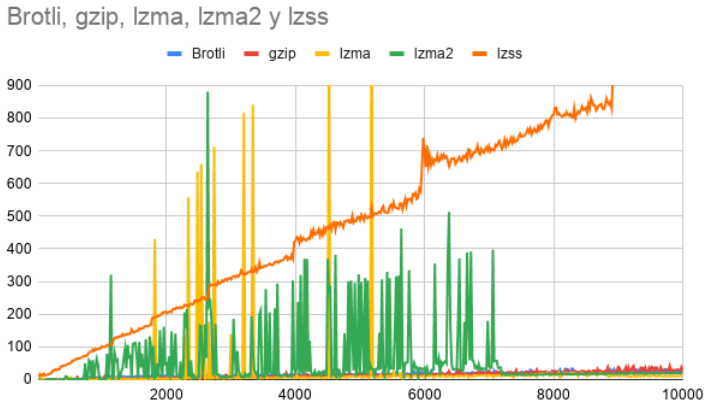



**Figura 7.2:** Comparación del tiempo de descompresión entre algoritmos

La figura 7.2, muestra el gráfico resultante de descomprimir la agrupación de 10000 facturas anterior. El eje horizontal, muestra de nuevo el número de facturas agrupadas en cada momento, mientras que el eje vertical muestra el tiempo, en milisegundos, de descomprimir cada una de las agrupaciones. Como se ve en el gráfico, las técnicas Brotli y GZip tienen un tiempo de descompresión casi constante, el cual se mantiene inferior a los 50 milisegundos.

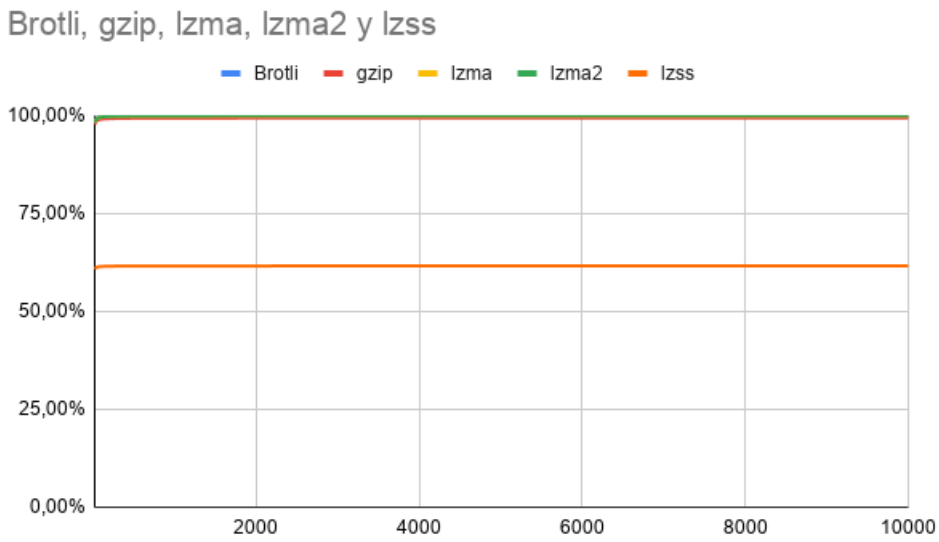



**Figura 7.3:** Ratio de compresión de cada una de las técnicas de compresión abordadas

Por último, la figura 7.3, muestra el ratio de compresión de cada una de las técnicas tratadas. El eje horizontal representa el número de facturas agrupadas, mientras que el eje vertical representa el ratio de compresión de las técnicas, en tanto por ciento. Tal como se ve, todas las técnicas, a excepción de Lzss, ofrecen un ratio de compresión cercano al 100 %, por lo que en ese aspecto todas serían una buena opción.

Teniendo en cuenta los datos anteriores y dadas las necesidades de la aplicación, se ha decidido utilizar el algoritmo Gzip. Esto se debe a que Gzip es la técnica que ofrece una mejor compresión y descompresión de los datos. Así mismo, tiene un ratio de compresión muy alto, por lo que reducirá considerablemente el tamaño de las facturas originales.

### 7.2.2. Pruebas sobre facturas individuales

En general, las pruebas a las que se han sometido los diseños de las BDs, tanto en MongoDB como en Cassandra, están estrechamente relacionadas con la obtención de los datos de las bases de datos, ya que es el aspecto más crítico. No obstante, dado que algunos de los cambios en los diseños están relacionados con los tiempos de inserción de los datos se ha decidido documentar esta parte junto con los problemas surgidos.

Tiempo de inserción de facturas

Los gráficos que se muestran a continuación representan los tiempos en realizar los pasos de obtención de datos a guardar en RAW, compresión de factura e inserción en la base de datos correspondiente.

### Pruebas sobre MongoDB

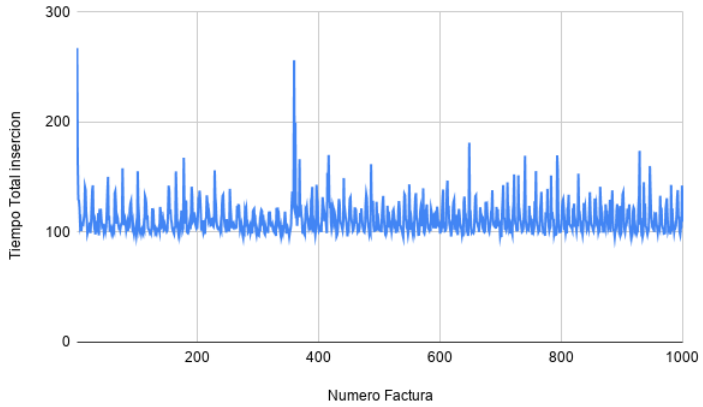



**Figura 7.4:** Tiempo total de inserción de facturas individuales con pocas lineas de detalle

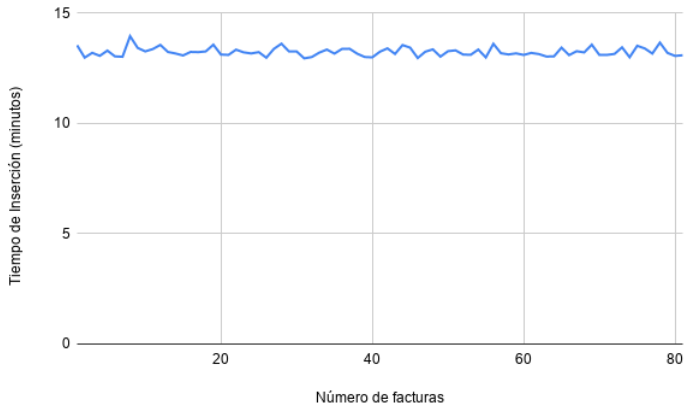



**Figura 7.5:** Tiempo total de inserción de facturas individuales con muchas lineas de detalle

Las figuras 7.4 y 7.5 muestran la diferencia del tiempo de inserción entre facturas "pequeñas" y "grandes", respectivamente. Tal como se ve en las figuras la diferencia de tiempo entre ambas es considerable. Para realizar una inserción de una factura con poco contenido, con una tamaño medio de 10KB, el tiempo de inserción esta entre 100 y 200 milisegundos,

mientras que para una factura con mucho contenido, un tamaño medio de 500KB, el tiempo de inserción se encuentra sobre los 13 minutos. Este enorme tiempo de espera entre factura y factura es el motivo de que la muestra de facturas grandes no llegue a las 100 facturas, no obstante, es de un tamaño suficiente como para ser una muestra representativa.

Dado que esta situación es insostenible, ya que para un sistema con un gran número de peticiones diarias no es permisible una espera de 13 minutos entre petición y petición. Tras realizar las pruebas necesarias para averiguar el punto en el que se pierde tanto tiempo el resultado ha sido el siguiente.

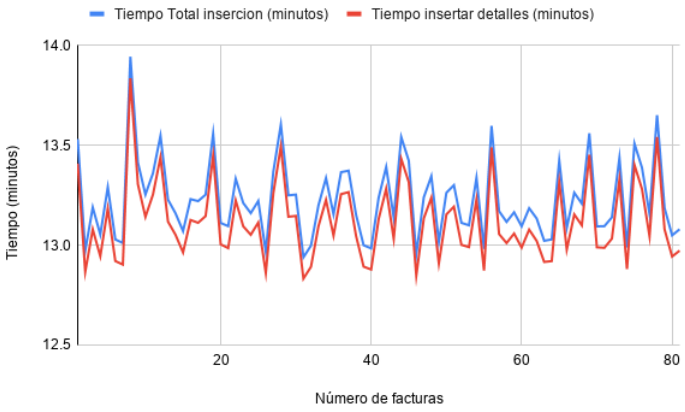



**Figura 7.6:** Tiempo de inserción total frente al tiempo de inserción de los detalles

Tal como se puede observar en la figura 7.6, a la hora de realizar la inserción de facturas de un gran tamaño, la gran mayoría del tiempo se emplea en la lectura y posterior inserción de los detalles de la factura en la base de datos. Para ver la corrección sobre el diseño correspondiente junto con los nuevos resultados obtenidos tras la corrección ver la sección 7.3.1, Corrección M.1.

#### Pruebas sobre Cassandra

Anteriormente se ha visto el problema surgido en MongoDB con el almacenamiento de los datos en RAW. No obstante, en Cassandra, desde el inicio no se planteó el almacenamiento de los detalles de la factura en RAW. Es por ello que en este SGBD no se ha tenido ningún problema relacionado con la inserción de las facturas, siendo los resultados de inserción y compresión los siguientes.

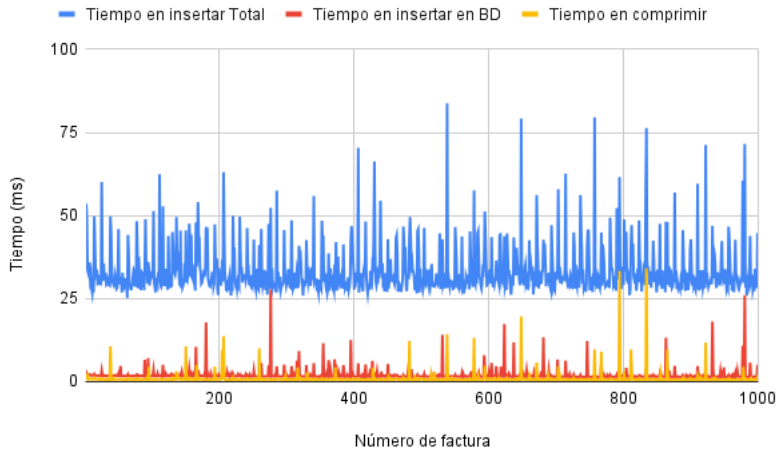



**Figura 7.7:** Tiempos de inserción y compresión de factura pequeña en Cassandra

Los resultados de este conjunto de facturas son bastante similares a los obtenidos en MongoDB, ver figura 7.31, no obstante, el tiempo de inserción total en Cassandra es ligeramente menor. A continuación se muestra un gráfico con la media de tiempos, para las facturas pequeñas, de realización de cada uno de los pasos de la inserción junto con el tiempo total.

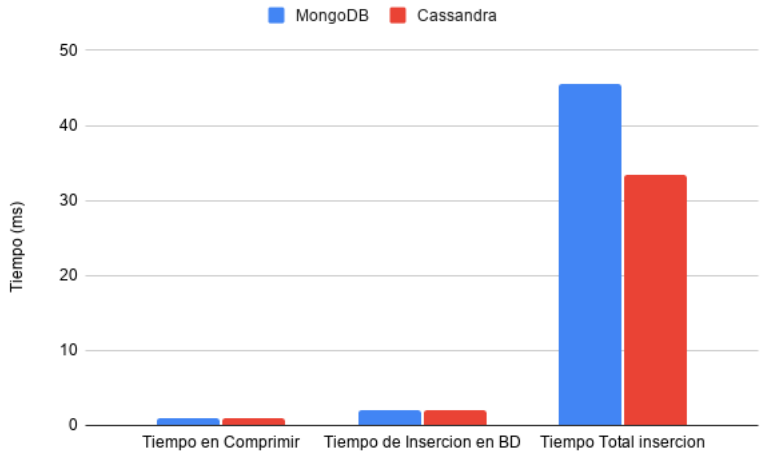



**Figura 7.8:** Comparación de fases de inserción entre Mongo y Cassandra con facturas pequeñas

En cuanto a las facturas de gran tamaño el tiempo total de inserción de estas es el siguiente.

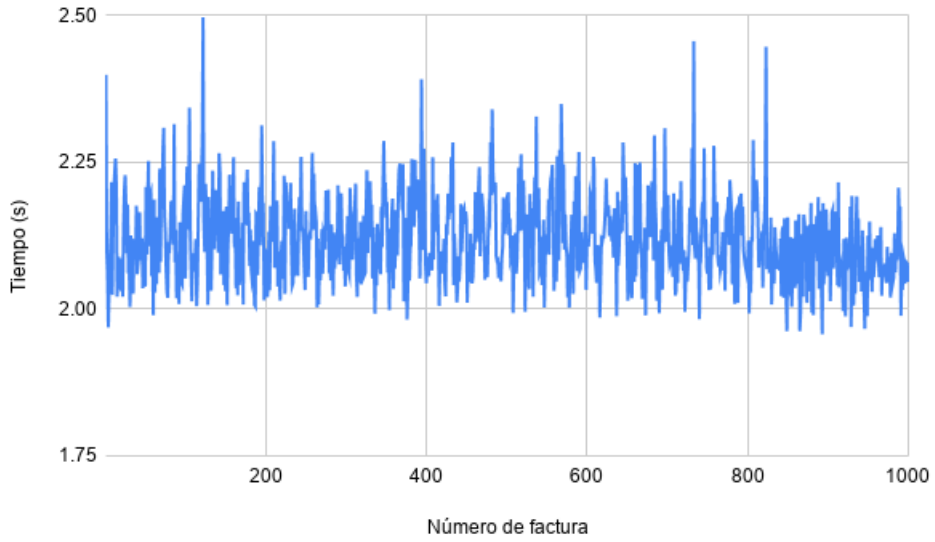



**Figura 7.9:** Tiempo de inserción total de facturas grandes en Cassandra

Si se comparan estos tiempos con los obtenidos en MongoDB tras la corrección, ver figura 7.32, la mejora es muy pequeña pero existente. El tiempo total de inserción de cada uno de los datos ha sido ligeramente más rápido, apenas le supera por 250 milisegundos.

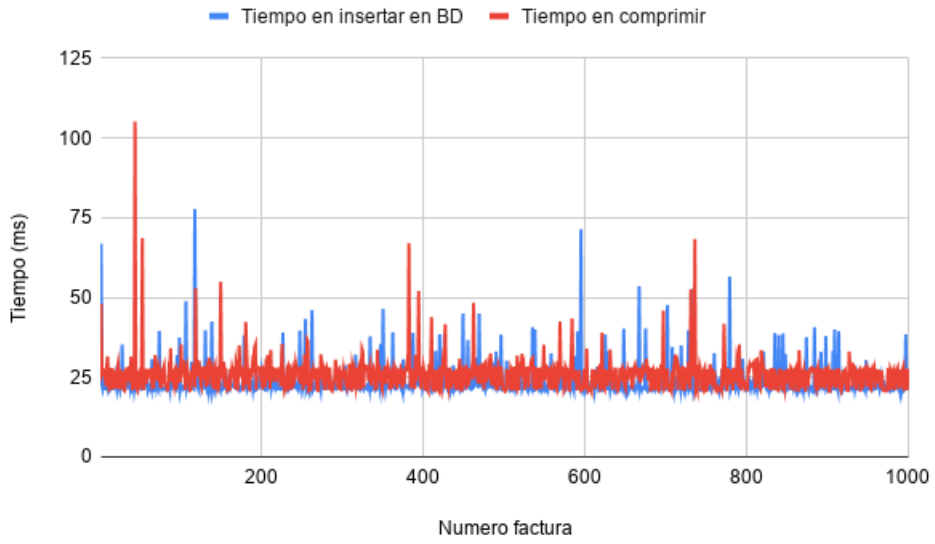



**Figura 7.10:** Tiempo de inserción en BD y compresión de facturas grandes en Cassandra

En cuanto al tiempo de inserción en la BD los tiempos en Cassandra son ligeramente peores que en MongoDB, unos 10 milisegundos de diferencia media. Aunque en principio

estos resultados no encajan con la diferencia de tiempo total mostrada antes, en la que Cassandra tenía un tiempo de insertar total inferior a MongoDB, el resultado es de esperar, ya que en MongoDB se almacenan algunos datos más que en Cassandra, como puede ser la descripción de la factura.

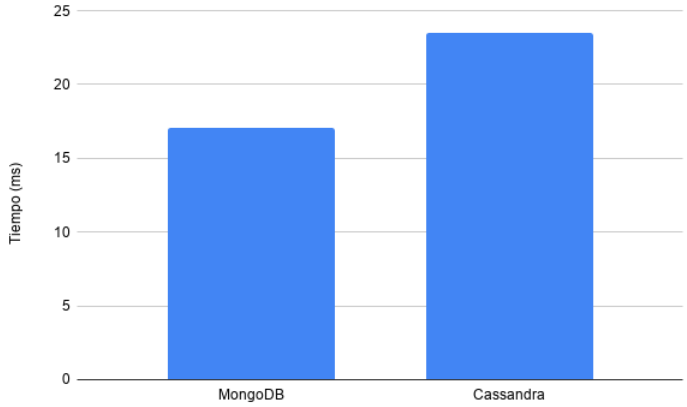



**Figura 7.11:** Tiempo de inserción de facturas grandes en ambos SGBDs

### Tiempo en recuperar una factura

En este grupo de pruebas se ha identificado el caso relacionado con la obtención de una factura mediante el QR, es decir, se busca una factura mediante el identificador TBAI, y se devuelve la factura completa.

#### Pruebas sobre MongoDB

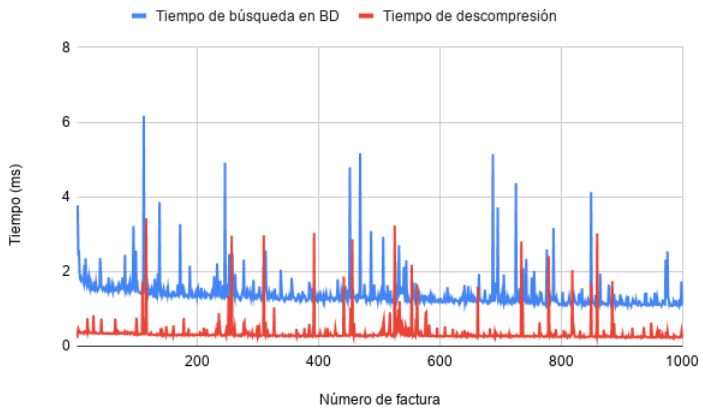



**Figura 7.12:** Tiempo de búsqueda de factura pequeña en MongoDB

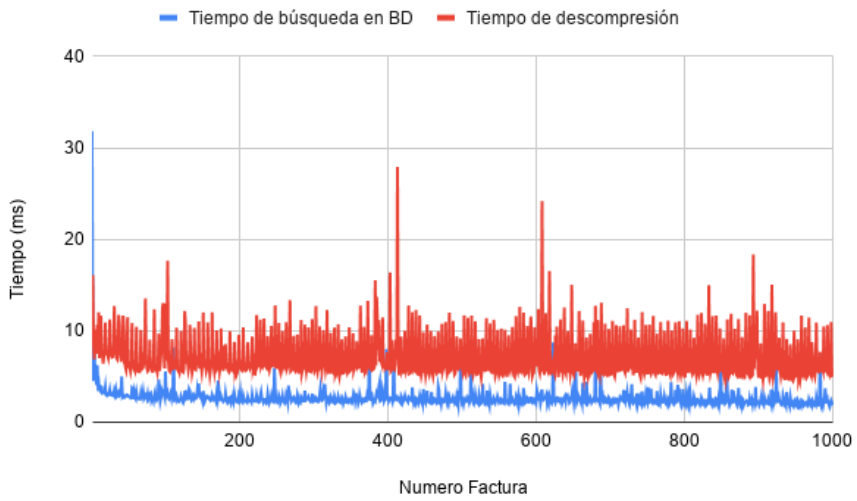



**Figura 7.13:** Tiempo de búsqueda de factura grande en MongoDB

Las dos gráficas anteriores, figuras 7.12 y 7.13, muestran el tiempo total de búsqueda de una factura pequeña y grande, respectivamente. El tiempo se divide entre tiempo de realización de la consulta y devolución del dato, y el tiempo de descompresión de la factura. Los tiempos en ambos casos son muy positivos, en las facturas pequeñas, el tiempo en realizar todos los pasos no supera los 5 milisegundos, mientras que para las facturas grandes tarda aproximadamente 20 milisegundos.

### Pruebas sobre Cassandra

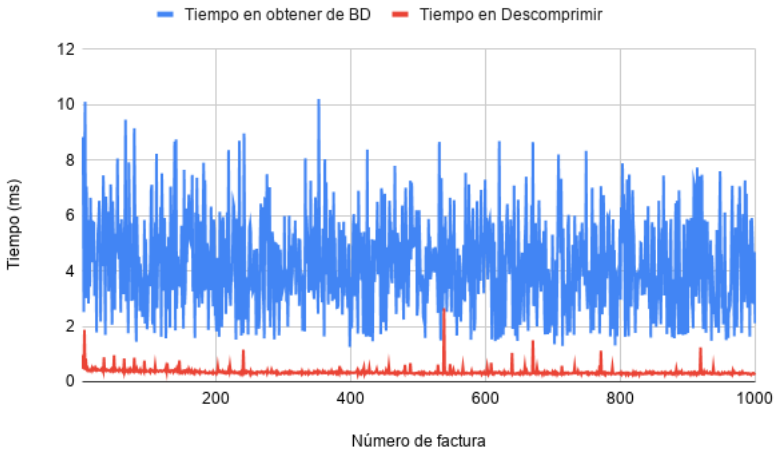



**Figura 7.14:** Tiempo de obtención y descompresión de facturas pequeñas en Cassandra

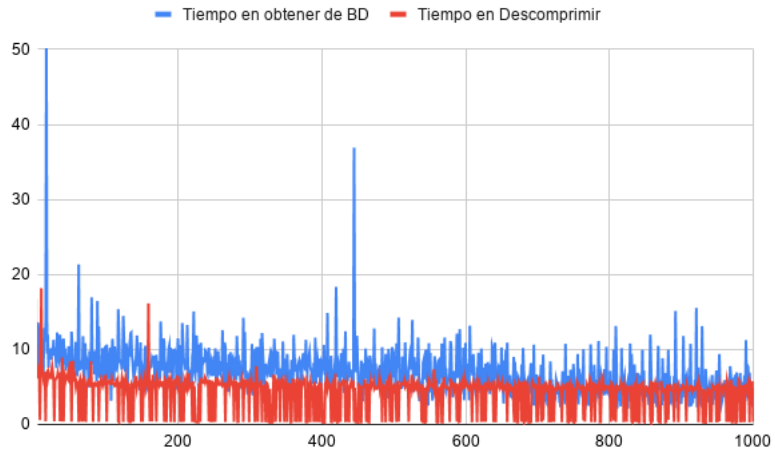



**Figura 7.15:** Tiempo de obtención y descompresión de facturas grandes en Cassandra

En las dos figuras anteriores se pueden ver los tiempos de obtención de facturas de la Base de Datos, es decir, el tiempo que tarda el SGBD en recuperar la factura solicitada, junto con el tiempo que tarda el sistema en descomprimirla.

El tiempo de búsqueda en la BD en Cassandra, para las facturas pequeñas, es algo más lento que en MongoDB, apenas tarda 2 milisegundos más. En cuanto al tiempo de descompresión ambos tiempos son muy similares, como es de esperar. En cuanto a las facturas grandes MongoDB es más rápido buscando que Cassandra. A pesar de que el tiempo medio de búsqueda de Cassandra no supera los 8 milisegundos MongoDB es capaz de realizar la búsqueda en menos de la mitad de tiempo. No obstante, no es una diferencia lo suficientemente significativa como para descartar el uso de uno de los sistemas.

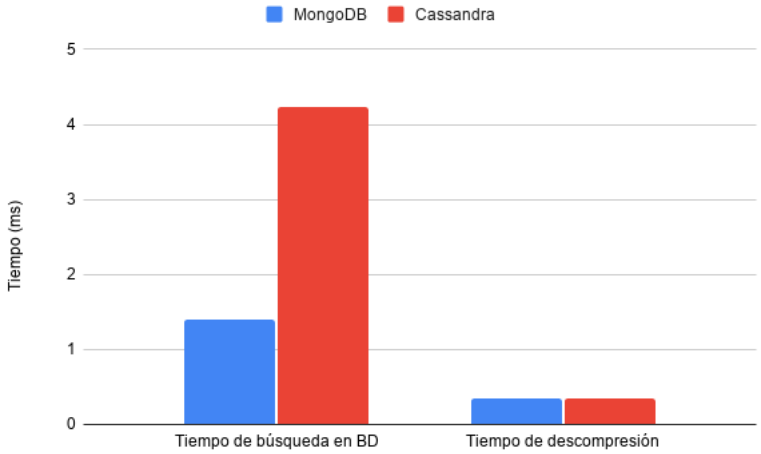



**Figura 7.16:** Tiempo medio de obtención de facturas pequeñas

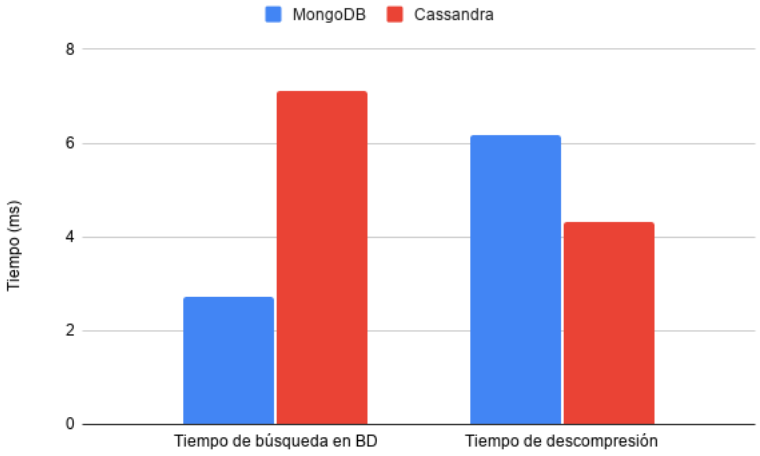



**Figura 7.17:** Tiempo medio de obtención de facturas grandes

Tiempo en recuperar un dato de una factura

El reducido tiempo de compresión y descompresión obtenido en las pruebas anteriores sugirió realizar la comparación entre la obtención de datos directamente de la base de datos, en RAW, frente a obtenerlos de la factura comprimida. Aunque el tiempo de descompresión sea muy reducido existe la posibilidad de perder una gran cantidad de tiempo buscando el dato pedido dentro de la propia factura.

### Pruebas sobre MongoDB

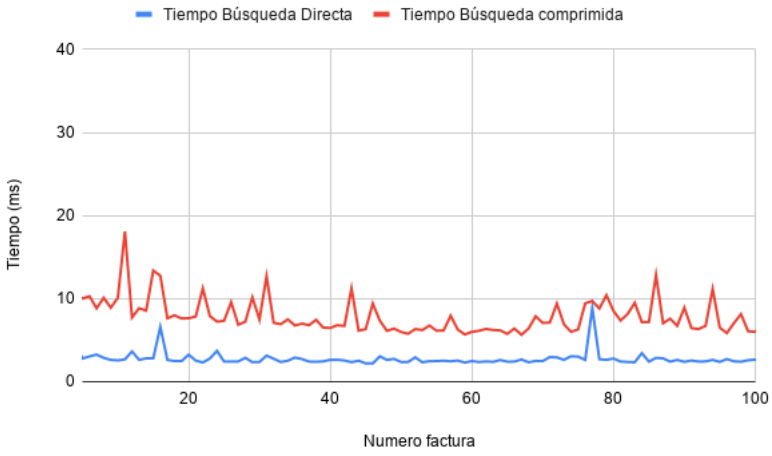



**Figura 7.18:** Tiempo de obtención de un dato concreto en facturas pequeñas

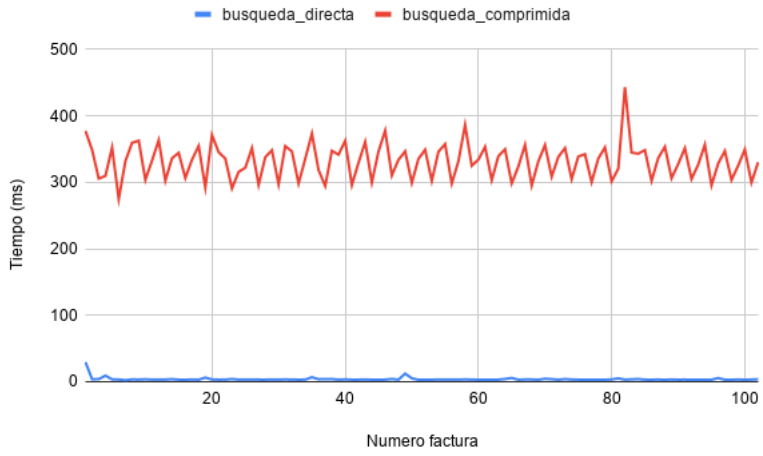



**Figura 7.19:** Tiempo de obtención de un dato concreto en facturas grandes

Los gráficos de las figuras 7.18 y 7.19 muestran la comparativa de tiempo de obtención de un dato, en este caso el campo *Importe Total Factura*, directamente de la base de datos, en *RAW*, frente a obtenerlo de la factura comprimida. En el primer caso simplemente se accede al documento correspondiente de la BD y se devuelve el dato. En el segundo caso, se accede al documento en cuestión y se devuelve la factura comprimida, la cual es descomprimida y accedida mediante XPath[4] para obtener el dato.

En la primera gráfica, figura 7.18, se muestra la comparativa para facturas pequeñas. Tal como se puede ver los tiempo son muy similares, apenas hay 10 milisegundos de diferencia entre ambos métodos de obtención del dato. No obstante, al analizar la gráfica de la figura 7.19 el resultado es diferente. El tiempo en obtener el dato directamente de la base de datos es similar al caso anterior, apenas supera los 10 milisegundos, sin embargo, al buscar el dato en la factura comprimida los tiempos se disparan a los 300 milisegundos.

Con estos datos, se puede concluir que es muy útil tener algunos datos en *RAW*, es decir, accesibles directamente en cada documento de la colección, para consultas que se ejecuten con gran frecuencia, este el caso de las consultas del QR. Los datos correspondientes a esta consulta estarían accesibles sin tener que realizar operaciones extra, y con un tiempo de respuesta muy bajo.

#### Pruebas sobre Cassandra

[4]Lenguaje que permite construir expresiones que recorren y procesan un documento XML

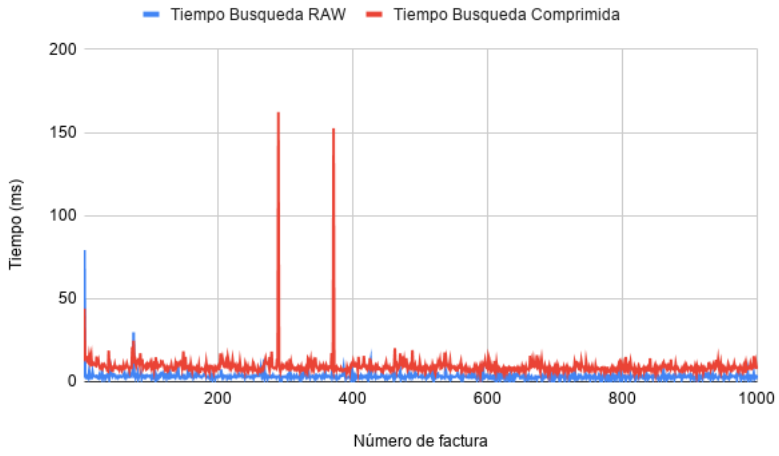



**Figura 7.20:** Tiempo de búsqueda en RAW frente a búsqueda en comprimido en Cassandra con facturas pequeñas

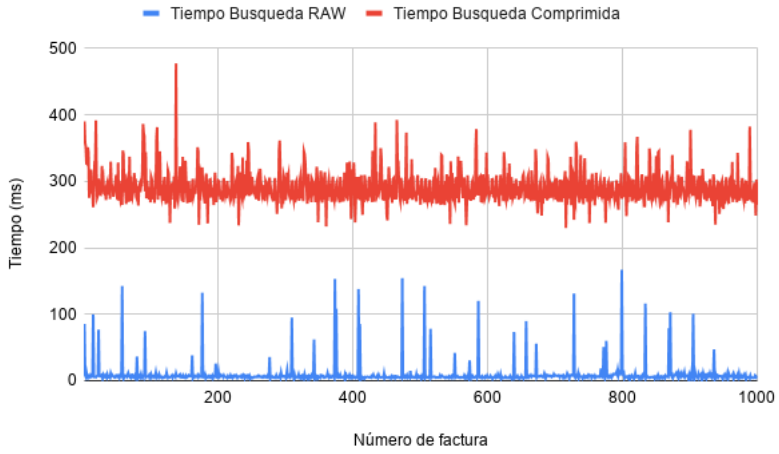



**Figura 7.21:** Tiempo de búsqueda en RAW frente a búsqueda en comprimido en Cassandra con facturas grandes

Las dos figuras anteriores muestran el resultado de comparar el tiempo de búsqueda de un dato en RAW frente a obtenerlo de la factura comprimida en Cassandra.

En términos generales, los resultados son similares a los obtenidos al realizar las pruebas en MongoDB. La diferencia entre obtener el dato en RAW y en comprimido para las facturas pequeñas es de unos 5 milisegundos de diferencia. En cuanto a las facturas grandes la diferencia es de unos 300 milisegundos, la misma que la ya vista en MongoDB.

En cuanto a la comparativa entre ambos SGBDs, en general, MongoDB ofrece unos mejores resultados, superando a Cassandra por unos pocos milisegundos de diferencia. No

obstante, cuando se trata de devolver gran volumen de datos Cassandra, ver figura 7.24, obtiene un mejor rendimiento.

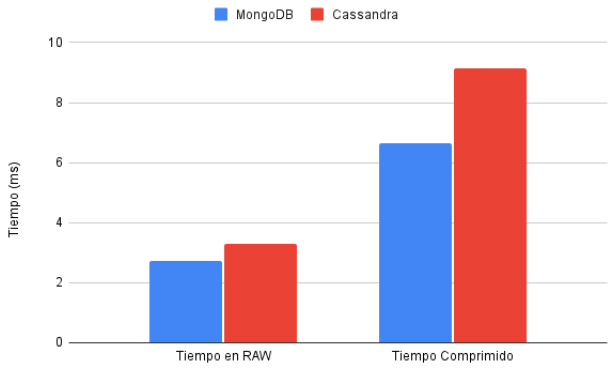



**Figura 7.22:** Comparativa obtención de dato concreto MongoDB y Cassandra con facturas pequeñas

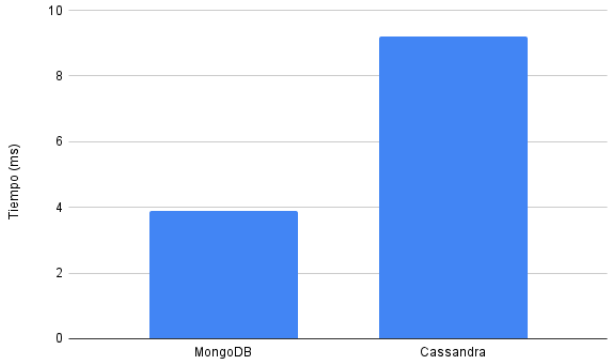



**Figura 7.23:** Comparativa obtención de dato concreto en RAW en MongoDB y Cassandra con facturas grandes

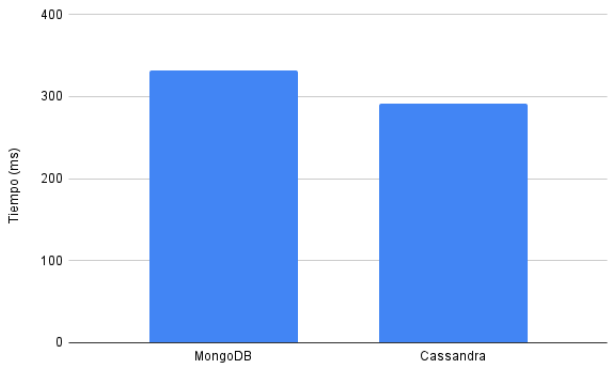



**Figura 7.24:** Comparativa obtención de dato concreto en comprimido en MongoDB y Cassandra con facturas grandes

### 7.2.3. Pruebas sobre facturas agrupadas

El objetivo de este conjunto de pruebas ha sido evaluar el rendimiento de las tablas o colecciones que contienen el conjunto de facturas agrupadas por NIF del emisor y rango de fechas de emisión. La meta de las pruebas ha sido también obtener el número óptimo de facturas agrupadas de cara a obtener un buen rendimiento y tiempos de respuesta.

Antes de comenzar, y tras realizar un nuevo análisis del diseño inicial de la colección de MongoDB enfocada a gestionar las facturas agrupadas, se ha decidido llevar a cabo la Corrección M.2, ver sección 7.3.2.

#### Tiempo en recuperar una factura mediante el identificador TBAI

La búsqueda por identificador se realiza de la siguiente forma. Partiendo del identificador TBAI[5], se extraen el NIF y la fecha de expedición de la factura a la que corresponde el identificador. A continuación, se realiza la consulta, la cual se podría definir como "*Obtener las agrupaciones cuyo NIF se corresponde con el del identificador y cuya fecha de expedición de la factura se encuentra entre las fechas de la agrupación*" en cada uno de los SGBD.

Cabe destacar que la factura a recuperar en todas las pruebas siempre será la última de la agrupación, es decir, para una agrupación de 1000 facturas, la factura con la que se ha realizado la prueba de recuperación es la número 1000. De esta manera, los tiempos que se obtienen corresponderán siempre con el peor de los casos.

##### Pruebas sobre MongoDB

[5]TBAI-NNNNNNNNN-DDMMAA-FFFFFFFFFFFFF-CRC

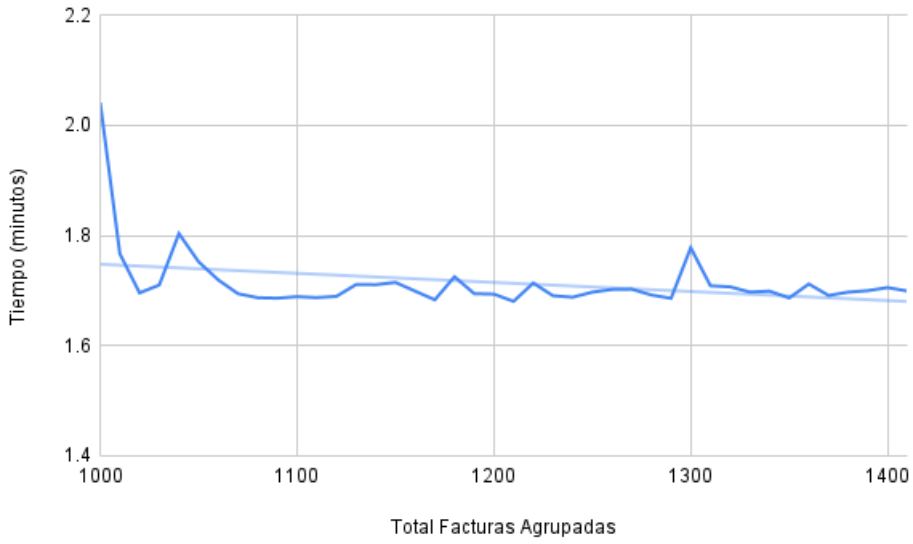



**Figura 7.25:** Tiempo de búsqueda en BD de la agrupación de facturas pequeñas

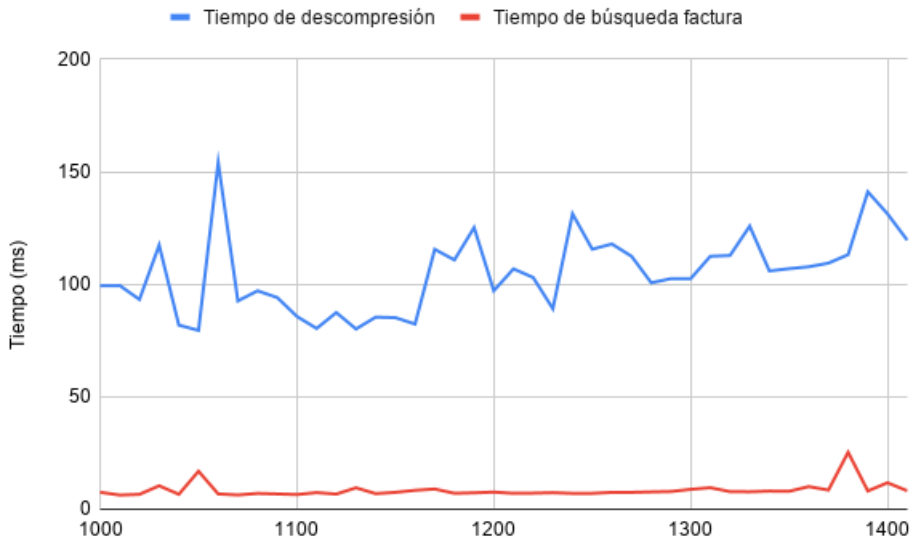



**Figura 7.26:** Tiempo de descompresión y búsqueda en la agrupación de facturas pequeñas

Tal como se aprecia en la figura 7.25, el tiempo de búsqueda en la base de datos no es muy óptimo. El tiempo medio de búsqueda de una agrupación es ligeramente inferior a los 2 minutos. Esto se debe a que en la búsqueda, ver listado B.6, se están comparando datos que no tienen índices aplicados sobre ellos.

Así mismo, si se comprueban los tiempos de búsqueda y recuperación de las agrupaciones de facturas grandes, el resultado que se obtiene es similar al de las facturas pequeñas.


**Figura 7.27:** Tiempo de búsqueda en BD de agrupaciones de facturas grandes

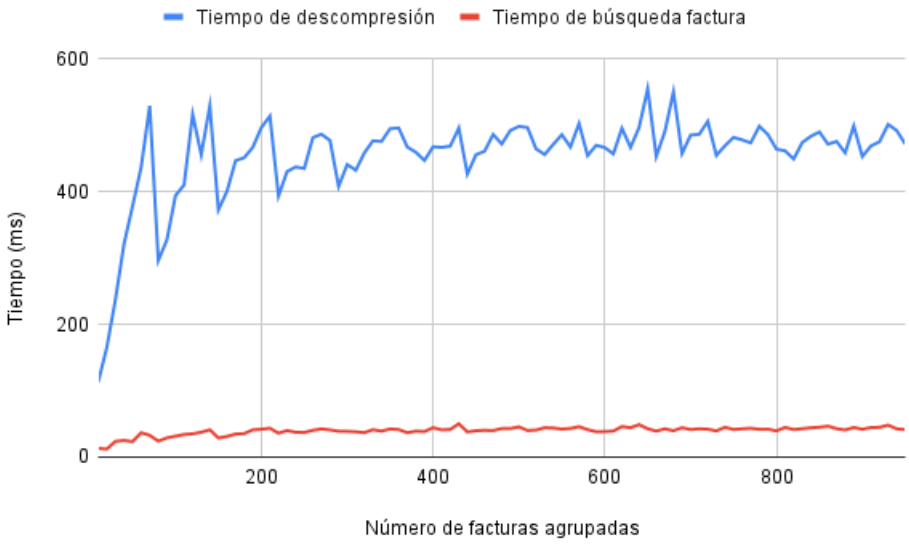



**Figura 7.28:** Tiempo de descompresión y búsqueda en la agrupación de facturas grandes

Tal como se adelantaba anteriormente, el motivo por el cual los tiempos de obtención de las agrupaciones es tan lento, es por la falta de índices. Por tanto, para solucionar el problema se ha llevado a cabo la Corrección M.3, ver sección 7.3.3.

#### Pruebas sobre Cassandra

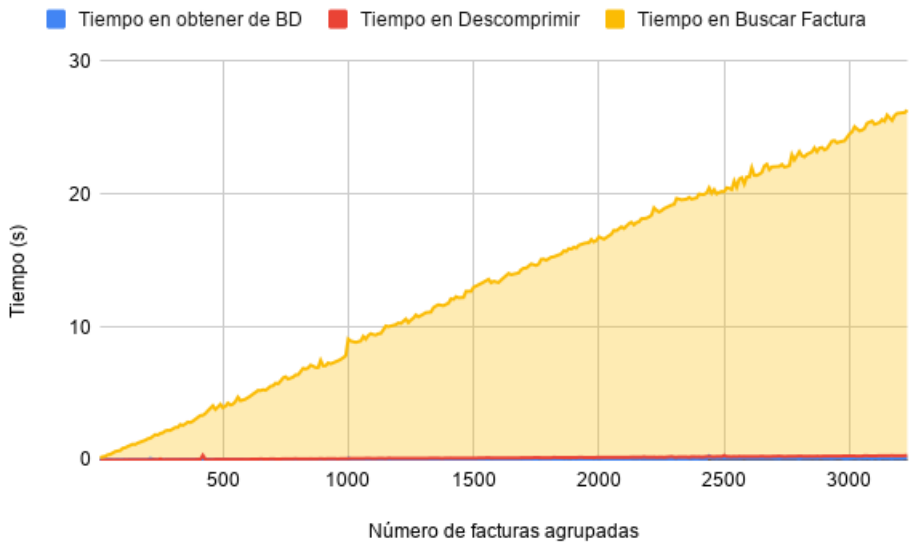



**Figura 7.29:** Tiempo de búsqueda de factura en agrupaciones pequeñas en Casssandra

Tal como se muestra en la figura anterior el tiempo de búsqueda es muy lento, para agrupaciones de 2000 facturas pequeñas el tiempo de búsqueda total es algo inferior a los 20 segundos.

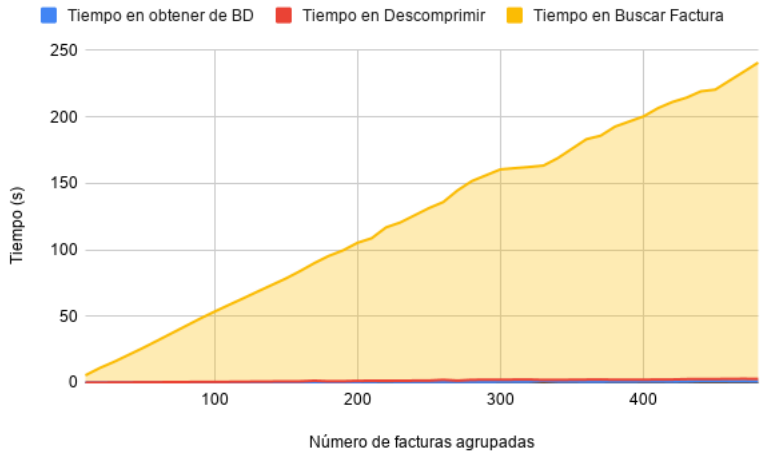



**Figura 7.30:** Tiempo de búsqueda de factura en agrupaciones grandes en Casssandra

En cuanto a las agrupaciones de facturas grandes los resultados son también muy lentos. Para agrupaciones de 200 facturas el tiempo de búsqueda dentro de la agrupación es de unos 100 segundos $\simeq$ 1,5 minutos.

El motivo de estos tiempos de respuesta es la forma de búsqueda de la factura concreta

dentro de la agrupación. Una vez es descomprimida la agrupación correspondiente, se obtienen todas las facturas y se calcula el identificador TBAI de cada una de ellas hasta encontrar el que coincida con el solicitado. Dado que este método es muy ineficiente se ha decidido aplicar la Corrección C.1, ver sección 7.3.4.

## 7.3. Correcciones en los diseños

Anteriormente se han especificado las correcciones llevadas a cabo en los diseños debido a diferentes requerimientos establecidos por la Diputación Foral. En esta sección, se detallan las correcciones realizadas en los diseños debido a errores o malos resultados obtenidos en las pruebas y que tienen como objetivo obtener el máximo rendimiento del sistema.

### 7.3.1. Corrección M.1

Anteriormente se ha podido comprobar que guardar campos de la factura como los detalles en *RAW* en la base de datos puede no llegar a ser óptimo, en función de la cantidad de detalles que tenga la factura. No obstante, con el objetivo de aligerar los tiempos de inserción y recuperación de facturas, se ha eliminado el campo *DetallesFacturas* junto con su contenido, del esquema mostrado en el listado B.1, de esta manera, este campo solo será accesible a través de la factura completa comprimida. Tras realizar el cambio, se ha procedido a realizar unas nuevas pruebas para comprobar si efectivamente ha mejorado el rendimiento de la base de datos, siendo los resultados los que se muestran a continuación.

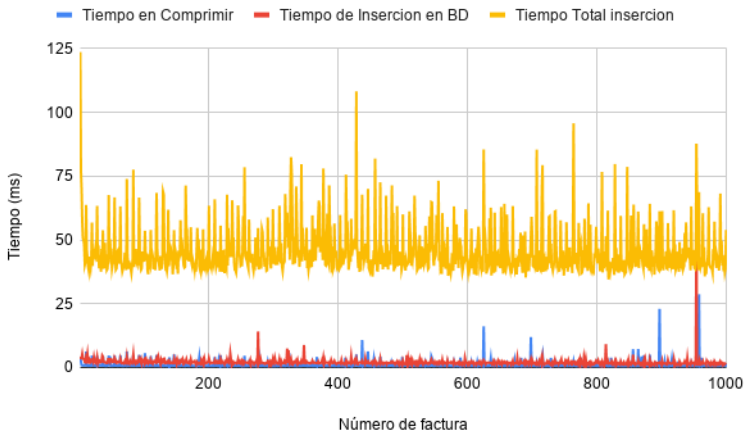



**Figura 7.31:** Tiempo total de inserción de facturas individuales pequeñas sin guardar los detalles en RAW

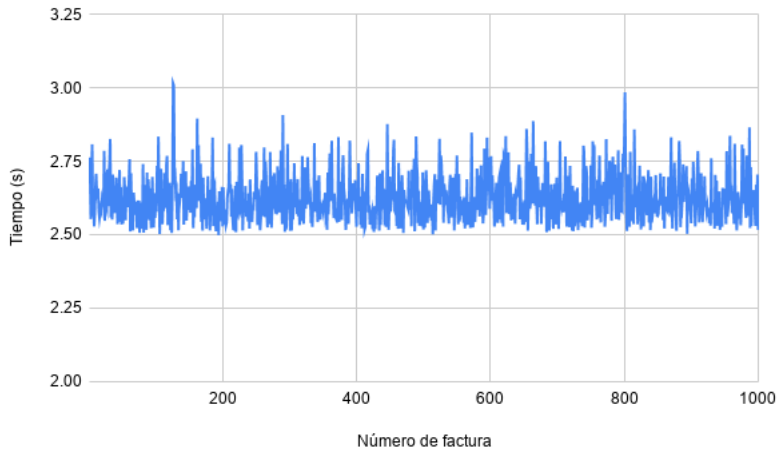



**Figura 7.32:** Tiempo total de creación e inserción de facturas individuales grandes sin guardar los detalles en RAW

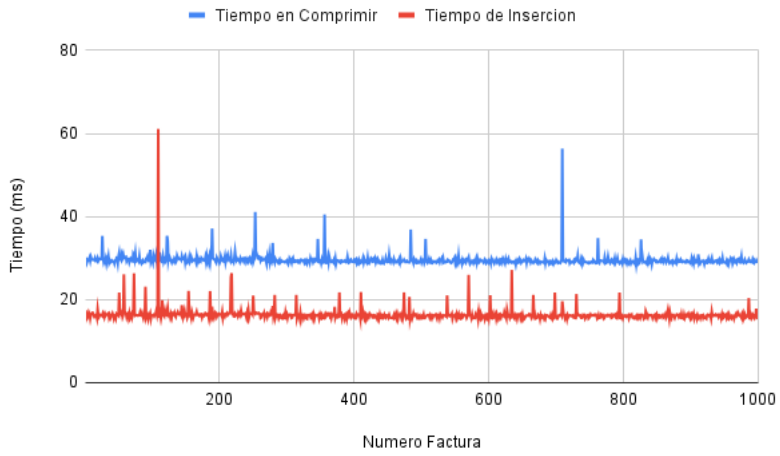



**Figura 7.33:** Tiempo de compresión en inserción en BD de las facturas grandes sin guardar los detalles en RAW

Si se comparan las dos figuras anteriores junto con las obtenidas con el diseño inicial, ver figuras 7.4 y 7.5, la mejora en el rendimiento ha sido notable. El tiempo de inserción de las facturas pequeñas se ha reducido a la mitad, unos 50 milisegundos. No obstante, en el caso de las facturas grandes, el tiempo total de inserción de facturas se ha reducido de 13 minutos a apenas 2 segundos, lo cual permite realizar una generación e inserción mucho más rápida de los datos.

Así mismo, en las figuras anteriores también se puede apreciar el tiempo de compresión e inserción de facturas en la base de datos. Salvo algunos picos en momentos concretos

los tiempos resultantes son muy positivos, para las facturas pequeñas los tiempo de compresión e inserción en BD medios no superan los 10 milisegundos. En cuanto a las facturas grandes, los tiempos de compresión e inserción y compresión son algo más lentos, unos 20 milisegundos en insertar las facturas en la BD y 30 milisegundos en comprimirlas, unos resultados lógicos, ya que son facturas con mucho más contenido.

### 7.3.2. Corrección M.2

Inicialmente, el esquema de la colección que contiene el conjunto de facturas agrupadas en MongoDB contenía la forma mostrada en el listado B.2, en el que el campo *_id* era de la forma "NIF/DD-MM-AAAA/DD-MM-AAAA", haciendo este campo único. No obstante, al realizar la inserción de datos de prueba se obtuvieron dos problemas. Por una parte, al estar las fechas contenidas en un campo de texto, la búsqueda de las agrupaciones por fecha resultaba muy compleja de realizar.

El segundo problema estaba más relacionado con la propia configuración de MongoDB. Como ya se ha comentado anteriormente, MongoDB almacena los datos en documentos, que a su vez se almacenan en colecciones. No obstante, el tamaño de estos documentos tiene un límite, 16 MB [MongoDB, 2021b], por lo que es imposible insertar un documento que supere este tamaño.

Para evitar este problema y no tener complicaciones a la hora de insertar documentos, la solución empleada ha sido dividir las agrupaciones que superen este tamaño. Es decir, en caso de que una agrupación de facturas supere los 16MB, supongamos que ocupa 53 MB, se dividirá su contenido en 4 documentos diferentes ($\left\lceil \frac{53}{16} \right\rceil = 4$), pero todos representando el mismo NIF emisor y rango de fechas. Es por ello, que el campo *_id* dejaría de ser único, agregando de esta manera otro motivo más para modificar el esquema.

Por tanto, el nuevo esquema para las facturas agrupadas en MongoDB quedaría de la siguiente forma. Cabe destacar que el campo *_id* no se especifica, por lo que MongoDB creará uno al momento de la inserción, garantizando de esta manera la unicidad de este campo.

```
{
    nif: {
        type: Schema.Types.String,
        required: true
    },
    fechaInicio: {
        type: Schema.Types.Date,
```

```
        required: true
    },
    fechaFin: {
        type: Schema.Types.Date,
        required: true
    },
    idents: {
        type: Schema.Types.Array,
        required: true
    },
    agrupacion: {
        type: Schema.Types.String,
        required: true
    }
}
```

**Listado 7.1:** Modificación del esquema de las facturas en MongoDB

### 7.3.3. Corrección M.3

Tras analizar los lentos resultados obtenidos al buscar las agrupaciones de facturas se ha decidido crear un índice sobre los campos sobre los que se realiza la búsqueda de este conjunto de datos. Este índice se ha creado sobre los campos *nif* y *fechaInicio*, campos principales de búsqueda de las agrupaciones, ver listado B.3. Tras volver a repetir las consultas ejecutadas en 7.2.3, la mejora obtenida es muy buena.

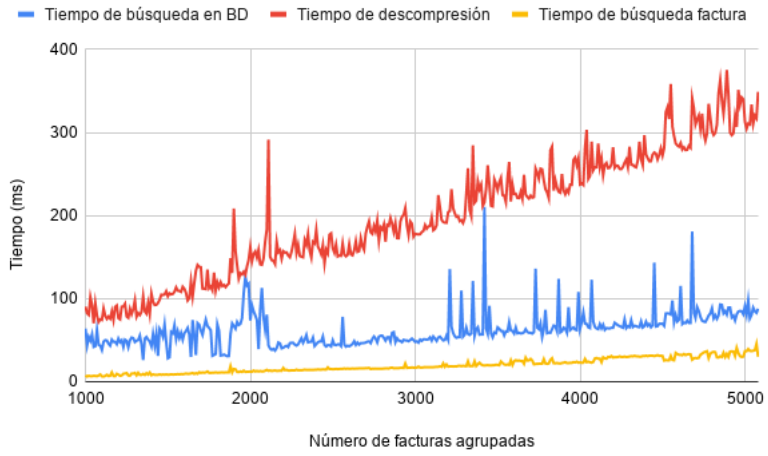



**Figura 7.34:** Desglose de tiempo de búsqueda de factura agrupada pequeña

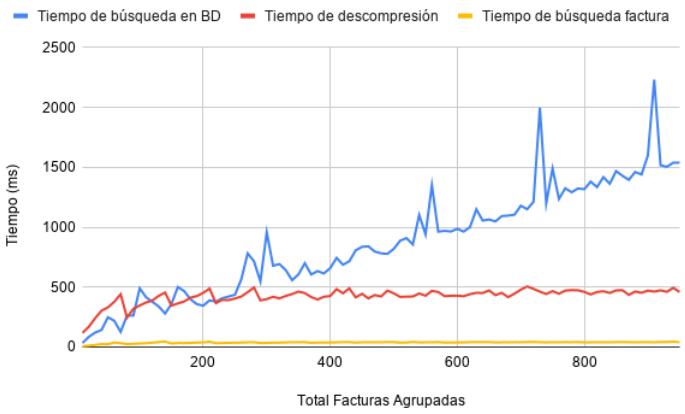



**Figura 7.35:** Desglose de tiempo de búsqueda de factura agrupada grande

Tal como se ve en las figuras anteriores, y como se comentaba antes, la mejora en el rendimiento gracias a los índices es notoria. Para agrupaciones de facturas pequeñas el tiempo de búsqueda de la agrupación correspondiente se ha reducido de los 2 minutos de la figura 7.25 a apenas 100 milisegundos y creciendo muy poco a poco.

En el caso de las agrupaciones de facturas grandes los tiempos también se han reducido bastante respecto a los mostrados en la figura 7.27. No obstante, la curva del tiempo de búsqueda en la BD crece de forma mucho más pronunciada que con las facturas pequeñas. Esto se debe al problema mencionado en 7.3.2 relacionado con el tamaño de los documentos en MongoDB. A medida que aumenta el número de particiones también lo hace el tiempo de búsqueda, esto se debe a que a cada partición extra, el sistema tiene que devolver mayor volumen de datos. La siguiente figura muestra el aumento del tiempo de búsqueda medio en función del número de particiones en las que esta dividida la agrupación guardada.

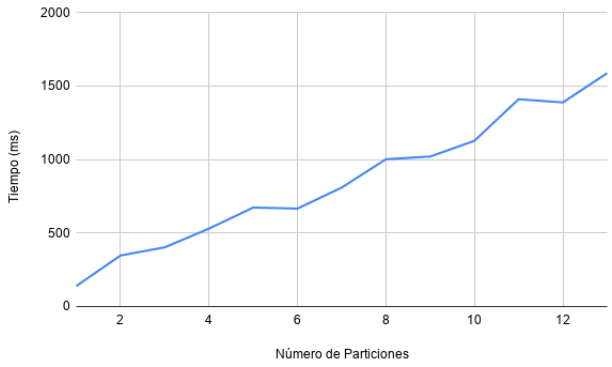



**Figura 7.36:** Tiempo medio de búsqueda por número de particiones de MongoDB

### 7.3.4. Corrección C.1

Anteriormente se ha podido comprobar que el método de búsqueda empleado en Cassandra no es nada óptimo, es por ello, que para realizar la búsqueda de las facturas se ha decidido emplear el mismo método que el utilizado en la BD de MongoDB. Aquí, los documentos de la colección que contiene las agrupaciones tienen un campo con los identificadores TBAI de las facturas que se están almacenando, de esta manera, no será necesario calcular el identificador TBAI de cada una de las facturas de la agrupación para después compararlo, sino que se realizará la comparación dentro de la lista auxiliar ahorrando tiempo y mejorando considerablemente el rendimiento.

| **Partition Key** | | **Clustering Key** | **Data** | | |
|---|---|---|---|---|---|
| Nif Emisor | Fecha Inicio | Fecha Inicio | Fecha Fin | Agrupación | Tbai_id_list |

**Tabla 7.1:** Modificación de la tabla que contiene las agrupaciones de facturas

En cuanto a los nuevos resultados de las consultas los siguientes gráficos muestran la mejora obtenida tras realizar la corrección.

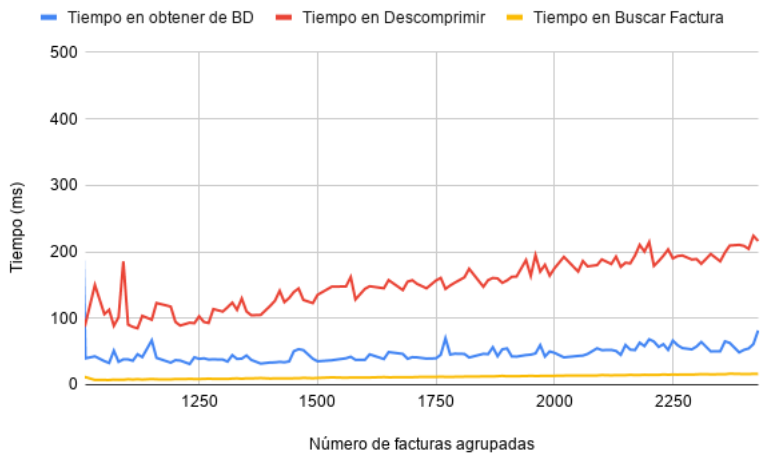



**Figura 7.37:** Tiempo de búsqueda de factura en agrupaciones pequeñas en Casssandra tras corrección

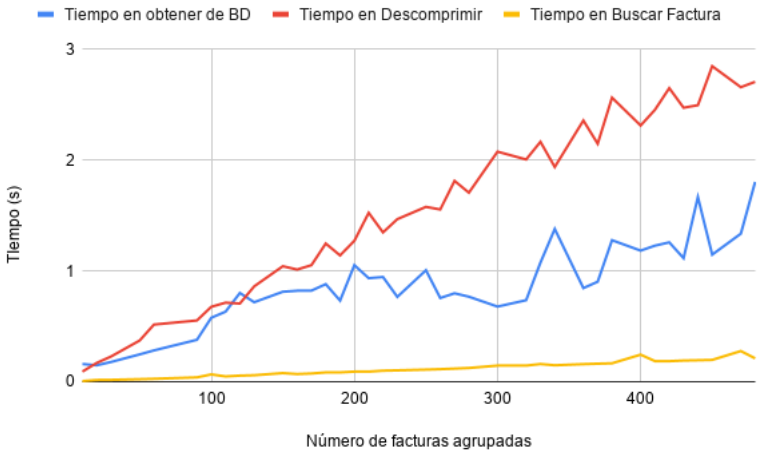



**Figura 7.38:** Tiempo de búsqueda de factura en agrupaciones grandes en Casssandra tras corrección

Tal como se ve en las figuras la mejora ha sido sustancial, mientras que los tiempos de obtención de la Base de Datos y de descompresión se han mantenido, el tiempo de búsqueda dentro de la agrupación se ha reducido considerablemente. El tiempo de búsqueda en agrupaciones de facturas pequeñas y grandes es de 150 milisegundos de media, y aumentando muy lentamente. A destacar, los tiempos de las facturas grandes, en los que el tiempo de descompresión de los datos es superior al de búsqueda en la Base de Datos.

Si comparamos estos resultados en ambos sistemas, el resultado es el siguiente.

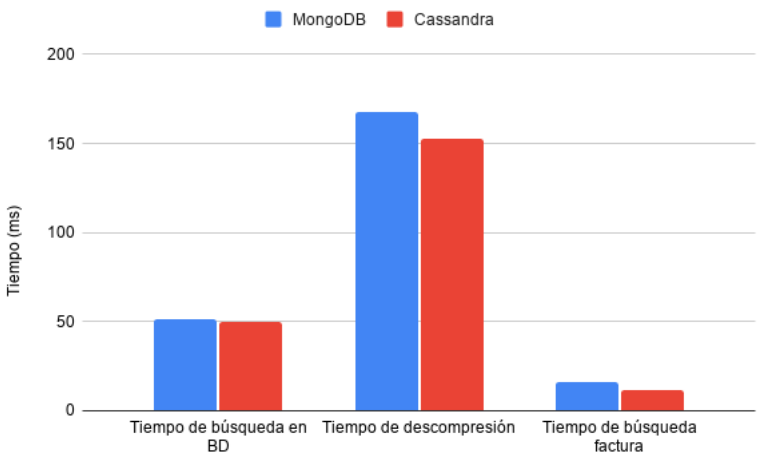



**Figura 7.39:** Comparación de búsqueda de facturas agrupadas pequeñas entre MongoDB y Cassandra

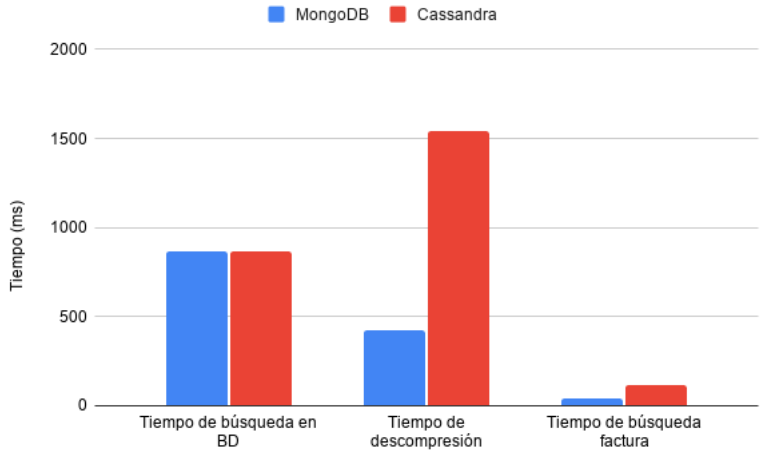



**Figura 7.40:** Comparación de búsqueda de facturas agrupadas grandes entre MongoDB y Cassandra

Tal como se ve en las figuras, para la búsqueda de facturas pequeñas en ambos sistemas los tiempos son similares, Cassandra es unos pocos milisegundos más rápido. No obstante, al realizar la búsqueda en agrupaciones de facturas grandes los resultados son diferentes. A pesar de que el tiempo de búsqueda en la Base de Datos es similar en ambos sistemas, los tiempos de descompresión y búsqueda de facturas son mucho más altos en Cassandra. Esto se debe a que en MongoDB, cuando se superaba el límite de 16MB de tamaño de agrupación esta se dividía en diferentes particiones, es por ello, que al descomprimir y buscar, el sistema deberá hacerlos en particiones mucho más pequeñas, mientras que Cassandra deberá buscar en la agrupación completa que tiene mayor volumen de datos.

## 7.4. Diseño final de las Bases de Datos

A lo largo de este capítulo se han mostrado las distintas pruebas realizadas sobre el sistema junto con los resultados de las mismas. Así mismo, también se han mostrado las distintas correcciones que se han llevado acabo sobre los esquemas iniciales de las Bases de Datos. Es por ello, que a continuación se muestran los esquemas finales de cada una de las Bases de Datos Diseñadas.

```
{
    _id : {
        type: Schema.Types.String,
        required: true
    },
    NIF : {
```

```
        type: Schema.Types.String,
        required : true
    },
    FechaExpediciónFactura : {
        type: Schema.Types.Date,
        required : true
    },
    ImporteTotalFactura : {
        type : Schema.Types.Number,
        required: true
    },
    SerieFactura : {
        type: Schema.Types.String,
        required : true
    },
    NumFactura : {
        type: Schema.Types.String,
        required : true
    },
    FacturaComprimida : {
        required : true,
        type : Schema.Types.String
    },
    status : {
        required : true,
        type: Schema.Types.Number
    }
}
```

**Listado 7.2:** Esquema final de la colección de facturas individuales de MongoDB

```
{
    nif : {
        type: Schema.Types.String,
        required: true
    },
    fechaInicio: {
        type: Date,
        required: true
    },
    fechaFin: {
        type: Date,
        required: true
    },
    idents : {
        type: Schema.Types.Array,
        required : true
    },
    agrupacion : {
        type: Schema.Types.String,
        required : true
```

```
    }
}
```

**Listado 7.3:** Esquema final de la colección de facturas agrupadas de MongoDB

| **Partition Key** | | **Clustering Key** | **Data** | | | |
|---|---|---|---|---|---|---|
| Nif Emisor | Fecha Exp | Ident-TBAI | SerieFact | Num Fact | Importe | xml |

**Tabla 7.2:** Esquema final de la tabla de facturas individuales en Cassandra

| **Partition Key** | | **Clustering Key** | | **DATA** | |
|---|---|---|---|---|---|
| Nif Emisor | Fecha Inicio | Fecha Inicio | Fecha Fin | Agrupación | Tbai_id_list |

**Tabla 7.3:** Esquema final de la tabla de facturas agrupadas en Cassandra

# 8. CAPÍTULO

# Otras aplicaciones del sistema

A lo largo de los capítulos anteriores se han visto algunas de las consultas a realizar sobre el sistema, las cuales se han tenido en cuenta a la hora de realizar el diseño del mismo. Durante este capítulo se detallan otros usos de las Bases de Datos diseñadas, basados en las consultas y estadísticas definidas en la sección 6.2.

## 8.1. Consultas estadísticas

Durante esta sección se describirán algunas, de las posibles, consultas estadísticas a las que podrá responder el sistema. El objetivo de estos datos estadísticos es ofrecer información de ventas a los distintos sectores emisores de facturas de Gipuzkoa.

Para poder probar el funcionamiento del sistema se han generado facturas emitidas por dos tipos de sectores.

- **Sector al por menor**. Las facturas emitidas por las entidades de este sector serán de tamaño reducido. Facturas, que individualmente, no contienen un importe total muy elevado, pero las entidades que las emiten lo hacen un número muy elevado de veces al día.
- **Sector al por mayor**. Las entidades de este sector representan el extremo opuesto al caso anterior. Las facturas emitidas contendrán un número elevado de productos junto con un importe total mayor, aunque la emisión diaria será muy reducida,

incluso cercana a cero. Otra característica fundamental de las facturas de este tipo será el país al que se exportan los productos de la factura.

### 8.1.1. Sector al por menor

Tal como se adelantaba anteriormente, las facturas de este sector se caracterizan por el reducido importe total de las facturas, junto con su elevado volumen de emisión diaria. Por ello, se han definido tres tipos de consultas que podrían resultar interesantes.

1. Importe medio de las facturas emitidas en un día concreto.
2. Importe medio de las facturas emitidas en una semana concreta.
3. Importe medio de las facturas emitidas a lo largo de un mes concreto.

La siguiente figura muestra un ejemplo de la primera consulta mencionada anteriormente. En ella se puede ver la comparativa del importe medio de las facturas emitidas durante las horas de un día concreto.


**Figura 8.1:** Comparativa de empresa concreta frente a su sector (Al por Menor)

Tomando como ejemplo la figura anterior, la empresa con el NIF **21039050F** emitiría, en general, facturas con importe medio superior a las empresas de su sector, exceptuando algunos bajones en ciertas horas.

Con la ayuda de este tipo de gráficos las empresas o locales podrán ver la situación exacta de su establecimiento en comparativa con el resto del sector pudiendo estimar las mejorar necesarias para su preferible desarrollo.

### 8.1.2. Sector al por mayor

En lo que al sector al por mayor se refiere las consultas definidas anteriormente podrían no ser idóneas, ya que, como se mencionaba antes, las empresas de este sector generan un número reducido de facturas.

Tal como se adelantaba anteriormente, una de las características de este tipo de facturas es su país de exportación, es por ello, que algunas de las consultas a definir para este grupo son las siguientes.

1. Número de facturas emitidas a los distintos países.
2. Importe medio de las facturas emitidas a los distintos países.

La gráfica que se muestra a continuación representa las dos consultas mencionadas anteriormente

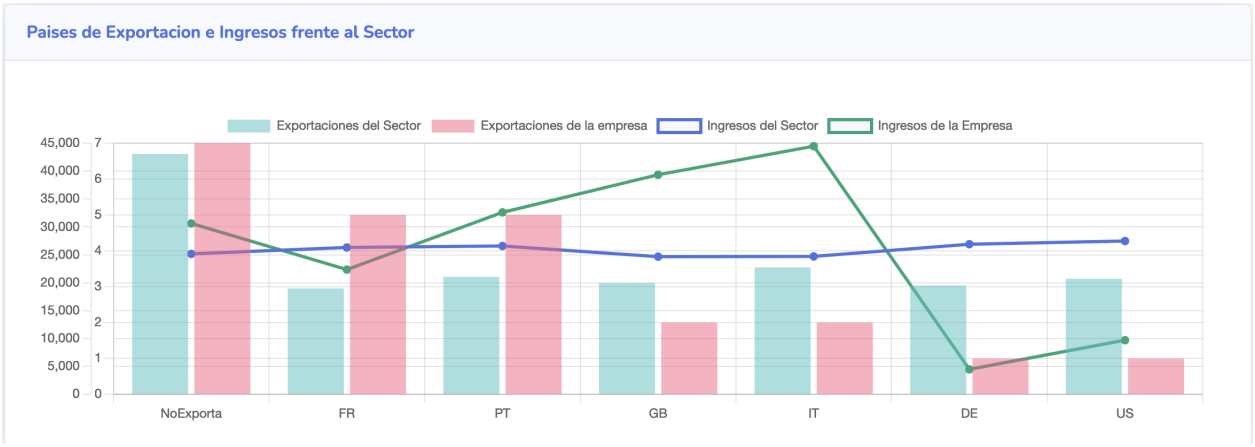



**Figura 8.2:** Comparativa de empresa concreta frente a su sector (Al por Mayor)

Tomando como ejemplo la figura anterior, la empresa con el NIF **96040464P** venderá a la mayoría de sus productos a empresas nacionales. No obstante, parte de sus ingresos los obtiene al exportar productos a países como Portugal o Italia, siendo este último el país del que obtiene mayores ingresos a pesar de emitir un número de facturas menor que su sector.

# 9. CAPÍTULO

# Seguimiento y Control del Proyecto

A lo largo del proyecto se ha realizado el seguimiento y control del mismo, realizando las modificaciones necesarias en la planificación de manera que se asegure la correcta finalización del TFG.

Durante este capítulo se detallan las distintas incidencias surgidas junto con los cambios resultantes en la planificación y desarrollo del proyecto.

## 9.1. Incidencias

Tal como se adelantaba, en este apartado se detallarán las incidencias más significativas surgidas a lo largo del proyecto. Algunas de las incidencias surgidas se han correspondido con los riesgos ya previstos al comienzo del proyecto pero que, sin embargo, no han podido ser eludidos.

### 9.1.1. Retraso en el diseño de las Bases de Datos

Durante la semana del 22 de febrero se tuvo una reunión con los promotores del proyecto, Diputación Foral, en la que se acordó que estos proporcionasen información sobre el proyecto más real, es decir, facturas de ejemplo que se asemejasen al comportamiento real de una empresa o establecimiento inventados.

Debido a ello, las fases de trabajo relacionadas con el diseño e implementación de las BD en los distintos SGBDs se han visto retrasadas, quedando a la espera de recibir los datos mencionados.

### 9.1.2. Exclusión del diseño e implementación de las BDs en Neo4J

Otro de los aspectos tratados en la reunión del día 22 de febrero guardaba relación con los SGBDs que se utilizarían. Concretamente, se descartó el diseño e implementación de la Base de Datos en Neo4J.

No obstante, al darse esta incidencia en una fecha temprana del proyecto no tuvo un impacto excesivamente grande, ya que únicamente se habían tratado las técnicas de compresión que utilizaba este SGBD sin haber empezado aún con el diseño de la Base de Datos.

## 9.2. Desviaciones de la Planificación

En la sección anterior se han mostrado las incidencias más significativas aparecidas durante el transcurso del proyecto

### 9.2.1. Nueva tarea en el paquete de trabajo *Conocimientos Técnicas de Compresión*

De cara a complementar la teoría mostrada sobre las técnicas de compresión sin pérdida se decidió añadir una tarea dedicada a la adquisición de conocimientos sobre las técnicas de compresión con pérdida más utilizadas. A pesar de que este tipo de técnicas no se utilizarían en el proyecto se decidió abordarlas para completar los conocimientos de las técnicas anteriores.

### 9.2.2. Necesidad de creación de un Generador de Facturas Aleatorio

Dado que para realizar las pruebas de rendimiento sobre los sistemas implementados era necesario trabajar con un elevado número de facturas se vio necesario crear un proceso que automatizase la creación de las mismas.

Por tanto, se incluyó una nueva tarea dentro del paquete de trabajo *Implementación*, **Generador de Facturas**. El objetivo de esta tarea, sería abarcar todos los trabajos a realizar relacionados con la generación de las facturas, incluyendo la compresión de la estructura de una factura y de los valores de los campos que estas incluyen.

## 9.3. Desviaciones de tiempo

Debido a las incidencias mostradas en el apartado anterior, los tiempos de finalización de algunas tareas y paquetes de trabajo se han desviado respecto a los planificados inicialmente.

En la figura 9.1 se muestra una comparación entre las fechas de finalización previstas y las fechas de finalización reales de los distintos paquetes de trabajo definidos en el proyecto.

| Paquetes de Trabajo | Fecha Finalización Prevista Inicial | Fecha Finalización Real | Desviación |
|---|---|---|---|
| **Adquisición de Conocimientos** | 03/03/2021 | 05/03/2021 | **+2 días** |
| Conocimientos SGBD (CSGBD) | 13/02/2021 | 16/02/2021 | **+6 días** |
| Conocimientos entorno WEB (CWEB) | 25/02/2021 | 17/02/2021 | **+1 día** |
| Conocimientos Técnicas de Compresión (CTC) | 03/03/2021 | 05/03/2021 | **+2 días** |
| **Diseño de la Aplicación** | 31/03/2021 | 25/05/2021 | **+1 mes y 8 días** |
| Diseño de las BDs (DBD) | 31/03/2021 | 22/03/2021 | **-7 días** |
| Diseño de la Aplicación WEB (DWEB) | 04/03/2021 | 25/05/2021 | **+1 mes y 27 días** |
| **Implementación** | 22/04/2021 | 10/06/2021 | **+1 mes y 3 días** |
| Implementación de las BDs (IBD) | 17/04/2021 | 12/04/2021 | **-5 días** |
| Implementación del entorno WEB (IWEB) | 22/04/2021 | 10/06/2021 | **+ 1 mes y 27 días** |
| Implementación Generador de Facturas (IGF) | -- | 23/03/2021 | **Inicialmente no Planificado** |
| **Pruebas y Validación** | 11/06/2021 | 14/06/2021 | **+ 3 días** |
| Pruebas y Validación de las BDs (PVBD) | 11/06/2021 | 14/06/2021 | **+ 3 días** |
| Pruebas y Validación del entorno WEB (PVWEB) | 07/06/2021 | 10/06/2021 | **+3 días** |
| Pruebas y Validación de las T.C (PVTC) | 31/03/2021 | 25/03/2021 | **-6 días** |
| **Gestión** | 20/06/2021 | 20/06/2021 | **0 días** |
| Planificación (P) | 20/06/2021 | 20/06/2021 | **0 días** |
| Seguimiento y Control (SyC) | 20/06/2021 | 20/06/2021 | **0 días** |
| **Documentación** | 20/06/2021 | 20/06/2021 | **0 días** |
| Elaboración de la Memoria (EM) | 20/06/2021 | 20/06/2021 | **0 días** |
| Elaboración y preapración de la defensa (EDEF) | 20/06/2021 | 20/06/2021 | **0 días** |

**Figura 9.1:** Fecha de finalización de paquetes de trabajo estimada frente a la real

Así mismo, en la figura 9.2 se muestra la comparación entre dedicaciones estimadas y reales a cada uno de los paquetes de trabajo del proyecto. Además se recoge el tiempo de dedicación total al proyecto, estimado y real.

| Paquetes de Trabajo | Horas de Dedicación Estimadas | Horas de Dedicación Reales |
|---|---|---|
| **Adquisición de Conocimientos** | 54 | 68 |
| Conocimientos SGBD (CSGBD) | 24 | 30 |
| Conocimientos entorno WEB (CWEB) | 8 | 10 |
| Conocimientos Técnicas de Compresión (CTC) | 22 | 28 |
| **Diseño de la Aplicación** | 64 | 35 |
| Diseño de las BDs (DBD) | 56 | 25 |
| Diseño de la Aplicación WEB (DWEB) | 8 | 10 |
| **Implementación** | 44 | 108 |
| Implementación de las BDs (IBD) | 34 | 35 |
| Implementación del entorno WEB (IWEB) | 10 | 53 |
| Implementación Generador de Facturas (IGF) | 0 | 20 |
| **Pruebas y Validación** | 61 | 81 |
| Pruebas y Validación de las BDs (PVBD) | 31 | 56 |
| Pruebas y Validación del entorno WEB (PVWEB) | 12 | 15 |
| Pruebas y Validación de las T.C (PVTC) | 18 | 10 |
| **Gestión** | 45 | 44 |
| Planificación (P) | 15 | 15 |
| Seguimiento y Control (SyC) | 30 | 29 |
| **Documentación** | 43 | 72 |
| Elaboración de la Memoria (EM) | 31 | 57 |
| Elaboración y preapración de la defensa (EDEF) | 10 | 15 |
| **Horas Totales** | 311 | 408 |

**Figura 9.2:** Horas de dedicación a las tareas y paquetes de traba

Como se puede observar, la dedicación final del proyecto ha sido superior a la prevista inicialmente. Gran parte de este aumento se debe a algunas de las incidencias antes mencionadas. La necesidad de crear un nuevo paquete de trabajo, *Implementación Generador de Facturas*, junto con un retraso en la implementación del entorno web, ha supuesto un gran aumento en la dedicación de horas estimadas a la fase de trabajo *Implementación*.

En general, la dedicación real del proyecto se ha desviado bastante de lo planificado inicialmente. Esto ha sido en parte por una mala previsión del tiempo estimado junto con una clara falta de experiencia en la gestión de proyectos de este tamaño, dando lugar a un claro margen de mejora en la gestión de proyectos.

# 10. CAPÍTULO

# Conclusiones y líneas futuras

## 10.1. Conlusiones

Este TFG se planteó como una prueba de concepto capaz de demostrar el potencial de los sistemas NoSQL en un entorno del *Big Data* relacionado con la gestión de facturas por parte de una Administración Públoca, y una vez finalizado el proyecto se pude afirmar que este ha sido un éxito. Se han cumplido correctamente los objetivos definidos al comienzo del proyecto. A pesar de que han surgido algunas incidencias durante el desarrollo del proyecto, realizando los cambios necesarios en la planificación, se han conseguido solventar logrando un resultado satisfactorio.

Las conclusiones detalladas de este TFG se han desarrollado desde dos puntos de vista diferentes: las conclusiones a nivel técnico y las conclusiones a nivel personal. A continuación se describen en profundidad cada una de estas conclusiones finales.

### 10.1.1. Conclusiones a nivel técnico

Las conclusiones a nivel técnico del proyecto son muy positivas. Durante el transcurso del mismo se han obtenido numerosas nuevas competencias que me han permitido realizar las siguientes tareas:

1. **Selección de la técnica de compresión sin pérdida más adecuada**. Se ha realizado una comprensión y análisis de algunos de los algoritmos de compresión de datos

sin pérdida más utilizados. De estos, se han evaluado los tiempos de compresión y descompresión de los datos junto con el ratio de compresión de los mismos.

2. **Diseño e implementación de dos BDs NoSQL reales**. Se han diseñado dos Bases de Datos en dos SGBDs diferentes, MongoDB y Cassandra, estas Bases de Datos debían tener un diseño y prestaciones capaces de suplir los requisitos establecidos para el dominio correspondiente.

3. **Evaluación de rendimiento de dos BDs NoSQL reales**. Se ha realizado una comparativa entre el rendimiento mostrado por cada una de las Bases de Datos diseñadas para el sistema. El objetivo no era decantarse por un diseño u otro, sino ver los puntos fuertes y débiles de cada uno de los sistemas en el dominio correspondiente.

4. **Diseño e implementación de un entorno web**. Se ha diseñado e implementado un entorno web que simula el entorno final de la aplicación TicketBai. Gracias a este entorno se podrán ver todas las posibilidades que ofrece el sistema diseñado.

### 10.1.2. Conclusiones a nivel personal

Por último, en lo que a las conclusiones a nivel personal se refiere, estas son también positivas. Se han adquirido diferentes tipos de conocimientos relacionados con el trabajo en un proyecto real. Las conclusiones son las que se muestran a continuación.

1. **Aprendizaje sobre las técnicas de compresión de datos**. Los conocimientos que tenía sobre este campo eran muy simples, nada más allá de los programas de compresión que todos conocemos. Gracias a este TFG he podido conocer como funcionan por detrás algunos de los algoritmos de compresión de datos más utilizados en la actualidad. Así mismo, también he podido conocer algunas nuevas aplicaciones de estos algoritmos que no conocía hasta la fecha.

2. **Aprendizaje sobre sistemas NoSQL**. A lo largo de la carrera se habían tratado muy poco los SGBD NoSQL, únicamente se habían trabajado MongoDB y Neo4J, ambos con los conocimientos básicos. Gracias a este TFG no solo he podido conocer otros SGBD NoSQL como Cassandra, sino que también he podido conocer más en profundidad el funcionamiento de los SGBDs trabajados.

3. **Creación de un entorno web que permita probar el rendimiento de los sistemas implementados**. A pesar de que no era un objetivo estrictamente necesario para

alcanzar el fin principal del proyecto, de cara a complementarlo y ofrecer un entorno de visualización de los datos y consultas se creó este entorno web. Me ha fascinado crear un entorno que permita obtener, mostrar y trabajar un gran volumen de datos.

4. **Interacción con usuario final del proyecto**. Una de las cosas que más me han fascinado del proyecto ha sido la posibilidad de interaccionar con el usuario final del mismo, poder mostrarles los avances del proyecto así como obtener los requisitos necesarios para el mismo.

5. **Realizar y gestionar un proyecto de esta magnitud de principio a fin**. A pesar de que el TFG se haya finalizado con éxito, considero que aún hay mucho margen de mejora. Debido a la falta de experiencia en la gestión de proyectos de esta envergadura muchos de los aspectos de la planificación son mejorables pudiendo haber previsto mejor algunas de las incidencias surgidas. A pesar de ello, considero que se ha obtenido un buen producto final acorde con los objetivos planteados al inicio del proyecto.

## 10.2. Lineas Futuras

El TFG se ha finalizado habiendo alcanzado todos los objetivos planteados al comienzo del mismo. No obstante, es posible realizar ciertas mejoras que podrían implementarse en un futuro cercano.

### 10.2.1. Mejoras sobre el diseño de las Bases de Datos

De cara a mejorar el rendimiento de las Bases de Datos algunos aspectos fuera del alcance de este proyecto que se pueden realizar son:

- **Pruebas de rendimiento con una carga de datos real**. Durante el TFG se ha evaluado el rendimiento de las Bases de Datos realizando consultas de forma individual. El objetivo de estas pruebas sería comprobar si el sistema ofrece unos tiempos de respuesta óptimos en un contexto en el que múltiples preguntas se ejecutarán de manera concurrente.

- **Mejorar la organización de los datos en RAW de las BDs**. Las Bases de Datos se han construido con algunos datos en RAW y la factura original comprimida. Realizar

un análisis exhaustivo sobre otros campos de la factura que podrían agregarse al conjunto de datos en RAW mejoraría el rendimiento de ciertas consultas estadísticas y ampliando así las funcionalidades ofrecidas por la Base de Datos.

- **Diseño de las BDs de forma distribuida**. Uno de los puntos más fuertes de los SGBDs NoSQL es el poder trabajar con ellos de forma distribuida. A lo largo de todo el TFG se ha trabajado con las BDs configuradas para trabajar en un único nodo, por tanto, una mejora en el sistema sería la configuración del mismo en un entorno distribuido.

### 10.2.2. Mejoras sobre el entorno web

A pesar de que el entorno web se haya desarrollado como "demo" de las capacidades del sistema, existen una serie de aspectos que se podrían mejorar:

- **Mejora en el *frontend***. De cara a que el entorno web permita ser utilizado, no solo como demo para este proyecto, sino como demo a nivel global, una mejora a realizar seria el *frontend* del mismo.
- **Optimización de los cálculos**. Algunas de las funciones que se utilizan en el entorno web resultan parcialmente ineficientes. Una optimización de estas funciones derivaría en unos mejores tiempos de respuesta de la página web.

# Anexos

# A. ANEXO

## Sentencias CQL de Cassandra

En este anexo se presentan las sentencias CQL de Cassandra más significativas y utilizadas durante todo el proyecto.

### A.1. Creación del Keyspace

```
1 CREATE KEYSPACE ticketbai WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor' :
      1};
```

**Listado A.1:** Creación del Keyspace de Cassandra

Dado que el sistema estará formado por una única máquina, para realizar todas las pruebas se ha utilizado un una estrategia simple (*SimpleStrategy*), la cual prepara el *Keyspace* para utilizar un único nodo y con un factor de replicación de 1. Como ya se comentaba esta estrategia esta pensada para realizar diseños "simples", para sacar el máximo rendimiento del SGBD se utiliza la estrategia *NetworkTopolyStrategy*, la cual está pensada para definir varios centros de datos, cada uno con un factor de replicación diferente, esta es la estrategia más utilizada y extendida.

## A.2. Sentencias de creación de las tablas

A continuación se muestran las sentencias empleadas para generar las tablas finales, es decir, las tablas utilizadas en cassandra junto con las correciones aplicadas en 7.3.

```
1 CREATE TABLE IF NOT EXISTS facturas (nif text, fecha date, tbai_id text, importe double,
      num_factura int, serie text, xml text, PRIMARY KEY ((nif, fecha), tbai_id)) WITH CLUSTERING
      ORDER BY (tbai_id ASC);
```

**Listado A.2:** Creación de la tabla facturas

```
1 CREATE TABLE IF NOT EXISTS facturas_agrupadas (nif text, fecha_inicio date, agrupacion text,
      fecha_fin date, tbai_id_list list<text>, PRIMARY KEY (nif, fecha_inicio, fecha_fin)) WITH
      CLUSTERING ORDER BY (fecha_inicio ASC, fecha_fin ASC);
```

**Listado A.3:** Creación de la tabla facturas_agrupadas

## A.3. Consultas más relevantes

Seguidamente, se listan algunas de las consultas más empleadas a lo largo del transcurso del proyecto.

```
1 SELECT nif, fecha, tbai_id, importe, num_factura, serie from facturas where nif=nif and fecha=
      fecha and tbai_id=tbai_id;
```

**Listado A.4:** Consulta del qr de una factura

```
1 SELECT fecha_fin, tbai_id_list, agrupacion from facturas_agrupadas where nif=nif and fecha_inicio
       <= fecha;
```

**Listado A.5:** Consulta del qr de una factura en la agrupación

# B. ANEXO

## Sentencias y consultas de MongoDB

En este anexo se presentan las sentencias de MongoDB más significativas y utilizadas durante todo el proyecto. Así mismo, también se muestran los esquemas iniciales de las colecciones de MongoDB, para ver los esquemas finales ver la sección 7.4

### B.1. Colecciones creadas en MongoDB

```
{
    _id : {
        type: Schema.Types.String,
        required: true
    },
    NIF : {
        type: Schema.Types.String,
        required : true
    },
    FechaExpedicionFactura : {
        type: Schema.Types.Date,
        required : true
    },
    HoraExpedicionFactura : {
        type: Schema.Types.Date,
        required : true
    },
    ImporteTotalFactura : {
        type : Schema.Types.Number,
        required: true
    },
```

```
    SerieFactura : {
        type: Schema.Types.String,
        required : true
    },
    NumFactura : {
        type: Schema.Types.String,
        required : true
    },
    Descripcion :{
        type: Schema.Types.String,
        required : true
    },
    DetallesFactura: {
        required : true,
        type : [
            {
                DescripcionDetalle : {
                    type: Schema.Types.String,
                    required : true
                },
                Cantidad : {
                    type: Schema.Types.Decimal128,
                    required : true
                },
                ImporteUnitario : {
                    type: Schema.Types.Number,
                    required : true
                },
                ImporteTotal : {
                    type: Schema.Types.Number,
                    required : true
                }
            }
        ]
    },
    FacturaComprimida : {
        required : true,
        type : Schema.Types.String
    }
}
```

**Listado B.1:** Esquema inicial de las facturas en MongoDB

```
{
    _id : {
        type: Schema.Types.String,
        required: true
    },
    idents : {
        type: Schema.Types.Array,
        required : true
```

```
    },
    agrupacion : {
        type: Schema.Types.String,
        required : true
    }
}
```

**Listado B.2:** Esquema inicial de la colección de agrupación de facturas

## B.2. Índices creados sobre los campos necesarios

```
db.facturas_agrupadas.createIndex({"nif": 1, "fechaInicio": 1}, {name: "nifFechaIndex"});
```

**Listado B.3:** Creación de índice para las facturas agrupadas

```
db.facturas.createIndex({"nif": 1, "fecha":1}, {name: "nifFechaIndex"});
```

**Listado B.4:** Creación de índice sobre el NIF y la fecha en las facturas individuales

## B.3. Consultas sobre las colecciones

```
db.facturas.find({NIF: "nif"});
```

**Listado B.5:** Obtención de facturas emitidas por un NIF

```
db.facturas_agrupadas.find(
{
    nif: nif,
    fechaInicio: {$lte: fecha},
    fechaFin: {$gte: fecha}
});
```

**Listado B.6:** Consulta de obtención de facturas agrupadas

```

db.facturas.aggregate([
    {
        $match : {
```

```
            $nif : nif,
            fecha: {
                $gte: fechaInicio,
                $lte: fechaFin
            }
        }
    },
    {
        $group : {
            $_id: null,
            total: {
                $sum : "$cantidad"
            },
            media : {
                $avg : "$cantidad"
            }
        }
    }
]);
```

**Listado B.7:** Total y media de ingresos de un NIF en un rango de fehcas

# C. ANEXO

## Diseño e implementación de la página web

Durante este anexo se presentarán las diferentes vistas del entorno web desarrollado para el proyecto. Así mismo, el código de la página web junto con el del generador de facturas quedan accesibles a través del enlace https://github.com/gorokotkd/tfg_nodejs

### C.1. Vistas de la página WEB

A continuación se muestran las distintas vistas implementadas para la página web. A través de estas vistas se quiere mostrar las distintas posibilidades que ofrecen las Bases de Datos implementadas.

La figura C.1 muestra la página de inicio del sistema. En esta, un usuario o empresa, tras iniciar sesión, podrá ver el "estado" de su negocio. En ella, podrá ver los ingresos que ha tenido durante un día y un mes concreto.

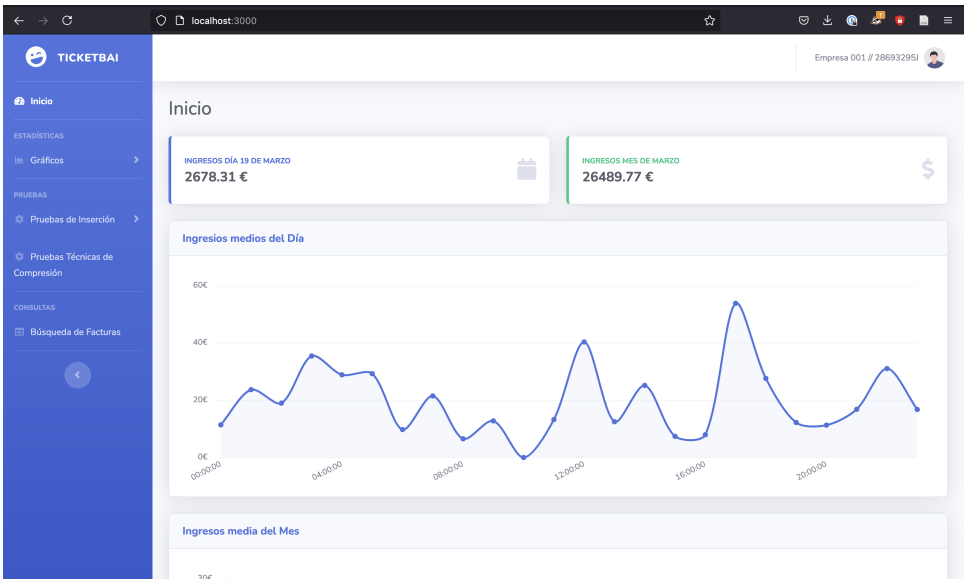



**Figura C.1:** Vista inicial de la web

Las figuras C.2 y C.3 muestran estadísticas para los sectores al por menor y al por mayor, respectivamente. En ambas se muestra una comparativa de una empresa, a través del NIF dado, frente a la media de las empresas de su sector.

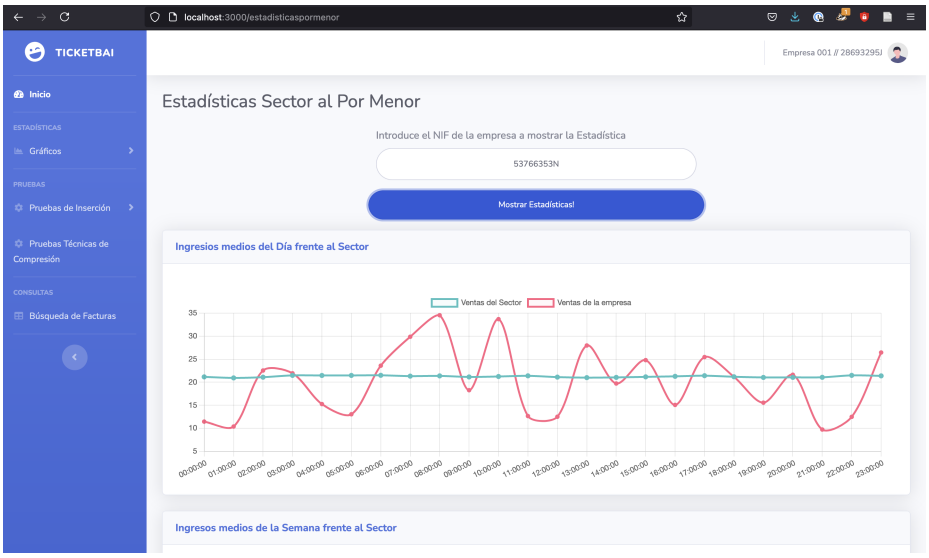



**Figura C.2:** Vista de consulta de estadísticas del sector al por menor

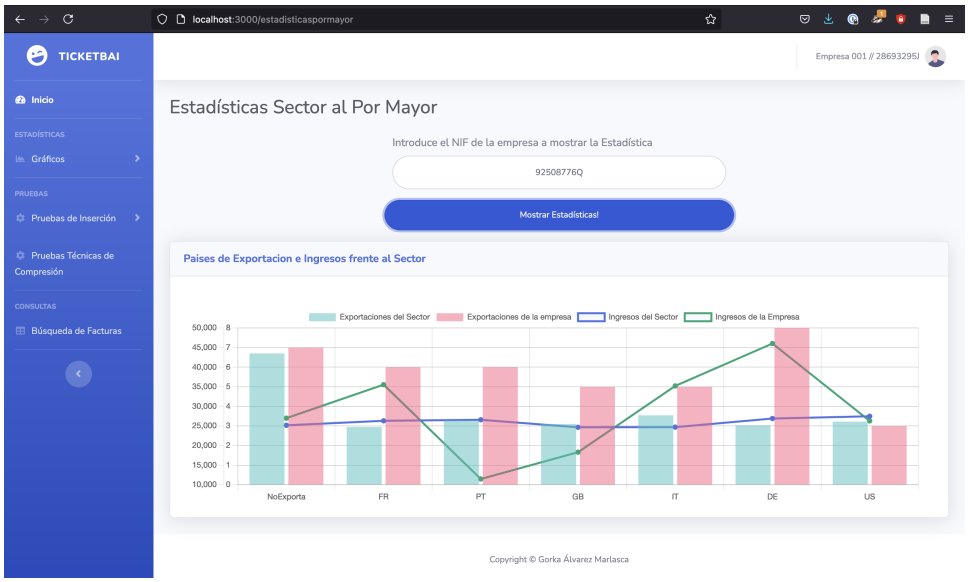



**Figura C.3:** Vista de consulta de estadísticas del sector al por mayor

la figura C.4 muestra una comparativa entre las distintas técnicas de compresión abordadas durante el proyecto. En ella, se muestran los resultados de comprimir y descomprimir un grupo de 50 facturas utilizando distintas técnicas de compresión sin pérdida.

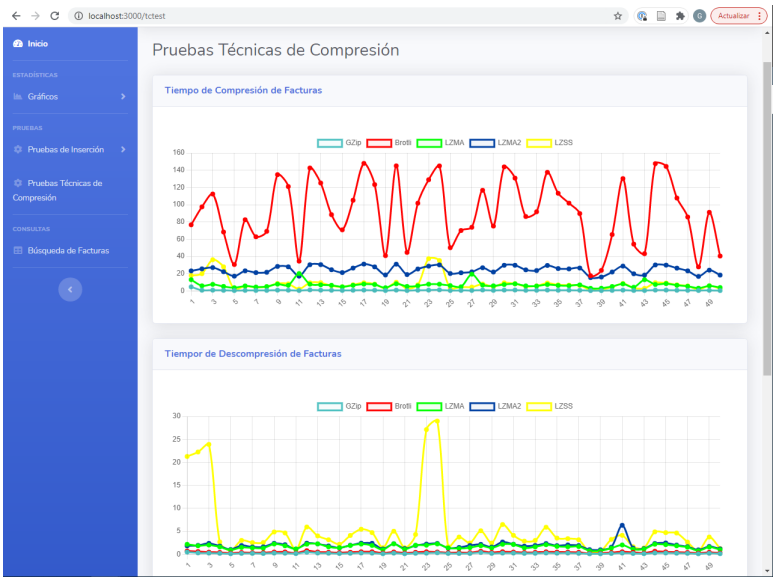



**Figura C.4:** Vista de pruebas sobre las técnicas de compresión

Las figuras C.5 y C.6 muestran los diferentes métodos de inserción de facturas. La primera, permite insertar facturas individuales en formato XML, tras la inserción devuelve el código QR que permite su búsqueda junto con el identificador correspondiente. la segunda figura,

permite realizar una inserción múltiple de facturas, a través de una lista despegable se puede seleccionar el número de facturas a agrupar e insertar, posteriormente, se muestra el código QR y el identificador junto con los gráficos con los datos resultantes de la inserción.


**Figura C.5:** Vista de pruebas sobre las técnicas de compresión

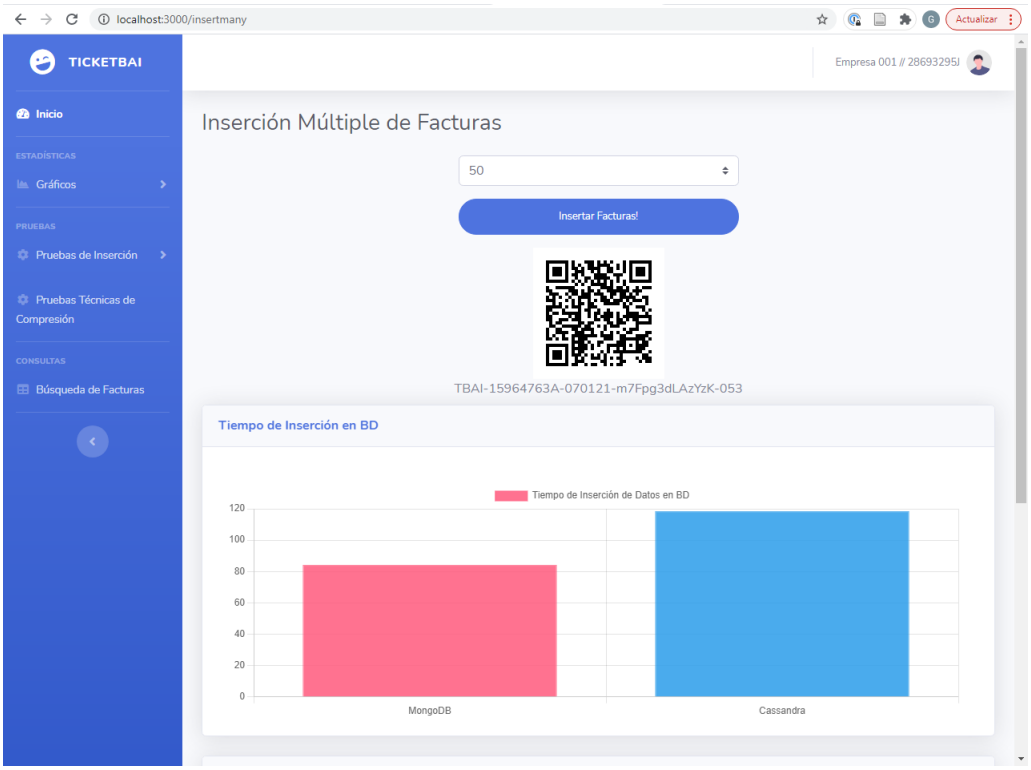



**Figura C.6:** Vista de pruebas sobre las técnicas de compresión

Por último, la figura C.7 muestra la vista resultante de buscar una factura por su identifica-

dor. La vista muestra una tabla con los datos básicos de la factura. Así mismo, se muestra una gráfica con el tiempo de búsqueda de la factura en cada BD.

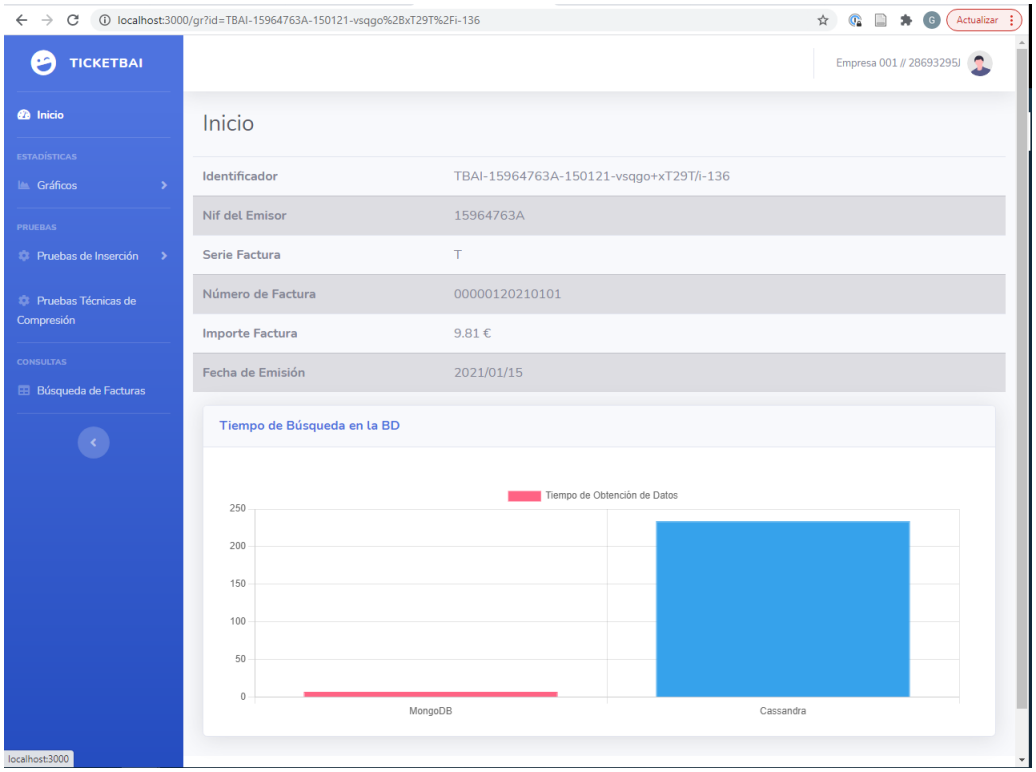



**Figura C.7:** Vista de búsqueda de una factura por su identificador

# D. ANEXO

## Diagrama de Gantt

De cara a complementar y ofrecer una mejor visualización del diagrama de Gantt mostrados en las figuras 3.3 y 3.4, a continuación se muestra el mismo diagrama pero organizado por semanas.

Para asegurar la correcta visualización el diagrama, este se ha divido en dos partes. La primera se corresponde con los meses de Enero a Marzo, figuras D.1 y D.2, la segunda, se corresponde con los meses de Abril a Julio, figuras D.3 y D.4.

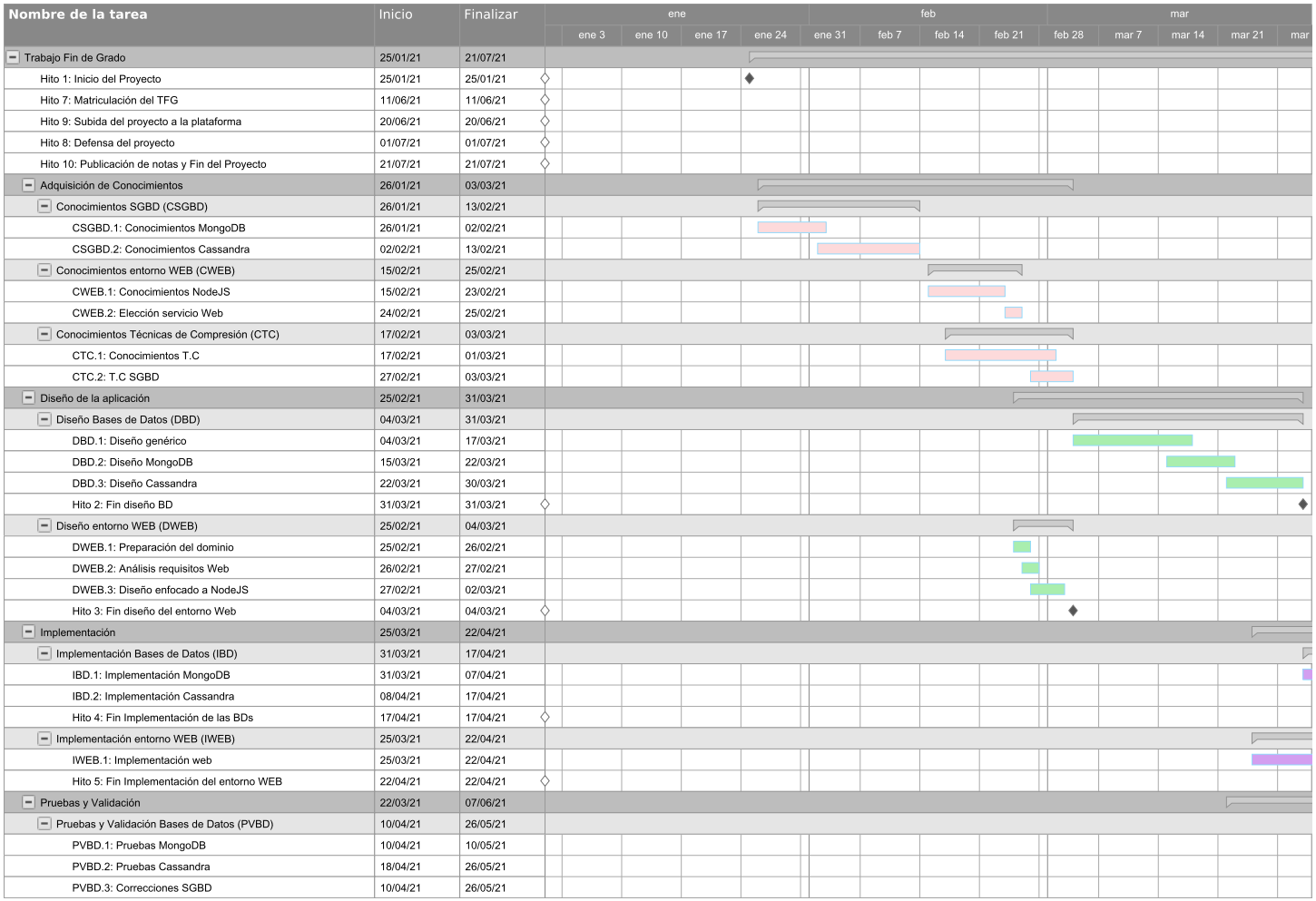
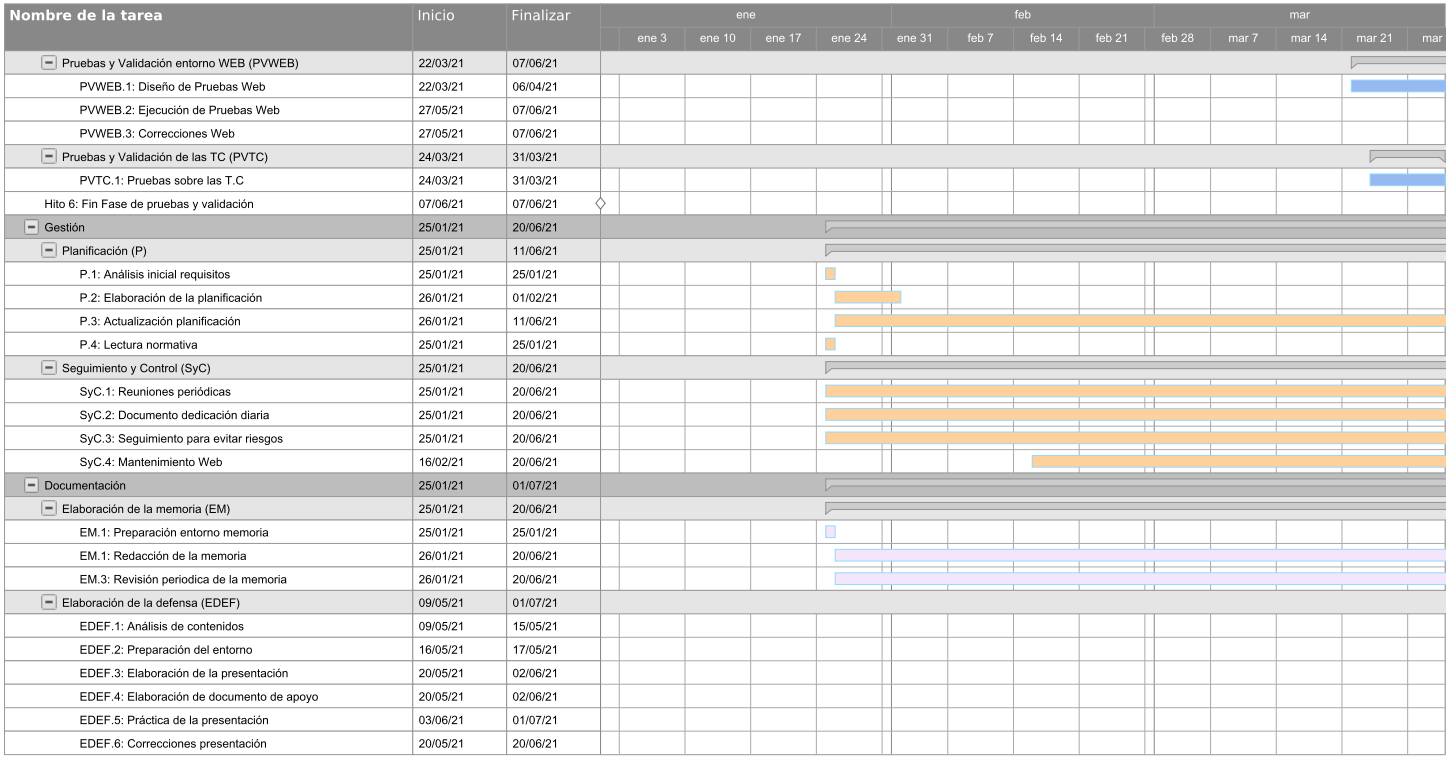



**Figura D.1:** Diagrama Gantt por semanas - Enero a Marzo - Primera Parte



| Nombre de la tarea | Inicio | Finalizar |
|---|---|---|
| Pruebas y Validación entorno WEB (PVWEB) | 22/03/21 | 07/06/21 |
| PVWEB.1: Diseño de Pruebas Web | 22/03/21 | 06/04/21 |
| PVWEB.2: Ejecución de Pruebas Web | 27/05/21 | 07/06/21 |
| PVWEB.3: Correcciones Web | 27/05/21 | 07/06/21 |
| Pruebas y Validación de las TC (PVTC) | 24/03/21 | 31/03/21 |
| PVTC.1: Pruebas sobre las T.C | 24/03/21 | 31/03/21 |
| Hito 6: Fin Fase de pruebas y validación | 07/06/21 | 07/06/21 |
| Gestión | 25/01/21 | 20/06/21 |
| Planificación (P) | 25/01/21 | 11/06/21 |
| P.1: Análisis inicial requisitos | 25/01/21 | 25/01/21 |
| P.2: Elaboración de la planificación | 26/01/21 | 01/02/21 |
| P.3: Actualización planificación | 26/01/21 | 11/06/21 |
| P.4: Lectura normativa | 25/01/21 | 25/01/21 |
| Seguimiento y Control (SyC) | 25/01/21 | 20/06/21 |
| SyC.1: Reuniones periódicas | 25/01/21 | 20/06/21 |
| SyC.2: Documento dedicación diaria | 25/01/21 | 20/06/21 |
| SyC.3: Seguimiento para evitar riesgos | 25/01/21 | 20/06/21 |
| SyC.4: Mantenimiento Web | 16/02/21 | 20/06/21 |
| Documentación | 25/01/21 | 01/07/21 |
| Elaboración de la memoria (EM) | 25/01/21 | 20/06/21 |
| EM.1: Preparación entorno memoria | 25/01/21 | 25/01/21 |
| EM.1: Redacción de la memoria | 26/01/21 | 20/06/21 |
| EM.3: Revisión periodica de la memoria | 26/01/21 | 20/06/21 |
| Elaboración de la defensa (EDEF) | 09/05/21 | 01/07/21 |
| EDEF.1: Análisis de contenidos | 09/05/21 | 15/05/21 |
| EDEF.2: Preparación del entorno | 16/05/21 | 17/05/21 |
| EDEF.3: Elaboración de la presentación | 20/05/21 | 02/06/21 |
| EDEF.4: Elaboración de documento de apoyo | 20/05/21 | 02/06/21 |
| EDEF.5: Práctica de la presentación | 03/06/21 | 01/07/21 |
| EDEF.6: Correcciones presentación | 20/05/21 | 20/06/21 |



**Figura D.2:** Diagrama Gantt por semanas - Enero a Marzo - Segunda Parte

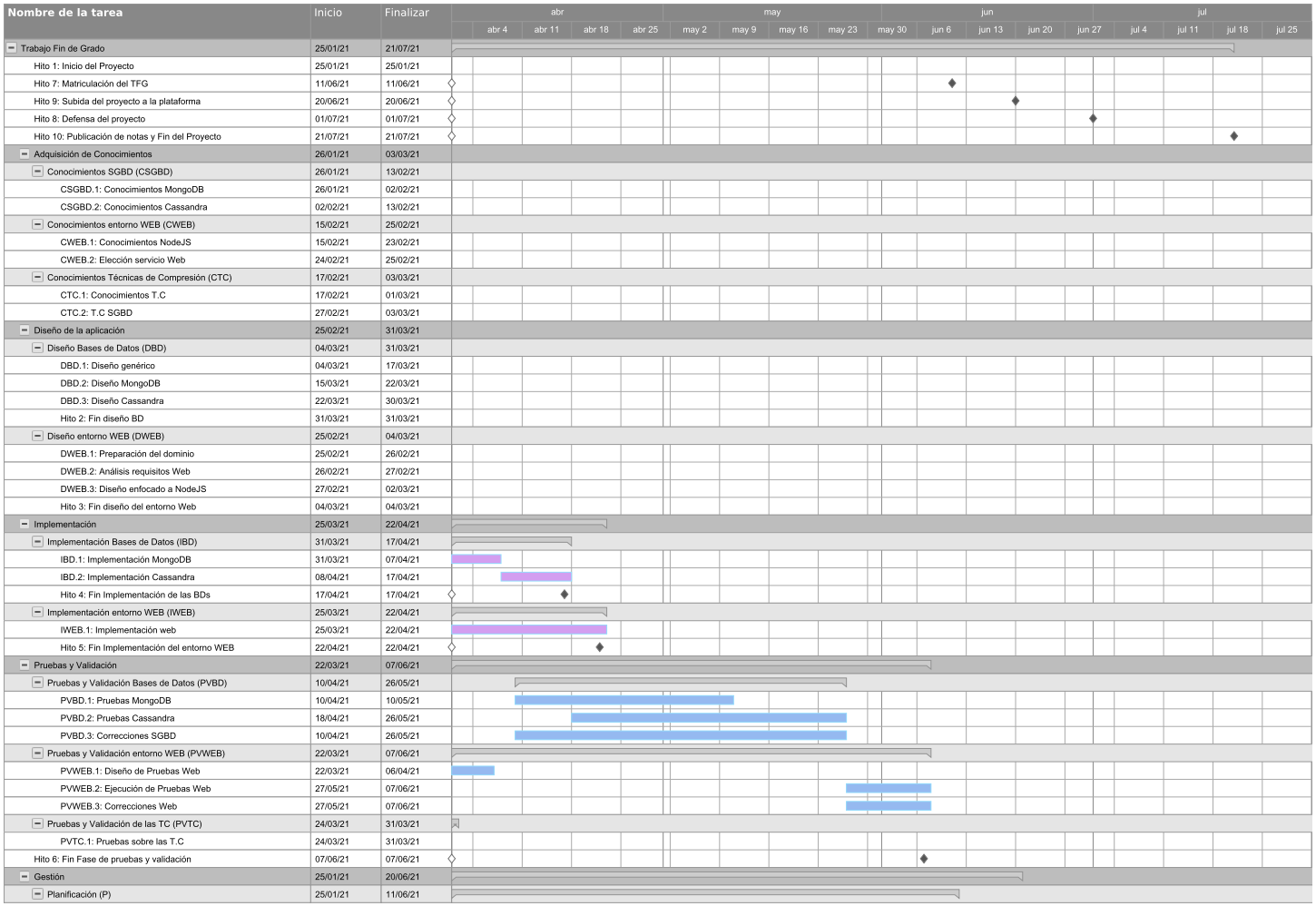



**Figura D.3:** Diagrama Gantt por semanas - Abril a Julio - Primera Parte

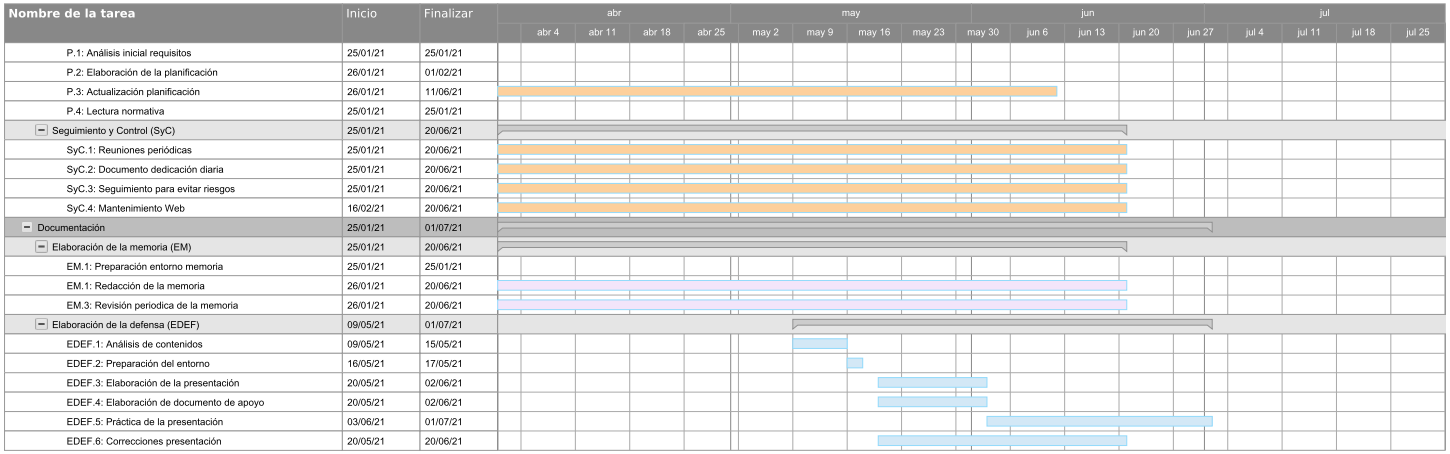



**Figura D.4:** Diagrama Gantt por semanas - Abril a Julio - Segunda Parte

# Bibliografía


[Acc, 2016] (2016). Accept-encoding: Br on HTTPS connection - Chrome Platform Status. https://www.chromestatus.com/feature/5420797577396224. [En línea; consultado el 16-febrero-2021].

[Fir, 2016] (2016). Firefox 44.0, See All New Features, Updates and Fixes. https://www.mozilla.org/en-US/firefox/44.0/releasenotes/. [En línea; consultado el 16-febrero-2021].

[Administración Electrónica, 2021] Administración Electrónica (2021). Módulo de firmas xml. https://administracionelectronica.gob.es/ctt/resources/Soluciones/138/Descargas/CFv3-4-Manual-de-firmas-XML.pdf?idIniciativa=138&idElemento=4147. [En línea; consultado el 11-abril-2021].

[Alakuijala and Szabadka, 2016] Alakuijala, J. and Szabadka, Z. (2016). Brotli Compressed Data Format. (RFC 7932). [En línea; consultado el 16-febrero-2021].

[Apache, 2016] Apache, C. (2016). Cassandra Documentation. https://cassandra.apache.org/doc/latest/operating/compression.html. [En línea; consultado el 18-febrero-2021].

[ASALE and RAE, 2021] ASALE, R. and RAE (2021). Comprimir | Diccionario de la lengua española. https://dle.rae.es/comprimir. [En línea; consultado el 13-febrero-2021].

[Budhrani, 2019] Budhrani, D. (2019). How data compression works: Exploring LZ77. https://towardsdatascience.com/how-data-compression-works-exploring-lz77-3a2c2e06c097. [En línea; consultado el 14-febrero-2021].

[Codd, 2002] Codd, E. F. (2002). A relational model of data for large shared data banks. In Broy, M. and Denert, E., editors, *Software Pioneers: Contributions to Software Engineering*, pages 263–294. Springer Berlin Heidelberg, Berlin, Heidelberg. [En línea; consultado el 07-marzo-2021].

[Collet, 2016] Collet, Y. (2016). Facebook/zstd. https://github.com/facebook/zstd. [En línea; consultado el 18-febrero-2021].

[Collet, 2020] Collet, Y. (2020). Lz4/lz4. https://github.com/lz4/lz4. [En línea; consultado el 18-febrero-2021].

[DataFlair, 2018] DataFlair (2018). 5 Important Cassandra Features That You Must Know. https://data-flair.training/blogs/cassandra-features/. [En línea; consultado el 07-mayo-2021].

[DataStax, 2021] DataStax (2021). How is the consistency level configured? | Apache Cassandra 3.0. https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlConfigConsistency.html. [En línea; consultado el 07-mayo-2021].

[Dave, 2012] Dave, M. (2012). SQL and NoSQL Databases. *International Journal of Advanced Research in Computer Science and Software Engineering*. [En línea; consultado el 08-marzo-2021].

[Deutsch and Katz, 1996] Deutsch, P. and Katz, P. (1996). DEFLATE Compressed Data Format Specification version 1.3. Technical Report RFC1951, RFC Editor. [En línea; consultado el 20-febrero-2021].

[Dipperstein, 2019] Dipperstein, M. (2019). LZSS (LZ77) Discussion and Implementation. https://michaeldipperstein.github.io/lzss.html. [En línea; consultado el 15-febrero-2021].

[Diputación Foral, 2021] Diputación Foral, G. (2021). Documentación y normativa - Ogasuna. https://www.gipuzkoa.eus/es/web/ogasuna/ticketbai/documentacion-y-normativa. [En línea; consultado el 22-marzo-2021].

[Gailly and Adler, 2002] Gailly, J.-L. and Adler, M. (2002). Zlib home site. http://zlib.net/apps.html. [En línea; consultado el 18-febrero-2021].

[Gally and Adler, 2002] Gally, J.-L. and Adler, M. (2002). An Explanation of the 'Deflate' Algorithm. https://zlib.net/feldspar.html. [En línea; consultado el 18-febrero-2021].

[GeeksforGeeks, 2012] GeeksforGeeks (2012). Huffman Coding | Greedy Algo-3. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/. [En línea; consultado el 21-febrero-2021].

[Ghemawat et al., 2003] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 20–43, Bolton Landing, NY. [En línea; consultado el 06-junio-2021].

[Gobierno Vasco, 2019] Gobierno Vasco (2019). Programa TicketBAI - Sistema Tributario y Financiero - Gobierno Vasco - Euskadi.eus. https://www.euskadi.eus/ticketbai/. [En línea; consultado el 03-junio-2021].

[Gobierno Vasco, 2020] Gobierno Vasco (2020). Programa ticketbai - especifiaciones técnicas. https://www.euskadi.eus/contenidos/informacion/ticketbai/es_14815/adjuntos/TicketBAI_Especificaciones_v_1_1.pdf. [En línea; consultado el 06-junio-2021].

[Google, 2021] Google, G. (2021). Google/snappy. https://github.com/google/snappy. [En línea; consultado el 16-febrero-2021].

[Gupta et al., 2017] Gupta, A., Bansal, A., and Khanduja, V. (2017). Modern lossless compression techniques: Review, comparison and analysis. [En línea; consultado el 10-febrero-2021].

[Guru99, 2021] Guru99 (2021). Performance Testing Tutorial: What is, Types, Metrics & Example. https://www.guru99.com/performance-testing.html. [En línea; consultado el 12-mayo-2021].

[Hosseini, 2012] Hosseini, M. (2012). *A Survey of Data Compression Algorithms and Their Applications*. [En línea; consultado el 10-febrero-2021].

[Hosting Data, 2020] Hosting Data (2020). NoSQL Databases List by Hosting Data - Updated 2021. https://hostingdata.co.uk/nosql-database/. [En línea; consultado el 06-junio-2021].

[loup Gailly and Adler, 2018] loup Gailly, J. and Adler, M. (2018). GNU Gzip. https://www.gnu.org/software/gzip/manual/gzip.html. [En línea; consultado el 13-febrero-2021].

[Media Wiki, 2020] Media Wiki (2020). History of lossless data compression algorithm. https://ethw.org/History_of_Lossless_Data_Compression_Algorithms. [En línea; consultado el 06-junio-2021].

[MongoDB, 2021a] MongoDB (2021a). Advantages of MongoDB. https://www.mongodb.com/advantages-of-mongodb. [En línea; consultado el 11-abril-2021].

[MongoDB, 2021b] MongoDB (2021b). Documents — MongoDB Manual. https://docs.mongodb.com/manual/core/document/#:~:text=Document%20Size%20Limit,MongoDB%20provides%20the%20GridFS%20API. [En línea; consultado el 10-mayo-2021].

[Palaniappan and Latifi, 2007] Palaniappan, V. and Latifi, S. (2007). Lossy text compression techniques. In Akhgar, B., editor, *ICCS 2007*, pages 205–210, London. Springer London. [En línea; consultado el 23-febrero-2021].

[Parlamento Europeo, 2014] Parlamento Europeo (2014). Reglamento (UE) n ° 910/2014 del Parlamento Europeo y del Consejo, de 23 de julio de 2014 , relativo a la identificación electrónica y los servicios de confianza para las transacciones electrónicas en el mercado interior y por la que se deroga la Directiva 1999/93/CE. [En línea; consultado el 11-abril-2021].

[Pavlov, 2011] Pavlov, I. (2011). 7-Zip / Discussion / Open Discussion: LZMA vs. LZMA2. https://sourceforge.net/p/sevenzip/discussion/45797/thread/2f6085ba/. [En línea; consultado el 24-febrero-2021].

[Schaefer, 2021] Schaefer, L. (2021). NoSQL vs SQL Databases. https://www.mongodb.com/nosql-explained/nosql-vs-sql. [En línea; consultado el 04-marzo-2021].

[W3C, 2013] W3C (2013). XML Signature Syntax and Processing Version 1.1. https://www.w3.org/TR/xmldsig-core1/. [En línea; consultado el 11-abril-2021].

[W3Schools, 2021] W3Schools (2021). SQL Introduction. https://www.w3schools.com/sql/sql_intro.asp. [En línea; consultado el 07-marzo-2021].

[Wallace, 1992] Wallace, G. K. (1992). The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv. [En línea; consultado el 18-febrero-2021].

[Wikipedia, 2021] Wikipedia (2021). Lempel–Ziv–Markov chain algorithm. *Wikipedia*. [En línea; consultado el 24-febrero-2021].