Proyecto Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Herramienta de Exportación de Datos de Entornos Productivos

Autor: Ezequiel Montero Ramírez
Tutor: Fernando Cárdenas Fernández

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Herramienta de Exportación de Datos de Entornos Productivos

Autor:
Ezequiel Montero Ramírez

Tutor:
Fernándo Cárdenas Fernández
Profesor a tiempo parcial

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera: Herramienta de Exportación de Datos de Entornos Productivos

Autor: Ezequiel Montero Ramírez

Tutor: Fernando Cárdenas Fernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

El presente trabajo de fin de grado fue realizado bajo la tutela y supervisión de D. Fernando Cárdenas Fernández, a quien me gustaría expresar mayor y más profundo agradecimiento por hacer posible la realización de este trabajo desde su inicio, proporcionándome la idea, hasta el final de la realización de este pasando por todos los momentos intermedios donde he requerido ayuda.

A mis padres, por traerme a este mundo y guiarme día a día hasta donde hoy mis propios pasos me han llevado.

A mi padre, por animarme a estudiar ciencias, por incentivarme a estudiar este grado y por siempre verme capaz de hacer cualquier cosa.

A mi madre, por ver donde nadie más ve, por hacerte cargo de tantas cosas y responsabilidades para que yo pueda estar completamente centrado en la consecución de mis muchos objetivos, la finalización de este trabajo uno de ellos.

A mis abuelos Andrés, Araceli, Ezequiel y Teresa, por ser ejemplos de valores de vida y moldearme como persona.

Al resto de mi familia por siempre mostrar interés por mis avances en el que sabían, duro y complejo proceso de estudiar este grado que hoy toca su fin.

A todos mis profesores que han dejado huella en mí en forma de recuerdos, conocimientos y enseñanzas imprescindibles para haber llegado a donde hoy estoy.

A mis amigos de la carrera, por el enorme significado que le han dado a la palabra compañerismo durante los momentos difíciles y por la enorme amistad forjada durante los momentos buenos que ya va más allá de cualquier estudio o formación académica. A Pepelu, David, Sebas, Fabio, Rafa, Javi, Miguel Ángel, Joseca y Ramsés.

A todas las personas que me ha dado el mundo del baloncesto, en especial a Fran, Jaime, Álvaro y a todos mis compañeros de equipo, los cuales han convertido mis pocos ratos libres en tiempo de disfrute pleno forjando así las mejores amistades que hoy en día tengo.

Por último, a todas esas amistades y personas especiales, que no entran en ninguna categoría por su forma de ser única, esas que nunca sobran y su valor es incalculable.

Ezequiel Montero Ramírez

Estudiante de Grado en Ingeniería de Tecnologías de Telecomunicación

Sevilla, 2019

Resumen

Cuando comencé a buscar temática de Trabajo Fin de Grado muchas dudas rondaban mi cabeza sobre qué temática escoger, qué tecnología usar y mi capacidad para llevar esto a cabo. Todo esto hasta que me reuní con D. Fernando Cárdenas Fernández y me ofreció la idea, entre otras, de realizar una **herramienta de exportación de datos de entornos productivos**. En ese momento mis dudas se disolvieron y dado mi interés por la ciberseguridad, la protección de datos y la privacidad enseguida acepté ilusionado por la posibilidad de implementar una herramienta de objetivo doble que diera a las empresas lo necesario para **probar nuevos entornos productivos y funcionalidades en desarrollo y/o detectar el origen de fallos en los datos**, todo esto **manteniendo intactos los derechos de las personas establecidos en el Reglamento General de Protección de Datos de la Unión Europea y llevados a la práctica en por la Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de los Derechos Digitales**. Una vez llevado a cabo este trabajo concluyo que esta herramienta, tanto como todas las similares que puedan surgir de aquí en adelante, será de gran provecho en un futuro inmediato dada su utilidad en las empresas para la obtención de datos con los que realizar las funcionalidades anteriormente definidas respetando las nuevas leyes de protección de datos.

Abstract

When I started looking for a thematic for my end-of-degree project I had lots of doubts surrounding my head, what thematic to choose, what technology to use to if I was going to be able to get it done. That feeling ended when I had a reunion with Mr. Fernando Cardenas Fernandez and he offered me the idea, between others, of implementing an **exportation data tool between productive environments**. In that moment all the doubts went away and given my interest in cybersecurity, data protection and privacy I got excited right away with the possibility of creating a tool with a double aim: giving to enterprises what is needed for **testing new productive environments and new developing functionalities, and/or troubleshooting problems located in the data while complying with the rights stablished in the union european "Reglamento General de Protección de Datos" and put into price in the spanish "Reglamento General de Protección de Datos de la Unión Europea y llevados a la práctica en por la Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de los Derechos Digitales"**. Once I have concluded this project I can affirm that the tool I developed, and the ones that will emerge from now on, will have a great utility for enterprises given the ability of obtaining a set of data to accomplish the functionalities said before while respecting the rights established in the new data protection laws.

Índice

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

No hay una segunda oportunidad para una primera impresión.

Oscar Wilde

Dada la reciente **actualización del marco legislativo europeo**, y por tanto del español, en lo respectivo a la **protección de datos** y la imperiosa necesidad existente en las aplicaciones telemáticas de manejar datos para llevar a cabo sus funcionalidades surge una oportunidad única para desarrollar una **herramienta de exportación de datos de entornos productivos que respete y acate dicho maco legal**.

Dicha herramienta será de gran utilidad para las empresas y otro tipo de organizaciones para, a partir de los datos existentes almacenados en base de datos, **obtener un/os nuevo/s conjunto/s de ellos que respetan las restricciones establecidas por la ley con los que llevar a cabo funciones, entre otras, tales como el testeo de nuevas funcionalidades o tareas de troubleshooting**.

La citada oportunidad se vuelve aún más única cuando las herramientas que existían con anterioridad han quedado obsoletas debido a la derogación de la Ley orgánica 15/1999 para la entrada en vigor de la actual legislación. El resto de las herramientas existentes, que serán comentadas en un capítulo posterior, son herramientas de gestión de datos muy genéricas cuyo uso para llevar a cabo las funciones que esta herramienta realizará no es sencillo para el usuario careciendo además de eficiencia en el proceso.

Para la realización de esta herramienta es necesario un **estudio en profundidad de la nueva legislación** para conocer tanto las pautas generales definidas en el Reglamento General de Protección de Datos de la Unión Europea como la aplicación específica del estado español a través de la Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de los Derechos Digitales. Es este estudio el que **define las posibles transformaciones a realizar a los datos para que cumplan la legislación y a su vez sigan siendo los datos útiles para la funcionalidad que se les desee dar.**

De esta manera dichos documentos legales determinan los objetivos que debe alcanzar la herramienta a la hora de trabajar con los datos almacenados en bases de datos. Para alcanzar los citados objetivos se hace necesario un análisis del estado del arte que viene ligado de forma intrínseca a una serie de tecnologías. La elección de una o varias de ellas condiciona el diseño a realizar del proyecto, y esto a su vez condiciona la implementación del mismo.

Este será por tanto el modo de proceder a la hora de la realizar este trabajo.

2 OBJETIVOS

Cuanto más alto coloque el hombre su meta, tanto más crecerá.

Friedrich Schiller

Los dos grandes objetivos de este trabajo fin de grado son, por un lado, **analizar las restricciones que imponen el Reglamento General de Protección de Datos de la Unión Europea y la Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de los Derechos Digitales** al tratamiento de datos para así definir los requisitos y restricciones del segundo objetivo a llevar a cabo, **implementar una herramienta de exportación de datos de entornos productivos que respete dicha legislación.**

Por tanto, estos objetivos pueden ser fácilmente clasificados y divididos en subobjetivos de la siguiente forma:

2.1. Subobjetivos de cumplimiento legal

Como ya se ha explicado **estos subobjetivos nacen del análisis de los textos legales considerado**s para la realización de este trabajo fin de grado **y suponen restricciones y requisitos a cumplir a la hora de diseñar e implementar la herramienta de exportación de datos de entornos productivos**.

Otra consideración de carácter legal a tener en cuenta es que el Reglamento General de Protección de Datos de la Unión Europea establece pautas legislativas, sobre qué condiciones se deben cumplir, en lo relativo a la protección de datos que son traducidas a leyes por los países miembros. De esta forma en España dichas pautas son traducidas a articulado en la Ley Orgánica 3/2018 de Protección de Datos Personales y Garantías de los Derechos Digitales llevando a detalle la forma concreta de cumplir con dichas condiciones establecidas en el Reglamento General de Protección de Datos de la Unión Europea.

De esta manera ambos textos apuntan en la misma línea legislativa y cualquiera de ellos son válidos para establecer los requisitos y restricciones para nuestra herramienta a implementar. Para este capítulo del texto se utilizará como referencia para citar las predisposiciones y el articulado del Reglamento General de Protección de Datos de la Unión Europea.

2.1.1 Clasificación de los datos

La herramienta a implementar debe respetar la clasificación que se obtiene de los citados textos legales y las restricciones e imposiciones que cada tipo de dato trae asociados.

El Reglamento General de Protección de Datos de la Unión Europea define los siguientes tipos de datos:

En su artículo nº 4 se define como **datos personales** a **toda información sobre una persona física identificada o identificable, considerándose persona física identificable toda persona cuya identidad pueda determinarse, directa o indirectamente, en particular mediante un identificador, como por ejemplo un nombre, número de identificación, datos de localización, un identificador en línea o uno o varios elementos propios de la identidad física, fisiológica, genética, psíquica, económica, cultural o social de dicha persona.**

Dentro de este tipo de dato se detallan tres subtipos que se detallan a continuación, siendo el último de especial interés para nuestro trabajo:

Datos genéticos: En la predisposición nº 34 se establece que debe entenderse por datos genéticos los datos personales relacionados con características genéticas, heredadas o adquiridas, provenientes del análisis de una muestra biológica de la persona física en cuestión. En particular un análisis de ADN, ARN o cualquier otro elemento que permita obtener información equivalente.

Datos relativos a la salud: Quedan definidos en la predisposición nº 35 como aquellos datos que representan cualquier estado de salud física o mental del pasado presente o futuro, todo símbolo o dato asignado a una persona física que la identifique de manera unívoca a efectos sanitarios, la información obtenida de pruebas o exámenes de una parte del cuerpo o sustancia corporal.

Datos sensibles: datos de carácter personal que por su naturaleza son particularmente sensibles en relación con los derechos y libertades fundamentales. Entre ellos datos que revelen el origen racial o étnico.

Estos últimos requieren especial protección y no deben ser tratados excepto en determinados contextos que se detallaran más adelante.

De forma paralela se define un tipo de dato en la predisposición nº 38, que puede englobar datos de los tipos y subtipos anteriores, que es el de datos pertenecientes a niños. Estos datos deben llevar consigo una protección especial.

De forma implícita se definen los **datos no personales** como aquellos que no aplican a lo definido por el artículo nº 4.

2.1.2 Análisis del tratamiento de datos

Al igual que nuestra herramienta de exportación ha de respetar la clasificación de los datos extraída del reglamento también ha de respetar las **restricciones que se imponen en este al tratamiento de datos sobre cada uno de los distintos tipos.**

Para la comprensión correcta de estas restricciones es necesario conocer **sobre qué casos es aplicable el reglamento**. En este mismo se define que:

Lo establecido en el reglamento **se aplica al tratamiento total o parcialmente autorizado de datos personales**, así como al tratamiento no automatizado de datos personales contenidos o destinados a ser incluidos en un fichero (Artículo 2).

Define además en el artículo 4 el concepto de **seudonimización** como:

"el tratamiento de datos personales de manera tal que ya no puedan atribuirse a un interesado sin utilizar información adicional, siempre que dicha información adicional figure por separado y esté sujeta a medidas técnicas y organizativas destinadas a garantizar que los datos personales no se atribuyan a una persona física identificada o identificable; "

Este concepto es vital para la comprensión de la predisposición nº 26 por la cual se indica que los principios de protección de **datos no deben aplicarse a los datos convertidos en anónimos de forma que el interesado deje de ser identificable (seudonimizados)**. En consecuencia, lo establecido en el reglamento no afecta al tratamiento de dicha información anónima, inclusive con

fines estadísticos o de investigación.

Una vez determinados los casos donde el reglamento es aplicable, se establecen las ya citadas **restricciones**:

Tal y como indica el artículo 6 para que el **tratamiento sea lícito** se debe cumplir **al menos una** de las siguientes **condiciones**, que el **interesado diera su consentimiento** para el tratamiento de datos para uno o varios fines específicos o que la **realización del tratamiento de los datos sea necesaria para la ejecución de un contrato** en el que el interesado es parte **o para la aplicación de medidas precontractuales a petición del interesado.**

Por otro lado, en el artículo 9 se **prohíbe explícitamente el tratamiento** de datos que revelen el origen étnico o racial, las opiniones políticas, las convicciones religiosas o filosóficas, o la afiliación sindical, y el tratamiento de datos genéticos, datos biométricos dirigidos a identificar de manera unívoca a una persona física, datos relativos a la vida sexual o la orientación sexual de una persona física (**datos sensibles**) **salvo en determinadas situaciones**.

Dichas situaciones se encuentran también definidas en el mismo artículo, siendo estas que el **interesado diera su consentimiento** (pudiendo el derecho de la Unión o de un país miembro establecer que la prohibición no pueda ser levantada por el consentimiento del interesado) **y/o que el tratamiento se refiera a datos que el interesado haya hecho manifiestamente públicos.**

Por último, el reglamento indica a través de la predisposición nº 45 que no se requiere que cada tratamiento individual se rija por una norma específica, sino que **una norma puede ser suficiente como base para operaciones de tratamiento de datos de distinta índole**.

2.1.3 Responsabilidades del tratamiento de datos

El último de los subobjetivos de cumplimiento legal es que nuestra herramienta de exportación sea **consecuente con las responsabilidades que recaen sobre el encargado** de tratar los datos, es decir, que **ayude mediante sus funcionalidades a cumplir con las responsabilidades que el reglamento le acarrea**. **Estas obligaciones son**:

El responsable debe adoptar políticas internas y aplicar medidas que cumplan en particular los principios de protección de datos desde el diseño y por defecto. Dichas medidas podrían consistir entre otras en **dar transparencia a las funciones y tratamiento de datos personales**. (Predisposición 78)

Esto se complementa con lo establecido en la predisposición nº 82 por la cual, para demostrar la conformidad con el reglamento, el responsable del tratamiento debe mantener registro de **actividades de tratamiento bajo su responsabilidad**.

El artículo nº 30 detalla que el contenido de **dicho registro debe incluir una descripción tanto de la categoría de datos tratados como de las medidas y reglas aplicadas al tratamiento de datos**, forzando así a que se cumpla lo establecido en la predisposición nº 63, por la cual todo **interesado debe tener el derecho a conocer la lógica implícita en todo tratamiento automático de datos personales.**

Otro aspecto que define el articulo nº 30 respectivo al citado registro es que deberá constar por escrito, inclusive formato electrónico y a disposición de la autoridad de control que lo solicite.

2.2. Subobjetivos de implementación

Estos subobjetivos suponen una forma de desmembrar y granularizar el objetivo final de este proyecto: la implementación de una herramienta de exportación de datos de entornos productivos.

Como ya se ha explicado **estos subobjetivos nacen, en una gran parte, de una traducción de los subobjetivos de cumplimiento legal y los requisitos y restricciones que estos traen consigo**. Por otro lado,

también habrá subobjetivos no **enfocados** a un cumplimiento legal sino **a aportar calidad a la herramienta**.

De esta forma el establecimiento de los subobjetivos de implementación supone uno de los dos pasos fundamentales para el comienzo del diseño de la herramienta. El otro paso fundamental será la elección de las tecnologías a emplear que se detallará en un capítulo posterior.

Una vez dicho esto se procede a enumerar los subobjetivos de implementación clasificándolos según su asociación o no con uno o varios subobjetivos de cumplimiento legal.

2.2.1 Subobjetivos de implementación propios

La herramienta deberá ser una **aplicación web** y tendrá la estructura de **fichero war**. Estos subobjetivos se marcan bajo el propósito de hacer **visualmente atractiva la aplicación al usuario** con las tecnologías existentes para ello en el mundo de la programación web y para que al colocar el proyecto en la carpeta adecuada del servidor web que se desee este se **exporte de forma automática**.

Se proveerá al usuario **un login** para evitar en la medida de lo posible la utilización de la herramienta por parte de personas no autorizadas y/o sobre las que no recae la responsabilidad legal del tratamiento de datos.

La aplicación ofrecerá al usuario un **formulario** de interacción cuya finalidad será la de permitir a este la **creación de una base de datos** a la cual se desee exportar los datos que vayan a ser tratados.

Se mostrará al usuario las bases de datos existentes disponibles, permitiendo seleccionarlas una por una para ver las tablas que estas contienen, con el objeto de no requerir un conocimiento de la estructura del servidor de bases de datos por parte del usuario de la aplicación. Por cada una de las tablas existirá un botón cuya funcionalidad sea iniciar el proceso de tratamiento de datos sobre los almacenados en dicha tabla.

Una vez seleccionada la tabla con la que sea desee trabajar (se podrá trabajar con varias, pero no de forma simultánea) la herramienta mostrará al usuario **una vista con todas las columnas e información descriptiva de cada una de estas**, por ejemplo: nombre de la columna, si es clave de algún tipo (primaria o externa), si tiene valor por defecto, si puede ser nula…

2.2.2 Subobjetivos de implementación asociados al cumplimento legal

La herramienta tiene que ser capaz de ofrecer a quien la use un **mecanismo para catalogar** los diferentes **datos almacenados** en la base de datos **conforme a la clasificación extraída del Reglamento General de Protección de Datos de la Unión Europea.**

En el **caso de que los datos sean clasificados como sensibles debe permitir al usuario de la herramienta indicar si tiene consentimiento para el tratamiento** de estos, representando dicho consentimiento el consentimiento explícito por parte del interesado o la circunstancia de que el interesado haya hecho públicos dichos datos.

La herramienta de exportación de datos debe proporcionar al usuario de esta la **capacidad de seudonimizar/anonimizar** datos permitiendo así usar estos para cualesquiera sean los fines.

Como respuesta a la necesidad legal de realizar un registro con las actividades de tratamiento la herramienta **debe generar de forma automática una serie de ficheros** (de nombre como la tabla de la base de datos a la que hacen referencia) **cuyo contenido abarcará el carácter del dato según la clasificación del reglamento, la función de tratamiento empleada (para así darle la transparencia necesaria de cara al usuario) y el instante exacto en el que se realizó el tratamiento**.

3 ESTADO DEL ARTE Y TECNOLOGÍAS

La tecnología por sí sola no basta. También tenemos que poner el corazón.

Jane Goodall

En este capítulo se analizará por un lado las herramientas de funcionalidad igual o similar a la herramienta a desarrollar y por otro lado se comentarán las tecnologías escogidas, la justificación de la elección y el estado de estas tecnologías.

Es importante recordar que la elección de una/s tecnología/s u otra/s afecta de forma sustancial a la forma de realizar el diseño de la herramienta de exportación de datos. Así pues, a la hora de elegir cada una de las tecnologías se ha procurado siempre que las tecnologías escogidas sean punteras o de uso mayoritario y que simplifiquen el diseño optimizando el tiempo necesario a dedicar a la implementación a través de la reutilización de código.

3.1 Estado del arte

Antes de comenzar a analizar el estado de las distintas alternativas a la herramienta de exportación de datos a implementar es bueno volver a incidir en un detalle ya comentado en la introducción de este texto.

El Reglamento General de Protección de Datos de la Unión Europea fue publicado el 27 de abril de 2016. Como ya se ha explicado en este texto el citado reglamento establecía una serie de directrices legislativas al respecto de los derechos de protección de datos, las cuales debían ser traducidas a leyes por cada uno de los estados miembros. Explicado de una forma más sencilla **el reglamento establece el qué y los países miembros a través de sus leyes deben establecer cómo llevarlo a cabo**.

En España, dicha traducción de directivas a leyes se materializa en la **Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales. Esta ley deroga la anterior Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal dejando así obsoletas las herramientas de exportación que existieran con anterioridad** conforme a dicha ley ya derogada.

Por otro lado existen como solución a este problema **soluciones de Oracle** muy genéricas y con una serie de **inconvenientes** a la hora de aplicarse a este problema. Estos inconvenientes resultan ser, en primer lugar un **elevado costo**, la restricción de que **solo se pueden aplicar sobre bases de datos de Oracle** y por último y no menos importante el hecho de que, **al ser una solución tan genérica** y válida para tantos otros problemas**, el esfuerzo en cumplir con la legalidad recae sobre el usuario de la aplicación en lugar de sobre la aplicación misma.**

3.2 Tecnologías

A continuación, se van a comentar las tecnologías escogidas y la justificación de por qué usar estas. Más adelante en la siguiente subsección se analizará el estado de estas tecnologías en la actualidad.

En primer lugar, la decisión más importante fue que tecnología a usar para el desarrollo de la aplicación web en sí misma. La elección fue utilizar el framework **Java Spring**, más concretamente **el módulo Spring MVC** que permite desarrollar aplicaciones web mediante el patrón de diseño Modelo-Vista-Controlador. La justificación de la elección de este módulo del framework es la posibilidad de con la misma tecnología **implementar tanto el front end como el back end de nuestra aplicación web**.

Como complemento a esto, para la generación de páginas web dinámicas se empleó **JSTL (Java Servlet Tag Library)** debido a su **perfecta integración con Spring MVC y el ahorro de tiempo que suponía con respecto al desarrollo de forma de directa de scriptlets JSP.**

Para el aspecto visual se decidió utilizar Apache Tiles y Bootstrap.

Apache Tiles es un framework de composición de plantillas que permite definir fragmentos los cuales pueden ser ensamblados en tiempo de ejecución. De esta forma definiendo una serie de fragmentos útiles para todas las páginas somos capaces de **darle un aspecto común a nuestra aplicación** durante todo su flujo de utilización. Además, al **reutilizar dichos códigos (fragmentos)** somos capaces de ahorrar una gran cantidad de tiempo a la hora de la implementación.

Bootstrap, por su parte, es una biblioteca multiplataforma que nos proporciona un conjunto de herramientas de código abierto y plantillas de diseño para nuestra aplicación web. El principal beneficio de su uso para este proyecto es la posibilidad **de generar vistas visualmente atractivas para el usuario** de la aplicación en muy poco tiempo **reutilizando los componentes ya existentes**, permitiendo así que el tiempo de implementación se dedique en mayor proporción al desarrollo de funcionalidades y en menor proporción a hacer los elementos visualmente agradables.

Dada la necesidad de exportar datos almacenados en base de datos será necesario un framework o herramienta que nos permita acceder y manipular datos relacionales y un conector que nos permita, como su nombre bien indica, conectarnos a la instancia de la base de datos.

Dado que a la hora de definir este proyecto se decidió usar como sistema de gestión de base de datos una instancia de MariaDB el conector que necesitaremos será mysql-connector. Esto se debe a que MariaDB es una bifurcación directa de MySQL que garantiza la existencia de una versión gratuita y de código libre de esta. En la práctica, las versiones de MariaDB reemplazan directamente a las mismas versiones de MySQL.

Por otro lado, para cubrir la necesidad del acceso y manipulación de datos relacionales recurrimos a Spring JDBC. **Spring JDBC** es un marco de código abierto cuyo elemento básico es el **JdbcTemplate**. Este elemento actúa como una plantilla que nos **permite simplificar las consultas a la base de datos ya que esta automáticamente se encarga de establecer y cerrar la/s conexión/es necesarias para llevar a cabo las consultas SQL, gestionar los recursos asociados a estas y manejar y gestionar las posibles excepciones del tipo SQLException que puedan surgir**. Esto optimiza el tiempo a dedicar a la codificación del proyecto y al ser una herramienta del mismo framework que estamos usando para desarrollar la aplicación web, la integración es muy sencilla.

Por otra parte, para responder al subobjetivos de implementación de proveer al usuario de un login necesitaremos los **módulos spring-security-web y spring-security-config**, como su nombre indica pertenecientes a Spring.

Además, para construir el proyecto e integrar todas estas dependencias (tecnologías) se utilizará **Maven** debido a su sencillo modelo de configuración de la construcción de proyectos basado en formato XML, el Project Object Model (POM, pom.xml).

Por último, todo esto será llevado a cabo a través de **Spring Tool Suite**, un entorno de desarrollo basado en Eclipse construido específicamente para el desarrollo de proyectos con el framework Spring.

3.2.1 Estado de las Tecnologías

A continuación, se procede a comentar la situación actual de cada una de las tecnologías a utilizar en la implementación en proyecto. Como consideración previa se indica que la actualidad del módulo Spring JDBC considerará englobada dentro de la actualidad de Java Spring y Java Spring MVC. De esta manera y siguiendo el mismo orden que en el apartado anterior:

Java Spring: Supone el **framework Java más usado actualmente** tanto en las empresas (Finoit, 2019) como en el ámbito general (Whizlabs, 2019), teniendo un uso superior a Java EE, continuando y confirmándose así la tendencia ya detectada en 2016 (Jrebel, 2016).

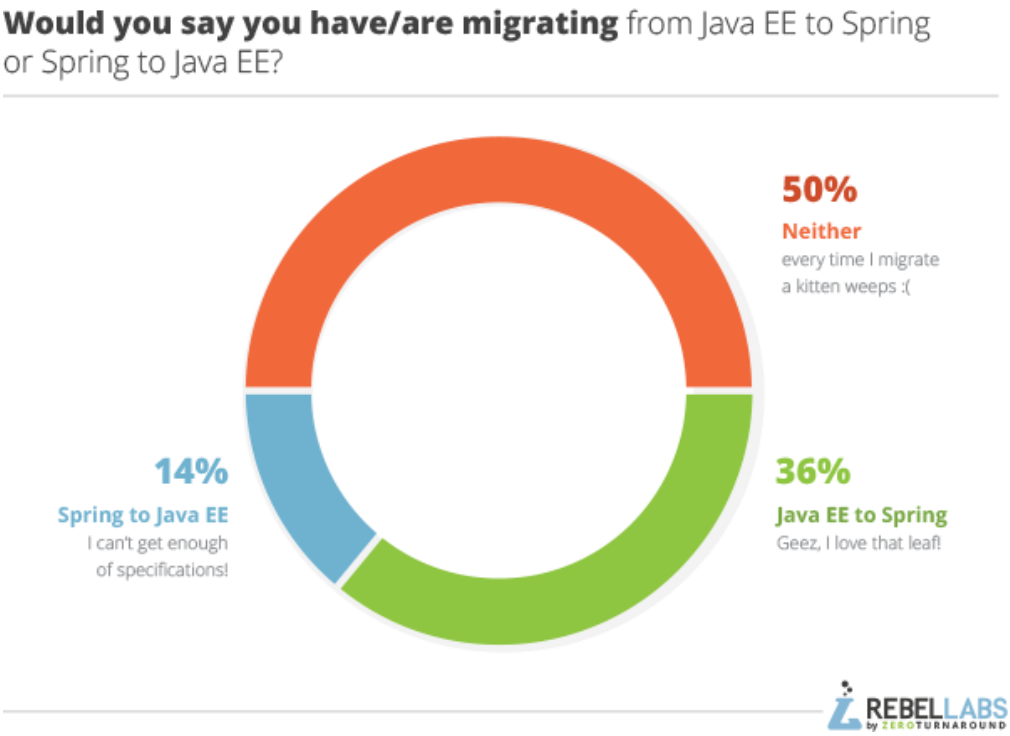


Figura 3-1. Tendencia Migratoria de Java Spring a Java EE y viceversa.

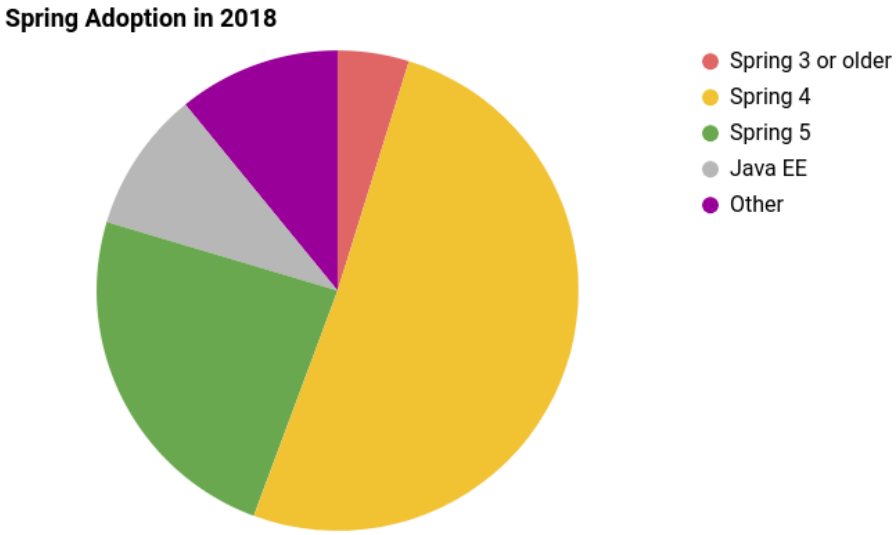


Figura 3-2. Porcentaje de adopción de Spring.

Parte de este éxito es debido al ascenso de Spring Boot que ha aumentado su porcentaje de uso en un 14% a lo largo del 2019 (JetBrains, 2019).

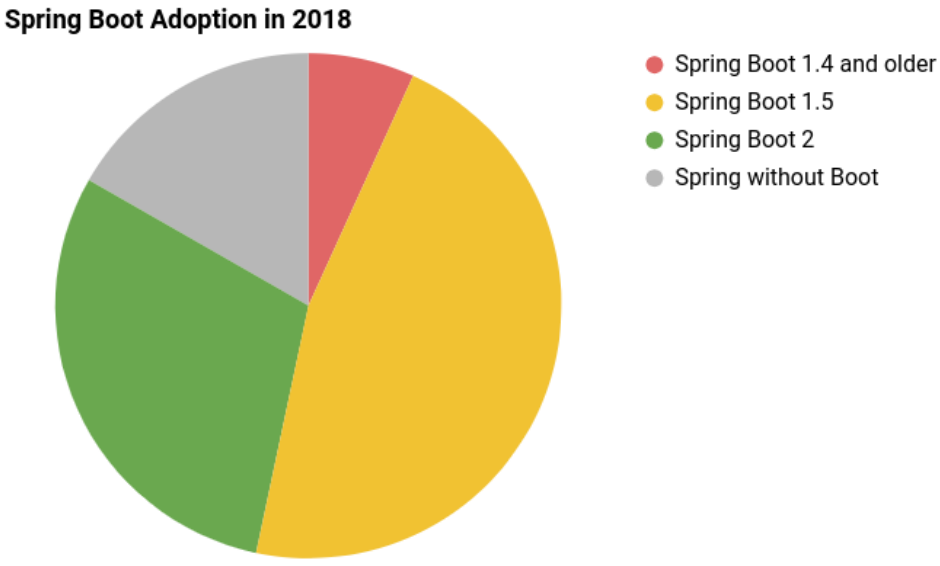


Figura 3-3. Porcentaje de uso de Spring Boot.

Java Spring MVC: Este módulo de Spring ha tenido en los últimos 5 años un gran éxito en parte debido a la fama del framework al que pertenece y que justo antes hemos analizado, pero en la más reciente actualidad se ha comenzado a ver superado por Spring Boot (JetBrains, 2019) como ya se podía prever en informes de tendencias de unos años atrás (Whizlabs, 2016)

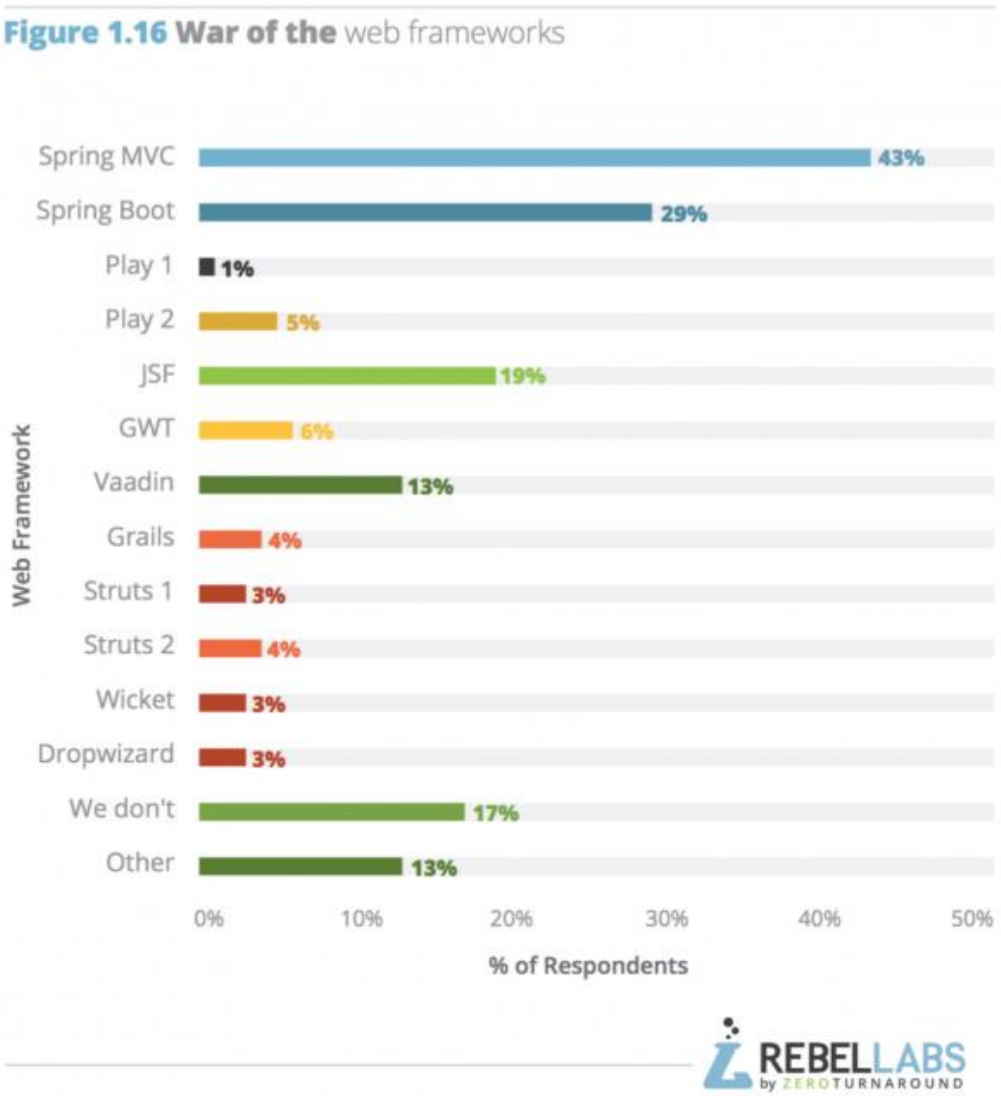


Figura 3-4. La guerra de los frameworks basados en java sobre el total de usuarios.

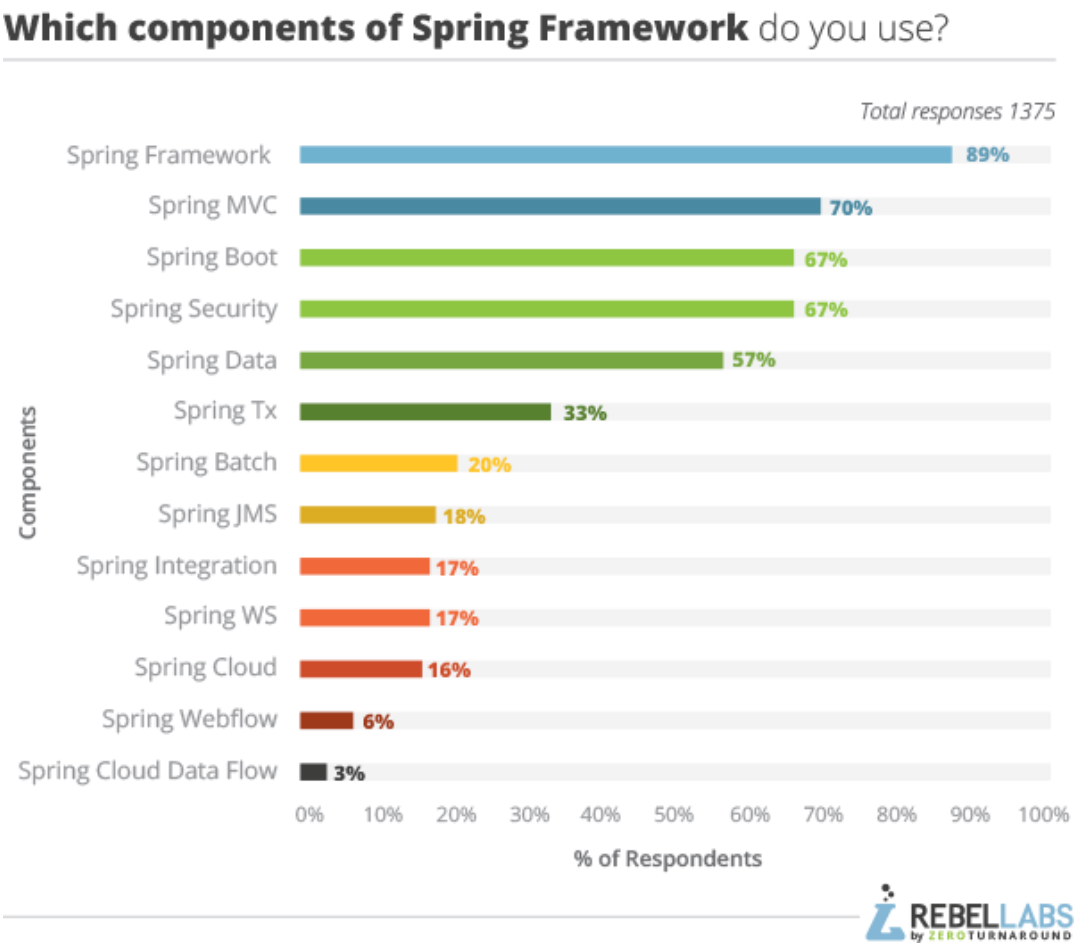


Figura 3-5. Componentes de Spring más usados sobre el total de usuarios.

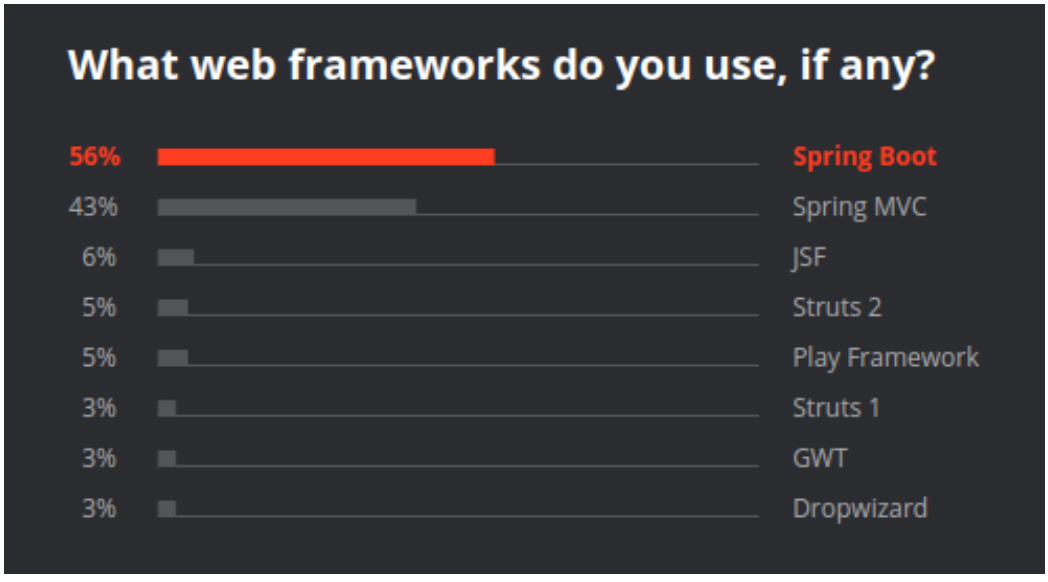


Figura 3-6. Componentes de Spring más usados sobre el total de usuarios.

JSTL: supone una más de las **alternativas** válidas a implementar los archivos JSP de forma tradicional mediante **scriptlets**.

Apache Tiles: Se trata de un proyecto abandonado en su desarrollo (Apache, 2019) pero que sigue siendo utilizado dada su alta utilidad a la hora de desarrollar interfaces de usuario

Bootstrap: En un estudio realizado a finales de 2018 se determinó que Bootstrap se encontraba **presente en el 18%**(en torno a 18 millones de webs en total) **de los sitios web**, teniendo además una **cuota de mercado entre los frameworks de igual funcionalidad de entre el 52% y el 72%**. Todo esto unido a que su **repositorio de Github** sea el **más popular** de todo el sitio web. Además, Bootstrap pese a seguir un ritmo más lento que los años anteriores, **continua con su crecimiento** y expansión (OS Training, 2018).

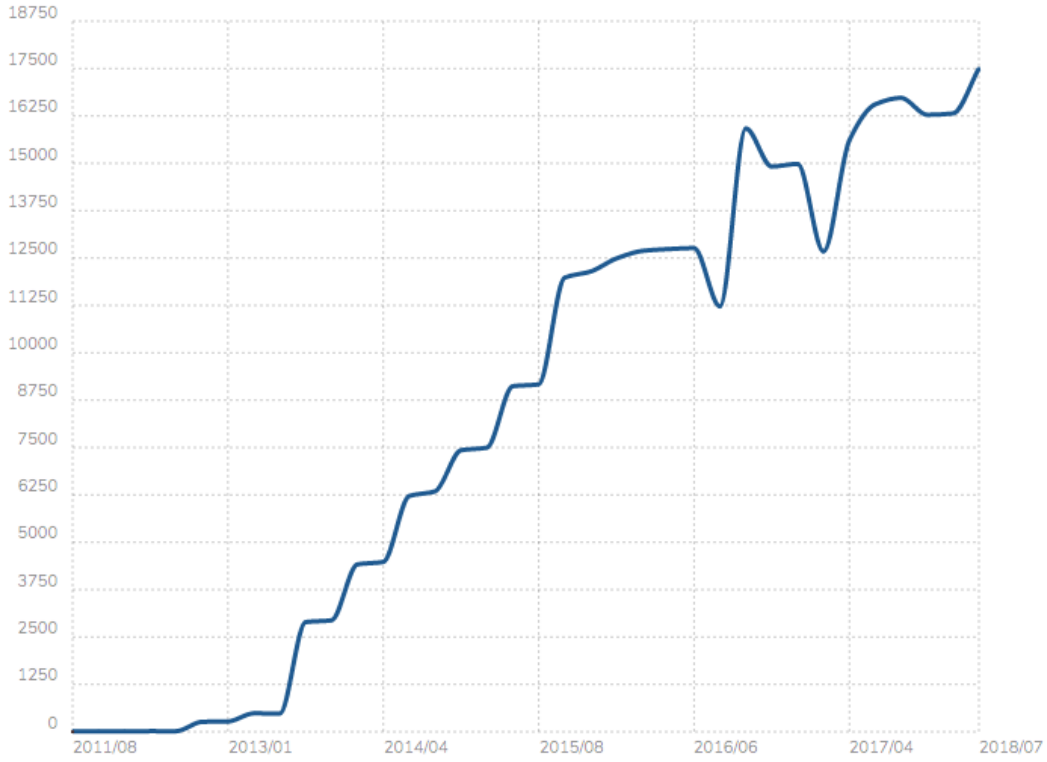


Figura 3-7. Evolución del número de webs con Bootstrap (en miles de webs).

Spring Security: Spring Security es un marco JAVA que proporciona autenticación y autorización, entre otras características de seguridad, para aplicaciones empresariales y figura como componente del framework Spring. Su utilización en la actualidad puede observarse en la figura 3.5, figurando en el **top 4 de componentes más usados de Spring** (Baeldung, 2018).

Maven: En la actualidad esta **herramienta** es referencia en lo que respecta a la construcción de proyectos siendo la **más utilizada**, solo seguida de cerca por Gradle y en menor medida por su por Ant (Baeldung, 2018 y JetBrains, 2019)

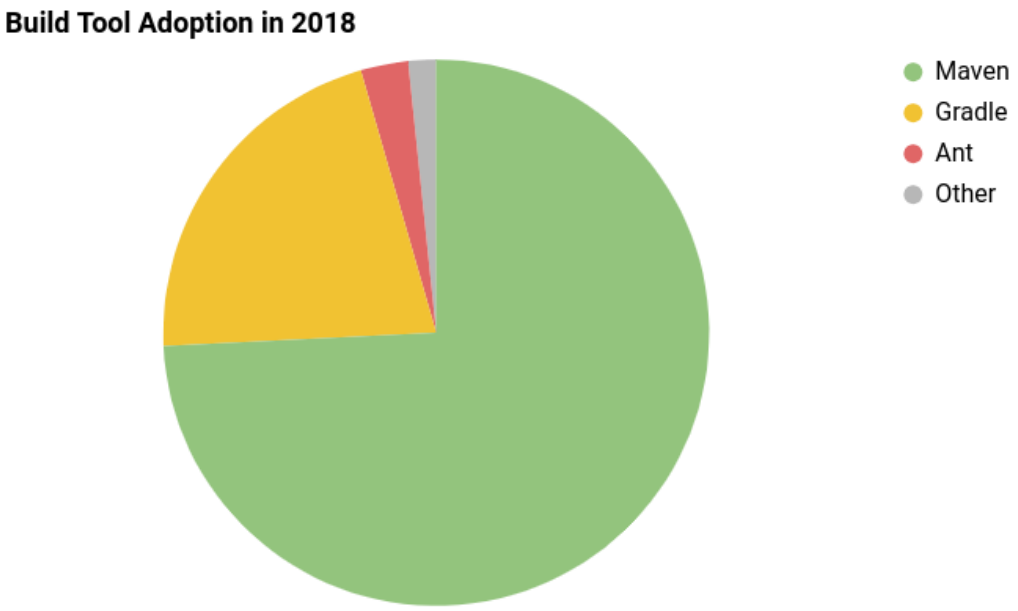


Figura 3-8. Uso de herramientas de construcción de proyectos en 2018.

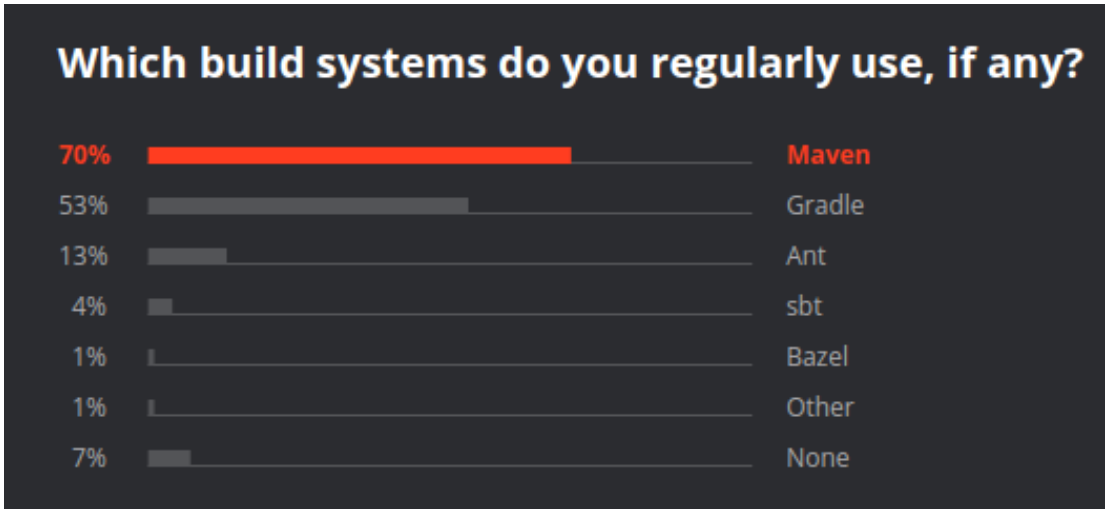


Figura 3-9. Uso de herramientas de construcción de proyectos en 2019 sobre el total de usuarios.

Spring Tool Suite: Englobado dentro de los entornos de desarrollo basados en eclipse, este grupo supone la **principal alternativa a InteliJ IDEA**, el kit de desarrollo más utilizado para aplicaciones Java (Baeldung, 2018 y JetBrains 2019).

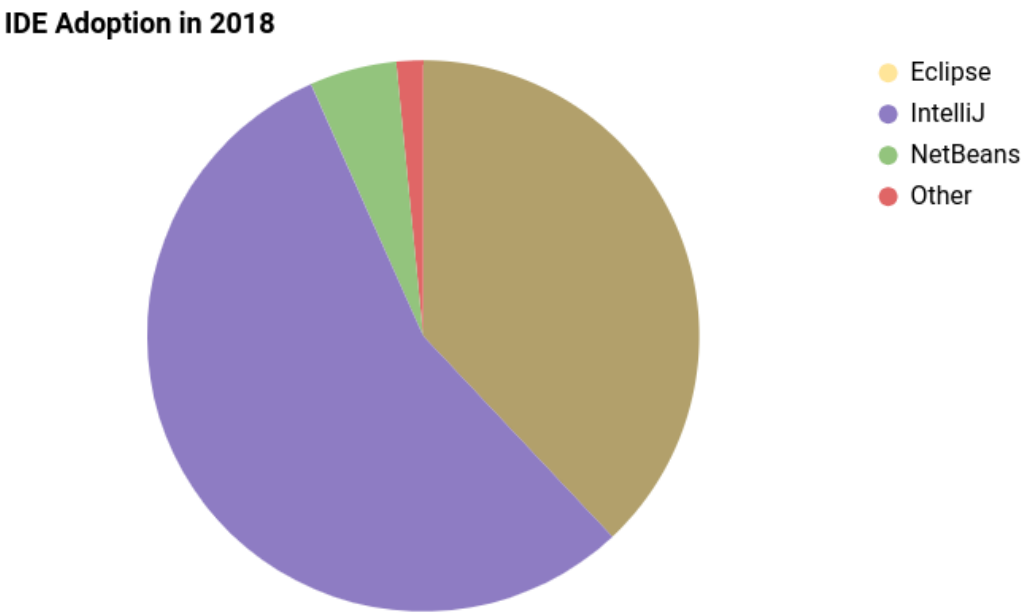


Figura 3-10. Uso de entornos de desarrollo en porcentaje total.

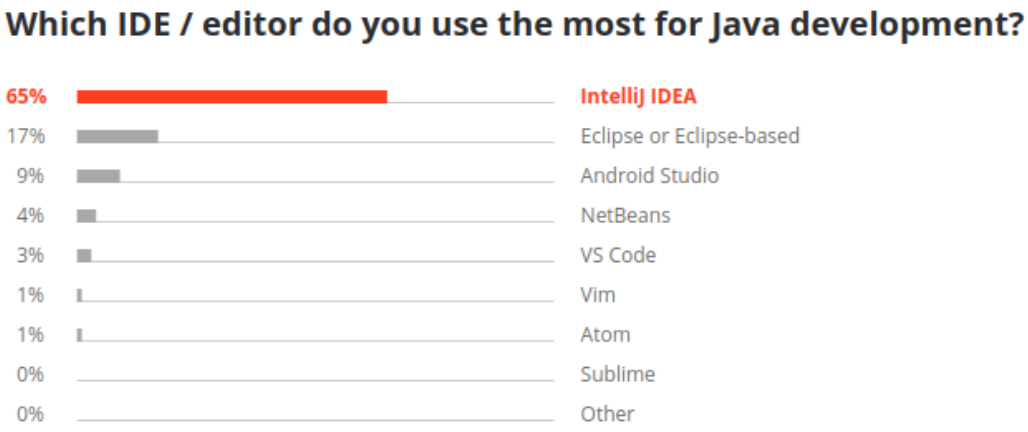


Figura 3-11.Entornos de desarrollo más usados para desarrollo java sobre el total de usuarios.

4 DISEÑO

Diseño es donde la ciencia y el arte llegan a un punto de equilibrio.

Robin Mathew

En este capítulo se procede a describir la forma en la que se ha diseñado el proyecto para su posterior implementación.

La forma de realizar el diseño de este proyecto ha estado claramente sujeta a dos factores: una serie de consideraciones iniciales y la tecnología escogida.

Las consideraciones iniciales son una serie de simplificaciones y supuestos sobre los subobjetivos de cumplimiento legal y de implementación que condicionan la forma de interactuar entre del usuario con la herramienta de exportación de datos y las funcionalidades que esta ofrece.

Por otro lado, el escoger una tecnología concreta como base para la implementación de un proyecto no debería, o al menos no de una forma tan acentuada, influir en el diseño previo de este, pero en este caso no es así.

Esto es debido a **que Spring MVC (*model-view-controller*) trae asociado, como su nombre indica, el patrón de diseño Modelo-Vista-Controlador además de una separación por capas para el acceso y manipulación de los datos del modelo** que unidos a los conceptos del ámbito del desarrollo web determinan de forma sustancial la estructura del diseño de este proyecto.

Una vez dicho esto, en las siguientes secciones se procede a detallar en primer lugar las consideraciones iniciales, posteriormente una serie de conceptos del contexto de la aplicación web para finalmente terminar detallando el diseño siguiendo la ya citada estructura de patrón modelo-vista-controlar unido al modelo de capas para el acceso y manipulación de los datos del modelo.

4.1 Consideraciones iniciales

Para definir qué funcionalidades y como serán ofrecidas al usuario se emplearán las siguientes consideraciones y simplificaciones:

En primer lugar, se establece que la **protección de los datos** con los que se desea trabajar **no es tarea a realizar por la herramient**a de exportación de datos sino por el responsable legal del tratamiento.

Se presupone además que el responsable hará un uso lícito de la herramienta, esto es, aplicará correctamente a través de ella la clasificación adecuada según lo establecido por el Reglamento General de Protección de Datos y solo manipulará a través de la aplicación datos para los cuales tenga consentimiento por parte del interesado.

Se definirán funciones de seudonimización/anonimización para los tipos de datos en los cuales se hace especial hincapié a lo largo de las predisposiciones y el articulado del reglamento. Estos serán: raza/etnia, orientación política, creencias religiosas, afiliación/es sindical/es, estado de salud, condición genética, orientación sexual, nombre, apellidos, identificación e identificaciones en línea, teléfonos tanto fijos como móviles y cuentas bancarias. Para los tipos de datos que no apliquen a ninguna de esas categorías se definirá una regla de seudonimización genérica.

Por último, aunque la aplicación generará un registro de las actividades de tratamiento de datos con el contenido obligatorio según el reglamento, **será tarea del responsable de datos hacer constar dicho registro por escrito y darle disponibilidad en formato electrónico para la autoridad de control que lo solicite**.

4.2 Contexto de la Aplicación Web

En las aplicaciones basadas en Spring los objetos que pertenecen a ella tienen su ciclo de vida en un contenedor de objetos encargado de crear dichos objetos, asociaciones entre ellos y gestionar el ciclo de vida de estos. A dichos objetos en el contexto de Spring se les denomina Spring managed beans, o simplemente beans, y al contenedor se le denomina Contexto de la Aplicación (Application Context).

El contexto de la aplicación utiliza **inyección de dependencias** para gestionar los beans que componen la aplicación. Además, estos contenedores propios de Spring (org.springframework.context.ApplicationContext) se encargan de crear, asociar y dispensar dichos **beans** bajo petición. En este caso nuestro procedimiento concreto para la **configuración de estos se realizará a través de clases Java**.

El contexto de una aplicación web como la nuestra está diseñado para trabajar con el contexto estándar de servlets (javax.servlet.ServletContext) y habitualmente contiene beans del front end como vistas y resolvedores de vistas (views & view resolvers). En Spring MVC, con el fin de ejecutar una aplicación web de forma exitosa, siempre será necesaria la existencia de un view resolver (org.springframework.web.servlet.ViewResolver). (Ganeshan, 2016)

Este y otros beans necesarios que usaremos en la aplicación web será definidos en un fichero de configuración Java cuyo nombre será WebApplicationContextConfig.

4.3 Patrón Modelo-Vista-Controlador

El otro **patrón Modelo-Vista-Controlador** consta de tres partes principales las cuales le dan nombre: el modelo, la vista y el controlador. El modelo se define como la parte encargada de manejar los datos, la vista por su parte se encarga de crear las interfaces de usuario y las distintas pantallas de la aplicación y por último el controlador es la parte que maneja las interacciones entre el usuario, la interfaz de usuario y los datos del modelo.

Un esquema del flujo de eventos y ordenes habituales de una aplicación diseñada bajo este patrón sería el mostrado a continuación:

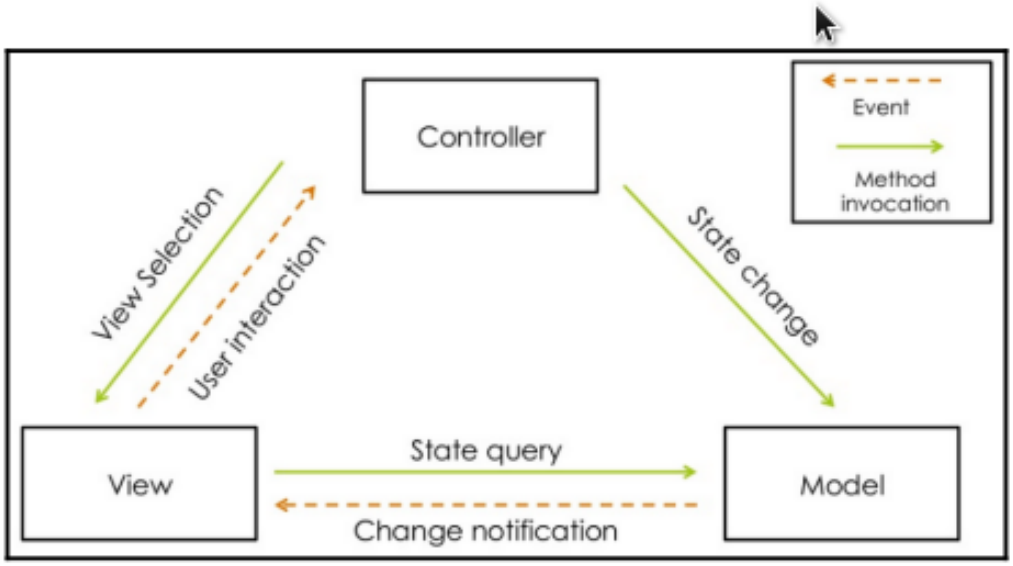


Figura 4-1.Interacciones del patrón MVC.

El uso patrón Modelo-Vista-Controlador "clásico" permiten la interacción entre el modelo y la vista de forma directa. Spring implementa lo que se denomina el patrón Modelo-Vista-Controlador Web, el cual no permite esta interacción debido a las limitaciones del protocolo HTTP sobre el que se sustenta.

Bajo este patrón MVC web, el punto de entrada principal para una petición web es a través del que se denomina *Dispatcher Servlet*. Este actúa como *front controller* y su función es encontrar el controlador adecuado para el posterior procesamiento de la petición.

El **flujo de peticiones** en una aplicación de Spring MVC sería el siguiente:

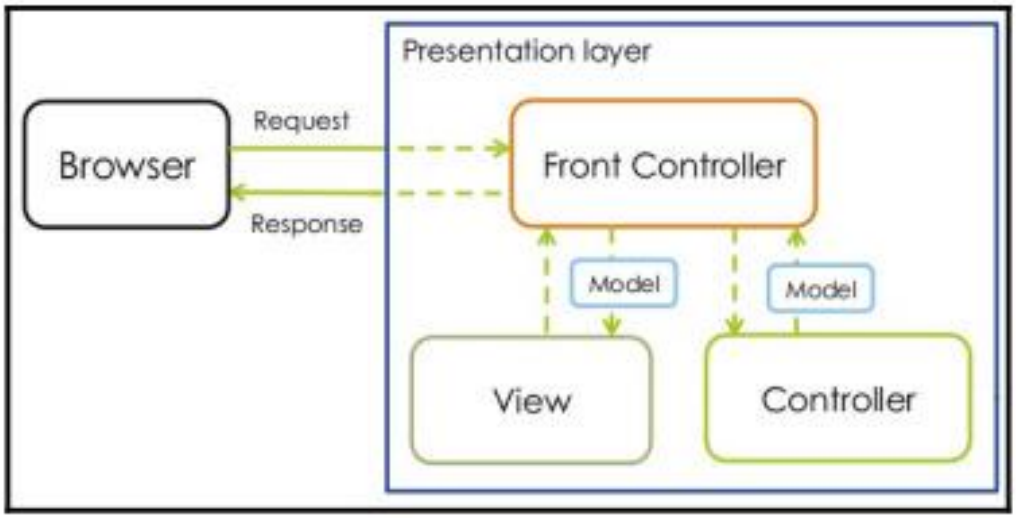


Figura 4-2.Flujo de peticiones en una aplicación Spring MVC.

Cuando introducimos una petición, esta es recibida por el *Dispatcher Servlet* que actúa como punto centralizado de entrada a la aplicación. El ***Dispatcher Servlet* al recibir la petición determina si hay algún controlador adecuado para manejar la petición y se la despacha**.

El **controlador** seleccionado al recibir la petición ejecuta un método que **actualiza los objetos del modelo y devuelve el nombre la vista al *Dispatcher Servlet***. Tras esto el ***Dispatcher Servlet* consulta con el *View Resolver* para determinar que vista renderizar y enviarle los datos del modelo**.

La vista rellena los valores dinámicos de la página web usando los datos del modelo y renderiza la página web definitiva la cual envía al *Dispatcher Servlet*. Tras esto finalmente el *Dispatcher Servlet* devuelve la página renderizada al navegador en forma de respuesta HTTP.

Como ya hemos explicado antes Spring MVC también trae asociado un modelo de capas para el desarrollo de aplicaciones empresariales que dividirá la aplicación en las siguientes capas: Presentación, Dominio de Aplicación, Persistencia y Servicio. (Ganeshan, 2016)

Estas capas serán explicadas a continuación como forma de estructurar el diseño.

4.4 Capa de Presentación

La capa de presentación engloba los elementos tales como el *Dispatcher Servlet*, los *View Resolver*, las vistas y los controladores.

Definiremos controladores para lo siguiente:

1. HomeController para el inicio de la aplicación.
2. LoginController para la gestión del sistema de inicio de sesión.
3. DatabaseController
4. TablesController
5. ColumnController
6. DataController

Estos cuatro últimos controladores serán los asociados a los objetos de la capa del dominio de aplicación que serán explicados en el próximo apartado.

4.5 Capa de Dominio de Aplicación

El dominio de aplicación es el asociado con el modelo dentro del patrón MVC. Así pues, en esta capa encontraremos **representaciones de los tipos de datos que se requieren almacenar para implementar la lógica de negocio**. Este dominio también abarca la descripción de las entidades, sus atributos, roles, relaciones y restricciones que gobiernan el problema del dominio.

La primera clase que definiremos será la **clase *Database*** la cual representará una instancia de base de datos dentro del sistema de gestión de bases de datos MariaDB. Dicha clase constará de un nombre y contendrá dentro de sí una serie de tablas.

Estas tablas a su vez serán otro participante de nuestro modelo de negocio y como tal será modelada por la **clase *Table*** la cual constará de un nombre y tendrá dentro de sí una serie de columnas.

Dichas columnas serán representadas en nuestra aplicación por la **clase *Column***. Esta clase contendrá, entre otras informaciones, el nombre de la columna, el tipo de dato con el que se almacena en la base de datos, si puede contener valores nulos, si es clave de cualquier tipo o no y si tiene valor por defecto.

Para nosotros cada columna representará un posible dato con el cual el usuario podría desear trabajar. Para ello definiremos la **clase *Data*** que contendrá toda la información necesaria para cumplir las restricciones que dicta el Reglamento General de Protección de Datos de la Unión Europea. Esta información será: el nombre de la tabla a la que pertenece, un identificador (el nombre de la columna de la que nace), el tipo con el que se almacena en la base de datos, el tipo aplicable según la clasificación extraída del reglamento, si tiene carácter personal y/o sensible, si se tiene consentimiento para el tratamiento en caso de que sea sensible, si se encuentra

anonimizado y el tratamiento que se le ha realizado incluida la fecha de este.

La relación entre estas clases se muestra a continuación:

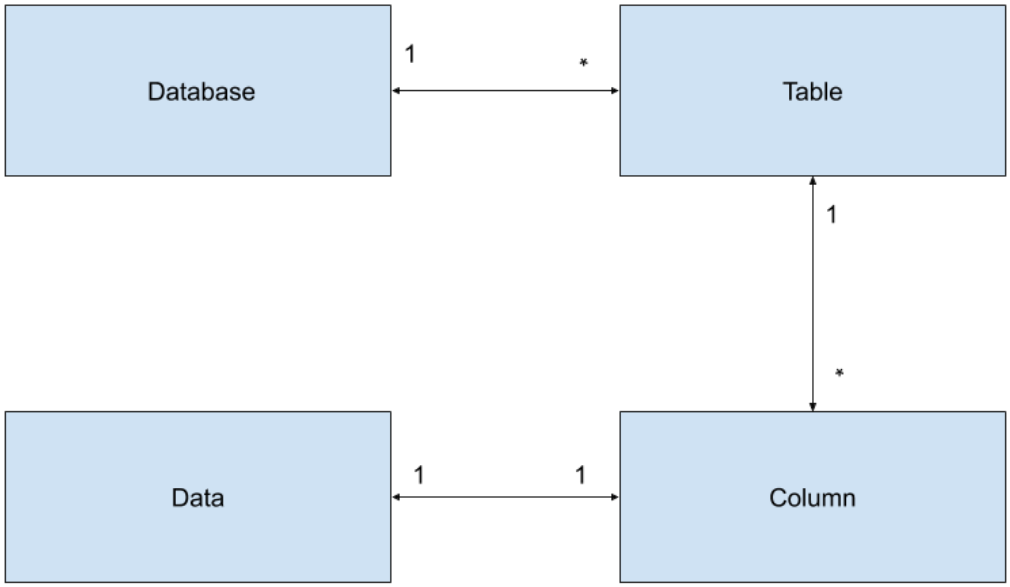


Figura 4-3. Relación de las clases a implementar.

Por ultimo necesitamos modelar dos herramientas: una que seudonimice/anonimice los datos (clase *Data*) y otra que se encargue de generar el registro de actividades. Estas clases serán *AnonymizationTool* y *PrintTool*.

4.6 Capa de Persistencia

La capa de persistencia, también conocida como de repositorio, se sitúa justo por encima de la capa del dominio de aplicación y es la encargada de ofrecer objetos de repositorio para el acceder a objetos del dominio de la aplicación.

Los objetos de repositorio **se encargan de ejecutar peticiones a la fuente de datos (base de datos), mapear la información obtenida como respuesta a un objeto del dominio y dar persistencia a los cambios realizados. Es decir, es responsable de las operaciones CRUD sobre objetos del dominio de aplicación**.

Estos objetos de repositorios se definen mediante una clase Java y la anotación de Spring *@Repository*. (Ganeshan, 2016)

Una vez explicado esto, se deduce que lo mejor para el diseño de la aplicación será que exista un tipo de objeto de repositorio por cada uno de los objetos del dominio obtenidos de la base de datos. Estos son *Database, Table, Column* y *Data*.

Para la posterior implementación de las clases que darán lugar a los objetos de repositorios **definiremos de forma previa una serie de interfaces**. Estas interfaces nos permitirán por un lado definir los moldes a adoptar por las clases que modelen los objetos de repositorio, y **para ofrecer una forma de acceso a esta capa, no dependiente de la implementación**, para la capa de servicio que se situará justo por encima como se explicará en el próximo apartado.

Las interfaces tomarán como nombre el de la clase de la capa del dominio de aplicación a la que darán acceso, seguido de *Repository*. Las clases que implementen estás interfaces tomaran el mismo nombre añadiendo al

final el sufijo *Impl*.

Dicho esto, se detalla a continuación la definición de dichas interfaces:

DatabaseRepository: permitirá leer las bases de datos disponibles, añadir una base de datos nuevas a la que exportar los datos y cambiar a una base de datos (Comando SQL "*USE database_name*").

```
package com.tfg.xportData.domain.repository;
import java.util.List;
import com.tfg.xportData.domain.Database;
public interface DatabaseRepository {
      List <Database> getAllDatabases();
      void addDatabaseByName(String name);
      void changeToDatabaseByName(String name);
}
```

TablesRepository: se usará para extraer todas las tablas disponibles dentro de una base de datos y copiar una tabla completa, tanto en esquema como en valores, a la base de datos de exportación.

```
package com.tfg.xportData.domain.repository;
import java.util.List;
            import com.tfg.xportData.domain.Table;
public interface TablesRepository {
       List <Table> getAllTables(String dbname);
       void copyTableByName(String tablename, String newDb);
}
```

ColumnRepository: su utilidad será exclusivamente la de obtener todas las columnas de una base de datos.

```
package com.tfg.xportData.domain.repository;
import java.util.List;
import com.tfg.xportData.domain.Column;
public interface ColumnRepository {
      List <Column> getAllColumns(String tablename);
}
```

DataRepository: Permitirá llevar a cabo el proceso de seudonimización/anonimización de los datos.

```
package com.tfg.xportData.domain.repository;
import com.tfg.xportData.domain.Data;
public interface DataRepository {
void anonimyze(Data data, String newDb);
}
```

4.7 Capa de Servicio

La capa de Servicio supone el nexo entre la capa de presentación y la capa de persistencia o repositorio, permitiendo así a los controladores mediante una sola llamada realizar operaciones del negocio, pudiendo estas estar compuestas de múltiples operaciones CRUD.

Dichas clases java podrán ser definidas mediante la anotación *@Service* (Ganeshan, 2016).

De forma análoga a lo realizado con la capa de persistencia definiremos una clase de servicio por cada clase de persistencia existente, y para moldear estás **definiremos una serie de interfaces**.

La nomenclatura para nombrar a las clases e interfaces continuará siguiendo el mismo esquema, pero en este caso sustituyendo el sufijo *Repository* por *Service*. Se detallan a continuación las distintas interfaces:

DatabaseService:

```
package com.tfg.xportData.service;
import java.util.List;
import com.tfg.xportData.domain.Database;
public interface DatabaseService {
      List <Database> getAllDatabases();
      public void addDatabaseByName(String name);
      public void changeToDatabaseByName(String name);
}
```

TablesService:

```
package com.tfg.xportData.service;
import java.util.List;
import com.tfg.xportData.domain.Table;
public interface TablesService {
      List <Table> getAllTables(String dbname);
      void copyTableByName(String tablename, String newDb);
}
```

ColumnService:

```
package com.tfg.xportData.service;
import java.util.List;
import com.tfg.xportData.domain.Column;
public interface ColumnService {
      List <Column> getAllColumns(String tablename);
}
```

DataService:

```
package com.tfg.xportData.service;
```

```
import com.tfg.xportData.domain.Data;
public interface DataService {
      void anonimyze(Data data, String newDb);
}
```

De esta forma es como terminamos de interconectar todas las capas y elementos para constituir de forma definitiva la que será la estructura de nuestra aplicación.

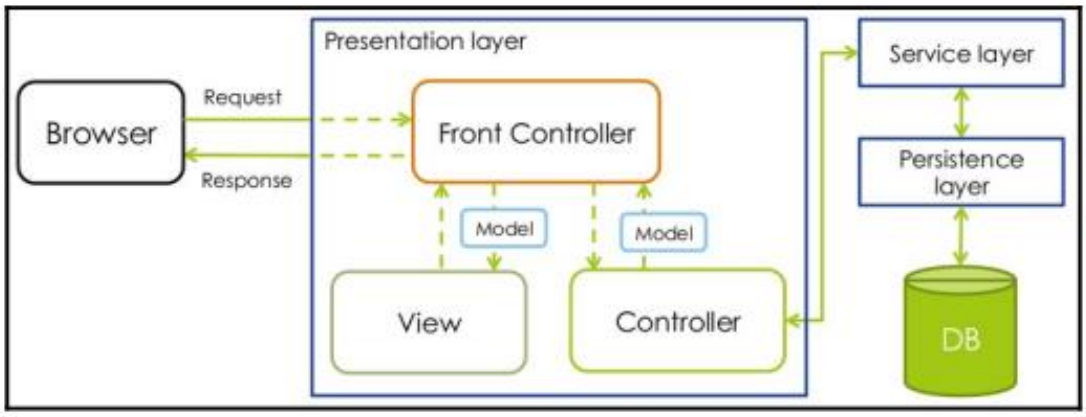


Figura 4-4. Relación de las clases a implementar.

5 IMPLEMENTACIÓN

En este capítulo se procederá a describir los pasos llevados a cabo para implementar el diseño descrito en el capítulo anterior y se mostrará el resultado en forma de código.

La estructura de subapartados para realizar esto es muy similar a la del apartado anterior para poder así mostrar al lectro de una forma clara como se plasman los diferentes aspectos tenidos en cuenta a la hora de realizar el diseño.

Pero antes de esto será necesario tener en cuenta cierta consideración inicial y mostrar como se constituye el proyecto a través de la herramienta STS.

5.1. Consideración inicial

Debido a la forma de proceder de Spring JDBC es necesario a la hora de leer una columna conocer el tipo en que está almacenado en la base de datos y tener un objeto del dominio de la aplicación que modele instancias de dicha columna que se está leyendo.

Como la lectura de tablas que realiza nuestra aplicación es genérica es imposible saber de antemano que columnas de que tablas van a ser leídas, lo cual impide a su vez tener definidas de antemano clases que modelen estas columnas que se están leyendo.

Antes esto se impone una restricción en el uso de la aplicación consistente en que **aquellos datos con los cuales se desee trabajar deben estar almacenados en la base de datos como *VARCHAR***. Con esto y no permitiendo que anonimicen datos no almacenados como *VARCHAR* conseguimos saber siempre de antemano el tipo de dato que vamos a leer y siempre lo mapearemos a un String de Java.

Esta simplificación **apenas afecta a la utilidad de la aplicación ya que la mayoría de los datos citados específicamente en el RGPD de la UE, aunque quizás existan otro tipo de datos más adecuados, pueden ser almacenados como *VARCHAR*** (raza/etnia, orientación política, creencias religiosas, afiliación/es sindical/es, estado de salud, condición genética, orientación sexual, nombre, apellidos, identificación e identificaciones en línea, teléfonos tanto fijos como móviles y cuentas bancarias).

5.2. Construcción del proyecto

Para llevar a cabo la implementación del proyecto será necesario, en primer lugar, crear el proyecto, darle la estructura de directorios de una aplicación web e indicar las dependencias. En esta sección es donde se detallan los pasos para llevar dicho procedimiento a cabo.

5.2.1 Inicio del Proyecto

En primer lugar, debemos arrancar el programa *Spring Tool Suite*. Una vez arrancado nos aparecerá una pantalla similar a esta

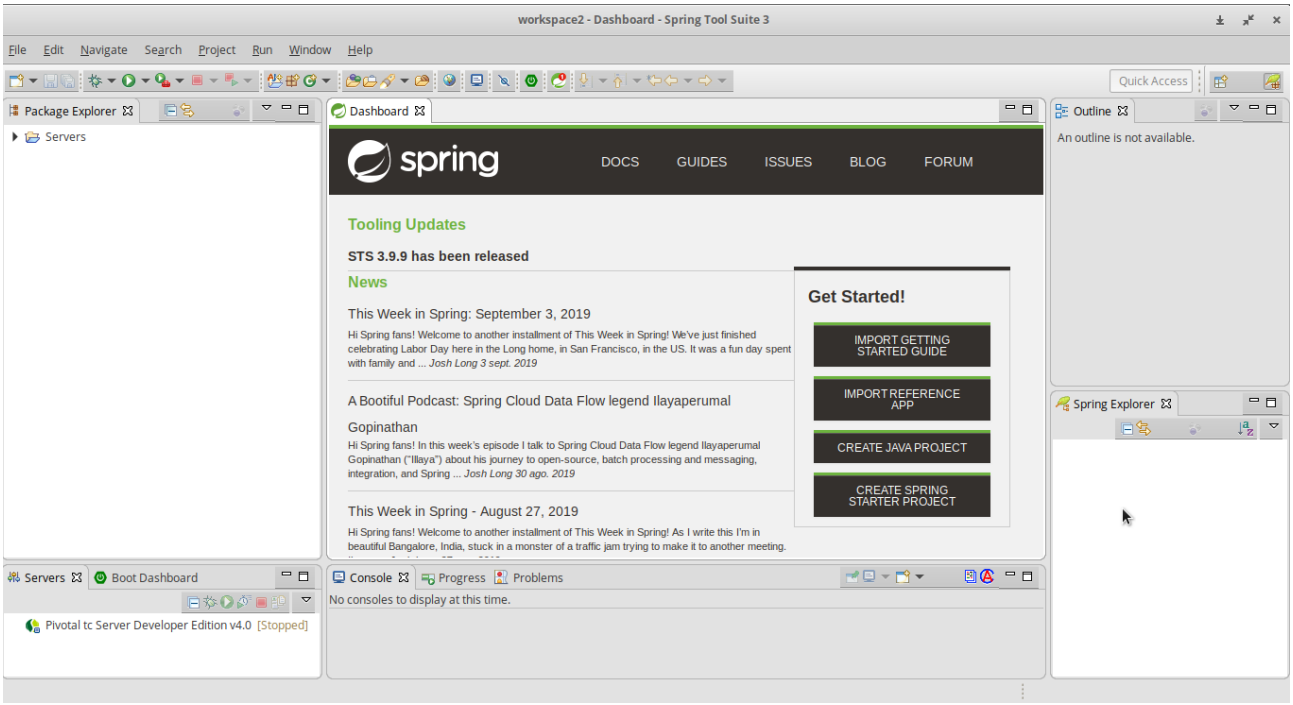


Figura 5-1. Pantalla de arranque STS.

Tras esto debemos seleccionar *File* > *New* > *Project* y nos aparecerá la siguiente ventana donde habrá que seleccionar *Maven Project* y pulsar *Next*

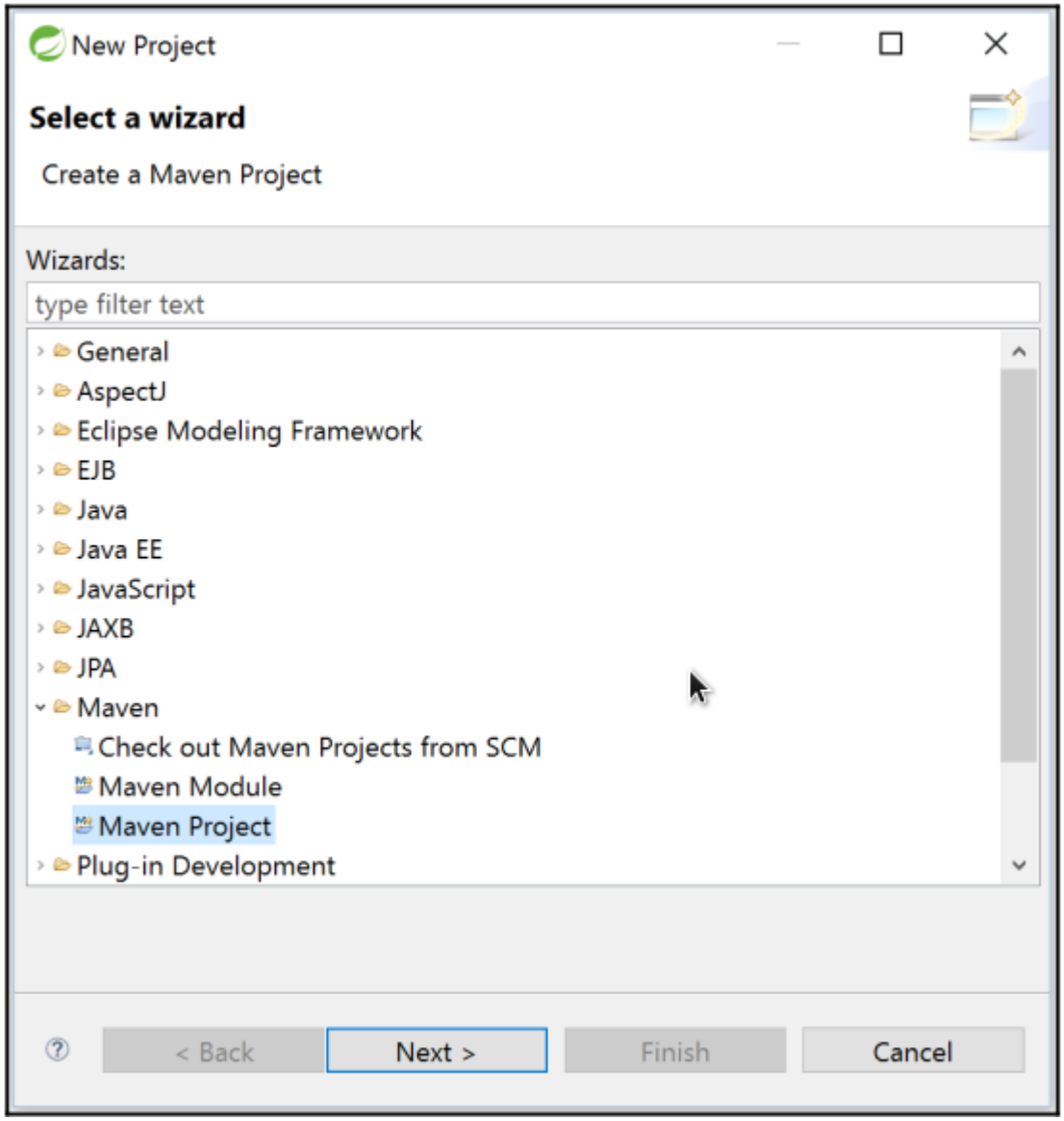


Figura 5-2. Menú contextual de creación de proyecto.

En el posterior menú contextual pulsaremos *Create a simple project (skip archetype selection)* y pulsar *Next*

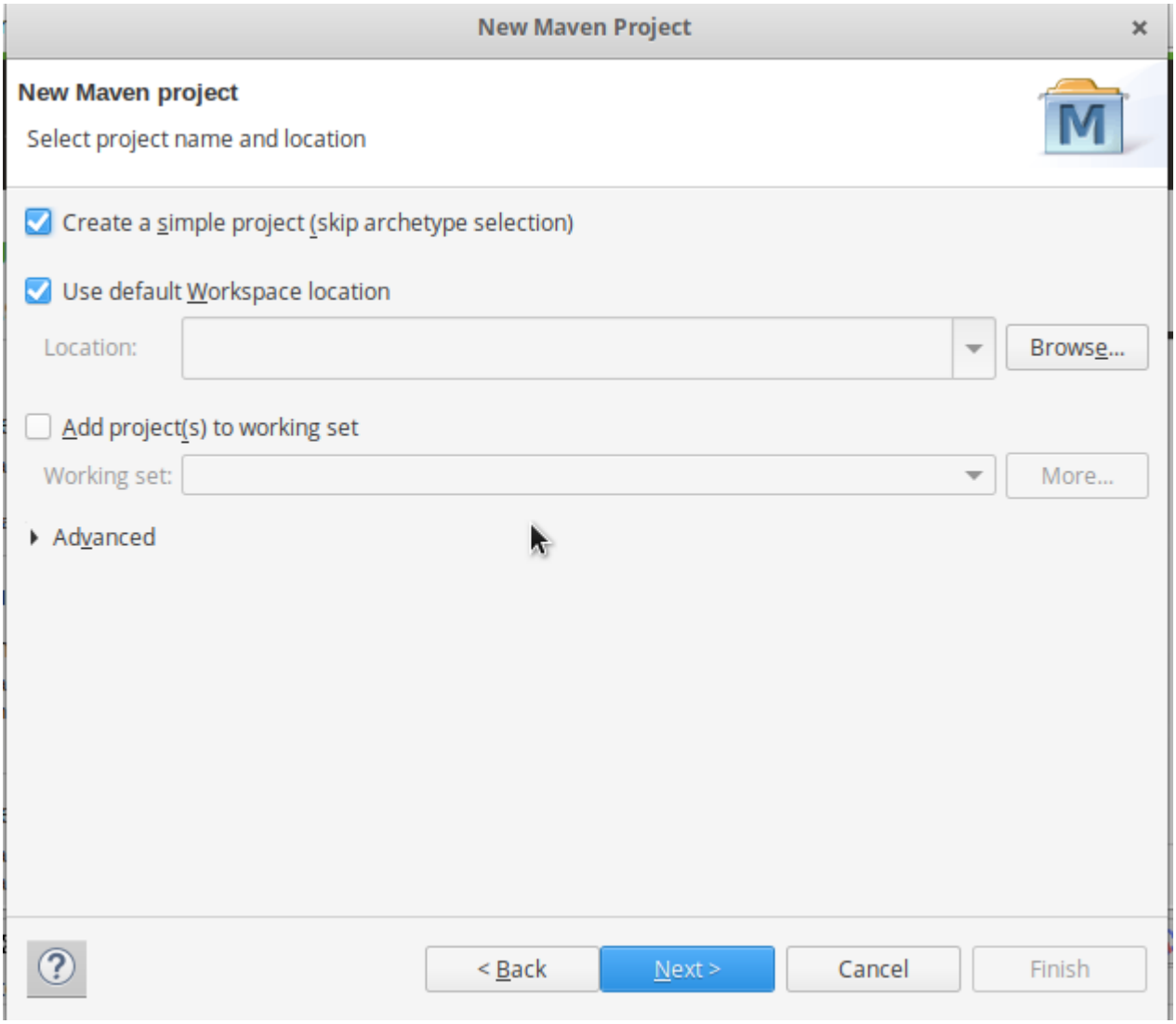


Figura 5-3. Menú de arquitectura de proyecto Maven.

Tras esto nos aparecerá el último menú contextual que deberemos rellenar de la siguiente manera y tras esto, pulsar *Finish*.

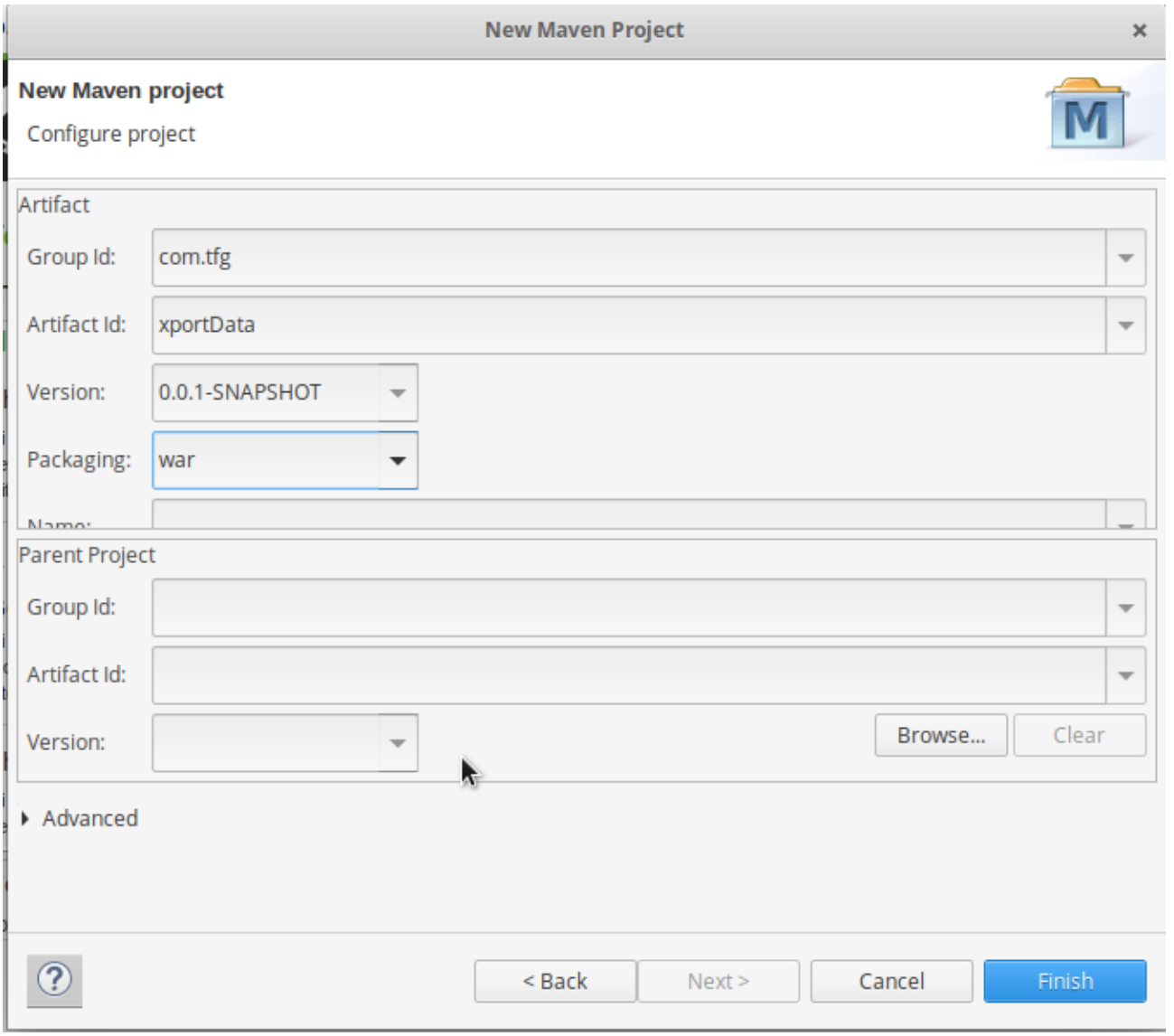


Figura 5-4. Menú final de proyecto Maven.

Una vez realizado esto se nos generará parte de la estructura de directorios que será explicada más adelante.

Lo siguiente a realizar es establecer Java 8 como la versión a utilizar por Maven en la compilación y construcción de nuestro proyecto. Para ello abriremos el fichero *pom.xml*, seleccionaremos la pestaña de *Overview*, expandiremos el cuadro de *Properties* y pulsaremos *Create*.

Después rellenaremos tal y como se muestra en la segunda imagen, pulsaremos OK y volveremos a rellenar de nuevo cambiando *maven.compiler.source* por *maven.compiler.target*. Tras esto debemos guardar los cambios realizados en el archivo *pom.xml*.

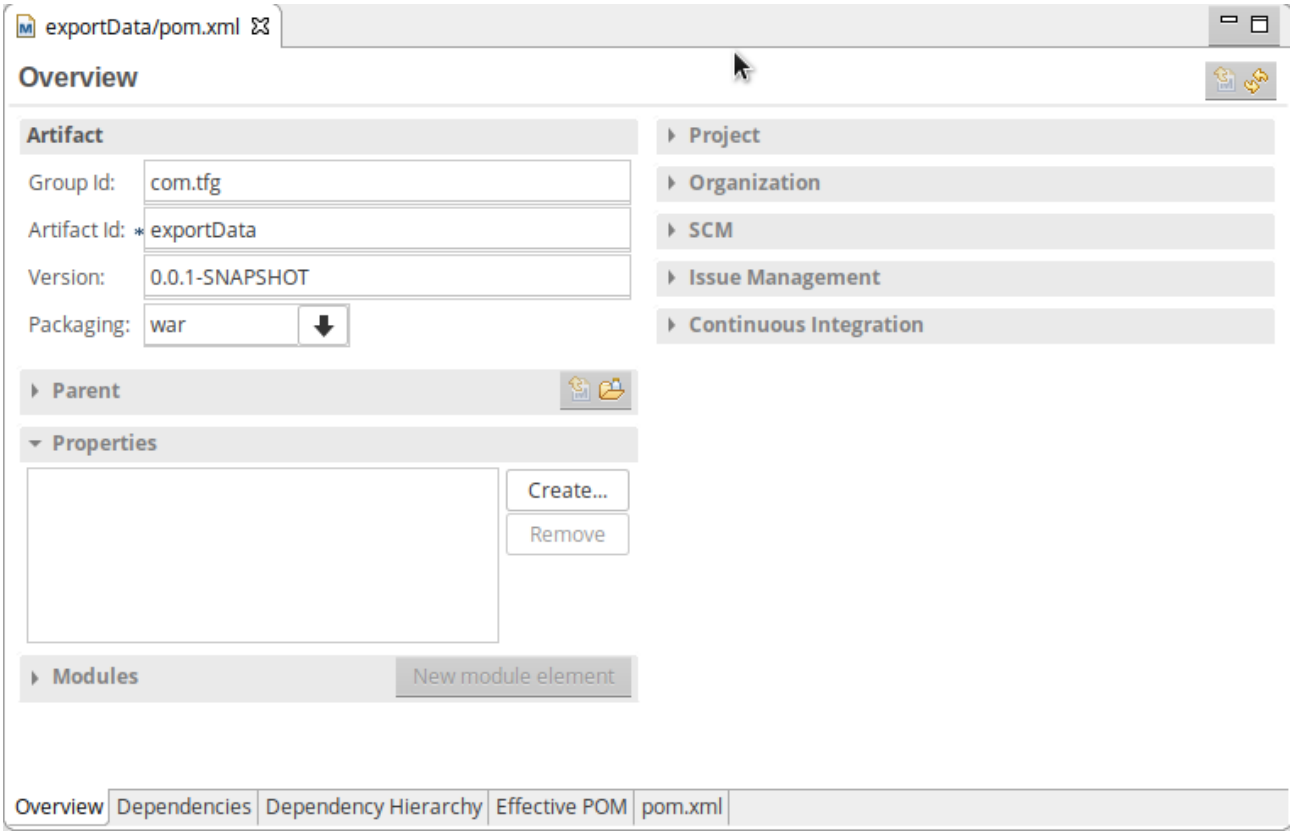


Figura 5-5. Menú Overview del archivo pom.xml.

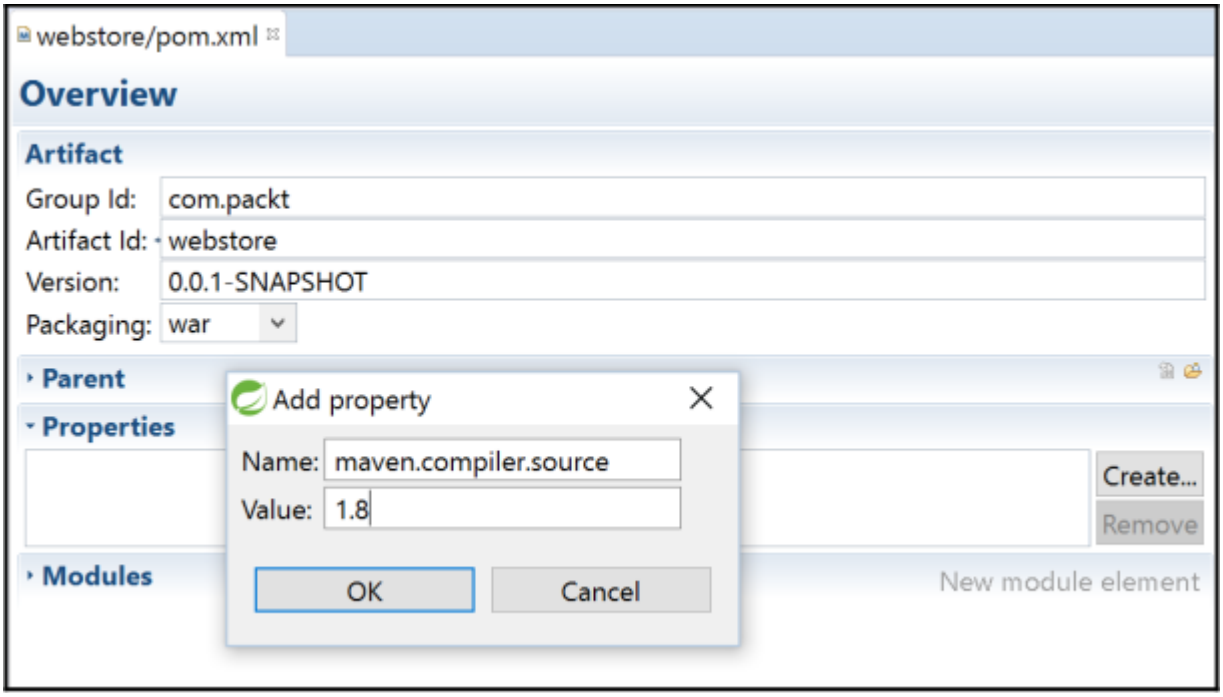


Figura 5-6. Menú contextual de adición de propiedades.

En este momento STS probablemente nos muestre un error indicándonos que no existe el archivo *web.xml*. Para solucionar esto añadiremos el siguiente bloque al archivo *pom.xml* dentro del *tag* *<project>*:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.6</version>
            <configuration>
                <failOnMissingWebXml>false</failOnMissingWebXml>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Figura 5-7. Bloque de código arreglar error pom.xml.

Por último, ya solo nos quedará añadir las dependencias que tenga nuestro proyecto. Para esto abriremos el archivo *pom.xml*, seleccionaremos la pestaña dependencies y pulsaremos el botón *Add* de la sección *Dependencies Management*.

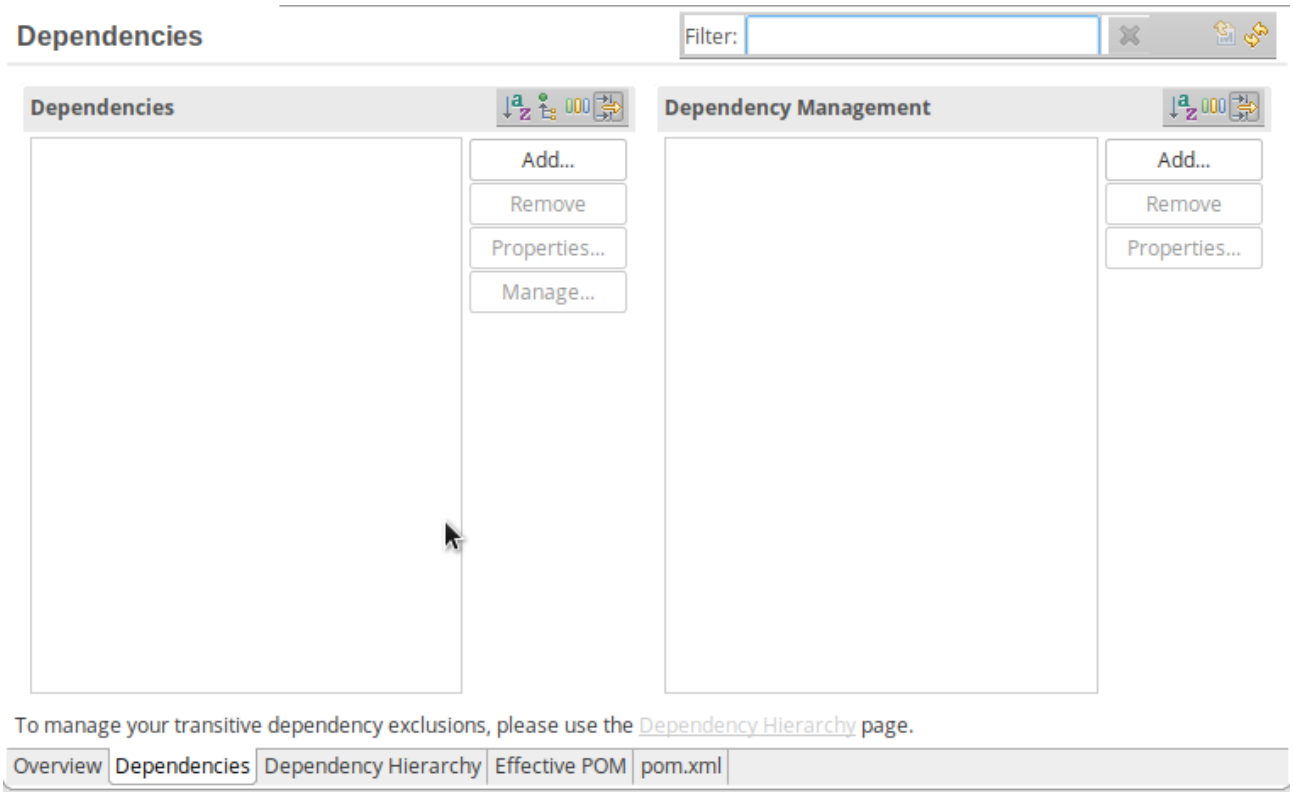


Figura 5-8. Menú de adición de dependencias.

Tras esto nos aparecerá el siguiente menú contextual el cual usaremos para añadir todas las dependencias de nuestro proyecto.

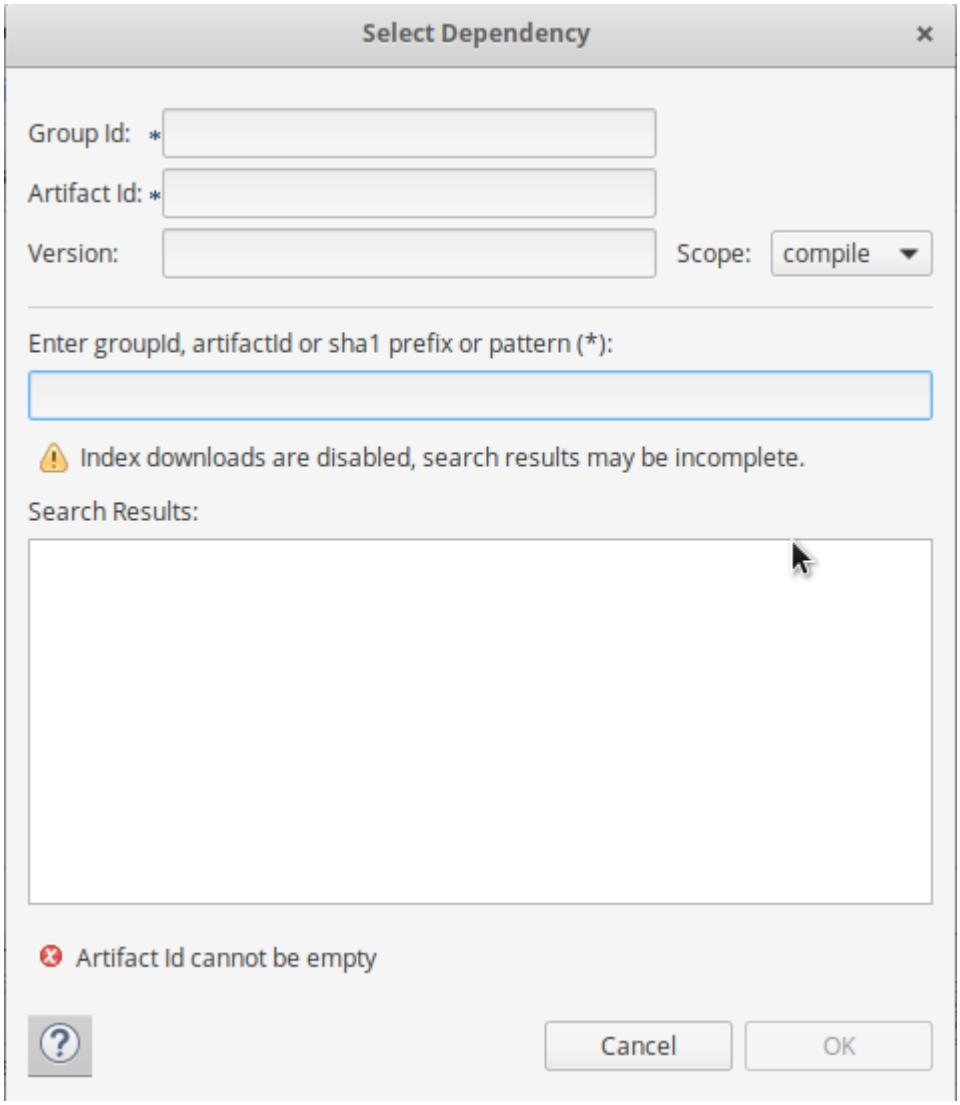


Figura 5-9. Menú contextual de adición de dependencias.

El único paso restante será añadir todas las dependencias asociadas a las tecnologías indicadas en el capítulo 3 que se decidieron que se iban a usar hasta que las dependencias queden de la siguiente forma:

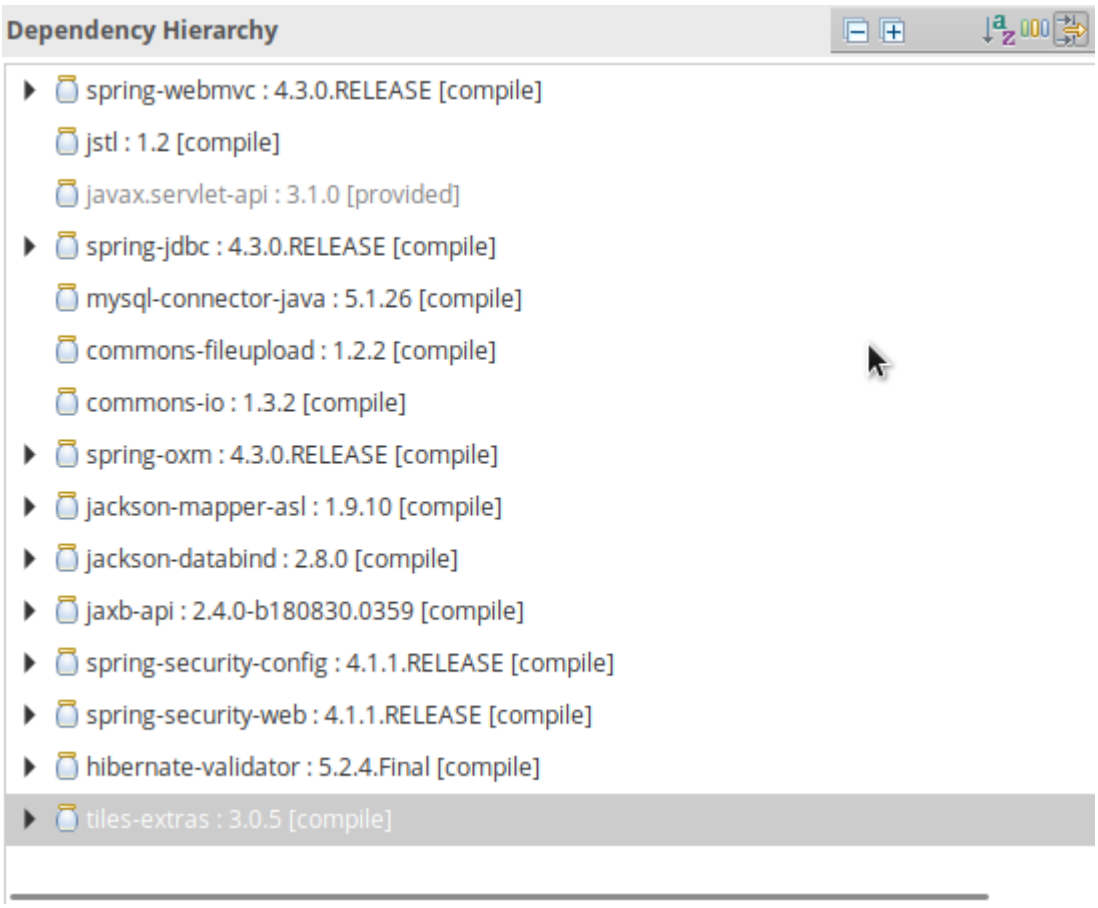


Figura 5-10. Lista final de dependencias para nuestra aplicación.

5.2.2 Estructura de Subdirectorios

Una vez realizado todo esto nos encontraremos con la siguiente estructura de directorios:

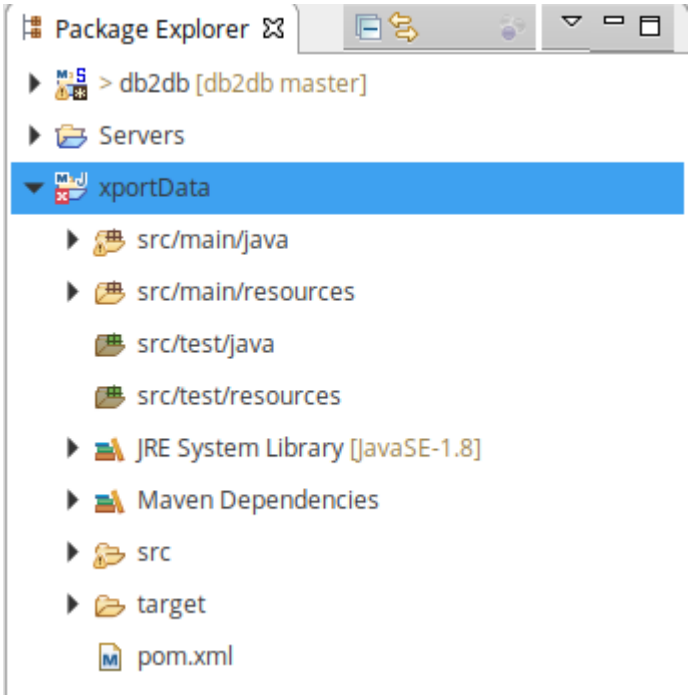


Figura 5-11. Estructura de directorios.

Si desplegamos las pestañas *JRE System Library* y *Maven Dependencies* podremos observar los archivos .jar de las librerías y las dependencias, respectivamente, añadidas a través del archivo *pom.xml*.

Si desplegamos la pestaña *src* > *main* > *webapps* > *WEB-INF* encontraremos las carpetas *layouts* y *views* en las cuales se almacena el front end de la aplicación.

La carpeta *layouts* contiene a su vez los directorios *definitions* y *templates*, en los cuales se situan los archivos necesarios para dar una forma visual común a la aplicación web a través de la utilización de *Apache Tiles*.

Por otro lado, en la carpeta *views* se encuentran los archivos *jsp* o vistas que serán renderizas y devueltas como páginas web al navegador.

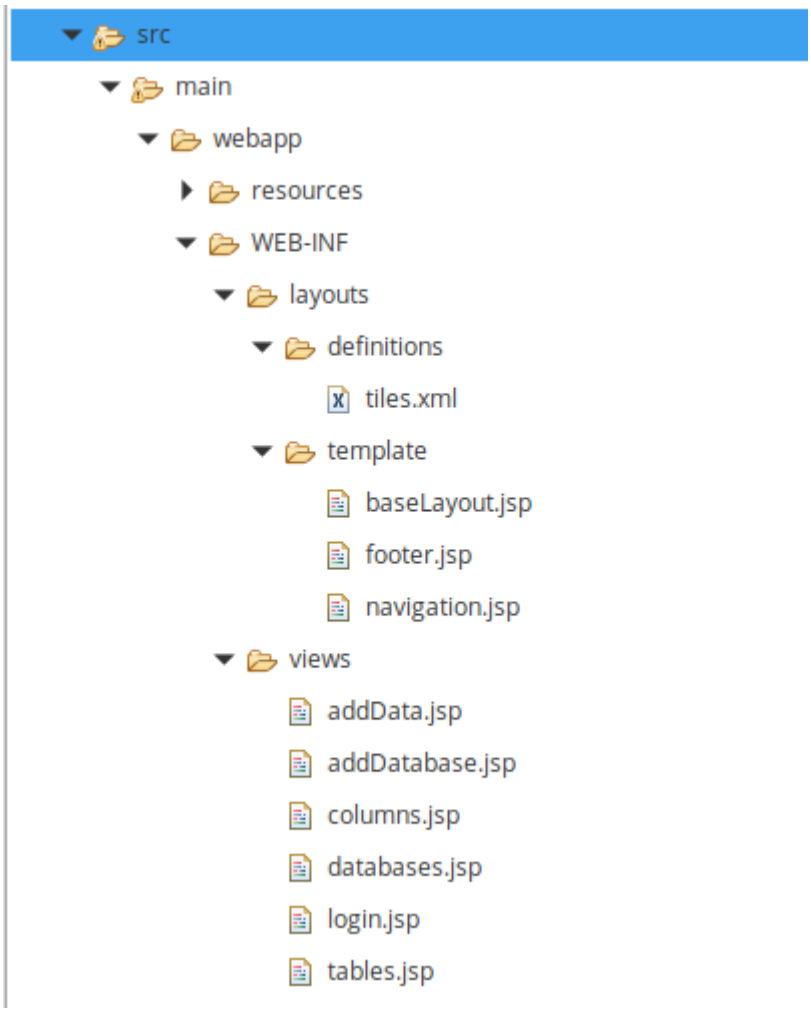


Figura 5-12. Directorio src.

Por otro lado, en la carpeta src/main/java encontraremos el *back end* de la aplicación estructurado según el diseño en los *packages* que se muestran a continuación:

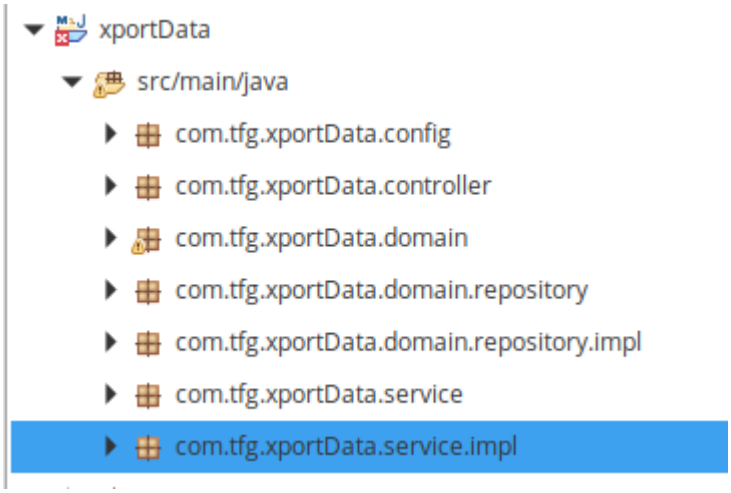


Figura 5-13. Directorio src/main/java.

El único paquete que no está relacionado con el diseño es *com.tfg.exportData.config* el cual contiene las clases java con las que Maven construye la aplicación web. A continuación, en la siguiente subsección se muestran los ficheros, su contenido y se explican sus funcionalidades.

5.2.3 Configuración del Proyecto

En el paquete *com.tfg.exportData.config* se encuentran las siguientes clase java:

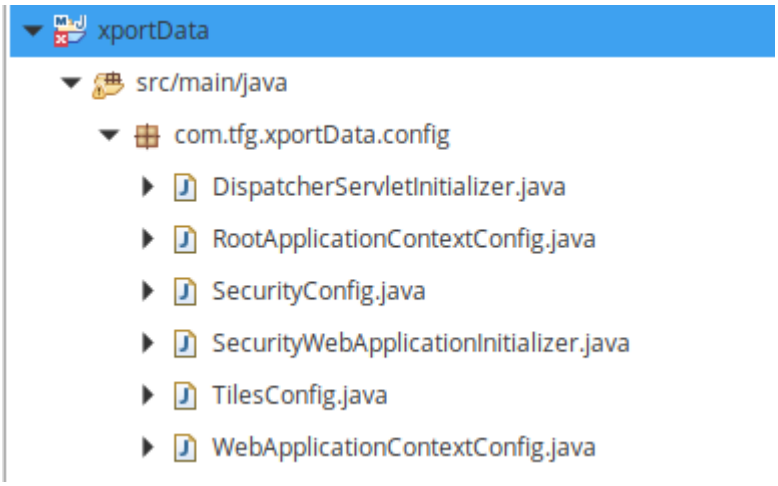


Figura 5-14. Paquete com.tfg.xportData.config .

El archivo *DispatcherServletInitializer.java* es el empleado para configurar el *DipatcherServlet* que se encargará de invocar a un u otro controlador en función de la url recibida, indicando las clases a través de las cuales definiremos el contexto de nuestra aplicación y que serán explicados justo después. Su contenido es el siguiente:

```
1  package com.tfg.xportData.config;
2
3  import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4
5  public class DispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
6      @Override
7      protected Class<?>[] getRootConfigClasses() {
8          return new Class[] { RootApplicationContextConfig.class};
9      }
10
11     @Override
12     protected Class<?>[] getServletConfigClasses() {
13         return new Class[] { WebApplicationContextConfig.class };
14     }
15
16     @Override
17     protected String[] getServletMappings() {
18         return new String[] { "/" };
19     }
20 }
21
```

El primero de los archivos de configuración citados es *RootApplicationContextConfig*, en el cual configuraremos la conexión con la base de datos y obtendremos un *template* para interaccionar con la fuente de datos. El código que define la clase es el siguiente:

```java
package com.tfg.xportData.config;

import javax.sql.DataSource;

@Configuration
@ComponentScan("com.tfg.xportData")
public class RootApplicationContextConfig {

    @Bean
    public DataSource mysqlDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/tfg");
        dataSource.setUsername("root");
        dataSource.setPassword("root");

        return dataSource;
    }

    @Bean
    public NamedParameterJdbcTemplate getJdbcTemplate() {
    return new NamedParameterJdbcTemplate(mysqlDataSource());
    }

}
```

El otro archivo de configuración será *WebApplicationContextConfig* y en el definiremos los prefijos y sufijos para encontrar las vistas asociadas a una petición web. El código se muestra a continuación:

WebApplicationContextConfig.java

```java
package com.tfg.xportData.config;

import org.springframework.context.annotation.Bean;


@Configuration
@EnableWebMvc
@ComponentScan("com.tfg.xportData")
public class WebApplicationContextConfig extends WebMvcConfigurerAdapter {
    @Override
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }

    @Override
    public void configurePathMatch(PathMatchConfigurer configurer) {
        UrlPathHelper urlPathHelper = new UrlPathHelper();
        urlPathHelper.setRemoveSemicolonContent(false);
        configurer.setUrlPathHelper(urlPathHelper);
    }

    @Bean
    public InternalResourceViewResolver getInternalResourceViewResolver() {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setViewClass(JstlView.class);
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
        return resolver;
    }
}
```

Además de los archivos ya explicados en el paquete se hallan las clases java de configuración de *Spring Security* y de *Apache Tiles*.

Para *Spring Security* serán necesarios dos archivos: *SecurityConfig.java* y *SecurityWebApplicationInitializer.java* .

El primero de ellos se emplea para definir los usuarios y contraseñas, las páginas web de login, cierre de sesión y acceso denegado y las rutas que requieran de permisos específicos.

```java
import org.springframework.beans.factory.annotation.Autowired;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    public void configureGlobalSecurity(AuthenticationManagerBuilder auth)throws Exception {
        auth.inMemoryAuthentication().withUser("user").password("user").roles("USER");
        auth.inMemoryAuthentication().withUser("admin").password("admin").roles("USER","ADMIN");
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin().loginPage("/login").usernameParameter("userId").passwordParameter("password");

        httpSecurity.formLogin().defaultSuccessUrl("/databases").failureUrl("/login?error");
        httpSecurity.logout().logoutSuccessUrl("/login?logout");
        httpSecurity.exceptionHandling().accessDeniedPage("/login?accessDenied");

        httpSecurity.authorizeRequests().antMatchers("/").permitAll().antMatchers("/**/tables/**/")
        .access("hasRole('ADMIN')").antMatchers("/**/databases/**/").access("hasRole('USER')");

        httpSecurity.csrf().disable();
        }
}
```

El segundo de ellos sirve para inicializar la configuración definida. La clase java creada para ello es muy sencilla.

```java
package com.tfg.xportData.config;

import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {

}

```

En último lugar para poder emplear *Apache Tiles* en nuestra aplicación debemos configurarlo a través de la siguiente clase *TilesConfig.java* de la siguiente manera:

.

TilesConfig.java

```java
package com.tfg.xportData.config;

import org.springframework.context.annotation.Bean;

@Configuration
public class TilesConfig {

    @Bean
    public UrlBasedViewResolver viewResolver() {
        UrlBasedViewResolver viewResolver = new UrlBasedViewResolver();
        viewResolver.setViewClass(TilesView.class);
        viewResolver.setOrder(-2);
        return viewResolver;
    }

    @Bean
    public TilesConfigurer tilesConfigurer() {
    TilesConfigurer tilesConfigurer = new TilesConfigurer();
    tilesConfigurer.setDefinitions("/WEB-INF/layouts/definitions/tiles.xml");
    tilesConfigurer.setCheckRefresh(true);
    return tilesConfigurer;
    }

}

```

Una vez implementada esta última clase de configuración podremos proceder a implementar el resto de clases y funcionalidades de nuestra herramienta de exportación de datos.

5.3. Capa de Visualización

En esta sección se procede a mostrar y comentar la implementación de los controladores definidos en el diseño.

Cada controlador constará de uno o varios métodos asociados a un patrón URL y/o un método de petición HTTP. Estos métodos devolverán al front controller el nombre de la vista a renderizar como ya se detalló al explicar el patrón MVC. Es por esto que tras detallar los controladores también se mostrará el código de las vistas.

5.3.1. Controladores

HomeController: Será el encargado de indicar la página de bienvenida, siendo esta la vista *login.jsp*

HomeController.java

```java
package com.tfg.xportData.controller;

import org.springframework.stereotype.Controller;

@Controller
public class HomeController {
    @RequestMapping("/")
    public String welcome(Model model) {
        return "login";
    }
}
```

LoginController: Responderá a las peticiones de login que no procedan de la ruta "/". Un ejemplo de esto será el cierre de sesión, la autenticación inválida o el acceso denegado.

LoginController.java

```java
package com.tfg.xportData.controller;

import org.springframework.stereotype.Controller;

@Controller
public class LoginController {

    @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String login() {
        return "login";
    }
}
```

DatabaseController: Será el encargado de mostrar las bases de datos disponible a través de *databases.jsp*. Cada una de las bases de datos mostradas constará de un botón que al pulsar redirigirá a la página de creación de la base de datos de exportación *addDatabase.jsp* la cual también será controlada por esta clase. Tras esto el usuario será renviado a una página que cambiará la base de datos que estaremos usando a la seleccionada (comando SQL "USE database") e inmediatamente será redirigido a *tables.jsp*.

DatabaseController.java

```java

@Controller
public class DatabaseController {

    @Autowired
    private DatabaseService databaseService;

    @RequestMapping(value="/databases", method=RequestMethod.GET)
    public String list(Model model) {
        model.addAttribute("databases", databaseService.getAllDatabases());
        return "databases";
    }

    @RequestMapping(value="/add/database/{name}", method=RequestMethod.GET)
    public String addDatabase(@MatrixVariable(pathVar="name") String name, Model model) {
        Database newDb = new Database();
        model.addAttribute("newDb", newDb);
        return "addDatabase";
    }

    @RequestMapping(value="/add/database/{name}", method=RequestMethod.POST)
    public String processNewDatabase(Model model,
            @ModelAttribute("newDb") Database newDb,
            @MatrixVariable(pathVar="name") String name) {
        databaseService.addDatabaseByName(newDb.getName());
        return "redirect:/database/old="+name+"/newDb="+newDb.getName();
    }

    @RequestMapping("/database/{old}/{newDb}")
    public String getDataseByName(@MatrixVariable(pathVar="old") String old,
            @MatrixVariable(pathVar="newDb") String newDb,Model model) {
        databaseService.changeToDatabaseByName(old);
        return "redirect:/tables/exp="+old+"/newDb="+newDb;
    }
}

```

TablesController: Mostrará las tablas disponibles dentro de la base de datos seleccionada a través de la vista *tables.jsp* y cuando el usuario seleccione una tabla la copiará a la base de datos de exportación y redirigirá al usuario a la vista *columns.jsp* gestionada por el siguiente controlador.

TablesController.java

```java
package com.tfg.xportData.controller;

import org.springframework.beans.factory.annotation.Autowired;

@Controller
public class TablesController {

    @Autowired
    private TablesService tablesService;

    @RequestMapping("/tables/{exp}/{newDb}")
    public String List(Model model,
            @MatrixVariable(pathVar="exp") String exp,
            @MatrixVariable(pathVar="newDb") String newDb ) {
        List <Table> tables = tablesService.getAllTables(exp);
        model.addAttribute("tables",tables);
        model.addAttribute("dbname", exp);
        model.addAttribute("newDb", newDb);
        return "tables";
    }

    @RequestMapping("/table/{name}/{db}/{newDb}")
    public String copyTable(Model model,
            @MatrixVariable(pathVar="name") String name,
            @MatrixVariable(pathVar="db") String db,
            @MatrixVariable(pathVar="newDb") String newDb){
        tablesService.copyTableByName(name, newDb);
        return "redirect:/columns/name="+name+"/db="+db+"/newDb="+newDb;
    }
}
```

ColumnController: Mostrará las columnas que pertenezcan a la tabla seleccionada a través de la vista *columns.jsp*. Cada columna mostrada contendrá un enlace dentro de un botón que redirigirá al formulario para la creación del dato del RGPD que permitirá la seudonimización/anonimización del mismo.

ColumnController.java

```java
package com.tfg.xportData.controller;

import org.springframework.beans.factory.annotation.Autowired;

@Controller
public class ColumnController {

    @Autowired
    ColumnService columnService;

    @RequestMapping("/columns/{name}/{db}/{newDb}")
    public String List(Model model,
            @MatrixVariable(pathVar="name") String name,
            @MatrixVariable(pathVar="db") String db,
            @MatrixVariable(pathVar="newDb") String newDb) {
        List <Column> columns = columnService.getAllColumns(name);
        model.addAttribute("columns", columns);
        model.addAttribute("name", name);
        model.addAttribute("db",db);
        model.addAttribute("newDb",newDb);
        return "columns";
    }
}
```

DataController: Gestionará el citado formulario y la anonimización del dato asociado a la columna seleccionada.

DataController.java

```java
package com.tfg.xportData.controller;

import org.springframework.beans.factory.annotation.Autowired;

@Controller
public class DataController {

    @Autowired
    private DataService dataService;

    @RequestMapping(value ="/data/add/{field}/{type}/{table}/{db}/{newDb}", method=RequestMethod.GET)
    public String getNewData(Model model,
            @MatrixVariable(pathVar="field") String field,
            @MatrixVariable(pathVar="type") String type,
            @MatrixVariable(pathVar="table") String table,
            @MatrixVariable(pathVar="db") String db,
            @MatrixVariable(pathVar="newDb") String newDb) {
        Data newData= new Data();
        newData.setId(field);
        newData.setTipoBbdd(type);
        newData.setTablename(table);
        model.addAttribute("newData", newData);
        return "addData";
    }

    @RequestMapping(value="/data/add/{field}/{type}/{table}/{db}/{newDb}", method=RequestMethod.POST)
    public String processNewData(Model model,
            @ModelAttribute("newData") Data newData,
            @MatrixVariable("db") String db,
            @MatrixVariable("newDb") String newDb) {
        dataService.anonimyze(newData, newDb);
        return "redirect:/columns/name="+newData.getTablename()+"/db="+db+"/newDb="+newDb;
    }
}

```

5.3.2. Vistas

Se detalla a continuación el código de las vistas nombradas en el apartado anterior. Su resultado visual se mostrará en el ejemplo de uso de la aplicación web del siguiente capítulo.

Login.jsp:

login.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
<title>Export Data Tool</title>
</head>
<body>
    <!--<section>
        <div class="jumbotron">
            <div class="container">
                <h1>Bienvenido</h1>
                <p>Herramienta de exportacion de datos</p>
            </div>
        </div>
    </section> -->
    <div class="container">
        <div class="row">
            <div class="col-md-4 col-md-offset-4">
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h3 class="panel-title">Log in</h3>
                    </div>
                    <div class="panel-body">
                        <c:url var="loginUrl" value="/login" />
                        <form action="${loginUrl}" method="post" class="form-horizontal">
                            <c:if test="${param.error != null}">
                                <div class="alert alert-danger">
                                    <p>Invalid username and password.</p>
                                </div>
                            </c:if>
                            <c:if test="${param.logout != null}">
                                <div class="alert alert-success">
                                    <p>You have been logged out successfully.</p>
                                </div>
                            </c:if>
                            <c:if test="${param.accessDenied != null}">
                                <div class="alert alert-danger">
                                    <p>Access Denied: You are not authorised!</p>
                                </div>
                            </c:if>
                            <div class="input-group input-sm">
                                <label class="input-group-addon" for="username"><i
                                    class="fa fa-user"></i></label> <input type="text" class="form-control"
                                    id="userId" name="userId" placeholder="Enter Username" required>
                            </div>
                            <div class="input-group input-sm">
                                <label class="input-group-addon" for="password"><i
                                    class="fa fa-lock"></i></label> <input type="password"
                                    class="form-control" id="password" name="password"
                                    placeholder="Enter Password" required>
                            </div>
                            <div class="form-actions">
                                <input type="submit"
                                    class="btn btn-block btn-primary btn-default" value="Log in">
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
```

databases.jsp:

databases.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet"
    href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
<title>Databases</title>
</head>
<body>
    <!--<section>
        <div class="pull-right" style="padding-right: 50px">
            <a href="<c:url value="/logout" />">Logout</a>
        </div>
    </section>
    <section>
        <div class="jumbotron">
            <div class="container">
                <h1>Databases available</h1>
                <p>Select the one you want to work with.</p>
                <p>Do not choose the exportation database.</p>
            </div>
        </div>
    </section>   -->
    <section class="container">
        <div class="row">
            <div class=caption>
                <c:forEach items="${databases}" var="database">
                    <div class="col-sm-6 col-md-3" style="padding-bottom: 15px">
                        <div class="thumbnail">
                            <div class="caption">
                                <h3>Database name: ${database.name}</h3>
                                <div>
                                    <p>
                                        <a
                                            href=" <spring:url value="add/database/name=${database.name}"
                                            /> "
                                            class="btn btn-primary"> Select <span
                                            class="glyphicon-oks glyphicon" /></span>
                                        </a>
                                    </p>
                                </div>
                            </div>
                        </div>
                    </div>
                </c:forEach>
            </div>
        </div>
    </section>
</body>
</html>
```

addDatabase.jsp:

addDatabase.jsp

```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet"
    href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
<title>Data</title>
</head>
<body>
    <!--  <section>
    <section class="container">
        <form:form method="POST" modelAttribute="newDb" class="form-horizontal">
            <form:errors path="*" cssClass="alert alert-danger" element="div" />
            <fieldset>
                <legend>Enter the name of the new database</legend>
                <div class="form-group">
                    <label class="control-label col-lg-2 col-lg-2" for="name">Database name</label>
                    <div class="col-lg-10">
                        <form:input  path="name" type="text" class="form:input-large" />
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-lg-offset-2 col-lg-10">
                        <input type="submit" id="btnAdd" class="btn btn-primary"
                            value="Create" />
                    </div>
                </div>
            </fieldset>
        </form:form>
    </section>
</body>
</html>
```

tables.jsp:

tables.jsp

```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet"
    href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
<title>Tables</title>
</head>
<body>
    <!--<section>
    <section class="container">
        <div class="row">
            <div class=caption>
                <c:forEach items="${tables}" var="table">
                    <div class="col-sm-6 col-md-3" style="padding-bottom: 15px">
                        <div class="thumbnail">
                            <div class="caption">
                                <h5>Table name: ${table.name}</h5>
                                <div>
                                    <p>
                                        <a href=" <spring:url
                                        value="/table/name=${table.name}/db=${dbname}/newDb=${newDb}" /> "
                                        class="btn btn-primary">
                                        Export from this table
                                        <span class="glyphicon-ok glyphicon" /></span>
                                        </a>
                                    </p>
                                </div>
                            </div>
                        </div>
                    </div>
                </c:forEach>
            </div>
```

```
48              </div>
49          </div>
50      </section>
51  </body>
52  </html>
```

columns.jsp:

columns.jsp

```
1  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2  <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
3  <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
4  <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
5  <html>
6  <head>
7  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8  <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
9  <title>Fields</title>
10 </head>
11 <body>
12     <!-- <section>
26     <section class="container">
27         <div class="row">
28             <div class=caption>
29                 <c:forEach items="${columns}" var="column">
30                     <div class="col-sm-6 col-md-3" style="padding-bottom: 15px">
31                         <div class="thumbnail">
32                             <div class="caption">
33                                 <h5>Column field: ${column.field}</h5>
34                                 <p>Database Type: ${column.type}</p>
35                                 <p>Can be null: ${column.is_null}</p>
36                                 <p>Is key: ${column.key}</p>
37                                 <p>Has default: ${column.has_default }</p>
38                                 <p>Extra information: ${column.extra}<p>
39                                 <c:if test="${column.isString==true}">
40                                 <p>
41                                     <a href=" <spring:url
42                                     value=
43                                     "/data/add/field=${column.field}/type=${column.type}/table=${name}/db=${db}/newDb=${newDb}"/> "
44                                     class="btn btn-primary" >
45                                      <span class="glyphicon glyphicon-send" /></span>
46                                      Anonimize this column
47                                     </a>
48                                 </p>
49                                 </c:if>
```

```
50                             </div>
51                         </div>
52                     </div>
53                 </c:forEach>
54             </div>
55         </div>
56         <div class=row>
57             <div class=caption>
58                 <div class=thumbnail>
59                     <div>
60                         <a href=" <spring:url value="/tables/exp=${db}/new=${newDb}"/> "
61                             class="btn btn-primary float-right">
62                             <span class="glyphicon glyphicon-circle-arrow-left"></span>
63                             Choose another table
64                         </a>
65                     </div>
66                 </div>
67             </div>
68         </div>
69     </section>
70 </body>
71 </html>
```

addData.jsp:

addData.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet"
    href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
<title>Data</title>
</head>
<body>
    <!-- <section>..
    <section class="container">
        <form:form method="POST" modelAttribute="newData" class="form-horizontal">
            <fieldset>
                <legend>Enter de needed information to anonimyze properly</legend>

                <div class="form-group">
                    <label class="control-label col-lg-2 col-lg-2" for="id">Field name</label>
                    <div class="col-lg-10">
                        <form:input id="id" path="id" type="text" placeholder="${newData.id}"
                            class="form:input-large" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-lg-2 col-lg-2" for="tipoRgpd">RGPD type:</label>
                    <div class="col-lg-10">
                        <form:select path="tipoRgpd">
                            <form:option value="RAZA" label="Raza/Etnia" />
                            <form:option value="POLITICA" label="Ideología Política" />
                            <form:option value="RELIGION" label="Creencias Religiosas" />
                            <form:option value="SINDICATOS" label="Afiliación Sindical" />
                            <form:option value="SALUD" label="Estado de Salud" />
                            <form:option value="GENETICA" label="Condición Genética" />

                            <form:option value="SEXUAL" label="Orientación Sexual" />
                            <form:option value="NOMBRE" label="Nombre" />
                            <form:option value="APELLIDO" label="Apellido" />
                            <form:option value="IDENTIFICACION"
                                label="Número de Identificación" />
                            <form:option value="IDENTIFICACION EN LINEA"
                                label="Identificación en linea" />
                            <form:option value="TELEFONO" label="Número de Teléfono movil" />
                            <form:option value="FIJO" label="Número de Teléfono fijo" />
                            <form:option value="CUENTABANCARIA" label="Cuenta Bancaria" />

                            <form:option value="OTRO" label="Otro" />
                        </form:select>
                        <form:errors path="tipoRgpd" cssClass="text-danger" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-lg-2 col-lg-2" for="tipoBbdd">DB type</label>
                    <div class="col-lg-10">
                        <form:input id="tipoBbdd" path="tipoBbdd" type="text" placeholder="${newData.tipoBbdd}"
                            class="form:input-large" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-lg-2" for="personal">Personal:</label>
                    <div class="col-lg-10">
                        <form:checkbox id="personal" path="personal" />
                    </div>
                </div>

                <!-- <div class="form-group">..

                <div class="form-group">
                    <label class="control-label col-lg-2" for="sensible">Sensible:</label>
                    <div class="col-lg-10">
```

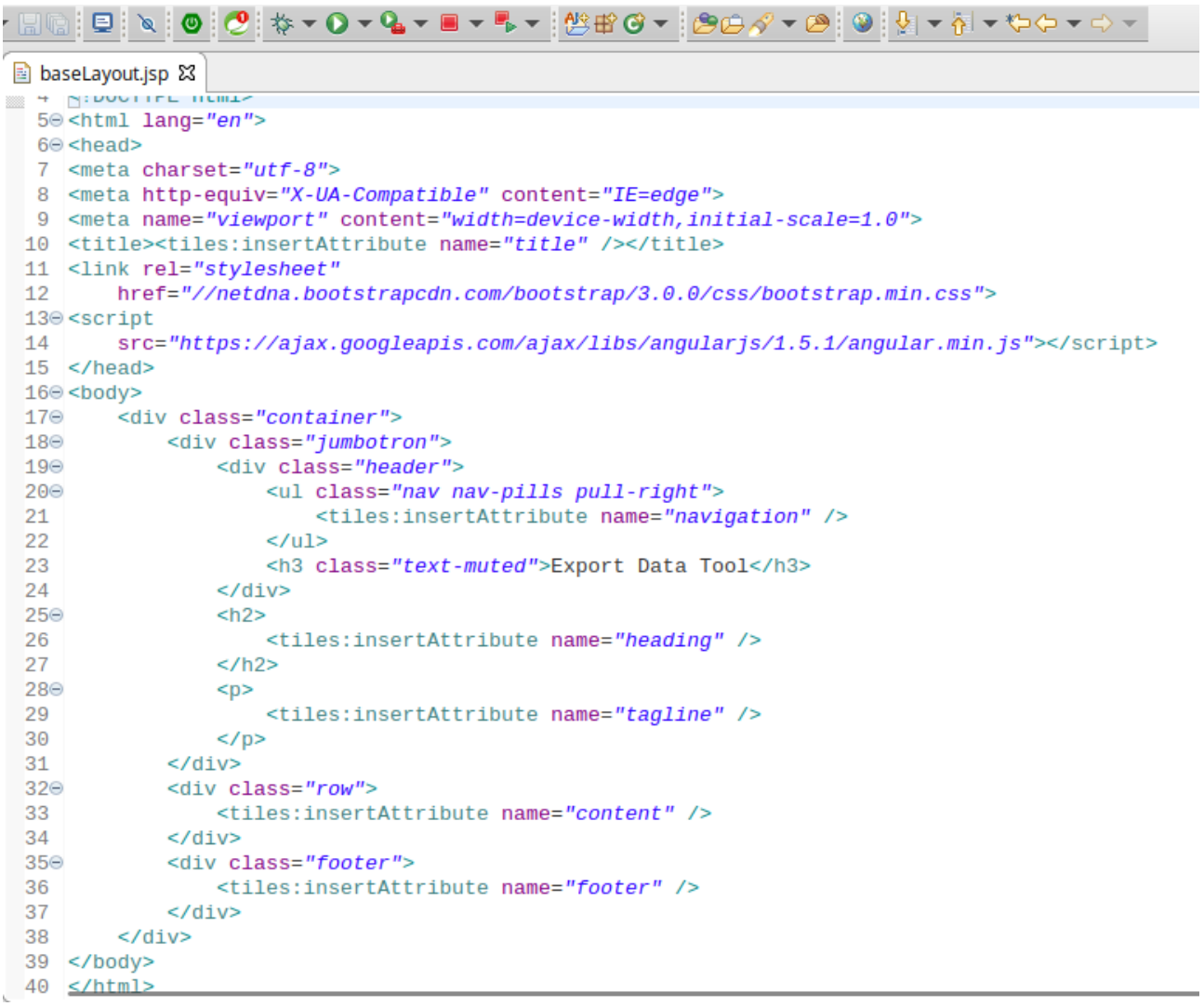
```
90                      <form:checkbox id="sensible" path="sensible" />
91                  </div>
92              </div>
93              <div>
94                  <p><strong>En caso de que sea sensible:</strong></p>
95              </div>
96              <div class="form-group">
97                  <label class="control-label col-lg-2" for="consentimiento_sensible">Consentimiento para tratamiento:</label>
98                  <div class="col-lg-10">
99                      <form:checkbox id="consentimiento_sensible" path="consentimiento_sensible" />
100                 </div>
101             </div>
102             <div class="form-group">
103                 <label class="control-label col-lg-2 col-lg-2" for="tablename">From
104                     table:</label>
105                 <div class="col-lg-10">
106                     <form:input id="tablename" path="tablename" type="text"
107                         placeholder="${newData.tablename}" class="form:input-large" />
108                 </div>
109             </div>
110             <div class="form-group">
111                 <div class="col-lg-offset-2 col-lg-10">
112                     <input type="submit" id="btnAdd" class="btn btn-primary btn-large"
113                         value="Anonymize" />
114                 </div>
115             </div>
116         </fieldset>
117     </form:form>
118 </section>
119 </body>
120 </html>
```

5.3.3. Layouts globales

Esta sección está dedicada en exclusiva a la explicación de la utilización de Apache Tiles para dotar a la aplicación web de un layout común. Como se explicó en la sección 5.2.2 los archivos que nos interesan se encuentran en la carpeta *src>main>webapps>WEB-INF>layouts* la cual contiene los directorios *definitions* y *template*.

En el directorio *templates* encontramos los elementos que queremos que se integren para formar un layout común. Estos son:

BaseLayout: Constituye como su nombre bien dice la base del marco de estilos.



footer.jsp: Se corresponde con el pie de página. La codificación es solo una línea:

© Company 2019

Navigation.jsp: Corresponde a la barra de navegación. La codificación es la siguiente:

```
<%@taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<li><a href="<spring:url value="/"/>">Home</a></li>
<li><a href="<spring:url value="/login?logout"/>">Logout</a></li>
```

Por último, en el directorio *definitions* en el fichero *tiles.xml* tendremos la definición del marco común y la extensión para todas las páginas de la aplicación web donde deseemos insertarlo. El contenido de este fichero será el siguiente:

tiles.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC "-//Apache
Software Foundation//DTD Tiles Configuration 3.0//EN"
"http://tiles.apache.org/dtds/tiles-config_3_0.dtd">
<tiles-definitions>
    <definition name="baseLayout" template="/WEB-INF/layouts/template/baseLayout.jsp">
        <put-attribute name="title" value="xportData" />
        <put-attribute name="heading" value="" />
        <put-attribute name="tagline" value="" />
        <put-attribute name="navigation" value="/WEB-INF/layouts/template/navigation.jsp" />
        <put-attribute name="content" value="" />
        <put-attribute name="footer" value="/WEB-INF/layouts/template/footer.jsp" />
    </definition>
    <definition name="tables" extends="baseLayout">
        <put-attribute name="title" value="Tables" />
        <put-attribute name="heading" value="Tables from the choosen database" />
        <put-attribute name="tagline" value="Choose the one you want to work with" />
        <put-attribute name="content" value="/WEB-INF/views/tables.jsp" />
    </definition>
    <definition name="addData" extends="baseLayout">
        <put-attribute name="title" value="Data" />
        <put-attribute name="heading" value="Data" />
        <put-attribute name="tagline" value="Add Data" />
        <put-attribute name="content" value="/WEB-INF/views/addData.jsp" />
    </definition>
    <definition name="login" extends="baseLayout">
        <put-attribute name="title" value="Login" />
        <put-attribute name="heading" value="Welcome to the data export tool!" />
        <put-attribute name="tagline" value="Log in to enter de application" />
        <put-attribute name="content" value="/WEB-INF/views/login.jsp" /> <!-- OK -->
    </definition>
    <definition name="databases" extends="baseLayout">
        <put-attribute name="title" value="Database menu" />
        <put-attribute name="heading" value="Select the database" />
        <put-attribute name="tagline" value="you want to work with" />
        <put-attribute name="content" value="/WEB-INF/views/databases.jsp" />
    </definition>
    <definition name="columns" extends="baseLayout">
        <put-attribute name="title" value="Fields" />
        <put-attribute name="heading" value="Fields from the choosen table" />
        <put-attribute name="tagline" value="Not varchar fields can't be anonymized" />
        <put-attribute name="content" value="/WEB-INF/views/columns.jsp" />
    </definition>
    <definition name="addDatabase" extends="baseLayout">
        <put-attribute name="title" value="New Database" />
        <put-attribute name="heading" value="Enter the name of the database" />
        <put-attribute name="tagline" value="you want to export data to." />
        <put-attribute name="content" value="/WEB-INF/views/addDatabase.jsp" />
    </definition>
</tiles-definitions>
```

5.4. Capa de Dominio de Aplicación

En este apartado se mostrará la implementación de las clases definidas en el diseño del problema y se comentará, en caso de que exista, algún aspecto singular o significativo.

Como comentario indicar que los constructores y métodos *setters* y *getters* generados automáticamente no se mostrarán en detalle.

Database.java:

Database.java

```
1  package com.tfg.xportData.domain;
2
3  import java.io.Serializable;
5
6  public class Database implements Serializable{
7
8      private static final long serialVersionUID = 2280245880265824012L;
9      private String name;
10     private ArrayList<Table> tables;
11
12     public Database() {
15     public Database(String name) {
18     public String getName() {
21     public void setName(String name) {
24
25     public void addTable(Table table) {
26         tables.add(table);
27     }
28
29     public Table getTableByName(String name) {
30         Table tableReturned = null;
31         for(Table table: tables) {
32             if(table.getName().equals(name)) {
33                 tableReturned=table;
34             }
35         }
36         return tableReturned;
37     }
38 }
39
```

Table.java:

```java
package com.tfg.xportData.domain;

import java.io.Serializable;

public class Table implements Serializable {

    private static final long serialVersionUID = -2117424686156308038L;
    private String name;
    private ArrayList<Column> columns;

    public String getName() {

    public void setName(String name) {


    public void addColumn(Column column) {

    public Column getColumnByFieldName(String name) {
        Column columnReturned = null;
        for(Column column: columns) {
            if(column.getField().equals(name)) {
                columnReturned=column;
            }
        }
        return columnReturned;
    }
}

```

Column.java:

```java
package com.tfg.xportData.domain;

import java.io.Serializable;

public class Column implements Serializable {

    private static final long serialVersionUID = -6054076767322884185L;

    private String field;
    private String type;
    private boolean is_null;
    private String key;
    private String has_default;
    private String extra;
    public boolean isString;

    public String getField() {...
    public void setField(String field) {...
    public String getType() {...
    public void setType(String type) {...
    public boolean isIs_null() {...
    public void setIs_null(boolean is_null) {...
    public String getKey() {...
    public void setKey(String key) {...
    public String getHas_default() {...
    public void setHas_default(String has_default) {...
    public String getExtra() {...
    public void setExtra(String extra) {...
    public boolean isIsString() {...
    public void setIs_String(boolean is_String) {...

}
```

Data.java:

```java
package com.tfg.xportData.domain;

import java.io.Serializable;

public class Data implements Serializable {

    private static final long serialVersionUID = 7635411584628432299L;

    private String id;
    private String tipoBbdd=null;
    private String tipoRgpd=null;
    private boolean personal = false;
    private boolean sensible = false;
    private boolean consentimiento_sensible = false;
    private boolean anonimizado = false;
    private String tratamiento;
    private Date fecha;
    private String tablename;


    public Data() {

    public Data(String id, String tipoBbdd) {
        this.id=id;
        this.tipoBbdd=tipoBbdd;
        this.fecha=new Date();
    }
    public String getId() {
    public void setId(String id) {
    public String getTipoBbdd() {
    public void setTipoBbdd(String tipoBbdd) {
    public String getTipoRgpd() {
    public void setTipoRgpd(String tipoRgpd) {
    public boolean getPersonal() {
    public void setPersonal(boolean personal) {
    public boolean getSensible() {
```

```java
    public void setPersonal(boolean personal) {
    public boolean getSensible() {
    public void setSensible(boolean sensible) {

    public boolean isConsentimiento_sensible() {

    public void setConsentimiento_sensible(boolean consentimiento_sensible) {

    public boolean getAnonimizado() {
    public void setAnonimizado(boolean anonimizado) {
    public String getTratamiento() {
    public void setTratamiento(String tratamiento) {
    public Date getFecha() {
    public void setFecha(Date fecha) {

    public String getTablename() {

    public void setTablename(String tablename) {

    public List<String> anonymize(List<String> actualValues){
        AnonymizationTool tool =new AnonymizationTool();
        List <String> newValues = tool.anonymize(actualValues, this);
        return newValues;
    }
    @Override
    public String toString() {
        return "Campo [tipo_bbdd=" + tipoBbdd + ", tipo_rgpd=" + tipoRgpd + ", personal=" + personal
                + ", sensible=" + sensible + ", anonimizado=" + anonimizado + ", tratamiento=" + tratamiento
                + ", fecha=" + fecha + "]";
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
```

```
115         return result;
116     }
117
118 }
```

AnonimizationTool.java: Dada la gran extensión de este archivo solo se mostrarán los aspectos importantes

```
55    public List <String> anonymize(List <String> actualValues, Data data){
56        List <String> newValues = new ArrayList<String>();
57
58        if(data.getSensible()==true && data.isConsentimiento_sensible()==false) { //Si el dato es sensible
59            for(String value: actualValues){                                       // y no tenemos permiso para tratarlo
60                newValues.add("XXXXXXXX");                                        // No realizamos tratamiento alguno
61            }
62        }
63        else {
64            //Si el dato es personal o sensible con autorizacion para ser tratado, lo anonimizamos
65            if((data.getSensible()==true && data.isConsentimiento_sensible()==true) ||
66                    (data.getPersonal()==true)) {
67                //Anonimizamos de una manera u otra en funcion del tipo aplicable del RGPD
68                switch(data.getTipoRgpd()) {
69                    case "RAZA":
70                        for(String temp: actualValues) {
71                            Random r1= new Random();
72                            int randomIndex = (int) (r1.nextDouble()*7);
73                            newValues.add(razas[randomIndex]);
74                        }
75                        data.setFecha(new Date());
76                        data.setTratamiento("Anonimizacion de indice aleatorio.");
77                        data.setAnonimizado(true);
78                        break;
79                    case "POLITICA":
80                        for(String temp: actualValues) {
81                            Random r2= new Random();
82                            int randomIndex = (int) (r2.nextDouble()*14);
83                            newValues.add(politicas[randomIndex]);
84                        }
85                        data.setFecha(new Date());
86                        data.setTratamiento("Anonimizacion de indice aleatorio.");
87                        data.setAnonimizado(true);
88                        break;
89                    case "RELIGION":
89                    case "RELIGION":
90                        for(String temp: actualValues) {
91                            Random r3= new Random();
92                            int randomIndex = (int) (r3.nextDouble()*11);
93                            newValues.add(religiones[randomIndex]);
94                        }
95                        data.setFecha(new Date());
96                        data.setTratamiento("Anonimizacion de indice aleatorio.");
97                        data.setAnonimizado(true);
98                        break;
99                    case "SINDICATOS":
100                       for(String temp: actualValues) {
101                           Random r4= new Random();
102                           int randomIndex = (int) (r4.nextDouble()*3);
103                           newValues.add(sindicatos[randomIndex]);
104                       }
105                       data.setFecha(new Date());
106                       data.setTratamiento("Anonimizacion de indice aleatorio.");
107                       data.setAnonimizado(true);
108                       break;
109                   case "SALUD":
110                       for(String temp: actualValues) {
111                           Random r5= new Random();
112                           int randomIndex = (int) (r5.nextDouble()*4);
113                           newValues.add(estadosClinicos[randomIndex]);
114                       }
115                       data.setFecha(new Date());
116                       data.setTratamiento("Anonimizacion de indice aleatorio.");
117                       data.setAnonimizado(true);
118                       break;
119                   case "GENETICA":
120                       for(String temp: actualValues) {
121                           Random r6= new Random();
122                           int randomIndex = (int) (r6.nextDouble()*5);
123                           newValues.add(condicionesGeneticas[randomIndex]);
124                       }
```

```
            for(String temp: actualValues) {
                Random r7= new Random();
                int randomIndex = (int) (r7.nextDouble()*3);
                newValues.add(orientacionesSexuales[randomIndex]);
            }
            data.setFecha(new Date());
            data.setTratamiento("Anonimizacion de indice aleatorio.");
            data.setAnonimizado(true);
            break;
        case "NOMBRE":
            for(String temp: actualValues) {
                Random r8= new Random();
                int randomIndex = (int) (r8.nextDouble()*20);
                newValues.add(nombres[randomIndex]);
            }
            data.setFecha(new Date());
            data.setTratamiento("Anonimizacion de indice aleatorio.");
            data.setAnonimizado(true);
            break;
        case "APELLIDO":
            for(String temp: actualValues) {
                Random r9= new Random();
                int randomIndex = (int) (r9.nextDouble()*14);
                newValues.add(apellidos[randomIndex]);
            }
            data.setFecha(new Date());
            data.setTratamiento("Anonimizacion de indice aleatorio.");
            data.setAnonimizado(true);
            break;
        case "IDENTIFICACION":
            for(String temp: actualValues) {
                int key = temp.hashCode();
                String anonymized="";
                int contador=0;
                for (int n=0 ; n <= 8; n++){
                    char letra = temp.charAt(n);
                    int letraAscii = (int) letra;
                    int codigoAscii = letraAscii + key;
                    while(codigoAscii < 48) {
                        codigoAscii = codigoAscii + 10;
                    }
                    while(codigoAscii > 57){
                        codigoAscii = codigoAscii - 10;
                    }
                    contador += (codigoAscii - 48)*(10^(n));
                    anonymized += (char) codigoAscii;
                }
                int resto = contador%23;
                anonymized += letras[resto];
                newValues.add(anonymized);
                }
            data.setFecha(new Date());
            data.setTratamiento("Desplazamiento de digitos bidireccional con clave autogenerada.");
            data.setAnonimizado(true);
            break;
        case "TELEFONO":
            Random randomMovil = new Random();
            for(String temp: actualValues) {
                StringBuilder builder=new StringBuilder(temp);
                String reversed = builder.reverse().toString();
                String cut = reversed.substring(1);
                String definitive = "";
                if(randomMovil.nextBoolean()) {
                    definitive = 6 + cut;
                }
                else {
                    definitive = 7 + cut;
                }
                newValues.add(definitive);
```

```
                }
                break;
            case "FIJO":
                Random randomFijo = new Random();
                for(String temp: actualValues) {
                    StringBuilder builder=new StringBuilder(temp);
                    String reversed = builder.reverse().toString();
                    String cut = reversed.substring(1);
                    String definitive = null;
                    if(randomFijo.nextBoolean()) {
                        definitive = 9 + cut;
                    }
                    else {
                        definitive = 8 + cut;
                    }
                    newValues.add(definitive);
                }
                data.setFecha(new Date());
                data.setTratamiento("Inversion de digitos con adecuacion al estandar.");
                data.setAnonimizado(true);
                break;
            case "CUENTABANCARIA":
                String begining = "ES98";
                for(String temp: actualValues) {
                    int key = temp.hashCode();
                    String ending = temp.substring(4);
                    String anonymized="";
                    for ( int n=0 ; n <ending.length (); n++){
                        char letra=ending.charAt (n);
                        int letraAscii = (int) letra;
                        int codigoAscii = letraAscii + key;
                        while(codigoAscii < 49) {
                            codigoAscii = codigoAscii + 10;
                        }
                        while(codigoAscii > 57){
                            codigoAscii = codigoAscii - 10;
                        }
                        anonymized += (char) (codigoAscii);
                    }
                    String completed = begining + anonymized;
                    String ultimated = completed.toUpperCase();
                    newValues.add(ultimated);
                }
                data.setFecha(new Date());
                data.setTratamiento("Generacion aleatoria con adecuacion al estandar.");
                data.setAnonimizado(true);
                break;
            default:
                for(String temp: actualValues) {
                    int key = temp.hashCode();
                    String anonymized = "";
                    for ( int n=0 ; n <temp.length (); n++){
                        char letra=temp.charAt (n);
                        int letraAscii = (int) letra;
                        int codigoAscii = letraAscii + key;
                        while(codigoAscii < 97) {
                            codigoAscii = codigoAscii + 25;
                        }
                        while(codigoAscii > 122){
                            codigoAscii = codigoAscii - 25;
                        }
                        anonymized += (char) (codigoAscii);
                    }
                    newValues.add(anonymized);
                }
        }
    }
    //Si el dato no es personal ni sensible lo copiamos directamente
    else {
        for(String value: actualValues) {
```

```
            else {
                for(String value: actualValues) {
                    newValues.add(value);
                }
            }
        }
        tool.log(data);
        return newValues;
    }

}
```

PrintTool.java:

```
1  package com.trg.xportData.domain;
2
3  import java.io.BufferedWriter;
7
8  public class PrintTool {
9
10     public PrintTool() {
11         super();
12     }
13
14     public void log(Data data) {
15         //Definimos el nombre del fichero y como serán las entradas de texto
16         String filename = data.getTablename()+".txt";
17         String line = data.getFecha().toString()+": Columna "+data.getId()
18         +" de caracter personal("+data.getPersonal() +") y de caracter sensible("+data.getSensible()+")."
19         +" Tipo aplicable RGPD: "+data.getTipoRgpd()+"."
20         +" Tratamiento realizado: "+ data.getTratamiento()+"\n";
21
22         //Creamos el fichero file
23         File file = new File("/home/ezequiel/logs",filename);
24
25         //Comprobamos si existe ya
26         //Si no existe lo creamos
27         if(!file.exists()) {
28             try{
29                 if (file.createNewFile()) {System.out.println("El fichero se ha creado correctamente");}
30                 else {System.out.println("No ha podido ser creado el fichero");}
31                 }
32             catch (IOException ioe) {ioe.printStackTrace();}
33         }
34         //Tanto si ya existia como si lo hemos tenido que crear, ahora escribimos en el
35         try {
36             BufferedWriter bw = new BufferedWriter(new FileWriter(file,true));
37             bw.write(line);
38             bw.close();
39         } catch (IOException e) {
40             e.printStackTrace();
```

Writable Smart Insert

5.5. Capa de Persistencia

En este apartado se va a mostrar el código de las clases que implementan las interfaces establecidas en el diseño y con la nomenclatura de nombrado adecuada.

DatabaseRepositoryImpl.java:

PrintTool.java | DatabaseRepositoryImpl.java

```java
@Repository
public class DatabaseRepositoryImpl implements DatabaseRepository {

    @Autowired
    private NamedParameterJdbcTemplate jdbcTemplate;

    @Override
    public List<Database> getAllDatabases() {
        Map<String, Object> params = new HashMap<String,Object>();
        List<Database> result = jdbcTemplate.query("SHOW DATABASES", params, new DatabaseMapper());
        return result;
    }

    @Override
    public void addDatabaseByName(String name) {
        String SQL="CREATE DATABASE "+name;
        Map<String, Object> params = new HashMap<>();
        jdbcTemplate.update(SQL, params);
    }

    private static final class DatabaseMapper implements RowMapper<Database> {
        public Database mapRow(ResultSet rs, int rowNum) throws SQLException {
            Database db = new Database();
            db.setName(rs.getString("Database"));
            return db;
        }
    }

    public void changeToDatabaseByName(String name) {
        String SQL="USE " + name;
        Map<String, Object> params = new HashMap<String,Object>();
        jdbcTemplate.update(SQL, params);
    }

}
```

TablesRepositoryImpl.java:

```java
package com.tfg.xportData.domain.repository.impl;

import java.sql.ResultSet;

@Repository
public class TablesRepositoryImpl implements TablesRepository {

    @Autowired
    private NamedParameterJdbcTemplate jdbcTemplate;

    @Override
    public List<Table> getAllTables(String dbname) {
        Map<String, Object> params = new HashMap<String,Object>();
        List<Table> result = jdbcTemplate.query(
                "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='"+dbname+"'",
                params, new TableMapper());
        return result;
    }

    private static final class TableMapper implements RowMapper<Table> {
        public Table mapRow(ResultSet rs, int rowNum) throws SQLException {
            Table table = new Table();
            table.setName(rs.getString("TABLE_NAME"));
            return table;
        }
    }

    @Override
    public void copyTableByName(String tablename, String newDb) {
        Map<String, Object> params = new HashMap<String,Object>();
        String SQL="CREATE TABLE "+newDb+"."+tablename+ " SELECT * FROM "+tablename;
        jdbcTemplate.update(SQL, params);
    }
}
```

ColumnRepositoryImpl:

ColumnRepositoryImpl.java

```java
package com.tfg.xportData.domain.repository.impl;

import java.sql.ResultSet;

@Repository
public class ColumnRepositoryImpl implements ColumnRepository {

    @Autowired
    private NamedParameterJdbcTemplate jdbcTemplate;

    @Override
    public List<Column> getAllColumns(String tablename) {
        Map<String, Object> params = new HashMap<String,Object>();
        List<Column> result = jdbcTemplate.query(
                "DESCRIBE "+tablename,
                params, new ColumnMapper());
        return result;
    }

    private static final class ColumnMapper implements RowMapper<Column> {
        public Column mapRow(ResultSet rs, int rowNum) throws SQLException {
            Column column = new Column();
            column.setField(rs.getString("Field"));
            column.setType(rs.getString("Type"));
            column.setIs_null(rs.getBoolean("Null"));
            column.setKey(rs.getString("Key"));
            column.setHas_default(rs.getString("Default"));
            column.setExtra(rs.getString("Extra"));
            if(column.getType().startsWith("varchar")) {
                column.setIs_String(true);
            }
            return column;
        }
    }

}
```

DataRepositoryImpl.java:

```java
package com.tfg.xportData.domain.repository.impl;

import java.sql.ResultSet;

@Repository
public class DataRepositoryImpl implements com.tfg.xportData.domain.repository.DataRepository {

    @Autowired
    private NamedParameterJdbcTemplate jdbcTemplate;

    @Override
    public void anonimyze(Data data, String newDb) {
        Map<String, Object> params = new HashMap<String,Object>();
        String selectSQL = "SELECT "+data.getId()+" FROM "+newDb+"."+data.getTablename();
        String columnName = data.getId();
        List<String> actualValues = jdbcTemplate.query(selectSQL, params, new StringMapper(columnName));
        List<String> newValues = data.anonymize(actualValues);
        Iterator<String> actualIt = actualValues.iterator();
        Iterator<String> newIt = newValues.iterator();
        while(newIt.hasNext() && actualIt.hasNext()) {
            String oldValue = actualIt.next();
            String newValue = newIt.next();
            String updateSQL =
                    "UPDATE "+newDb+"."+data.getTablename()+
                    " SET "+data.getId()+"='"+newValue+"' "+
                    "WHERE "+data.getId()+"='"+oldValue+"'";
            Map<String, Object> params2 = new HashMap<String,Object>();
            jdbcTemplate.update(updateSQL, params2);
            }
        }
    }

    class StringMapper implements RowMapper<String> {
```

```java
    class StringMapper implements RowMapper<String> {

        private String fieldname;

        public StringMapper(String fieldname) {
            this.fieldname=fieldname;
        }
        public String mapRow(ResultSet rs, int rowNum) throws SQLException {
            String value= rs.getString(fieldname);
            return value;
        }
    }
```

En estas clases se puede observar que en todos los casos es necesario definir una clase que implemente la interfaz RowMapper. Esta clase nos servirá para convertir los valores obtenidos de la base de datos a objetos de las clases que hayamos definido a través de las llamadas a los métodos de la clase ResultSet (*getString()*, *getInt()*, *getBoolean()*, etc).

Este es el motivo de que sea necesario realizar la simplificación explicada al comienzo de este capítulo.

5.6. Capa de Servicio

Al igual que en la sección anterior se mostrará el código de las clases que implementan las interfaces definidas en el diseño. Estas clases también siguen la nomenclatura de nombrado explicada en el capítulo anterior.

DatabaseServiceImpl.java:

DatabaseServiceImpl.java

```java
package com.tfg.xportData.service.impl;

import java.util.List;

@Service
public class DatabaseServiceImpl implements DatabaseService {

    @Autowired
    private DatabaseRepository databaseRepository;

    @Override
    public List<Database> getAllDatabases() {
        return databaseRepository.getAllDatabases();
    }

    @Override
    public void addDatabaseByName(String name) {
        databaseRepository.addDatabaseByName(name);
    }

    public void changeToDatabaseByName(String name) {
        databaseRepository.changeToDatabaseByName(name);
    }

}
```

TablesServicesImpl.java:

TablesServiceImpl.java

```java
package com.tfg.xportData.service.impl;

import java.util.List;

@Service
public class TablesServiceImpl implements TablesService {

    @Autowired
    private TablesRepository tablesRepository;

    @Override
    public List<Table> getAllTables(String dbname) {
        return tablesRepository.getAllTables(dbname);
    }

    @Override
    public void copyTableByName(String tablename, String newDb) {
        tablesRepository.copyTableByName(tablename, newDb);
    }

}
```

ColumnServicesImpl.java:

ColumnServiceImpl.java

```java
package com.tfg.xportData.service.impl;

import java.util.List;

@Service
public class ColumnServiceImpl implements ColumnService{

    @Autowired
    private ColumnRepository columnRepository;

    @Override
    public List<Column> getAllColumns(String tablename) {
        return columnRepository.getAllColumns(tablename);
    }

}
```

DataServiceImpl.java:

DataServiceImpl.java

```java
package com.tfg.xportData.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class DataServiceImpl implements DataService {

    @Autowired
    private DataRepository dataRepository;

    @Override
    public void anonimyze(Data data, String newDb) {

        dataRepository.anonimyze(data,newDb);

    }
}
```

6 RESULTADOS

En este capítulo se procederá a mostrar de forma gráfica de la herramienta de exportación de datos a través de capturas de las distintas páginas de la aplicación web, los resultados observables a través de la base de datos y el registro generado. Para ello se utilizará una base de datos creada a propósito para una muestra simple de las funcionalidades de la aplicación.

Dicho esto, cuando accedemos a la página de inicio de la herramienta de exportación de datos nos encontramos con el login:

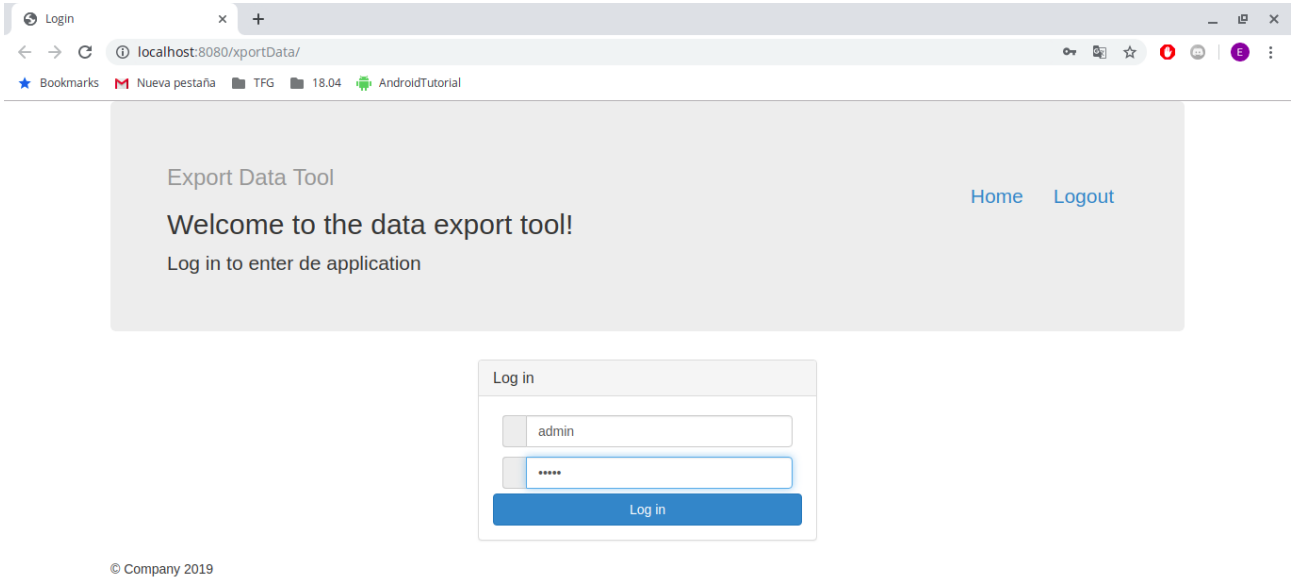


Figura 6-1. Página de inicio.

Al pulsar en *Log in* se nos mostrarán las distintas bases de datos disponibles en MariaDB. Se muestra a continuación la concordancia entre lo obtenido del cliente de terminal y lo obtenido por la herramienta.

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| tfg                |
+--------------------+
4 rows in set (0.001 sec)
```

Figura 6-2. Bases de datos obtenidas a través del cliente mysql.

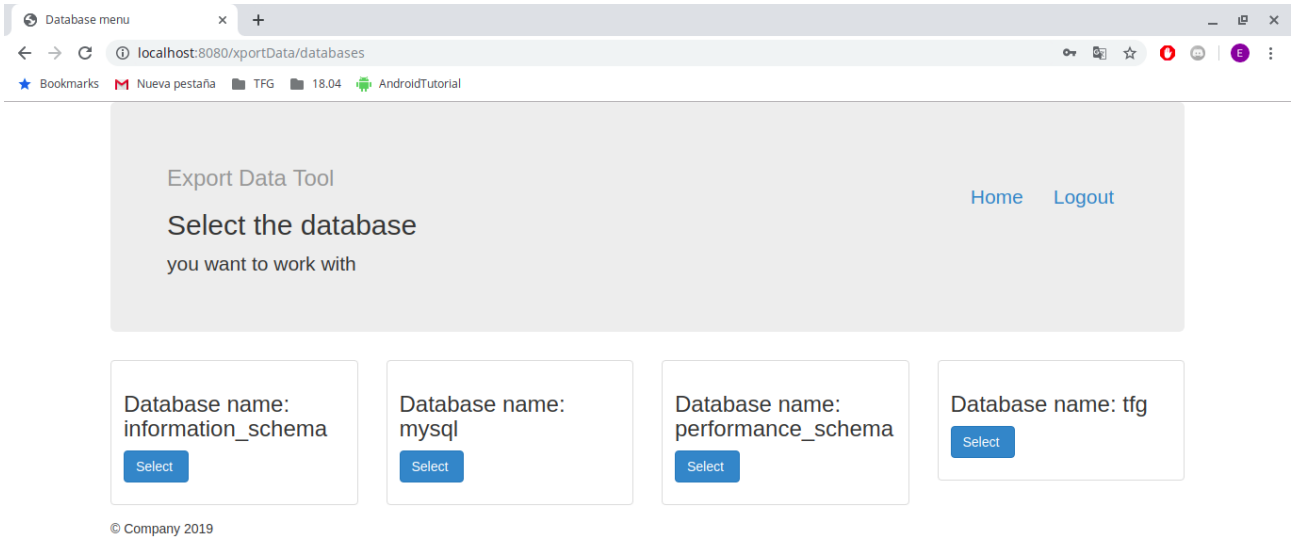


Figura 6-3. Base de datos a través de la aplicación.

Cuando seleccionemos la base de datos con la que deseemos trabajar (en este caso para el ejemplo la base de datos tfg) se nos pedirá el nombre de la base de datos a la que deseamos exportar.

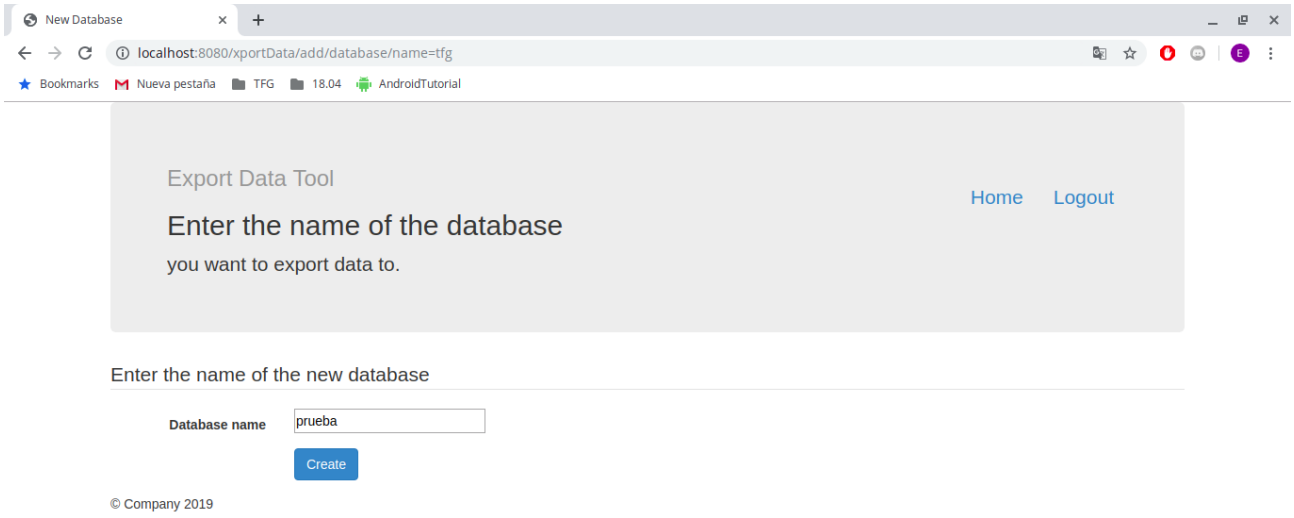


Figura 6-4. Formulario de creación de base de datos de exportación.

Cuando pulsemos el botón *Create* se creará la base de datos y se nos mostrarán las tablas almacenadas en la base de datos seleccionada como fuente de datos. Se muestra ahora la comprobación de la creación de la base de datos de exportación, la página que permite elegir la tabla a utilizar que en este caso solo tendrá

una tabla “USUARIOS” definida al ejemplo y la comprobación de que el listado de tablas es coincidente con el obtenido a través del cliente *mysql*.

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| prueba             |
| tfg                |
+--------------------+
5 rows in set (0.001 sec)
```

Figura 6-5. Comprobación creación base de datos de exportación.

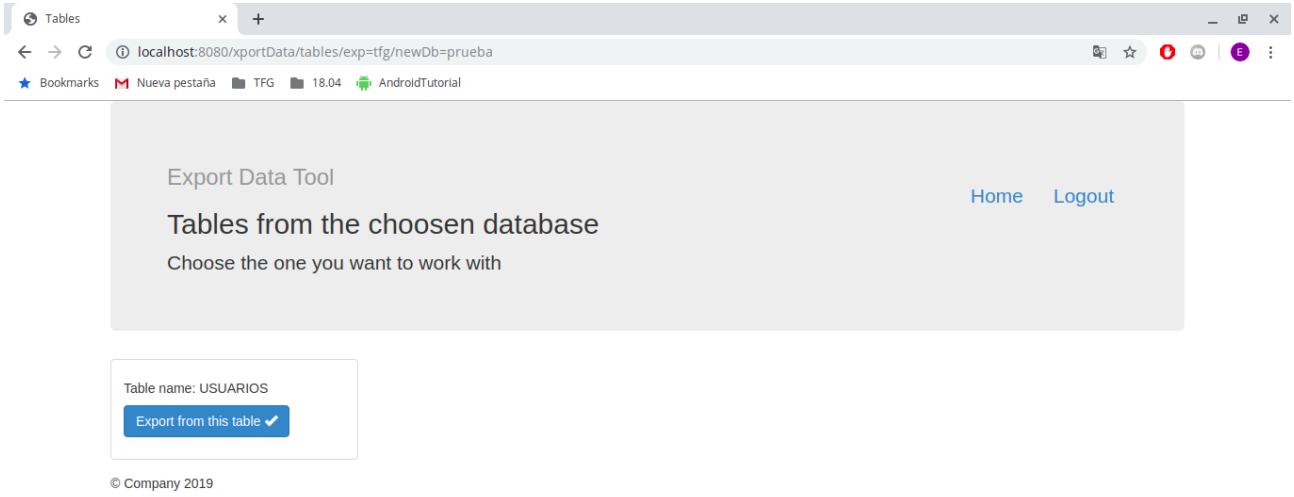


Figura 6-6. Listado de las tablas de una base de datos a través de la aplicación web.

```
MariaDB [(none)]> SHOW FULL TABLES FROM tfg;
+---------------+------------+
| Tables_in_tfg | Table_type |
+---------------+------------+
| USUARIOS      | BASE TABLE |
+---------------+------------+
1 row in set (0.001 sec)
```

Figura 6-7. Comprobación del listado a través del cliente mysql.

Ahora mismo la base de datos que hemos creado se encuentra vacía,

```
MariaDB [(none)]> SHOW FULL TABLES FROM prueba;
Empty set (0.001 sec)
```

Figura 6-8. Estado inicial de la base de datos de exportación.

pero cuando seleccionemos una tabla en la página que nos encontramos esta tabla será copiada a la base de datos de exportación. Esta copia será tanto en esquema como en valores.

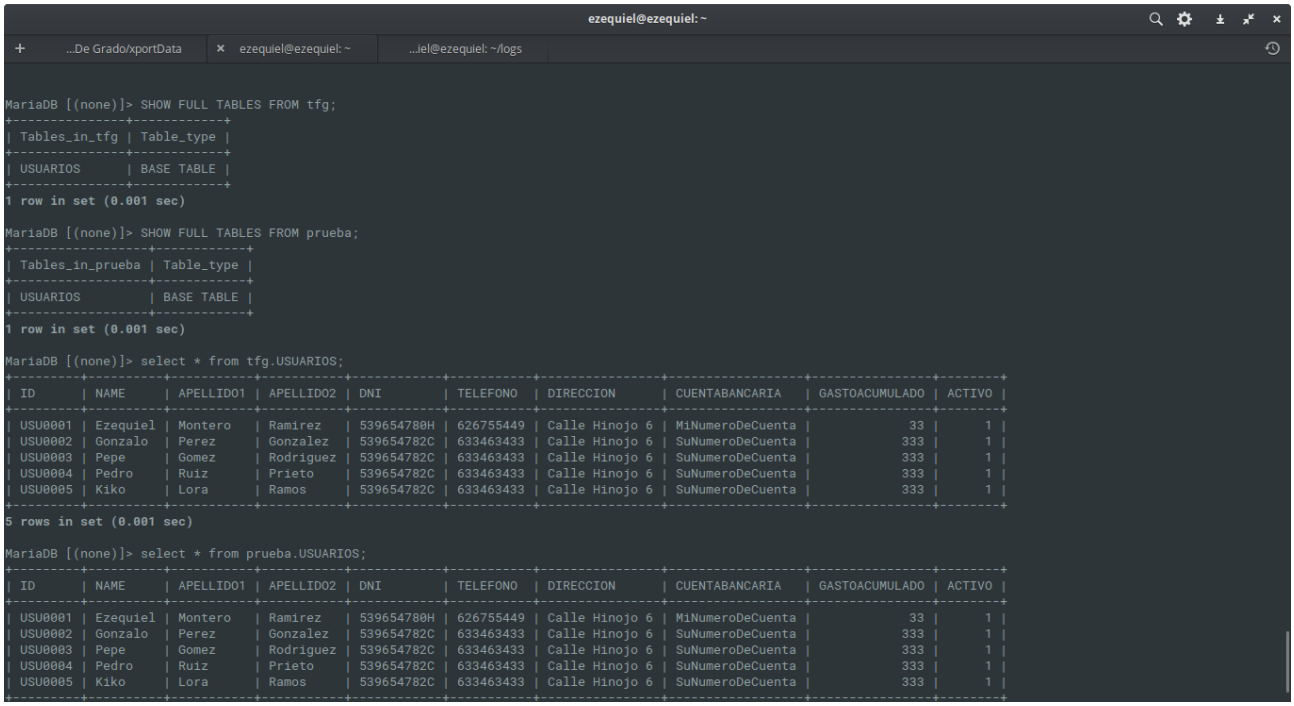


Figura 6-9. Estado de la base de datos de exportación tras la selección de una tabla.

Tras esto la página web mostrará las columnas de la tabla seleccionada.

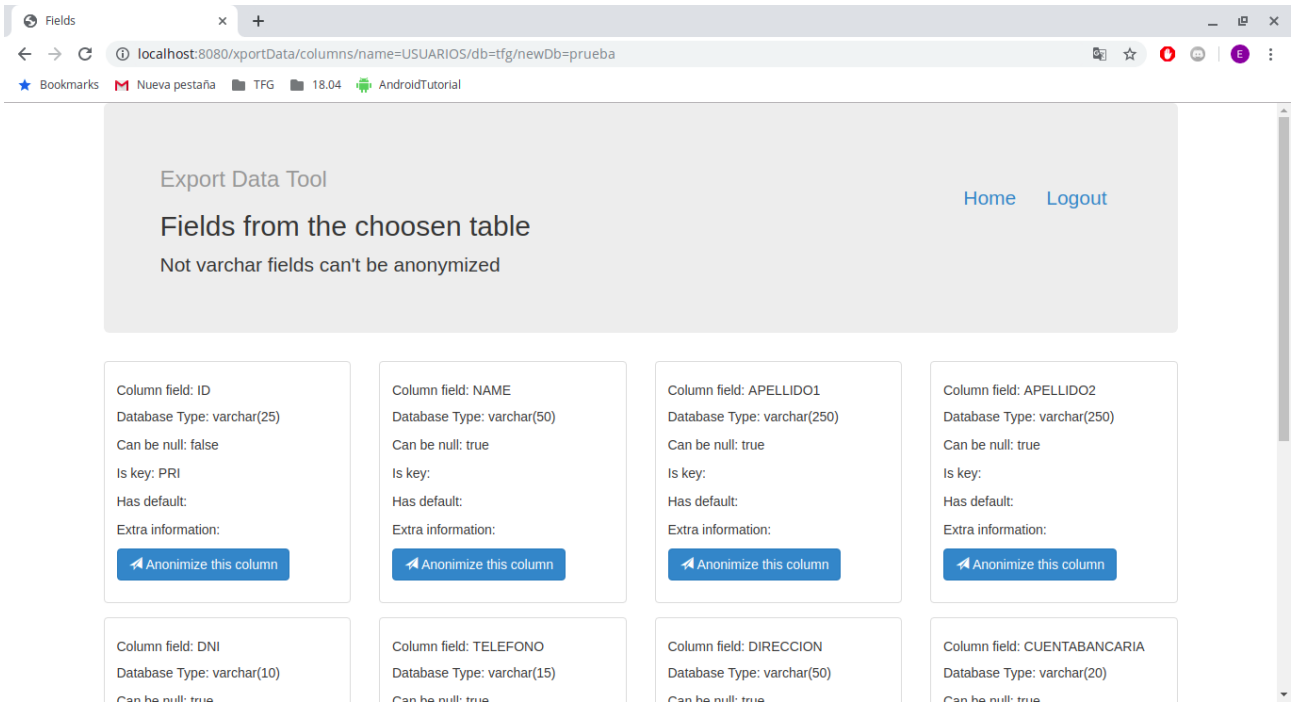


Figura 6-10. Columnas de la tabla seleccionada (I).

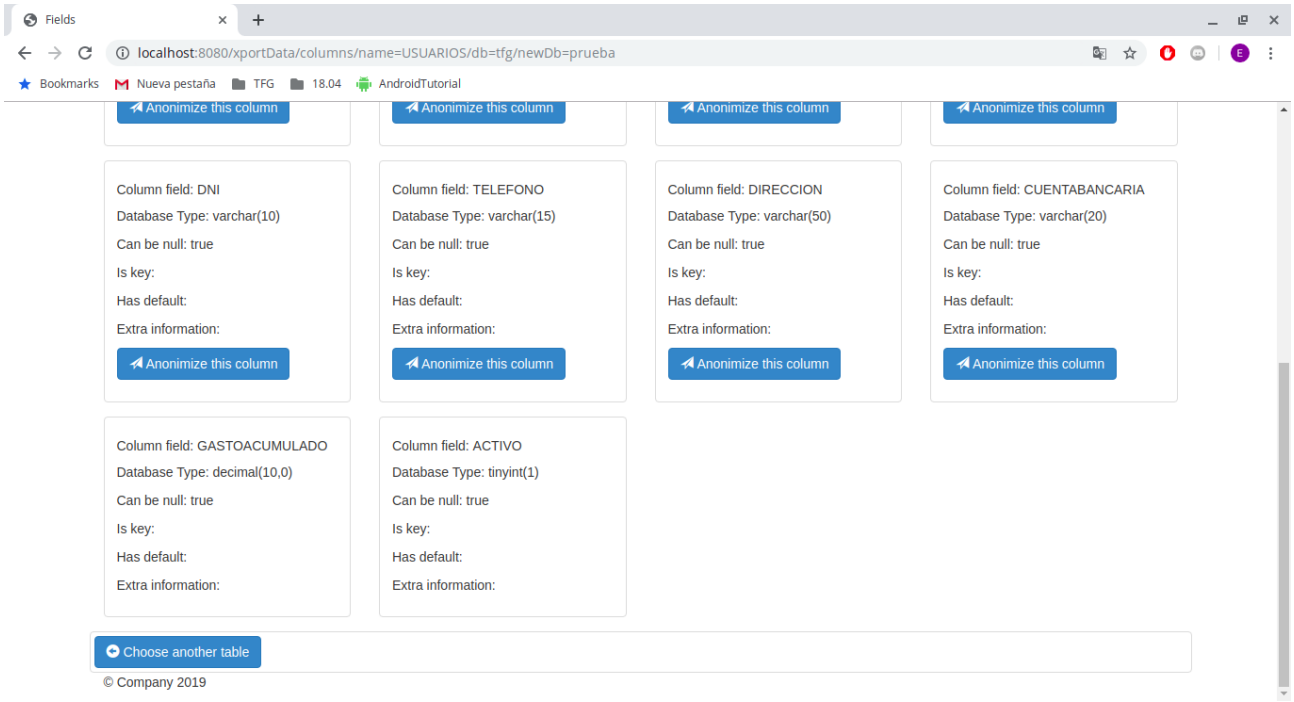


Figura 6-11. Columnas de la tabla seleccionada (II).

```
MariaDB [(none)]> Describe tfg.USUARIOS;
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| ID             | varchar(25)   | NO   | PRI | NULL    |       |
| NAME           | varchar(50)   | YES  |     | NULL    |       |
| APELLIDO1      | varchar(250)  | YES  |     | NULL    |       |
| APELLIDO2      | varchar(250)  | YES  |     | NULL    |       |
| DNI            | varchar(10)   | YES  |     | NULL    |       |
| TELEFONO       | varchar(15)   | YES  |     | NULL    |       |
| DIRECCION      | varchar(50)   | YES  |     | NULL    |       |
| CUENTABANCARIA | varchar(20)   | YES  |     | NULL    |       |
| GASTOACUMULADO | decimal(10,0) | YES  |     | NULL    |       |
| ACTIVO         | tinyint(1)    | YES  |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
10 rows in set (0.003 sec)
```

Figura 6-12. Concordancia con las columnas mostradas por la aplicación web.

Tras esto seleccionaremos la columna que deseemos anonimizar/seudonimizar y nos aparecerá el siguiente formulario que debemos cumplimentar con los valores que no aparecen rellenados.

Se mostrará en el ejemplo para la columna NAME, pero se realizará también para las columnas APELLIDO1, APELLIDO2, DNI, Y CUENTABANCARIA.

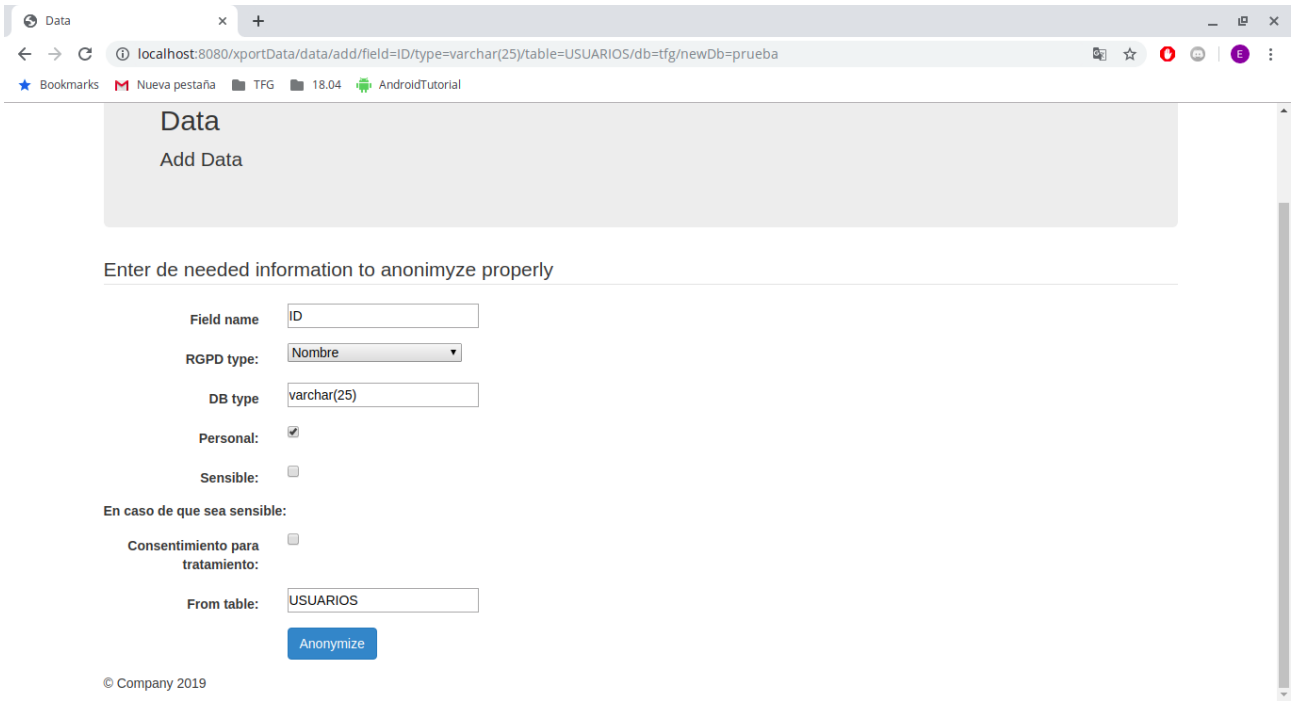


Figura 6-13. Formulación de anonimización de datos.

Desde el cliente mysql se podrán observar los cambios realizados por la herramienta.

```
MariaDB [(none)]> SELECT * FROM prueba.USUARIOS;
+---------+----------+-----------+-----------+------------+-----------+---------------+------------------+----------------+--------+
| ID      | NAME     | APELLIDO1 | APELLIDO2 | DNI        | TELEFONO  | DIRECCION     | CUENTABANCARIA   | GASTOACUMULADO | ACTIVO |
+---------+----------+-----------+-----------+------------+-----------+---------------+------------------+----------------+--------+
| USU0001 | Ezequiel | Montero   | Ramirez   | 539654780H | 626755449 | Calle Hinojo 6 | MiNumeroDeCuenta |             33 |      1 |
| USU0002 | Gonzalo  | Perez     | Gonzalez  | 539654782C | 633463433 | Calle Hinojo 6 | SuNumeroDeCuenta |            333 |      1 |
| USU0003 | Pepe     | Gomez     | Rodriguez | 539654782C | 633463433 | Calle Hinojo 6 | SuNumeroDeCuenta |            333 |      1 |
| USU0004 | Pedro    | Ruiz      | Prieto    | 539654782C | 633463433 | Calle Hinojo 6 | SuNumeroDeCuenta |            333 |      1 |
| USU0005 | Kiko     | Lora      | Ramos     | 539654782C | 633463433 | Calle Hinojo 6 | SuNumeroDeCuenta |            333 |      1 |
+---------+----------+-----------+-----------+------------+-----------+---------------+------------------+----------------+--------+
5 rows in set (0.001 sec)

MariaDB [(none)]> SELECT * FROM prueba.USUARIOS;
+---------+----------+-----------+-----------+------------+-----------+---------------+------------------+----------------+--------+
| ID      | NAME     | APELLIDO1 | APELLIDO2 | DNI        | TELEFONO  | DIRECCION     | CUENTABANCARIA   | GASTOACUMULADO | ACTIVO |
+---------+----------+-----------+-----------+------------+-----------+---------------+------------------+----------------+--------+
| USU0001 | Antonio  | Perez     | Diaz      | 640765891N | 626755449 | Calle Hinojo 6 | ES98247414004390 |             33 |      1 |
| USU0002 | Antonio  | Lopez     | Perez     | 317432560K | 633463433 | Calle Hinojo 6 | ES98025292882178 |            333 |      1 |
| USU0003 | Teresa   | Sanchez   | Gomez     | 317432560K | 633463433 | Calle Hinojo 6 | ES98025292882178 |            333 |      1 |
| USU0004 | Pilar    | Perez     | Garcia    | 317432560K | 633463433 | Calle Hinojo 6 | ES98025292882178 |            333 |      1 |
| USU0005 | Cristina | Gonzalez  | Martin    | 317432560K | 633463433 | Calle Hinojo 6 | ES98025292882178 |            333 |      1 |
+---------+----------+-----------+-----------+------------+-----------+---------------+------------------+----------------+--------+
5 rows in set (0.001 sec)
```

Figura 6-14. Estado inicial y final de los datos exportados.

Tras esto podríamos seleccionar otra tabla si existiera y continuar con el proceso. Si no es lo deseado podríamos pulsar en Logout, lo cual nos redirigiría a lo siguiente.

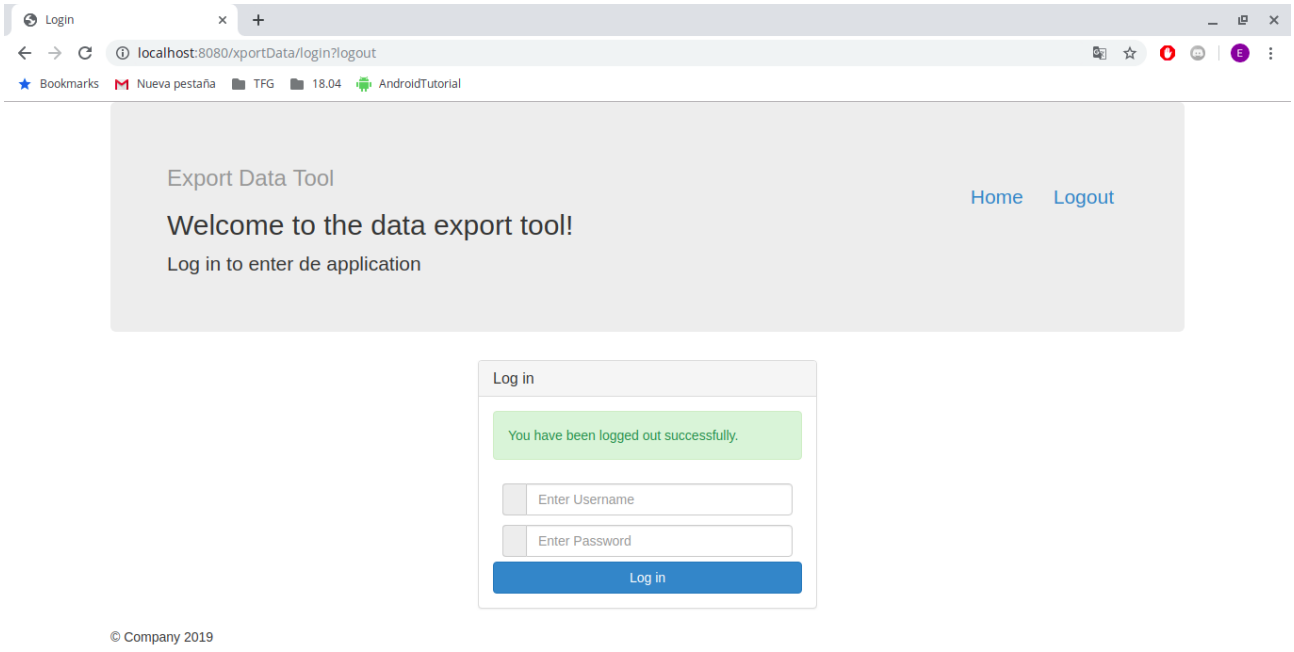


Figura 6-15. Efecto de hacer clic en Logout.

Si en esta página introdujésemos un par usuario-contraseña incorrectos obtendríamos lo siguiente:

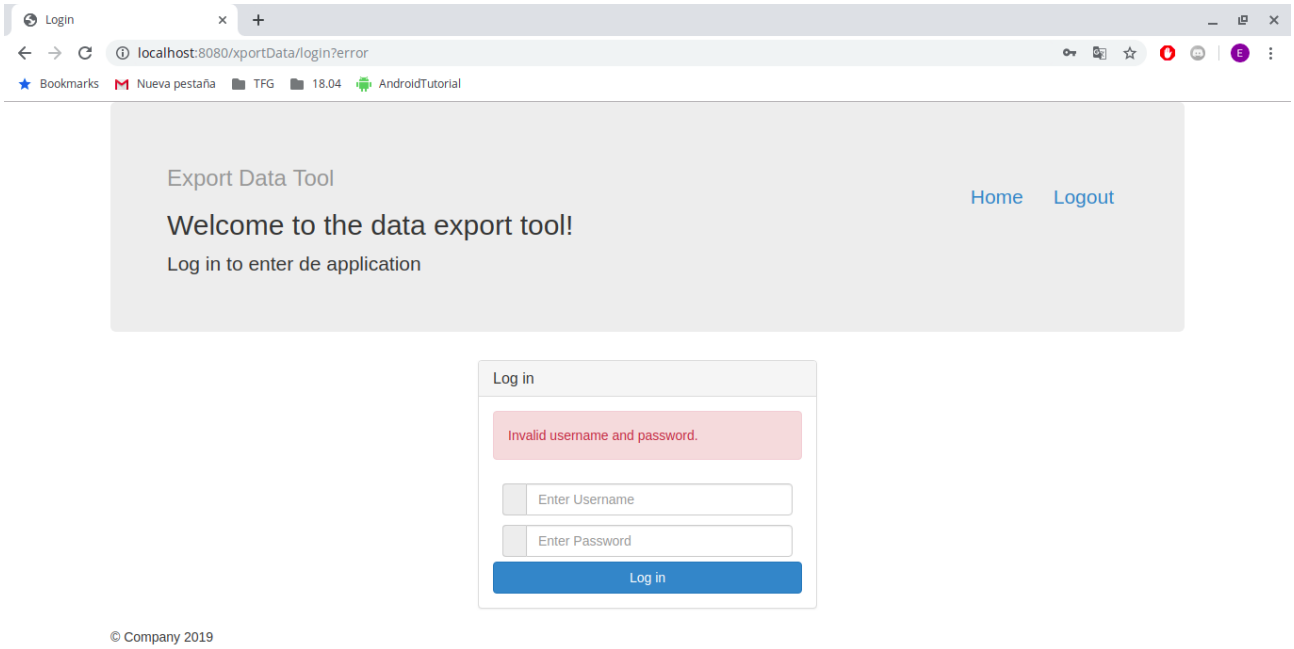


Figura 6-16. Log in incorrecto.

Por último, mostrar como en la carpeta "/*logs*" se ha generado un registro en un fichero de nombre la tabla de la cual hemos exportado con las operaciones realizadas:

```
ezequiel@ezequiel:~/logs$ ls
USUARIOS.txt
ezequiel@ezequiel:~/logs$ cat USUARIOS.txt
Thu Aug 22 18:30:34 CEST 2019: Columna APELLIDO2 de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: APELLIDO. Tratamiento realizado: Anonimiza
cion de indice aleatorio.
Thu Aug 22 18:33:09 CEST 2019: Columna NAME de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: NOMBRE. Tratamiento realizado: Anonimizacion de
 indice aleatorio.
Mon Sep 02 11:33:35 CEST 2019: Columna NAME de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: NOMBRE. Tratamiento realizado: Anonimizacion de
 indice aleatorio.
Tue Sep 03 10:12:56 CEST 2019: Columna NAME de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: NOMBRE. Tratamiento realizado: Anonimizacion de
 indice aleatorio.
Tue Sep 03 10:13:05 CEST 2019: Columna APELLIDO1 de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: APELLIDO. Tratamiento realizado: Anonimiza
cion de indice aleatorio.
Wed Sep 04 17:07:19 CEST 2019: Columna NAME de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: NOMBRE. Tratamiento realizado: Anonimizacion de
 indice aleatorio.
Wed Sep 04 17:07:31 CEST 2019: Columna APELLIDO1 de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: APELLIDO. Tratamiento realizado: Anonimiza
cion de indice aleatorio.
Wed Sep 04 17:07:40 CEST 2019: Columna APELLIDO2 de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: APELLIDO. Tratamiento realizado: Anonimiza
cion de indice aleatorio.
Wed Sep 04 17:07:57 CEST 2019: Columna DNI de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: IDENTIFICACION. Tratamiento realizado: Desplazam
iento de digitos bidireccional con clave autogenerada.
Wed Sep 04 17:08:30 CEST 2019: Columna CUENTABANCARIA de caracter personal(true) y de caracter sensible(false). Tipo aplicable RGPD: CUENTABANCARIA. Tratamiento realizado
: Generacion aleatoria con adecuacion al estandar.
```

Figura 6-17. Registro generado.

7 RESULTADOS

Como forma de concluir esta memoria, se procede a la realización de un análisis general de los objetivos a cumplir como medida para el éxito de este trabajo y como medio para obtener las conclusiones y posibles líneas futuras de ampliación del alcance de este proyecto.

Dicho esto, y comenzando con los objetivos definidos en el capítulo 2, se puede afirmar que **dentro del marco definidos por las consideraciones iniciales de diseño e implementación se ha cumplido con todos los objetivos.**

Siendo esto así la herramienta de exportación de datos proporciona al usuario en forma de aplicación web una herramienta visual y sencilla para llevar datos de un entorno productivo a otro realizando de por medio transformaciones de seudonimización/anonimización de forma transparente al usuario.

Estas transformaciones, siempre y cuando el usuario realice un uso digno de la aplicación, garantiza la legalidad de las operaciones realizadas tanto en la exportación como en el tratamiento que se desee dar a posteriori cualesquiera que sean los fines.

Todo esto mientras, también de forma automática y transparente, se va generando un registro de actividades al cual el responsable debe dar accesibilidad en formato electrónico, para la autoridad de control que lo solicite, y persistencia en formato físico.

Una vez establecido esto, es posible concluir de la misma manera que la **forma de mejorar la herramienta** de exportación no iría en la dirección de mejorar las funcionalidades, que ya cumplen con sus objetivos, sino de **definir más objetivos que den lugar a nuevas funcionalidades** que aporten calidad a la aplicación de cara al usuario.

Estas **posibles líneas futuras** de ampliación de ampliación del alcance podrían ser, entre otras:

La posibilidad de **configurar mediante un archivo XML o json la conexión a la base de datos** (tipo de base de datos, dirección IP, puerto, usuario…), **los diferentes usuarios y contraseñas con sus permisos asociados para el login, etc**.

Por otro lado, también sería interesante la implementación de **una lectura "inteligente"** de las columnas de la base de datos eliminando la restricción impuesta al inicio de la implementación de tener que conocer el tipo con el que el dato se encuentra almacenado en la base de datos. De esta **forma no solo los datos almacenados como VARCHAR podrían ser transformados y anonimizados, sino cualquier en general.**

Por último, otra utilidad interesante para el responsable de datos sería la posibilidad de **no realizar transformaciones a los datos columna por columna sino en conjunto.** Ejemplo claro de esto sería no trabajar con las columnas de nombre, primer apellido y segundo apellido por separado sino en conjunto aplicando una misma norma a las transformaciones.

REFERENCIAS

[1] Ley Orgánica 3/2018, de 5 de diciembre, de Protección de datos Personales y Garantía de los Derechos Digitales. Boletín Oficial del Estado, 294, de 6 de diciembre de 2018, 119788 a 119857. Recuperado de https://www.boe.es/eli/es/lo/2018/12/05/3.

[2] Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo de 27 de abril, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos. Recuperado de https://www.boe.es/doue/2016/119/L00001-00088.pdf.

[3] Baeldung (2018). The state of java in 2018. Recuperado el 1 de Septiembre de 2019, de https://www.baeldung.com/java-in-2018

[4] Jrebel (2016). Spring vs. Java EE, what is more popular? Recuperado el 1 de Septiembre de 2019, de https://jrebel.com/rebellabs/spring-vs-java-ee-survey-results/

[5] Whizlabs (2019). Top 5 Java Frameworks. Recuperado el 1 de Septiembre de 2019, de https://www.whizlabs.com/blog/top-java-frameworks/

[6] Finoit (2019). 7 Popular Frameworks Java Development Companies Use in 2019. Recuperado el 1 de Septiembre de 2019, de https://www.finoit.com/blog/7-popular-java-frameworks-2019/

[7] JetBrains (2019). Java. Recuperado el 1 de Septiembre de 2019, de https://www.jetbrains.com/lp/devecosystem-2019/java/?gclid=CjwKCAjwtajrBRBVEiwA8w2Q8LzKGaoPdPklm7V7423aGrzmb6Kh_u6NClIZtzH-S6RLajKnSarNoxoCGgQQAvD_BwE&gclsrc=aw.ds

[8] Apache (2019). Announce mailing list archives. Recuperado el 1 de septiembre de 2019, de http://mail-archives.apache.org/mod_mbox/www-announce/201901.mbox/%3c1546417225.1572488.1623318168.13AD2ADC@webmail.messagingengine.com%3e

[9] StackOverflow (2019). Pros and Cons of Apache Tiles framework. Recuperado el 1 de Septiembre de 2019, de https://stackoverflow.com/questions/14473817/pros-and-cons-of-apache-tiles-framework

[10] OSTraining (2018). How Popular is Bootstrap in 2018?. Recuperado el 1 de Septiembre de 2019, de https://www.ostraining.com/blog/webdesign/bootstrap-popular/

[11] Ganeshan, A. (2016). Spring MVC. Recuperado de https://usermanual.wiki/Pdf/1springmvcbeginnersguide2nd.1041009060.pdf

GLOSARIO

MVC: Modelo Vista Controlador 13

HTTP: Hypertext Transfer Protocol 13

CRUD: Create, Read, Update and Delete 15

STS: Spring Tool Suite 20

RGPD: Reglamento General de Protección de Datos 19

UE: Unión Europea 19