UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Sistema de recomendaciones automáticas de soluciones basado en el ámbito de la gestión de incidencias

Autor:
Javier Val Barbeira

Directores:
Purificación Cariñena Amigo
Daniel Ríos Val

Grao en Enxeñaría Informática

Febreiro 2020

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñaría da Universidade de Santiago de Compostela para a obtención do Grao en Enxeñaría Informática

Dª. Purificación Cariñena Amigo, Profesora do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, e **D. Daniel Ríos Val**, Business Applications Engineer, en ES Field Delivery Spain S.L.,

INFORMAN:

Que a presente memoria, titulada *Sistema de recomendacións automáticas de solución baseado no ámbito da xestión de incidencias*, presentada por **D. Javier Val Barbeira** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo nosa dirección no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a 3 Febreiro de 2020:

A directora, O codirector, O alumno,

Purificación Cariñena Amigo Daniel Ríos Val Javier Val Barbeira

Agradecimientos

A mi mujer, la Dra. Gemma Eibes González. Sin su apoyo, ánimo y paciencia infinita, habría sido imposible finalizar esta titulación.

A mis hijos, Xabi y Antón, por disculpar las ausencias cuando tocaba desaparecer en épocas de exámenes.

A mis compañeros de clase, por 'adoptarme' con sorna, pero con cariño. Es de buena gente ayudar a las personas mayores.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación del proyecto

La mayoría de las corporaciones o grandes proyectos tienen sistemas de ayuda para que sus empleados, clientes o usuarios puedan hacer un correcto uso de los productos o servicios aportados. Estos sistemas de ayuda son atendidos por un sistema de soporte que recopila las incidencias comunicadas para que sean solucionadas por un equipo de técnicos especializados.

Cuando el sistema está correctamente organizado, se presentan varios niveles de asistencia, siendo el nivel N1 aquel que está en contacto directo con el usuario, y que recopila y soluciona la incidencia trivial, escalando a un nivel superior (N2) aquellas incidencias para las que no se ve capacitado. Es en este nivel secundario donde un técnico especialista recoge los tickets[1] asignados y los resuelve o escala en mayor o menor tiempo según su conocimiento. En casi todos estos sistemas, se recomienda al técnico que detalle la solución aplicada para potenciales auditorías ante resultados no esperados o para ser replicada posteriormente.

Muchas veces, los técnicos más noveles son incapaces de resolver ciertas incidencias usuales debido a la falta de experiencia en determinadas áreas, haciéndose necesario el consejo o intervención de un técnico senior capacitado para resolver la incidencia. Esto desemboca en una dilatación de los tiempos de resolución de incidencias, que pueden llegar a perjudicar los SLA [2] del centro de soporte.

Lo que buscamos en este proyecto es recopilar esa base de conocimiento de

[1]Un ticket hace referencia a la anotación de una incidencia en un sistema gestor de incidencias. Un ticket presenta dos estados, abierto (o en proceso) y cerrado (resuelto). El ticket permanecerá abierto durante todo el seguimiento de la incidencia hasta que se haya resuelto la incidencia

[2]Acuerdo de nivel de servicio: contrato que describe el nivel de servicio que un cliente espera de su proveedor

tickets resueltos y ponerla a disposición de todos los técnicos, de manera ágil, para que sea aplicable instantáneamente y reduzca los tiempos de resolución de incidencias, desde que se notifican, hasta que son cerradas por alguno de los niveles del servicio de soporte.

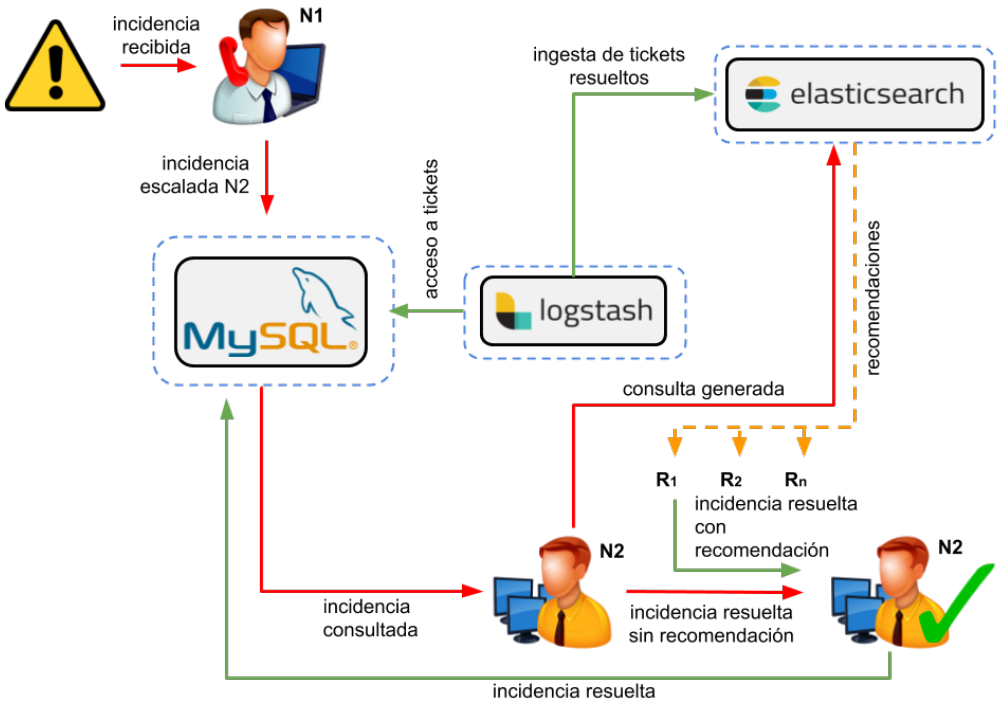


Figura 1.1: Representación del Recomendador de Soluciones

En la imagen podemos observar como ante una incidencia, el técnico de primer nivel N1 toma nota de esta mediante un gestor de incidencias. El gestor almacena dicha incidencia abierta en una base de datos MySQL[1], a la espera de un técnico de nivel superior, capacitado para su resolución. Esta resolución se puede producir en más o menos tiempo dependiendo de la experiencia del técnico asignado. Las lineas rojas representarían una incidencia abierta (no solucionada). Vemos como nuestro sistema recomendador recoge las incidencias solucionadas y las pone a disposición del técnico de nivel N2 como recomendaciones de solución (lineas naranjas) cuando el técnico consulta sus incidencias abiertas. Una vez resuelta, la incidencia vuelve a la base de datos para ser archivada.

1.2. Objetivos generales

El objetivo principal de este trabajo es implementar un sistema de recomendación capaz de complementar el proceso de resolución de incidencias, mediante una búsqueda de resoluciones ya aplicadas con un mayor nivel de coincidencia en el título del ticket. Este sistema debe ser adaptable a distintos clientes, sin un coste elevado.

Para ello, será necesario abordar unos objetivos previos:

- Poner en funcionamiento un ecosistema Elasticsearch encargado de recopilar, almacenar, clasificar y ofrecer información sobre los tickets resueltos. Este ecosistema se compone de las herramientas Elasticsearch[2], logstash[3] y Kibana[4], que determinarán también sus objetivos en el proyecto. A este conjunto de herramientas se le conoce como stack ELK [3]

- Diseñar un sistema de extracción, transformación y carga de los ticket resueltos de la base de datos del cliente. Así como el diseño de un analizador y clasificador de los datos facilitados a la hora de almacenar estos en el índice de Elasticsearch. Para determinar la mejora en el proceso de solución de incidencias será necesario la evaluación y representación de los datos mediante un cuadro de mandos realizado en Kibana.

- Generación y configuración de un sistema básico de gestión de tickets, para la realización de pruebas y validación del correcto funcionamiento del stack ELK. Este gestor se realizará sobre una base de datos relacional MySQL, utilizando Spring Boot MVC[5] y Thymeleaf[6] sobre un patrón Modelo Vista Controlador[7], utilizando las complementos de Spring Boot JPA[8] Hibernate[9] y Spring Security[10].

Cuando grandes empresas optan a pliegos, concursos públicos o licitaciones, compiten en igualdad de condiciones, ya que la mayoría de ellas pueden dar respuesta de manera eficaz, a las solicitudes propuestas en los pliegos. La mayoría de dichas ofertas presentan un apartado que premia las características o mejoras que puede aportar una empresa aspirante al proyecto. De esta manera se gestó este sistema, como una característica diferenciadora e innovadora, que aportaría un valor añadido en una competición o concurso.

[3]ELK son las siglas identificativas para tres proyectos open source: Elasticsearch, Logstash y Kibana. Elasticsearch es un motor de búsqueda y analítica. Logstash es un pipeline de procesamiento de datos del lado del servidor que ingesta datos de una multitud de fuentes simultáneamente, los transforma y luego los envía a Elasticsearch. Kibana permite a los usuarios visualizar los datos en cuadros y gráficos con Elasticsearch.

1.3. Detalle técnico

1.3.1. Tecnologías usadas

La consecución de estos objetivos ha sido posible utilizando las siguientes tecnologías, aplicaciones y lenguajes de programación::

- **Docker:** [11] Docker es una plataforma de software que permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema y código.

- **Elasticsearch:** Es un motor de analítica y análisis distribuido, open source, para todos los tipos de datos, incluidos textuales, numéricos, geoespaciales, estructurados y desestructurados. Elasticsearch está desarrollado en Apache Lucene y es el componente principal del ELK Stack, un conjunto de herramientas open source para la ingesta, el enriquecimiento, el almacenamiento, el análisis y la visualización de datos.

- **Logstash:** Es una herramienta pensada originalmente para recolectar y analizar logs. En el proyecto se utilizará su versatilidad y los plugins disponibles para recolectar información de la base de datos del cliente y añadirlos a un índice. Llamaremos a este proceso ingesta de datos.

- **Kibana:** Es un panel de visualización de datos de código abierto para Elasticsearch. Proporciona capacidades de visualización sobre el contenido indexado en un clúster Elasticsearch.

- **Json:** [12] Es un formato basado en texto estándar para representar datos estructurados. Es comúnmente utilizado para transmitir datos en aplicaciones.

- **Hibernate:** Es una herramienta que facilita el mapeo de atributos en una base de datos tradicional, y el modelo de objetos de un aplicación mediante anotaciones en los beans de las entidades que permiten establecer estas relaciones.

- **Thymeleaf:** Motor de plantillas HTML (+XML), implementado sobre Java, sencillo de usar y totalmente integrado con Spring y el patrón MVC (modelo vista controlador).

- **MySQL:** Sistema de gestión de bases de datos relacional. En nuestro sistema también utilizaremos su programa de administración MySQL Workbench[13], herramienta visual que integra desarrollo de software, administración, gestión, diseño y mantenimiento de bases de datos.

- **Java:** [14] lenguaje de programación independiente de la plataforma, con el que podemos realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general.

- **Maven:** [15] Herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida, como por ejemplo en todas las fases de compilación, empaquetado y distribución de librerías, para que puedan ser utilizadas por otros desarrolladores y equipos de trabajo.

- **Spring Boot:** [16] Framework que nace con la finalidad de simplificar el desarrollo de aplicaciones basadas en el framework Spring Core. En el sistema utilizaremos Spring Tool Suite 4, entorno de desarrollo basado en eclipse, customizado para implementar aplicaciones en Spring.

- **YAML:** [17] Es un formato de serialización de datos legible por humanos. En nuestro sistema lo utilizaremos para coordinar los múltiples contenedores docker, y que estos se comuniquen entre sí.

- **Docker Compose:** [18] Herramienta para definir y desplegar múltiples aplicaciones sobre contenedores docker. Mediante la utilización de un archivo de configuración yaml, se determinarán los servicios a desplegar.

1.3.2. Metodología de desarrollo

Se ha optado por un modelo incremental[19], combinando elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. El modelo incremental avanza sumando funcionalidades a la aplicación a desarrollar, aplica secuencias lineales de forma escalonada, mientras progresa en el tiempo de desarrollo. Cada secuencia lineal produce un incremento en el software. En el capítulo 2 se desarrollará y argumentará esta decisión.

1.4. Estructura de la memoria

Este documento está divido en capítulos en los que se tratan los siguientes temas:

- En este **Capítulo 1** se hace una breve introducción al proyecto en la que se comentan la motivación del mismo, objetivos y un apartado de detalle técnico.

- En el **Capítulo 2** se documenta todo lo relacionado con la gestión del proyecto: planificación temporal, estimación de costes y gestión de riesgos.

- En el **Capítulo 3** se procederá con la fase de análisis funcional del proyecto, determinando requisitos y casos de uso.
- En el **Capítulo 4** se documentan las tecnologías utilizadas, lenguajes e IDEs implicados, tanto en el desarrollo del sistema, como en la redacción de la documentación.
- En el **Capítulo 5** se describe el proceso de diseño e implementación del sistema recomendador, así como su integración en el gestor de incidencias. Todo ello virtualizado en contenedores Docker.
- En el **Capítulo 6** se comentan las conclusiones del trabajo, con la mirada puesta en posibles mejoras a desarrollar en un trabajo futuro.

Se entregarán a modo de apéndices:

- Los diagramas de Gantt de cada incremento.
- El manual de despliegue de la aplicación y el ciclo de una incidencia.
- Los diagramas de clases del gestor de incidencias, su árbol de clases y el índice generado en Elasticsearch.

Capítulo 2

Gestión del proyecto

2.1. Alcance del proyecto

El objetivo es implementar un sistema de ingesta de datos configurable y adaptado a la base de datos del cliente, un sistema capacitado para recoger información de tickets de incidencias resueltas y almacenar estas soluciones en un índice de Elasticsearch.

Este índice de soluciones debe ser accesible mediante consulta en tiempo real desde el gestor de tickets del cliente o aplicación equivalente, para ofrecer soluciones aplicables, desde el histórico de tickets resueltos, a las nuevas incidencias que se plantean.

También se planifica el desarrollo de un panel de control que permita al cliente validar la implantación y uso del sistema recomendador de soluciones en su gestor de incidencias, así como otras métricas de su interés.

Como módulo de pruebas, se plantea entregar un gestor básico de incidencias sobre una base de datos relacional, que integrará y validará todo el funcionamiento del sistema recomendador.

2.1.1. Entregables del proyecto

Una vez finalizado el proyecto, se entregará lo siguiente:

- **E1.** Un gestor de incidencias básico, con roles y usuarios, capaz de un proceso CRUD[1] sobre las incidencias. También se entregará un conjunto de

[1]CRUD es el acrónimo de 'Crear, Leer, Actualizar y Borrar' (del original en inglés: Create, Read, Update and Delete), que se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

incidencias de prueba, almacenadas en la base de datos relacional.

- **E2.** Un conjunto de scripts que permitan la virtualización de todo el sistema en contenedores docker.
- **E3.** Un conjunto de scripts que permitan generar, dentro de la virtualización anterior, un stack ELK, para la ingesta, indexación y visualización de datos.
- **E4.** Incorporación del sistema recomendador al gestor de incidencias.
- **E5.** Un panel de control en el que se muestra la implantación del sistema de recomendación mediante una serie de gráficas y estadísticas.
- **E6.** Memoria del proyecto y manual de despliegue del sistema.

2.1.2. Criterios de aceptación

Se dará por finalizado el proyecto cuando se consigan los siguientes hitos:

- **CA1.** Se puede lanzar una ingesta de datos de incidencias recuperadas de la base de datos del cliente, que pasan a ser indexadas en un índice determinado de Elasticsearch.
- **CA2.** Ante la apertura de una nueva incidencia, el técnico asignado recibe una serie de recomendaciones de solución recuperadas del índice Elasticsearch.
- **CA3.** Ante el cierre de una nueva incidencia, esta es recuperada de la base de datos del cliente, transformada y almacenada en el índice de Elasticsearch.
- **CA4.** Disponemos de un panel de control en el que se muestra la implantación del sistema de recomendación mediante una serie de gráficas y estadísticas.

2.1.3. Exclusiones del proyecto

Quedan fuera del alcance de este proyecto la generación de un gestor de tickets más allá de su uso para la validación del correcto funcionamiento del sistema, así como el diseño de la base de datos de tickets.

Han sido desarrollados, gestor y base de datos, de una manera funcional, para facilitar la visualización del ciclo completo de un ticket. No han sido desarrollados buscando la robustez del sistema gestor, ni su completa funcionalidad.

2.2. Planificación del proyecto

Se determina un trabajo diario aproximado de 5 horas, de lunes a viernes, pudiéndose variar por causas justificadas la carga de un día, siendo recuperado en la misma semana el tiempo sacrificado. Esto implica unas 25 horas semanales de trabajo aproximadamente, estimando un total de 17 semanas para la conclusión del proyecto. En total implicaremos unas 425 horas que incluyen trabajo autónomo, reuniones informativas y de control, así como tutorías, evaluación de los avances y entregas.

2.2.1. Estructura de Descomposición del Trabajo (EDT)

Mediante esta representación estructural (EDT)[2] procedemos a visualizar la planificación del proyecto, definiendo y organizando el alcance total aprobado. Observamos los principales conjuntos de actividades que determinan el trabajo a desarrollar.

[2]Consiste en la descomposición jerárquica del trabajo a ser ejecutado por el equipo de proyecto, para cumplir con los objetivos de éste y crear los entregables requeridos, donde cada nivel descendente de la EDT representa una definición con un detalle incrementado del trabajo del proyecto.

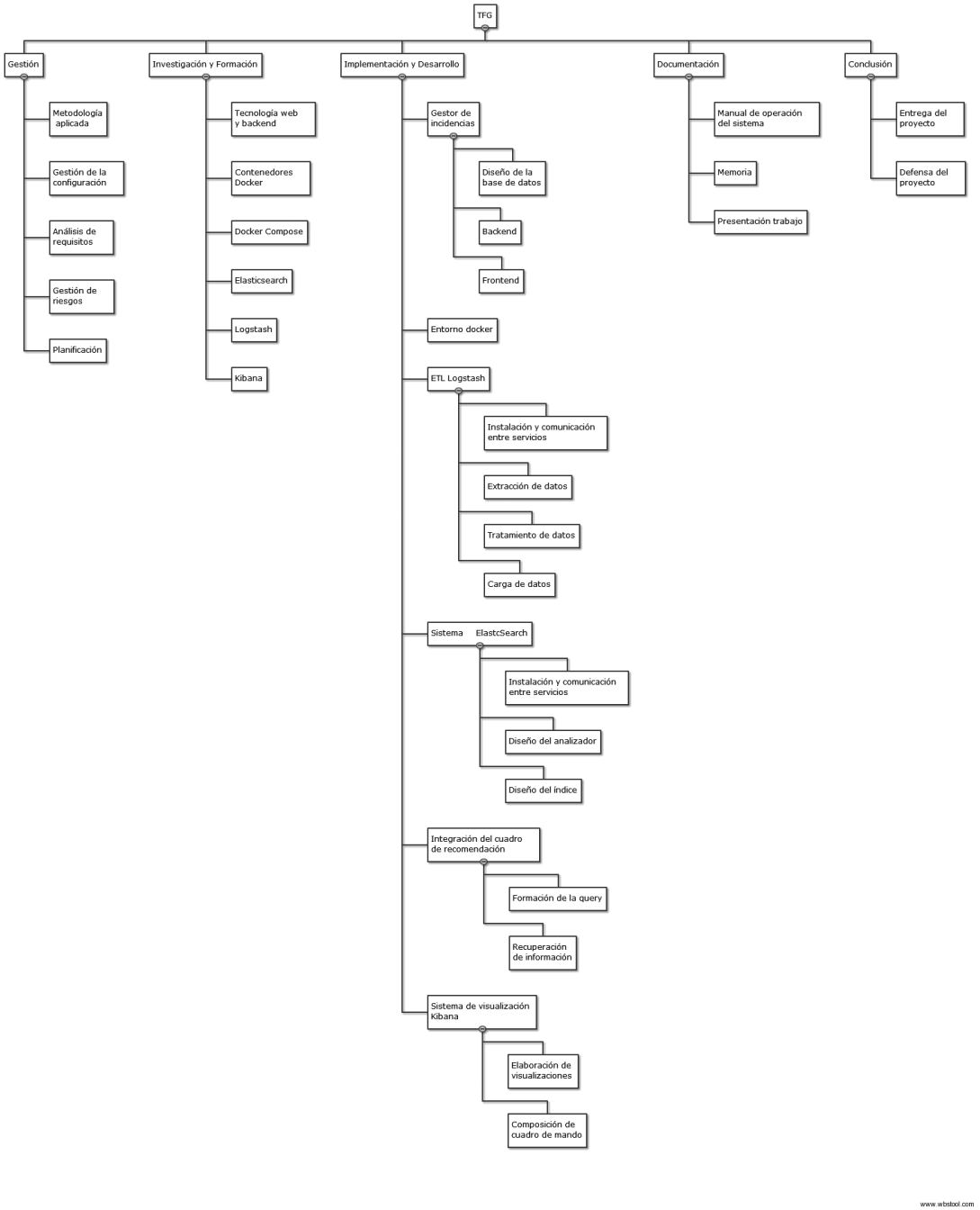


Figura 2.1: Estructura de descomposición de trabajo

2.2.2. Definición de actividades

De manera más detallada podemos describir los procesos a desarrollar en cada rama:

1. **Gestión del proyecto** estimación temporal: 2 semanas.

   - **Estudio y selección de metodología.**
     Análisis de diferentes metodologías de desarrollo y selección de la más apropiada para este proyecto.
   - **Planificación.**
     Distribución de las diferentes tareas en los incrementos determinados.
   - **Gestión de la configuración.**
     Elaboración del plan de control de cambios y gestión de versiones.
   - **Análisis de requisitos.**
     Proceso de estudio de necesidades del usuario que deberán ser resueltas por el proyecto.
   - **Gestión de riesgos.**
     Análisis de los riesgos potencialmente presentes durante el desarrollo y establecimiento de contramedidas.

2. **Formación e investigación** estimación temporal: 3 semanas.

   - **Formación en tecnologías web, frontend y backend.**
     Adquirir conocimientos sobre Spring MVC y Thymeleaf, Spring Security, Spring Data JPA e Hibernate para la realización de la estructura básica de un gestor de incidencias.
   - **Virtualización de aplicaciones en contenedores Docker**.
     Desarrollar las habilidades necesarias para el despliegue de las aplicaciones sobre contenedores docker que faciliten la replicación de los servicios en cualquier máquina.
   - **Trabajo con aplicaciones multicontenedor.**
     Formación en gestión de múltiples contenedores intercomunicados mediante docker Compose.
   - **Puesta en marcha de un stack ELK.**
     Formación para la puesta en marcha de un stack ELK (Elasticsearch, Logstash y Kibana) virtualizado en contenedores docker, intercomunicados y operativos.
   - **Ingesta de datos.**
     Formación en el diseño de los scripts necesarios para la configuración del pipeline de Logstash, así como la creación de índices y analizadores en Elasticsearch.
   - **Panel de control.**
     Conocimiento necesario para la elaboración de un panel de control operativo en Kibana.

3. **Implementación A** estimación temporal: 3 semanas.

    - Se procederá a desarrollar en local un gestor de incidencias multiusuario, con capacidad de administración de tickets (abrir nuevos tickets, actualizar, cerrar y borrar tickets). También se implementará una base de datos de tickets, que servirá de backend a dicho gestor.

4. **Implementación B1** estimación temporal: 2 semanas.

    - Se migrará el sistema implementado en local a un entorno virtualizado en contenedores Docker, realizando las adaptaciones pertinentes. Se gestará una red de contenedores, creando los necesarios para interconectar el stack ELK a la base de datos y posteriormente al gestor de incidencias.

5. **Implementación B2** estimación temporal: 2 semanas.

    - Configuración y operatividad del entorno ELK, puesta en funcionamiento del sistema de recuperación e ingesta de datos, creación del analizador, índice y configuración de la consulta de recomendación.

6. **Implementación C** estimación temporal: 1 semanas.

    - Elaboración de un panel de control para el seguimiento de la implantación del sistema.

7. **Documentación** estimación temporal: 4 semanas.

    - Elaboración de la memoria.
    - Diseño de la presentación del proyecto.
    - Documentación de la aplicación.

2.2.3. Planificación temporal (inicial)

Se arranca el proyecto el lunes 16 de septiembre de 2019, determinando el esfuerzo semanal dedicado en unas 25 horas aproximadamente, a lo largo de 17 semanas, estimando la finalización del proyecto el 12 de enero de 2020

Se pasará por 5 etapas coincidentes con las ramas del EDT.

1. **Gestión del proyecto** *50 horas.*
   Se invertirá el tiempo en el estudio y selección de metodología más apropiada para este proyecto, distribución de las diferentes tareas, gestión de la configuración, análisis de requisitos, gestión de riesgos, etc.

2. **Formación e investigación** *75 horas.*
   Formación en tecnologías web, frontend y backend, así como en virtualización de aplicaciones en contenedores Dockers. Formación para la puesta en marcha de un stack ELK.

3. **Implementación y desarrollo** *200 horas.*
   Se procederá a desarrollar en local una base de datos de tickets, que servirá de backend a un gestor de incidencias multiusuario, con capacidad de administración de tickets (abrir nuevos tickets, actualizar, cerrar y borrar tickets). Posteriormente se migrará el sistema implementado en local a entorno virtualizado en contenedores, realizando las adaptaciones pertinentes. Sobre esta red de contenedores se crearán los necesarios para interconectar el stack ELK a la base de datos y posteriormente al gestor de incidencias. Se procederá con la configuración y operatividad del entorno ELK, puesta en funcionamiento del sistema de recuperación e ingesta de datos, creación del analizador, índice y configuración de la consulta de recomendación. Finalmente se elaborará un panel de control para el seguimiento de la implantación del sistema.

4. **Documentación** *100 horas.*
   Se procederá con la elaboración de la memoria, diseño de la presentación, así como la elaboración de la documentación necesaria para ejecutar la aplicación y los procesos de carga.

5. **Entrega**
   Conclusión del proyecto, entrega de la documentación y de la aplicación desarrollada.

Ciclo de vida incremental

A la hora de seleccionar el modelo de ciclo de vida, se tuvo en cuenta las características del proyecto, el equipo de diseño y desarrollo (unipersonal), los requisitos definidos y la entrega final determinada en el alcance. Tras partir de una idea inicial de desarrollo utilizando metodologías ágiles como Scrum, se llegó a la conclusión de que esta metodología de trabajo es útil para equipos de desarrollo medios, pero que no aportaría mejoras en un desarrollo unipersonal, en el que se tendría que asumir varios roles en una misma persona, llegando al rocambolesco punto de reuniones "conmigo mismo" para determinar las tareas a asumir en el siguiente sprint.

De esta manera se decide variar el modo de afrontar el proyecto, optando por un modelo incremental, combinando elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. El modelo incremental avanza sumando funcionalidades a la aplicación a desarrollar, aplica secuencias lineales

de forma escalonada mientras progresa en el tiempo de desarrollo. Cada secuencia lineal produce un incremento en el software.

Este modelo se centra en la entrega de un producto operativo con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

De esta manera se podría recortar el alcance del proyecto, en caso necesario, al vernos ajustados de tiempo, entregando un producto operacional, con menos funcionalidades, pero manteniendo el objetivo marcado al comienzo del proyecto.

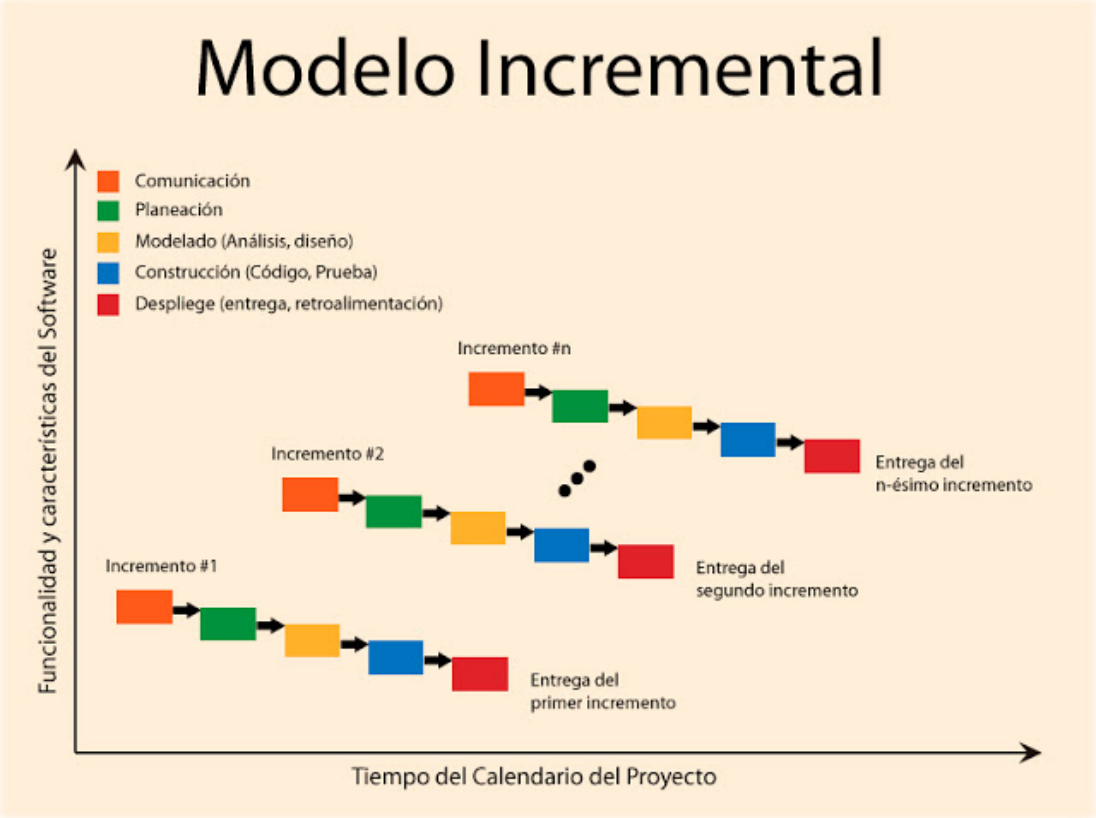


Figura 2.2: Esquema de modelo incremental

Entre las **ventajas** que puede proporcionar un modelo de este tipo se encuentran las siguientes:

- Mediante este modelo se genera software operativo de forma rápida y en etapas tempranas del ciclo de vida del software.

- Es un modelo más flexible, por lo que se reduce el coste en el cambio de alcance y requisitos.
- Es más fácil probar y depurar en una iteración más pequeña.
- Es más fácil gestionar riesgos.
- Cada iteración es un hito gestionado fácilmente.

Para el uso de este modelo se requiere capacidad para definir los incrementos y distribuir en ellos las tareas de forma proporcionada. Entre los **inconvenientes** que aparecen en el uso de este modelo podemos destacar los siguientes:

- Cada fase de una iteración es rígida y no se superponen con otras.
- Pueden surgir problemas referidos a la arquitectura del sistema porque no todos los requisitos se han reunido.
  Este era uno de los inconvenientes más amenazadores desde un principio, ya que el sistema recomendador necesitaba para su implementación un sistema de gestión de tickets que debía estar implementado y operativo.

El proyecto se desarrollará en 3 fases

1. **Fase inicial**.
   Engloba la definición de la gestión, planificación del proyecto y formación.
2. **Fase de implementación**.
   Se materializan los cuatro incrementos siguientes, que se detallan posteriormente:
   - Incremento 1: Implementación de base de datos y gestor de tickets
   - Incremento 2: Migración del sistema a contenedores (virtualización)
   - Incremento 3: Configuración y operatividad del entorno ELK
   - Incremento 4: Elaboración de un panel de control
3. **Fase de conclusión**.
   Se cierra el proyecto, elaborando la documentación necesaria, memoria y presentación.

Incremento 1: Implementación de Bases de datos y gestor de tickets

Con una duración estimada de tres semanas, en este incremento se abordará la creación de la base de datos de tickets, así como las tablas necesarias para la gestión de usuarios y roles del gestor de tickets. Durante este incremento también se desarrollará todo el gestor de incidencias, controladores, vistas y accesos a repositorios.

Se pueden consultar los gráficos Gantt de las tareas determinadas en estos incrementos en el apéndice C.

Incremento 2: Migración del sistema a contenedores (virtualización)

En este incremento, de duración estimada de 2 semanas, se procederá a migrar la base de datos y el gestor desarrollado en local a contenedores Docker, creando a su vez los contenedores necesarios para el stack ELK y la red que conectará todos ellos. Todo el sistema se virtualizará sobre una imagen de Centos 7.

Incremento 3: Configuración y operatividad del entorno ELK

Recuperando los contenedores creados sobre los que se despliega el stack de Elastic, Logstash y Kibana, procederemos a configurar la ingesta de datos, la creación del analizador y el índice. Posteriormente integraremos la consulta de búsqueda en el gestor, así como la representación de resultados.

Incremento 4: Elaboración de un panel de control

Como incremento final, con un tiempo estimado de una semana, se presenta el desarrollo de un panel de control en Kibana, que permitirá al cliente realizar un seguimiento de la implantación del sistema recomendador y su utilidad en la reducción de tiempo consumido en la resolución de cada ticket.

2.3. Gestión de la configuración

Para el correcto desarrollo del proyecto se hace necesario llevar un control de cambios, así como disponer del versionado de cada nueva entrega. Estos procesos nos permitirán asegurar la calidad de todo el software desarrollado.

2.3.1. Sistemas y herramientas de Gestión de la Configuración

El código utilizado para la creación del gestor de tickets será controlado mediante un repositorio de github[3]. Para la gestión de los archivos del proyecto se utiliza una cuenta en Google Drive con un almacenamiento extendido de 100 GB con un coste de 2€al mes. Esto se debe a que se realizan copias periódicas de backup de la imagen virtualizada de Centos 7, para evitar posibles errores de corrupción, y facilitar el trabajo remoto en múltiples ordenadores. Los archivos de uso común, como la estructura de Docker se guardarán manteniendo su estructura en Google Drive.

El servicio de almacenamiento posee un sistema de versionado automático, generando un historial de archivos con cada actualización que se realice sobre el archivo, siempre y cuando se mantenga idéntico nombre. De esta manera si es necesario recuperar un archivo anterior, simplemente se accede al histórico de almacenamiento de ese archivo en cuestión.

[3]GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git

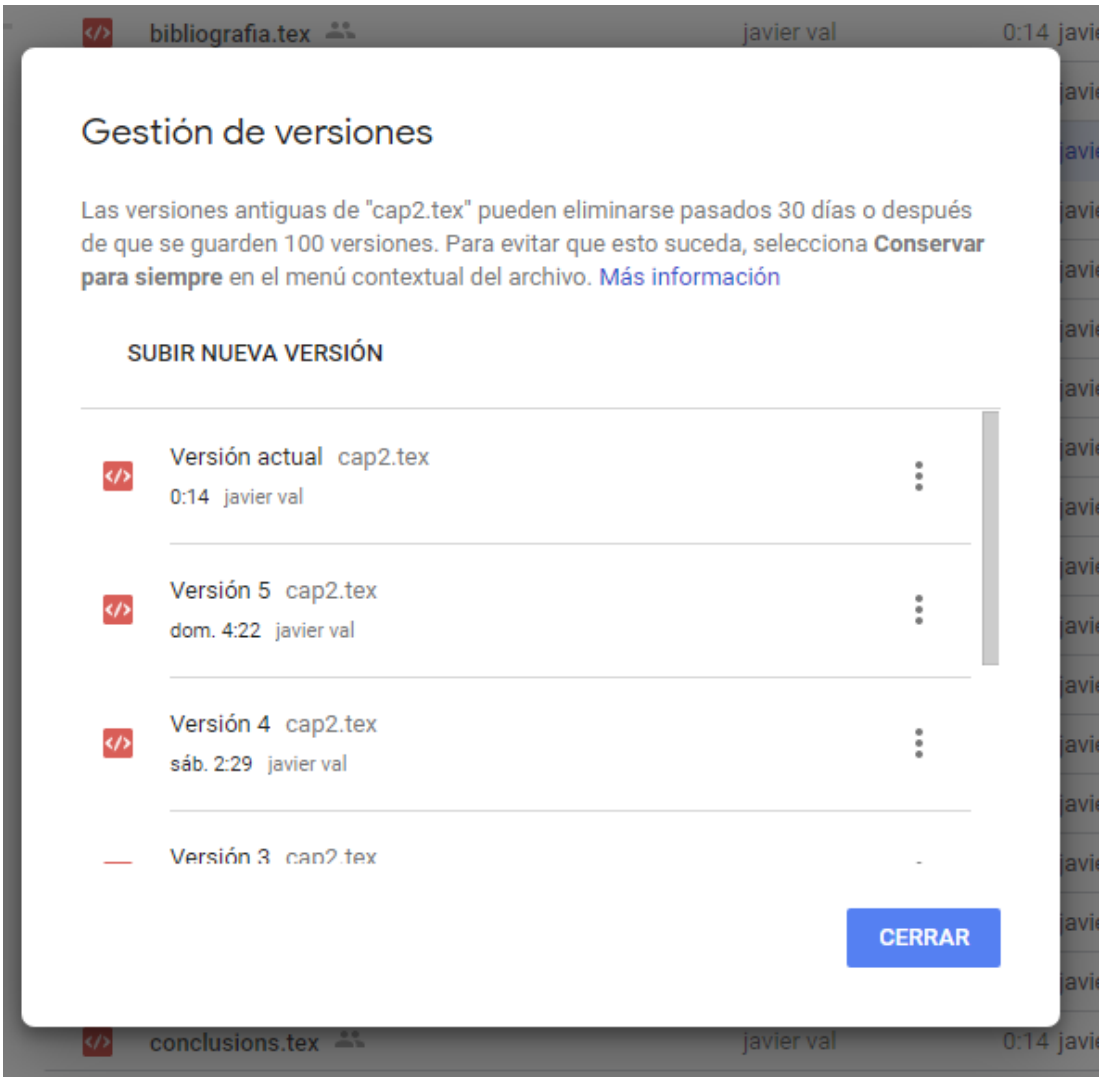


Figura 2.3: Sistema de versionado de Google Drive

2.3.2. Nomenclatura de ficheros

Todos los archivos se identificarán siguiendo el patrón definido en la siguiente imagen:

Proyecto_identificador_Fecha(yyyy_mm_dd)_nombre_versión(xx.yy)<_comentario>

Figura 2.4: Patrón de nomenclatura de archivos

Se identifica el Proyecto como Trabajo fin de grado (TFG), la fecha en formato año, mes, día, el nombre identificativo del archivo, su versionado en 2 dígitos

Código	Documento relativo a
BBDD	Base de datos
DKR	Docker
SPR	Spring Boot | JPA | Security
MEM	Redacción elaboración de la memoria
VID	Vídeo presentación
DTF	Gestión administrativa TFG
DGP	Gestión del proyecto
EDT	Descomposición de tareas
GNT	Gantt
PGC	Costes del proyecto
IMG	Elemento visual
BBL	Bibliografía

Tabla 2.1: Tabla de identificadores de archivos.

con 2 decimales y un comentario, no obligatorio, para aportar información adicional. También se añade un campo identificador de 3 a 4 caracteres que permite determinar la naturaleza del archivo. En la tabla 2.1 detallamos su clasificación.

2.3.3. Estructura del directorio del proyecto

SE puede observar en la siguiente imagen una captura de la estructura de directorios almacenada en el repositorio de Google Drive.

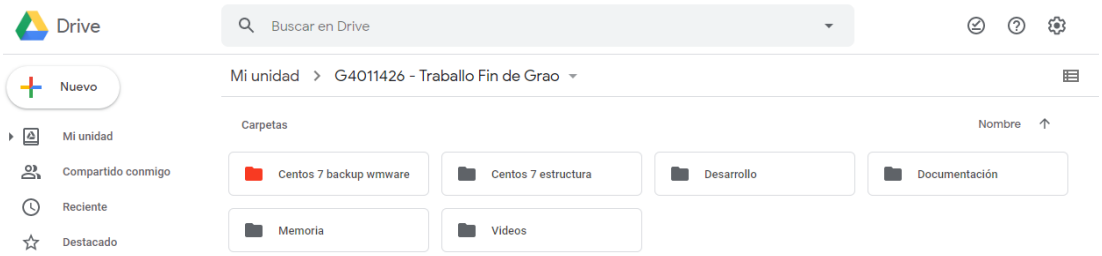


Figura 2.5: Estructura de directorios

Dentro de la carpeta 'Desarrollo', se encuentran, por ejemplo, las carpetas relativas a los elementos implementados. Si se visualiza la carpeta correspondiente a bases de datos, se puede apreciar la nomenclatura de ficheros establecida.

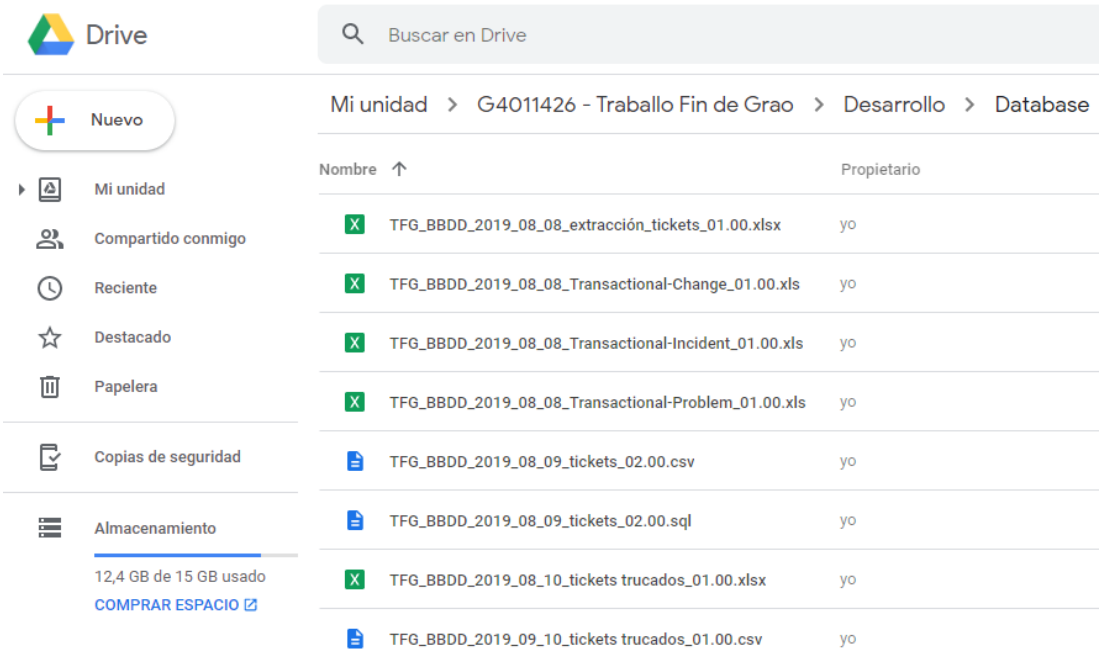


Figura 2.6: Nomenclatura definida

2.3.4. Comunicación entre miembros del equipo

Se mantendrán reuniones semanales con el cotutor, manteniendo informada a la tutora de la evolución mensual. Las reuniones semanales serán mayoritariamente presenciales, ya que el cotutor comparte espacio de trabajo, siendo necesario el contacto vía Skype, cuando este se encuentre en itinerancia por motivos laborales. Para la comunicación con el tutor, se emplearán correo electrónico mayoritariamente, vídeos demostrativos de la evolución del proyecto y reuniones presenciales

en el despacho.

2.3.5. Gestión de cambios

Dadas las características del proyecto, siendo un equipo de desarrollo unipersonal, con un contacto semanal con el cotutor, se decidió que el uso de plantillas para la gestión de cambios eliminaría la agilidad, sobrecargando de documentación el proyecto. Se tiene presente la necesidad y utilidad de estas en proyectos de mayor envergadura, pero para nuestro desarrollo ágil se discutirán las solicitudes de cambio y su prioridad durante las reuniones semanales, valorando si serán necesarios o no dichos cambios.

2.4. Estimación de costes

Para determinar el coste del proyecto, se analiza cada tipo de coste presente durante el desarrollo.

2.4.1. Costes de recursos humanos

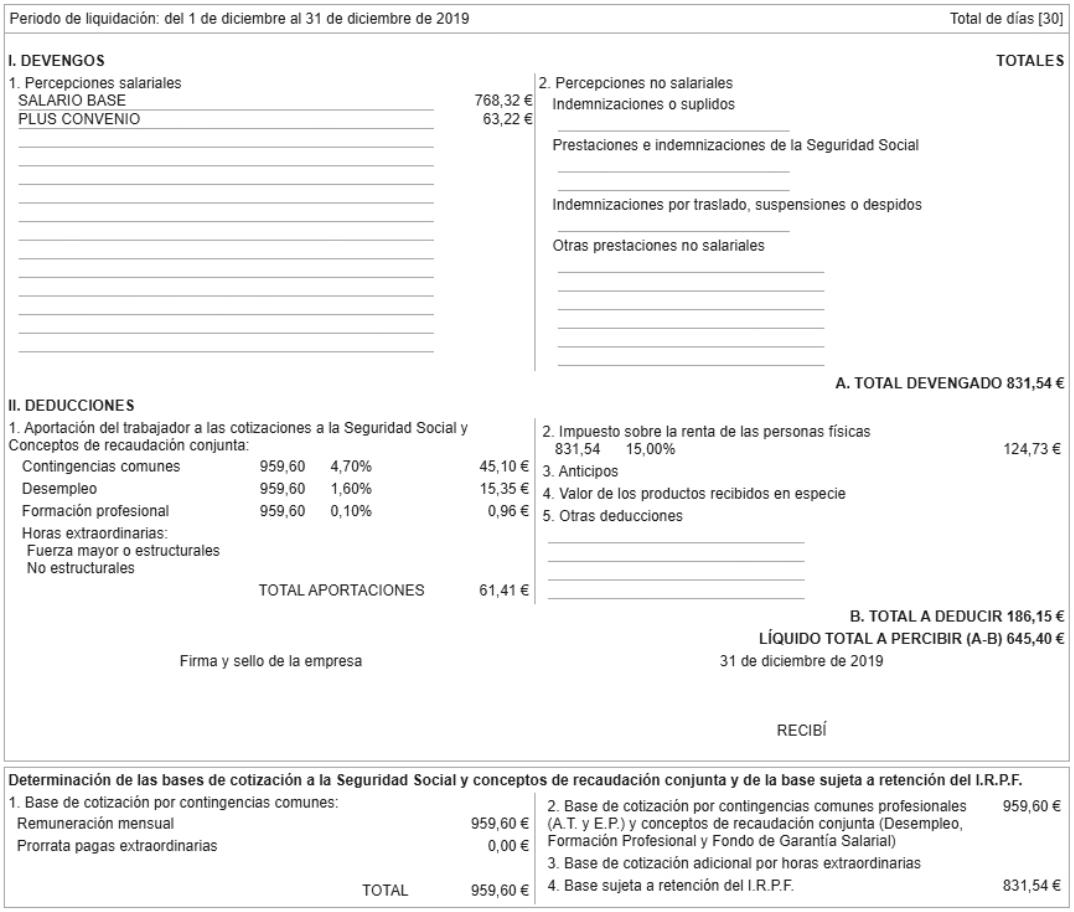
Para determinar el coste de recursos humanos, acudimos al convenio colectivo aplicable a las empresas de consultoría (XVII Convenio colectivo estatal de empresas de consultoría, y estudios de mercados y de la opinión pública) publicado en el BOE el martes 6 de marzo de 2018.[4] Según este, un desarrollador correspondiente al área 3 grupo D nivel I percibirá de salario base **16.548,44€** plus convenio de **1.167,21€** lo que derivaría en un salario total de **17.715,65€**

La jornada laboral será de **25 horas semanales**, lo cual aplica un porcentaje del 62,5 % del sueldo marcado por el convenio. Aplicamos una mejora para compensar la carga irregular de horas, ofreciendo un 65 % del sueldo efectivo, gestando una mejora sobre el convenio colectivo que deberá ser aceptada por el empleado al firmar el contrato por obra.

La duración del contrato será de 6 meses, para adaptarse a la duración del proyecto, quedando unos 15 días desde la finalización del proyecto, hasta la finalización del contrato, que serán compensados como vacaciones retribuidas.

Se genera una nómina de ejemplo para determinar los costes para la empresa:

[4]https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf



Periodo de liquidación: del 1 de diciembre al 31 de diciembre de 2019 — Total de días [30]

I. DEVENGOS — **TOTALES**

1. Percepciones salariales
- SALARIO BASE 768,32 €
- PLUS CONVENIO 63,22 €

2. Percepciones no salariales
- Indemnizaciones o suplidos
- Prestaciones e indemnizaciones de la Seguridad Social
- Indemnizaciones por traslado, suspensiones o despidos
- Otras prestaciones no salariales

A. TOTAL DEVENGADO 831,54 €

II. DEDUCCIONES

1. Aportación del trabajador a las cotizaciones a la Seguridad Social y Conceptos de recaudación conjunta:

Contingencias comunes	959,60	4,70%	45,10 €
Desempleo	959,60	1,60%	15,35 €
Formación profesional	959,60	0,10%	0,96 €
Horas extraordinarias: Fuerza mayor o estructurales			
Horas extraordinarias: No estructurales			
	TOTAL APORTACIONES		61,41 €

2. Impuesto sobre la renta de las personas físicas: 831,54 — 15,00% — 124,73 €
3. Anticipos
4. Valor de los productos recibidos en especie
5. Otras deducciones

B. TOTAL A DEDUCIR 186,15 €

LÍQUIDO TOTAL A PERCIBIR (A-B) 645,40 €

Firma y sello de la empresa — 31 de diciembre de 2019

RECIBÍ

Determinación de las bases de cotización a la Seguridad Social y conceptos de recaudación conjunta y de la base sujeta a retención del I.R.P.F.

1. Base de cotización por contingencias comunes:
- Remuneración mensual 959,60 €
- Prorrata pagas extraordinarias 0,00 €

TOTAL 959,60 €

2. Base de cotización por contingencias comunes profesionales (A.T. y E.P.) y conceptos de recaudación conjunta (Desempleo, Formación Profesional y Fondo de Garantía Salarial) 959,60 €
3. Base de cotización adicional por horas extraordinarias
4. Base sujeta a retención del I.R.P.F. 831,54 €

Figura 2.7: Nómina tipo mensual

Sobre la base de cotización de **959,60€** aplicaremos los porcentajes a la Seguridad Social a cargo de la empresa

- 23,6 % de contingencias comunes
- 5,5 % prestación por desempleo
- 3,5 % accidentes de trabajo / enfermedad profesional
- 0,6 % formación profesional
- 0,2 % aportación FOGASA

Lo cual genera un coste para la empresa de **831,54€+ 320,50€= 1152€** mensuales.

Al finalizar los 6 meses de contrato, debemos abonar una extra correspondiente y el equivalente a 6 días trabajados en concepto de finiquito por finalización

de contrato (12 días por año trabajado).

El coste total quedará de la siguiente manera **1152€* 6 + 768,32€+ 291,20€= 7971,52€**.

2.4.2. Costes materiales

Para el cálculo de los costes materiales de activo fijo imputamos el equipo utilizado para el desarrollo del proyecto, un ordenador portátil, con 2 años de antigüedad, un monitor externo, y un trackball. Para estos activos calculamos el coste de amortización, sin presentar costes derivados de reparaciones ni mantenimiento.

Cuando se compra un equipo, este no puede ser imputado como un coste por su totalidad en el mismo momento de la compra, ya que se le estima una vida útil de ciertos años. Se hace necesario calcular la cuota de amortización imputable anualmente, siempre sin sobrepasar un porcentaje marcado por el estado. Para los equipos informáticos, este porcentaje asciende como máximo al 25 % anual del valor de adquisición, pudiendo ser amortizado a lo largo de 8 años como máximo.[5]

Esta cuota de amortización (CA) que nos permite imputar a un ejercicio fiscal viene determinado por la fórmula:

CA = (Valor de adquisición – Valor residual) / Nº de años de vida útil

Se debe pues determinar previamente el valor residual de los activos (el valor que tendrán una vez finalice su vida útil, calculado en base a su depreciación), así como el número de años de vida útil.

Se determinan los cálculos pormenorizados para uno de los activos en modo de ejemplo. La cuota de amortización anual para el portátil es de (1000-300)/4 = 175€. Se debe de tener en cuenta que nuestro equipo se puede utilizar en otros proyectos durante el desarrollo de nuestra aplicación, así como en los 6 meses restantes del año, por lo que debemos prorratear esta cuota a lo largo de las horas de todo el año (40 horas semana por 52 semanas = 2080 horas) si nuestro proyecto tiene una estimación de 425 horas, la cuota amortizable, el coste imputable sobre el ordenador, debido al proyecto es de 35,75€.

Finalmente quedan de la siguiente manera los costes imputables de los activos empleados.

[5]https://www.agenciatributaria.es/

Activo	vida útil	valor residual	coste imputable
Portátil	4 años	300€	35,75€
Monitor	8 años	50€	3,8€
Trackball	8 años	15€	0,3€

Tabla 2.2: Vida útil, valor residual y coste imputablede los activos

Por lo tanto, el coste total de materiales sería en torno a 40€.

2.4.3. Costes de software

Para el desarrollo del software se utilizaron aplicaciones de software libre, sin coste alguno, excepto la suscripción ampliada al servicio de almacenamiento Google Drive, para los backups de las imágenes de Centos 7. El coste de la suscripción es de 2€mes, durante 6 meses, lo que implica un coste imputable de 12€.

2.4.4. Costes indirectos

Son aquellos recursos necesarios para la elaboración del proyecto, pero que no pueden ser imputables únicamente a este en su totalidad, tales como la electricidad, calefacción o agua corriente. Al ser difícilmente mensurables los consumos para cada proyecto, se suele determinar un porcentaje imputable a éstos sobre el total del coste del proyecto. A modo de ejemplo la Universidad de Santiago de Compostela asigna un 21 % del coste total a los costes indirectos. Así nuestro coste indirecto será el siguiente.

Costes indirectos = (7971,52+40+12)*0,21 = 1.685€

2.4.5. Coste total

Una vez calculados los costes materiales, indirectos y de recursos humanos, el total del coste del proyecto asciende a:

Costes totales = (7971,52+40+12+1685) = 9708,52€

2.5. Gestión de riesgos

Cuando hablamos de gestión de riesgos, se debe determinar el plan de gestión que identificará, analizará y clasificará los potenciales eventos o amenazas que

explotarán una vulnerabilidad de un activo, afectando de manera negativa al desarrollo del proyecto. Mediante este estudio se determinan un número de riesgos que se clasifican como más probables, y con mayor impacto. Sobre ellos se realizará un análisis pormenorizado que buscará determinar medidas para su prevención, o contrarrestar su impacto una vez producidos.

2.5.1. Identificación de riesgos

Se clasifican los riesgos identificados mediante la abreviatura 'RSG' seguido de una numeración de 2 cifras, generando la siguiente plantilla en la que se determinará nombre, descripción, probabilidad (materialización del riesgo analizado) e impacto (consecuencias del riesgo en el devenir del proyecto). En apartados posteriores analizaremos las acciones de prevención y contingencia.

- **RSG-01:** Imposible cumplir con la planificación del proyecto.
- **RSG-02:** Conocimiento insuficiente para finalizar etapas de implementación.
- **RSG-03:** Reducción de las horas imputables al proyecto.
- **RSG-04:** Se presentan nuevos requisitos.
- **RSG-05:** Cambios en los requisitos ya establecidos.
- **RSG-06:** La gestión de configuración no está siendo aplicada.
- **RSG-07:** Imposibilidad de reunión con el cotutor o tutor.
- **RSG-08:** Incidencias en software.
- **RSG-09:** Incidencias en hardware.
- **RSG-10:** Estimación errónea del período de una tarea.
- **RSG-11:** Corrupción en archivos del proyecto.

2.5.2. Análisis y evaluación de riesgos

Se aplica la siguiente plantilla para señalar con más detalle los riesgos anteriores, determinar su probabilidad e impacto. Posteriormente estableceremos acciones preventivas y de contingencia sobre los riesgos principales señalados en la matriz de probabilidad-impacto.

Seguiremos el siguiente modelo:

Identificador	RSG-XX
Nombre	Nombre
Descripción hline Probabilidad	Descripción Alta, media, baja
Impacto	Alto, medio, bajo

Tabla 2.3: Modelo para el análisis de riesgos

A continuación se muestran las tablas de análisis de riesgos

Identificador	RSG-01
Nombre	Imposible cumplir con la planificación del proyecto.
Descripción	Un diseño demasiado optimista o falto de experiencia no permite determinar correctamente los plazos necesarios para cada tarea, derivando en un sobre coste de tiempo necesario para finalizar el proyecto.
Probabilidad	Alta
Impacto	Alto

Tabla 2.4: Análisis riesgo RSG-01

Identificador	RSG-02
Nombre	Conocimiento insuficiente para finalizar etapas de implementación.
Descripción	Ciertas partes de la implementación no se vieron cubiertas durante la etapa de formación, haciéndose necesario retomar la formación.
Probabilidad	Alta
Impacto	Medio

Tabla 2.5: Análisis riesgo RSG-02

Identificador	RSG-03
Nombre	Reducción de las horas imputables al proyecto.
Descripción	El tiempo dedicado al proyecto no coincide con el estipulado en la planificación.
Probabilidad	Media
Impacto	Alto

Tabla 2.6: Análisis riesgo RSG-03

Identificador	RSG-04
Nombre	Se presentan nuevos requisitos.
Descripción	Nuevos requisitos marcados como relevantes son presentados durante el desarrollo del proyecto.
Probabilidad	Media
Impacto	Medio

Tabla 2.7: Análisis riesgo RSG-04

Identificador	RSG-05
Nombre	Cambios en los requisitos ya establecidos.
Descripción	Ciertos requisitos son redefinidos.
Probabilidad	Media
Impacto	Medio

Tabla 2.8: Análisis riesgo RSG-05

Identificador	RSG-06
Nombre	La gestión de configuración no está siendo aplicada.
Descripción	No aplicar la configuración establecida puede derivar en pérdida de documentos e información relevante.
Probabilidad	Media
Impacto	Medio

Tabla 2.9: Análisis riesgo RSG-06

Identificador	RSG-07
Nombre	Imposibilidad de reunión con el cotutor o tutor.
Descripción	Suspensión de reuniones de seguimiento.
Probabilidad	Baja
Impacto	Bajo

Tabla 2.10: Análisis riesgo RSG-07

Identificador	RSG-08
Nombre	Incidencias en software.
Descripción	Cierto software empleado en el desarrollo presenta incompatibilidades, o no desempeña de forma adecuada su cometido, pudiendo desembocar en pérdida de trabajo realizado.
Probabilidad	Baja
Impacto	Bajo

Tabla 2.11: Análisis riesgo RSG-08

Identificador	RSG-09
Nombre	Incidencias en hardware.
Descripción	Algún componente hardware implicado en el desarrollo falla de forma parcial o completa.
Probabilidad	Media
Impacto	Bajo

Tabla 2.12: Análisis riesgo RSG-09

Identificador	RSG-10
Nombre	Estimación errónea del período de una tarea.
Descripción	Dimensionar de manera equivocada puede implicar que no sea posible concluir en plazo dicha tarea.
Probabilidad	Alta
Impacto	Alto

Tabla 2.13: Análisis riesgo RSG-10

Identificador	RSG-11
Nombre	Corrupción en archivos del proyecto.
Descripción	Debido al trabajo remoto y continua descarga y carga de documentos de trabajo, se puede presentar corrupción en los archivos de trabajo.
Probabilidad	Alta
Impacto	Medio

Tabla 2.14: Análisis riesgo RSG-11

2.5.3. Matriz probabilidad impacto

Mediante esta matriz podemos visualizar de una manera sencilla los riesgos que focalizarán nuestra atención y nuestros recursos. Aquellos situados en la zona roja o amarilla requerirán medidas de prevención y contingencia.

Probabilidad	Impacto		
	alto	medio	bajo
alta	01 - 10	02 - 11	
media	03	04 - 05 - 06	09
baja			07 - 08

Tabla 2.15: Matriz probabilidad impacto

2.5.4. Acciones de prevención y contingencia

Una vez acotados los riesgos con mayor probabilidad e impacto, se debe planificar acciones para evitarlos, y si esto no es posible, actuaciones de contingencia que reduzcan su impacto sobre el proyecto. Para ello se utiliza la siguiente plantilla.

De esta manera, tenemos las siguientes actuaciones sobre los riesgos de mayor probabilidad e impacto.

Identificador - nombre	RSG-XX
Acción de prevención	Actuación que busca evitar la aparición del riesgo.
Acción de contingencia	Actuación que reduzcan el impacto sobre el proyecto

Tabla 2.16: Cuadro tipo actuación sobre riesgo

Identificador - nombre	RSG-01 - imposible cumplir con la planificación del proyecto.
Acción de prevención	Realizar las estimaciones teniendo en cuenta la propuesta de todos los implicados en la tarea. Aplicar un colchón de seguridad.
Acción de contingencia	Con el contexto del proyecto presente, redefinir el alcance.

Tabla 2.17: Actuación sebre RSG-01

Identificador - nombre	RSG-10 Estimación errónea del período de una tarea.
Acción de prevención	Estimar las tareas con amplitud temporal.
Acción de contingencia	Recalcular la planificación o ponderar la eliminación de tareas.

Tabla 2.18: Actuación sobre RSG-10

Identificador - nombre	RSG-02 - Conocimiento insuficiente para finalizar etapas de implementación.
Acción de prevención	Determinar con detalle cada uno de los pasos de la implementación, para que la formación aborde todos los puntos.
Acción de contingencia	Con el contexto del proyecto presente, redefinir el alcance.

Tabla 2.19: Actuación sobre RSG-02

Identificador - nombre	RSG-11 - Corrupción en archivos del proyecto.
Acción de prevención	Sistema de backup incremental durante el trabajo, realizando backup total al finalizar el día, manteniendo la notación de versiones.
Acción de contingencia	Recuperación del backup versionado inmediatamente anterior.

Tabla 2.20: Actuación sobre RSG-11

Identificador - nombre	RSG-03 - Reducción de las horas imputables al proyecto.
Acción de prevención	Desarrollar una planificación realista que tenga en cuenta posibles eventos que reduzcan las horas disponibles para dedicar al proyecto.
Acción de contingencia	Dilatar el plazo de finalización del proyecto, o reducir el alcance.

Tabla 2.21: Actuación sobre RSG-03

Identificador - nombre	RSG-04 - Se presentan nuevos requisitos..
Acción de prevención	La definición de requisitos debe ser amplia y detallada.
Acción de contingencia	Dilatar el plazo de finalización del proyecto, o reducir el alcance.

Tabla 2.22: Actuación sobre RSG-04

Identificador - nombre	RSG-05 - Cambios en los requisitos ya establecidos.
Acción de prevención	La definición de requisitos debe ser amplia y detallada.
Acción de contingencia	Dilatar el plazo de finalización del proyecto, o reducir el alcance.

Tabla 2.23: Actuación sobre RSG-05

Identificador - nombre	RSG-06 - La gestión de configuración no está siendo aplicada.
Acción de prevención	Revisiones periódicas de la correcta utilización de la gestión.
Acción de contingencia	Determinar el porqué de la no utilización de la gestión de configuración y encaminar su solución.

Tabla 2.24: Actuación sobre RSG-06

2.5.5. Planificación temporal (real)

Los plazos de desarrollo de las tareas se calcularon con holgura. La etapa de gestión se finalizó antes de tiempo, comenzando la etapa de formación con una mayor disponibilidad temporal. Aun con esta ventaja de tiempo desde el inicio se atisbo posibles lagunas en la formación, que se completarían durante el desarrollo de dichas tareas en la etapa de implementación.

Durante esta etapa el desarrollo el gestor se salió de plazo. Se manifestaron dos de los riesgos más probables **(RSG-02, RSG-10)** lo que desembocó en la implementación de un gestor menos sólido de lo esperado. Bien es cierto que este no era un objetivo principal del proyecto, lo cual resta impacto a dichos riesgos.

Algunas decisiones tomadas, por ejemplo la de utilizar la versión más actual de MySQL se tornaron contraproducentes, debido a que sus mejoras en seguridad implican un nivel de desarrollo desde el gestor que complicaba la puesta en funcionamiento con los conocimientos Spring obtenidos. Claramente se manifiesta un nuevo riesgo **(RSG-08)**. Para la conexión JPA se optó por eliminar el cifrado, y para la conexión logstash se optó por bajar la versión del conector a la 5.14 para facilitar el proceso, obteniendo avisos en la ejecución, debido a que la conexión entre los servicios no es codificada. Digamos que la securización de todo el proceso recomendador es una tarea pendiente a investigar en un entorno de producción, y sería un trabajo futuro que se debería abordar.

Finalmente, se produjo un incidente durante el movimiento de archivos de la imagen virtualizada de Centos 7 a la hora de copiar su contenido para presentar el trabajo ante el cotutor **(RSG-11)**. recurriendo al sistema de versionado de Google Drive el incidente no repercutió en los tiempos de manera señalable.

Capítulo 3

Análisis de requisitos

Antes de proceder con la implementación del proyecto, debemos detenernos y analizar el objetivo a resolver por nuestra aplicación. Este paso es necesario para determinar cómo se dará solución a dicho problema, definir y especificar requisitos para dar solución a las necesidades del usuario.

3.1. Especificación de requisitos

La especificación de requisitos determinará el comportamiento final del proyecto, los compromisos que se deberán cumplir para satisfacer los requerimientos del cliente. Para especificar estos requisitos, los dividiremos entre funcionales y no funcionales, determinando para cada uno de ellos su importancia y prioridad.

Aunque en este proyecto hemos construido el gestor de incidencias, el supuesto teórico es que partimos de un gestor ya operativo en el cliente, sobre el que se deberán realizar pequeñas modificaciones para introducir la conexión con el sistema recomendador. Es por esto que no aparecerán requisitos sobre la implementación del gestor, pero sí sobre su adaptación.

Se aplica la plantilla mostrada en la figura 3.1:

Identificador	RQF-XX | RNF-XX
Nombre	Nombre
Descripción	Descripción
Importancia	alta media baja
Urgencia	alta media baja

Tabla 3.1: Plantilla para especificación de requisitos

3.1.1. Requisitos funcionales

Estos requisitos determinan funcionalidades útiles para el usuario.

Identificador	RQF-01
Nombre	Acceso e ingesta de datos.
Descripción	El sistema debe estar capacitado para conectar con la base de datos del cliente, transferir los datos a indexar (para la ingesta) y de realizar anotaciones para facilitar el seguimiento de la implantación.
Importancia	alta
Urgencia	alta

Tabla 3.2: Requisito funcional RQF-01

Identificador	RQF-02
Nombre	Indexación de los datos.
Descripción	La aplicación ha de ser capaz de analizar la información recibida.
Importancia	alta
Urgencia	alta

Tabla 3.3: Requisito funcional RQF-02

Identificador	RQF-03
Nombre	Filtrado durante el análisis.
Descripción	El sistema será capaz de eliminar términos que no aporten valor en la búsqueda.
Importancia	alta
Urgencia	media

Tabla 3.4: Requisito funcional RQF-03

Identificador	RQF-04
Nombre	Gestión de la consulta de búsqueda.
Descripción	El sistema gestará la consulta de búsqueda del título del ticket seleccionado. Se debe modificar el código del gestor, para capacitar el rescate del título del ticket, y enviarlo como una consulta de búsqueda al analizador del sistema de recomendación
Importancia	alta
Urgencia	alta

Tabla 3.5: Requisito funcional RQF-04

Identificador	RQF-05
Nombre	Aplicación de sinónimos.
Descripción	Durante la gestión de la consulta se aplicarán sinónimos.
Importancia	media
Urgencia	media

Tabla 3.6: Requisito funcional RQF-05

Identificador	RQF-06
Nombre	Aplicación de búsqueda borrosa.
Descripción	Durante la gestión de la consulta se aplicará búsqueda borrosa.
Importancia	alta
Urgencia	media

Tabla 3.7: Requisito funcional RQF-06

Identificador	RQF-07
Nombre	Integración del resultado de búsqueda.
Descripción	Los resultados recomendados se integran en el interfaz del gestor.
Importancia	alta
Urgencia	alta

Tabla 3.8: Requisito funcional RQF-07

Identificador	RQF-08
Nombre	Realimentación del proceso y seguimiento.
Descripción	El sistema integrará la recomendación nuevamente en la base de datos del gestor en el momento en que se selecciona de entre las recomendaciones y se cierra el ticket, marcando la solución como recomendada para facilitar el seguimiento.
Importancia	alta
Urgencia	alta

Tabla 3.9: Requisito funcional RQF-08

Identificador	RQF-09
Nombre	Visualización de tiempos de resolución.
Descripción	El sistema integrará un visualizador de tiempos de resolución de incidencias.
Importancia	media
Urgencia	baja

Tabla 3.10: Requisito funcional RQF-09

Identificador	RQF-10
Nombre	Visualización de tiempos medios de resolución por semana.
Descripción	El sistema integrará un visualizador de tiempos medios de resolución de incidencias por semanas.
Importancia	media
Urgencia	baja

Tabla 3.11: Requisito funcional RQF-10

Identificador	RQF-11
Nombre	Visualización de utilización del recomendador.
Descripción	El sistema integrará un visualizador de utilización del recomendador para dar solución a la incidencia.
Importancia	media
Urgencia	baja

Tabla 3.12: Requisito funcional RQF-11

Identificador	RQF-12
Nombre	Panel de control aglutinador de visualizaciones.
Descripción	El sistema integrará un panel de control que permitirá aglutinar todas las visualizaciones de control.
Importancia	baja
Urgencia	baja

Tabla 3.13: Requisito funcional RQF-12

3.1.2. Requisitos no funcionales

Estos requisitos determinan restricciones en el diseño o la implementación del sistema que delimitan la calidad del sistema. Estos requisitos pueden ser de rendimiento, disponibilidad, datos de entrada, estabilidad o funcionalidad entre otros.

Requisitos de rendimiento

Identificador	RNF-01
Nombre	Ingesta de datos periodificable y no intrusiva.
Descripción	La ingesta de datos debe ser periodificable y no requerir la parada de la base de datos del cliente.
Importancia	alta
Urgencia	alta

Tabla 3.14: Requisito no funcional de rendimiento RNF-01

Identificador	RNF-02
Nombre	Procesamiento de la consulta en tiempo real.
Descripción	En el momento en que se selecciona la incidencia para su solución debemos recuperar las recomendaciones.
Importancia	alta
Urgencia	alta

Tabla 3.15: Requisito no funcional de rendimiento RNF-02

Requisitos de disponibilidad

Identificador	RNF-03
Nombre	El índice debe estar distribuido.
Descripción	El índice debe de estar distribuido en un cluster de al menos dos máquinas.
Importancia	alta
Urgencia	alta

Tabla 3.16: Requisito no funcional de disponibilidad RNF-03

3.1.3. Matriz de dependencia entre requisitos

Determinados requisitos pueden mostrar dependencias, relacionándose entre ellos. Para visualizar dichas dependencias, podemos utilizar la matriz de dependencia entre requisitos.

		Requisitos dependencia (requisitos que originan las dependencias)														
		RQF-01	RQF-02	RQF-03	RQF-04	RQF-05	RQF-06	RQF-07	RQF-08	RQF-09	RQF-10	RQF-11	RQF-12	RNF-01	RNF-02	RNF-03
Requisitos dependientes	RQF-01															
	RQF-02	x														
	RQF-03		x													
	RQF-04		x													
	RQF-05				x											
	RQF-06				x											
	RQF-07														x	
	RQF-08	x						x								
	RQF-09		x													
	RQF-10		x													
	RQF-11		x													
	RQF-12										x	x				
	RNF-01	x														
	RNF-02		x					x								
	RNF-03	x														

Figura 3.1: Matriz de dependencia entre requisitos

3.2. Casos de uso

Los casos de uso representan las interacciones entre usuarios y sistema. Son la forma de capturar los requisitos del sistema desde el punto de vista del usuario. En los siguientes puntos, mostraremos el diagrama reducido de casos de uso, su relación entre sistemas y actores, así como un detalle de los casos de uso recogido en las plantillas de descripción textual (especificación de casos de uso).

3.2.1. Diagrama de casos de uso

Se describen los potenciales casos de uso del sistema en la figura 3.2:

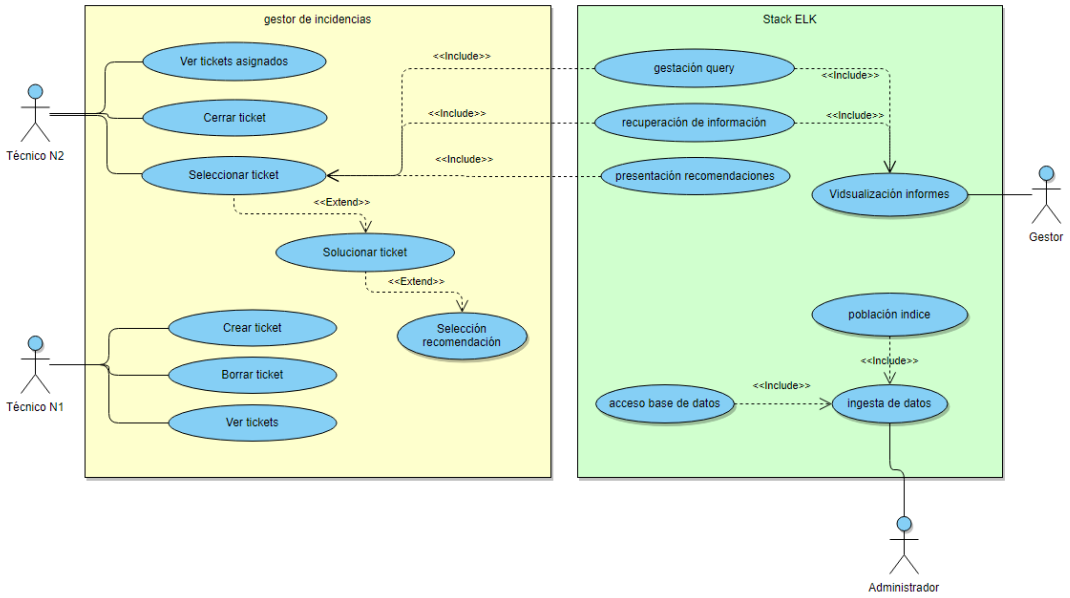


Figura 3.2: Casos de uso

De los casos de uso indicados, se puede observar que algunos de ellos no tendrían relación con el sistema teóricamente desarrollado (por ejemplo los relacionados con el actor técnico N1 o aquellos vinculados con la operación básica del gestor de incidencias). Se dejan descritos los casos de uso básicos, pero se incide en aquellos casos de uso que implican interacción con el sistema de recomendación (del caso de uso 5 en adelante).

3.2.2. Plantillas de descripción textual

Se describe a continuación detalladamente los principales casos de uso, definiendo precondiciones necesarias para su desarrollo, flujo de operaciones que determinan el caso y estado del sistema una vez completada su ejecución. Se aplica la plantilla de la tabla 3.3 como referencia.

Así, podríamos determinar los siguientes casos de uso:

Identificador	CU-XX
Nombre / Actor	Nombre del caso / actor del caso
Descripción	Descripción pormenorizada del caso
Precondición	Condiciones necesarias para la ejecución
Secuencia	Secuencia de acciones desarrolladas durante la ejecución
Postcondición	Estado del sistema tras la ejecución
Excepciones	Excepciones a la Secuencia de acciones del caso de uso

Tabla 3.17: Plantilla de descripción textual

Identificador	CU-01
Nombre / Actor	Crear un ticket / Técnico Nivel 1.
Descripción	El TN1 genera una nueva incidencia y la asigna a un TN2.
Precondición	El actor debe estar registrado con privilegios de TN1.
Secuencia	El actor registrado debe poseer privilegios de nivel TN1. El actor pulsa el botón de crear ticket. El actor introduce el título descriptivo de la incidencia. El actor asigna en el menú desplegable el técnico a asignar. El actor pulsa el botón de abrir ticket.
Postcondición	El ticket se guarda en base de datos, asignado a un técnico TN2.
Excepciones	Ninguna.

Tabla 3.18: Descripción textual CU-01

Identificador	CU-02
Nombre / Actor	Ver tickets / Técnico Nivel 1.
Descripción	El TN1 visualiza la lista total de tickets.
Precondición	El actor debe estar identificado con privilegios de TN1.
Secuencia	El actor identificado debe poseer privilegios de administrador TN1. El actor visualiza los tickets existentes para todos los técnicos TN2.
Postcondición	El sistema muestra la lista total de tickets.
Excepciones	Ninguna.

Tabla 3.19: Descripción textual CU-02

Identificador	CU-03
Nombre / Actor	Borrar un ticket / Técnico Nivel 1.
Descripción	El TN1 elimina una incidencia.
Precondición	El actor debe estar identificado con privilegios de TN1.
Secuencia	El actor identificado debe poseer privilegios de administrador TN1. El actor visualiza los tickets existentes para todos los técnicos TN2. El actor pulsa el icono de borrado, presente a la derecha del título de ticket.
Postcondición	El ticket se elimina de la base de datos .
Excepciones	Ninguna.

Tabla 3.20: Descripción textual CU-03

Identificador	CU-04
Nombre / Actor	Ver tickets asignados / Técnico Nivel 2.
Descripción	El TN2 visualiza su lista de tickets asignados.
Precondición	El actor debe estar registrado con privilegios de TN2.
Secuencia	El actor registrado debe poseer privilegios de TN2. El actor visualiza únicamente los tickets asignados.
Postcondición	El sistema muestra únicamente la lista de tickets asignados al técnico identificado.
Excepciones	Ninguna.

Tabla 3.21: Descripción textual CU-04

Identificador	CU-05
Nombre / Actor	Seleccionar ticket / Técnico Nivel 2.
Descripción	El TN2 visualiza un ticket seleccionado de su lista.
Precondición	Se muestra su lista de tickets asignados.
Secuencia	El actor registrado TN2 visualiza su lista de tickets. El actor selecciona el ticket a visualizar presionando en el icono de visualizar. El stack ELK elabora la consulta de búsqueda en función del título del ticket. El stack ELK ejecuta la consulta y recupera la información . El sistema presenta las recomendaciones disponibles para solucionar el ticket.
Postcondición	El sistema muestra la información disponible sobre ese ticket, así como las recomendaciones para su solución.
Excepciones	En caso de no encontrar recomendación posible no se mostrará resultado alguno en el área de recomendaciones.

Tabla 3.22: Descripción textual CU-05

Identificador	CU-06
Nombre / Actor	Solucionar ticket / Técnico Nivel 2.
Descripción	El TN2 cumplimenta el cuadro de solución y cierra el ticket.
Precondición	El actor debe haber seleccionado previamente el ticket.
Secuencia	El actor TN2 cubre el campo solución con el texto necesario para la solución. El actor puede seleccionar una solución de la lista de recomendaciones, pero decide NO hacerlo El actor selecciona la opción de cerrar el ticket.
Postcondición	El sistema computa el ticket como cerrado, añade la solución a la base de datos, pero marca esta como solucionada sin utilizar el recomendador
Excepciones	Ninguna.

Tabla 3.23: Descripción textual CU-06

Identificador	CU-07
Nombre / Actor	Seleccionar recomendación para solución/ Técnico Nivel 2.
Descripción	El TN2 cumplimenta el cuadro de solución mediante una solución recuperada por el recomendador.
Precondición	El actor debe haber seleccionado previamente el ticket.
Secuencia	El actor selecciona una solución de la lista de recomendaciones. El campo solución se cumplimenta automáticamente con el contenido de la solución recomendada y seleccionada por el actor. El actor selecciona la opción de cerrar el ticket.
Postcondición	El sistema computa el ticket como cerrado, añade la solución a la base de datos, y marca esta como solucionada utilizando el recomendador
Excepciones	Ninguna.

Tabla 3.24: Descripción textual CU-07

Identificador	CU-08
Nombre / Actor	Periodifica o lanza una ingesta de datos / Administrador.
Descripción	El administrador procede a cargar los tickets marcados como solucionados en el índice de Elasticsearch.
Precondición	El actor se conecta a la máquina Logstash.
Secuencia	El actor ejecuta la invocación necesaria para arrancar un pipeline de carga unitaria o periodificada. El pipeline accede a la base de datos. El pipeline recupera los tickets marcados como cerrados. El pipeline conecta con el índice Elasticsearch y añade los nuevos datos. El actor finaliza el pipeline y se desconecta de la máquina.
Postcondición	Los nuevos tickets quedan a disposición del analizador.
Excepciones	El proceso expone un error si no están operativas la base de datos o el servidor Elasticsearch.

Tabla 3.25: Descripción textual CU-08

Identificador	CU-09
Nombre / Actor	Visualizar informe / Gestor.
Descripción	El gestor accede al cuadro de mandos para ver la implantación del recomendador.
Precondición	El sistema Kibana debe estar operativo y en conexión con Elasticsearch.
Secuencia	El actor ejecuta la invocación necesaria para visualizar el panel de control. El panel de control recopila las visualizaciones. La visualizaciones gestionan la consulta y recuperan la información desde el índice. Se visualizan las gráficas de implantación.
Postcondición	El panel de control muestra las gráficas de implantación del recomendador.
Excepciones	Si el índice no está disponible se visualiza un aviso de error.

Tabla 3.26: Descripción textual CU-09

3.2.3. Matriz de requisitos/casos de uso

Mediante esta matriz podemos observar la relación entre los casos de uso y los requisitos, detallando dependencias entre ellos.

Casos de uso	Requisitos dependencia														
	RQF-01	RQF-02	RQF-03	RQF-04	RQF-05	RQF-06	RQF-07	RQF-08	RQF-09	RQF-10	RQF-11	RQF-12	RNF-01	RNF-02	RNF-03
CU-01															
CU-02															
CU-03															
CU-04															
CU-05				x	x	x	x							x	
CU-06				x	x	x	x							x	
CU-07				x	x	x	x	x						x	
CU-08	x	x	x										x		x
CU-09									x	x	x	x			

Figura 3.3: Matriz Casos de uso

Capítulo 4

Análisis de tecnologías y herramientas

Se procede a describir aquellas tecnologías empleadas en el desarrollo, tanto del sistema gestor de incidencias, como en la puesta en funcionamiento del sistema recomendador, así como las empleadas para la elaboración de gráficas, figuras y redacción de la memoria.

4.1. Tecnologías empleadas en el desarrollo

4.1.1. Librerías

- **QueryDSL**[20] nos permite generar consultas de una manera rápida, consistente y segura. Esta librería fue en principio diseñada para HQL Hibernate pero hoy en día soporta varias tecnologías de persistencia de datos, como: JPA, JDBC, Lucene, MongoDB, etc.

- **API Elasticsearch + Java High Level REST Client** implementado en la versión 6 y superiores, viene a substituir al cliente de transporte (**TransportClient**) anterior, que implicaba la gestión REST a base de peticiones generadas por nosotros. El nuevo cliente acepta todas las peticiones disponibles en la API de Elasticsearch

- **GSON**[21] es un API en Java, desarrollada por Google, que se utiliza para convertir objetos Java a JSON (serialización) y JSON a objetos Java (deserialización). Facilitará el trabajo de presentación de la información devuelta por la consulta formada con el cliente de Elasticsearch.

4.1.2. Frameworks

- **Spring boot** nace para simplificar los pasos de creación de un proyecto, descarga de dependencias y despliegue en un servidor, dejando al desarrollador todo el tiempo necesario para la creación de la aplicación. Se seleccionó este framework debido a tres de sus características que lo hacen tremendamente popular.

  - **Configuración:** Spring Boot cuenta con un complejo módulo que configura automáticamente todos los aspectos de nuestra aplicación para poder ejecutar esta, sin tener que definir absolutamente nada.
  - **Resolución de dependencias:** Con Spring Boot solo hay que determinar qué tipo de proyecto estaremos utilizando y él se encarga de resolver todas las librerías/dependencias para que la aplicación funcione.
  - **Despliegue:** Spring Boot se puede ejecutar como una aplicación Standalone, pero también es posible ejecutar aplicaciones web, ya que es posible desplegar las aplicaciones mediante un servidor web integrado, como es el caso de Tomcat, Jetty o Undertow.

- **Thymeleaf** es un motor de plantillas Java para web, construido sobre estándares HTML5 e integrado con spring framework de manera sencilla. Se recomienda su uso si el proyecto se desarrolla utilizando el patrón MVC, ya que la integración es casi transparente para el desarrollador.

- **Maven** es una herramienta para proyectos Java que facilita enormemente la administración de dependencias, construcción y empaquetado de aplicaciones. Instalado con spring boot suite se utilizó para realizar de manera mucho más fácil el proceso de integración de funcionalidades y librerías, simplemente añadiendo sus dependencias en el archivo pom.xml.

4.1.3. Virtualización

- **VMware Workstation** es una suite de VMware Inc, filial de EMC Corporation que proporciona software de virtualización disponible para ordenadores compatibles X86. En nuestro desarrollo utilizaremos VMware Player 15, aplicación de virtualización que la empresa pone a disposición de manera gratuita (en proyectos de uso no comercial), y sobre la que se monta una distribución Centos 7.

- **Docker** es un sistema de virtualización, que permite ejecutar aplicaciones en 'contenedores', pero sin necesidad de incluir la imagen completa de un sistema operativo, únicamente librerías y configuraciones necesarias para hacer funcionar el software contenido, ya que los contenedores comparten

el kernel del sistema anfitrión. De esta manera se consiguen sistemas ligeros, autocontenidos, que garantizan que el software siempre responderá de igual manera. Esto permitirá la portabilidad del sistema, y ahorrará mucho tiempo en la configuración de las máquinas contenedoras de los servicios a desplegar.

4.1.4. Servidor web

- **Apache Tomcat** es el servidor embebido en Spring Boot, framework empleado para la realización del gestor de tickets, que será desplegado en un jar, sobre un contenedor con una imagen de java 8. Al ser un servicio embebido, se facilita la configuración del servidor web.

4.1.5. IDE de desarrollo

- **Spring tool suite (STS4)** es un IDE de código abierto basado en Eclipse. Mediante la adición de diferentes módulos se adapta su funcionalidad al proyecto a desarrollar. En nuestro caso se aplicaron los módulos de Spring MVC, para implementar el patrón modelo-vista-controlador, el módulo Hibernate, para facilitar la interacción con la base de datos, y el módulo de seguridad, para la creación de roles y perfiles de acceso a la aplicación

4.1.6. Lenguajes

1. **Java** Para la programación del gestor de incidencias
2. **Mysql** Para la gestión de la base de datos e importación y exportación de los tickets
3. **HTML5** Para la creación de las vistas en thymeleaf
4. **CSS** Para la maquetación de las plantillas Thymeleaf
5. **Yaml** Para la configuración de contenedores y red de contenedores
6. **QueryDSL** Para las consultas a la base de datos mediante Hibernate

4.2. Tecnologías empleadas en la documentación

- **TeXstudio IDE**[1] **(sobre LaTeX**[2]**)** Para la elaboración de la documentación necesaria, utilizando las plantillas facilitadas por la USC, se utilizó una instalación de LaTeX sobre Centos 7. Para facilitar la gestión de archivos

[1]https://www.texstudio.org/
[2]https://www.latex-project.org/

se utilizó el IDE TeXstudio, que facilita la exportación a PDF de los documentos generados.

- **WBSTool**[3], generador On-line de esquemas EDT empleado en la redacción del anteproyecto y posteriormente en el apartado de gestión del proyecto.
- **Microsoft Project**[4] software de gestión de proyectos de Microsoft. Se ha empleando la versión de prueba para la realización de los diagramas de Gantt y la planificación del proyecto.
- **Visual Paradigm**[5] software de generación de diagramas que admite una amplia variedad de disgramas comerciales y técnicos. Se utilizó su versión gratuita online para la realización de los diagramas de casos de uso.
- **Notepad++**[6] editor de textos con utilidades avanzadas. Se usó para la elaboración de scripts o para tomar notas cuando no se disponía de los programas adecuados.
- **MySQL Workbench** cliente hábil para la realización de labores administrativas sobre una base de datos Mysql. Se utilizó para la creación de las tablas, importación de registros ficticios, exportación de scripts y la mayoría de la interacción con la base de datos Mysql.
- **OBS Studio**[7] aplicación gratuita para la grabación de vídeos. Utilizada para para recoger en vídeo pequeñas presentaciones de los incrementos finalizados, para comunicar la evolución del proyecto a la tutora.
- **Google Drive**[8] servicio de alojamiento de archivos de Google. Se utiliza para el versionado de archivos y de la imagen de trabajo Centos 7. Una de las funciones más utilizada ha sido la posibilidad de compartir archivos mediante enlace privado, para presentar los vídeos evolutivos del avance del proyecto.

[3]http://www.wbstool.com/
[4]https://products.office.com/es-es/project/project-management-software
[5]https://www.visual-paradigm.com/
[6]https://notepad-plus-plus.org/
[7]https://obsproject.com/es
[8]https://www.google.com/intl/es_ALL/drive/

Capítulo 5

Diseño e implementación

5.1. Arquitectura del sistema

Mediante la siguiente figura se determina cómo funciona el sistema a desarrollar.

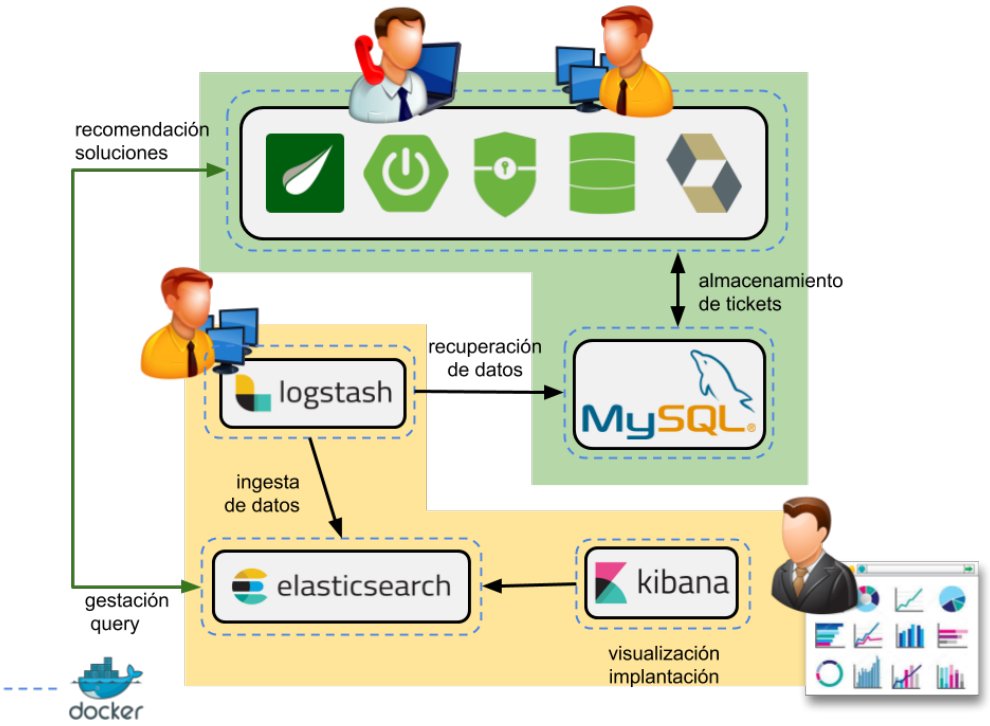


Figura 5.1: Arquitectura del sistema

Se señalan dos módulos diferenciados:

El **primer módulo** abarcaría la parte del cliente, que presentaría su sistema de gestión de incidencias, apoyado en un servidor de base de datos sobre el cual solicita la implementación del sistema recomendador de incidencias. En el proyecto se ha simulado esta parte del cliente, generando un gestor muy básico, pero necesario para poder visualizar el objetivo final del proyecto.

El **segundo módulo** incluiría todo el sistema ELK, compuesto por los servicios de Elasticsearch, Logstash y Kibana. Este módulo deberá ser fácilmente acoplable a la infraestructura del cliente, no invasivo, y autogestionado por el personal de la empresa cliente. El módulo recomendador debe presentar la capacidad de ser portable a los sistemas de otros clientes, sin presentar un coste de adaptación elevado.

Para fomentar dicha portabilidad, todos los elementos se encapsulan en contenedores Docker, fácilmente replicables en distintos sistemas operativos.

La cualidad no invasiva del sistema hace referencia a la capacidad de realizar la ingesta de datos en caliente, sin necesidad de detener la base de datos del cliente a la hora de hacer la extracción de los tickets resueltos. La posibilidad de configurar la extracción de información desde el pipeline de Logstash implica que, a vista de los administradores de la base de datos, seremos un usuario más, realizando una consulta simple, que no llegará a sobrecargar ni penalizar el rendimiento del cliente.

Se describe a continuación más detalladamente la estructura del módulo cliente, que se implementó en un primer paso en local. Posteriormente se incidió en la migración de este gestor sobre tecnologías de virtualización. Una vez que se replicó su funcionalidad se continuó desarrollando el segundo módulo, que implementaba la funcionalidad recomendadora.

5.2. Gestor de incidencias (módulo cliente)

Para la visualización operativa del sistema de recomendación, se hizo necesario implementar un pequeño gestor de incidencias, muy básico, que permitiese roles diferenciados (administrador y técnico) así como la capacidad de crear, almacenar y recuperar tickets de incidencias.

Se utiliza el patrón Modelo Vista Controlador (MVC), patrón de arquitectura de software que separa la lógica de la aplicación (el modelo) de su representación (la vista) empleando para la interacción entre ambas el concepto de controlador.

Será este el encargado de trasladar las solicitudes provenientes de la vista a la aplicación, para que esta desarrolle la lógica de negocio y posteriormente, devuelva el resultado al controlador, y que sea este el que le envíe a la vista el resultado para su representación.

El objetivo del patrón MVC es separar la vista de la lógica de la aplicación para facilitar la reutilización de código y simplificar el desarrollo de ambas capas de forma separada.

Anteriormente, a la hora de crear un proyecto con spring, se creaba inicialmente el proyecto maven (o cualquier otro gestor de dependencias) y se añadían las dependencias necesarias para utilizar spring MVC en el archivo *pom.xml*. En la actualidad se parte de un proyecto pregenerado, utilizando la herramienta on-line **Spring initializr**[1] que aportará la estructura inicial, según las configuraciones indicadas.

El primer paso en la configuración será señalar el tipo de proyecto. Se determina el tipo Maven, a desarrollar bajo la versión 2.2.2 en lenguaje Java versión 1.8 y con las dependencias iniciales básicas necesarias para el patrón MVC (Web y Thymeleaf). Posteriormente se añade de forma manual JPA, MySQL y Security, según avancemos en las capas a implementar. Se genera el proyecto en zip, obteniendo una estructura a importar en el workspace del IDE STS4 como un proyecto maven existente. El proyecto comenzará a descargar las dependencias cuando se ejecute desde línea de comandos, en la carpeta de nuestro workspace, la orden *mvn clean install*. Es momento de comenzar a programar la lógica de la aplicación.

Antes de la versión 2.5 de Spring, cada uno de los beans[2] que se emplean dentro del proyecto debían ser declarados de manera manual dentro de un archivo XML, generando un gran problema para el mantenimiento de un desarrollo o de un sistema.

De esta necesidad nace la inyección de dependencias y anotaciones en Spring 2.5 y superiores. Todas las anotaciones al arrancar se añaden como beans al contenedor de spring. Posteriormente se irán llamando mediante inyección de dependencias, con la anotación ***@Autowired*** determinando el nombre del bean con la anotación ***@Qualifier***.

La anotación principal será ***@Component***, con sus hijas, según nos encon-

[1] https://start.spring.io/

[2] Los Beans son objetos que maneja el contendor de spring. Su fin es el de suministrar objetos a una clase sin que la propia clase tenga que crearlos.

tremos desarrollando en una capa u otra de la aplicación, ***@Controller @Service @Repository***.

- **@controller** es un componente con funcionalidades para la capa de presentación.
- **@Service** es un componente con funcionalidades para la capa de servicios, lógica de negocio.
- **@Repository** es un componente con funcionalidades para la capa de acceso a datos.

Para desarrollar este gestor, se aplicó un diseño por modelo de capas, utilizando MySql para la capa de base de datos, Hibernate para el acceso a datos, (mediante Spring JPA), Spring boot y Spring Security para la lógica de negocio, y finalmente Thymeleaf, HTML5 y CSS para la capa de presentación.

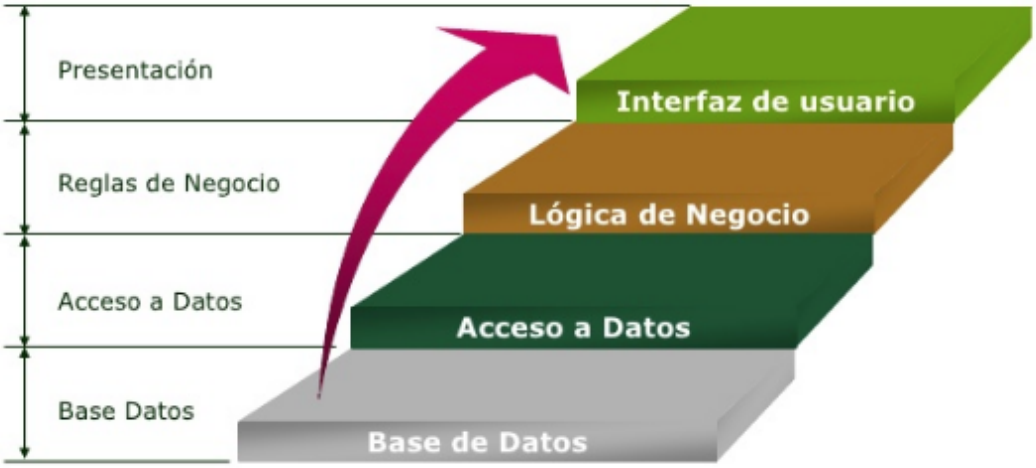


Figura 5.2: Modelo de capas

A continuación se detalla cada parte del proceso. Comenzando con la capa de reglas de negocio y presentación, abordando posteriormente toda la creación y acceso a la base de datos. Para una descripción de las clases implicadas en el proyecto, se puede acceder a los diagramas de clases que se encuentran en el apéndice correspondiente.

5.2.1. Capa lógica de negocio y presentación

La lógica de negocio, en resumen, se basa en la capacidad de un administrador para generar una incidencia, y asignar esta a uno de los técnicos disponibles. Se ha prescindido de todo el proceso de administración de usuarios, asignando directamente estos en base de datos, creando un administrador y tres técnicos. Una vez que un administrador crea una incidencia, esta se almacena en estado abierto en la base de datos a la espera de que un técnico la atienda. El administrador puede acceder a todos los tickets generados.

En el momento en que un técnico se identifica en la aplicación, procede a visualizar sus tickets asignados. Mediante el icono de visualización podrá acceder a los tickets y dar solución a aquellos en estado abierto.

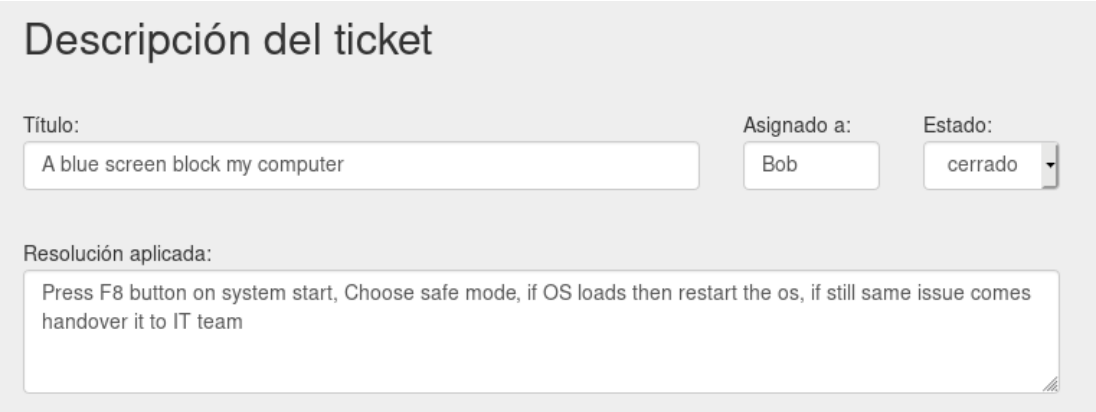


Figura 5.3: Visualización del cuerpo de un ticket ya solucionado

En esta capa se encuentra el núcleo de Spring MVC que pasaremos a interpretar.

Spring MVC - Modelo

El modelo estará formado por los objetos a programar, por lo tanto Spring no nos va a limitar ni ayudar en este aspecto. Para poder enviar estos objetos a la vista y que esta pueda identificar y saber qué tiene que hacer con ellos nos ofrece Model, ModelMap y ModelAndView.

ModelMap es una clase, Model es una interfaz y ModelAndView reúne en un solo objeto el modelo (un ModelMap) y la vista a la que se quiere enviar. En la siguiente captura se observa cómo se instancia un objeto TicketModel que se gestiona mediante la interfaz model al crear un nuevo ticket. Este objeto se poblará

de contenido en la vista.

```
//crear un ticket nuevo

@GetMapping("/ticketform")
public String redirectTicketForm (Model model) throws IOException {
    TicketModel ticket = new TicketModel();
    model.addAttribute("ticketModel", ticket);
    return "ticketform.html";
}
```

Figura 5.4: Instanciación del objeto para poblarlo en la vista

En esta capa de desarrollo también se implementan los servicios necesarios, fragmentos de código reutilizable que se almacena en el contenedor spring y que pueden ser llamados mediante la inyección de dependencias.

Para la creación ordenada de estos servicios, se debe gestionar primeramente la creación de sus métodos en un interfaz, que posteriormente los implementará en una nueva clase.

A continuación se observa un ejemplo de un servicio de nuestro gestor, con su interfaz, implementación y cómo se hace uso del servicio en el controlador mediante inyección de dependencias. En la imagen siguiente se presenta la interfaz del servicio, en el que se muestran los métodos disponibles.

```
public interface TicketService {

    public abstract TicketModel addTicket(TicketModel ticketModel);

    public abstract TicketModel updateTicket(TicketModel ticketModel);

    public abstract List<TicketModel>  listAllTickets(); //este método no recibe argumentos,

    public abstract List<TicketModel>  listAllTicketsByDesignated(String designated);

    public abstract Ticket findTicketById(int id);

    public abstract TicketModel findTicketModelById(int id);

    public abstract void removeTicket(int id);

    public abstract List<String> listAllUsers();

}
```

Figura 5.5: Interfaz de métodos disponibles

El siguiente paso será crear la clase que implementará este servicio y anotarla como ***@Service***, para que Spring la genere como un bean.

```java
@Service("ticketServiceImpl") // generamos un servicio
public class TicketServiceImpl implements TicketService {
//aqui implementamos los métodos anunciados en la interface

    private static final Log LOGGER = LogFactory.getLog(TicketServiceImpl.class);

    @Autowired
    @Qualifier("ticketRepository")
    private TicketRepository ticketRepository;
```

Figura 5.6: Creación del servicio

Nótese cómo esta clase hace uso de inyección de dependencias para utilizar el servicio repositorio de acceso a datos, que se verá más adelante. A continuación se señala el detalle del método que permite eliminar un ticket, solicitando como argumento el id de este. Mediante el servicio de repositorio gestionado con Hibernate, realizamos una consulta y un borrado dentro del método sin necesidad de escribir la consulta de Mysql.

```java
@Override
public void removeTicket(int id) {
    Ticket ticket = findTicketById(id);
    if (ticket != null) {
        ticketRepository.delete(ticket);
    }
}
```

Figura 5.7: Implementación del método de borrado

Posteriormente, en el controller, se realiza la inyección de dependencias para recuperar este servicio, indicando con @Qualifier el nombre que se otorgó al servicio durante su creación.

```java
@Autowired
@Qualifier("ticketServiceImpl")
private TicketService ticketService;
```

Figura 5.8: Inyección de dependencias del bean creado

De esta manera se invoca el método de borrado con, por ejemplo un ***@GetMapping*** desde el controlador, gestionando una redirección al finalizar el proceso.

```
@GetMapping("/removeticket")
public ModelAndView removeTIcket(@RequestParam(name="id", required=true) int id) {
    ticketService.removeTicket(id);
    return showTickets();

}
```

Figura 5.9: Llamada al método

Spring MVC - Vista

Para la creación de vistas se utiliza el motor de plantillas Thymeleaf que ofrece un set completo de integración que facilita su incorporación dentro del patrón MVC. Mediante atributos propios podemos ejecutar de manera muy sencilla interacciones con los objetos manejados desde el controlador. En la siguiente captura del formulario de creación de un ticket se observa alguno de los atributos más característicos.

```
<form action="#" th:action="@{/tickets/addticket}"
    th:object="${ticketModel}" method="post">
    <input type="hidden" th:field="*{id}"></input>
    <input type="hidden" th:field="*{state}" th:value="abierto"></input>
    <div class="row">
        <div class="col-sm-8">
    Título:<input
        type="text" class="form-control" th:field="*{title}"
        th:value="${title}"></input>
        </div>
        <div class="col-sm-2">
                <div class="form-group">
                 Asignado a: <select class="form-control" th:field="*{designated}">
                    <option class="form-control"
                    th:each="soportista : ${listasoportistas}"
                    th:value="${soportista}" th:text="${soportista}"></option>
                            </select>
                </div>
    </div>
    </div>
    <div class="col-sm-2">
        <button class="btn btn-primary" type="submit">
            <span class="glyphicon glyphicon-ok" aria-hidden="true"></span>
            Abrir ticket
        </button>
    </div>

</form>
```

Figura 5.10: Código vista del formulario de creación de un ticket

Se observa como el atributo **th:action** mediante la expresión de enlace @ redirecciona a la vista addticket, utilizando el atributo **th:object** y la expresión variable *$* indicamos el objeto que recibirá la información recopilada. Tenemos

dos atributos de ese objeto que, usando las etiquetas **th:field** se completan con los valores determinados mediante expresiones de selección,"*{ }", a una de ellas se le determina un valor estático por defecto con el atributo **th:value="abierto"** (todos los tickets se crean por defecto como abiertos). Más adelante vemos como se utiliza nuevamente este atributo para completar el título del ticket, recogiendo su valor del campo title con una expresión de variable *${title}*. Para recuperar los técnicos asignables a este ticket, se hace lectura de los valores proporcionados por un array enviado desde el controlador. Este array se itera mediante **th:each**, indicando el nombre del objeto por el que iterar *${Listasoportistas}*. Esto se mostrará en un desplegable con los valores de la lista **th:text="${soportista}"** y sus valores subyacentes en un **th:value**. El elemento seleccionado se adjuntará en el campo *designated* del objeto **th:object=${ticketModel}"** a enviar al controlador mediante el atributo **th:field="*{designated}"**

Spring MVC - Controlador

El controlador es el punto de unión entre el modelo y la vista. Encauza los datos hacia las vistas a mostrar, así como recopila los datos enviados desde la vista para actuar en consecuencia (vincularlos a un repositorio, hacer una consulta, etc.).

Para crear un controlador en Spring simplemente se añade la anotación ***@Controller*** a una clase java y se anotan los métodos que queramos relacionar con las peticiones http con ***@RequestMapping*** o con alguna de sus abreviaturas (***@GetMapping***, ***@PostMapping***, etc). En la siguiente captura se aprecia una sección de la clase *ticketController*, que muestra la petición ***@PostMapping*** a la hora de actualizar un ticket.

```
@PostMapping("/updateticket")
public String updateTicket(@ModelAttribute(name="ticketModel")TicketModel ticketModel, Model model) {/
    LOGGER.info("Metodo: updateTIcket() -- parametros: "+ ticketModel.toString());
    if(ticketModel.getState().equals("cerrado")) {
        Timestamp fecha_open = ticketModel.getFecha_open();
        Timestamp fecha_close = new Timestamp(System.currentTimeMillis());
        ticketModel.setFecha_close(fecha_close);
        long milliseconds = fecha_close.getTime() - fecha_open.getTime();
        int tiempo = (int) milliseconds / 60000; //pasamos milisegundos a minutos
        ticketModel.setTiempo_medio(tiempo);
    }

    if(null != ticketService.updateTicket(ticketModel)) {
        model.addAttribute("updated", 1);//esto disparará el if en la vista, visualizando el add conta
        LOGGER.info("result 1 updated");
    }else{
        model.addAttribute("updated", 0);//esto disparará el if en la vista, visualizando el add conta
        LOGGER.info("updated 0 error");
    }; //le pasamos el contactModel que viene del formulario post
    return "redirect:/tickets/showtickets";
}
```

Figura 5.11: Petición @PostMapping en actualización de ticket

5.2.2. Capa base de datos

El gestor simulado almacenaría sus tickets sobre una base de datos MySql. Se seleccionó este motor debido a su universalidad, pero se debía tener en cuenta que el potencial cliente podría presentar cualquier otro (Postgres, SQL Server, Oracle o con menos certeza, aunque también posible, MongoDB u otra base no relacional).

A la hora de diseñar la base de datos, se opta por reducirla a lo más básico. Una tabla de tickets y las tablas necesarias para la gestión de los usuarios, que vendrá determinada por las necesidades del sistema Spring Security.

Para diseñar la tabla de tickets, primero se debe definir qué elementos conformarán el ticket básico. De entrada se mostrará el título del ticket (title), a qué técnico se asignará (designated), el estado del ticket (state), su resolución (resolution) y los campos ya no visibles de Id, fecha de apertura, actualización y cierre, así como un cálculo del tiempo transcurrido entre su apertura y cierre, que se almacenará en el campo tiempo medio. También se añade un campo que indicará si el ticket se ha solucionado utilizando el recomendador.

Para poblar de contenido la tabla de tickets, se procedió a buscar algún tipo de base de conocimiento que nos proporcionara tickets de referencia reales. Esta tarea a priori sencilla dilató el proceso más de lo deseado, ya que no es tarea trivial, como sí lo es por ejemplo en entornos big data, encontrar un repositorio útil. Finalmente se encontró un archivo ***train_tickets.csv*** proveniente de un servicio helpdesk que pudo ser de utilidad, realizando sobre él ciertas modificaciones antes de importarlo a la base de datos. El archivo csv presentaba 4 campos (id, title, resolution y class) del que únicamente se aprovecha title y resolution. De manera aleatoria se genera el campo designated, con los tres técnicos creados para la aplicación (Bob, Charles, Danna). Para el total de tickets importados, se asume que su estado es cerrado (ya han sido solucionados, al tener contenido en el campo resolution). Para poblar los campos de fecha, especialmente fecha de apertura y cierre del ticket, se procede a generar entre una fecha inicial y final, todo un rango de valores aleatorios. Con estos valores se completa el campo fecha_open. Para poblar las fechas de cierre de los tickets, se genera un nuevo valor aleatorio sobre la fecha de apertura, con un rango máximo de 3 horas. Posteriormente se procede a poblar los tickets realizando una importación del csv desde MySQL Workbench y generando el código para los inserts. El tiempo medio de cierre de ticket se calculó con una operación aritmética en la base de datos.

```
UPDATE tickets
SET tiempo_medio = timestampdiff(minute, fecha_open, fecha_close);

UPDATE tickets
```

```
SET usa_recomendacion = 0;
```

Todo este proceso fue necesario para poblar inicialmente la tabla de tickets. Para las nuevas incidencias que se crearán desde el gestor, el propio código de este, y los campos autocompletados de la tabla se encargarán de poblar las fechas.

El código de creación de la tabla queda de la siguiente manera:

```
CREATE TABLE `tickets` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `resolution` varchar(255) DEFAULT NULL,
  `designated` varchar(25) NOT NULL,
  `state` varchar(10) DEFAULT 'abierto',
  `fecha_update` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `fecha_open` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `fecha_close` timestamp NULL DEFAULT NULL,
  `tiempo_medio` int(11) DEFAULT NULL,
  `usa_recomendacion` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Figura 5.12: Script creación tabla de tickets

El diseño de las tablas se aprecia en la siguiente captura.

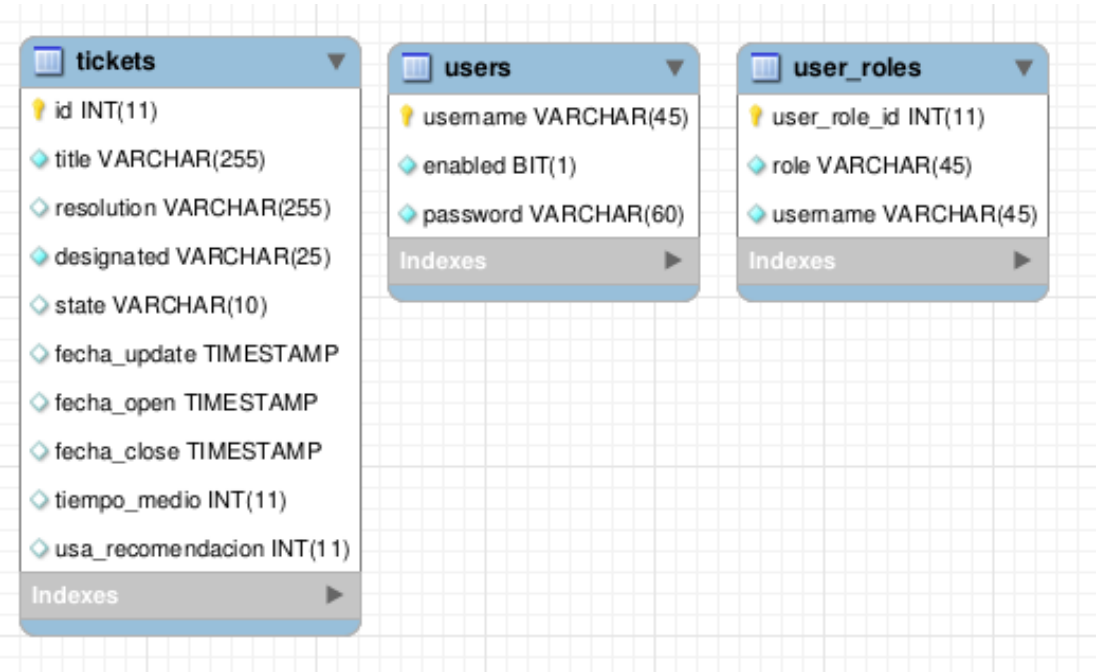


Figura 5.13: Diseño de tablas del gestor

5.2.3. Capa acceso a datos

Para facilitar el acceso a la base de datos MySQL se utiliza la tecnología Spring JPA con Hibernate. Para añadir estas funcionalidades, se debe descargar las dependencias necesarias y añadirlas al archivo **pom.xml** para que Maven se ocupe del proceso de descarga y disposición para el proyecto. Se añade el siguiente código al archivo **pom.xml**.

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Figura 5.14: Dependencias JPA y conector Mysql

La primera dependencia es el starter de spring JPA y segunda es el artefacto de conexión a una base de datos MySQL. El siguiente paso será desde un terminal, acceder al workspace y lanzar el comando *maven clean install*, para descargar las dependencias añadidas. Antes de acceder a la base de datos, se debe cubrir la configuración del starter para dar acceso al conector de MySQL. Dentro del archivo **aplication.properties** se asigna username y password, así como la url de conexión con la base de datos. También se debe indicar la validación por parte de Hibernate de la coincidencia entre entidades creadas y tablas accedidas. Si no es así, actualizará las tablas. JPA mostrará en el log de la aplicación las consultas sql.

```
spring.datasource.password=pw3358pec
spring.datasource.url = jdbc:mysql://localhost:3306/mydb?serverTimezone=UTC
spring.datasource.username=root
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Figura 5.15: Detalle de la configuración del aplication.properties

Para que el proyecto pueda interactuar con la base de datos, se deben crear las entidades con las que trabajará Hibernate. Para el proyecto , se creará la entidad Ticket con anotación ***@Entity*** en la que se indica la tabla con la que se corresponde. Se añaden los campos, indicando ***@Column*** para determinar el nombre de la columna correspondiente, ***@Id*** para señalar la clave primaria y ***@GeneratedValue*** para recuperar el siguiente valor si se trata de un campo

autogenerado.

```
@Entity
@Table(name = "tickets")
public class Ticket {


    @Id
    @GeneratedValue
    @Column(name="id")
    private int id;
    @Column(name="title")
    private String title;
    @Column(name="resolution")
    private String resolution;
    @Column(name="designated")
    private String designated;
    @Column(name="state", columnDefinition = "varchar(10) default 'abierto'")
    private String state;
    @Column(name="fecha_open")
    private Timestamp fecha_open;
    @Column(name="fecha_close")
    private Timestamp fecha_close;
    @Column(name="tiempo_medio")
    private int tiempo_medio;
    @Column(name="usa_recomendacion")
    private int usa_recomendacion;
```

Figura 5.16: Entidad Hibernate

El siguiente paso será crear un interfaz repositorio que comunicará controlador y entidad, presentando los métodos que estarán disponibles en la implementación. Se debe anotar como ***@Repository*** señalando el nombre identificativo del bean para el controlador y extender la interfaz de JpaRepository, para disponer de todos los métodos útiles de JPA que se encargarán de gestionar y formalizar las consultas necesarias para buscar en la tabla de tickets.

```
@Repository("ticketRepository")
public interface TicketRepository  extends JpaRepository<Ticket, Serializable> {

    public abstract Ticket findById(int Id);

    public abstract Ticket findByDesignated(String designated);

    public abstract List<Ticket> findAllByDesignated(String designated);

}
```

Figura 5.17: Interfaz del repositorio

Para hacer uso de este interfaz, en el controlador se debe añadir la inyección de dependencias, mediante la anotación ***@Autowired***, seleccionando el nombre del bean con la anotación ***@Qualifier***.

Es importante, por motivos de calidad de software a la hora de trabajar con JPA, que los controladores no interactúen directamente con entidades ya que el objetivo final de estas es mapear contenido contra la base de datos, no ser utilizadas para la representación de ese contenido en una vista (puede que alguno de los campos de la base de datos no pueda ser serializable). Es por ello que se recomienda crear una clase que convierta nuestras entidades en modelos y viceversa. En el proyecto, crearemos la clase *TicketConverter* que contendrá dos métodos, uno transformará de entity recibida como argumento a model, y el siguiente de model a entity.

5.3. Securización mediante Spring Security

Para la gestión de usuarios y roles de manera segura, se implementa el módulo Spring Security, añadiendo sus dependencias al archivo **pom.xml** y ejecutando nuevamente el comando de Maven *clean install*.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Figura 5.18: Dependencias del módulo Spring Security

Hasta ahora en el proyecto, la validación para los accesos se incluía en el código. Para añadir cierto nivel de seguridad, estos accesos deben ser cifrados y almacenados en tablas de nuestra base de datos. Será necesario crear nuevas entidades, repositorios y servicios para tramitar el acceso de los usuarios. Estas entidades mapearán los campos creados en las tablas users y user_roles.

El servicio creado, tendrá un método que recibe un string por parámetro (nombre de usuario). El servicio llamará al repositorio, que ejecutará la consulta de búsqueda por nombre de usuario, devolverá una entidad usuario y roles, que serán transformados en los objetos específicos requeridos por Spring Security (*UserDetails* y lista de *GrantedAutority*).

La potencia del módulo permitiría dar complejidad a la gestión de cuentas, añadiendo disponibilidad de usuarios (activos, inhabilitados), credenciales expiradas, cuentas bloqueadas, etc. Pero dejaremos de lado todas estas opciones ya

que se escapan del alcance y no son necesarias en el desarrollo del gestor de incidencias.

También se gestiona la securización de los request generados (exceptuando css e imágenes) así como el proceso de login, añadiendo la clase *SecurityConfiguration* con las anotaciones ***@Configuration*** para manejar configuraciones adicionales no disponibles en el archivo *aplication.properties* y ***@EnableWebSecurity,*** para habilitar la seguridad web. Esto implicará que los logins de usuarios deben facilitarse cifrados, para lo que nos vemos en situación de obtenerlos mediante un clase adicional que nos facilite los passwords que se almacenan en la tabla de usuarios. En un gestor completo y funcional, este proceso lo desarrollaría la interfaz de creación de usuarios, lo que implicaría generar todo un CRUD para los usuarios que se prescinde de implementar, ya que no aporta en el proyecto; simplemente daremos de alta cuatro usuarios y generaremos sus passwords cifrados mediante una clase auxiliar *TestCrypt*, que devolverá por consola el usuario cifrado, que se añade manualmente a la base de datos.

En las clases de los controladores se activa mediante las anotaciones ***@EnableGlobalMethodSecurity()*** y ***@Preauthorize*** el acceso a determinados métodos únicamente en caso de la presencia del rol indicado en el usuario autenticado, de esta manera, podemos facilitar el acceso a la creación de tickets únicamente a los usuarios autenticados con el rol de administrador.

```
//crear un ticket nuevo
@PreAuthorize("hasRole('ROLE_N1')")
@GetMapping("/ticketform")
public String redirectTicketForm (Model model) throws IOException {
    TicketModel ticket = new TicketModel();
    model.addAttribute("ticketModel", ticket);
    return "ticketform.html";
}
```

Figura 5.19: Método solo autorizado a roles tipo administrador

5.4. Virtualización del sistema en contenedores Docker

En el momento en que el gestor de incidencias está validado en un entorno local, es necesario empaquetar su funcionalidad en un archivo .jar que sea factible de despliegue en un entorno virtualizado, utilizando contenedores Docker.

A modo de introducción rápida, tenemos que definir ciertos términos para entender mejor la tecnología que estamos utilizando.

- **Dockerfile e imágenes docker**
  Dockerfile es un archivo de configuración que se utiliza para crear imágenes. En dicho archivo indicamos qué es lo que queremos que tenga la imagen, y los distintos comandos para instalar las herramientas.

  Ejecutando el comando *docker build* sobre ese DockerFile, se creará la imagen correspondiente, lista para desplegar un contenedor.

  Muchas empresas de software ponen a disposición de los usuarios imágenes de sus productos para acceso libre; de esta manera, podemos descargar dichas imágenes y generar los contenedores necesarios para operar en nuestra aplicación de manera mucho más rápida.

- **Contenedores**
  Son instancias en ejecución de una imagen. A partir de una única imagen, podemos ejecutar varios contenedores.

- **Volúmenes**
  Los volúmenes suponen el modo en el que Docker permite persistir los datos de una aplicación. Los datos se alojan fuera del propio contenedor, en el sistema de archivos del host donde está corriendo Docker, de tal manera que se puede cambiar, apagar o borrar el contenedor sin que afecte a los datos almacenados en el volumen. Hay que tener en cuenta que estos volúmenes persisten ante un borrado del contenedor, no se dispone de un "recolector de basura" que limpie volúmenes huérfanos de contenedores eliminados, se debe gestionar el espacio adecuadamente a la hora de usar esta funcionalidad.

Para la instalación de Docker en la imagen de Centos 7 se debe instalar previamente una serie de utilidades, siguiendo las recomendaciones de la propia página de Docker.[3]

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

El siguiente paso será añadir el repositorio oficial:

```
$ sudo yum-config-manager --add-repo https://download.docker.com
/linux/centos/docker-ce.repo
```

Una vez completados estos pasos, procedemos a instalar la versión de la comunidad (Docker Engine - Community), versión no de pago, con menos funcionalidades, pero perfectamente válida para nuestro proyecto.

[3]https://docs.docker.com/install/linux/docker-ce/centos/

```
$ sudo yum install docker-ce
```

Se debe añadir nuestro usuario al docker group:

```
$ sudo usermod -aG docker $(whoami)
```

Para completar la instalación, ejecutamos los comandos que habilitan e inicializan Docker:

```
$ sudo systemctl enable docker.service
$ sudo systemctl start docker.service
```

Si todo se ha completado correctamente, podremos crear nuestra primera imagen mediante el archivo de configuración Dockerfile. Para desplegar el jar contenedor del proyecto, se creará y ejecutará el siguiente archivo:

```
FROM java:8
LABEL maintainer="javier.val.barbeira@gmail.com"
EXPOSE 8080
COPY soporte-0.0.1-SNAPSHOT.jar soporte-0.0.1-SNAPSHOT.jar
ENTRYPOINT ["java","-jar","soporte-0.0.1-SNAPSHOT.jar"]
```

- El comando **From** es obligatorio al comienzo del archivo, indica cual es la imagen base que utilizaremos (imagen de java en versión 8).
- **Label** añade metadatos sobre la imagen, en le archivo se indica maintainer para señalar el creador de la imagen. Es un campo prescindible.
- **Expose** indica el *puerto de entrada:puerto de salida* (en este caso únicamente el puerto de entrada)
- **Copy** realiza la copia del archivo indicado desde la máquina local al contenedor
- **Entrypoint** indica el comando que se ejecutará en el contenedor una vez que se arranque este. En el proyecto se lanza el gestor mediante la ejecución java del archivo jar.

Hay que recordar que hemos modificado el gestor para que no busque la base de datos Mysql en local, si no que esta también se ejecutará en un contenedor. La idea final es que el sistema recomendador también resida en contenedores, lo que plantea la necesidad de interconectar todos los contenedores mediante un sistema aglutinador. Para ello utilizaremos **Docker-compose**.

Compose es una herramienta para definir y ejecutar aplicaciones Docker de múltiples contenedores. Utiliza un archivo **docker-compose.yml** para configurar los servicios de su aplicación. Posteriormente, utilizando un solo comando, creará e iniciará todos los servicios desde su configuración.

Para su ejecución se debe previamente instalar ciertos paquetes extra.

```
$ sudo yum install epel-release
$ sudo yum install -y python-pip
```

Ahora sí, se procede con la instalación de Compose:

```
$ sudo pip install docker-compose
```

Para finalizar, se debe actualizar los paquetes de python, para que Compose se ejecute sin problemas:

```
$ sudo yum upgrade python*
```

Para arrancar los contenedores, debemos gestionar la configuración del archivo **docker-compose.yml**. Una vez finalizado, se lanza la ejecución mediante el comando *docker-compose up*. El archivo yml quedará como se puede apreciar en la siguiente imagen.

```yaml
version: '3'
services:

  mysql:
    image: mysql:8.0.17
    container_name: mysql-docker-container
    environment:
      - MYSQL_ROOT_PASSWORD=pw3358pec
      - MYSQL_DATABASE=mydb
      - MYSQL_USER=app_user
      - MYSQL_PASSWORD=test123
    volumes:
      - /data/mysql
    networks:
      - esnet

  spring:
    container_name: spring
    build: ./Spring/
    depends_on:
      - mysql
    ports:
      - 8087:8080
    volumes:
      - /data/spring-boot-app
    networks:
      - esnet


networks:
  esnet:

volumes:
  mysql:
    driver: local
  spring:
    driver: local

```

Figura 5.20: Archivo docker-compose.yml

Compose se basa en python y YAML, que es un lenguaje de marcado ligero basado en la indentación, por lo que el archivo yml debe estar correctamente formateado. Un tabulado con fallos puede derivar en errores al compilar no fácilmente reconocibles.

Al comienzo del archivo, el indicativo de la versión de sintaxis, a continuación, la sección de servicios a ejecutar. Por ahora se ejecutarán unicamente dos servicios, mysql y spring. En el mismo orden de tabulación, vemos más adelante como señalamos las redes creadas. En el proyecto creamos la red **esnet** en la que estarán todos los servicios desplegados. También se determinan los volúmenes a generar, indicando el driver como local. De esta manera, los volúmenes creados serán accesibles únicamente por sus servicios (las configuraciones aplicables en los volúmenes son variadas, permitiendo volúmenes de datos compartidos, de solo lectura, etc.).

Volviendo a los servicios creados, nótese que desde Compose, tenemos las mismas funcionalidades que un archivo Dockerfile. Así, en el servicio mysql, se utiliza la imagen base de mysql, version 8.0.17, creando un contenedor con el nombre **mysql-docker-container**. Mediante las modificaciones de variables de entorno, se indica la configuración de la base de datos. El siguiente comando determina la ruta de creación del volumen en el ordenador host, así como la red en la que estará disponible el servicio.

Para el servicio de spring, se nombra el contenedor como "spring", aplicando el Dockerfile creado anteriormente, almacenado dentro de la carpeta Spring utilizando la orden *build*. Debemos prestar atención al comando *depends_on*, que determina una dependencia entre servicios. Hasta que el servicio indicado, mysql, no esté levantado, no se procederá a levantar el servicio spring. Esto no implica que el servicio esté operativo, simplemente que esté levantado por Docker. Con el comando ports se indican los puertos de entrada y salida que mapeará el contenedor. De igual manera que en el servicio anterior, indicamos ruta para el volumen de persistencia, y la red en la que se desplegará el contenedor.

5.4.1. Adaptación del gestor para ejecución en contenedores

El primer paso a realizar es la modificación del archivo **application.properties** para desvincular la conexión local del gestor y añadir la nueva localización de la base de datos, que ahora será ejecutada desde un contenedor de Docker. Se debe modificar la linea que apunta al servicio en local:

```
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
```

cambiando la dirección del jdbc, para que apunte ahora al contenedor del servicio mysql:

```
spring.datasource.url =jdbc:mysql://mysql-docker-container:3306/
spring_app_db?useSSL=false
```

Debido a que Maven se continúa ejecutando en entorno local, no en la red gestada con docker-compose, a la hora de realizar el empaquetado con el comando *mvn package*, fallará al no poder realizar los test de conexión previos a lanzar el empaquetado. Se debe ejecutar el comando *mvn package -DskipTests* para que el empaquetado finalice sin problemas.
Accediendo a la ruta del workplace, encontraremos en archivo .jar contenedor del gestor de tickets, listo para ser desplegado en el contenedor java. Se puede ejecutar el jar autocontenido mediante el comando siguiente:

```
java -jar target/soporte-0.0.1-SNAPSHOT.jar
```

5.5. Módulo recomendador de soluciones

Una vez que el gestor de incidencias está operativo y "dockerizado" se procede a preparar el sistema de recomendación. Para ello, se crean tres contenedores que almacenen cada uno de ellos uno de los componentes del stack ELK (Elasticsearch, logstash y Kibana).

5.5.1. Cluster Elasticsearch

Comenzaremos creando el Dockerfile de Elasticsearch, y añadiremos la configuración necesaria al archivo **docker-compose.yml** para que se lance con el resto de contenedores.

El archivo Dockerfile del contenedor elastic será muy básico, ya que la mayoría de las configuraciones se aplican desde el archivo docker-compose.

```
FROM docker.elastic.co/elasticsearch/elasticsearch-oss:6.2.4
COPY ./config /usr/share/elasticsearch/config/
COPY ./plugins /usr/share/elasticsearch/plugins/
```

Se direcciona la ruta de la imagen base, señalando la versión concreta a desplegar. Es importante utilizar idénticas versiones para todo el stack ELK. Con el comando copy copiaremos de nuestra estructura local las carpetas de configuración y plugins necesarios. Para el contenedor no se añade ningún plugin esencial, pero sí un archivo de configuración para la memoria de la máquina virtual de java.
En el archivo docker-compose se añaden dos nuevos servicios, correspondiéndose con los dos nodos que tendrá nuestro cluster Elasticsearch. Se genera un cluster de dos nodos únicamente, para no sobrecargar la necesidad de memoria del proyecto. Con dos nodos se demuestra la escalabilidad del proceso, al poder distribuir los shards[4] del índice que más adelante se reparte entre cada uno de los nodos.

[4]Shard es la unidad en la que Elasticsearch distribuye datos en el cluster

```
elasticsearch:
  build: ./Elastic/
  #para la construccion de la imagen, debe acceder a la
  container_name: elasticsearch:nodo01
  environment:
    - node.name=elasticsearch
    - cluster.name=docker-cluster
    - bootstrap.memory_lock=true
    - discovery.zen.ping.unicast.hosts=elasticsearch2
  ulimits:
    memlock:
      soft: -1
      hard: -1
  volumes:
    - elastic01:/usr/share/elasticsearch/data
  ports:
    - 9200:9200
    - 9300:9300
  networks:
    - esnet
```

Figura 5.21: Detalle del servicio Elasticsearch en el archivo docker-compose

Se indica, como se había comentado con anterioridad, nombre del servicio, ubicación del Dockerfile de construcción de la imagen, nombre del contenedor y en el apartado de enviroment las siguientes configuraciones para permitir la comunicación entre contenedores y generar el cluster.

- node.name=elasticsearch
  Un nodo se puede unir a un cluster en el momento en que comparte su nombre de nodo (por defecto el nombre es Elasticsearch). Cuando se genere la configuración del siguiente nodo, se deberá añadir un nombre diferente.

- cluster.name=docker-cluster
  Indicamos el nombre del cluster bajo el que se unirán todos los nodos.

- bootstrap.memory_lock=true
  El rendimiento de Elasticsearch no es óptimo cuando el sistema intercambia la memoria. Mediante este comando elastic bloqueará el *process address space*.

- discovery.zen.ping.unicast.hosts=elasticsearch2
  Mediante este comando se determina un array de nodos a buscar, para la formación del cluster. En el proyecto únicamente buscará a su nodo hermano.

También se aprecia el hecho de no establecer limitaciones de memoria para los nodos, mediante la configuración *memlock soft hard* a -1. Como en otros servicios, se

asignan volúmenes de persistencia, puertos de comunicación y red de despliegue.

Para el segundo nodo, se replica la configuración, cambiando nombre de servicio y contenedor. En las configuraciones de enviroment, el nombre de nodo coincidirá con el determinado en la parametrización de búsqueda (discovery.zen.ping.unicast.hosts) del servicio anterior. El nombre de cluster será idéntico, para unirse al mismo cluster. El parámetro de búsqueda de nodo señalará al nodo hermano (elasticsearch). Los puertos de entrada serán necesariamente diferentes.

```
elasticsearch2:
  build: ./Elastic/
  container_name: elasticsearch2:nodo02
  environment:
    - node.name=elasticsearch2
    - cluster.name=docker-cluster
    - bootstrap.memory_lock=true
    - discovery.zen.ping.unicast.hosts=elasticsearch
  depends_on:
    - elasticsearch
  ulimits:
    memlock:
      soft: -1
      hard: -1
  volumes:
    - elastic02:/usr/share/elasticsearch/data
  ports:
    - 9201:9200
    - 9301:9300
  networks:
    - esnet
```

Figura 5.22: Detalle del servicio Elasticsearch2 en el archivo docker-compose

5.5.2. Contenedor Logstash

Para la preparación del contenedor encargado de la ingesta de datos, se crea un Dockerfile similar al utilizado en el contenedor de elastic.

```
FROM docker.elastic.co/logstash/logstash-oss:6.2.4
COPY ./config /usr/share/logstash/pipeline
COPY ./plugins /etc/logstash/plugins/
CMD tail -f /dev/null
```

Se indica la imagen base de partida. Recordemos que es recomendable que mantenga misma versión que los demás contenedores del stack. Mediante el comando copy se añade desde la carpeta local de plugins el conector necesario para acceder

a la base de datos, en el contenedor docker mysql se está desplegando la versión 8.0.17. por tanto se debe copiar este conector a la carpeta de plugins del contenedor Logstash (etc/logstash/plugins). También se copia el pipeline (órdenes de trabajo) a la carpeta de ejecución. Como paso final de este Dockerfile, se añade una ejecución "fantasma" que mantendrá el contenedor a la espera del comando a introducir para ejecutar el arranque del motor de ingesta de datos.

La configuración necesaria del docker-compose será la siguiente. Se indica ruta al Dockerfile para que se proceda a la generación, nombre del contenedor, volumen, puertos entrada:salida y red.

```
logstash:
  build: ./Logstash/
  container_name: logstash:6.2.4
  volumes:
   - logstash:/usr/share/logstash/data
  ports:
   - "5044:5044"
  networks:
   - esnet
```

Figura 5.23: Detalle del servicio Logstash en el archivo docker-compose

5.5.3. Contenedor Kibana

Para la creación de este contenedor, se lanza todos sus parámetros desde el archivo docker-compose. Se indica imagen base en version 6.2.4, y se añade la dependencia con el servicio elasticsearch, así como volumen, puertos y red.

```
kibana:
  image: docker.elastic.co/kibana/kibana-oss:6.2.4
  container_name: kibana:6.2.4
  depends_on:
   - elasticsearch
  volumes:
   - kibana:/usr/share/kibana/data
  ports:
   - "5601:5601"
  networks:
   - esnet
```

Figura 5.24: Detalle del servicio kibana en el archivo docker-compose

5.5.4. Configuración del pipeline de Logstash

Logstash es un motor de procesamiento y recopilación de datos basado en plugins. Incluye una gran variedad de plugins que permiten configurarlo fácilmente para recopilar, procesar y reenviar datos en muchas arquitecturas diferentes.

El procesamiento se organiza en uno o más pipelines. En cada pipeline, uno o más plugins de entrada reciben o recopilan datos que luego se colocan en una cola interna. De manera predeterminada, esta es pequeña y se almacena en la memoria, pero puede configurarse para ser más amplia y permanecer en el disco para mejorar la confiabilidad y la persistencia.

Uno de los primeros pasos que realizaremos a la hora de poner en marcha el recomendador será la ingesta inicial de datos. Se accede a la base de datos del cliente para recopilar todos los tickets resueltos que posteriormente serán enviados a Elasticsearch para pasarlos por el analizador y almacenarlos en un índice distribuido.

Los hilos de procesamiento leen datos de la cola y los procesan a través de cualquier plugin de filtro configurado en secuencia. Logstash viene de fábrica con una gran cantidad de plugins destinados a tipos específicos de procesamiento, y así es como se analizan, procesan y enriquecen los datos.

Una vez que se procesan los datos, los hilos de procesamiento envían los datos a los plugins de salida correspondientes, que son los responsables de formatear y enviar los datos, en nuestro caso, a Elasticsearch.

Los plugins de entrada y salida también pueden tener un plugin de códec configurado. Esto permite formatear datos antes de colocarlos en la cola interna o antes de enviarlos a un plugin de salida.

La manera más fácil de arrancar Logstash es crear un único pipeline basado en un solo archivo de configuración. lanzar el motor de ingesta arrancando Logstash e indicando la ruta al archivo de configuración:

```
logstash -r -f /usr/share/logstash/pipeline/logstash.conf
```

En nuestro caso no utilizaremos plugins intermedios para procesar los datos, simplemente aplicaremos un plugin de entrada, para recopilar los datos de mysql y un plugin de salida para enviar estos a Elasticsearch.

```
input {
        jdbc {
                jdbc_connection_string => "jdbc:mysql://mysql-docker-container:3306/mydb"
                jdbc_user => "root"
                jdbc_password => "pw3358pec"
                jdbc_driver_library => "/etc/logstash/plugins/mysql-connector-java-5.1.48/mysql-connector-java-5.1.48-bin.jar"
                jdbc_driver_class => "com.mysql.jdbc.Driver"
                schedule => "* * * * *"
                statement => "SELECT * FROM tickets where state ='cerrado' and fecha_update > :sql_last_value order by fecha_update"
                # statement => "SELECT * FROM tickets where fecha_update > '2019-10-08 19:30:00' and state ='cerrado'"
                use_column_value => false
                record_last_run => true
                last_run_metadata_path => "/usr/share/logstash/logstash_jdbc_last_run_t_data.txt"
                # tracking_column => "fecha"
                tracking_column => "fecha_update"
                tracking_column_type => "timestamp"
                clean_run => true
        }
}
output {
        stdout { codec => json_lines }
        elasticsearch {
                "id" => "ticket_ETL"
                "hosts" => "elasticsearch:9200"
                "index" => "tick"
                "document_type" => "_doc"
                }
}
```

Figura 5.25: Configuración del pipeline de Logstash

El plugin de entrada será el conector jdbc, que se debe configurar con los siguientes parámetros:

- jdbc_connection_string
  Indica la ruta y puerto al contenedor mysql, así como el nombre de la base de datos.

- jdbc_user / jdbc_password
  Indica usuarios y password de conexión.

- jdbc_driver_library / jdbc_driver_class
  El plugin no posee el paquete de librerías JDBC, se debe determinar su ruta y la clase a cargar. Siguiendo una sintaxis similar a la de cron, se puede periodificar la ejecución del comando *statement* para lanzar la ingesta de datos. En el proyecto la configuración "* * * * *" equivale a una periodicidad de un minuto. Cada minuto logstash accederá a la base de datos y ejecutará el comando *statement*. Para una ingesta diaria de datos se aplicaría la configuracion "0 0 * * *" que lanzaría en el minuto cero de la hora cero de todos los días del año la sentencia *statement*.

- statement
  Es la sentencia que se ejecutará contra base de datos. Con una consulta como la siguiente, recuperaremos toda la información de aquellos tickets cerrados posteriores al 8 de octubre a las 19:30 horas.

```
"SELECT * FROM tickets where fecha_update > '2019-10-08 19:30:00'
 and state ='cerrado'"
```

Esto no sería un problema si únicamente lanzamos una carga por ejemplo mensual y vamos actualizando la fecha de este select con el paso del tiempo. Se presentaría un problema si se periodifica la carga sin alterar la fecha del select. Con cada nueva ejecución del pipeline estaríamos cargando datos duplicados.

Para solucionar esta eventualidad, utilizamos el parámetro *sql_last_value* dentro del select a ejecutar.

```
"SELECT * FROM tickets where state ='cerrado' and fecha_update >
 :sql_last_value order by fecha_update"
```

Para el correcto funcionamiento de *sql_last_value*, se deben añadir ciertos parámetros.

- last_run_metadata_path
  Ruta dentro del contenedor que albergará el archivo de metadata que almacena el valor sql_last_value.

- tracking_column
  Columna de la tabla sobre la que se realiza el seguimiento de continuidad, en nuestro caso "fecha_update".

- tracking_column_type
  Tipo de la columna sobre la que se realiza el seguimiento, en nuestro caso timestamp.

- record_last_run
  Activamos la opción de guardado en el archivo de metadata, de la última ejecución sql sobre dicha columna. En el archivo *logstash_jdbc_last_run_t_data.txt* se almacenará una anotación como esta:

  ```
  --- 2019-12-08 22:19:00.286698000 Z
  ```

  La siguiente vez que se ejecute el pipeline, se cargarán aquellos tickets con fecha posterior a la indicada en el metadata.

- clean_run
  Esta parametrización activada a *true* implica el reseteo del archivo metadata a fecha 1 Enero 1970, lo que implica que se cargarán todos los tickets desde el inicio. En el ejercicio permanece activa por motivos de estudio de carga completa, pero en un pìpeline de producción deberá conmutarse a *false*.

Los plugins de salida serán stdout utilizando el codec jason_lines por motivos de debugging (permite ver por consola como se van enviando los datos recuperados) y el plugin Elasticsearch para enviar los tickets al índice elastic.

La parametrización del plugin de salida Elasticsearch es la siguiente:

- id
  Se indica el id del índice a ingestar.
- hosts
  Nombre del host que contactaremos para encontrar el índice. Se puede indicar un array para balancear la entrada, evitando apuntar a los *dedicated master nodes*[5] si es que utilizamos esta configuración para controlar el cluster.
- index
  Nombre del índice a poblar.
- document_type
  Esta opción está marcada para su desaparición. Indica el tipo de documento en el que se escribirán los eventos.

En el sistema, una vez que tenemos copiada la configuración del pipeline durante la creación del contenedor, lanzaremos la ingesta de datos accediendo a la carpeta de binarios dentro del contenedor Logstash, y ejecutando la aplicación indicando el pipeline adecuado.

```
docker exec -it contenedor_id /bin/bash
cd /usr/share/logstash/bin
logstash -r -f /usr/share/logstash/pipeline/logstash.conf
```

5.5.5. Creación del índice

Antes de lanzar la ingesta de datos, es importante crear y configurar el índice que se encargará de mapear y almacenar toda la información proveniente de la base de datos de tickets. En nuestro sistema aplicaremos las configuraciones por defecto a la hora de crear un índice, en lo referente al número de shards, verificación de integridad, codificación y compresión de datos, número de réplicas, asignación de shards a nodos, etc. Prestaremos especial atención al proceso de análisis, el proceso de conversión del input de entrada en tokens o términos que serán añadidos al índice de búsqueda. Partiendo del analizador por defecto, se crea un analizador específico, adaptado al sistema de incidencias del cliente.

[5]Un dedicated master node en contraposición a un data node en Elasticsearch son aquellos nodos no dedicados a tareas de almacenamiento de datos si no a tareas de administración y estabilidad de un cluster.

Creación del analizador

Los analizadores pueden ser aplicados tanto a la hora del indexado, como a la hora de la búsqueda. En el sistema recomendador se incide en personalizar el analizador de indexado, aplicándolo en ambos procesos para optimizar los resultados.

Elasticsearch ofrece en su configuración un abanico de analizadores a nuestra disposición, más allá del analizador estándar. También ofrece la posibilidad de crear y customizar un analizador, conociendo su estructura.

La anatomía base de un analizador está formada por tres elementos. Los **filtros de caracteres**, los **tokenizadores** y los **filtros sobre tokens**.

Filtros de caracteres
Un filtro de caracteres recibe un texto como un todo y es capaz de transformar este añadiendo, cambiando o eliminando caracteres. Por ejemplo, podemos cambiar caracteres no latinos de numeración de un texto recibido por sus equivalentes latinos antes de almacenar el texto en el índice, o eliminar caracteres típicos de HTML como las marcas de etiquetas si hemos parseado una web y nos interesa limpiar el string recibido antes de almacenarlo. En nuestro sistema prescindiremos de configurar filtro de caracteres.

Tokenizadores
Un tokenizador recibe un texto y lo divide en tokens (elementos mínimos con sentido, usualmente palabras). Elasticsearch proporciona multitud de tokenizadores, por ejemplo whitespace, que divide el texto recibido por espacios en blanco, o letter, que genera un token en el momento que detecta un caracter no alfabético.

Utilizaremos el tokenizador estándar que realiza la división de tokens en base a la especificación del Unicode Standard Annex #29[6]

Desde la consola de Kibana, podemos lanzar un análisis de un texto cualquiera, para observar el funcionamiento de un tokenizador estándar

[6]http://unicode.org/reports/tr29/

```
POST _analyze
{
  "tokenizer": "standard",
  "text": "Resultado de un tokenizado estandar."
}
```

(a) Análisis estándar

```
{
  "tokens": [
    {
      "token": "Resultado",
      "start_offset": 0,
      "end_offset": 9,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
      "token": "de",
      "start_offset": 10,
      "end_offset": 12,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "un",
      "start_offset": 13,
      "end_offset": 15,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "tokenizado",
      "start_offset": 16,
      "end_offset": 26,
      "type": "<ALPHANUM>",
      "position": 3
    },
    {
      "token": "estandar",
      "start_offset": 27,
      "end_offset": 35,
      "type": "<ALPHANUM>",
      "position": 4
    }
  ]
}
```

(b) División en tokens

Figura 5.26: Análisis de un texto y su tokenizado

Es habitual combinar los tokenizadores con los filtros tokenizadores para obtener el resultado buscado a la hora de realizar el análisis.

Filtros sobre tokens

Un filtro aplicado a un token permite realizar modificaciones (por ejemplo, pasarlo a minúsculas), eliminaciones (suprimir las denominadas stopwords) o añadir tokens nuevos (utilizando sinónimos). Trabajaremos y configuraremos los siguientes filtros que actuarán sobre la salida del tokenizador estandar:

- **Filtro lowercase**
  Normaliza los tokens transformándolos a minúsculas.
- **Filtro asciifolding**
  Transforma aquellos caracteres no pertenecientes a "Basic Latin" en su equivalente ASCII. Elimina acentos, diéresis, cedillas y los reduce a caracter latino, para facilitar el token.

- **Filtro default_english_stemmer**
  Este filtro permite reducir una palabra a su raíz (stem) para así aumentar la devolución de resultados potencialmente útiles en una búsqueda. Por ejemplo, una consulta sobre 'bibliotecas' también devolvería documentos en los que solo aparezca 'bibliotecario', porque el stem de las dos palabras es el mismo ('bibliotec'). El filtro stemmer utiliza el algoritmo más común para stemming, el algoritmo de Porter.[7]

  Se debe señalar que, en el sistema recomendador aplicamos el filtro stemmer indicando el idioma inglés como base del índice, asumiendo que todas las incidencias se reportan en ese idioma. En caso de detectar múltiples idiomas, se debería generar un índice diferente para cada idioma, tanto a la hora de analizar y almacenar los tokens, como a la hora de realizar las búsquedas. En esta situación, seguramente sería aplicable algún proceso de filtrado durante el trabajo de ingesta de datos en logstash, aplicando un filtro intermedio entre el input y el output, discriminando por idioma, y redireccionando el resultado a un índice u otro.

- **Filtro adapted_english_stopwords**
  Es un filtro personalizado sobre el filtro stop words inglesas. Un filtro de tipo stop elimina determinados tokens presentes en una lista preconfigurada de tokens. Esta lista puede ser ampliada o reducida a nuestras necesidades. Elasticsearch tiene predefinidos multitud de lenguajes sobre los que aplica stop words, en nuestro filtro partimos de la matriz de tokens ingleses siguiente:

  ```
  a, an, and, are, as, at, be, but, by, for, if, in, into, is,
  it, no, not, of, on, or, such, that, the, their, then, there,
  these, they, this, to, was, will, with
  ```

  Sobre esta lista se añaden determinados tokens que pueden aparecer en los asuntos del correo, que no aportan a la hora de hacer la búsqueda, como los indicadores de reenvío de correo, palabras como 'help me' o 'please', y pronombres como 'my'. Finalmente la lista de tokens quedará como sigue, siendo siempre abierta a añadir más tokens según se afinen los resultados de búsquedas en el sistema.

  ```
  [ "a", "an", "and", "are", "as", "at", "be", "but", "by", "for",
  "if", "in", "into", "is", "it", "or", "such", "that", "the",
  "their", "then", "there", "these", "they", "this", "to", "was",
  "will", "with", "re", "fw", "my", "i", "help", "me" ]
  ```

[7] http://snowball.tartarus.org/algorithms/porter/stemmer.html

La principal ventaja con la eliminación de estos tokens es el rendimiento. Elastic determina las recomendaciones a presentar calculando una puntuación de relevancia. No es lo mismo calcular la relevancia para 20 resultados que para 200, debido a que en estos resultados se incluyen todos los documentos que presenten el token RE debido a que simplemente el correo llega rebotado, o que el usuario, presa del pánico, incluye un 'help me please' en el asunto.

- **Filtro synonym**
  Al aplicar este filtro, parece que contradecimos el uso del filtro anterior, ya que añadimos resultados "adicionales" cuando determinado token se presenta en el texto a analizar. Esto es debido a que, en determinadas circunstancias, es interesante que estos resultados también sean devueltos para su ponderación ya que aportan significado. Así, si un usuario escribe "cosmos" podemos añadir los resultados que aparezcan relativos a "universe", si el usuario escribe "i pod" también añadiremos los resultados de "i-pod" e "ipod", siendo equivalentes en ambas direcciones.
  Para nuestro analizador, simplemente hemos añadido un pequeño grupo de sinónimos directamente en la especificación del filtro. Para un uso más intenso, podemos indicar la ruta a un archivo de sinónimos en la configuración el filtro. La lista aplicada de sinónimos es la siguiente:

```
"synonyms" : ["inet, internet","can't, can not","don't, do not"]
```

En nuestro sistema, debido a que la base de datos de incidencias encontrada estaba en inglés, se ha trabajado toda la configuración del analizador en este idioma. En caso de trabajar con incidencias en español, se debería prestar atención a un filtro a mayores, el filtro fonético, que permitiría reducir los tokens a su expresión fonética, para evitar así errores ortográficos a la hora de identificar tokens. Para activar este filtro, es necesario instalarlo previamente como un plugin adicional, y añadirlo a la lista de filtros. Esta solución tiende a multiplicar tokens, generando los sinónimos posibles. Por ejemplo 'barquito' devolvería resultados de 'varquito'. Otra opción para solucionar este problema, es utilizar búsqueda borrosa a la hora de componer la consulta desde el gestor de incidencias, pero esta opción se estudiará más adelante, cuando lleguemos a ese punto.

Si realizamos la siguiente consulta desde Kibana, aplicando nuestro analizador:

```
GET tick/_analyze
{
  "analyzer": "tick_analyzer",
  "text":  "RE: FW help me!! my credentials don't work"
}
```

(a) Análisis adaptado


```
{
  "tokens": [
    {
      "token": "credenti",
      "start_offset": 20,
      "end_offset": 31,
      "type": "<ALPHANUM>",
      "position": 5
    },
    {
      "token": "do",
      "start_offset": 32,
      "end_offset": 37,
      "type": "SYNONYM",
      "position": 6
    },
    {
      "token": "don't",
      "start_offset": 32,
      "end_offset": 37,
      "type": "<ALPHANUM>",
      "position": 6,
      "positionLength": 2
    },
    {
      "token": "not",
      "start_offset": 32,
      "end_offset": 37,
      "type": "SYNONYM",
      "position": 7
    },
    {
      "token": "work",
      "start_offset": 38,
      "end_offset": 42,
      "type": "<ALPHANUM>",
      "position": 8
    }
  ]
}
```


(b) División en tokens

Figura 5.27: Análisis de un texto mediante analizador configurado

Vemos cómo se han eliminado en el análisis, 'RE' 'FW' 'help' 'me' 'my', aún presentándose los dos primeros tokens en mayúsculas, ya que el filtro lowercase ha realizado su trabajo. El stemmer ha acortado el token a su raíz 'credenti'. El filtro de sinónimos se aplica también devolviendo resultados no apostrofados de "do not".

Mapeo de las entradas

Se utiliza el mapeo para definir de qué manera serán almacenados determinados campos en el índice y si se utilizará un analizador específico sobre ellos. En nuestro caso determinaremos dos campos como tipo text para facilitar las búsquedas de tipo full-text que serán la base de nuestro recomendador. El resto del mapeo se realizará de forma dinámica y auto detectada por elasticsearch.

5.5.6. Validación de búsquedas sobre Kibana

Una vez creados analizador e índice, y lanzada la ingesta de tickets desde logstash, es hora de realizar desde la consola de Kibana alguna consulta para validar el correcto funcionamiento del sistema.

Se lanza la siguiente búsqueda desde la consola de Kibana, componiendo un QueryDSL que devuelva los documentos almacenados en el índice que tengan coincidencia en el campo título con la sentencia 'RE: FW: help me!!'

```
GET /tick/_search
{
    "query": {
        "match" : {
            "title" : {
                "query" : "RE: FW: help me!!"
            }
        }
    }
    , "size": 30
}
```

(a) Consulta de búsqueda

```
{
  "took": 17,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 0,
    "max_score": null,
    "hits": []
  }
}
```

(b) Resultado

Figura 5.28: Consulta de búsqueda con la sentencia 'RE: FW: help me!!'

Se ha escaneado los 5 shards pero no existe coincidencia, debido a que los tokens 'RE', 'RF', 'help' o 'me' no existen, ya que no habían sido mapeados debido al filtro stop words.

Si completamos la consulta a buscar

```
"query" : "RE: FW: help me!! my credentials don't work"
```

se aprecia como ahora se devuelven 22 resultados coincidentes que serán priorizados en base a un **_score**, presentándose con un valor más alto aquellos resultados

con tokens coincidentes, en mismo orden y próximos entre sí.

Mediante el comando explain:true podemos validar los elementos analizados y el score asignado para completar la puntuación. En la imagen siguiente podemos observar el primer resultado devuelto

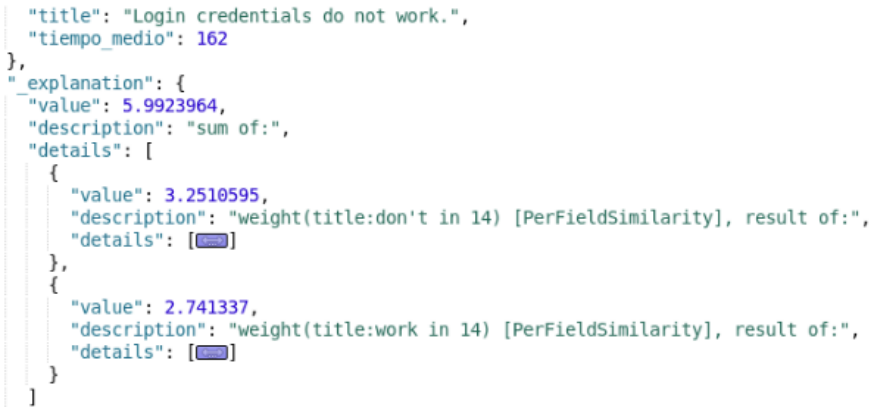


Figura 5.29: Primer resultado devuelto

Observamos cómo 3.25 puntos son debidos a la presencia de *don't* y 2.74 debidos al análisis de *work*. Tanto "my" como "credentials" no han devuelto score ya que no están presentes en el análisis, la primera por stopword y la segunda debido a que no existe el token en el índice.

Resultados con score menor únicamente hacen uso del token *work*, devolviendo incidencias que no tienen relación con el problema planteado por el ticket inicial.

Pero,¿por qué no ha analizado "credentials"? Recordemos que para la ingesta de datos hemos utilizado un analizador específico que hacía uso de un stemmer que reducía a su raíz los tokens, y en esta consulta no hemos especificado el uso de ese analizador. Al no determinar analizador, ha usado el estándar, que busca el token "credentials" y no lo encuentra.

Repitamos la consulta forzando el uso del analizador configurado:

```
GET /tick/_search
{
    "query": {
        "match" : {
            "title" : {
                "query" : "RE: FW: help me!! my credentials don't work",
                "analyzer" : "tick_analyzer"
            }
        }
    }
        , "size": 10
        ,"explain": true
}
```

Figura 5.30: Consulta, utilizando el analizador diseñado

El primer resultado devuelve una puntuación mucho más alta de score, debido a que reconoce el token “credenti” así como el sinónimo:


```
  "title": "Login credentials do not work.",
  "tiempo_medio": 162
},
"_explanation": {
  "value": 14.649335,
  "description": "sum of:",
  "details": [
    {
      "value": 3.2510595,
      "description": "weight(title:credenti in 15) [PerFieldSimilarity], result of:",
      "details": [
        {...}
      ]
    },
    {
      "value": 8.656939,
      "description": "sum of:",
      "details": [
        {
          "value": 5.4058795,
          "description": """weight(title:"do not" in 15) [PerFieldSimilarity], result of:""",
          "details": [...]
        },
        {
          "value": 3.2510595,
          "description": "weight(title:don't in 15) [PerFieldSimilarity], result of:",
          "details": [
            {...}
          ]
        }
      ]
    },
    {
      "value": 2.741337,
      "description": "weight(title:work in 15) [PerFieldSimilarity], result of:",
      "details": [
        {...}
      ]
    }
  ]
```


Figura 5.31: Primer resultado devuelto, utilizando el analizador diseñado

Pongamos un nuevo ejemplo, esta vez añadiendo búsqueda borrosa en la consulta.

```
GET /tick/_search
{
    "query": {
        "match" : {
            "title" : {
                "query" : "RE: FW: help me!! my credetials don't work",
        "analyzer" : "tick_analyzer",
        "fuzziness": "AUTO",
                "max_expansions": 50,
                "prefix_length": 0
            }
        }
    }
  ,"size" : 10
  ,"explain" : true
}
```

Figura 5.32: Consulta, utilizando en analizador diseñado y búsqueda borrosa

Nótese que aquí estamos simulando un error en la frase a buscar, eliminando la 'n' de credentials. Sin búsqueda borrosa, el analizador buscaría el token 'credeti' que no está almacenado, y no devolvería resultados sobre este token. En nuestro caso, al utilizar búsqueda borrosa, realizará variaciones sobre los tokens de la consulta en búsqueda de posibles resultados de tokens coincidentes.

```
      {
        "value": 2.132296,
        "description": "weight(title:credenti in 11) [PerFieldSimilarity], result of:",
        "details": [
          {...}
        ]
      }
    ]
  },
  {
    "value": 5.2970347,
    "description": "sum of:",
    "details": [
      {
        "value": 3.0744739,
        "description": """weight(title:"do not" in 11) [PerFieldSimilarity], result of:""",
        "details": [...]
      },
      {
        "value": 2.222561,
        "description": "weight(title:don't in 11) [PerFieldSimilarity], result of:",
        "details": [...]
      }
    ]
  },
  {
    "value": 1.4343047,
    "description": "weight(title:work in 11) [PerFieldSimilarity], result of:",
    "details": [...]
  }
]
```

Figura 5.33: La búsqueda borrosa soluciona el error de deletreo

El nivel de variaciones se puede determinar en función de un número considerable de configuraciones[8], siempre teniendo en cuenta la complejidad añadida a la búsqueda en términos de rendimiento. En nuestro ejemplo hemos mantenido

[8]https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-term-rewrite.html

borrosidad en los valores por defecto, en una consulta de producción buscaremos el punto de equilibrio entre rendimiento y calidad de los resultados devueltos.

Es importante afinar el analizador en base al contenido de nuestra base de datos de ingesta. Quizás sería recomendable añadir el término *don't* a la lista de stop words debido a la cantidad de resultados que puede añadir como "falsos positivos" que no aportan al recomendador soluciones precisas.

5.6. Integración en el gestor

Una vez validada la correcta operabilidad de ingesta, analizador e índice, vamos a preparar la consulta que se deberá implementar en el gestor de incidencias. El primer paso será añadir las dependencias necesarias al archivo **pom.xml** para que Maven se encargue de las librerías.

```
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
</dependency>
```

Figura 5.34: Dependencias cliente y Gson

La primera dependencia a descargar será la correspondiente al cliente java REST de alto nivel de Elasticsearch. Este nuevo cliente viene a substituir al cliente de transporte (TransportClient) anterior, que implicaba la gestión REST a base de peticiones generadas por nosotros. El nuevo cliente acepta todas las peticiones disponibles en la API de Elasticsearch.

GSON es una API en Java, desarrollada por Google, que se utiliza para convertir objetos Java a JSON (serialización) y JSON a objetos Java (deserialización). Nos facilitará el trabajo de presentación de la información devuelta por la consulta formada con el cliente de Elasticsearch.

Procederemos pues a crear un servicio, con un único método **listRecomendedTickets**(String titulo) que devolverá un List de ticketModel con las recomendaciones extraídas del índice elastic, utilizando el **RestHighLevelClient** en su implementación para realizar la conexión con el cluster elastic.

```
public List<TicketModel> listRecomendedTickets(String titulo) throws IOException {
    // -------------------------
    //Conexion con ElasticSearch
    RestHighLevelClient client = new RestHighLevelClient(
    RestClient.builder(new HttpHost("localhost", 9200, "http"), new HttpHost("localhost", 9300, "http")));
    List<TicketModel> ListaRecomendacionesRecibida = new ArrayList<TicketModel>();
    LOGGER.info("Cliente conectado. ");
    SearchRequest searchRequest = new SearchRequest("tick");
    //Se prepara un constructor de busqueda
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    QueryBuilder matchQueryBuilder = QueryBuilders.matchQuery("title",titulo)
      .fuzziness(2)
      .prefixLength(3)
      .maxExpansions(10)
      .analyzer("tick_analyzer");
      searchSourceBuilder.query(matchQueryBuilder);

    LOGGER.info("titulo a buscar: "+ titulo);
    searchSourceBuilder.from(0);
    searchSourceBuilder.size(5);
    searchRequest.source(searchSourceBuilder);
    SearchResponse searchResponse = null;
    try {
        searchResponse = client.search(searchRequest, RequestOptions.DEFAULT);
    }catch (IOException e) {
        e.printStackTrace();
    }
```

Figura 5.35: Conexión con el Cluster Elasticsearch

Creamos un nuevo **RestHighLevelClient** indicando el nombre de los contenedores y puertos de entrada. Para la gestión de la consulta, se crea una petición de búsqueda **SearchRequest** sobre el índice, en nuestro caso "tick". A continuación se prepara el constructor de búsqueda **SearchSourceBuilder** al que debemos indicar la consulta a lanzar. Para ello programaremos un **QueryBuilder**.

Disponemos de gran variedad de tipos de búsquedas, con sus constructores específicos. Por ejemplo *Match all query*, devuelve todos los resultados, no aplicable en nuestro caso. *Term-level query*, devuelve los documentos que contienen un valor exacto determinado en un term. Otros tipos podrían ser *Geo query, Span query,Joining query*, etc.

Para integrar nuestro sistema, utilizaremos las **Full Text query**, búsquedas específicamente diseñadas para textos. Más concretamente utilizaremos el tipo **Match**, que devolverá los documentos que coincidan con el texto enviado en la consulta. Este texto es analizado antes de realizar el match.

Algunas de las configuraciones que aplicamos en el constructor de la consulta son:

- *Fuzziness*
  Nos permite ajustar el nivel de borrosidad, la capacidad de alterar la posición de las letras en una palabra, para corregir potenciales errores de deletreo. La flotabilidad de las variaciones viene determinada por la Distancia

de Damerau-Levenshtein[9] (número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra) que indicaremos en la configuración del parámetro[10]. Como ya comentamos, esta configuración ha de tomarse con precaución y debe ser ensayada, ya que puede tornarse incluso en perjudicial para el rendimiento del sistema.

- *Analyzer*
  Podemos indicar el analizador que se utilizará para la consulta de búsqueda. Como se vio durante las pruebas en la consola de Kibana, es interesante indicar el mismo analizador que personalizamos para la indexación.

Una vez que tenemos listo el **QueryBuilder**, se lo indicaremos al SearchSourceBuilder.

Aquí también podemos señalar ciertas configuraciones interesantes, sobre todo la que determina cuántos resultados queremos cargar en el List de recomendados. Mediante el parámetro **size()** indicaremos que devuelva los cinco primeros resultados con mayor puntuación 'score'.

Finalmente indicaremos a la petición de búsqueda (searchRequest) su constructor de búsqueda (searchRequest).

Utilizando el cliente REST generado, se produce el **searchRequest** y se espera el **searchResponse**, que devolverá un Json que deberemos recorrer y transformar en objetos ticketModel con ayuda de los métodos proporcionados por Gson, para poblar el List que finalmente devolverá el método.

[9] https://en.wikipedia.org/wiki/Levenshtein_distance
[10] https://www.elastic.co/guide/en/elasticsearch/reference/6.2/common-options.html#fuzziness

```
// -------------------------
SearchHits searchHits = searchResponse.getHits();
Map<?, ?> hitMap;
JsonElement jsonElement;
TicketModel ticket_recomendado;
Gson gson=new Gson();

for(SearchHit hit : searchHits) {
    hitMap = hit.getSourceAsMap();
    jsonElement = gson.toJsonTree(hitMap);
    ticket_recomendado = gson.fromJson(jsonElement, TicketModel.class);
    LOGGER.info("titulo ticket recomendado: "+ ticket_recomendado.getTitle());
    LOGGER.info("solucion: "+ ticket_recomendado.getResolution());
    ListaRecomendacionesRecibida.add(ticket_recomendado);
}

client.close();
LOGGER.info("Cliente desconectado.");
//--------------------------------
return ListaRecomendacionesRecibida;
```

Figura 5.36: Transformación de elementos Json en objetos TicketModel

Este nuevo servicio implementado será llamado en el momento en que un técnico accede a revisar sus tickets asignados. Es por ello que debemos añadir en el controlador TicketController la inyección de dependencias para poder usar el servicio.

```
@Autowired
@Qualifier("elasticServiceImpl")
private ElasticService elasticService;
```

Figura 5.37: Inyección de dependencias del servicio Elastic

Así, el controlador poblará una nueva sección en la vista, en la que mostrará los tickets devueltos por el recomendador y almacenados en un atributo adicional del model.

```
model.addAttribute("ticketModel", ticket);
model.addAttribute("recomendaciones", elasticService.listRecomendedTickets(ticket.getTitle()));
model.addAttribute("recomendacion", recomendacion);
model.addAttribute("usa_recomendacion", usa_recomendacion);
```

Figura 5.38: Un nuevo atributo almacenará las recomendaciones

5.6.1. Visualización de recomendaciones

Para poder visualizar los resultados devueltos por el recomendador y almacenados como un atributo en el model, debemos crear una iteración que recorrerá

ese objeto List, mostrando id del ticket, título y resolución, así como un pequeño botón que al pulsar, añadirá la solución recomendada al campo resolución del ticket.

```
<tr th:each="rec : ${recomendaciones}">
    <td th:text="${rec.id}"></td>
    <td th:text="${rec.title}"></td>
    <td th:text="${rec.resolution}"></td>
    <td>
    <a href="#" th:href="@{/tickets/ticketin
        <button type="button" class="btn btn
        </a>
        </td>
</tr>
```

Figura 5.39: Iteración en la vista para obtener las recomendaciones

Podemos ver como quedaría el resultado final de la vista en la siguiente captura:

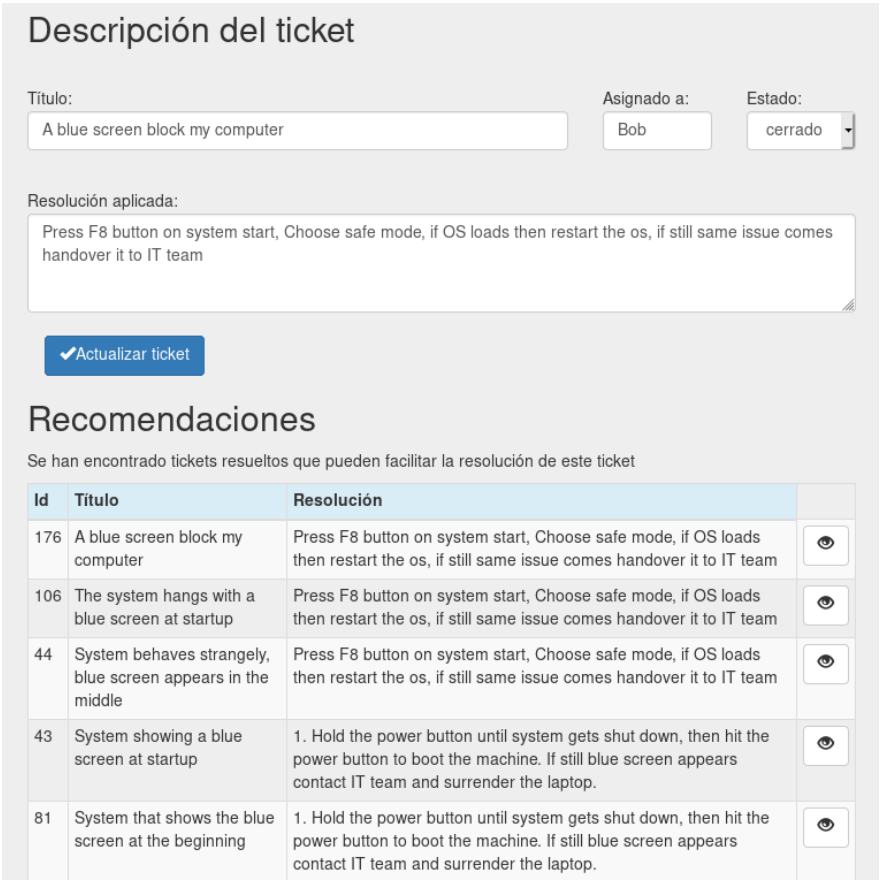


Figura 5.40: Ticket con recomendaciones de solución

5.6.2. Incorporación de la recomendación en el Crud

En el momento en que el técnico asigna la solución pulsando el botón de su derecha, poblará el campo 'resolution' y el atributo usa_recomendacion.

```
<a href="#" th:href="@{/tickets/ticketinfo?id=__${ticketModel.id}__&recomendacion=__${rec.resolution}__&usa_recomendacion=1}">
```

Figura 5.41: Botón de recomendación en la vista

Cuando cierre el ticket y lo actualice, la información se guardará en base de datos, marcando el ticket como cerrado y solucionado mediante el uso del servicio recomendador. Esto permitirá el seguimiento de la implantación del recomendador como sistema útil a la hora de solucionar y cerrar los tickets.
En el momento que el ticket se marca como cerrado, estará disponible para la próxima ingesta de datos en el índice elastic y, por tanto, pasará a ser una solución recomendada para futuras incidencias coincidentes.

5.7. Elaboración del panel de control de seguimiento

Para que el cliente pueda validar la implantación del sistema de recomendación, y aprovechando que tenemos disponible el contenedor con Kibana, procederemos a crear un panel de control con el que podrá visualizar gráficas de implantación, tiempos medios de resolución, tickets asignados a cada técnico, tickets cerrados mediante solucionador, o cualquier requerimiento de información que aporte utilidad a la hora de tomar decisiones por parte del cliente.

5.7.1. Acceso al índice desde Kibana

El primer paso será acceder a la sección **management** de Kibana, para indicar un modelo de índice que pueda reconocer elasticsearch como un índice de datos. En nuestro caso, elastic solo tiene un índice creado, y conocemos su nombre, así que indicamos el nombre **tick** y Kibana realiza la conexión sin problemas.

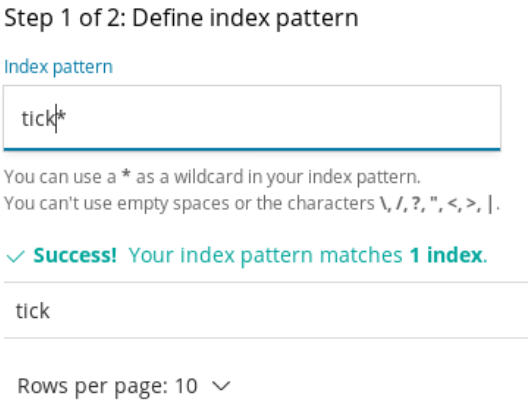


Figura 5.42: Reconocimiento del índice por Kibana

El siguiente paso será filtrar los datos por un valor temporal. En un sistema real, aquí sería interesante seleccionar la opción @timestamp si la ingesta de tickets cerrados se hace en tiempo real. En nuestro caso, hemos realizado una ingesta de datos instantánea (todos los datos han entrado en una única carga), lo que generaría un idéntico timestamp para la mayoría de los tickets, por tanto nos vemos forzados a seleccionar un campo fecha de los disponibles en los propios tickets, por ejemplo el fecha_close para tener una línea temporal con un rango de fechas utilizable para la visualización.

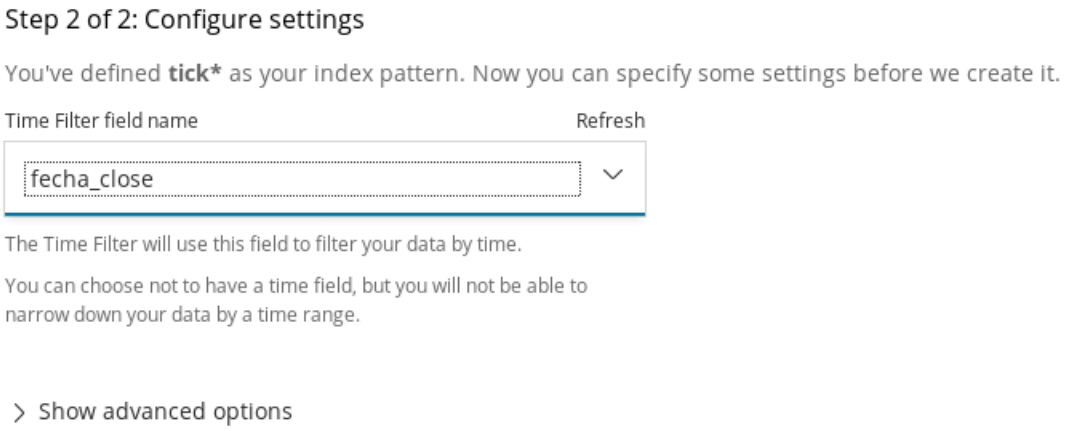


Figura 5.43: Filtrado por un campo temporal

Se nos mostrará entonces el resultado de los campos recogidos del índice, y si estos pueden ser utilizados para búsquedas o agregaciones.

name	type	format	searchable	aggregatable
@timestamp	date		✔	✔
@version	string		✔	
@version.keyword	string		✔	✔
_id	string		✔	✔
_index	string		✔	✔
_score	number			
_source	_source			
_type	string		✔	✔
designated	string		✔	
designated.keyword	string		✔	✔
fecha_close	date		✔	✔
fecha_open	date		✔	✔
fecha_update	date		✔	✔

Figura 5.44: Campos presentes en el índice

Si acudimos a la sección Discover, podremos visualizar nuestros datos, siempre y cuando previamente hayamos modificado las fechas de visualización, adaptándolas al período de datos cargados. Recordemos que en caso de utilizar una ingesta en tiempo real, y utilizando @timestamp, no sería necesario acudir a la variación de fechas de visualización, ya que podríamos ver en tiempo real la creación de datos en el índice.

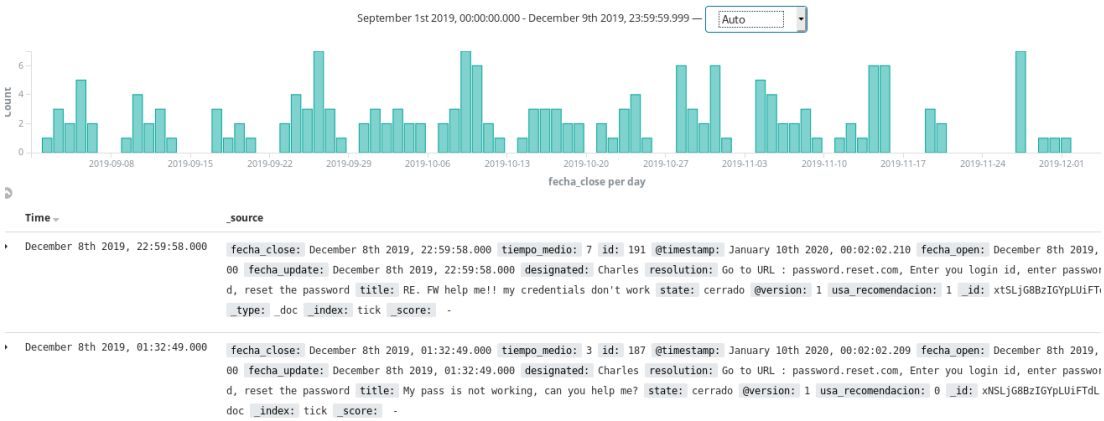


Figura 5.45: Timeline de datos ordenados por el campo filtrado

5.7.2. Creación de visualizaciones

El siguiente paso será la creación de las visualizaciones que iremos añadiendo al cuadro de mando. Las visualizaciones son distintos tipos de gráficas que podemos generar con Kibana. Para crear nuestra primera visualización, acudimos a la sección Visualize, y creamos una nueva visualización desde el botón Create a visualization. Debemos seleccionar el tipo: Basic Chart, Data, Maps o Time Series.

En nuestro caso realizaremos Basic Charts y Data. Comenzaremos seleccionando el clásico gráfico de tarta.

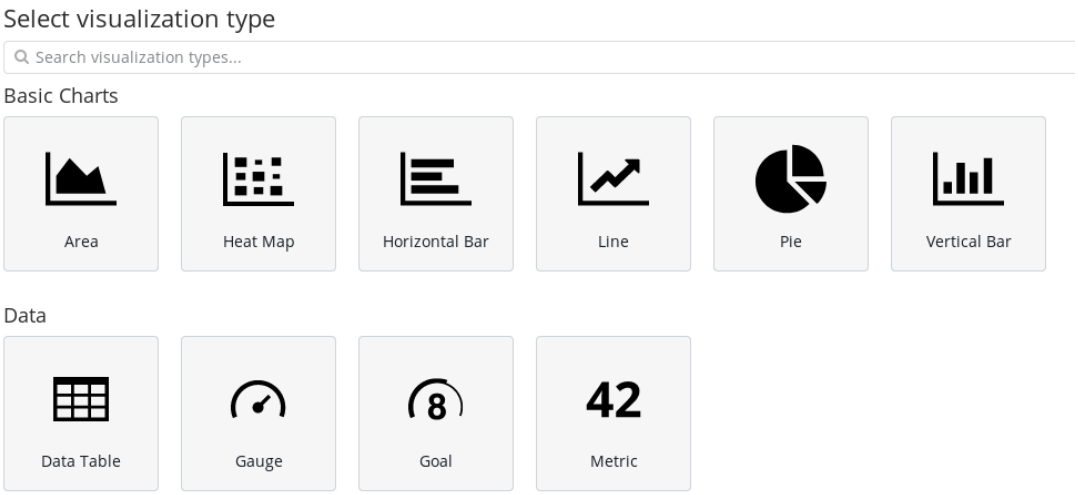
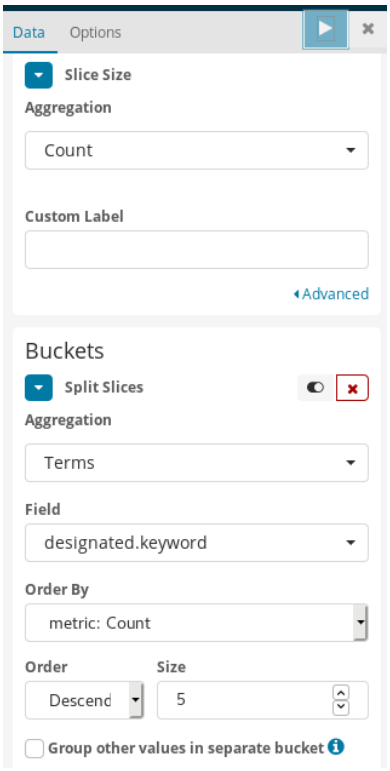


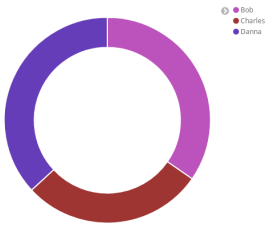
Figura 5.46: Tipos habituales de visualizaciones

Podemos crear el gráfico desde una búsqueda guardada, o componerlo desde el índice. En nuestro caso seleccionamos el índice.

Ahora debemos configurar la sección de métricas y buckets. Los buckets serán "cubos de datos" resultantes de la métrica selecionada. En nuestro ejemplo crearemos una métrica básica de tipo count, contaremos los tickets resueltos por cada técnico. En la sección Metrics seleccionamos Count y en la sección Buckets marcaremos **split slices**, definiendo la agregación en base a **terms**, seleccionando el campo **designated** (que indicaba el técnico al que se le asignaba el ticket). Ordenamos la representación por cantidad (no es muy influyente en este caso en la presentación de los datos) en orden descendente y mostrando como máximo 5 valores en la tarta (solo disponemos de tres técnicos). Una vez configurado, presionamos el icono de **play** y tenemos nuestra representación gráfica.



(a) Configuración de la visualización

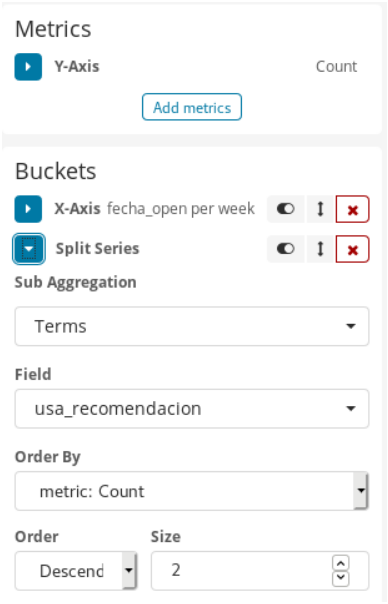


(b) Visualización de tickets asignados a cada técnico

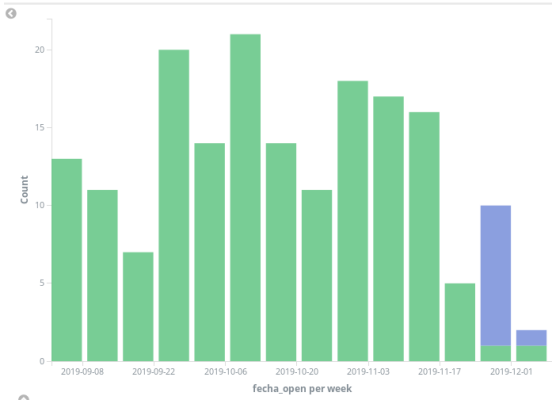
Figura 5.47: Visualización básica de tipo tarta

El paso final será guardar nuestra visualización con un nombre identificativo, cubriendo el campo Save visualization.

Una visualización con mayor aporte informativo para el gestor sería por ejemplo la cantidad de tickets que han aplicado el recomendador para su cierre. En la siguiente imagen podemos apreciar su configuración y gráfica.



(a) Configuración de la visualización



(b) Visualización

Figura 5.48: Visualización de tickets resueltos con recomendador

La gráfica muestra la cantidad de tickets resueltos por semana (marcados en verde) y cómo después de la implantación del sistema recomendador los tickets cerrados con su ayuda (color violeta) comienzan a tener protagonismo.

5.7.3. Composición del panel de control

Para facilitar la visualización al gestor, generaremos un único cuadro de mando que aglutinará las visualizaciones diseñadas. El acceso al panel se proporcionará desde un enlace independiente del área de trabajo de Kibana.

Accedemos a la sección 'dashboard' para crear un panel vacío. Mediante el botón Add añadiremos las visualizaciones, que seleccionamos de las ya generadas.

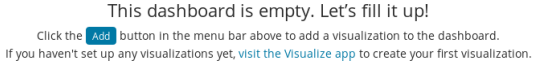


Figura 5.49: Un panel vacío necesita visualizaciones para mostrar

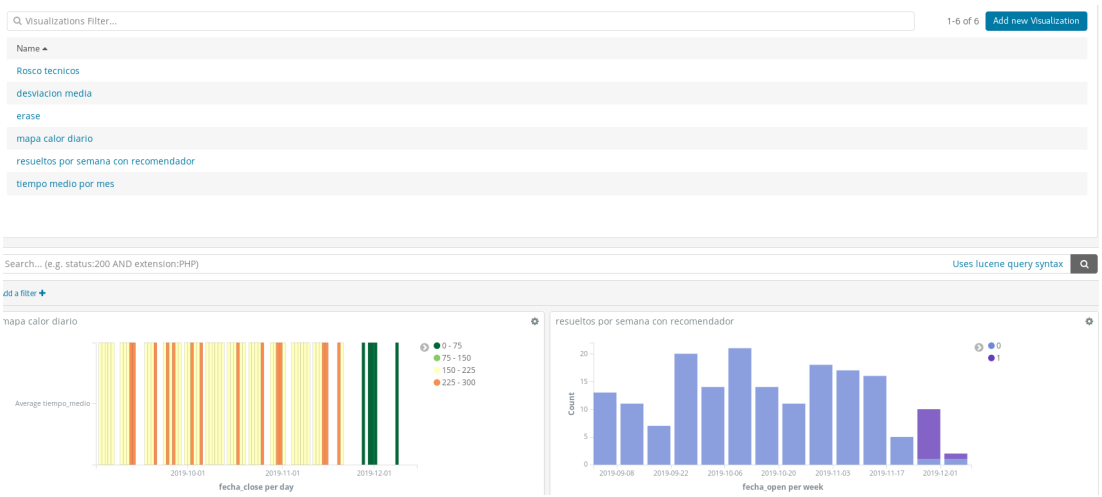


Figura 5.50: Seleccionamos las visualizaciones disponibles

Se guardará el panel creado con un nombre identificativo y se procede a seleccionar la opción compartir, para obtener el enlace que facilitar al gestor.

Figura 5.51: Desde la etiqueta Share obtenemos el enlace

Finalmente el gestor puede visualizar el panel sin mayor complicación.

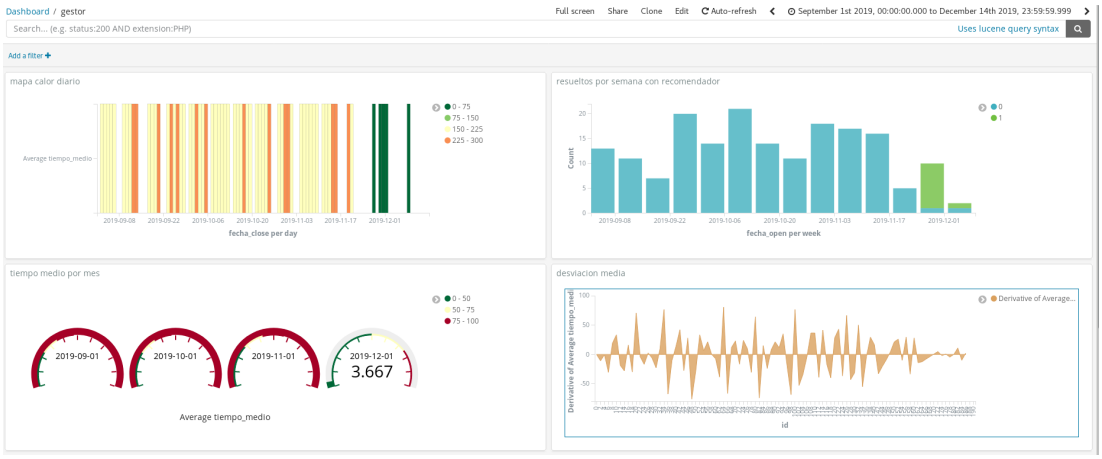


Figura 5.52: Panel facilitado al gestor

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Una vez finalizado el proyecto, podemos concluir que el objetivo inicial de este, el ofrecer una característica diferenciadora a la hora de competir en la consecución de un nueva oportunidad de negocio en un sistema de soporte es viable, fácilmente replicable y adaptable a diferentes clientes y con un coste reducido y escalable.

El sistema recomendador es capaz de encapsularse en contenedores fácilmente portables al sistema operativo del cliente. Es capaz de realizar una ingesta de datos del sistema cliente en caliente y de manera no intrusiva, sin necesidad de detener la base de datos cliente y sin bloquear tablas durante el proceso. El almacenamiento necesario para el sistema de recomendación es escalable, adaptando el coste a las necesidades y dimensión del sistema cliente, lo cual es crucial, ya que este servicio será sufragado por la empresa concursante, al ser ofrecido sin coste al cliente.

El cliente percibe este servicio, gracias al sistema de seguimiento de implantación, como una ventaja que permitirá reducir los tiempos de respuesta, lo que incrementará el grado de satisfacción de los usuarios.

Desde el punto de vista de la empresa que concursa, la principal ventaja es la portabilidad del sistema, la capacidad de ofrecer el servicio a diferentes clientes sin mayor esfuerzo que adaptar el plugin de acceso a la base de datos del cliente, y la consulta de extracción.

Bien es cierto que uno de los problemas que se encontrará la empresa será el mismo que nos encontramos nosotros a la hora de comenzar el proyecto, la integración en el sistema de incidencias del cliente. Este punto requiere como

base, que el cliente desarrolle su gestor de incidencias en un proyecto de código abierto[24, 23], al que poder acceder y modificar, lo cual sí puede suponer una barrera a la implantación (si nos encontramos con un gestor privativo[26, 25], que será lo más habitual) o un coste de desarrollo a asumir, si nos encontramos ante un gestor de código abierto, pero suficientemente complejo a la hora de integrar el recomendador en el visor de incidencias.

Una solución intermedia podría ser, y con ello se plantea un trabajo futuro, plantear el recomendador como una aplicación externa al gestor de incidencias, una aplicación de escritorio quizás, lo cual no se plantea tan "atractivo" para el cliente, pero sí mucho más fácil de adaptar y portar a otros clientes, desde el punto de vista de la empresa que concursa al servicio.

Desde el punto de vista del desarrollo del proyecto, varios puntos han de ser señalados. Uno de ellos fue el trabajo de búsqueda de material base para la creación de tickets. Algo que a priori se entendía casi como un trámite automático se tornó en un proceso colectivo de búsqueda por internet. Estos repositorios son fácilmente localizables cuando desarrollamos con big data, lo que hizo suponer que con igual facilidad se encontrarían repositorios de incidencias. Nada más lejos de la realidad. Se hizo necesario un tratamiento de limpieza y completado de los datos hasta obtener una base de trabajo relativamente válida para el proyecto, lo cual dilató los tiempos necesarios para completar el sistema.

Otro punto señalable fue la necesidad de construir de cero un gestor de incidencias. Una pequeña losa que mermó el tiempo disponible para ahondar en configuraciones y pruebas del recomendador. Una decisión ponderada, una vez visto el esfuerzo que habría supuesto atacar la modificación del código de uno de los sistemas open source disponibles.

6.2. Trabajo futuro

Muchas ideas se han quedado en el tintero, según se avanzaba en el proyecto.

A la hora de gestar el cluster Elasticsearch se optó por mantener configuraciones en modo automático o por defecto, sabiendo que únicamente se había asomado la cabeza a un mundo de configuraciones y optimizaciones. Hay una cantidad importante de material de estudio en lo referente a la arquitectura de Elastic, la configuración de los diferentes tipos de nodos disponibles (master, master-eligible, data, client, tribe, etc.) determinan un mar de posibilidades que deberían ser estudiadas.

La base de datos de tickets es a toda vista escasa. A la hora de trabajar la

configuración que determine un punto de equilibrio entre rendimiento y calidad de resultado, no podíamos apreciar diferencias. En su momento se contactó con los servicios informáticos de la USC, para interesarse por el sistema de soporte implantado, el cual es gestionado por una empresa externa a la universidad, lo cual dificultó el paso de solicitar un backup anonimizado de tickets reales. Por otro lado, consultas realizadas con el personal encargado (técnicos de soporte) nos desalentaron sobre la calidad del servicio de repositorio de incidencias, ya que no se solían detallar las soluciones al cerrar los tickets. Algo que puede ser un punto problemático a la hora de implantar nuestro sistema, y que requeriría un compromiso por parte de los técnicos implicados. En cualquier caso, un trabajo futuro debería plantear un conjunto de datos (dataset) más amplio sobre el que trabajar.

Una posible rama de trabajo surgió a la hora de realizar la ingesta de tickets. ¿Qué pasaría si nuestro gestor de incidencias da soporte a más de un país, si nuestros tickets se reciben en multitud de idiomas?. Se plantean soluciones rápidas, como la creación de un índice por idioma, con su analizador específico, pero entonces, ¿cómo discriminamos el idioma? ¿Quizás desde logstash[28] con un plugin de tratamiento?,¿desde el propio índice?. Un pequeño acercamiento al tema nos ofrece una posible solución desde el proceso de mapeo del índice, aplicando plugins de mapeo para detección de lenguajes, una rama interesante por la que se podría seguir investigando.

Otra interesante opción de estudio sería la rama de aprendizaje autónomo. Podríamos investigar la posibilidad de solución automática de aquellas incidencias básicas que se repiten constantemente, por ejemplo un bloqueo de login por 3 fallos consecutivos en el acceso. Estas incidencias podrían ser reconocidas por un sistema de aprendizaje autónomo que podría dar solución instantánea al ticket. Elasticsearch ofrece soluciones para la detección de anomalías y valores atípicos, pronóstico con base en tendencias e identificación de áreas de interés en los datos aplicando aprendizaje automático de Elastic[27].

Un punto interesante a tratar surgió de la visita al CITIUS, durante su tercera edición de 'Ciencia Singular'[1] atendiendo a la charla divulgativa del Doctor Gamallo Otero[29] sobre procesamiento de lenguaje natural y extracción de información. Una vez finalizada la charla y al corriente del proyecto, planteó el uso de **transformers bert**[30] para el procesado de lenguaje natural para proporcionar una funcionalidad al sistema más allá de una simple búsqueda de patrones coincidentes, si no el entendimiento de la necesidad planteada por la incidencia en sí, la comprensión del lenguaje natural, para ofrecer una respuesta 100 % efectiva.

[1] https://cienciasingular.usc.es/

Apéndice A

Esquemas EDT y Gantt

A.0.1. Estructura de Descomposición del Trabajo (EDT)

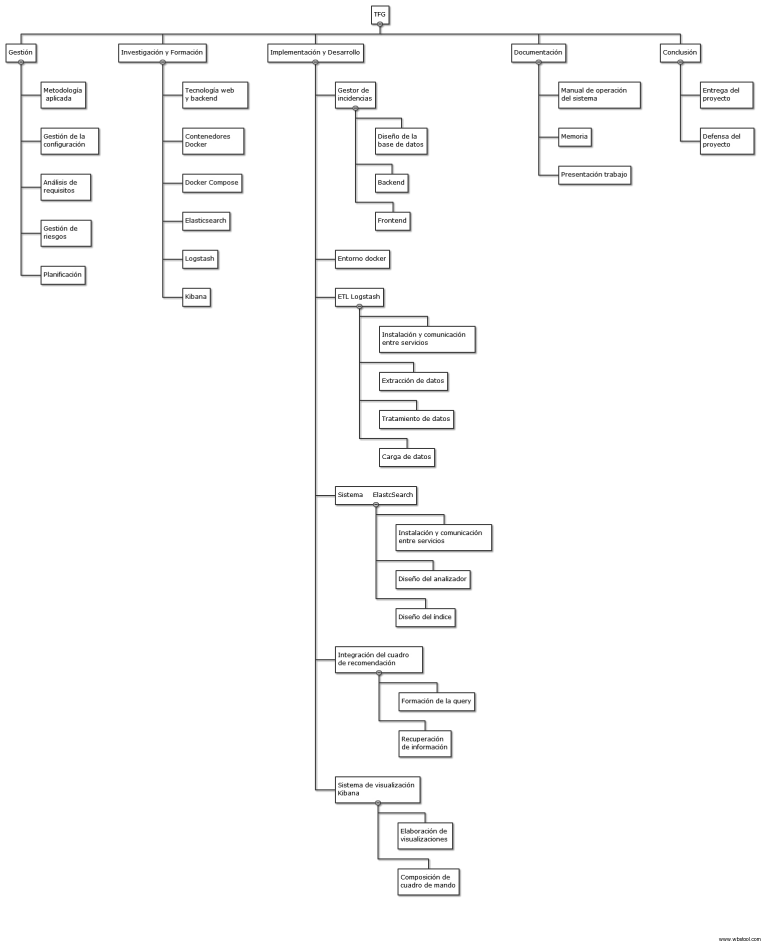


Figura A.1: Estructura de descomposición de trabajo

A.0.2. Esquemas Gantt

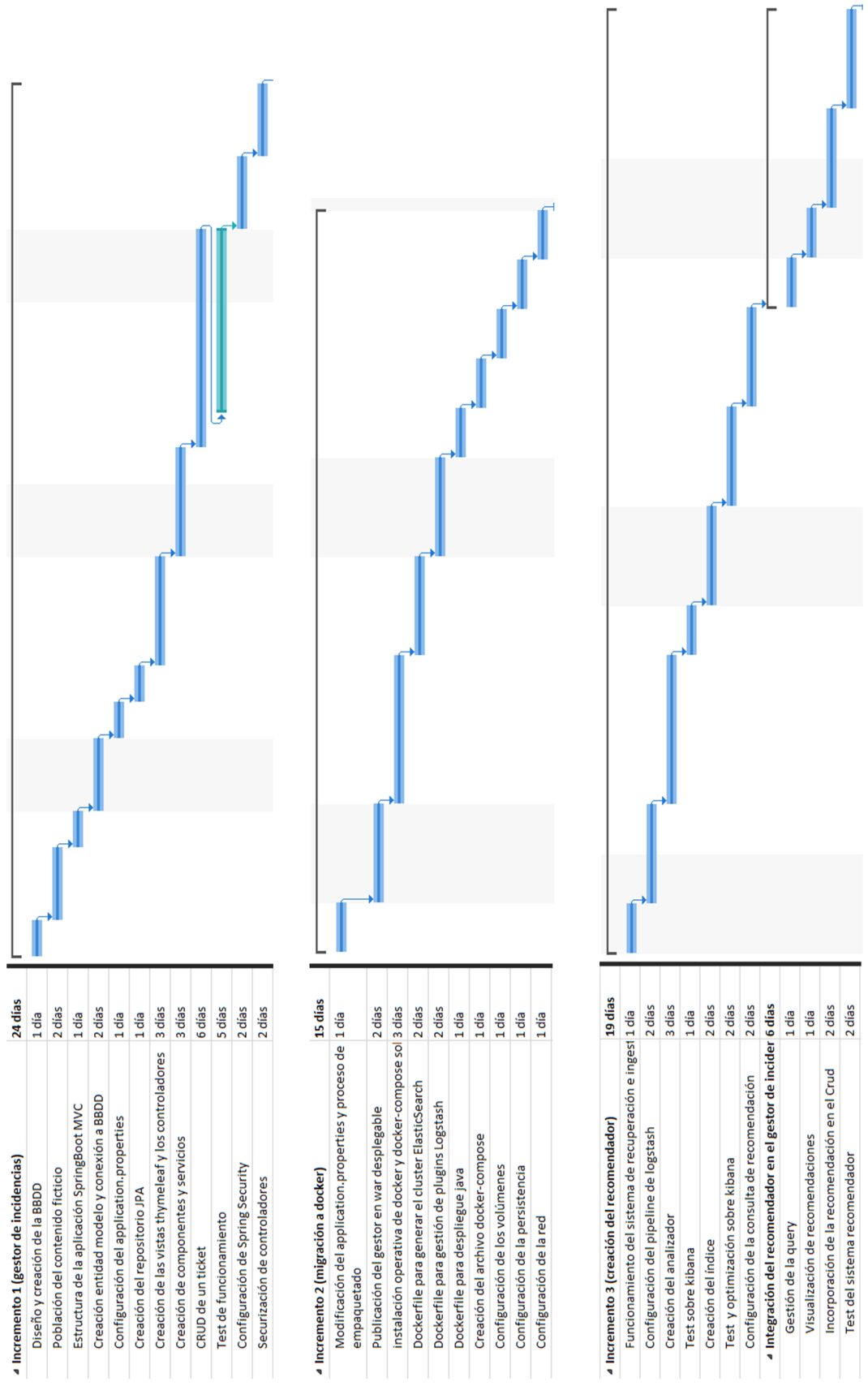


Figura A.2: Diagramas de Gantt correspondientes a los tres primeros incrementos

Incremento 4 (elaboración de un panel de control)	6 días
Acceso al índice	1 día
Creación de visualizaciones	4 días
Composición del panel de control	1 día
Test de usabilidad	1 día

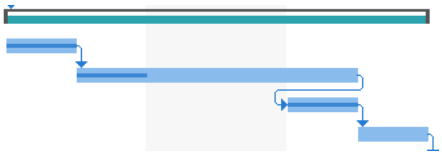


Figura A.3: Diagrama de Gantt correspondiente al cuarto incremento

Apéndice B

Manual de despliegue

A continuación se describe el proceso de puesta en funcionamiento de la aplicación, partiendo del material suministrado. Como requerimiento previo, el sistema operativo anfitrión debe tener instalado de manera satisfactoria las aplicaciones Docker y Docker-compose.

B.1. Despliegue

Se describe el proceso de despliegue sobre una maquina virtual Centos 7 con instalación completada de Docker y Docker-compose

1. El primer paso consistirá en copiar a nuestro sistema anfitrión la estructura de carpetas 'TICKER' suministrada. Dentro de esta encontraremos todos los archivos de configuración necesarios para desplegar el sistema.

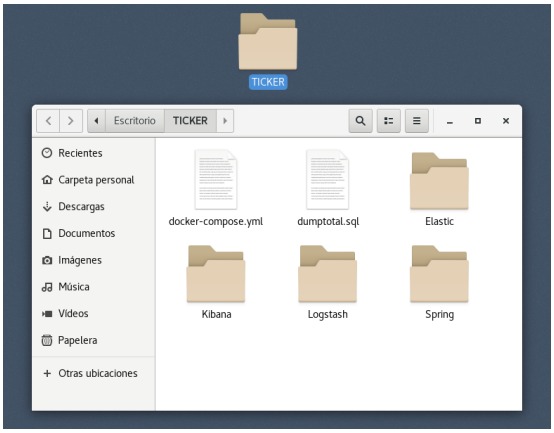


Figura B.1: Estructura de carpetas TICKER.

2. Una vez dentro de ella, ejecutamos el comando que levantará todos los contenedores Docker necesarios. Apreciamos como se van descargando las imágenes, creando los volúmenes y contenedores.

   ```
   docker-compose up
   ```

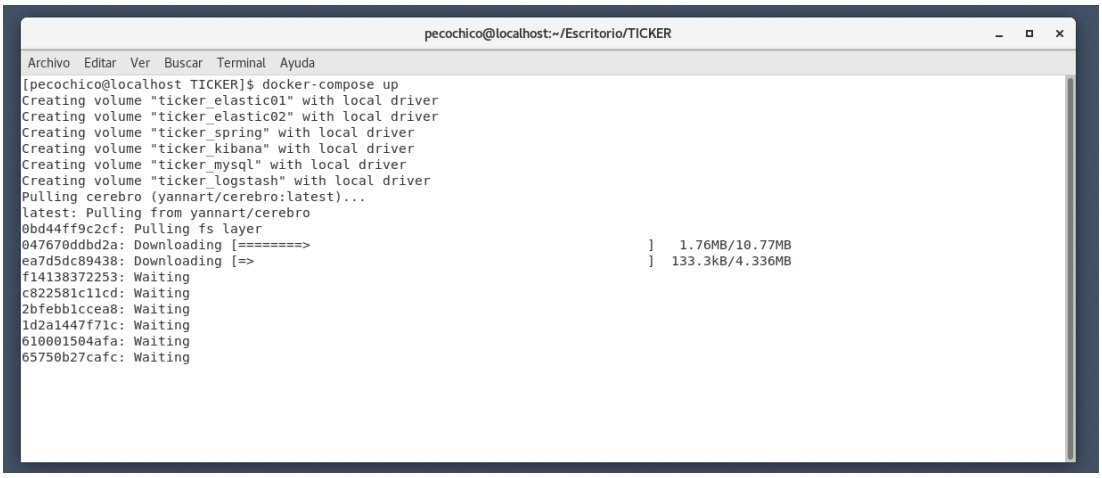


Figura B.2: Lanzamiento del archivo docker-compose.yml

Una vez finalizados los procesos de creación, comienzan a arrancar los servicios señalados en el archivo compose. Entre ellos podemos apreciar como arranca el sistema gestor de incidencias ticker server.

Figura B.3: Puesta en funcionamiento del servicio Spring.

Podemos validar que todos los servicios están levantados ejecutando el comando Docker que lista los contenedores operativos. Es importante prestar atención a la columna 'CONTAINER ID' que nos ofrece información sobre los códigos identificadores de los contenedores.

```
docker ps -a
```

```
[pecochico@localhost TICKER]$ docker ps -a
CONTAINER ID        IMAGE                                          COMMAND                  CREATED
4ef598547376        docker.elastic.co/kibana/kibana-oss:6.2.4      "/bin/bash /usr/loca…"   3 minutes ago
30ed661719b6        ticker_elasticsearch2                          "/usr/local/bin/dock…"   3 minutes ago
afa972d0343f        ticker_spring                                  "java -jar soporte-0…"   3 minutes ago
03ed62132a9a        ticker_elasticsearch                           "/usr/local/bin/dock…"   3 minutes ago
9d19e6434320        ticker_logstash                                "/usr/local/bin/dock…"   3 minutes ago
3cbfb8472b19        yannart/cerebro:latest                         "./bin/cerebro"          3 minutes ago
d073c1f57b35        mysql:8.0.17                                   "docker-entrypoint.s…"   3 minutes ago
[pecochico@localhost TICKER]$
```

Figura B.4: Contenedores en funcionamientro tras el despliegue.

3. El siguiente paso será poblar la base de datos con el dataset de tickets de ejemplo. Desde la carpeta ticker se ejecutan los comandos siguientes:

   ```
   docker cp dumptotal.sql d073:/data/mysql
   docker exec -it d073/bin/bash
   ```

   Con la primera línea, se copia el backup de la base de datos en el interior del contenedor MySQL. En la siguiente línea se accede al bash del contenedor para poder ejecutar el acceso a la base de datos. Nótese que para acceder al contenedor no es necesario indicar todo el container id, simplemente las cuatro primeras cifras nos permiten acceder sin problemas.

   Una vez dentro del contenedor se vuelca el backup en la base de datos creando a la vez la base de datos contenedora, mediante el comando siguiente:

   ```
   mysql -u root -p mydb </data/mysql/dumptotal.sql
   ```

   El sistema nos solicita la password que habremos detallado en el archivo Dockerfile de creación del contenedor Mysql. Para validar la correcta creación de las tablas, se accede a la base de datos.

   ```
   mysql -u root -p
   ```

   Una vez autenticados en el sistema, se indica la base de datos que se desea utilizar y se solicitan las tablas que forman la base de datos, mostrándose la tabla de tickets, usuarios y roles.

   ```
   Welcome to the MySQL monitor.  Commands end with ; or \g.
   Your MySQL connection id is 9
   Server version: 8.0.17 MySQL Community Server - GPL
   ```

```
Copyright (c) 2000, 2019, Oracle and/or its affiliates.
All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

mysql> use mydb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;

+----------------+
| Tables_in_mydb |
+----------------+
| tickets        |
| user_roles     |
| users          |
+----------------+
3 rows in set (0.00 sec)

mysql>
```

4. Una vez poblada la base de datos, se procede a crear el índice de Elasticsearch. Este debe ser creado antes de realizar la primera ingesta de tickets. Mediante la utilidad 'CEREBRO' (**http://localhost:9000/#/conne**ct) se puede administrar de manera visual los índices actuales en Elastic. Para ello se debe indicar la dirección de conexión al cluster.

Figura B.5: Utilidad de administración de índices CEREBRO

Se puede observar como en estos momentos no tenemos ningún índice creado. Simplemente se aprecian los dos nodos creados en el cluster desplegado

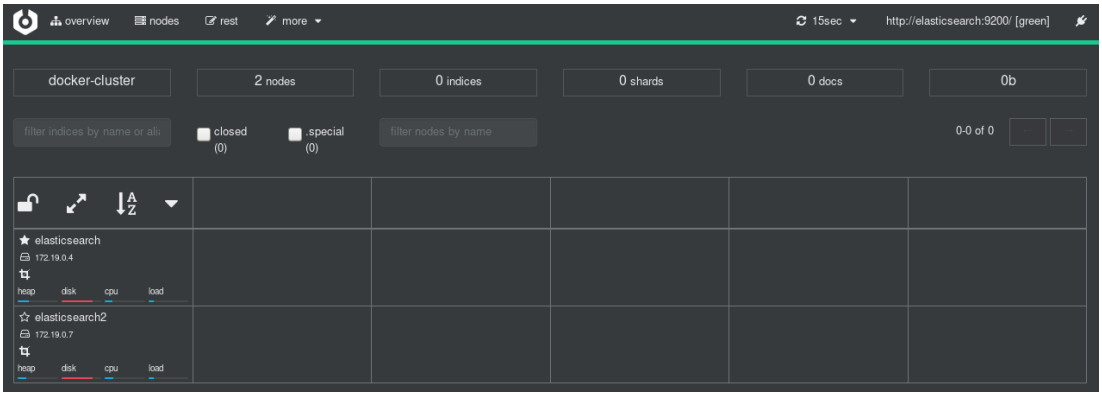


Figura B.6: Estado actual del cluster Elastic

Se accede al servicio Kibana desde **http://localhost:5601/app/kibana** para la creación del índice. Seleccionando en la barra izquierda el icono de una llave inglesa, se accede a la consola, en la que se debe pegar el código facilitado para la creación del índice que se encuentra en la carpeta Kibana de la estructura de carpetas inicial. Una vez pegado, se presiona el pequeño icono verde de 'play', que lanza la ejecución en la consola de Kibana. Si no se produce ningún error, devuelve un mensaje de ejecución correcta

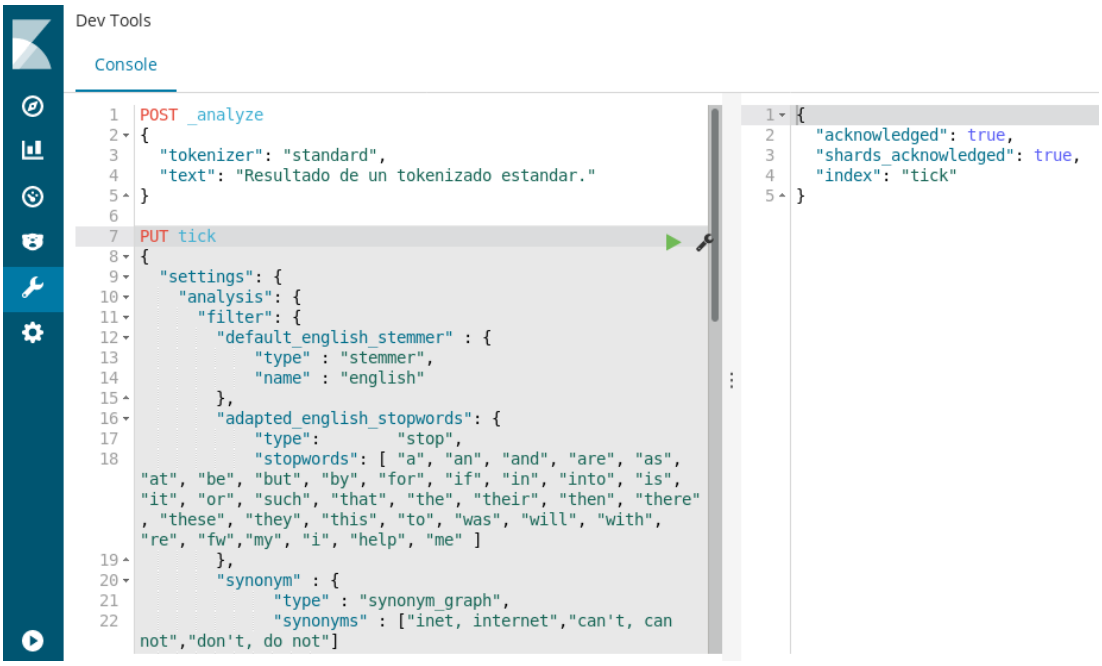


Figura B.7: Correcta creación del índice tick

Se puede validar la creación, recuperando la pantalla de **cerebro**, para observar como se ha generado un índice, distribuido sobre dos nodos de cinco shards (valor por defecto) cada uno. El índice utiliza únicamente 5 shards, que distribuye sobre los nodos de manera alternativa. Es importante señalar que el número de documentos es cero, ya que no hemos reallizado la ingesta todavía.

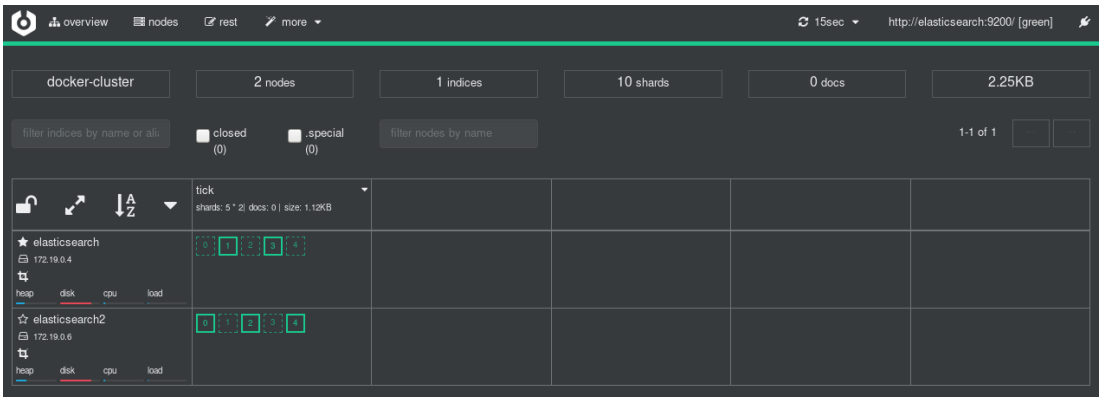


Figura B.8: Distribución del índice sobre los nodos del cluster

5. Para trasladar los tickets cerrados desde la base de datos, se debe configurar y lanzar la ingesta desde el contenedor de Logstash. Para ello, primero se

procede a copiar el archivo de configuración del pipeline a lanzar.

Desde la carpeta logstash/config abrimos terminal y copiamos el archivo *logstash.conf* a la carpeta de pipelines del contenedor Logstash. Accedemos al contenedor, y dentro de este a la carpeta que almacena el bin ejecutable de Logstash. Desde alli, lanzamos el programa, indicando la configuración del pipeline deseado.

```
docker cp logstash.conf 9d19:/usr/share/logstash/pipeline
docker exec -it 9d19 /bin/bash
logstash -r -f /usr/share/logstash/pipeline/logstash.conf
```

Se aprecia como arranca el programa, realizando la conexión y comenzando la ingesta. Si se visualiza a la vez la aplicación **cerebro** podemos apreciar como se incrementa el número de documentos inyectados en el índice.

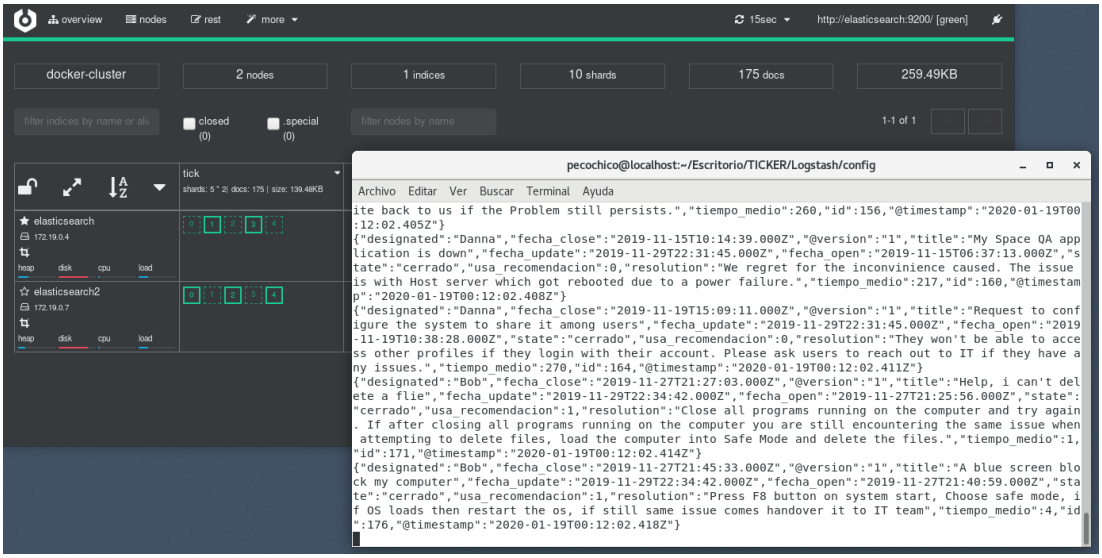


Figura B.9: Población del índice con los documentos ingestados

En estos momentos el sistema ya está plenamente operativo. Para validar el funcionamiento, se puede realizar un ciclo completo de creación, acceso, recomendación y solución de una incidencia.

B.2. Ciclo de una incidencia

Se representa a continuación el ciclo completo de una incidencia, para validar el funcionamiento operativo de los sistemas.

1. Se accede al sistema gestor (**http://localhost:8087/login**) identificándose como técnico de nivel uno. En la base de datos se ha creado un usuario con ese perfil, denominado Adam, con password Adam. Una vez identificado, visualiza las últimas incidencias y su estado. Se procede a crear un nuevo ticket.

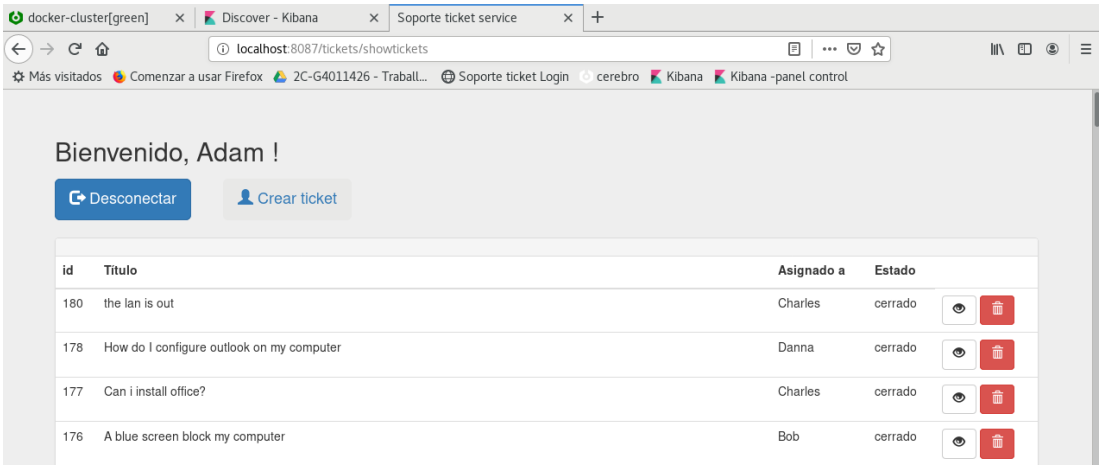


Figura B.10: El administrador Adam accede al servicio gestor

En el interfaz de creación añade el título de la incidencia y asigna esta a un técnico de nivel dos de los disponibles en el desplegable de la derecha. Si todo está correcto pulsa el botón 'Abrir ticket'. Así un nuevo registro se almacena en la base de datos, en estado abierto y asignado en nuestro ejemplo al técnico Charles.

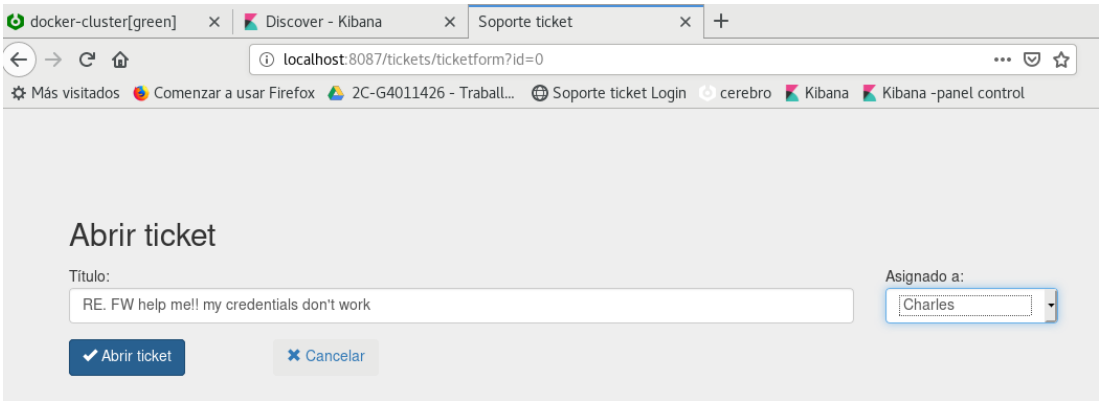


Figura B.11: El administrador Adam crea una nueva incidencia

Cuando Charles se identifique, accederá a su lista de incidencias y observará una nueva en estado abierto. Procederá a su resolución mediante el botón de visualización de la incidencia, situado a la derecha de esta.

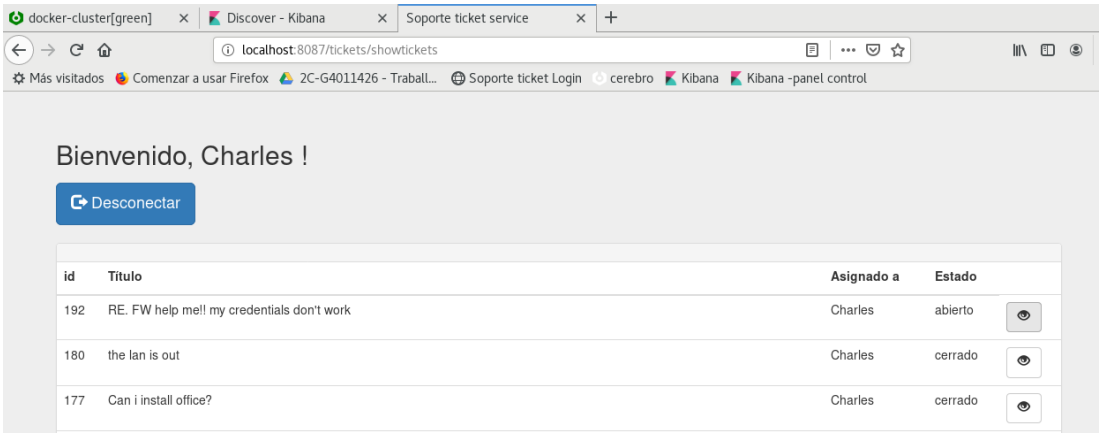


Figura B.12: Visualización y acceso a la incidencia abierta

En el momento en que accede a la nueva incidencia abierta para su resolución, el proceso recomendador ya ha consultado el título, recopilando las recomendaciones y las procede a mostrar debajo del campo de resolución de la incidencia. El técnico pondera las soluciones recomendadas y aplica la primera mediante el botón de la derecha. Automáticamente se cumplimenta el campo resolución recomendada y el técnico procede a cambiar el estado de la incidencia a cerrado, actualizando el ticket mediante el botón de actualizado.

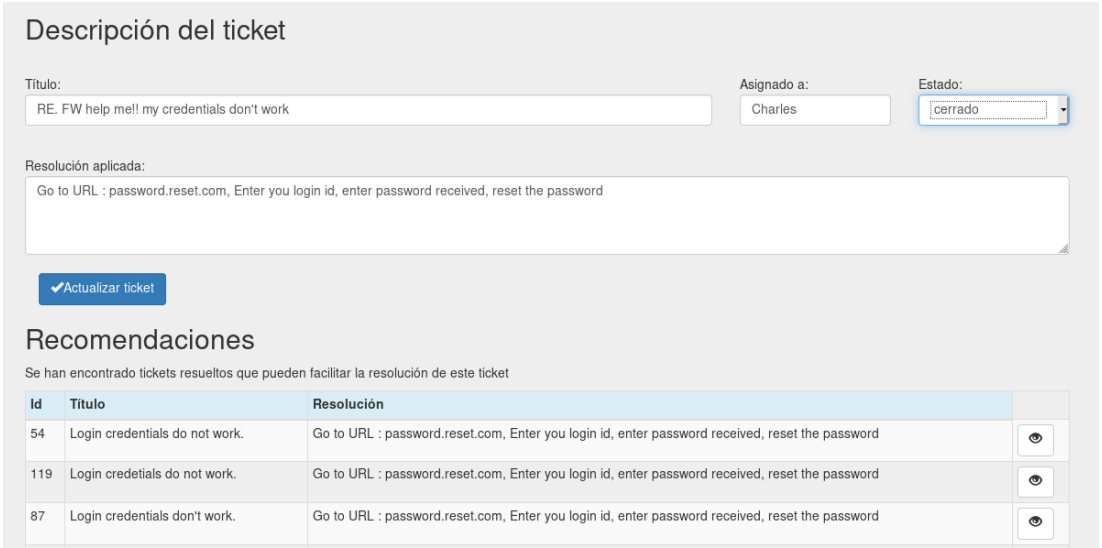


Figura B.13: Solucionado de incidencia mediante recomendador

En este momento, la incidencia cambia su estado a cerrado y pasa a ser candidata a ingesta en el próximo ciclo de carga que ejecute Logstash. Se

puede apreciar como la nueva incidencia se recoge en la salida del proceso, y como el número de documentos se ha incrementado en un nuevo valor.

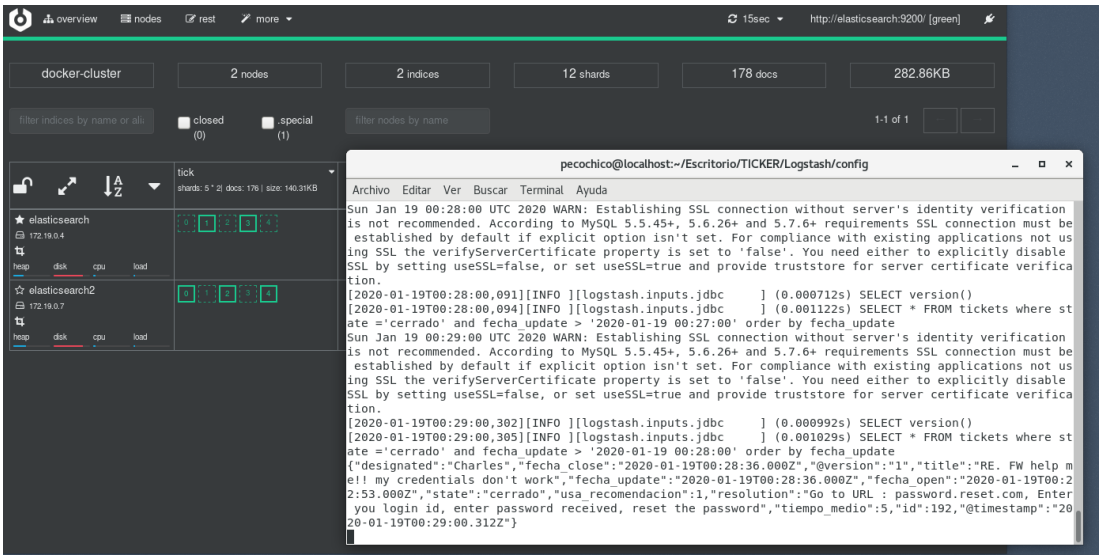


Figura B.14: La incidencia cerrada se ingesta en el índice

Apéndice C

Manuales técnicos

C.1. Diagramas de clases del sistema gestor de incidencias.

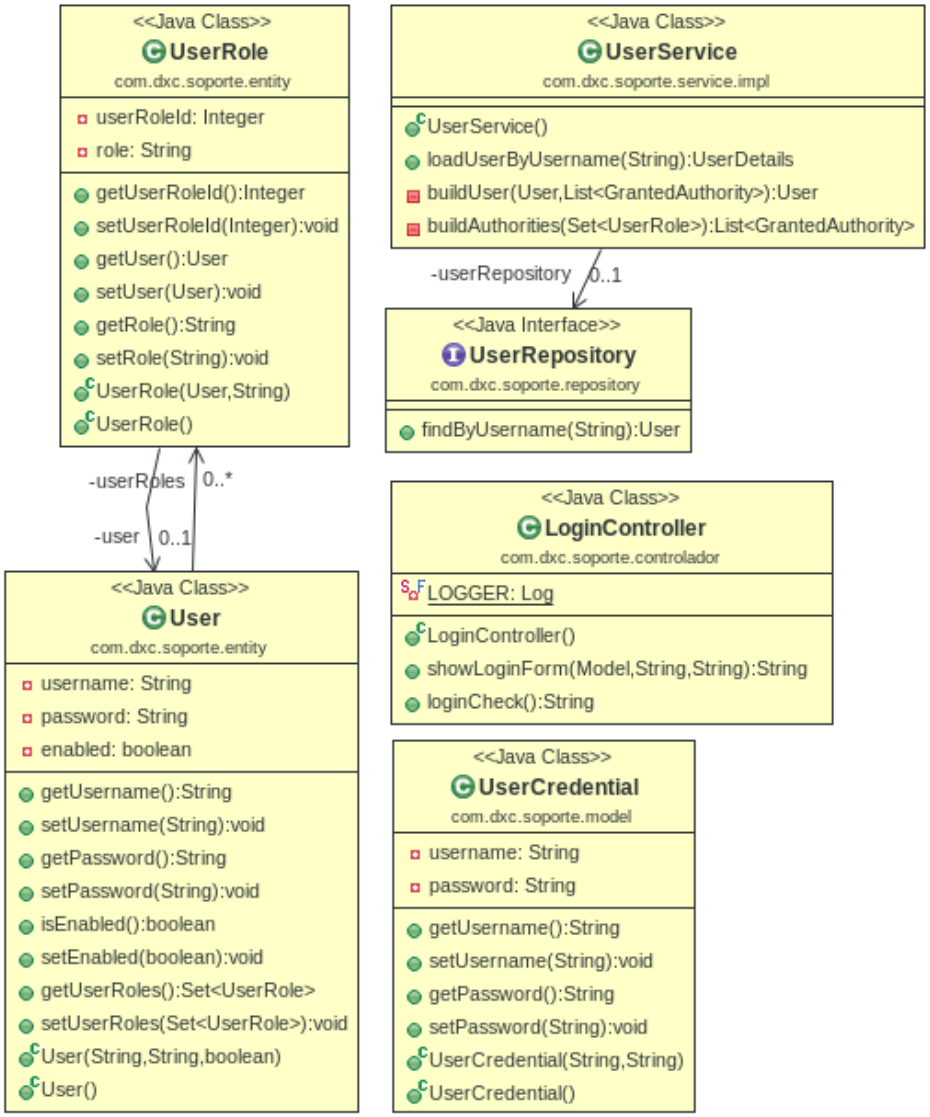


Figura C.1: Clases (1 de 3)

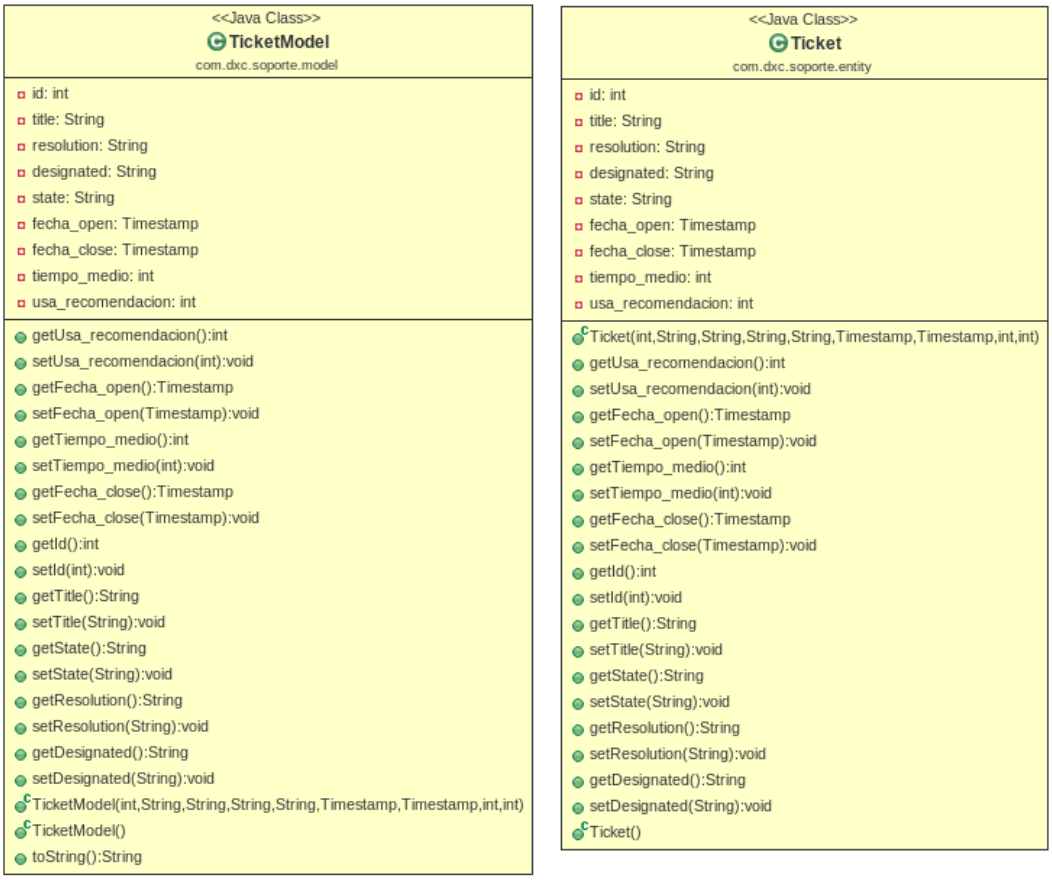


Figura C.2: Clases (2 de 3)

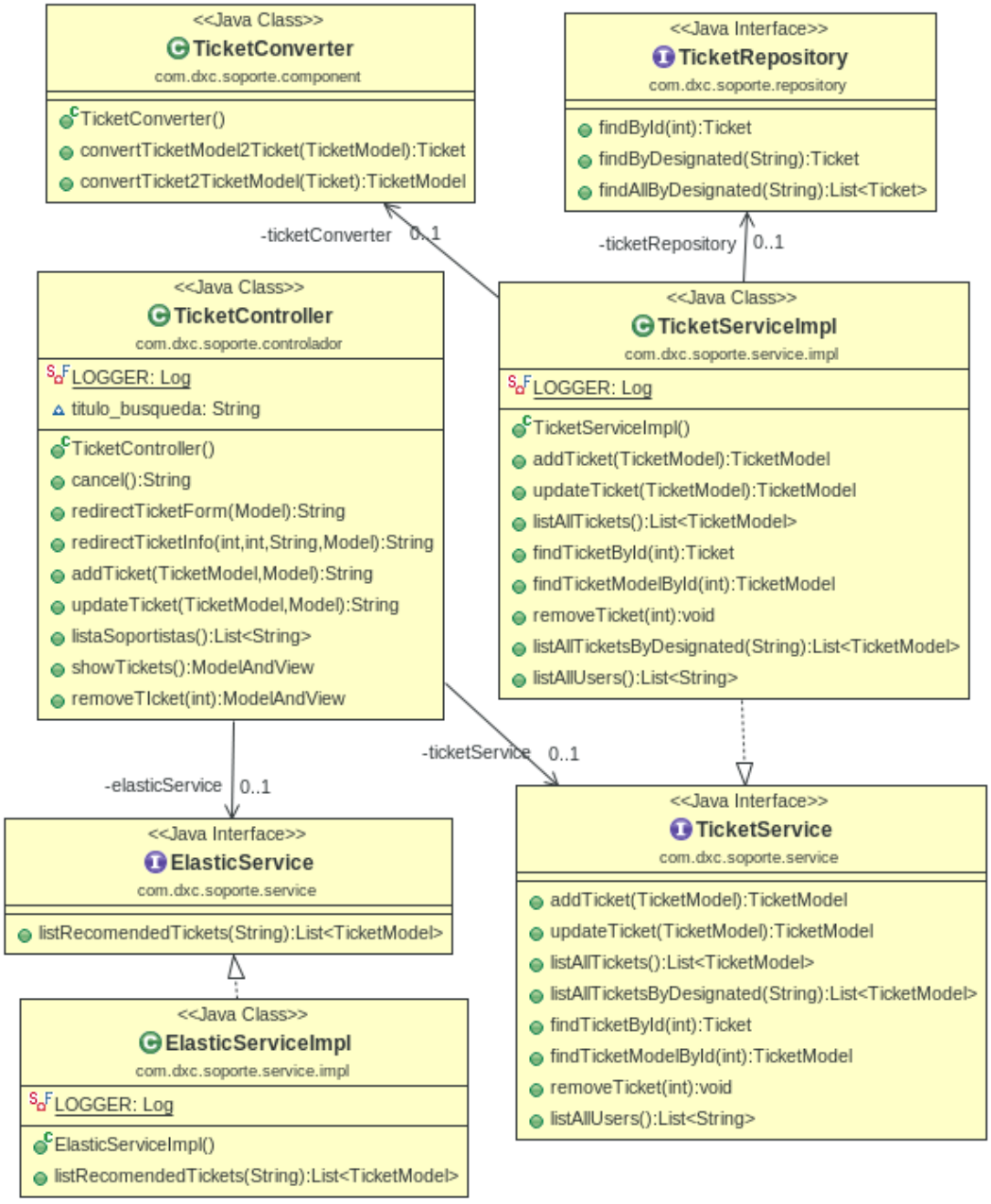


Figura C.3: Clases (3 de 3)

C.2. Árbol de clases

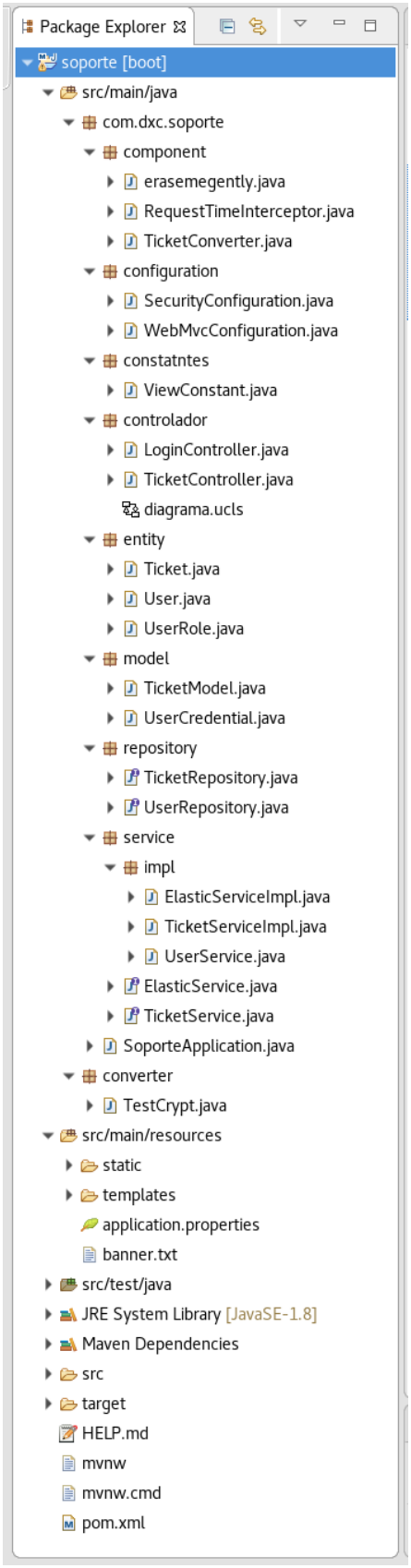


Figura C.4: Árbol de clases del gestor de incidencias

C.3. Índice recomendador

```
PUT tick
{
  "settings": {
    "analysis": {
      "filter": {
        "default_english_stemmer" : {
            "type" : "stemmer",
            "name" : "english"
        },
        "adapted_english_stopwords": {
            "type":        "stop",
            "stopwords": [ "a", "an", "and", "are", "as", "at", "be", "but", "by", "for",
            "if", "in", "into", "is", "it", "or", "such", "that", "the", "their", "then",
            "there", "these", "they", "this", "to", "was", "will", "with", "re", "fw","my",
            "i", "help", "me" ]
        },
        "synonym" : {
              "type" : "synonym_graph",
              "synonyms" : ["inet, internet","can't, can not","don't, do not"]
                  }
      },
      "analyzer": {
        "tick_analyzer": {
          "type": "custom",
          "tokenizer": "standard",
          "filter": ["lowercase", "asciifolding", "adapted_english_stopwords",
          "default_english_stemmer","synonym"]
        }
      }
    }
  },
  "mappings": {
    "_doc": {
      "properties": {
        "title": {
          "type": "text",
          "analyzer": "tick_analyzer",
          "search_analyzer": "tick_analyzer"
        },
        "resolution": {
          "type": "text",
          "analyzer": "tick_analyzer",
          "search_analyzer": "tick_analyzer"
        },
        "fecha_open": {
          "type": "date"
        },
        "fecha_close": {
          "type": "date"
        }
      }
    }
  }
}
```

Figura C.5: Cuerpo del índice recomendador

C.4. Enlace descarga Imagen Centos 7

Imagen virtual VMWare Centos 7 con los materiales para el despliegue

Bibliografía

[1] MySQL Reference. Versión 8.0. Página del producto (*https://dev.mysql.com/doc/refman/8.0/en/*). Consultado por última vez el 2 de enero de 2020.

[2] Elasticsearch Reference. Versión 6.2.4. Página del producto (*https://www.elastic.co/guide/en/elasticsearch/reference/6.2/index.html*). Consultado por última vez el 2 de enero de 2020.

[3] Logstash Reference. Versión 6.2.4. Página del producto (*https://www.elastic.co/guide/en/logstash/6.2/index.html*). Consultado por última vez el 2 de enero de 2020.

[4] Kibana Reference. Versión 6.2.4. Página del producto (*https://www.elastic.co/guide/en/kibana/6.2/index.html*). Consultado por última vez el 2 de enero de 2020.

[5] Amuthan Ganeshan, *Spring MVC Beginner's Guide*, 2ª edición, Packt Publishing; Edición: 2nd Revised edition (29 de julio de 2016).

[6] Thymeleaf. Version 3.0.4. Página del proyecto (*https://www.thymeleaf.org/documentation.html*). Consultado por última vez el 2 de enero de 2020.

[7] GAMMA, Erich et al. , *Patrones de Diseño: elementos de software orientado a objetos reutilizable.*, Madrid: Addison-Wesley, 2002. ISBN 84-7829-059-1.

[8] Spring Data JPA. Version 2.2.3.RELEASE, 2019-12-04. Página del proyecto (*https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/ht*). Consultado por última vez el 2 de enero de 2020.

[9] Reference Guide. 1.0.0.Final. Página del proyecto (*https://docs.jboss.org/hibernate/jpamodelgen/1.0/reference/en-US/html_single*). Consultado por última vez el 2 de enero de 2020.

[10] Spring Security Reference. 5.2.2.BUILD-SNAPSHOT. Página del proyecto (*https://docs.spring.io/spring-security/site/docs/5.2.2.BUILD-SNAPSHOT/reference/htmlsingle/*). Consultado por última vez el 2 de enero de 2020.

[11] Docker community edition. Página del producto (*https://docs.docker.com/install/*). Consultado por última vez el 2 de enero de 2020.

[12] Introducción a JSON. Página del proyecto (*https://www.json.org/json-es.html*). Consultado por última vez el 2 de enero de 2020.

[13] MySQL Workbench. Versión 8.0.19. Página del producto (*https://www.mysql.com/products/workbench/*). Consultado por última vez el 2 de enero de 2020.

[14] Java. Version 8. Página del producto (*https://www.java.com/es/about/whatis_java.jsp*). Consultado por última vez el 2 de enero de 2020.

[15] Apache Maven Project. Versión 3.6.3. Página del proyecto (*https://maven.apache.org/*). Consultado por última vez el 2 de enero de 2020.

[16] Spring Boot Reference Guide. 2.1.12.BUILD-SNAPSHOT. Página del proyecto (*https://docs.spring.io/spring-boot/docs/2.1.12.BUILD-SNAPSHOT/reference/html/*). Consultado por última vez el 2 de enero de 2020.

[17] YALM. Versión 1.2.(3rd Edition) Página del proyecto (*https://yaml.org/*). Consultado por última vez el 2 de enero de 2020.

[18] Overview of Docker Compose. Versión 1.25.0. Página del producto (*https://docs.docker.com/compose/*). Consultado por última vez el 2 de enero de 2020.

[19] Roger S Pressman *Ingeniería del Software. Un enfoque práctico.*, Ed. Mc Graw Hill, España. 2005. ISBN: 970-10-5473-3.

[20] Querydsl Reference Guide. Versión 4.1.3. Página del proyecto (*http://www.querydsl.com/static/querydsl/4.1.3/reference/html_single/*). Consultado por última vez el 2 de enero de 2020.

[21] A Java serialization/deserialization library to convert Java Objects into JSON and back. Versión 2.8.6. Página del proyecto (*https://github.com/google/gson*). Consultado por última vez el 2 de enero de 2020.

[22] The Apache Lucene project develops open-source search software. Versión 8.4.1. Página del proyecto (*https://lucene.apache.org/*). Consultado por última vez el 2 de enero de 2020.

[23] Redmine. Página del producto (*https://www.redmine.org/*). Consultado por última vez el 2 de enero de 2020.

[24] RT (Request Tracker). Página del producto (*https://bestpractical.com/request-tracker*). Consultado por última vez el 2 de enero de 2020.

[25] OTRS. Página del producto (*https://otrs.com/es/home/?lang=es*). Consultado por última vez el 2 de enero de 2020.

[26] C-desk. Página del producto (*http://www.cdesk.in/*). Consultado por última vez el 2 de enero de 2020.

[27] Aprendizaje autónomo Elasticsearch. Página del producto (*https://www.elastic.co/es/what-is/elasticsearch-machine-learning/*). Consultado por última vez el 2 de enero de 2020.

[28] GitHub plugin detector de idioma sobre Logstash. Página del proyecto (*https://github.com/thehybridtechnician/logstash-filter-language*). Consultado por última vez el 2 de enero de 2020.

[29] Publicaciones de Pablo Gamallo Otero. Página del investigador (*https://citius.usc.es/equipo/persoal-adscrito/pablo-gamallo-otero/*). Consultado por última vez el 2 de enero de 2020.

[30] Entendiendo el proceso de búsqueda. (*https://www.blog.google/products/search/search-language-understanding-bert/*). Consultado por última vez el 2 de enero de 2020.

[31] Miguel A. M., *Desarrollo Web con Spring Boot* , https://www.udemy.com/course/desarrollo-web-con-spring-framework-4-de-cero-a-ninja/ (12 de diciembre de 2016).