Facultad de Ciencias

Desarrollo de una herramienta de Business Intelligence para ayudar a la toma de decisiones de negocio a la empresa CIC

Development of a Business Intelligence tool to help business decision making in CIC

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Víctor Gómez Cobo

Director: Patricia López Martínez

Codirector: Sergio Herrera Iglesias

Junio – 2016

Agradecimientos

Este trabajo fin de grado cierra una de las etapas más importante en mi vida. Hay muchas personas sin las cuales este trabajo no habría sido posible.

En primer lugar quiero agradecer a mis padres su apoyo tanto moral como económico, que me ha permitido estudiar esta carrera.

En segundo lugar tengo que agradecer la labor de los profesores que me han dado clase, y en especial a la directora de este trabajo, sin la cual, no tendría la misma calidad. También me gustaría recordar a mis compañeros de clase por las experiencias vividas y el mutuo apoyo durante estos años.

Finalmente, quiero agradecer a CIC la oportunidad para haber desarrollado este proyecto. En especial me gustaría agradecer al responsable del mismo, por haberme propuesto la idea y haber confiado en mí para su desarrollo. Tampoco quiero olvidarme de mis compañeros en la empresa y quisiera agradecerles su ayuda y paciencia conmigo.

A todos ellos y a los que me haya podido olvidar, muchas gracias.

Resumen

En la actualidad, las empresas cada vez necesitan más datos para trabajar y tomar decisiones adecuadas para su negocio. Es por eso que surge la necesidad de utilizar herramientas que faciliten y mejoren la interpretación de esos datos para convertirlos en información que será usada para mejorar el proceso de toma de decisiones. Esto es lo que se denomina *Business Intelligence* y es totalmente necesario para ayudar a las empresas a obtener una ventaja competitiva con respecto a sus competidores.

Este trabajo fin de grado consiste en el desarrollo e integración de un módulo de *Business Intelligence* en una herramienta interna de la empresa CIC Consulting Informático de Cantabria. El objetivo es permitir a los usuarios de dicha herramienta visualizar los datos de forma gráfica para simplificar y agilizar la toma de decisiones. Para ello, el módulo permite la creación de gráficas con los datos que hasta ahora se mostraban en forma de tabla así como la inserción de dichas gráficas en dashboards personalizados. Finalmente, también existe la opción de crear informes que pueden exportarse en formato PDF con el objetivo de llevar las gráficas fuera del sistema.

Palabras clave: *Business Intelligence*, Vaadin, Spring, aplicación web, visualización de datos.

Summary

Nowadays companies need more and more data to make the best business decisions. That is why the need to use tools that facilitates the interpretation of those data and its transformation on information useful for the companies appears. This is called Business Intelligence, which is a must to help companies to obtain a competitive edge over their competitors.

This work consists on the development and integration of a Business Intelligence software unit into an internal tool of the company CIC Consulting Informático de Cantabria. The objective is allowing users to view data in charts, in order to simplify and speed up decision making. For that purpose, the application allows for the creation of charts, which show data that were shown in tables until now. Another functionality is the creation of personalized dashboards including charts. Finally, the user can create reports and export them to PDF.

Keywords: Business Intelligence, Vaadin, Spring, web application, data visualization tools.

Índice general

Índice de figuras

Índice de tablas

1 Introducción y objetivos

El propósito del presente trabajo es el desarrollo de un módulo de *Business Intelligence* que será integrado en una aplicación web desarrollada en la empresa CIC Consulting Informático de Cantabria, desde ahora, CIC. A continuación se describe qué es la inteligencia de negocio o *Business Intelligence* y qué se espera del módulo desarrollado. Además se explica en qué consiste y cómo funciona la herramienta en la que será integrado.

1.1 Business Intelligence

Las empresas cada vez necesitan utilizar más datos para tomar sus decisiones de negocio. Ante tal cantidad de datos es necesario utilizar tecnologías que permitan extraer la información de la manera más eficiente posible, de forma que se facilite el proceso y se ayude a que las decisiones que se tomen sean las mejores.

Business Intelligence [1] es el conjunto de estrategias utilizadas para transformar los datos en información y la información en conocimiento, de forma que se pueda utilizar ese conocimiento para tomar decisiones de negocio que proporcionen a la empresa una ventaja competitiva. Esa ventaja consiste en poseer la información necesaria para responder a problemas de negocio tales como la entrada a nuevos mercados, control financiero, optimización de costes o rentabilidad de un producto concreto, entre otros.

La empresa CIC cuenta con una aplicación web que permite realizar consultas que proporcionan datos de interés sobre diferentes sistemas de la organización. Estos datos proporcionan información al usuario que los visualiza, sin embargo es difícil y costoso extraer toda la información ya que los datos se muestran en forma de tabla. Por eso surge la necesidad de mostrar esos datos de forma gráfica, para facilitar el análisis y ayudar a la toma de decisiones. Para ello se ha desarrollado el módulo que será explicado a lo largo de este documento.

1.2 Herramienta LUCA

La herramienta en la que será integrado el módulo consiste en una aplicación web denominada LUCA. LUCA es una herramienta de monitorización y explotación de sistemas. Su objetivo es obtener información de un conjunto de sistemas para ayudar a tomar las mejores decisiones. Para ello tiene una serie de componentes y maneja un conjunto de conceptos que se explican a continuación.

1.2.1 Entornos

Un entorno consiste en una agrupación de sistemas de información. Por ejemplo, se podría definir un entorno de una aplicación que utiliza varios sistemas como puede ser una base de datos o un servicio web. En LUCA existen cuatro tipos de entornos que tendrán declarados distintos datos de conexión: Desarrollo, Preproducción, Producción

y QA. De esta manera una vez configurado el entorno con sus sistemas, se pueden ejecutar las consultas en el entorno que se indique en el momento de ejecución.

1.2.2 Sistemas

Un sistema en LUCA representa un sistema de información que puede ser una base de datos o un servicio web (SOAP, REST u OData). Cada sistema deberá tener asociados datos de conexión para el entorno al que pertenece, de forma que si un sistema existe en dos entornos, deberá tener unos datos de conexión para un entorno, y otros para el otro. Al indicar en la ejecución de una consulta el entorno, se comprobará que exista un dato de conexión del sistema para el entorno indicado, si no es así, no será posible la ejecución.

1.2.3 Data Sources

Como se describe en el punto anterior, se permiten dos tipos de orígenes de datos: bases de datos o servicios web. La configuración de éstos es tarea del administrador del sistema.

1.2.4 Custom Queries

Una *Custom Query* es una consulta simple, compuesta por un conjunto de variables de entrada y otro conjunto de variables de salida. Las consultas se ejecutarán contra un sistema de información determinado. Dependiendo del sistema contra el que se ejecuten, se deben indicar unos datos distintos, aunque hay un conjunto de datos comunes a todos los sistemas:

- **Nombre**: Nombre identificativo de la *Custom Query*.
- **Estado**: Estado en que se encuentra la *Custom Query*. Puede encontrarse en dos estados:
    - Edición: Una *Custom Query* en estado de edición se encuentra en construcción y, por tanto, no puede ser ejecutada.
    - Ejecución: Una *Custom Query* se encuentra en estado de ejecución una vez que se ha construido y comprobado su correcto funcionamiento.

- **Tipo**: Tipo de la *Custom Query*.
- **Variables de entrada**: Listado de las variables requeridas por la consulta. Por cada una, debe indicarse su nombre y su tipo.
- **Variables de salida**: Listado de las variables de salida generadas por la consulta. Por cada una, debe indicarse su nombre y su tipo.

A continuación se muestran los datos necesarios en función del tipo de sistema contra el que se ejecute la *Custom Query*:

- **Base de datos:**
    - **Consulta SQL:** Código SQL correspondiente a la consulta que ha de realizar la *Custom Query*.

- **Servicio Web SOAP:**
  - **SOAP Action**: *SOAP Action* (se usa para indicar el propósito de la petición HTTP) que será incluido en la cabecera HTTP al realizar la consulta correspondiente a la *Custom Query*.
  - **Service**: Nombre correspondiente al método que ha de ser ejecutado en el correspondiente Servicio Web SOAP.
  - **SOAP**: Plantilla SOAP que será utilizada para conformar el mensaje SOAP con que ejecutar la consulta correspondiente a la *Custom Query*.

- **Servicio Web OData:**
  - **Service**: Nombre del método que ha de ser ejecutado en el correspondiente Servicio Web al realizar la consulta.
  - **Filtro**: Filtro que debe aplicarse a la hora de realizar la consulta correspondiente a la *Custom Query*.

- **Servicio Web REST:**
  - **Recurso**: URI del recurso que debe ser consultado por la *Custom Query*.
  - **Filtro**: Filtro que debe aplicarse a la hora de realizar la consulta correspondiente a la *Custom Query*.

Los resultados de la ejecución de la *Custom Query* se retornan en formato JSON. A continuación se muestra un ejemplo del resultado de ejecución de una *Custom Query* cuyo objetivo es obtener el número de fallos en las ejecuciones de las consultas por cada día del mes de enero (solo se muestran tres días para no ocupar demasiado espacio):


```
{
   "variablesSalida":[
      { "nombre":"FECHA", "tipo":"STRING"},
      { "nombre":"FALLOS", "tipo":"STRING"}
   ],
   "resultados":[
      { "row":["28/01/2016", "3"] },
      { "row":["27/01/2016", "2"] },
      { "row":["26/01/2016", "2"] }
   ],
   "stateResponse":{ "code":"001" },
   "paginado":{"page":1,"sizePage":3000,"count":0,"isWithCount":false },
   "timeInicio":1465480139472,
   "timeFin":1465480139497,
   "timeCountInicio":0,
   "timeCountFin":0
}
```


El objeto `variablesSalida` define las variables de salida de la consulta. El objeto `resultados` contiene todos los resultados agrupados por filas. Cada fila contiene el valor que devuelve la consulta para cada variable de salida. De modo que la primera fila devuelve que el 28/01/2016 se produjeron tres fallos en la ejecución de consultas. El resto de objetos y atributos son parámetros de configuración que no son relevantes para la creación de gráficas.

1.2.5 Seguridad

Los usuarios deberán autenticarse en el sistema utilizando un nombre de usuario y una contraseña. Estas credenciales serán las que se utilizan para todas las aplicaciones de CIC o también se podrán crear nuevos usuarios a través de la aplicación. Además de proteger el acceso a la herramienta mediante autenticación, se establece un sistema de roles para restringir el acceso y operaciones que se podrán realizar dentro de ésta, y un sistema de autorizaciones para controlar el acceso a los entornos y sistemas.

1.2.5.1 Roles

Un rol representa una agrupación de funcionalidades, por lo que el usuario que posea el rol podrá realizar todas las operaciones que agrupe dicho rol. Los roles son administrados por un usuario especial, en este caso el/los usuarios que posean el rol de administrador. Las funcionalidades permitidas para cada rol en la aplicación original se describen en la siguiente tabla:

Tabla 1.1 Roles definidos en la herramienta LUCA antes de incluir el nuevo módulo

ROL	DETALLE
ROLE_ADMINISTRADOR	Será el rol que más permisos tenga en el sistema. Encargado de la administración de la aplicación; creará usuarios, asignará roles, definirá los entornos de ejecución, sistemas y datos de conexión.
ROLE_CREADOR_CQ	Este rol tendrá permisos para poder ver, crear, modificar y eliminar únicamente *Custom Queries.*
ROLE_VIEW_EXEC_CQ	Los usuarios que posean este rol únicamente podrán ver y ejecutar *Custom Queries*. Para ello deberán poseer las autorizaciones correspondientes a nivel de entorno y sistema.

1.2.5.2 Autorizaciones

Al crear un entorno o un sistema se le debe asignar un nombre de autorización. Este nombre será el que use el sistema de autorizaciones para comprobar si un usuario tiene la autorización necesaria para ejecutar o ver consultas dentro de ese entorno o sistema.

El nombre de la autorización siempre deberá tener el formato **AUTH_***. Cuando se introduce un usuario a la lista de un entorno o de un sistema, automáticamente se le asigna la autorización del entorno o sistema correspondiente. De esta forma, cada vez que un usuario vaya a ejecutar una consulta se comprueba que tiene la autorización correspondiente para ejecutar consultas en el entorno, y en el sistema específico al que pertenece la consulta.

1.3 Objetivos del proyecto y organización de la memoria

En esta sección se detallan cuáles son los objetivos del proyecto y como está organizada esta memoria.

1.3.1 Objetivos

El objetivo del proyecto es el desarrollo de un módulo de *Business Intelligence* para la herramienta LUCA desarrollada en CIC.

De este módulo se espera que permita al usuario generar gráficas con los datos procedentes de consultas (*Custom Queries*) de tal forma que consiga extraer y visualizar la información necesaria de manera cómoda y fácilmente interpretable. El usuario podrá también crear *dashboards* para agrupar las gráficas que desee. Finalmente podrá generar informes en los que se añadirán gráficas y exportarlos en formato PDF.

1.3.2 Organización de la memoria

Tras este apartado introductorio, la organización de la memoria es la siguiente:

- **Apartado 2. Tecnologías y herramientas utilizadas**: En este apartado se explican las tecnologías y herramientas que se han utilizado para desarrollar la herramienta, así como la metodología seguida en el desarrollo del proyecto.

- **Apartado 3**. **Captura y análisis de requisitos**: En este apartado se detallan las técnicas utilizadas para capturar los requisitos del sistema y se analizan los mismos, para lo que se muestran los diagramas de casos de uso obtenidos y se detallan los casos de uso menos triviales.

- **Apartado 4. Diseño del sistema**: En este apartado se muestra cual es el diseño del sistema. Se explica el diseño arquitectónico y el diseño detallado.

- **Apartado 5. Construcción del sistema**: En este apartado se detalla cómo se desarrolló el sistema. Se explican los pasos seguidos para implementar cada una de las partes del sistema y se muestran ejemplos del código desarrollado.

- **Apartado 6. Pruebas del sistema**: En este apartado se define el proceso de pruebas que se ha seguido para verificar el correcto funcionamiento de la aplicación y se muestra un ejemplo de cada tipo de prueba realizada.

- **Apartado 7. Conclusiones y trabajos futuros**: En este apartado se explican las conclusiones obtenidas tras la realización de este trabajo y se detallan las tareas que se pueden realizar en un futuro para mejorar la aplicación.

2 Tecnologías y herramientas utilizadas

Las tecnologías utilizadas para el desarrollo del módulo han sido las mismas que las utilizadas para desarrollar la aplicación principal: los *frameworks* Vaadin y Spring para la vista y la lógica de negocio, respectivamente, y MySQL junto a JPA para la capa de persistencia. Para la gestión de dependencias del proyecto se ha utilizado Maven. Finalmente, el despliegue de la aplicación se ha realizado en un servidor Apache Tomcat. A continuación, se explican algunos detalles de éstas y otras tecnologías utilizadas en el proyecto.

2.1 Vaadin

Vaadin [2] es un *framework* que permite desarrollar aplicaciones web profesionales y modernas programando únicamente en Java. Proporciona dos modelos de programación, para el lado del cliente (navegador) y para el lado del servidor.

La programación en el lado del cliente utiliza AJAX (*Asynchronous JavaScript and XML*) para la comunicación entre el navegador y el servidor. Este modelo permite el desarrollo de aplicaciones y *widgets* que son compilados de Java a JavaScript y ejecutados en el navegador. La programación en el lado del servidor es la más potente, permitiendo desarrollar aplicaciones sin utilizar HTML ni JavaScript, solamente Java. Para ello utiliza un motor que es ejecutado como JavaScript en el cliente que es el que se encarga de cargar la interfaz de usuario en el navegador y enviar la interacción del usuario al servidor. Para la comunicación de eventos entre el cliente y el servidor se utiliza JSON (*JavaScript Object Notation*). A grandes rasgos, la programación en el lado del cliente se utiliza para el desarrollo de componentes personalizados y *widgets* (como por ejemplo un menú desplegable con distintas opciones al que proporciona el *framework* y que puede ser añadido a cualquier aplicación) y la programación en el lado del servidor se utiliza para el desarrollo de aplicaciones web. La Ilustración 2.1 muestra un esquema de la arquitectura de las aplicaciones desarrolladas con Vaadin.

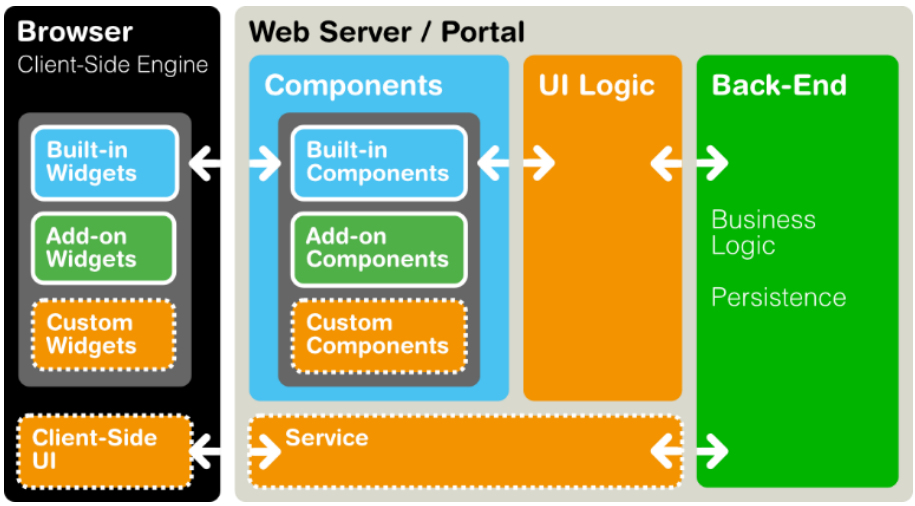

Ilustración 2.1 Arquitectura de las aplicaciones desarrolladas con Vaadin [2]

Una aplicación web desarrollada en el lado del servidor se ejecuta como un *servlet* en un servidor web Java o en un contenedor de *servlets* (como Apache Tomcat), sirviendo peticiones HTTP. El *servlet* recibe las peticiones del cliente y las interpreta como eventos para una determinada sesión. Los eventos son asociados con los componentes de la interfaz de usuario y se envían a los manejadores de eventos definidos en la aplicación. Si se realiza algún cambio en los componentes de la interfaz de usuario, el *servlet* los modifica en el navegador generando una respuesta. El motor del lado del cliente recibe las respuestas y las utiliza para realizar todos los cambios necesarios a la página en el navegador. La Ilustración 2.2 muestra la arquitectura de una aplicación Vaadin en ejecución.

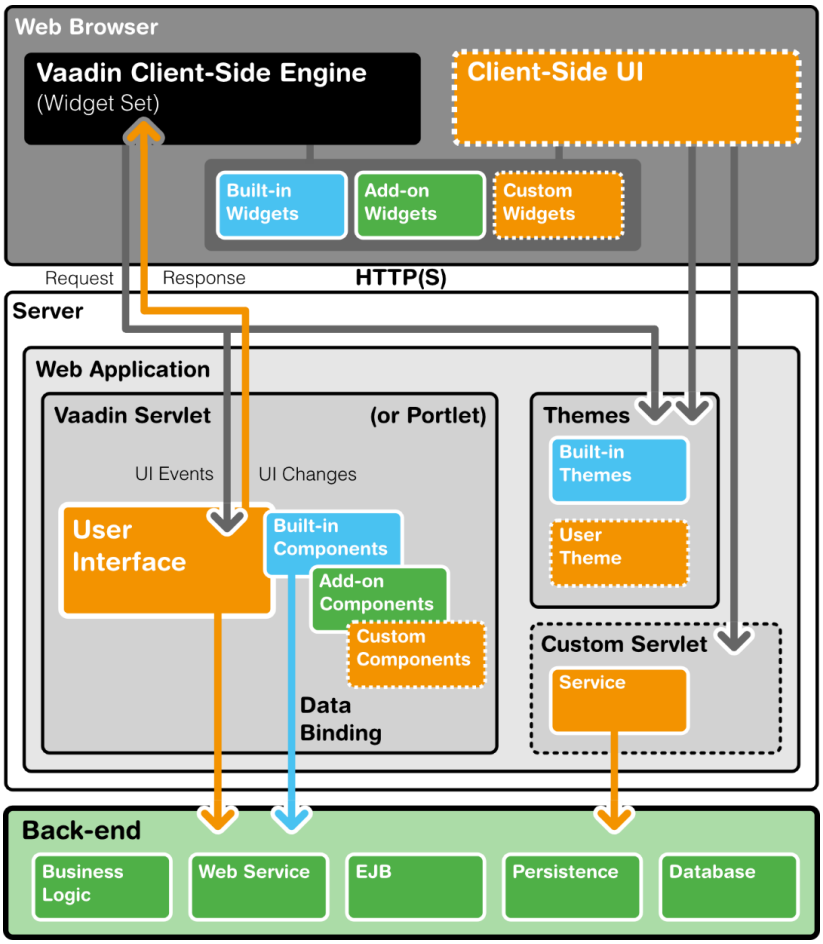

Ilustración 2.2 Arquitectura de una aplicación Vaadin en ejecución [2]

2.2 Vaadin Charts

Vaadin Charts [3] es una herramienta comercial que permite crear todo tipo de gráficas interactivas para visualizar datos. Permite configurar programáticamente todos los elementos de las gráficas e incluso el tema. Una vez comprada la licencia, se puede descargar y añadir a nuestros proyectos. Para su funcionamiento Vaadin Charts utiliza la librería JavaScript Highcharts [4].

2.3 Spring

Spring [5] es una plataforma Java que provee una infraestructura que sirve de soporte para desarrollar aplicaciones empresariales Java. Entre los servicios que proporciona cabe destacar la autenticación, la autorización y la gestión de pruebas. Además, la característica principal es la inyección de dependencias a través de la inversión de control. Spring está compuesto por varios módulos, sin embargo solo es necesario utilizar los que nos sirvan para la aplicación que estemos desarrollando. Los módulos de los que está compuesto Spring se muestran en la Ilustración 2.3.

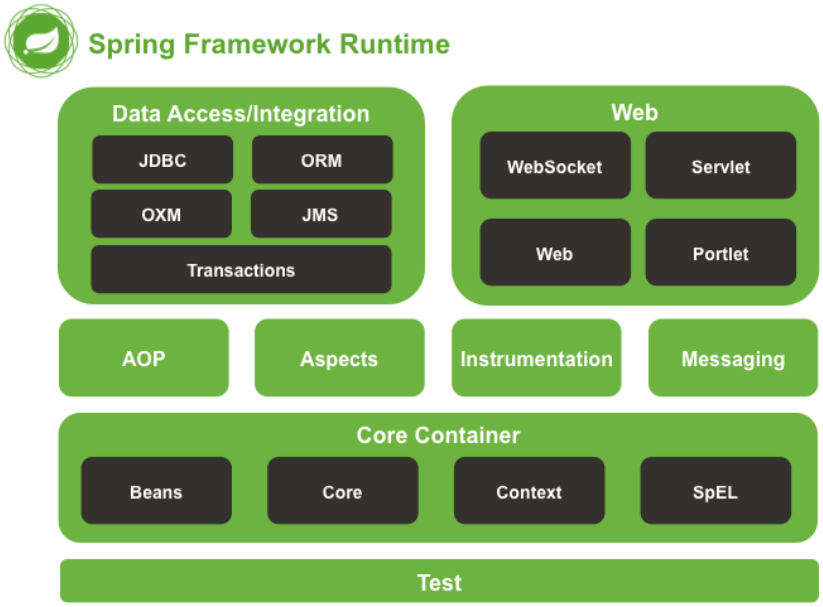

Ilustración 2.3 Módulos de los que está compuesto Spring [4]

Las ventajas [6] del uso de Spring son las siguientes:

- Permite desarrollar aplicaciones empresariales usando POJOs (Plain Old Java Objects). Las ventajas de usar POJOs es que no es necesario un contenedor de EJB como por ejemplo un servidor de aplicación Glashfish, si no que se puede utilizar un contenedor de *servlets* como Tomcat.

- Está organizado de forma modular. Aunque el número de paquetes y clases sea grande, solo es necesario preocuparse de las que se necesitan en cada momento ignorando el resto.

- Probar una aplicación escrita con Spring es fácil ya que usando POJOs es más fácil usar la inyección de dependencias para inyectar los datos de prueba.

- Proporciona una API para traducir excepciones específicas de cada tecnología (lanzadas por JDBC, Hibernate o JDO por ejemplo) en excepciones consistentes y no comprobadas.

2.4 MagicDraw

MagicDraw [7] es una aplicación para modelado sistemas y su arquitectura. Facilita el diseño de aplicaciones orientadas a objetos, ya que además de facilitar su modelado, permite generar código a partir del modelo (aunque esa funcionalidad no se ha utilizado para este desarrollo).

2.5 MySQL y JPA

MySQL [8] es un sistema gestor de bases de datos relacional de código abierto que se distribuye bajo la licencia GNU GPL [9]. Es uno de los sistemas más simples y de los más utilizados en todo el mundo.

JPA (Java Persistence API) es la API de persistencia desarrollada para la plataforma Java EE [10]. Proporciona un modelo para el mapeado objeto-relacional. Para la implementación de esta API se ha utilizado Hibernate.

Hibernate [11] es un framework de código abierto para el mapeado objeto-relacional en Java. Hibernate facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación mediante archivos XML declarativos o anotaciones en las propias clases.

2.6 Maven

Maven [12] es un gestor de dependencias de proyectos Java. Basado en el concepto de POM (*Project Object Model*), puede gestionar la construcción de una aplicación desde una pieza central de información. Maven facilita y automatiza el proceso de compilación y la gestión de dependencias, liberando al programador de esas tareas.

2.7 Tomcat

Tomcat [13] es un contenedor de *servlets* desarrollado por Apache. Puede funcionar como servidor web, y aunque al principio existía la percepción de que su uso de forma autónoma era solo recomendable para entornos de desarrollo, hoy en día se usa en entornos con alto nivel de tráfico y necesidad de una alta disponibilidad. Para desplegar una aplicación en Tomcat basta con colocar el fichero de despliegue WAR (*Web Application Archive*) en la carpeta correspondiente e iniciar el servidor.

2.8 Eclipse

Eclipse [14] es un IDE (*Integrated Development Environment*, Entorno de Desarrollo Integrado) para programación, especialmente orientado a Java (aunque da soporte a otros lenguajes). Es el IDE de referencia por sus características tales como editor de código con resaltado de sintaxis, compilación en tiempo real, depuración de código, posibilidad de añadir plugins o extensiones para ampliar sus funcionalidades, entre

otras cosas. Por eso, y por ser el IDE con el que más familiarizado estoy, ha sido el elegido para el desarrollo del proyecto.

2.9 Metodología utilizada

En este apartado se explica la metodología que se ha seguido para el desarrollo de la herramienta y los motivos por los que se ha utilizado.

La metodología que se ha seguido se basa en el Modelo Iterativo e Incremental [15]. Esta metodología se basa en la idea de desarrollar una implementación inicial, mostrárselo al cliente y desarrollar sucesivas versiones hasta obtener el sistema requerido. Se asemeja mucho a la forma en la que resolvemos los problemas. Normalmente elaboramos una solución en varios pasos y no de una vez, volviendo a un paso anterior si nos hemos equivocado. Desarrollar el software incrementalmente hace que sea más fácil y barato realizar cambios. La siguiente imagen representa gráficamente este modelo:

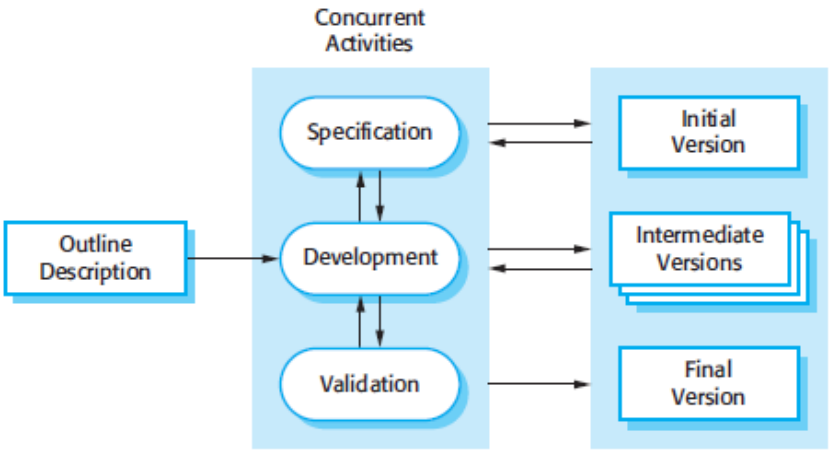

Ilustración 2.4 Modelo Iterativo e Incremental [6]

Cada incremento o versión del sistema incluye una nueva funcionalidad. Normalmente las primeras versiones incluyen las funcionalidades más importantes o más urgentes. En nuestro sistema la primera versión que se desarrolló contenía la gestión de gráficas ya que es la más importante porque las demás partes del sistema dependen de ella. En posteriores versiones se añadieron nuevas funcionalidades como la gestión de *dashboards* y la gestión de informes.

Las ventajas [15] de este modelo con respecto al modelo tradicional en cascada son:

- El coste de introducir cambios en los requisitos es reducido. La cantidad de análisis y documentación que hay que volver a realizar es menor que en el modelo en cascada.
- Es más fácil obtener las impresiones del cliente sobre el software que se está desarrollando. Es más fácil para los clientes juzgar el progreso a través de entregas periódicas de software funcional que a través de documentos.

- Los usuarios pueden comenzar a obtener valor del software antes debido a las entregas periódicas de software funcional.

El motivo del uso de esta metodología reside en el hecho de que el cliente de esta aplicación es la propia empresa CIC. Esto hace que sea más normal la necesidad de introducir cambios ya sea pequeñas modificaciones o cambios en los requisitos. Además, al poder hablar con el responsable del proyecto cara a cara cada día, es más fácil verificar que todo marcha según lo previsto y que la solución propuesta es la esperada.

3 Captura y análisis de requisitos

En esta sección se detalla el proceso seguido para la captura de requisitos y se detallan y analizan los requisitos obtenidos.

3.1 Captura de requisitos

La captura de requisitos es el proceso de descubrir y obtener las funcionalidades que deberá tener el sistema. De esta actividad se obtienen una serie de requisitos funcionales y no funcionales. Para capturar los requisitos se utilizaron dos técnicas: la entrevista y la observación de otros sistemas. La técnica principal fue la entrevista y la observación de otros sistemas sirvió para obtener nuevas ideas y requisitos.

3.1.1 Entrevista

Esta fue la técnica principal y gracias a la cual se obtuvieron la mayor parte de requisitos del sistema. El cliente de esta aplicación es la propia empresa CIC por lo que las entrevistas se realizaron con el gerente y responsable de este proyecto. En un primer momento él expuso la idea y en posteriores reuniones comentó que funcionalidades debía tener la aplicación. Una vez extraídos los requisitos, se realizó un documento de requisitos funcionales que posteriormente fue aprobado por él.

3.1.2 Observación de otros sistemas

Esta técnica sirvió de complemento a la entrevista y permitió identificar nuevas ideas así como afianzar las ya existentes. Para su ejecución se observaron algunas de las principales herramientas de *Business Intelligence* del mercado. La más útil fue Tableau [16]. Otras herramientas no comerciales que proporcionaban ejemplos de lo que se quería implementar fueron Redash [17] y la herramienta de ejemplo de Vaadin, QuickTickets Dashboard [18]. Este tipo de herramientas comerciales podrían servir para suplir las necesidades de CIC. Sin embargo, desarrollar una herramienta propia les proporciona un mayor control y siempre se adecuará mejor a sus necesidades que cualquier otra. Además, la intención de la empresa es, aparte de utilizar la herramienta, comercializarla en un futuro.

3.2 Análisis de requisitos

En esta sección se analiza el sistema que se quiere implementar así como los requisitos obtenidos de las actividades de captura.

3.2.1 Visión y alcance

El objetivo de la herramienta es poder generar gráficas que muestren los datos obtenidos tras la ejecución de *Custom Queries*. Los datos de cada gráfica se podrán filtrar de acuerdo a los parámetros de cada *Custom Query*. El usuario podrá crear

dashboards y añadir las gráficas generadas, de tal forma que cada *dashboard* mostrará los datos necesarios para el usuario. Finalmente se podrán generar informes que incluyan las gráficas para posteriormente exportarlos en formato PDF. La Ilustración 3.1 muestra el diagrama general del sistema.

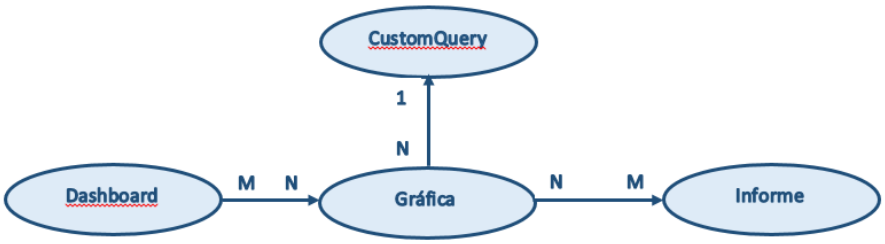

Ilustración 3.1 Diagrama general del sistema

El sistema estará formado por tres zonas, dos de ellas conectadas entre sí. La zona de gestión de gráficas, la zona de gestión de *dashboards* y la zona de gestión de informes, como se muestra en la Ilustración 3.2.

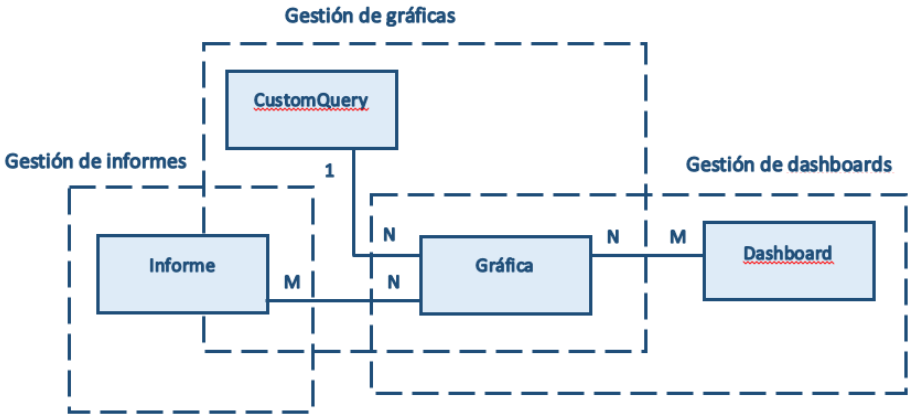

Ilustración 3.2 Partes del sistema

La zona de gestión de gráficas será donde se creen y se gestionen las gráficas a partir de los datos y parámetros de las *Custom Queries*. El usuario creará una gráfica del tipo más adecuado a sus necesidades. Durante el proceso de creación, deberá elegir la *Custom Query* que tomará la gráfica como entrada para mostrar los datos. Una vez creada una gráfica se podrá configurar cambiando los parámetros de la *Custom Query*. También se podrá cambiar la *Custom Query* que toma como entrada, así como el tipo de gráfica, la visibilidad y el periodo de refresco. Para configurar la gráfica, ésta tiene que estar en modo edición.

La zona de gestión de *dashboards* es donde el usuario puede crear y gestionar *dashboards*. Una vez creados, se podrán añadir gráficas existentes. Las gráficas se podrán ampliar para verlas a pantalla completa, se podrán configurar y se podrán eliminar del *dashboard* sin que eso implique su eliminación del sistema.

La zona de gestión de informes es donde el usuario creará los informes que contendrán las gráficas. Una vez creado un informe, el usuario tendrá a su disposición

un editor en el que podrá añadir las gráficas existentes en el sistema, así como texto para enriquecerlo. Una vez creado, se podrá generar un PDF.

3.2.2 Requisitos funcionales

Los requisitos funcionales [15] son declaraciones de los servicios que debe proporcionar el sistema, como debería reaccionar ante entradas particulares y como debería comportarse en situaciones particulares. Según la información obtenida del proceso de captura de requisitos, los requisitos funcionales del sistema son los siguientes:

Tabla 3.1 Requisitos funcionales

Referencia	Requisito
RF0.01	El usuario podrá acceder a los menús “Gráficas”, “Dashboards” e “Informes” desde el menú principal.
RF1.01	El usuario podrá generar gráficas que muestren los datos obtenidos tras la ejecución de una *Custom Query*. Las entradas de cada gráfica serán la *Custom Query* correspondiente además de un título, descripción, visibilidad y estado.
RF1.02	El usuario podrá seleccionar el tipo de gráfica más adecuado a sus necesidades.
RF1.03	La visibilidad de las gráficas podrá ser pública o privada. Si la gráfica es privada solo puede verla y editarla el propietario.
RF1.04	Existirán dos estados para las gráficas. El modo edición en el que es posible cambiar la configuración y el modo publicación en el que no es posible.
RF1.05	Una vez generadas las gráficas, el usuario podrá interactuar con ellas de varias formas. Podrá cambiar parámetros y realizar filtrados, resaltar datos, eliminar datos, ampliar zonas, cambiar colores y revertir todos los cambios realizados.
RF1.06	El propietario de la gráfica o cualquier usuario si es pública, podrá cambiar la configuración. Esta configuración consiste en los parámetros de la *Custom Query*, el tipo de gráfica, la visibilidad y el periodo de refresco.
RF1.07	El sistema mostrará las gráficas existentes en una tabla pudiendo el usuario ver cada una en un panel aparte.
RF1.08	El usuario podrá eliminar las gráficas que haya creado. La eliminación de una gráfica no implica la eliminación de la *Custom Query* a partir de la que se generó.
RF2.01	El usuario podrá crear *dashboards* introduciendo un título, una descripción y un tipo.
RF2.02	El tipo de *dashboard* permite especificar el número y disposición de los contenedores.
RF2.03	Una vez creado el *dashboard*, el usuario podrá añadir gráficas existentes en el sistema a cada contenedor.
RF2.04	El usuario podrá ampliar cada contenedor para mostrar su contenido a pantalla completa.
RF2.05	El usuario podrá configurar cada gráfica dentro de un *dashboard*. La configuración se realizará de la misma forma explicada en el requisito RF1.06.
RF2.06	El usuario podrá eliminar gráficas de un *dashboard*. La eliminación de una gráfica de un *dashboard* no implica su eliminación del sistema. La eliminación del sistema se hará desde el panel “Gráficas” como se explica en el requisito

	RF1.08.
RF2.07	El usuario podrá eliminar los *dashboards* que haya creado. Cuando se elimina un *dashboard* las gráficas contenidas en él no serán eliminadas del sistema.
RF3.01	Por cada *dashboard* el usuario podrá generar un informe que muestre todas las gráficas contenidas en él.
RF3.02	El usuario podrá crear informes introduciendo un título y una descripción.
RF3.03	Una vez creado un informe se abrirá un editor en el que el usuario puede añadir gráficas y bloques de texto.
RF3.04	Los informes se guardarán en el sistema, pudiendo el usuario exportarlos en formato PDF para su uso externo.

3.2.3 Requisitos no funcionales

Los requisitos no funcionales son restricciones en los servicios o funciones ofrecidas por el sistema. Pueden incluir restricciones de tiempo o en el proceso de desarrollo, entre otras. Los requisitos no funcionales del sistema propuesto son:

Tabla 3.2 Requisitos no funcionales

Referencia	Requisito	Tipo
RNF01	Los usuarios de la herramienta se validarán a través del servicio de seguridad.	Seguridad
RNF02	Los roles necesarios para realizar las operaciones de la herramienta serán los que se muestran en la tabla 4.	Seguridad
RNF03	En caso de producirse una pérdida de los datos de la herramienta, se podrá recuperar un backup de la base de datos.	Seguridad
RNF04	Las transiciones entre pantallas deberán ser fluidas.	Rendimiento
RNF05	Las operaciones más pesadas deberán realizarse en un tiempo razonable.	Rendimiento
RNF06	La aplicación será fácil de usar y cualquiera que esté acostumbrado a usar la herramienta principal no debería tener problemas con su utilización.	Usabilidad
RNF07	Se podrá descargar un manual de ayuda en el cual se realiza una explicación detallada de manejo de la herramienta para el usuario.	Usabilidad

Tabla 3.3 Roles definidos en la herramienta LUCA después de incluir este módulo

ROL	DETALLE
ROLE_CREADOR_GRAFICAS	Este rol podrá crear, ver, modificar y eliminar gráficas.
ROLE_CREADOR_DASHBOARDS	Este rol podrá crear, ver, modificar y eliminar *dashboards*.
ROLE_CREADOR_INFORMES	Este rol podrá crear, ver, modificar y eliminar informes.
ROLE_VIEW_GRAFICAS	Este rol solo podrá ver gráficas. Para ello el usuario deberá poseer las autorizaciones correspondientes a nivel de entorno y sistema para ejecutar la *Custom Query* asociada a la gráfica.
ROLE_VIEW_DASHBOARDS	Este rol solo podrá ver *dashboards*. Para ello el usuario deberá poseer las autorizaciones correspondientes a nivel de entorno y sistema para ejecutar las *Custom Queries* asociadas a las gráficas.
ROLE_VIEW_INFORMES	Este rol solo podrá ver informes. Para ello el usuario deberá poseer las autorizaciones correspondientes a nivel de entorno y sistema para ejecutar las *Custom Queries* asociadas a las gráficas.

3.2.4 Análisis de casos de uso

Con la información anterior, se pueden extraer los casos de uso del sistema. Se dividen en dos niveles: casos de uso de alto nivel y casos de uso de bajo nivel. A continuación se muestran todos los casos de uso.

3.2.4.1 Casos de uso de alto nivel

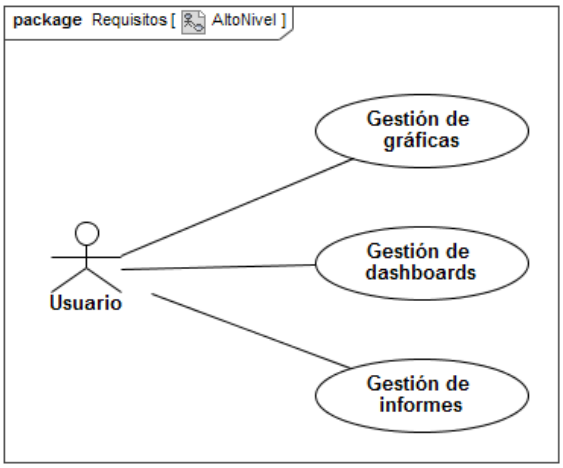

Ilustración 3.3 Casos de uso de alto nivel

Como muestra la Ilustración 3.3, estos casos de uso corresponden a las tres grandes zonas del sistema. Cada uno de ellos contiene una serie de casos de uso de bajo nivel que son los que definen el sistema. A continuación se muestran esos casos de uso así como el análisis detallado de los que son menos triviales.

3.2.4.2 Casos de uso de Gestión de Gráficas

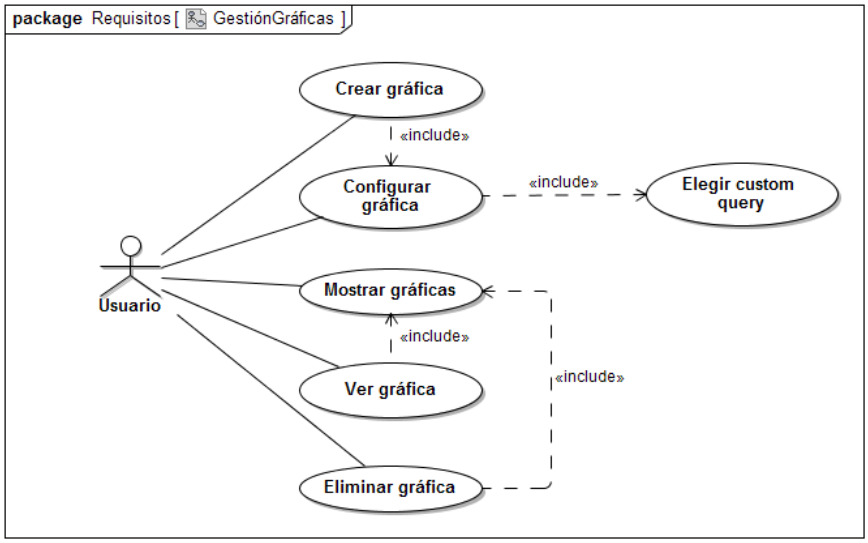

Ilustración 3.4 Casos de uso de Gestión de gráficas

Los casos de uso menos triviales son “Configurar gráfica” y “Elegir Custom Query” que se analizan a continuación:

CU-Gráficas-02	**Configurar gráfica**	
Actor principal	Usuario	
Descripción	El usuario configura una gráfica para que muestre los datos que él desea.	
Precondición	El usuario debe estar registrado en la herramienta, tener seleccionada una gráfica y ser propietario de la misma.	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario elige la opción de configurar sobre la gráfica seleccionada.
	2	El sistema comprueba el rol del usuario
	3	El sistema comprueba que la gráfica está en modo edición y abre la pantalla de configuración de la gráfica.
	4	El usuario ejecuta el caso de uso “Elegir Custom Query”.

	5	El usuario introduce el resto de campos necesarios y confirma los cambios.
	6	El sistema comprueba que están introducidos todos los campos obligatorios y guarda la configuración.
	7	El sistema muestra la gráfica con la configuración elegida.
Postcondición	La gráfica muestra los datos de la *Custom Query* seleccionada con la configuración indicada por el usuario.	
Excepciones	**Paso**	**Acción**
	2.a	Si el usuario no posee el rol adecuado, el sistema muestra una advertencia y el caso de uso finaliza.
	3.a	Si la gráfica no está en modo edición finaliza el caso de uso.
	6.a	El sistema comprueba que todos los campos obligatorios no están introducidos o que su validación no es correcta, muestra un mensaje de alerta y espera a que el usuario lo corrija.

CU-Gráficas-03	**Elegir Custom Query**	
Actor principal	Usuario	
Descripción	El usuario selecciona una *Custom Query* entre la lista de *Custom Queries* disponibles.	
Precondición	El usuario debe estar registrado en la herramienta y debe haber ejecutado el caso de uso “Configurar gráfica”.	
Secuencia Normal	**Paso**	**Acción**
	1	El sistema muestra la lista de *Custom Queries* prefiltradas (sus variables de entrada ya tienen valores definidos).
	2	El usuario selecciona la *Custom Query* que desea.
Postcondición	La *Custom Query* es añadida a la gráfica.	
Excepciones	**Paso**	**Acción**

3.2.4.3 Casos de uso de Gestión de Dashboards

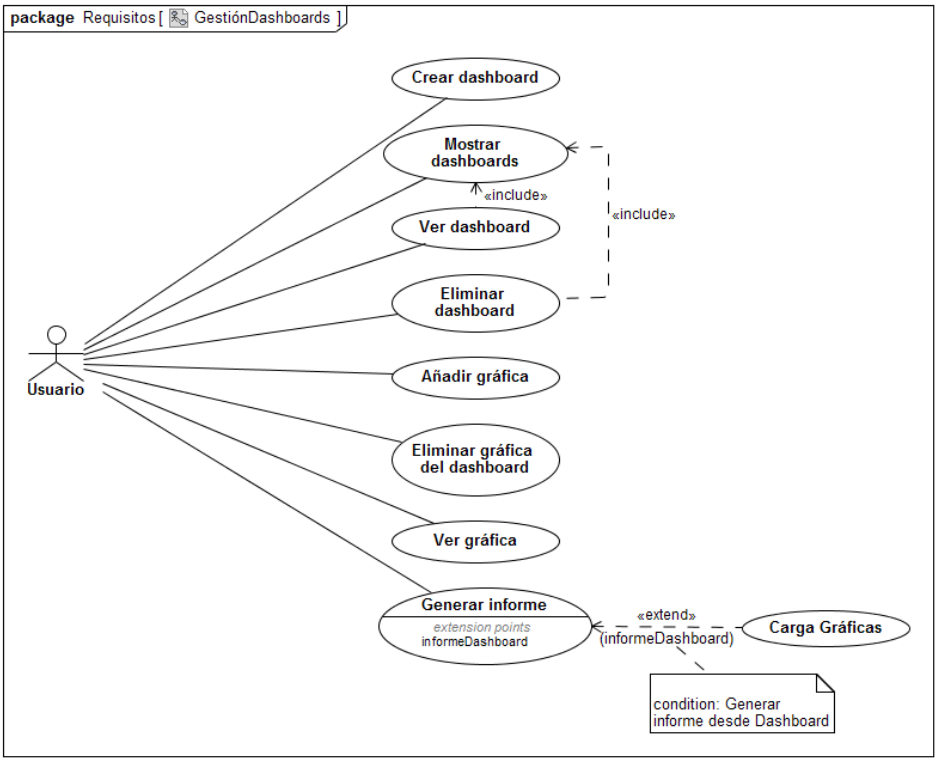

Ilustración 3.5 Casos de uso de Gestión de dashboards

Los casos de uso menos triviales son "Ver dashboard" y "Generar informe" que se analizan a continuación:

CU-Gráficas-03	**Ver *dashboard***	
Actor principal	Usuario	
Descripción	El usuario verá el *dashboard* seleccionado en una pantalla aparte.	
Precondición	El usuario debe estar registrado en la herramienta.	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario selecciona la opción "Gestión de dashboards" en el menú de "Dashboards".
	2	El sistema comprueba que el usuario posee el rol adecuado y abre la pantalla de gestión de *dashboards*.
	3	El usuario selecciona el botón refrescar y se ejecuta el caso de uso "Mostrar dashboards".
	4	El usuario selecciona un *dashboard* y elige la opción de "Ver dashboard".

	5	El sistema abre una pantalla que muestra el *dashboard*.
Postcondición	Se abre una nueva pantalla que muestra el *dashboard* con las gráficas correspondientes.	
Excepciones	**Paso**	**Acción**
	2.a	El sistema comprueba que el usuario no posee el rol adecuado, muestra una advertencia y el caso de uso finaliza.

CU-Dashboards-10	**Generar informe**	
Actor principal	Usuario	
Descripción	El sistema generará un informe que mostrará las gráficas contenidas en el *dashboard*.	
Precondición	El usuario debe estar registrado en la herramienta y haber seleccionado un *dashboard*.	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de generar informe.
	2	Si se ha elegido generar el informe desde un *dashboard* se ejecuta el caso de uso Cargar Gráficas.
	2	El usuario modifica el informe añadiendo o eliminando nuevas gráficas (caso de uso “Añadir Gráfica a Informe”) o añadiendo texto (caso de uso “Añadir texto”).
	3	El usuario selecciona la opción de guardar.
Postcondición	El sistema crea un informe que por defecto mostrará las gráficas contenidas en el *dashboard* o la información introducida en el editor.	
Excepciones	**Paso**	**Acción**

3.2.4.4 Casos de uso de Gestión de informes

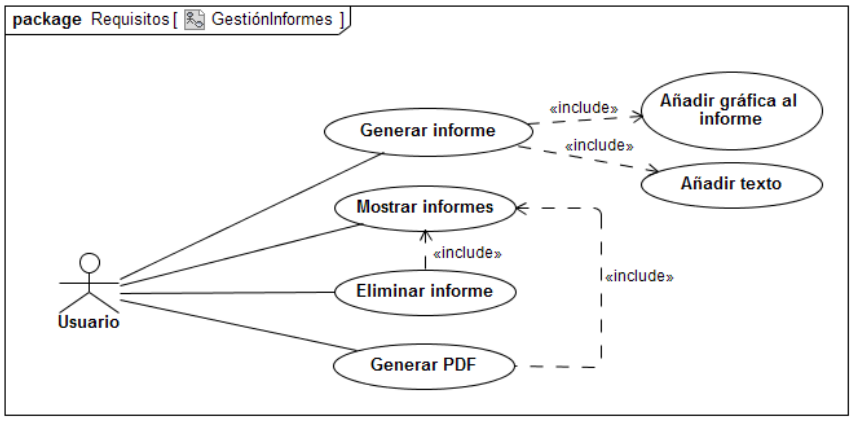

Ilustración 3.6 Casos de uso de Gestión de informes

El caso de uso menos trivial es "Añadir gráfica al informe" que se analiza a continuación:

CU-Informes-06	**Añadir gráfica al informe**	
Actor principal	Usuario	
Descripción	El usuario añadirá una gráfica al informe.	
Precondición	El usuario debe estar registrado en la herramienta y haber ejecutado el caso de uso "Generar informe".	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario selecciona la opción añadir gráfica en el editor de informes
	2	El usuario elige una gráfica de entre las gráficas disponibles.
	3	El sistema añade la gráfica al informe
Postcondición	La gráfica es añadida al informe.	
Excepciones	**Paso**	**Acción**

4 Diseño del sistema

El diseño software es una actividad creativa en la que se identifican los componentes del sistema y sus relaciones, basándose en los requisitos obtenidos.

En esta sección se detalla cómo se diseñó el sistema. Para diseñar el sistema lo primero fue elegir la arquitectura. A continuación, con la información obtenida de la fase de captura y análisis de requisitos se definieron tanto el conjunto de componentes, con sus correspondientes interfaces, como el conjunto de clases que conforman el diseño detallado del sistema.

4.1 Arquitectura basada en componentes

La arquitectura basada en componentes [15] surgió a finales de los 90 como necesidad de reutilizar partes de los sistemas que se desarrollaban, ya que el desarrollo orientado a objetos no permitía esa reutilización.

La arquitectura basada en componentes consiste en organizar los sistemas mediante componentes independientes especificados por sus interfaces. La separación entre las interfaces y su implementación debe de ser clara, lo que significa que una implementación de un componente puede ser cambiada por otra sin cambiar otras partes del sistema. La integración entre componentes se deja a cargo de algún *middleware*. De tal forma que gracias a ese middleware, componentes independientes pueden comunicarse para formar un sistema completo.

En el caso de la aplicación que se ha desarrollado, podría considerarse un componente de LUCA. Desde el primer momento se pensó en esta aplicación como un módulo a integrar en LUCA. Además existe la posibilidad si fuese necesario de integrar este módulo en otras aplicaciones de la empresa que tengan la misma arquitectura. Así mismo, el propio módulo posee una estructura interna basada en componentes.

4.2 Arquitectura de la aplicación

La arquitectura del módulo está definida por la arquitectura de la aplicación en la que será integrado. Esta arquitectura es una arquitectura en tres capas, compuesta por la capa de presentación, la capa de negocio y la capa de acceso a datos.

4.2.1 Arquitectura en tres capas

La arquitectura en tres capas es la arquitectura más usada en este tipo de sistemas sencillos. Esta arquitectura consigue separar la lógica de negocio de la presentación y de la gestión de los datos. Para ello, una aplicación que utiliza esta arquitectura estará dividida en capas que implementan la funcionalidad correspondiente. Cada capa solo se comunica con la capa inferior. Esta organización permite que el sistema sea más resistente a cambios y por tanto más mantenible. Por ejemplo, si el día de mañana se

quiere cambiar el sistema gestor de base de datos, solo habría que modificar la capa de acceso a datos. A continuación se explica cuál es la función de cada capa:

- **Capa de presentación:** Esta es la capa con la que interactúa directamente el usuario. Su responsabilidad es mostrar y gestionar la interfaz de usuario del sistema. Dicha interfaz se utilizará para mostrar datos o para introducirlos. Para ello esta capa se comunica con la capa de negocio. A la hora de introducir datos en el sistema es responsabilidad de esta capa que esos datos sean correctos. Por ejemplo, en una tienda online, si queremos introducir el precio de un artículo, en esta capa se gestiona la restricción de que no se permita introducir un número negativo. En nuestro caso, la capa de presentación gestiona la interfaz web de la aplicación construida con Vaadin.
- **Capa de negocio**: Esta capa es la encargada de gestionar las restricciones de negocio de la aplicación. Aquí es donde se gestiona qué se hace con los datos. Se encarga de enviar y recibir los datos de la capa de presentación gestionando las peticiones que se realizan desde la misma. Se comunica con la capa de acceso a datos para obtener o guardar los datos pedidos. Siguiendo con el ejemplo anterior, una vez que tenemos un precio que no es un número negativo, es responsabilidad de esta capa gestionar la restricción de negocio de que el precio de un artículo no sea inferior a diez euros para que sea rentable su venta, por ejemplo.
- **Capa de acceso a datos:** Esta capa es la encargada de persistir los datos del sistema. La función de esta capa es guardar y recuperar datos independientemente de las capas que haya por encima. Para ello, se utiliza el modelo DAO (*Data Access Object*) que proporciona interfaces que realizan las típicas operaciones CRUD (*Create, Read, Update y Delete*).

La Ilustración 4.1 representa la arquitectura de la aplicación LUCA, y por tanto, de la aplicación bajo desarrollo. La capa de presentación está desarrollada con Vaadin. Para ese desarrollo se utilizó el patrón Modelo-Vista-Presenter explicado en la siguiente sección. La capa de negocio contiene una serie de servicios que son los que se encargan de gestionar y aplicar la lógica de negocio a los datos. Finalmente, la capa de acceso a datos proporciona interfaces DAO con las que se comunican los servicios para realizar las operaciones CRUD. Para dar soporte a todas las capas se utiliza Spring, que proporciona los servicios que fueron explicados previamente.

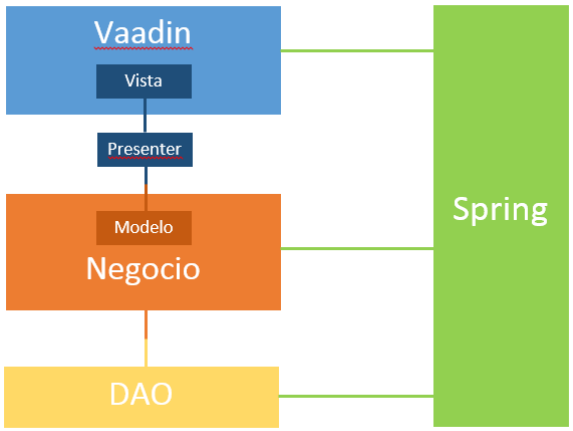

Ilustración 4.1 Arquitectura de la aplicación

4.2.2 Patrón Modelo-Vista-Presenter

El patrón Modelo-Vista-Presenter (MVP) [2] es uno de los más comunes en el desarrollo de aplicaciones con Vaadin. Es similar al patrón Modelo-Vista-Controlador. La diferencia es que el patrón Modelo-Vista-Presenter permite separar la capa de presentación de la lógica de la misma manejando la Vista a través de un interfaz. La Vista no interactúa directamente con el modelo. La Ilustración 4.2 muestra un ejemplo sencillo de aplicación del patrón MVP.

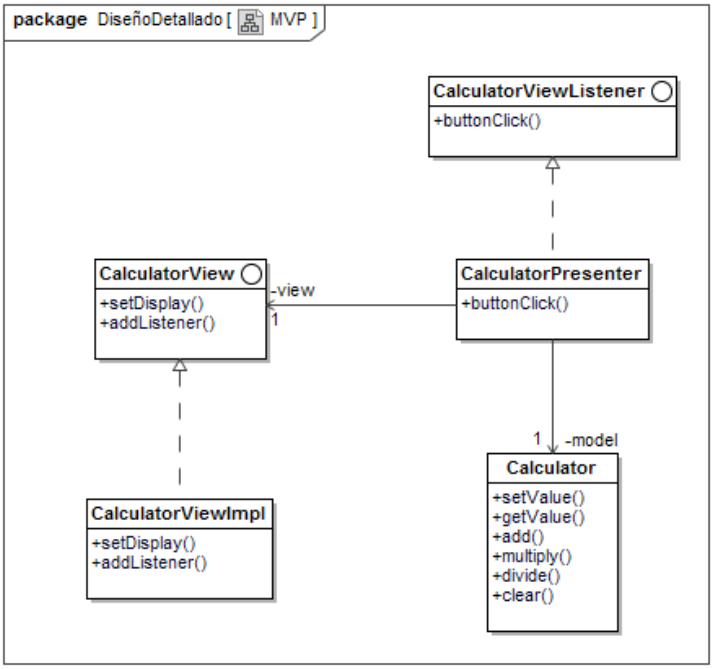

Ilustración 4.2 Ejemplo de aplicación del patrón MVP

El Modelo está representado en la clase `Calculator`. Esta clase contiene un modelo de datos y algunas operaciones. La clase `CalculatorViewImpl` es la implementación en Vaadin de la Vista definida en `CalculatorView`. La interacción del usuario es gestionada por el Presenter que solo conoce la interfaz de la vista por lo que no necesita conocer nada sobre Vaadin. A continuación se explican las funciones de cada parte del patrón.

- **Modelo**: El modelo es el encargado de guardar los datos gestionados por el sistema. Las operaciones del modelo se realizan a petición del Presenter.
- **Vista**: El propósito de la vista es mostrar datos y recibir la interacción del usuario. La interacción del usuario es gestionada por el Presenter de forma independiente de la implementación, por lo que el Presenter no necesita conocer nada de Vaadin. Cuando el usuario realiza una interacción en la Vista, se genera un evento que controla el Presenter y normalmente resulta en una operación en el Modelo.
- **Presenter**: El Presenter se coloca entre la Vista y el Modelo y se encarga de gestionar la interacción del usuario con la Vista. Además se encarga de obtener los datos provenientes del modelo que serán mostrados en la Vista.

En esta aplicación, cuando se abre una nueva zona (por ejemplo el panel de gestión de gráficas) lo que se abre en realidad es un nuevo Presenter que tiene vinculados sus correspondientes Modelo y Vista.

4.2.3 Arquitectura específica de la aplicación

A continuación se muestra la arquitectura específica de la aplicación desarrollada. Para ello, se muestran dos diagramas de componentes, uno muestra los componentes de las capas de negocio y persistencia y otro los de la vista. Así mismo, se utilizan tres diagramas de clases para mostrar las operaciones de cada interfaz.

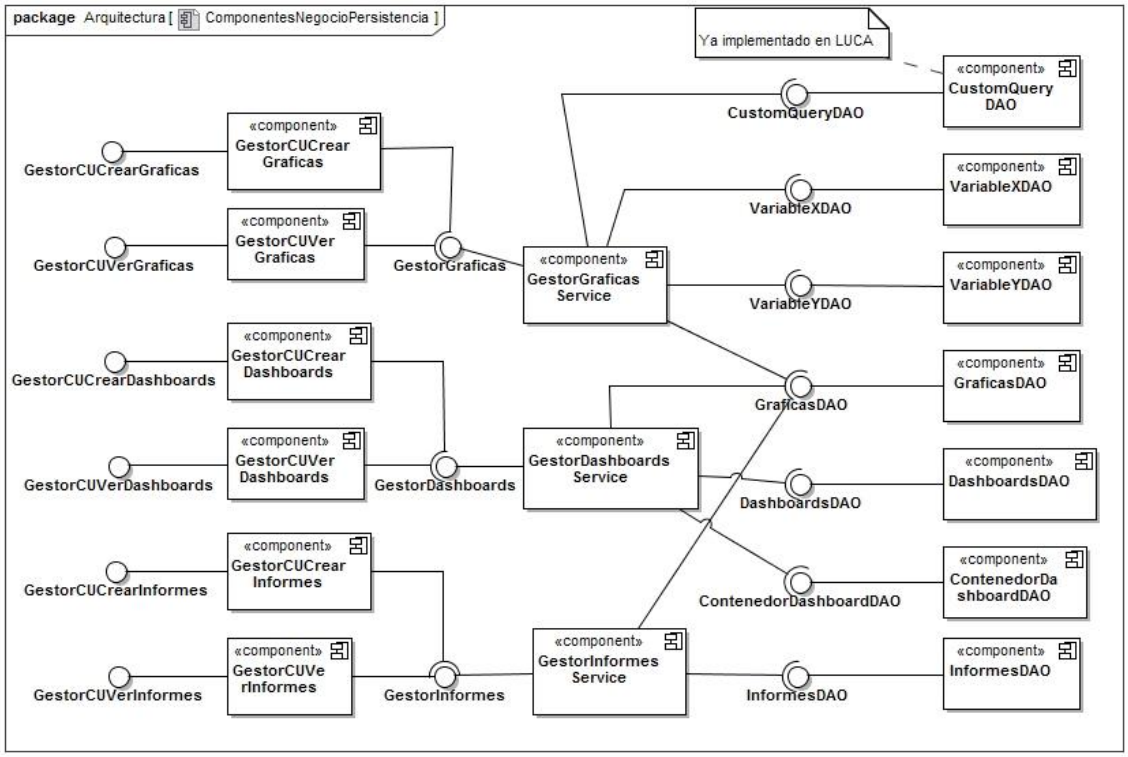

Ilustración 4.3 Diagrama de componentes de negocio y persistencia

Como se observa en la Ilustración 4.3, la capa de negocio se ha dividido a su vez en dos subcapas para facilitar la gestión de la seguridad. En LUCA, los componentes denominados "*GestorCUXXX*" son los encargados de aplicar la seguridad. Los métodos de estos componentes se anotan con las anotaciones @PreAuthorize de Spring para gestionar la seguridad a nivel de método. Estos componentes utilizan a los de la capa de servicio, que son los que implementan la funcionalidad deseada haciendo uso de los componentes DAO. De tal forma que con esta organización, se facilita al programador la implementación de estas interfaces.

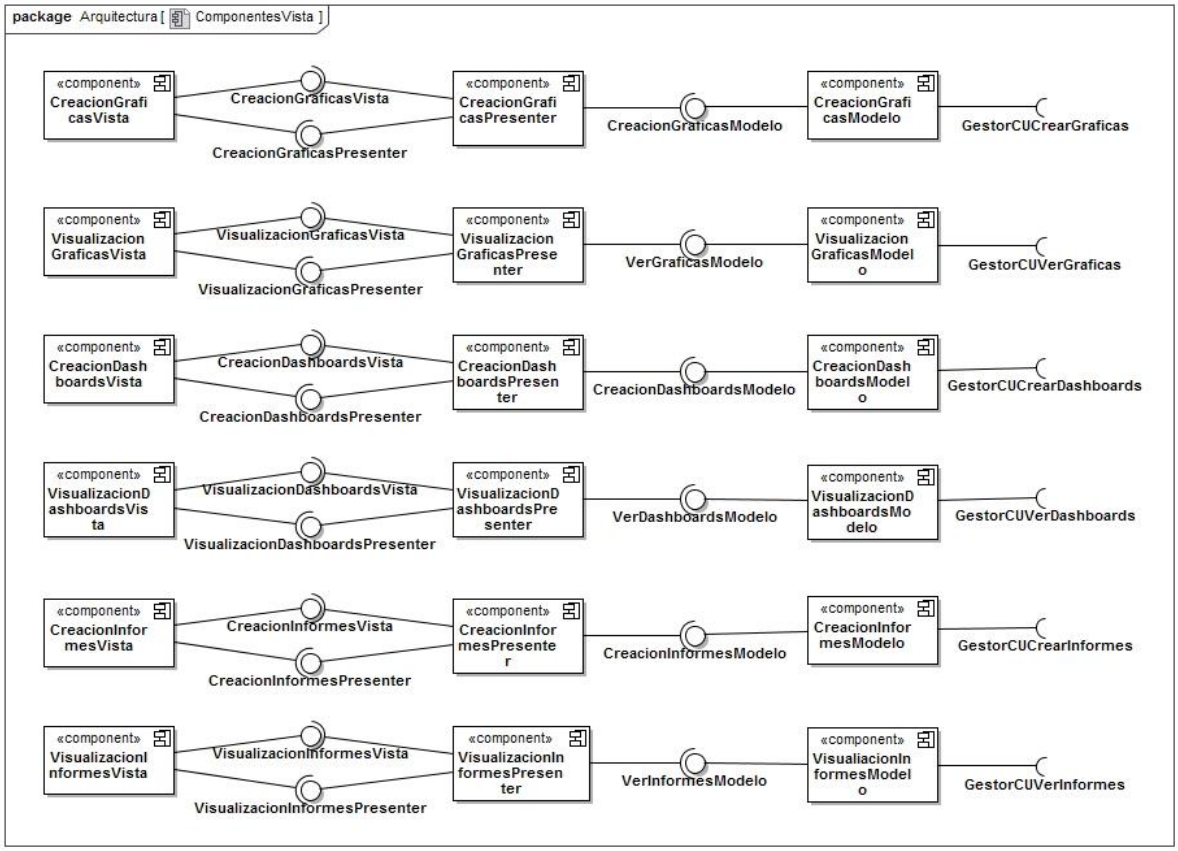

Ilustración 4.4 Diagrama de componentes de la vista

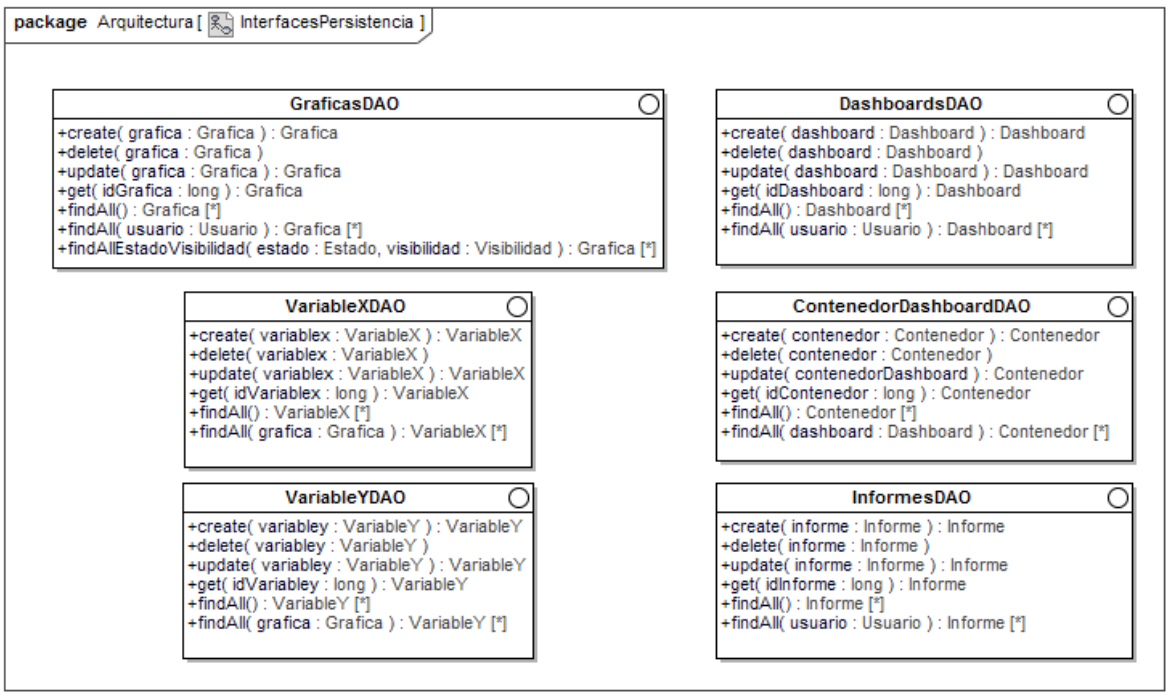

Ilustración 4.5 Operaciones de las interfaces de persistencia

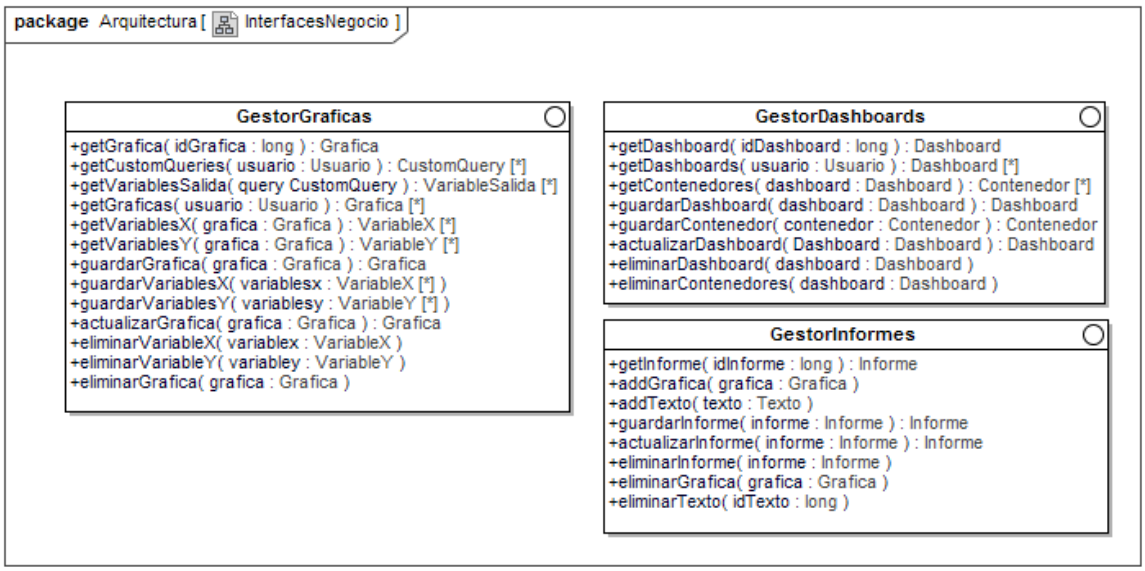

Ilustración 4.6 Operaciones de las interfaces de negocio

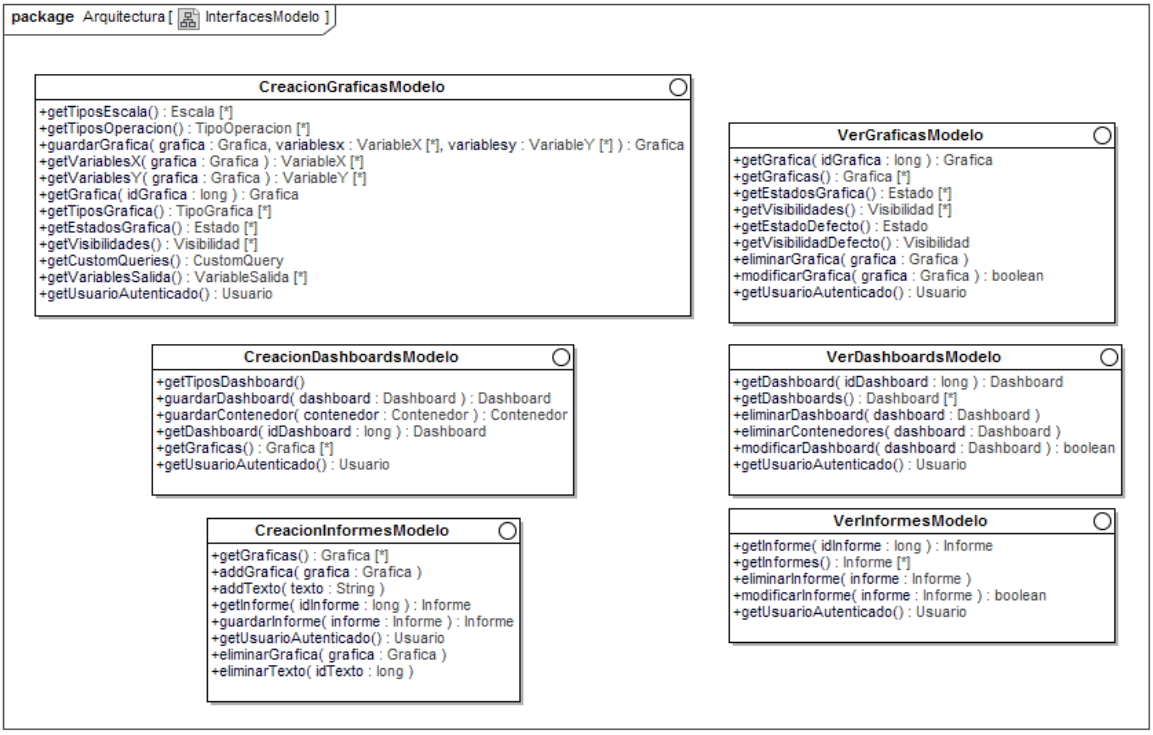

Ilustración 4.7 Operaciones de las interfaces del modelo

Finalmente, la Ilustración 4.8 muestra el diagrama de despliegue de LUCA para indicar cómo se ha empaquetado y desplegado el módulo desarrollado en este proyecto. En *luca-recursos.jar*, *luca-registros.jar* y *luca-web.war* se incluyen todos los componentes definidos anteriormente, que representan la implementación del módulo desarrollado, sin embargo solo se muestra el despliegue de `GestorGraficasService`, `VariableYDAO` y `CreacionGraficasVista` a modo de ejemplo para no saturar el diagrama. En el resto de artefactos no se ha añadido ningún componente del módulo que está siendo descrito en esta memoria.

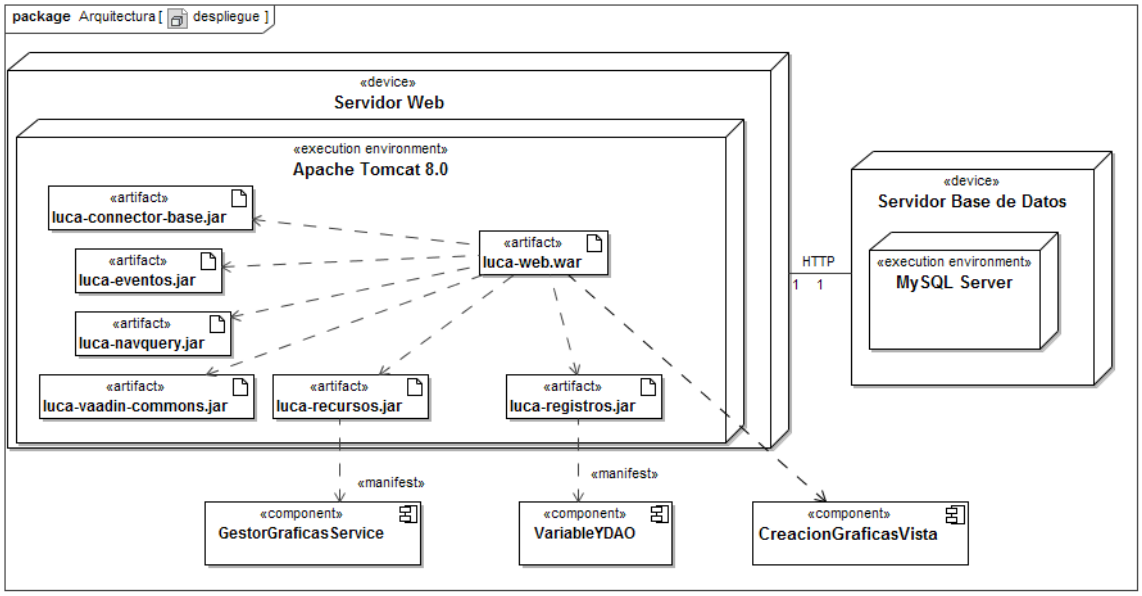

Ilustración 4.8 Diagrama de despliegue de LUCA

4.3 Diseño detallado

A continuación se muestra el modelo de diseño detallado (en forma de diagrama de clases) del sistema desarrollado. El modelo de clases define los objetos que contendrá el sistema y como se relacionan entre ellos para reflejar la realidad deseada. Este modelo representa a alto nivel, las entidades del sistema y sus relaciones. Para su realización se utilizó UML y se desarrolló el siguiente diagrama, dividido en dos (Ilustraciones 4.9 y 4.10), para una mejor visualización.

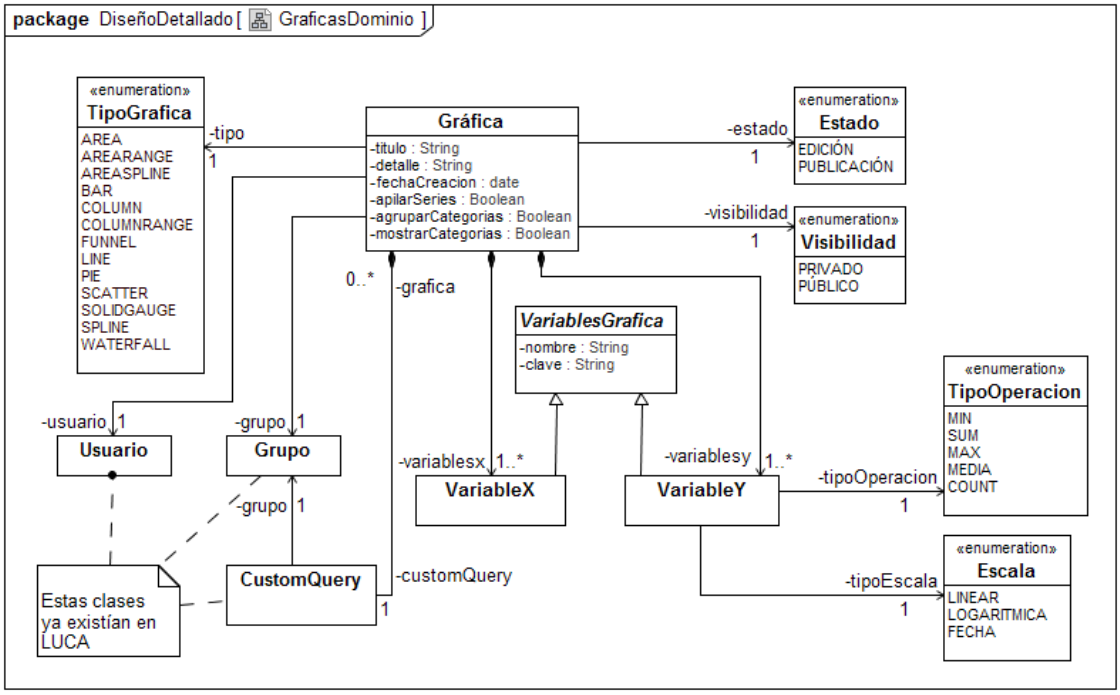

Ilustración 4.9 Modelo de dominio centrado en la gestión de gráficas

Las clases `VariableX` y `VariableY` representan las variables que puede tener una gráfica. Estas variables se corresponden con las variables de salida de la correspondiente *Custom Query*. Las variables "x" son las variables por las que se agrupan los datos, mientras que las variables "y" representan los valores de esos datos. Por ejemplo, usando el mismo ejemplo utilizado en la introducción, si queremos mostrar el número de fallos en las ejecuciones de consultas por día, la variable "x" será *fecha,* para mostrar en el eje "x" todos los días, mientras que la variable "y" será *fallos*, para mostrar en el eje "y" el número de fallos.

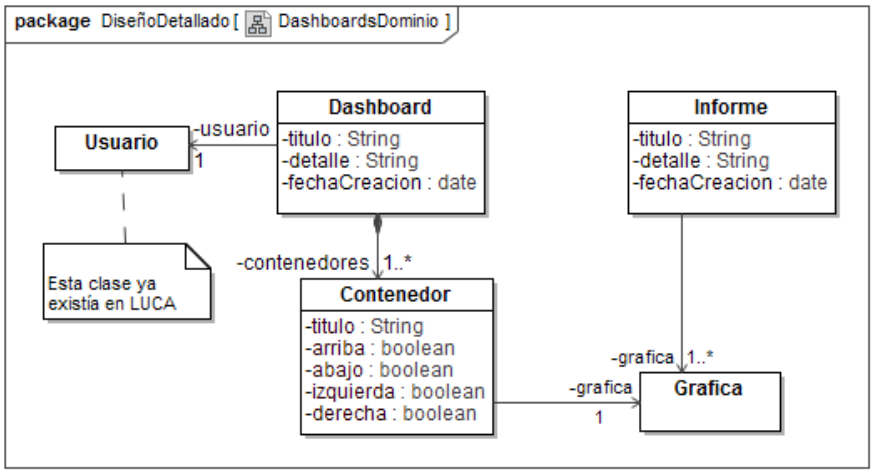

Ilustración 4.10 Modelo de dominio centrado en la gestión de informes y dashboards

5 Construcción del sistema

En este apartado se detallan los pasos seguidos para implementar la aplicación y algunos detalles de la misma.

5.1 Capa de acceso a datos

El primer paso fue la implementación de la capa de acceso a datos. Para ello lo primero fue añadir las tablas necesarias a la base de datos existente, lo cual se realizó con la ayuda de MySQL Workbench [19]. La siguiente imagen muestra la definición de la tabla "grafica" en MySQL Workbench:

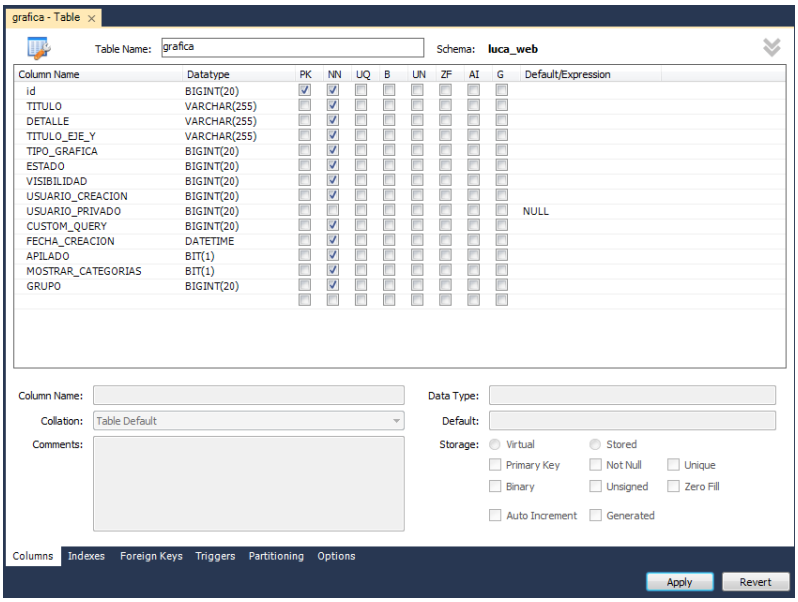

Ilustración 5.1 Tabla " grafica" en MySQL Workbench

Una vez creadas todas las tablas, se implementaron y anotaron apropiadamente las clases de dominio en Java para la realización del mapeo objeto-relacional. A continuación se muestra, a modo de ejemplo, una parte de la clase `Grafica` con las anotaciones necesarias para el mapeo basado en Hibernate:

```
@Entity
@Table(name="GRAFICA")
public class Grafica implements Serializable {

	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private Long id;

	@Column(name="TITULO", nullable=false)
	private String titulo;

	@Column(name="TITULO_EJE_Y", nullable=false)
	private String tituloEjey;

	@JoinColumn(name = "USUARIO_PRIVADO", nullable=true)
	@ManyToOne(fetch = FetchType.EAGER)
	private Usuario usuarioPrivado;

	@JoinColumn(name = "USUARIO_CREACION", nullable=false)
	@ManyToOne(fetch = FetchType.EAGER)
```

```
        private Usuario usuarioCreacion;

        @Column(name = "FECHA_CREACION", nullable=false)
        private Date fechaCreacion;

        @JoinColumn(name = "CUSTOM_QUERY", nullable = false)
        @ManyToOne(fetch = FetchType.EAGER)
        private CustomQueryFavorita queryFavorita;

        @JoinColumn(name = "GRUPO")
        @ManyToOne(fetch = FetchType.EAGER, nullable = false)
        private Grupo grupo;

        . . .
}
```

El siguiente paso consistió en la declaración e implementación de los componentes de la capa de persistencia. Todos ellos heredan de un componente ya existente, que siguiendo el patrón GenericDAO [20], implementa las operaciones CRUD (*create*, *delete*, *update* y *get)* genéricas, por lo que solo hubo que implementar los métodos que requieren realizar consultas personalizadas a la base de datos, como por ejemplo obtener la lista de gráficas filtradas por estado y visibilidad. A continuación se muestra el código de la implementación del método *findAllEstadoVisibilidad()* de la interfaz `GraficaDAO`:

```
@Override
public List<Grafica> findAllEstadoVisibilidad(Usuario usuario,
                VisibilidadGrafica visibilidad, EstadoGrafica estado) {

        CriteriaBuilder cb = em.getCriteriaBuilder();
        CriteriaQuery<Grafica> criteria = cb.createQuery(domainClass);
        Root<Grafica> grafica = criteria.from(domainClass);

        List<Predicate> preds = new ArrayList<Predicate>();

        preds.add(cb.and(cb.equal(grafica.get("estado"), estado),
                cb.or(cb.notEqual(grafica.get("visibilidad"),  visibilidad),
                cb.equal(grafica.get("usuarioCreacion"), usuario))));

        criteria.select(grafica);
        Predicate[] predicados = preds.toArray(new Predicate[0]);
        criteria.where(predicados);

        return em.createQuery(criteria).getResultList();
}
```

5.2 Capa de negocio

Una vez implementada la capa de persistencia, se implementó la capa de negocio. Como ya se explicó en el apartado 4.3.2, la capa de negocio consta en realidad de dos capas, una con los servicios que proporcionan la funcionalidad deseada y otra capa por encima que realiza la gestión de la seguridad. A continuación se muestran, a modo de ejemplo, las implementaciones del método *guardarGrafica()* tanto en el componente `GestorGraficasService` como en GestorCUCrearGrafica. En el caso de `GestorCUCrearGrafica` también se muestra la declaración del método en su correspondiente interfaz, dónde se realiza la configuración de la seguridad.

```
public interface GestorCUCrearGrafica {

        @PreAuthorize(AUTHORIZE_CREAR_GRAFICA)
```

```
        public Grafica guardarGrafica(Grafica grafica,
                                List<VariableX> variablesx,
                                List<VariableY> variablesy);
        . . .

}


@Service
public class GestorCUCrearGraficaImpl implements GestorCUCrearGrafica {

        @Override
        public Grafica guardarGrafica(Grafica grafica,List<VariableX> variablesx,
        List<VariableY> variablesy) {

                return gestorGraficasService.guardarGrafica(grafica);
        }
        . . .
}

@Service
public class GestorGraficasServiceImpl implements GestorGraficasService {


        @Override
        public Grafica guardarGrafica(Grafica grafica) {
                // Guardamos las variables "x" y las variables "y"
                for (VariableX variableX: grafica.getVariablesX()) {
                        variableXDAO.guardarVariableX(variableX);
                }

                for (VariableY variableY: grafica.getVariablesY()) {
                        variableYDAO.guardarVariableY(variabley);
                }

                return graficaDAO.guardarGrafica(grafica);
        }
        . . .
}
```

5.3 Capa de presentación

En la capa de presentación se aplica el patrón Modelo-Vista-Presenter explicado anteriormente. En LUCA existen clases abstractas que implementan este patrón, por tanto todos los modelos, vistas y presenters creados extienden a esas clases.

5.3.1 Modelo

El modelo usa las interfaces de negocio *"GestorCUXXX"* para implementar sus métodos. De tal forma que el método *guardarGrafica()* de la clase `CreacionGraficasModelo` quedaría:

```
@Component
@Scope("session")
public class CreacionCustomQueriesModeloImpl implements CreacionCustomQueriesModelo {

        @Override
        public Grafica guardarGrafica(Grafica grafica, List<VariableX> variablesx,
        List<VariableY> variablesy) {

                Grafica result = null;
                Long idVisibilidad = grafica.getVisibilidad() != null ? grafica
                                .getVisibilidad().getId() : null;
                boolean isPrivada = idVisibilidad != null ? idVisibilidad
                                .equals(VisibilidadGrafica.PRIVADA) : false;
```

```
                // Si es privada para el usuario lo indicamos
                grafica.setUsuarioPrivado(isPrivada ? getUsuarioAutenticado(): null);

                // Si la gráfica tiene id es que se va a actualizar en vez de crear
                if (grafica.getId() == null) {
                        // Usuario y fecha de creacion.
                        grafica.setUsuarioCreacion(getUsuarioAutenticado());
                        grafica.setFechaCreacion(new Date());

                        result = gestorCU.guardarGrafica(grafica, variablesx, variablesy);
                } else {
                        result = gestorCU.actualizarGrafica(grafica, variablesx, variablesy);
                }
                return result;
        }
        . . .
}
```

5.3.2 Presenter

Cada presenter tiene enlaces a un modelo y una vista. Continuando con el ejemplo de las secciones anteriores, a continuación se muestra el método *guardarGrafica()* de la clase `CreacionGraficasPresenter`:

```
@Component
@Scope(value = BeanDefinition.SCOPE_PROTOTYPE)
public class CreacionGraficasPresenter extends
                AbstractPresenter<CreacionGraficasVista, CreacionGraficasModelo> {

        public boolean guardarGrafica() {
                Grafica grafica = getVista().getGrafica();
                List<VariableX> variablesx = null;
                List<VariableY> variablesy = null;
                boolean ok = false;

                // Comprobamos que no sea nula
                if (grafica == null) {
                        Notification.show(messages
                        .getMessage(ConstantesStrings.CRTCQ_NT_EXCCAMPOSNOCMPLT_TITLE),
                        messages.getMessage(ConstantesStrings.CRTCQ_NT_EXCCAMPOSNOCMPLT_DET),
                                Notification.Type.WARNING_MESSAGE);
                } else {
                        try {
                                variablesx = getVista().getVariablesX();
                                variablesy = getVista().getVariablesY();

                                Grafica graficaGuardar = getModelo().guardarGrafica(
                                                grafica, variablesx, variablesy);

                                if (graficaGuardar != null) {
                                        // Mensaje de confirmación de guardado
                                        this.grafica = graficaGuardar;
                                        Notification.show(messages
                                .getMessage(ConstantesStrings.CRTGRAFICA_NT_GRAFICACREADA_DET));
                                        ok = true;
                                }
                        } catch (VariableInvalidaException e) {

                                Notification.show(messages
                                .getMessage(ConstantesStrings.CRTCQ_NT_EXCCAMPOSNOCMPLT_TITLE),
                                e.getMessage(),Notification.Type.WARNING_MESSAGE);
                        }
                }
                return ok;
        }
}
```

5.3.3 Vista

En la vista es donde se utiliza Vaadin para generar la interfaz de usuario. A partir de los componentes que proporciona Vaadin, se crearon todas las vistas necesarias.

En la Ilustración 5.2 se muestra el aspecto que tiene la vista de creación de gráficas, cuya clase de soporte (`CreacionGraficasVista`) tiene la siguiente cabecera:

```
@SpringView
@Scope(value = BeanDefinition.SCOPE_PROTOTYPE)
public class CreacionGraficasVista extends
                AbstractVista<CreacionGraficasPresenter> implements PanelColapsedListener {
```

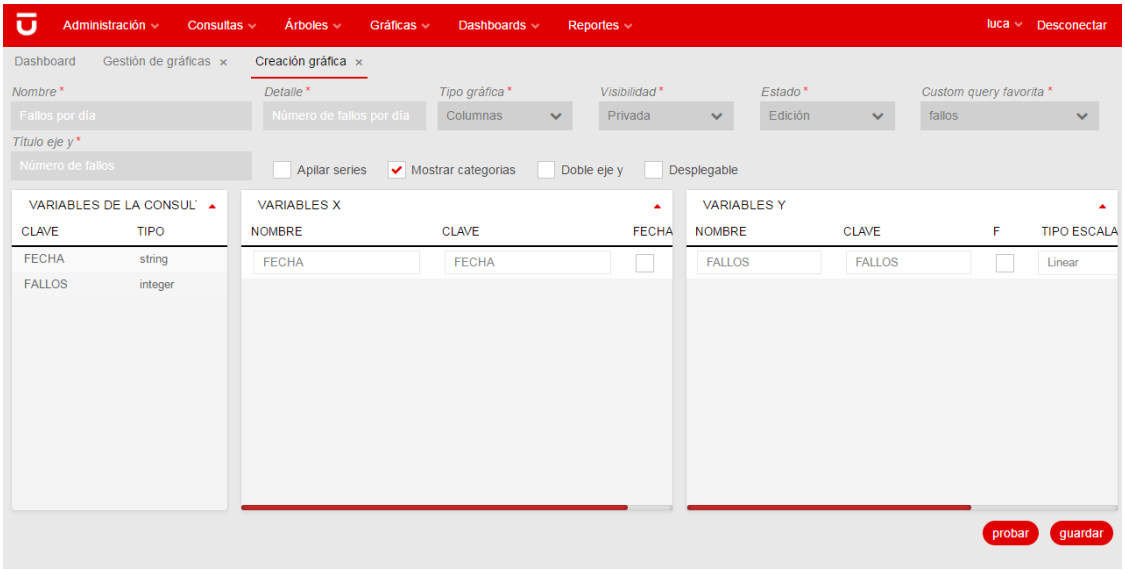

Ilustración 5.2 Vista de creación de gráficas

En este ejemplo se está creando una gráfica para mostrar el número de fallos por día en las ejecuciones de consultas durante el mes de enero. En la Ilustración 5.3 se muestra el aspecto de la gráfica una vez generada.

Ilustración 5.3 Gráfica "Fallos por día"

6 Pruebas

Las pruebas intentan mostrar que el programa realiza lo que se pretende correctamente, además de descubrir el mayor número posible de defectos antes de ponerlo en uso. Cuando se prueba una aplicación, se ejecuta con datos predefinidos y se comprueban los resultados en busca de errores o anomalías. El proceso de pruebas tiene dos objetivos distintos:

- Demostrar al desarrollador y al cliente que la aplicación cumple sus necesidades. Esto quiere decir que debería existir una prueba por cada requisito.

- Descubrir situaciones en las que el comportamiento de la aplicación no es el deseable.

6.1 Pruebas Unitarias

Las pruebas unitarias sirven para comprobar que cada módulo de código funciona correctamente por separado. Para la realización de estas pruebas se ha utilizado el *framework* JUnit [21].

A continuación se muestra un fragmento de la clase de prueba del componente `GraficaDAO`:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath:es/cic/luca/registros/test/context-test.xml" })
public class GraficaDAOTest {

	@Autowired
	private GraficaDAO graficaDAO;

	private Grafica graficaTest = new Grafica();

	@BeforeClass
	public void saveGrafica() {
		graficaTest.setTitulo("Gráfica de prueba");
		graficaTest.setDetalle("Prueba");
		graficaTest.setEstado(EstadoGrafica.EDICION);
		graficaTest.setTipoGrafica(TipoGrafica.COLUMN);
		graficaTest.setVisibilidad(VisibilidadGrafica.PRIVADA);
		graficaTest.setApilado(false);
		graficaTest.setCategorias(true);
		graficaTest.setDobleEjey(false);
		graficaTest.setDesplegable(false);
		graficaTest.setTituloEjey("Datos");
		graficaTest.setCustomQueryFavorita(new CustomQueryFavorita());
		graficaTest.setGrupo(new Grupo());

	}




	@Test
	public void saveAndUpdateGrafica() {
		Grafica g = graficaDAO.create(graficaTest);
```

```
                assertNotNull(g);
                assertTrue(graficaDAO.get(g.getId()).titulo.equals("Gráfica de prueba");

                graficaTest.setTitulo("Gráfica actualizada");
                graficaTest.setTipoGrafica(TipoGrafica.BAR);
                graficaTest.setApilado(true);

                assertNotNull(graficaDAO.update(graficaTest));
                assertTrue(graficaDAO.get(g.getId()).titulo.equals("Gráfica actualizada");

        }
}
```

6.2 Pruebas de Integración

Las pruebas de integración se realizan después de las pruebas unitarias y sirven para verificar que todos los componentes del sistema funcionan correctamente de manera conjunta. Para automatizar estas pruebas se realizaron pruebas con Selenium [22]. Selenium permite transformar las pruebas en clases JUnit, de forma que al ejecutar los métodos de prueba, se ejecutan en el navegador. A continuación se muestra el caso de prueba de crear gráfica automatizado con Selenium:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = { LucaWebApplication.class })
@WebAppConfiguration
@SeleniumTest(driver = LucaWebDriver.class, baseUrl = "http://localhost:8080/luca-web/")
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class CrearGraficaTest {

        @Autowired
        private WebDriver driver;
        private StringBuffer verificationErrors = new StringBuffer();

        private String credenciales[] = { "luca", "1234" };
        private String nombre = "Prueba";
        private String detalle = "Gráfica de prueba";
        private String tituloEjey = "Valores";

        @Before
        public void setUp() throws Exception {
                driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        }

        @After
        public void tearDown() throws Exception {
                String verificationErrorString = verificationErrors.toString();
                if (!"".equals(verificationErrorString)) {
                        fail(verificationErrorString);
                }
        }

        @Test
        public void testCrearGrafica() throws InterruptedException {
                UtilTests.login(driver, credenciales);
                abreCreacionGraficas();
                rellenaDatos();
                guardar();

                // La ejecución ha sido correcta si se muestra la gráfica.
                assertTrue(driver.findElement(By.cssSelector("div.highcharts-container vaadin-
                chart")).isDisplayed());
        }



        private void abreCreacionGraficas() {
                driver.findElement(By.cssSelector(ConstantesIds.getMenuItem(
```

```
                        ConstantesIds.MENU_GRAFICAS))).click();
                driver.findElement(By.cssSelector(ConstantesIds.getMenuItem(
                        ConstantesIds.MENU_GRAFICAS_VIS))).click();
                driver.findElement(By.cssSelector("div.v-button.v-widget")).click();
        }

        private void rellenaDatos() {
                driver.findElement(By.cssSelector(
                        "input.v-textfield v-widget v-textfield-required v-required v-
                    has-width")).sendKeys(nombre);
                driver.findElement(By.cssSelector(
                        "#GRAFICA_DETALLE > input.v-textfield v-widget v-textfield-
                    required v-required v-has-width")).sendKeys(detalle);

                driver.findElement(By.cssSelector("div.v-filterselect-button")).click();
                driver.findElement(By.cssSelector("td.gwt-MenuItem")).click();

                driver.findElement(By.cssSelector(
                        "#GRAFICA_VISIBILIDAD > div.v-filterselect-button")).click();
                driver.findElement(By.cssSelector("td.gwt-MenuItem")).click();

                driver.findElement(By.cssSelector(
                        "#GRAFICA_ESTADO > div.v-filterselect-button")).click();
                driver.findElement(By.cssSelector("td.gwt-MenuItem")).click();

                driver.findElement(By.cssSelector(
                        "#GRAFICA_QUERY > div.v-filterselect-button")).click();
                driver.findElement(By.cssSelector("td.gwt-MenuItem")).click();

                driver.findElement(By.cssSelector(
                        "#GRAFICA_EJEY > input.v-textfield v-widget v-textfield-required
                    v-required v-has-width")).sendKeys(tituloEjey);

        }

        private void guardar() {
                driver.findElement(By.id(ConstantesIds.GRAFICA_GUARDAR)).click();
        }

        private void probar() {
                driver.findElement(By.id(ConstantesIds.GRAFICA_PROBAR)).click();
        }
}
```

6.3 Pruebas de rendimiento, seguridad y usabilidad

Las pruebas de rendimiento y usabilidad fueron realizadas por el jefe del proyecto. Si alguna zona de la aplicación tardaba mucho en cargar o era poco usable, se revisaba hasta que cumpliera los criterios del jefe del proyecto.

En cuanto a la seguridad, por un lado se realizaron pruebas basadas en Selenium, y por otro, los usuarios finales son los que reportan algún fallo como por ejemplo que no puedan realizar ciertas operaciones teniendo el rol adecuado.

6.4 Pruebas de aceptación

Las pruebas de aceptación son realizadas por los usuarios finales para comprobar que se cumplen los requisitos de la aplicación. Son las pruebas más importantes ya que verifican que el sistema funciona como se definió. Estas pruebas las realizó el responsable del proyecto y codirector de este trabajo y algunos de los usuarios finales de la aplicación.

7 Conclusiones y trabajos futuros

En este apartado se plantean las conclusiones obtenidas tras la realización del proyecto tanto a nivel técnico como personal. Además, se muestra una serie de tareas que se desarrollarán en un futuro para mejorar la aplicación.

7.1 Conclusiones

Tras los comentarios de los usuarios de la aplicación después de su uso, podemos afirmar que se ha cumplido el objetivo del proyecto, que era añadir a LUCA la funcionalidad necesaria para generar gráficas, de tal forma que los datos obtenidos de las *Custom Queries* sean más entendibles y manejables. Se han implementado todos los requisitos analizados en esta memoria y en el momento de presentar este trabajo, la aplicación esta lista tanto para su uso interno como para ser presentada a algún cliente.

A nivel personal, el desarrollo de este trabajo supone un antes y un después en mi conocimiento sobre desarrollo de software. Hasta ahora, nunca había estado involucrado en un proyecto de estas dimensiones. Además, he aprendido a utilizar tecnologías que ahora mismo son un referente en el desarrollo de software como Vaadin, Spring o Maven. Además, cabe destacar que los conocimientos adquiridos en la carrera, me han servido para diseñar este módulo y me han ayudado a asimilar todos los conceptos necesarios para su implementación.

7.2 Trabajo futuro

En una aplicación de estas características, siempre van surgiendo nuevas ideas y necesidades de añadir nuevas funcionalidades. A continuación se muestran funcionalidades que fueron surgiendo a medida que se desarrollaba el proyecto y a medida que lo probaban los usuarios.

- **Permitir mostrar datos calculados a partir de los datos devueltos por las consultas**: Existen ocasiones en los que se necesitan datos que no son devueltos directamente por la consulta a partir de la cual se genera la gráfica, pero que son fácilmente calculables a partir de los datos devueltos. Un ejemplo sería el cálculo de la ganancia en un mes cuando la consulta nos devuelve los gastos y los ingresos para ese mismo mes. Para facilitar la tarea al usuario, se le permitirá crear fórmulas al configurar la gráfica que utilicen las variables de salida de la *Custom Query*. Para interpretar estas fórmulas, se utilizará un motor JavaScript nativo en Java.

- **Permitir mostrar más de una variable "y" cuando se han configurado dos variables "x":** Si se quiere añadir más de una variable "y" cuando se han configurado dos variables "x" la gráfica no se visualiza correctamente. Esto se debe a que cuando se configuran dos variables "x" los valores se agrupan por la primera y estos a su vez por la segunda. Esto hace que se muestre una barra

por cada agrupado si la gráfica es de tipo barras, o una línea por cada agrupado si la gráfica es de tipo líneas, etc. Por lo tanto, para permitir mostrar más de una variable “y” en las gráficas configuradas con dos o más variables “x” una opción sería que los valores de cada variable “y” se mostraran en gráficas distintas dentro de la misma gráfica. Por ejemplo los datos de una variable “y” se mostrarían en una gráfica de barras y a su vez los datos de la segunda variable “y” se mostraría en una gráfica de líneas superpuesta a la gráfica de barras.

- **Permitir cambiar el estilo de las gráficas**: En LUCA se permite subir ficheros CSS para cambiar el estilo de la aplicación. Con las gráficas se podría hacer lo mismo para que por ejemplo, los colores de las mismas sigan el mismo patrón que los de LUCA.

8 Referencias

[1] ¿Qué es Business Intelligence? (Consultado el 14/06/2016).
http://www.sinnexus.com/business_intelligence/

[2] Grönroos, M. (2014). Book of Vaadin: Vaadin 7 edition – 6th revision. Vaadin Ltd.

[3] Página web de Vaadin Charts. (Consultado el 14/06/2016)
https://vaadin.com/charts

[4] Página web de Highcharts. (Consultado el 14/06/2016)
http://www.highcharts.com/

[5] Introduction to the Spring Framework (2016). (Consultado el 14/06/2016)
http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html#overview-modules/

[6] Spring Framework – Overview (2016). (Consultado el 14/06/2016)
http://www.tutorialspoint.com/spring/spring_overview.htm/

[7] Página web de MagicDraw. (Consultado el 14/06/2016)
http://www.nomagic.com/products/magicdraw.html

[8] Página web de MySQL. (Consultado el 14/06/2016)
https://www.mysql.com/

[9] Licencia GNU GPL. (Consultado el 14/06/2016)
http://www.gnu.org/licenses/gpl-3.0.en.html

[10] The Java EE Tutorial. (Consultado el 14/06/2016)
https://docs.oracle.com/javaee/7/tutorial/

[11] Página web de Hibernate. (Consultado el 14/06/2016)
http://hibernate.org/

[12] Página web de Maven. (Consultado el 14/06/2016)
https://maven.apache.org/

[13] Página web de Tomcat. (Consultado el 14/06/2016)
http://tomcat.apache.org/

[14] Página web de Eclipse. (Consultado el 14/06/2016)
https://eclipse.org/

[15] Sommerville, I. (2011). Software Engineering 9th edition. Pearson.

[16] Página web de Tableau. (Consultado el 14/06/2016)
http://www.tableau.com/

[17] Página web de Redash (Consultado el 14/06/2016)
http://redash.io/

[18] Aplicación Vaadin QuickTickets Dashboard. (Consultado el 14/06/2016)
https://demo.vaadin.com/dashboard/

[19] Página web de MySQL Workbench. (Consultado el 14/06/2016)
https://www.mysql.com/products/workbench/

[20] The Generic DAO Pattern in Java with Spring 3 and JPA 2.0 (2011). (Consultado el 14/06/2016)
http://www.codeproject.com/Articles/251166/The-Generic-DAO-pattern-in-Java-with-Spring-3-and

[21] Página web de JUnit. (Consultado el 14/06/2016)
http://junit.org/junit4/

[22] Página web de Selenium. (Consultado el 14/06/2016)
http://www.seleniumhq.org/