UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Servicio de exportación de datos desde bases de datos

Autor/es

DANIEL GÓMEZ OCHAGAVÍA

Director/es

ARTURO JAIME ELIZONDO

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2018-19

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Servicio de exportación de datos desde bases de datos

Realizado por:

Daniel Gómez Ochagavía

Tutelado por:

Arturo Jaime Elizondo

Logroño, Junio, 2019

Resumen

Durante la realización de las prácticas en la empresa Hiberus Osaba se utilizó una librería de exportación de datos. Ésta permitía conectarse a bases de datos, extraer información mediante consultas y exportar los resultados en diferentes formatos de archivo, como XML. La empresa planteó la necesidad de generalizar la librería para permitir su uso desde diferentes plataformas de programación como .Net o Java y también a través de una interfaz de usuario que da acceso a una aplicación web.

En este proyecto se han desarrollado ambas soluciones y se han implantado para uso interno de la empresa. Aunque el objetivo inicial se restringía a .Net se reorientó pasados unos meses al producto multiplataforma desarrollado finalmente.

La librería desarrollada ya ha sido utilizada en un proyecto real y se prevé su utilización en nuevos proyectos.

Abstract

During the internship at Hiberus Osaba, a data export library was used. This made it possible to connect to databases, extract information through queries and export the results in different file formats, such as XML. The company raised the need to generalize the library to allow its use from different programming platforms such as .Net or Java and also through a user interface that gives access to a web application.

In this project, both solutions have been developed and implemented for internal use within the company. Although the initial objective was restricted to .Net, it was reoriented after a few months to the multiplatform product finally developed.

The developed library has already been used in a real project and is expected to be used in new projects.

Índice

1. Introducción

En este apartado justificaremos la elección del proyecto, los motivos que llevaron a realizarlo, la situación por la que la empresa decidió poner en marcha este proyecto y los objetivos principales.

1.1. Contexto

Hoy en día, la mayor parte de la información se guarda en bases de datos (BD). Esta información queda a disponibilidad de todo aquel que sepa utilizar dichas BD. Los que no tienen los conocimientos suficientes para realizar estas consultas de la información dependen en gran medida de personal cualificado. Bien es cierto que existen procesos de creación de aplicaciones que hacen más fácil el acceso a estos datos, pero suelen ser costosas de realizar, y requieren de conocimientos básicos de programación y acceso a BD. Además de esto, en las aplicaciones que realizan las empresas informáticas, suele ser corriente la necesidad de realizar consultas y exportaciones de datos para más adelante tratarlas para hacer informes. Esto último suele darse en un contexto en el cual se necesiten automatizar procesos de consulta y exportación de datos.

Ante estas necesidades, surge una idea de realizar una aplicación que solvente estos problemas, ayudando a usuarios sin conocimientos de BD a obtener información que puedan necesitar de manera sencilla, rápida y cómoda; además de construir una solución global que pueda ser de utilidad en futuros proyectos de la empresa Hiberus Osaba que requieran este tipo de funcionalidad de consulta y exportación de datos.

1.2. Motivación

La propuesta de este proyecto, en comparación con alguna otra, era la más interesante y tentadora por varios motivos.

Primeramente, suponía una continuación de las tecnologías que había conocido y con las que había trabajado durante el periodo de prácticas, lo que me permitía desarrollar los nuevos conocimientos que había adquirido, así como seguir formándome en ello. De igual forma, suponía una aplicación real donde poder plasmar ese conocimiento y también poder aprender otros nuevos.

Además, como esta necesidad ya había empezado a ser notable antes de que se empezaran las prácticas, la empresa ya había desarrollado código que empezaba a solucionar el problema. Este código necesitaba ser ampliado, lo cual requería de un estudio de trazas de código, así como reimplementación de funcionalidades existentes. Este tipo de conocimiento es muy importante en el mundo laboral y da valor añadido al proyecto.

Las capacidades de adaptación ante el cambio y el conocimiento de nuevas tecnologías son también muy importantes y valoradas. Existía la posibilidad de trabajar con tecnologías desconocidas para nosotros, ya que las formas de implementación de una solución concreta al problema son muy variadas. Si bien el objetivo y la forma de implementación parecía concreta en un principio, en el desarrollo del proyecto podía cambiar, lo cual haría que, además, tuviéramos que adaptarnos y reajustar el proceso de creación de esta solución.

No contentos con esto, incluimos además una gestión del proyecto con una metodología ágil, llamada Scrum, muy utilizada en el mundo laboral, la cual se adaptaba muy bien a las necesidades del proyecto por sus amplias posibilidades de cambio estructural y lógico.

En esta memoria, comenzaremos plasmando la planificación inicial del proyecto, compuesta por el diagrama de hitos, cronograma y horas de dedicación. Tras esto pasaremos a describir la estructura del producto final, y explicando posteriormente la idea inicial del proyecto. Lo haremos así para que se pueda apreciar el cambio de rumbo que se tuvo en el proceso de implementación de una solución viable y adecuado a las necesidades cambiantes de la empresa.

Después se explicará el diseño de la estructura de clases que han regido la forma del proyecto, para pasar al proceso de implementación. El apartado de implementación contendrá la especificación de las tecnologías utilizadas y conceptos base utilizados en la memoria; implementación del proceso seguido, en el cual relato los principales problemas y las soluciones obtenidas; y el proceso seguido desde una perspectiva de gestión de metodología ágil.

Más adelante, hablaremos de la implantación del producto desarrollado en la empresa, para terminar con las conclusiones finales. Al final existe un anexo con información referenciada a lo largo de la memoria.

2. Antecedentes

La realización del presente proyecto vino precedida de una estancia a tiempo parcial en la empresa Hiberus Osaba de Logroño durante el primer cuatrimestre. Esta empresa se dedica al desarrollo de software. Ese periodo sirvió como toma de contacto con la mayoría de tecnologías y métodos a utilizar en el proyecto que se describe en esta memoria. Destacamos las siguientes:

- Aplicación del patrón MVC con lenguaje Java.
- Framework de código abierto Spring para Java.
- Librería Java Thymelaf que ofrece un motor de plantillas XML/XHTML/HTML5 para conectar la vista con las clases controlador.
- Herramienta Hibernate de transformación objeto-relacional ORM para Java.

Este proyecto fue propuesto por la empresa partiendo de un desarrollo previo realizado por su equipo y que se venía utilizando en proyectos reales. Se trata de una **librería para la extracción de datos de bases de datos** en formato XML, Excel o CSV. La librería ofrece una interfaz de programación de aplicaciones (API) cuyos métodos permiten la conexión con una base de datos (BD) relacional y la extracción de datos de dicha BD en formato XML, Excel o CSV.

El uso típico de esta librería constaba de los siguientes pasos:

- Conexión con la BD aportando la información necesaria.
- Especificar los datos de la BD a transformar. Se ofrecen dos opciones. La primera es facilitar una consulta SQL y la segunda es instanciar algunos objetos con valores apropiados.
- Indicar qué formato tendrán los resultados obtenidos (XML, Excel o CSV).

- También hay algunos parámetros adicionales para configurar el resultado.
- Por último se invoca el método de generación del resultado.

Para más detalle, ver el anexo, apartado Antecedentes de la librería de exportación.

El periodo de utilización de esta API sirvió para identificar algunas mejoras interesantes y nueva funcionalidad, además de hacerla accesible desde plataformas como .NET.

Las mejoras identificadas son las siguientes:

- Limpiar y reestructurar el código en funciones más pequeñas (refactoring).
- Lectura y exportación del archivo de datos teniendo en cuenta su codificación de caracteres (como UTF-8, ANSI, etcétera).

Las funcionalidades a ampliar son:

- Accesibilidad a cualquier BD.
- Extensiones adicionales para los archivos exportados.

Las plataformas desde las que se pretende que sea accesible son las siguientes:

- Java
- .Net
- Cualquier plataforma que permita realizar peticiones REST.

Además de lo anterior, se propuso utilizar el método ágil Scrum para la dirección de este trabajo de fin de grado (TFG), que es un proyecto unipersonal.

3. Alcance y objetivos

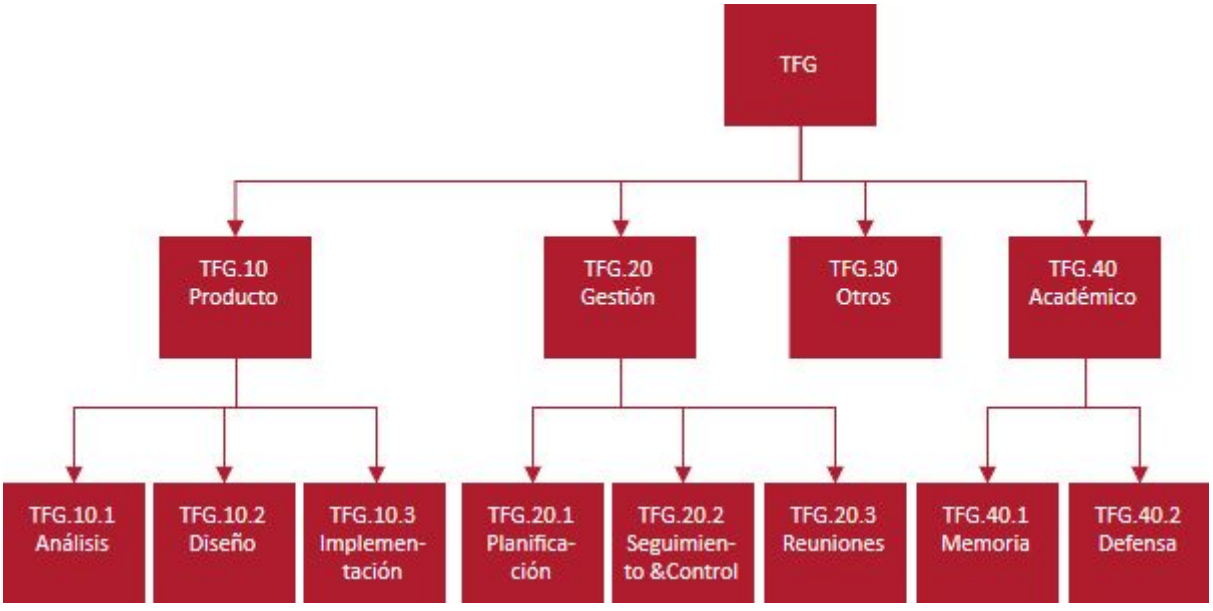

Figura 1. EDT del proyecto.

En un principio íbamos a hacer una aplicación monolítica en un único lenguaje, el cual pensábamos desplegar en un servidor que atendiera peticiones de clientes, consiguiendo deslocalizar el servicio de exportación de datos. Esta aplicación iba a contener conexión segura y remota a las bases de datos de los clientes, y sería capaz de realizar envios seguros de los ficheros con los datos exportados. Aunque este proyecto era viable en las horas

pactadas para este, al final se acabó realizando una API para manejar BD y extraer información en diferentes formatos. Esta API es una aplicación multiplataforma con grandes posibilidades de ampliación y adaptación. Queremos que esta aplicación sea capaz de comunicarse con aplicaciones de diferentes lenguajes, exportar datos de diversas BD, y que pueda ser utilizado a nivel de red privada para utilizarlo en futuros productos de la empresa.

4. Gestión

4.1. Alcance (requisitos)

En la figura 1 se muestra la descomposición del TFG. El apartado de Otros TFG.30 contiene la integración del producto TFG.10 en un proyecto empresarial real.

4.2. Cronograma

En la figura 2 se incluyen los hitos referentes al EDT de la figura 1. En la figura 3 se plasma que actividades se podrán realizar en cada semana.

Hito	Febrero				Marzo				Abril				Mayo					Junio				Julio	
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
Inicio TFG	◆																						
JAR				◆																			
API REST							◆																
WEB de config.										◆													
API de consultas										◆													
Integración del módulo																	◆						
Memoria revisar																			◆				
Depósito																					◆		
Defensa TFG																							◆

Figura 2. Diagrama de hitos principales del proyecto.

Periodo ejecución paquete TFG	Febrero				Marzo				Abril				Mayo					Junio				Julio	
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
1.1. Análisis																							
1.2. Diseño																							
1.3. Implementación																							
2.1. Planificación																							
2.2. Segu&control																							
2.3. Reuniones																							
3.1. Memoria																							
3.2. Defensa																							

Figura 3. Cronograma del proyecto.

4.3. Recursos (dedicaciones)

Dedicación en horas paquete TFG	Febrero				Marzo				Abril				Mayo					Junio				Julio	
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
1.1. Análisis	1	1	1	1	1	1	2	1	1														
1.2. Diseño	2	2	2	2	2	2	3	2	1														
1.3. Implementación	20	19	20	19	20	20	17	20	11														
2.1. Planificación	2	2	2	2	2	2	2																
2.2. Segu&control		1		1			1																
2.3. Reuniones	1	1	1	1	1	1	1																
3.1. Memoria	2	2	2	2	2	2	2	5	15		5			5		5	5	10	10				
3.2. Defensa																							
Total.	28	28	28	28	28	28	28	28	28					5		5	5	10	10				

Figura 4. Horas de dedicación a cada paquete por semana redondeadas a horas enteras.

5. Análisis y diseño

El análisis del problema y parte de su diseño fue facilitado por la propia empresa. Como ya se ha dicho, partíamos de una solución en funcionamiento y a partir de la experiencia con ella se habían detectado las mejoras y ampliaciones que nos disponemos a explicar. Quizá lo más complejo de explicar es la arquitectura de la solución compuesta de varias capas sobre la solución de la API original. La causa de este anidamiento de capas es la necesidad de utilizar la API desde otras plataformas diferentes a Java como .NET. La creación de una API multiplataforma se ha conseguido mediante el desarrollo de una API intermedia que ofrece su funcionalidad mediante el protocolo HTTP.

Figura 5. Arquitectura del producto realizado (objeto azul) y representación de la lógica de negocio.

5.1. Arquitectura del producto realizado

En la figura 5 se representa el resultado de este proyecto. A continuación describimos las diferentes partes de esta solución en las que se han utilizado varias tecnologías:

1. **API JAR.** Ofrece métodos para la conexión y consulta a BD y la transformación de datos a formato XML, Excel o CSV. El proyecto partía de una versión anterior del JAR.
2. **API REST**. Está implementada con Spring Framework Web para Java. Permite que los métodos de la API JAR puedan invocarse desde HTTP. Los métodos de la API REST utilizan JSON para la transferencia de datos. Los métodos de esta última API retransmiten la información entre objetos JSON y parámetros de la API JAR. Los resultados producidos por la API JAR en XML, Excel o CSV se devuelven codificados en base64, que es un formato de codificación textual que utiliza únicamente caractéres imprimibles del código ASCII.

   Como ya se ha dicho, la API REST se invoca mediante HTTP y ello permite que su funcionalidad pueda ser utilizada desde otros entornos de programación, como Microsoft .NET representado en la figura 5.
3. **API de consultas**. Es una nueva capa de acceso a la funcionalidad de transformación de datos codificada en lenguaje C# de .NET. Los métodos de esta nueva API invocan a los métodos de la API REST, reciben el número de la consulta a ejecutar, la buscan y recuperan los ficheros XML con la consulta, datos para incluir en la consulta y retransmiten dichos XML a los métodos correspondientes de la API REST.
4. **Ficheros XML de consultas.** Conjunto de ficheros que contienen información necesaria para realizar consultas. Cada fichero contiene:
   - URL del servicio REST.
   - Datos para la conexión a la BD.
   - Especificación de los datos a ser exportados.
   - Formato de exportación.

   Esta información se estructura en formato XML, lo que agiliza el proceso de lectura de la API de consultas. Estos ficheros se pueden crear manualmente o de manera gráfica mediante la interfaz WEB desarrollada en el proyecto.
5. **Aplicación WEB de configuración de consultas**. ¿Es una aplicación WEB? Es un módulo autónomo (standalone) desarrollado en .NET y estructurado según el patrón MVC. Incluye una interfaz gráfica WEB que permite configurar visualmente tanto consultas como conexiones a BD, que se almacenan en los ficheros XML explicados anteriormente.
6. **Cliente de consola.** Es una interfaz textual, similar a las consolas de línea de instrucciones de algunos sistemas operativos. Se ha codificado en lenguaje C# de .NET. Permite invocar a los métodos de la API de consultas y tiene acceso a las conexiones y consultas a BD almacenadas en los ficheros XML presentados anteriormente. Recordemos que las transformaciones de datos que devuelve la API de consultas vienen codificadas en base64. Este cliente también permite descodificar los resultados a XML, Excel o CSV en el disco duro del usuario. Este cliente permite hacer pruebas con las API desarrolladas y su implementación sirve como modelo para la realización de otros clientes de la API de consultas.

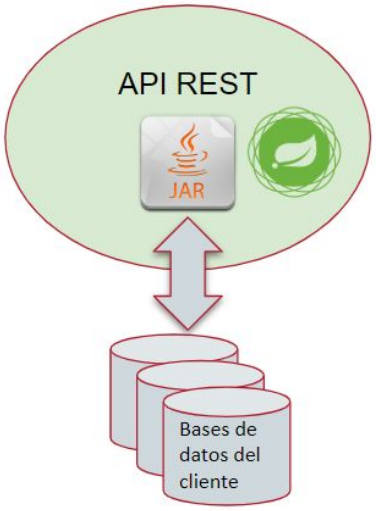

Figura 6. Conexión entre API JAR y BD. Esta API está integrada en una API REST realizada con Spring.

5.2. Versión anterior de la arquitectura

Inicialmente se planteó la realización de una aplicación Java en tres capas como la de la figura 7. La aplicación gestionaría usuarios, quienes configurarían y almacenarían sus conexiones y consultas a BD. La aplicación permitiría realizar consultas en BD y guardar su resultado en los mismos formatos de nuestra solución (Excel, XML...).

Una vez iniciado el proyecto surgieron nuevas necesidades y la empresa decidió cambiar los objetivos del mismo. Se planteó entonces la reescritura de la API en .NET, debido a la necesidad de disponer de vínculos dinámicos o DLL, pero el producto a obtener tenía menos interés como proyecto para un TFG. Finalmente se decidió construir una API REST, que integrase la API JAR existente y que se pudiera utilizar desde cualquier plataforma mediante HTTP. Esta reorientación del problema era asumible, considerando las limitaciones de un TFG. Las dimensiones y complejidad de este trabajo eran similares a las del problema inicial.

Hay que tener en cuenta que un inconveniente de esta solución es que cualquier aplicación que necesite utilizar la API REST, por ejemplo una aplicación .NET, deberá disponer de un servidor Tomcat. Esto implica la tarea adicional de desplegar y mantener el Tomcat, algo innecesario en soluciones monoplataforma, como se preveía en el planteamiento original del proyecto. A pesar de este inconveniente es interesante disponer de una API REST para su utilización en proyectos internos de la empresa, más manejables y permisivos.

Tras implementar la API REST, y para almacenar las consultas y conexiones a BD utilizadas por la API de consultas se decidió utilizar archivos XML abandonando otras alternativas como SQLite. Esta decisión se tomó basándose en los siguientes argumentos:

- *Tiempo disponible*. La lectura de XML estaba implementada. Utilizar SQLite implicaba diseñar y crear la BD e implementar la lectura de los datos desde la API de consultas.
- *Las consultas* las pueden escribir usuarios con conocimiento y experiencia con BD y SQL muy variados. El XML facilitará la comprensión de qué hay que hacer a los usuarios con menos conocimientos.
- *Estudio de alternativas,* con las ventajas e inconvenientes de cada una(Consultar en el anexo, apartado 'Toma de decisión. Fichero XML o SQLite').

También se decidió disponer la información en dos ficheros, uno almacena las conexiones a BD y el otro las consultas, con referencias a las conexiones de BD definidas en el primero.

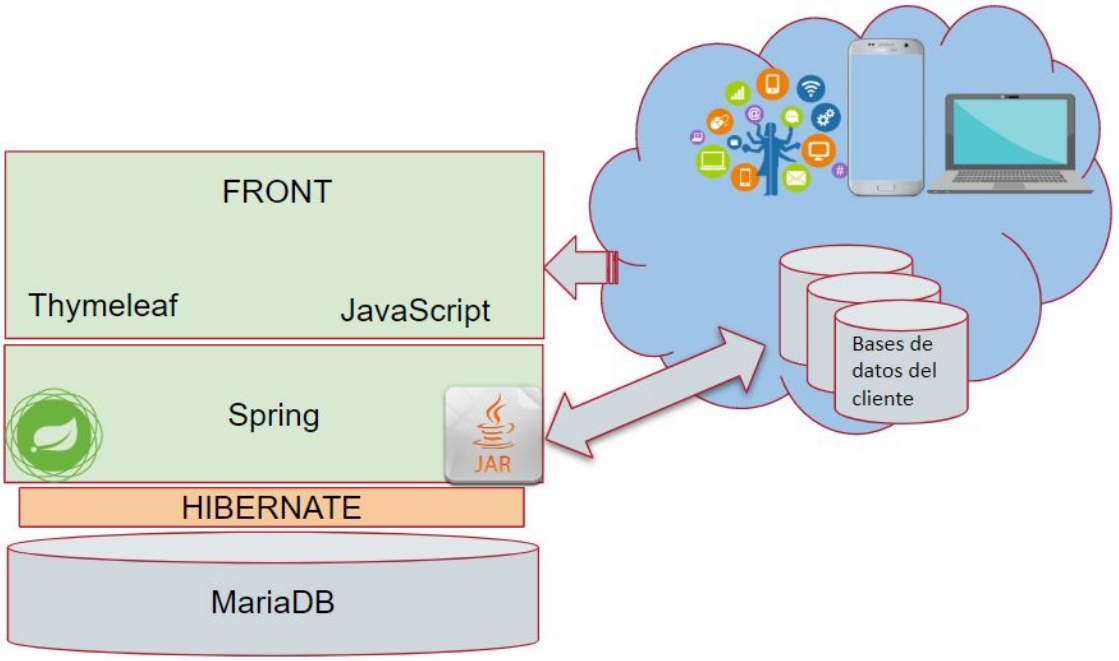

Figura 7. Lógica de negocio del producto al inicio del proyecto

5.3. Diseño

El diseño de los ficheros XML de consultas tienen la estructura que deriva de la librería inicial. Esta librería tiene varias clases, como podemos ver en la figura 8, siendo la principal ExportaEntity. Esta clase es equivalente a una consulta. Esta consulta puede dar como resultado un archivo Excel, para el cual se usa la clase ExportaExcelEntity, o un archivo codificado(XML o CSV), para el cual se utiliza la clase ExportaConCodificacionEntity. Estas dos últimas clases heredan de la clase ExportaEntity.

Además, los datos de conexión a bases de datos se guardan en la clase **ConexiónBDEntity**. Por último, cabe destacar la clase **OpcionesExportacion**, que contiene los parámetros necesarios para realizar las exportaciones, como por ejemplo la codificación de caracteres del fichero a exportar.

El resto de clases guardan los datos de las consultas.

6. Implementación

Este apartado explica el proceso que se ha seguido para desarrollar el proyecto. Al estar usando Scrum como metodología, la implementación se dividió en Sprints de entre 1 y 2 semanas de duración. Al terminar cada uno de estos Sprints se obtendría código potencialmente entregable a la empresa. Además, en el inicio de cada Sprint se planificaría que se va a hacer en este, dependiendo de las tareas del backlog y su prioridad.

A continuación se detallan las tecnologías que han intervenido en la realización del proyecto.

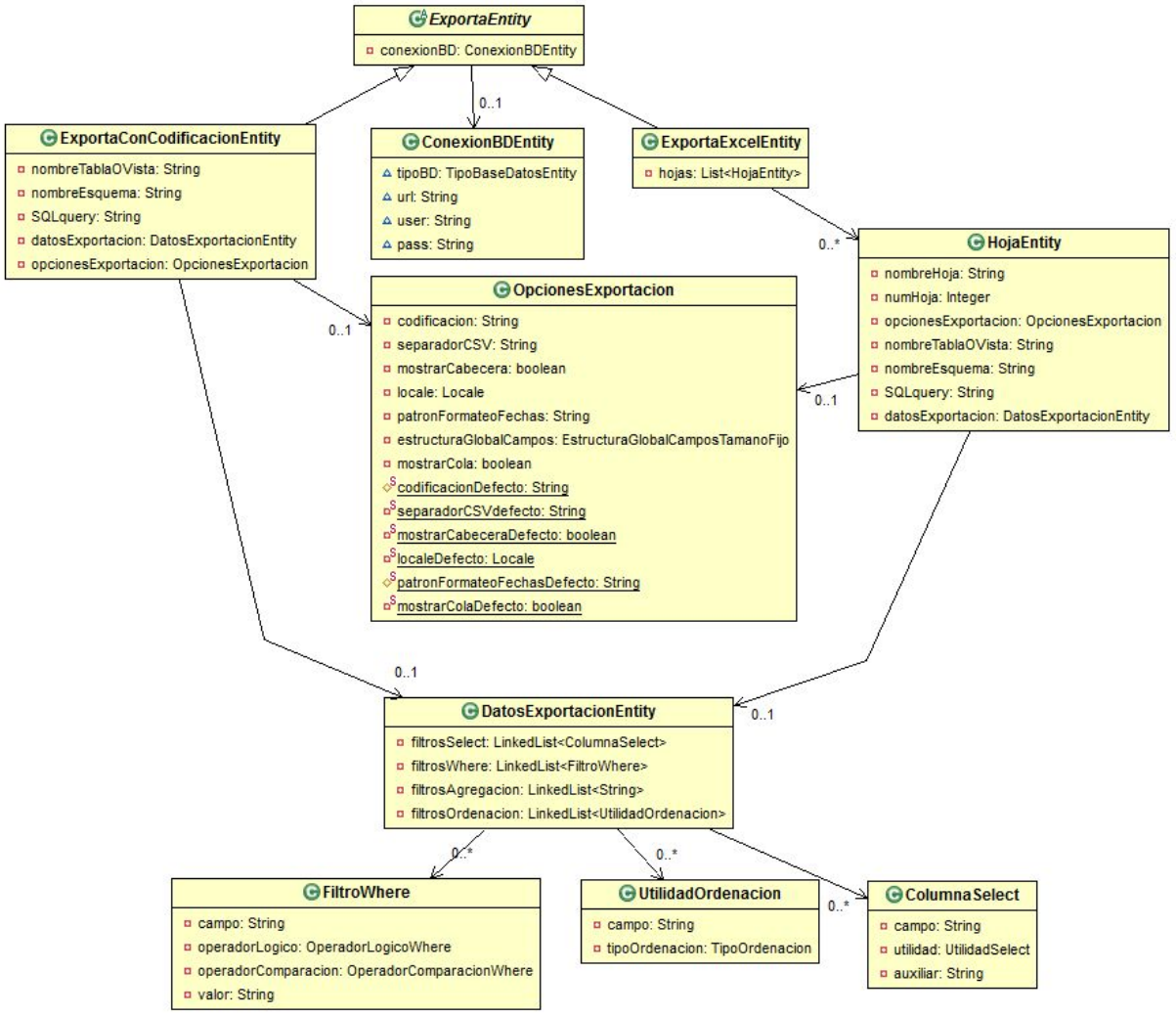

Figura 8. Estructura de clases que se mapean en las peticiones JSON.

6.1. Tecnologías

6.1.1. Gestión de versiones y de sprints

GIT y GIT Flow. Utilizado para realizar las subidas de cambios a los repositorios de manera rápida y cómoda. Utilizado a nivel de consola de comandos.

Fork. Cliente GIT.

Hansoft. Es una herramienta de gestor de proyectos ágiles gratuito para uno o dos usuarios.

En esta herramienta tenemos todo lo necesario para:

- Crear sprints para el proyecto.
- Crear un backlog con las tareas y subtareas del proyecto.
- Apuntar los puntos de historia que asignemos a cada tarea.
- Apuntar las horas que invirtamos en cada tarea.
- Apuntar el día de inicio y fin, la prioridad en el spring y en el backlog.
- Crear tareas para el sprint o usar las del backlog para ese sprint.
- Crear tareas programadas con sus dependencias y puntos de lanzamiento. Con esto podremos crear un diagrama gantt el cual podremos ir actualizando con el tiempo con el porcentaje de la tarea que se ha realizado.

- Manejar varios gráficos que simulan un tableros ágiles, para los sprints y el backlog.
- Crear gráficos de varios tipos que puedan ser orientativos con los datos del proyecto.
- Guardar el estado del proyecto en cada sprint para poder analizarlo más adelante.

6.1.2. Tecnologías de programación

Eclipse. Entorno de desarrollo que se utilizará para programar en java.

Hibernate. Herramienta de mapeo objeto-relacional (ORM).

Spring Framework. Ofrece soporte de infraestructura a nivel de aplicación. Como servicios de importancia para este proyecto

- Permite separar el código como MVC
- Realizar pruebas de testeo
- Framework web
- Conexión a BD con Hibernate.

Spring Boot. Tecnología desarrollada con Spring Framework que permite crear con Java un proyecto Maven, descargar las dependencias necesarias para poder implementar un proyecto web y desplegarlo en un servidor.

Se utilizó Spring Boot como tecnología que permitió crear con Java un proyecto Maven, descargar las dependencias necesarias para poder implementar un proyecto web y desplegarlo en un servidor. Spring Boot está desarrollado con Spring. Spring nos permite crear un sitio web con un modelo MVC, además de tener clases que permiten la implementación de API REST rápidamente.

.NET. IDE para C#.

Visual Studio.

Sublime. Editor de textos con gran capacidad de edición, selección múltiple, atajos de teclado. Utilizado para la edición y reutilización de código.

Notepad++. Editor de textos simple con una gran capacidad de trabajo en lo referente a codificación de caracteres. Utilizado para la comprobación de la codificación de caracteres de los archivos exportados en el proyecto.

6.1.3. Otras tecnologías y conceptos

Advanced REST client(ARC). Herramienta de google utilizada para realizar peticiones REST, con los diversos métodos GET, POST, ... Además, da la posibilidad de enviar peticiones JSON con una interfaz gráfica bastante intuitiva. Interpreta las respuestas y las presenta de manera legible, aunque también existe la posibilidad de ver las respuestas textualmente.

BitBucket: Utilizado para crear un repositorio privado. Esto evita conflictos de privacidad con la información que se use de la empresa. Este repositorio contendrá las actualizaciones que hagamos del proyecto, pudiendo hacer commits en local para más tarde hacerlos al repositorio.

Sistemas de gestión de bases de datos(SGBD). HeidiSQL, MariaDB, MySQL, Oracle, PostgreSQL, Microsoft SQL Server, SQLitle.

DBeaver. Herramienta universal de bases de datos como tecnología recomendada.

Modelo Vista Controlador (MVC). es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Base64. Sistema de numeración posicional que usa 64 como base. Sirve para codificar información.

CRUD. Acrónimo de Crear, Leer, Actualizar y Borrar. Son las acciones que cualquier aplicación de gestión de información contiene para modificar los datos de la base de datos.

6.2. Implementación del resultado obtenido

A continuación pasaré a relatar los problemas surgidos en el proceso de implementación de cada una de las partes del proyecto descritas en el apartado 5.1.

6.2.1. API JAR.

Librería de exportación. Como ya se ha explicado, la librería que forma el API del JAR estaba creada ya, pero se necesitaba funcionalidad extra para completarla. Al ser código de otra persona esto resultó ser una tarea de comprensión del código, con poca implementación extra. Surgieron problemas como implementar código que posteriormente se descubrió que estaba hecho ya. Además se descubrió casi al final del proyecto que se poseía el código de una versión desactualizada. Al comprobar las versiones más recientes se comprobó que mucha funcionalidad implementada estaba hecha en esta versión posterior.

Codificación de caracteres. Una de las partes más importantes de la librería de exportación era ser capaz de leer BD y exportar los datos en cualquier codificación de caracteres. Estas tareas fueron hechas mediante pruebas, pues se pensó que iba a ser un proceso más rápido que leer todo el código. Tras varias pruebas, se entendió que las librerías JDBC utilizadas se encargaban del problema de las bases de datos codificadas. En lo referente a la exportación:

- Para **CSV** hubo que hacer la conversión a bytes con la codificación que se requiera. Solución aquí. Esta conversión se introdujo en el método que devolvía el array de bytes para posteriormente ser escrito como un fichero.
- Para **XML**, tras mirar el código, resultó que usa una clase Transformer detecta la codificación que el usuario del código de exportación le proporciona (a través de la clase OpcionesExprotacion) y codifica los caracteres, y no solo pone la etiqueta encoding en la etiqueta xml raíz como se pensaba en un principio.
- Para **Excel**. La librería de POI, que es la que realiza la conversión de datos de un objeto java a un array de bytes con la codificación excel, trata todos los excel con codificación UTF-8.

Problemas con los drivers de JDBC. Al implementar 5 drivers distintos para que se pudiera conectar a MySQL, Oracle, PostgreSQL, MariaDB y Microsoft SQL Server, tuvimos varios problemas. La mayor parte de los problemas tenían relación con el acceso de la aplicación a las bases de datos. Se gastó bastante tiempo averiguando qué datos eran los correctos (nombre de esquema, nombre de la base de datos, usuario, contraseña). Un ejemplo claro de estos problemas fue la identificación de un usuario mediante las credenciales de windows. Estos datos no se podían obtener de manera fácil, y cuando se averiguó cuál era el problema, se cambió a credenciales de usuario y contraseña, para poder usarlos en la librería.

Otros problemas derivados de los permisos de los usuarios de las diferentes bases de datos también fueron importantes.

Problemas con el JAR. Crear un JAR operativo con el API no fue trivial. Siguiendo esta guía se creó, pero al utilizarlo en un proyecto de prueba, la conexión a determinadas BD daba problemas. Tras mirar en varias páginas que hablaban de nuestro problema, llegamos a la conclusión de que el driver JDBC que se estaba usando para acceder contenía un bug solucionado en una versión posterior. Este bug solo aparecía al usar el API a través del JAR.

Surgió otro error al usar el JAR resultó ser un bug del driver ojdbc8 que teníamos. Usando la versión ojdbc7 funciona correctamente. La solución al problema, tras mucho buscar en el código, la encontré aquí.

6.2.2. API REST.

Creación del proyecto. Para realizar este API utilizamos Spring boot con una estructura MVC e Hibernate, además de utilizar la librería del API JAR de exportación. Se recuerda que el objetivo era obtener un .war para poder desplegarlo en un servidor TomCat.

Ya se tenían proyectos con estas mismas tecnologías, así que se optó por usar un proyecto anterior y eliminar toda funcionalidad extra. Tras varias pruebas, se observó que no era tan fácil utilizar un proyecto ya operativo, pues había demasiados aspectos a modificar.

Se comenzó a realizar el proyecto desde cero. Accediendo a la página de Spring Boot, se obtuvo un proyecto simple con las características necesarias. Tras compilarlo y ejecutarlo, hubo problemas con el puerto que utilizaba, además, por si solo Spring no era capaz de crear las tablas necesarias para la sesión de los usuarios. Tras cambiar el número de puerto en el archivo de propiedades de la aplicación y crear a mano las tablas que necesitaba la aplicación en la BD, el programa ejecutaba.

Hubo varios problemas con la configuración del proyecto y con la recepción de las peticiones con las clases controladores, así que se decidió buscar un proyecto por internet que tuviera controladores ya definidos. Tras probar con un par de ellos, el segundo responde los datos de la BD gestionada con MariaDB. La conexión a BD ya la tenia del primer proyecto, así que no costó demasiado tiempo implementarla.

En la realización de la obtención de datos mediante peticiones REST se tuvo un problema costoso de resolver. Tras hacer la estructura de recogida de datos con una clase llamada JpaRepository, la aplicación indicaba que no encontraba las clases que estaba usando en el controller. Se probaron todo tipo de soluciones, desde escribir atributos que indican la ubicación de los ficheros que estaba utilizando, hasta atributos de spring que eran redundantes pero que parecía que solucionaban el problema. Al final era un problema de estructuración de paquetes. Esto era porque el atributo @SpringBootApplication que se especifica en el Application, clase que arranca el servicio, indica implícitamente que va a mapear las clases que estén en su paquete y en los paquetes hijos. Tras reestructurar los paquetes y las clases, ya está hecho el proyecto base para empezar el proyecto.

Peticiones JSON. Tras probar el envío de JSON, parece que no se pueden utilizar clases abstractas, pues hay que indicar que clase concreta es, Spring no sabe distinguir qué objeto es. Así que lo más cómodo será hacer una clase que contenga todos los atributos, y que el usuario deje en blanco los atributos que no se vayan a usar.

Tras simplificar el modelo, nos topamos con otro error. Las clases de la librería no tienen constructor válido para el conversor de JSON de Spring. Se explica aquí. Tras explorar el código, se deciden escribir los constructores necesarios.

Creación de .war. Al crear el war, se obtuvo un problema costoso de resolver (unas 10 horas). Al exportar el código del API REST, existían problemas de dependencia con la librería del API JAR. La explicación es bien sencilla una vez se entiende como funcionan las dependencias del pom.xml de los proyectos con Spring y Maven. El JAR estaba incluido en el proyecto a través de eclipse. Esto provocaba que al exportar el API REST a un .war, estas dependencias se perdían. Lo correcto, al estar utilizando las dependencias con maven, es instalar la librería del JAR con maven, como vemos en la figura 9.

Tras esto, añadimos la dependencia del JAR al pom.xml (figura 10), y maven interpreta que el jar está en la librería personal del usuario del ordenador. Posteriormente se crea el WAR y funciona correctamente.

Cabe destacar que en proceso de creación del .war, Spring se actualizó y se tuvo que rehacer el proyecto con el último repositorio guardado. Este problema surgió un par de veces, así que se tuvo que investigar cómo desactivar en eclipse las descargas automáticas de nuestras tecnologías, ya que provocan errores e incompatibilidades serios.

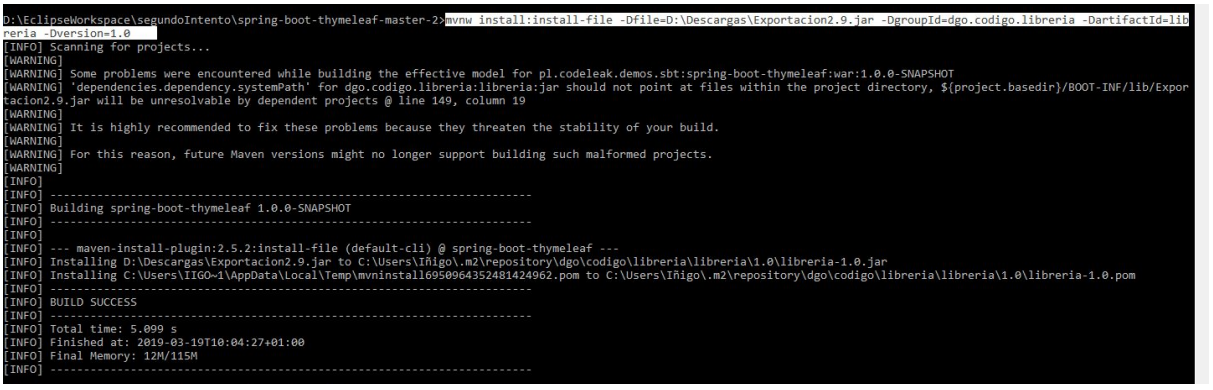

Figura 9. Instalación de la librería API JAR con maven.

```
<dependency>
    <groupId>dgo.codigo.libreria</groupId>
    <artifactId>libreria</artifactId>
    <version>1.0</version>
</dependency>
```

Figura 10. Dependencias del pom.xml del API JAR.

6.2.3. API de consultas y Ficheros XML de consultas

Esta parte, y la descrita en el siguiente punto(6.2.5) están realizadas mediante .Net y Visual Studio. Este entorno me dió problemas a la hora de instalarlo. Tras intentar instalar varias veces .Net, entendí que windows 10 por defecto ya tiene instalado este servicio y que lo único que había que hacer era habilitar el servicio en uno de los apartados de desinstalar programas, en el panel de control. Tras esto instalé Visual Studio. Recordamos que el alumno solo tenía conocimientos muy básicos de .Net y su funcionamiento, luego estas tareas tardaron más que si se hubieran realizado en Java.

El API de consultas fue sencilla de implementar, sin problemas derivados del código. Solo hubo un problema con la lectura de los ficheros XML de consultas y alguno derivado de las

pruebas del código en BD. Hay que tener en cuenta que el desarrollador de este proyecto solo poseía un conocimiento muy básico de .Net y su nomenclatura.

Tipos abstractos. Para almacenar las consultas se necesitaba un array de ExportaEntity. Los tipos que heredan de la clase abstracta ExportaEntity no podían ser serializados. Existía un problema con la clase XMLSerializer encargada de hacer la conversión de los objetos en .Net a XML y viceversa. Tras investigar algún ejemplo sencillo en la web se obtuvo la correcta especificación de los tipos abstractos, como podemos observar en la figura 11.

```
public class PeticionExportacionEntity
{
    public string MetodoHTTP;
    public string URL_Servicio_API_REST;

    //Uso la clase de la librería JSON porque es la misma y no va a cambiar.
    [XmlElement("ExportaConCodificacion", typeof(ExportaConCodificacionEntity))]
    [XmlElement("ExportaExcel", typeof(ExportaExcelEntity))]
    public ExportaEntity ExprotaEntity;
}
```

Figura 11. Especificación de tipo abstracto para el enlace con la conversión a XML

Así, en el archivo XML se guardan objetos con nombre ExportaConCodificacion o ExportaExcel y la clase encargada de serializar los objetos entiende que clase se está usando(clase que hereda de la clase abstracta ExportaEntity).

Pruebas con BD. Tras realizar unas pruebas automáticas que se implementaron para hacer consultas en varias bases de datos a la vez, se obtuvieron problemas con Oracle. Uno de los problemas es que Oracle es ‘’case sensitive’’, y no encontraba la base de datos. El otro es que Oracle necesita el nombre del propietario en vez del nombre del namespace de la base de datos.

6.2.4. Aplicación WEB de configuración de consultas.

Al realizar la web de modificación de consultas, se encontraron varios problemas.

El primero fue entender cuál es la filosofía de los controladores y las vistas en ASP.NET. Tras investigar sobre este tema, y hacer varias pruebas, se comprobó que los controladores y las vistas se comunican a través de un objeto, llamado modelo, el cual es pasado por los controllers a las vistas. Las vistas usan los datos y envían el modelo a los controladores.

Al solo poder usar un objeto (hay varias variables que se pueden usar, como ViewBag, aunque usarlas no es la filosofía), las consultas plantean un problema.

La configuración de consultas en la web se hará solamente con SQL. No se usarán datos de exportación. Las consultas tienen una o varias hojas, entendiendo por hojas los datos de una exportación. Al modificar una consulta, también se necesitaba modificar sus hojas, así que necesitábamos una forma de utilizar javascript con la vista y el controlador de consultas para crear, editar y borrar todas las hojas de una consulta en la misma vista.

Para ello se utilizará una tecnología que vale precisamente para hacer un crud dinámico. En este blog se explica como hacerlo. Se utilizó el blog para consultar, y directamente se descargó el proyecto de ejemplo para examinarlo y copiar la funcionalidad en el proyecto web de modificación de consultas y conexiones.

Tras implementar la funcionalidad, se vió que no se podía utilizar la tecnología ya que el proyecto a realizar utilizaba Core y el usado utilizaba Framework. Estas son dos APIs de .Net

que comparten varios métodos, pero no todos, lo que hacía incompatible la tecnología que se estaba intentando usar con el proyecto a desarrollar.

Antes de dar con la solución, se probaron varias técnicas para cambiar el API del proyecto, pero tras varios intentos se vió que no era la mejor solución. Se intentó también crear un proyecto .Net Framework con las clases que se habían desarrollado hasta ahora, y tras casi terminarlo, nos dimos cuenta de que existía el paquete NuGet para .Net Core.

6.3. Proceso seguido (Scrum)

En el anexo 'Historias de usuario y reuniones' se incluyen las reuniones con el cliente en cada uno de los sprints y las historias de usuario que derivaron de estas reuniones.

Para este proyecto se ha elegido realizar una gestión con Scrum. Está metodología está recomendada para equipos de entre 2 y 5 personas. Sin embargo existen varias ventajas que se han tenido en cuenta para organizar este proyecto con Scrum.

- El proyecto va a abordar uno o varios módulos que se añadirán a un código ya implementado. Lo importante no es desarrollar un producto al completo, sino desarrollar cierta funcionalidad que pueda ser utilizada por la empresa. Esto implica que las funcionalidades que se propongan desarrollar serán más de las que podremos abordar y habrá que priorizar. Estas prioridades pueden cambiar en el transcurso del TFG.
- Las funcionalidades que la empresa nos ha proporcionado son asequibles para una persona. Consideramos que cada dos semanas se podrá entregar alguna de las funcionalidades. Este periodo de tiempo es lo que llamaremos sprint.
- Scrum confía en la capacidad de autoorganización de los desarrolladores como nosotros confiamos en las capacidades del nuestro.
- Scrum es bastante usado hoy en dia. Aprender Scrum es una habilidad muy útil en el mundo laboral que servirá al estudiante para su futuro laboral.
- Pensamos que lo importante es crear módulos funcionales, no el tiempo. Esto quiere decir que no tendremos en cuenta cuánto tardaremos en realizar las tareas, sino la funcionalidad que desarrollemos en ese tiempo.

El Sprint 0 y 1 duraron una semana. El resto duraron dos semanas. Entendemos por una semana 5 días de 6,5 horas cada uno.

6.3.1. Sprint 0. Inicio del proyecto

El proyecto se empezó a desarrollar en un ordenador sin ningún tipo de programa instalado útil para el desarrollo de este. El primer dia se instalaron Eclipse,Hansoft, Git, Git Flow y HeidiSQL. Creamos un repositorio privado en Bitbucket

Tras esto, el estudiante se formó en el manejo de Git Flow, pues se carecía de conocimiento previo.

Código de exportación anterior al proyecto

Se empezó realizando la descripción de la funcionalidad del código de exportación prestado por la empresa antes de empezar el proyecto. Se crearon los roles de la aplicación y se

especificaron las definiciones básicas de Scrum(ver Anexo, apartado roles del proyecto y definiciones básicas de scrum).

Implementación del proyecto compilado e instalado 'Hello World'

El objetivo de este Sprint es diferente al resto de Sprints. El Sprint 0 debería ser el resultado de un compilable e instalable 'Hello World'.

Al empezar el Sprint se pensaba realizar un proyecto en Spring Boot utilizando Spring y Maven, una estructura MVC y Hibernate. Además, se utilizaría la funcionalidad del JAR de en el mismo proyecto.

Ya se tenían proyectos con estas mismas tecnologías, así que se optó por usar un proyecto anterior y eliminar toda funcionalidad extra. Tras varias pruebas, se observó que no era tan fácil utilizar un proyecto ya operativo, pues había demasiados aspectos a modificar, así que se comenzó a realizar el proyecto desde cero.

Se empezó a preparar la reunión con el cliente. Esta preparación se hizo con una cierta idea de cuáles eran sus necesidades. Se creó un esquema básico con las tecnologías que se podrían usar y las funcionalidades que creímos importantes. Con esto se esquematizó un backlog con los posibles requisitos que más tarde el cliente aclararía.

El siguiente paso fue establecer las prioridades y características necesarias para trabajar de manera eficiente, como la apertura de archivos con una estructura visual correcta, algún programa que estandarice el código de programación tras el guardado, eliminar el uso de caché del entorno en el cual vayamos a trabajar, que la aplicación no elimine los datos al inicializarse, etc.

6.3.2. Sprint 1.

Este sprint será el primero dedicado a crear código potencialmente entregable a la empresa. Éste consistirá en una reunión con el cliente, una planificación de las tareas a realizar tras la consecuente implementación de estas tareas y una revisión de la integración que haremos en cada una de los sprints.

Las primeras horas sirvieron para poner en marcha el código de exportación con sus correspondientes pruebas de acceso a la BD de prueba creada con MariaDB. Además se integró en eclipse un programa de reestructuración del código llamado Lombok para mayor limpieza del código del proyecto.

Se realizó la reunión con el cliente, especificando las características principales del proyecto, y de manera más genérica las características del API JAR. Estas últimas características generaron las tareas para realizar en el sprint. Se eligió realizar un Sprint de 1 semana para ir entrando en contacto con las estimaciones en puntos de historia, y su equivalencia en tiempo real de trabajo.

Antes de empezar a programar y estudiar el código de exportación para ver los cambios que teníamos que realizar, se inició con Hansoft un proyecto de gestión. Se incluyeron las historias de usuario y las tareas que derivan de estas en el backlog. Se decidió cuáles de ellas se iban a hacer en el sprint. También se pueden ver estas tareas en el kanban (para más información, ir al apartado del anexo 'Gestión Scrum con hansoft').

Se estimaron las historias de usuario tras haber fijado el tamaño de un punto de historia. El tamaño de un punto de historia tiene que ser bien conocido por todo el grupo de desarrollo (en este caso solo está formado por el alumno) para que este equipo pueda estimar bien el

resto de tareas del proyecto. En este caso, el tamaño de un punto de historia es equivalente a la complejidad de realizar una conexión a BD con nuestro código de exportación y obtener los datos concretos.

Se definió un breakpoint de entrega del jar para dentro de una semana. Esta entrega incluirá las funcionalidades de las historias de usuario del sprint 1. Esta entrega coincidirá con el final del sprint 1.

En mitad del sprint se realizó otra reunión con el cliente para verificar que las tareas de codificación de caracteres y de generalización de bases de datos seguían el rumbo adecuado.

- En primer lugar, se observó que los excel tenían codificación por defecto UTF-8, y que no se podía cambiar a otra codificación de guardado (desde Excel 2003 no se permite). El representante de la empresa da a entender que no es algo muy importante, así que cerramos el problema.
- Tras esto, se aclara que la codificación de guardado de caracteres de las extensiones CSV y XML son cruciales. Se tendrá que trabajar en esto como punto importante.Se tiene que poder escribir un archivo en la codificación que queramos, aunque ''no se entiendan los datos''. Esta recomendación hizo que el enfoque del problema cambiara, pues no se conocía el problema adecuadamente y se estaba trabajando en funcionalidad que no se necesitaba y obviando el verdadero problema.
- Tras esto se nos comunica la posibilidad de exportar a JSON y YML como posible tarea para el backlog.
- Al final, se habla del problema de la necesidad de un driver para exportar en una nueva BD. El representante de la empresa insiste en que mientras el programa sea modular y transparente para el usuario una vez hecha la conexión, no hay problema. Aún así, el cliente nos pide que hagamos unas 4 conexiones tipo para ofrecer en el .jar. Conexiones a Oracle, SQLServer, Postgre y MySQL.

Al finalizar el sprint hubo tareas que no pudimos realizar. Entre ellas, la conexión a PostgreSQL, SQLServer y Oracle, debido a que surgieron problemas al utilizar el JAR en un proyecto externo y que se añadieron a mitad del sprint. En el siguiente sprint se volvieron a tener estas tareas para realizarlas al completo. Naturalmente, la tarea de realizar un JAR tampoco estaba hecha, así que se también se incluyó en el backlog del sprint 2.

Si volviera a hacerlo, qué haría diferente. Que hay que cambiar(malas prácticas), que hay que hacer más.

En la retrospectiva señalamos que no se deberían desarrollar nuevas funcionalidades en el API JAR sin comprobar que las funcionalidades no están implementadas ya. También habrá que tener cuidado con utilizar drivers(JDBC, en nuestro caso) de versiones nuevas que podrían ser inestables.

6.3.3. Sprint 2.

En el Sprint anterior se completaron la mayoría de las funcionalidades del jar. En este sprint se realizará la API REST y se completará la funcionalidad de la API JAR y su correcto funcionamiento al exportarlo a un archivo jar.

En lo referente al API REST, el usuario se registra, con usuario y contraseña, y proporciona los datos de su conexión a base de datos. El servicio guarda los datos y manda al usuario una clave que tendrá que usar cuando necesite exportar datos de esa base de datos.

Para exportar datos, habrá dos rutas, de momento, que utilizarán objetos JSON para mandar los parametros al API. Para exportar excel, en el cual se puede especificar más de una hoja; y para exportar CSV y XML, en el cual se puede especificar la codificación de caracteres de la exportación del archivo. El resto de datos es el mismo en las tres extensiones que se van a realizar.

Se implementó la funcionalidad que permite exprotar de PostgreSQL y SQL Server.

La experiencia de usuario del JAR que se ha experimentado implementando la base de datos es bastante buena. Se pueden añadir conexiones a bases de datos relativamente rápido si se saben los datos de la base de datos correctamente. Lo que más tiempo ha costado ha sido la instalación de los gestores de bases de datos y entender cómo funcionan.

Tras implementar la estructura de datos JSON que se va a utilizar en el API REST, obtenemos los siguientes ejemplos de prueba(mirar anexo, 'apartado estructura JSON inicial').

Tras programar la funcionalidad de recepción de datos a través de peticiones JSON, nos damos cuenta de una funcionalidad que no existe. No se puede hacer una exportación Excel con varias hojas mezclando hojas con SQL y hojas con datosExportación. Esta funcionalidad se añadió al backlog, y como parecía prioritaria, se añadió también al del sprint.

Hecha la exportación con éxito mediante un envío de datos estructurados con JSON se procedió a la implementación de las extensiones XML y CSV.

El siguiente paso era crear el sistema de logeo. La aplicación requerirá de un registro para poder usar el servicio. Una vez logeado, la aplicación tendrá:

- El tipo de base de datos del cliente.
- Su url de conexión.
- Su usuario y contraseña.

Una vez el usuario se logue, podrá realizar las exportaciones.

Sin embargo, tras hablar con el representante de la empresa, la conexión de base de datos se incluirá con los datos incluidos en la misma petición JSON de exportación. Esto es debido a que se quiere que el API REST sea una simple traducción del API JAR, pero con peticiones REST. Simplemente se creó una clase conexión y un enumerados con los tipos de conexión a base de datos implementadas en el JAR.

El siguiente paso fue crear pruebas automáticas que probaran las exportaciones de datos de las diferentes BD.

Al final del sprint conseguimos crear la API REST, la cual es capaz de recibir peticiones JSON, interpretar sus datos y realizar la exportación de un archivo con la extensión determinada. También se consiguió exportar el .war con éxito, pero con errores derivados del funcionamiento del código que habrá que depurar en el siguiente sprint. Además falta que el servicio REST devuelva los ficheros en base64.

6.3.4. Sprint 3.

Una vez terminado el API REST, se plantea la posibilidad de realizar un cambio en uno de los requisitos fundamentales del proyecto. Este cambio es, a partir del API REST, realizar los servicios que utilicen este API en otro entorno de programación, en nuestro caso, pasar de JAVA a .NET.

La ventaja de hacer este proyecto que sabíamos cambiante es la siguiente. La estimación en puntos de historia de las tareas programadas es la misma estén en Java o .Net. Esto es debido a que la complejidad, recursos e incertidumbre no ha cambiado.

Con otros métodos de programación del tiempo y los recursos, esta estimación habría que hacerla de nuevo tras un cambio significativo como este.

Lo bueno de Scrum es que el único cambio en nuestra programación es que haremos más o menos puntos de historia, dependiendo de cuánto nos cueste realizar las tareas estimadas. Tras acabar el proyecto podremos estimar cuánto vale un punto de historia para nosotros mismos, y poder guiarnos en futuros proyectos.

Este cambio de rumbo hizo que tuviéramos que añadir una tarea de instalación de .Net.

En este sprint se realizaron las tareas pendientes del sprint anterior, corrigiendo los errores del .war y haciendo que el servicio devuelva los archivos resultado de la consulta en base64. Tras esto, se procedió a especificar de manera más concisa la funcionalidad del módulo standalone.

Lo siguiente fue un estudio de .NET y aprendizaje de creación de una aplicación web. Esto tardó tres días.

API de consultas y Aplicación Web.

Tras la formación de .Net, se procede a la creación de las clases que representan la información de las consultas. Estas clases se utilizarán para almacenar las consultas en los ficheros XML de consultas y realizar las peticiones JSON. Además, también servirán como clases para la aplicación web de consultas.

En un principio, el API de consultas hará lo siguiente. Tendrá almacenada en un fichero XML las consultas. A esta API se le dirá que utilice una de estas consultas, y donde quiere que se guarde el archivo. El api mandará el json al api rest y este le devolverá el fichero en base64. El api de consultas traducirá este fichero y lo escribirá en la ruta especificada por el usuario.

Tras la reunión con el cliente, se especifica que el API de consultas simplemente recibirá una consulta, la cual enviará a través de JSON al API REST, obteniendo el resultado en base64, y devolviendo esta cadena. Será deber del usuario de esta API crear la consulta con los datos adecuados y decodificar el archivo en base64.

Además del cambio anterior nombrado, en este punto del sprint se plantea la posibilidad de realizar el guardado de consultas a nivel de fichero XML o a través de un SQLite contenido en la Aplicación Web de consultas. Se decidió seguir utilizando los ficheros XML. Para ver la toma de decisión, consultas el Anexo, apartado ‘Toma de decisión. Fichero XML o SQLite’.

Al principio se pensaban incluir en un único fichero XML de consultas las consultas y las bases de datos, pero acabamos separando estas dos listas de objetos en ficheros diferentes respectivamente. Además, las consultas no podrán ser guardadas todas en el mismo fichero. Esto es debido a que el encargado de la empresa nos dijo que la Aplicación Web de consultas solo trabajaría con consultas que tuvieran SQL. Recordamos que las consultas podían especificar qué datos obtener a partir de datos de exportación. Teniendo todo esto en cuenta, se opta por hacer dos ficheros, y que ambos puedan ser modificados manualmente por los usuarios, mientras que solo las consultas con SQL se utilizarán en la Aplicación Web.

Para la realización de la Aplicación Web de consultas, inicialmente se realizó el CRUD de las consultas. Para ello se utilizó MVC con controladores, que más tarde interactuarán con las

vistas que creemos para ofrecer la interfaz gráfica. Se plantea la posibilidad de incluir reglas para la introducción de datos en las vistas. El encargado de la empresa nos aconseja simplemente indicar que campos son obligatorios y hacerlos obligatorios. No preocuparnos por si los campos están bien rellenos y demás.

El siguiente paso fue realizar un cliente para probar el API de consultas y las consultas guardadas por la aplicación web de consultas. Este cliente lee los ficheros XML de consultas y conexiones, y da la posibilidad de elegir que consulta ejecutar. Esta consulta sería utilizada con la API de consultas, y obtendrá la respuesta en base64, traduciéndola y guardando el archivo en el ordenador.

Tras acabar con la aplicación web de consultas, se puede observar que el costo de tiempo ha sido mucho mayor del previsto. Esta tarea se estimó con 5 puntos de historia, pero se subestimó la complejidad de utilizar javascript para añadir las hojas a las consultas.

La tarea de la aplicación web de consultas no está terminada del todo, pues faltaron de implementar los CSS. En el siguiente sprint se terminará esta tarea.

6.3.5. Sprint 4.

En este sprint nos centraremos en la memoria y en acabar la aplicación web de consultas. También se realizará la integración del proyecto en un producto real de la empresa.

Se aconseja por parte de los desarrolladores que prueban el producto terminado realizar errores más descriptivos. Se realizó un esquema de los distintos tipos de errores que mostraría la aplicación dependiendo de donde fuera el error y porque se ha generado.

El esquema siguiente muestra el código de la respuesta del API REST dependiendo de los errores que puedan dar.

Cuando se obtienen los datos correctamente:

- 200 OK. Exportación correcta. En el cuerpo de la respuesta está la cadena en base64.

Errores:

- 500 Internal Error. Error no capturado. Póngase en contacto con el administrador.

Estos errores son los errores del JAR no capturados...

- Errores de SQL(Ej. tabla no existe)
- Errores de malformación Excel (Ej. dos hojas con mismo nombre)
- 400 BAD REQUEST. Otros con mensaje descriptivo propio. Pueden dar errores por:
- Error de conexión con la base de datos.
- Error de conversión JSON.
- Falta de datos clave para la exportación.

Esta captura de errores no se realizó por falta de tiempo, pero se dejó documentado para futuras ampliaciones del código.

7. Implantación

Tras acabar todas las partes integrantes del proyecto quedó tiempo suficiente para incorporarlo en un proyecto real de la empresa. Se eligió un proyecto terminado recientemente.

Tras descargarlo y tenerlo operativo se organizó el código de exportación en carpetas en vez de en módulos. Como podemos ver en la figura 12, se incorporó todo el código en dos módulos. Uno con la funcionalidad de exportación y otro con la web de configuración de consultas.

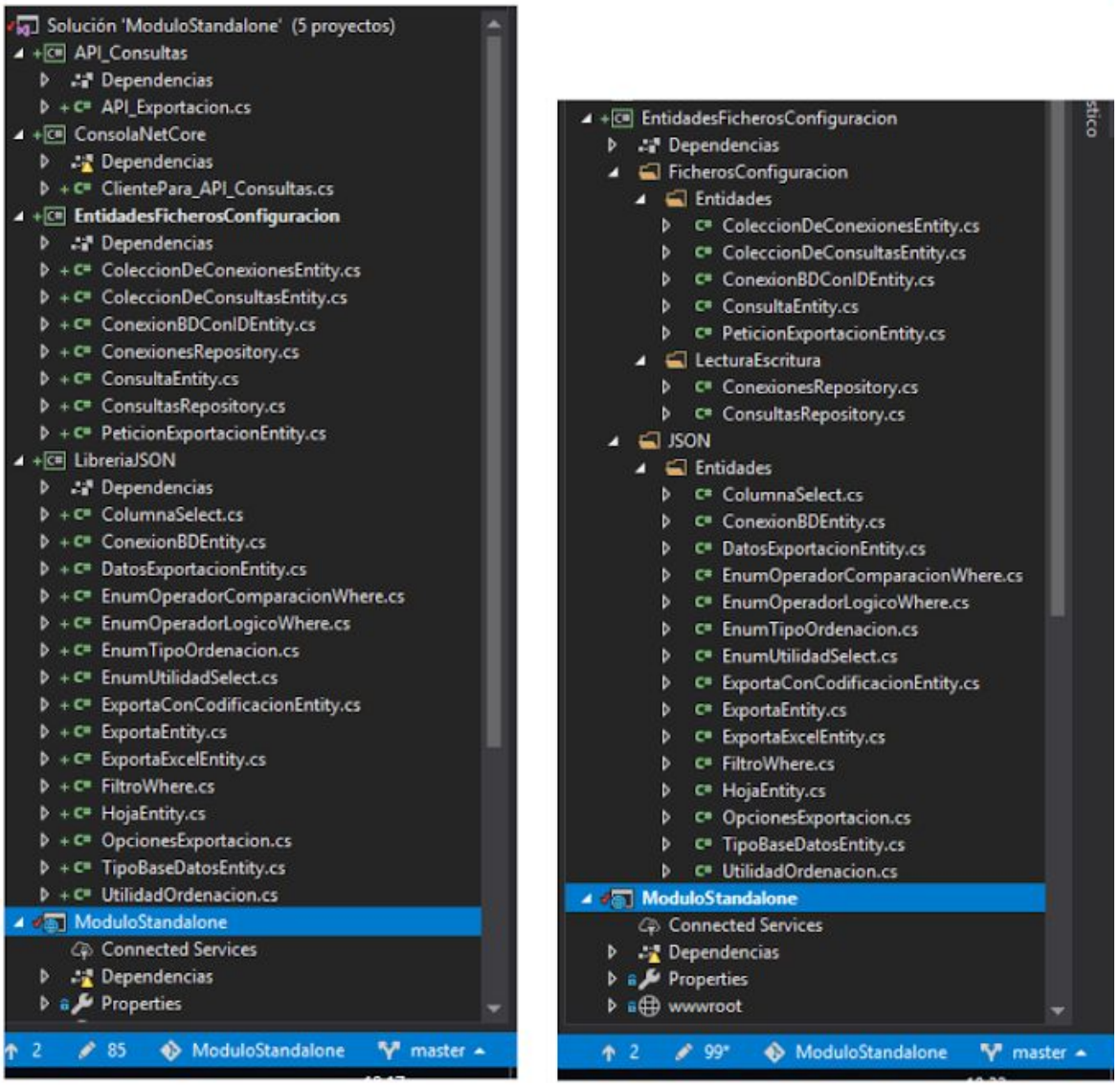

Figura 12. Estructura de clases antes y después.

Tras tener los dos proyectos fusionados, se realizaron pruebas de conexión a la BD. Tras varios intentos, conseguimos realizar las exportaciones con éxito, sin modificar el código ya implementado. En la figura 13 podemos ver la consulta generada por la API de Consultas, y el archivo creado por el cliente de prueba con los datos recibidos de la API REST. Como casi todos los intentos fallidos relacionados con la exportación, el problema estaba en la

introducción errónea de los datos de conexión o de los datos a exportar (lo cual es información que posee el usuario).

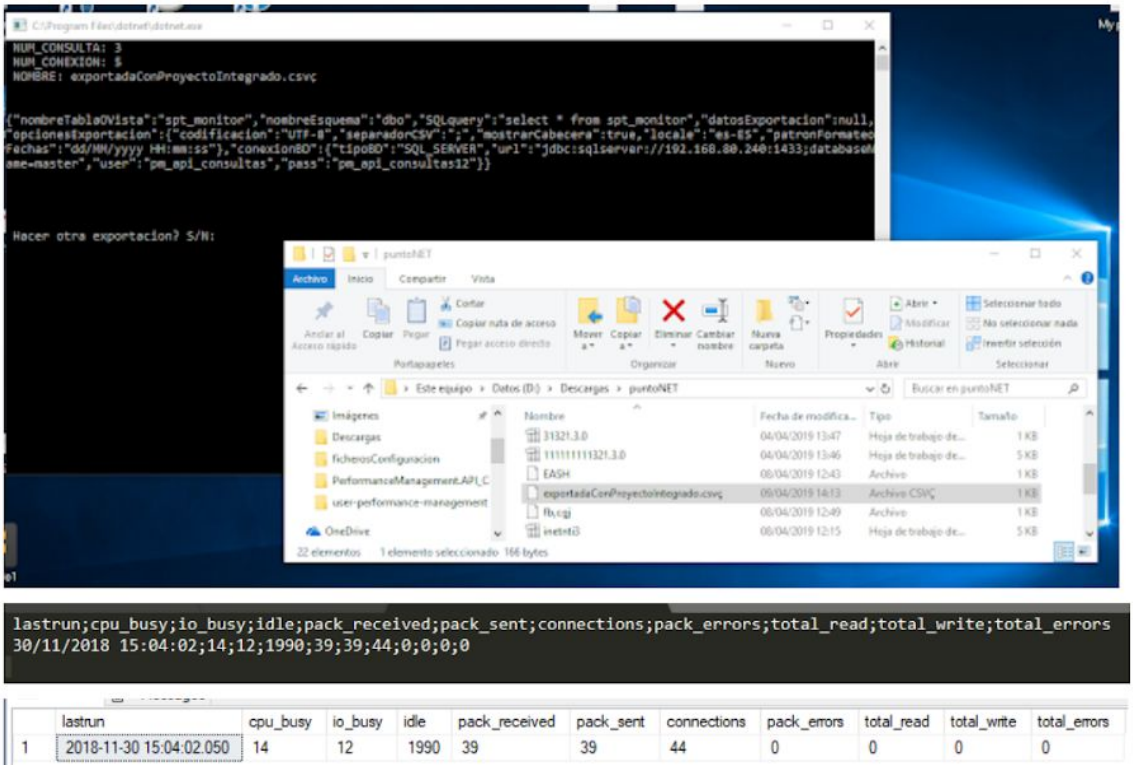

Figura 13. Resultado de la exportación del cliente de prueba.

Tras subir los cambios al repositorio del proyecto real, programadores de este proyecto hicieron de usuarios de la aplicación. A continuación se reflejan las principales mejoras recomendadas por estos usuarios.

- Los ficheros XML de configuración deberán ser accesibles desde todos los ordenadores que usen los clientes de la API de consultas.
- Se plantean problemas de cara a la integración del proyecto, ya que la API rest requiere un servidor Tomcat para desplegarse y funcionar.
- El servidor Tomcat debería tener acceso a las bases de datos del cliente para poder funcionar.
- Sería recomendable desarrollar errores más descriptivos para que los usuarios, utilizando la API, tuvieran mayor información al respecto.
- Ver qué usuarios van a poder acceder a la gestión de consultas, si van a poder configurar todos las consultas para las exportaciones o solo algún tipo de usuarios.

Puntos fuertes.

- El proyecto está bien estructurado en capas, lo que lo hace manejable y fácil de utilizar en futuras implementaciones.

8. Conclusiones

En este proyecto se ha generalizado y extendido una librería ya existente y se ha facilitado su utilización sin programar mediante una aplicación web. Hay que destacar el fuerte reajuste que sufrió el proyecto tiempo después de su inicio. Esta reorientación ha dado lugar a un producto de mayor calidad técnica y más accesible, ya que se puede utilizar desde cualquier plataforma de programación. Hay que reconocer también que para lograrlo se introduce la necesidad de disponer de un servidor para desplegar la parte programada en Java y un segundo servidor para la parte hecha en .Net. Una mejora que se podría realizar es traducir la librería escrita para java a C# para lograr un producto completamente desarrollado en .Net.

En este proyecto hemos utilizado varias tecnologías y se han seguido varios métodos de trabajo. Se ha usado *Scrum* como forma de organizar el trabajo con el responsable técnico de la empresa. En las reuniones semanales o quincenales se gestionaron los fuertes reajustes mencionados. Se esperaba mayor disposición de los participantes en estas reuniones. Los reajustes o tuvieron consecuencias negativas en el proyecto. Los puntos de historia han sido útiles y su número varió entre 30 y 50 por sprint. Las reuniones diarias sirvieron para enfocar constantemente las tareas a realizar. La herramienta *Handsoft* es recomendable. Sirve para guardar, estructurar y consultar el estado de las historias de usuario y sus tareas. Git es un gestor de versiones, pero al haber un solo desarrollador se ha convertido en una herramienta para realizar copias de seguridad. Para este fin se podrían haber usado otras soluciones. También es recomendable el uso de *DBeaber* en proyectos que utilicen varios SGBD para reducir el número de instalaciones.

Además de la parte técnica está la escritura de esta *memoria*. Como base para su realización se ha utilizado un diario, asociado a la gestión con el método Scrum, donde se anotaban los progresos, problemas, acuerdos tomados en reuniones, nuevos requisitos, etc. Pienso que la recogida cronológica de lo que ha ido sucediendo es una práctica recomendable para facilitar la tarea de escritura aunque ha consumido una cantidad de tiempo nada despreciable (15-20% del tiempo del proyecto). Entiendo que la experiencia puede ayudar a filtrar mejor la información interesante y reducir notablemente este esfuerzo.

La escritura de la memoria exige un esfuerzo mucho mayor del que uno pudiera prever. Escribir un documento técnico comprensible es una tarea ardua y las 45 horas previstas no han sido suficientes. Esta memoria se ha escrito con Google Drive y creo que su uso es muy recomendable. Al disponer de un sistema de versiones integrado, permite, por ejemplo, recuperar párrafos descartados inicialmente. Al tener el documento compartido con el tutor se evita el envío y recepción de versiones por email y se simplifica la tarea de incorporar sus comentarios y correcciones, ya que las puede poner como comentarios e incluso escribir aportaciones directamente.