GENERADOR DE FORMULARIOS WEB AUTOGESTIONABLES Y DE LISTADOS COMPLEJOS PARA OPERACIONES CRUD

MÁSTER DE INGENIERÍA DEL SOFTWARE, MÉTODOS FORMALES Y SISTEMAS DE INFORMACIÓN

Director de Máster: Juan Sánchez Díaz

Autor: Juan Manuel Ruíz Pons

Agradecimientos

Gracias a toda la gente que me ha apoyado en proseguir con mis estudios, animándome en los momentos difíciles, en especial a mi familia y mi pareja Alba, que han estado apoyándome en todo momento.

Tabla de contenido

1 Lista de figuras

2 Introducción

Hoy en día internet está presente en todas las compañías, ya sean públicas o privadas, hace pocos años una empresa fuese del tamaño que fuese, no disponía de la infraestructura en lo que las comunicaciones se refiere que dispone actualmente, internet abrió un potencial enorme de negocio para las empresas, toda la información que necesitan las empresas para operar día a día generalmente está almacenada en servidores accesibles únicamente a través de internet, un ejemplo a muy alto nivel es la ***Ilustración 1 - Representación de como almacenan los documentos las empresas en internet*** viendo cómo los ordenadores mandan infinidad de documentos a la red, quedándose las aplicaciones de sobremesa sin conexión a internet totalmente obsoletas, es por ello que las aplicaciones web son prácticamente lo único demandado a las consultoras informáticas, disponiendo la información que se requiera desde cualquier lugar y en cualquier momento, automatizando cada vez más los procesos de comunicación y producción que disponen, ya sea a menor o mayor escala dependiendo el tamaño de la empresa.

Ilustración 1 - Representación de como almacenan los documentos las empresas en internet

2.1 Planteamiento del problema

En la actualidad, la gran mayoría de las consultoras informáticas tienen un ámbito específico y muchos de sus proyectos tienen requisitos comunes. Debido al apretado tiempo que tienen para las entregas de los proyectos fomentan la reutilización de código de una manera inadecuada. El concepto ideal de la reutilización de código es aprovechar un trabajo anterior, economizando el tiempo. La manera más fácil y posiblemente la más utilizada de reutilizar código es copiarlo parcialmente desde un programa antiguo al proyecto que se está desarrollando, entrando en una dinámica de trabajo que fomenta el error y en la dispersión del código fuente, siendo más difícil de mantenerse ante futuros cambios.

La mayoría de los proyectos que tiene Capgemini Valencia son sistemas web, éstos son desarrollados en forma de página web, ganando homogeneidad y valor en el negocio. Uno de los puntos claves del por qué se realizan los sistemas web a través de páginas web y no de una aplicación de sobremesa con conexión a internet, es que no se liga a ningún tipo de plataforma en lo que a sistema operativo se refiere y no se necesita ningún tipo de software adicional para poder acceder al sistema web. Todos los sistemas web disponen de algún tipo de formulario y listado de datos, y muchos de ellos son proyectos CRUD (Create, Read, Update y Delete) o disponen de alguna parte CRUD. Para la creación o actualización de algún registro, suele realizarse a través de un formulario web, ya que nos permite introducir datos los cuales son enviados a un servidor para que sean procesados.

El problema que se nos plantea, es el abstraer los requisitos comunes de los proyectos estándar, y crear una nueva extensión sobre el Framework que utilizan para este tipo de proyectos, para poder suplir así los requisitos comunes de una manera eficiente y no realizar malas prácticas de reutilización, reduciendo los costes de desarrollo, mantenimiento y aumentando a su vez la calidad del producto ofrecido al cliente.

2.2 Solución propuesta

Todos los sistemas web desarrollados por Capgemini, a través del Framework Devon [1] que se explicarán con más detalle en el capítulo 4 El Framework Devon, disponen de formularios que permiten insertar datos o actualizar registros y/o listados para poder visualizar la información que se inserta o actualiza. El Framework con el que se desarrolla este tipo de proyectos facilita al programador la creación tanto del back-end, como del front-end. En el back-end, se da soporte a la creación de servicios web, así como en la definición de las dependencias que tienen las operaciones de negocio y

con la configuración estándar que se proporciona para Hibernate [14]. En el front-end, se ha creado una capa de abstracción sobre un Framework Java Script, llamado Ext-JS [8] explicado en el capítulo 4.2 Sencha Ext-JS, dando una configuración personalizada para la empresa, se optó por utilizar este Framework debido a que es el utilizado por los proyectos realizados con Devon y el módulo debía seguir con la línea de trabajo y los patrones de diseño que se utilizan en este tipo de proyectos.

Frente a la recolección de los requisitos comunes de los sistemas web que se ostentan en Devon, se obtuvo que la mayoría de ellos disponen de las funciones básicas en base de datos, en lo que en el mundo de la computación comúnmente se denomina CRUD, para realizar estas funcionalidades básicas, es necesario el uso de un formulario siendo el medio para poder añadir o actualizar un registro, y una listado de los registros para seleccionar el elemento que se quiere eliminar. Es por ello, por lo que se decidió abordar una extensión sobre Devon, que generase formularios y listados complejos Java Script de manera automática, facilitando al programador la creación de éstos, ofreciendo por un lado una serie de servicios web que gestionen de manera automática los formularios y por otro lado la conexión con terceros servicios para poder gestionar los registros ofrecidos en el listado. Para que el generador sea práctico debe disponer de los elementos básicos de un formulario, así como las validaciones correspondientes de sus campos tanto en el cliente como en el servidor, a su vez, estos formularios deben ser adaptables, permitiendo añadir y agrupar elementos requeridos por el formulario, permitiendo a los elementos externos interactuar con éste de una manera exacta a como si no se hubiese generado automáticamente, es por ello por lo que los formularios y listados generados se basan en Ext-JS.

Todos los elementos de un formulario web son parámetros de entrada, teniendo éstos un formato definido y un objetivo común. El objetivo común de todos ellos es recopilar de una manera correcta la información solicitada al usuario, por lo que existen diferentes tipos de componentes dentro de un formulario. En el punto 5.3.1.1 Estudio de formularios y 5.3.1.2 Estudio de listados complejos, se obtienen los elementos visuales comunes más utilizados en los proyectos Devon, para así poder realizar una extensión de Devon que genere elementos para la interfaz de usuario, siendo auto gestionables y centralizando los elementos comunes de los sistemas web, para que así sean mantenibles, de mayor calidad y menos costosos a la hora de ser desarrollados, solucionando así el problema propuesto en el punto anterior.

2.3 Estructura del documento

La estructura que se va a seguir en la tesis del máster está compuesta por ocho capítulos principales, incluyendo el capítulo introductorio en el que se sitúa este punto:

- 2 Introducción, se introduce al lector en el contexto de la tesis del máster, explicando la situación actual y cuál es la motivación para llevar a cabo este trabajo.
- 3 Estado del arte, en este capítulo se explica el contexto en que se sitúa el trabajo explicado en esta tesina de máster y se explican dos Frameworks que tienen un objetivo similar al trabajo final de esta memoria.
- 4 El Framework Devon, este capítulo define qué es Devon, analizando y reflexionando de un modo más minucioso sobre las tecnologías en las que se ostenta.
- 5 Módulo Devon Forms Web, en este capítulo se explica todo lo relacionado con la extensión creada Devon Forms Web, trata de: la arquitectura que ha seguido y las tecnologías que ha utilizado; los estudios realizados para llevar un cierto camino en la abstracción de los componentes; los servicios web que ofrece el módulo, así como el modelo de base de datos utilizado para la abstracción de formularios; las explicaciones de todas las anotaciones que se han creado en Devon Forms Web para la generación de formularios y listados complejos.
- 6 Creación desde cero de un formulario a través de Devon Forms Web, se explica cómo crear un formulario paso a paso, utilizando todas las anotaciones y algunos de los servicios que ofrece Devon Forms Web para la gestión de los formularios.
- 7 Creación desde cero de un listado complejo a través de Devon Forms Web, se explica cómo generar un listado a través de las anotaciones que ofrece Devon Forms Web para la creación y gestión de listados complejos.
- 8 Conclusiones y trabajo futuro, se exponen las conclusiones obtenidas tras la realización del trabajo y las posibles ampliaciones futuras.

3 Estado del arte

Como se ha mencionado anteriormente en el trabajo de fin de Máster se ha extendido la funcionalidad del Framework Devon de Capgemini (véase para detalles el capítulo 4 El Framework Devon).

El módulo resultante tras la realización de este trabajo de Fin de Máster, fue nombrado como Devon Forms Web y va dirigido para desarrolladores experimentados en proyectos basados en Devon. El objetivo del módulo por un lado, es la creación automática de listados que permiten realizar gestiones sobre los registros listados, vinculándose con servicios web ajenos al módulo y por otro lado, es la creación de formularios auto gestionables, disponiendo el mismo módulo de servicios web que permiten la gestión de los mismos sin que el desarrollador tenga que implementar ningún servicio, ni tabla en base de datos para poder gestionar el formulario. Devon Forms Web está basado en tecnologías que ofrece Devon, por lo que las configuraciones y las ventajas que ofrece Devon han facilitado el desarrollo del módulo, disponiendo de la seguridad que ofrecen las tecnologías y servicios encapsulados en Devon. A continuación comentaremos brevemente algunos Frameworks similares al nuestro.

3.1 Frameworks relacionados

En este punto se van a presentar dos Frameworks que están relacionados con el trabajo presentado en esta tesina de Máster:

- AutoCRUD [6]
- Yii Framework [7]

3.1.1 AutoCRUD

AutoCRUD es un Framework basado en Java, permitiendo diseñar esquemas de bases de datos MySQL, generando a raíz de este esquema una aplicación web completa, teniendo como objetivo el sistema la realización de operaciones CRUD sobre el esquema MySQL. La aplicación web generada, está basada en Spring MVC Framework [13] [4] y ofrece una clase de acceso por cada tabla declarada, permitiendo obtener y

guardar cada columna de la tabla, conectándose a la base de datos a través de un JDBC dinámico utilizando Spring JDBC Template [5].

Para que el sistema web sea generado correctamente se han de seguir ciertas reglas a la hora de crear el esquema de base de datos:

- Los nombres de las entidades han de ser en singular y no contener ningún guión.
- Las relaciones de muchos a muchos o de uno a muchos debe estar identificado como el nombre de las dos tablas que se quieren relacionadas y unidas por un guión bajo.
- El identificador de las tablas debe ser un autogenerado que pueda ser transformado a un entero en Java.

En futuras versiones se pretenden incorporar las siguientes características:

- Soporte para múltiples bases de datos, dando soporte a Oracle y PostgreSQL, además del soporte actual a MySQL.
- Soporte a tablas relacionadas, actualmente las operaciones CRUD son únicamente para tablas sin relaciones entre otras tablas.
- Uso de más entradas en los formularios, en la versión actual únicamente se generan formularios con entradas de texto plano.
- Incorporar Hibernate para abstraerse del motor de base de datos y así poder dar soporte a múltiples bases de datos.

Este proyecto queda limitado por ir adaptado a MySQL, no ligar las campos de entrada de los formularios con los datos que se almacenan en base de datos, así como la falta de configuración que ofrece a la hora de generar el formulario. Va dirigido a sistemas web muy sencillos que se limitan únicamente a insertar registros en tablas no relacionadas en MySQL.

3.1.2 Yii Framework

Yii Framework se autodefine como un Framework PHP rápido, seguro y profesional.

- Rápido: porque únicamente carga las funcionalidades que necesita el usuario, disponiendo de un potente soporte con la caché y diseñado explícitamente para funcionar de manera eficiente con AJAX.
- Seguro: ya que incluye validación en las entradas de los formularios, así como prevención en la inyección SQL y Cross-Site scripting.
- Profesional: porque ayuda a un desarrollo del código limpio y reusable, siguiendo el patrón Modelo Vista Controlador.

Yii Framework genera al usuario un sistema web básico a través de una línea de comando, incluyendo autentificación en el sistema web y un formulario de contacto, a raíz del sitio generado. El desarrollador debe crear el esquema de base de datos que dispondrá la aplicación web, generando en función al esquema de base de datos las tablas en código PHP funcional de una manera automática, simulando el acceso a base de datos a través de un modelo orientado a objetos. A través de una consola web, permite seleccionar que tablas quieres que dispongan de las operaciones básicas CRUD, generándote Yii un código PHP siguiendo el patrón de diseño Modelo Vista Controlador, ofreciendo todo lo necesario para poder gestionar las tablas, creando un formulario y listado con componentes básicos, para poder listar los datos de las tablas y gestionar los registros. Una vez llegado a este punto, el desarrollador debe personalizar el formulario creando el comportamiento deseado.

Yii Framework es un sistema mucho más completo que AutoCRUD, explicado en el punto anterior (3.1.1 AutoCRUD), permitiendo relacionar tablas entre ellas y ofreciendo un abanico más amplio de funcionalidades, nos ofrece una buena base para empezar un sistema web y ahorrar tiempo en funcionalidades que por defecto, suelen estar dotados todos los sistemas web.

4 El Framework Devon

Ilustración 2 - Logo Devon

Devon es desde hace más de cuatro años el Framework oficial de desarrollo Java de Capgemini España. Puede considerarse un Framework de aplicaciones, ya que dispone de un conjunto de componentes, organizados en servicios, construyendo un diseño reutilizable para la construcción de aplicaciones de negocio. Devon se sostiene en dos principios fundamentales:

- Clara separación entre lógica de negocio y lógica de presentación.
- La lógica de negocio se divide en servicios, con procedimientos de acceso y uso claramente definidos.

4.1 Arquitectura Software del Framework

Devon es un Framework Java basado en el estándar de mercado Java 2 Platform Enterprise Edition, utilizando "Spring Framework para las llamadas a la lógica de negocio e Hibernate" [16] [17] y Spring JDBC Template para el acceso a la base de datos. La seguridad de las aplicaciones creadas en Devon está basada en Spring Security y son por funcionalidad y roles, permitiendo vistas diferentes dependiendo el rol del usuario. Para la interfaz de usuario, incorpora el Framework Java Script Sencha Ext-JS, para aplicaciones de escritorio y móviles respectivamente, comunicándose con el servidor vía JSON [10] y disponiendo de una arquitectura en el cliente Modelo-Vista-Controlador, permitiendo diferencias por elementos, el manejo de los datos, la lógica y la interfaz de usuario.

Devon dispone de componentes para la aceleración del desarrollo software, a nivel de infraestructura, tiene preconfigurado: Spring, Hibernate, iBatis, EHCache, Spring Integration para eventos, Spring Webservices, JasperReports y la Seguridad (Autorización basada en IP y por usuarios en base de datos). Respecto al mundo web, dispone de una estructura base de proyecto, proporcionando acceso directo desde la presentación a las operaciones de negocio, sin el gasto añadido que supone un controlador; automatiza la transferencia de datos entre la Web, DTO y entidad. Los componentes del core de Devon, permiten la definición de operaciones de negocio y

operaciones asíncronas, así como la monitorización online de operaciones de negocio y gestionar la caché.

4.2 Sencha Ext-JS

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript [21]. Se presenta orientado a objetos, basado en prototipos (para simular la herencia), imperativo, tipado débilmente y dinámico, siendo la flexibilidad una de sus características más importantes. Se puede realizar el mismo trabajo de muchas maneras diferentes, pero esta característica viene con el inconveniente de no ser fácilmente previsible, al no disponer de una estructura unificada, el código JavaScript puede ser muy difícil de entender, mantener y reutilizar. Por otro lado la programación basada en clases, dispone de la programación orientada a objetos, la cual lleva consigo, un fuerte acoplamiento, proporcionando encapsulación y un estándar a la hora de codificar, siendo predecible, extensible y escalable a lo largo del tiempo; sin embargo, no dispone de la misma capacidad dinámica que encontramos en JavaScript. Cada aproximación tiene sus pros y contras, Ext-JS unifica ambas aproximaciones para disponer de todos los beneficios, mitigando a su vez los contras que trae cada lenguaje.

Sencha Ext-JS o simplemente Ext-JS, es un Framework JavaScript para el desarrollo de aplicaciones web robustas a nivel empresarial, teniendo alrededor de 1.000 APIs Java Script, "cientos de componentes reutilizables disponibles con una amplia documentación" [8] [11]. Entre ellos hay una amplia gama de componentes a nivel de interfaz de usuario, trayendo consigo a su vez, un paquete de datos que permite a los desarrolladores utilizar una arquitectura MVC (Modelo-Vista-Controlador) en la construcción de su aplicación [12], dando así, un nuevo nivel de interactividad en las aplicaciones web, permitiendo una separación en el cliente entre la gestión, la lógica y la interfaz de usuario, haciendo de esta manera que sea más fácil el desarrollo de una aplicación. Todo el patrón de diseño MVC se da en el lado del cliente, ofreciendo como modelo la colección de datos pudiendo ser ésta persistente en el cliente únicamente o estar sincronizada a su vez con un servicio web. A esta colección de datos se le denomina '*store*', invocando al servicio con los datos cada vez que se realiza algún tipo de cambio en el modelo de datos, por otro lado la vista es cualquier tipo de componente visual y

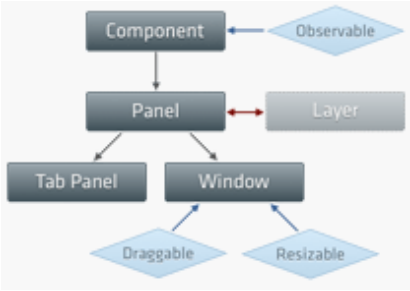

Ilustración 3 - Diseño de un componente en Ext JS

el controlador es el encargado de que la aplicación funcione y se pueda interactuar con ella.

Una de las características más importantes de Ext-JS es su diseño de componentes, pudiéndose ampliar fácilmente otros componentes predeterminados para satisfacer necesidades más específicas, encapsulándolo en un nuevo componente reutilizable en cualquier otro proyecto web.

Ext-JS permite a los desarrolladores que el sistema web funcione en una increíble gama de navegadores, sin tener que modificar nada de código para que se visualice correctamente en algún navegador en concreto, soportando concretamente los siguientes navegadores:

- Internet Explorer 6 o superior
- Firefox 3.6 o superior
- Safari 4 o superior
- Google Chrome 10 o superior
- Opera 11 o superior.

Ilustración 4 - Logo Sencha Ext JS

Ilustración 5 - Navegadores compatibles con Ext JS

4.3 Spring Framework

Para poder gestionar la lógica de negocio de las aplicaciones que están basadas en Devon, se utiliza Spring Framework, siguiendo un diseño modular permitiendo una adopción incremental de partes individuales dentro del proyecto y aprovechando lo que todo este proyecto incluye, como puede ser la inyección flexible de dependencias ya sea por archivos XML o por anotaciones [15]; teniendo un avanzado soporte para la programación orientada a aspectos basada en proxy; dando soporte a las transacciones declarativas y a proyectos de código abierto como es Hibernate; ofreciendo un marco flexible para “aplicaciones web RESTful” [3] con el patrón de diseño Modelo Vista Controlador.

5 Módulo Devon Forms Web

Devon Forms Web, es un nuevo módulo de Devon, teniendo como objetivo la creación de formularios web Java Script. La creación de éstos es posible enriqueciendo clases Java con anotaciones propias del módulo y de unos servicios web predefinidos para poder obtener los formularios y gestionar su contenido.

5.1 Arquitectura Devon Forms Web

Devon Forms Web se ostenta en la arquitectura de Devon, y al igual que él, en Java 2 Platform Enterprise Edition. Para poder recopilar las anotaciones propias del módulo, inspeccionando las anotaciones de las clases y ejecutando métodos de ellas, se utiliza Reflection. Reflection es una utilidad Java comúnmente utilizada por los programas que requieren la capacidad de examinar o modificar el comportamiento de ejecución de las aplicaciones que se ejecutan en la máquina virtual Java.
Para el acceso a base de datos se utiliza Hibernate, abstrayéndose así del motor de base de datos. Respecto a la tecnología utilizada para la creación de los formularios, siendo éstos homogéneos para el programador, se utiliza la misma tecnología que los proyectos estándar de Devon, Ext-JS. Todos los mensajes que muestra el módulo son configurables a través de un fichero de propiedades.

5.1.1 Abstracción de los formularios

Devon Forms Web dispone de una serie de servicios web y un modelo de datos para poder almacenar cualquier tipo de formulario generado por el mismo. En lo que respecta a los servicios web, se ofrecen los servicios necesarios para crear, actualizar, leer y borrar una versión de un formulario.

5.1.2 Modelo de datos

A lo que base de datos respecta, un formulario queda registrado como un documento versionado, por lo que un documento dispondrá de tantas versiones como veces se haya guardado, almacenando únicamente los campos del documento y anexos que hayan sufrido alguna modificación. Esto es posible gracias al diseño en base de datos, como se aprecia en la *Ilustración 6 - Modelo de datos utilizado en Devon Forms Web*, la raíz de todo documento es *DocInfo*, el cual dispone como mínimo de una versión que se creará la primera vez que se guarde el formulario hasta un número indefinido de veces, las cuales quedarán registradas en *DocInfoVersion*. Por otro lado, una versión

del formulario dispone de tantos segmentos como parámetros de entrada dispone el formulario almacenándose en la tabla *DocSegment* y de tantos anexos como adjunte el usuario al formulario, almacenándose la meta información del binario en la tabla *DocFileInfo* y el binario en si en *DocFileBin.*

DocInfo		
Nombre	**Tipo**	**Descripción**
id	Long	Identificador autogenerado del registro
mimeType	String	Descripción del tipo de documento

DocInfoVersion		
Nombre	**Tipo**	**Descripción**
id	Long	Identificador autogenerado del registro
version	Integer	Número de versión del documento
title	String	Título del formulario
fileName	String	Nombre asignado del formulario
lastModified	Long	Fecha de creación del documento, milisegundos desde Enero de 1970
modifiedBy	String	Identificador del usuario que creó la versión del documento

DocSegment		
Nombre	**Tipo**	**Descripción**
id	Long	Identificador autogenerado del registro
name	String	Nombre asignado al parámetro del formulario
mimeType	String	Indica de qué tipo es el parámetro de entrada del formulario (un parámetro de entrada de texto, un campo seleccionable)

content	String	Valor del parámetro de entrada
meta	String	Descripción del parámetro de entrada
hasCode	Integer	Identificador único del objeto java que representa el registro

DocFileInfo		
Nombre	**Tipo**	**Descripción**
id	Long	Identificador autogenerado del registro
fileName	String	Nombre del fichero adjunto
mimeType	String	Tipo del fichero adjunto
length	Long	Tamaño en bytes del fichero adjunto
hasCode	Integer	Identificador único del objeto java que representa el registro

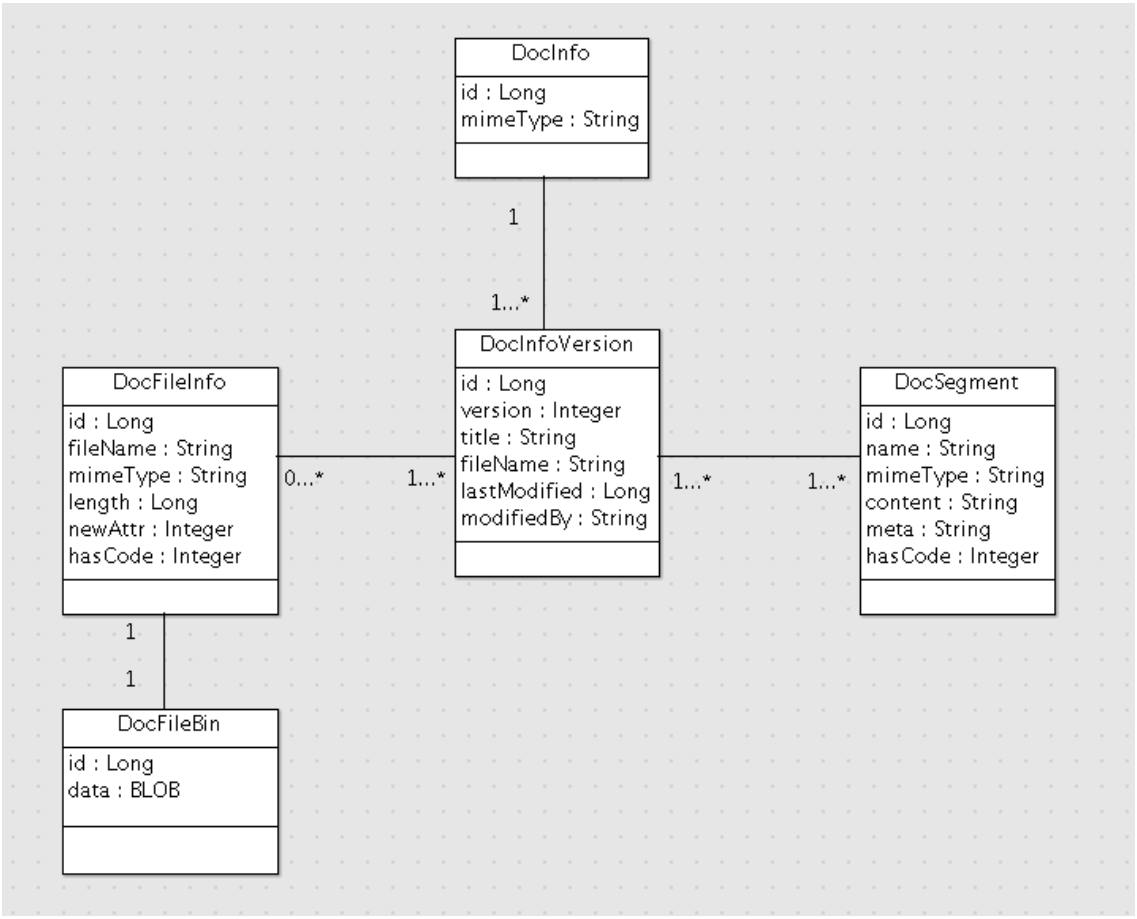

Ilustración 6 - Modelo de datos utilizado en Devon Forms Web

Para abstraernos del motor de base de datos y poder tratar como un objeto lo que hay o se va a almacenar en base de datos, se optó por utilizar Hibernate, de esta manera toda la gestión de la información se realiza mediantes objetos Java, por lo que cada tabla en base de datos representa la traducción al modelo relacional de una clase Java.

Un ejemplo práctico de cómo actuaría el modelo de datos frente a tres almacenamientos en base de datos sobre un mismo formulario que dispone de diez parámetros de entrada y permite adjuntar archivos adjuntos sería:

- La primera vez que el usuario va a guardar el formulario, ha rellenado siete de los diez campos que dispone el formulario y no ha adjuntando ningún archivo. En base de datos, se creará un registro en *DocInfo*, *DocInfoVersion* y diez registros en *DocSegment*, con el valor asignado en el formulario. En el caso de que los valores no tengan valor, el *content* del *DocSegment* será nulo.

- La segunda vez que el usuario anexa un archivo adjunto, esto supone un guardado automático en base de datos, creando así un registro en DocInfoVersion, DocFileInfo y DocFileBin. DocInfo dispondrá de una nueva referencia a la versión creada y la versión creada dispondrá diez referencias a los campos de la versión anterior, ya que éstos no han sufrido modificación alguna.
- La tercera vez que el usuario guarda el formulario, ha modificado dos de los campos que había rellenado la primera vez y completa un nuevo parámetro de entrada. En base de datos queda reflejado como la creación de una nueva versión del formulario con tres nuevos segmentos y los segmentos que no han sido modificados, como el anexo, hacen referencia a la versión anterior.

5.1.3 EXT-JS en Devon Forms Web

En este punto se explica cómo se ha utilizado y que ventajas ofrece Ext JS en el módulo Devon Forms Web. Que ExtJS disponga del patrón Modelo Vista Controlador completamente en el cliente es una gran ventaja, la principal ventaja de este patrón es la separación de capas. Devon Forms Web, necesitaba este patrón para poder abstraerse y crear capas genéricas que funcionasen con cualquier tipo de formulario o listado.

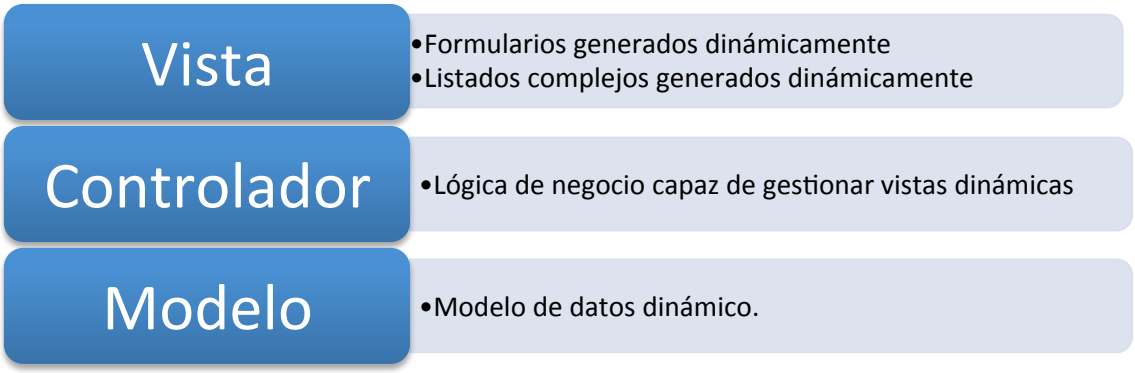

Ilustración 7 - Abstracción MVC adaptada a Devon Forms Web

La figura anterior es una abstracción de cómo Devon Forms Web gestiona los formularios en el cliente. Todas las capas son generadas en el servidor en tiempo de compilación y se proveen al navegador web. La vista es la interfaz gráfica que se mostrará al usuario, el controlador es el encargado de sincronizar el modelo de datos con la vista y el modelo de datos está configurado para que sea síncrono. Para que esto sea funcional, tiene configurado un proxy asociando las inserciones, actualizaciones y borrados que se realicen sobre el modelo de datos con una serie de

servicios web, de este modo cuando el controlador realice una operación sobre el modelo de datos, automáticamente se ejecutara el servicio web que tiene asociado dicha operación.

5.1.4 Interpretación de formularios y listados

Para la interpretación de los formularios y listados, Devon Forms Web procesa el formulario o listado y lo convierte en una estructura Java en forma de árbol. Los elementos que forman este árbol, son de dos tipos: Elementos base, que representan un elemento en el formulario o listado; Elementos contenedores, representando una agrupación de elementos base y elementos contenedores, debiendo existir como mínimo un elemento de los dos. En un árbol que representa un formulario, podrán existir los dos tipos de elementos mencionados anteriormente de forma conjunta. En cambio, un árbol que representa un listado complejo, únicamente podrá contener elementos base, ya que en los listados no es posible la combinación de elementos.

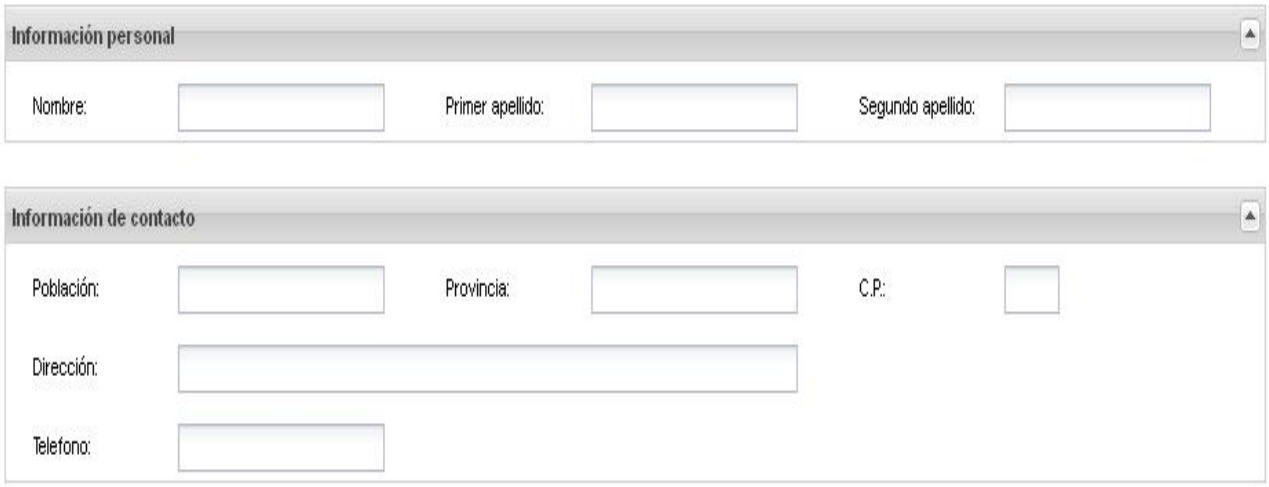

Ilustración 8 - Formulario de contacto web

En la *Ilustración 8 - Formulario de contacto web*, disponemos de un formulario que consta tanto de elementos contenedores como de elementos base, los elementos contenedores pueden ser visibles o no, en el formulario disponemos de dos elementos contenedores visibles y de dos no visibles para el usuario. Se denominan elementos visibles aquellos que son perceptibles por el usuario final, como aquellas agrupaciones que disponen de un título y que apreciamos en las secciones de 'Información personal' e 'Información de contacto'. Por otro lado, los elementos no visibles son aquellos que

el usuario final no percibe, pero son necesarios para que el formulario disponga de la estructura deseada. En el contexto de Devon Forms Web, los elementos contenedores no visibles son las agrupaciones que no disponen de título y aquellos elementos que están agrupados en una misma fila, por lo que dentro de la agrupación 'Información personal', los campos 'Nombre', 'Primer apellido' y 'Segundo apellido' estarán representados en el árbol Java por un elemento contenedor no visible. Lo mismo ocurre en la agrupación de campos 'Información de contacto' con los campos 'Población', 'Provincia' y 'C.P.'.

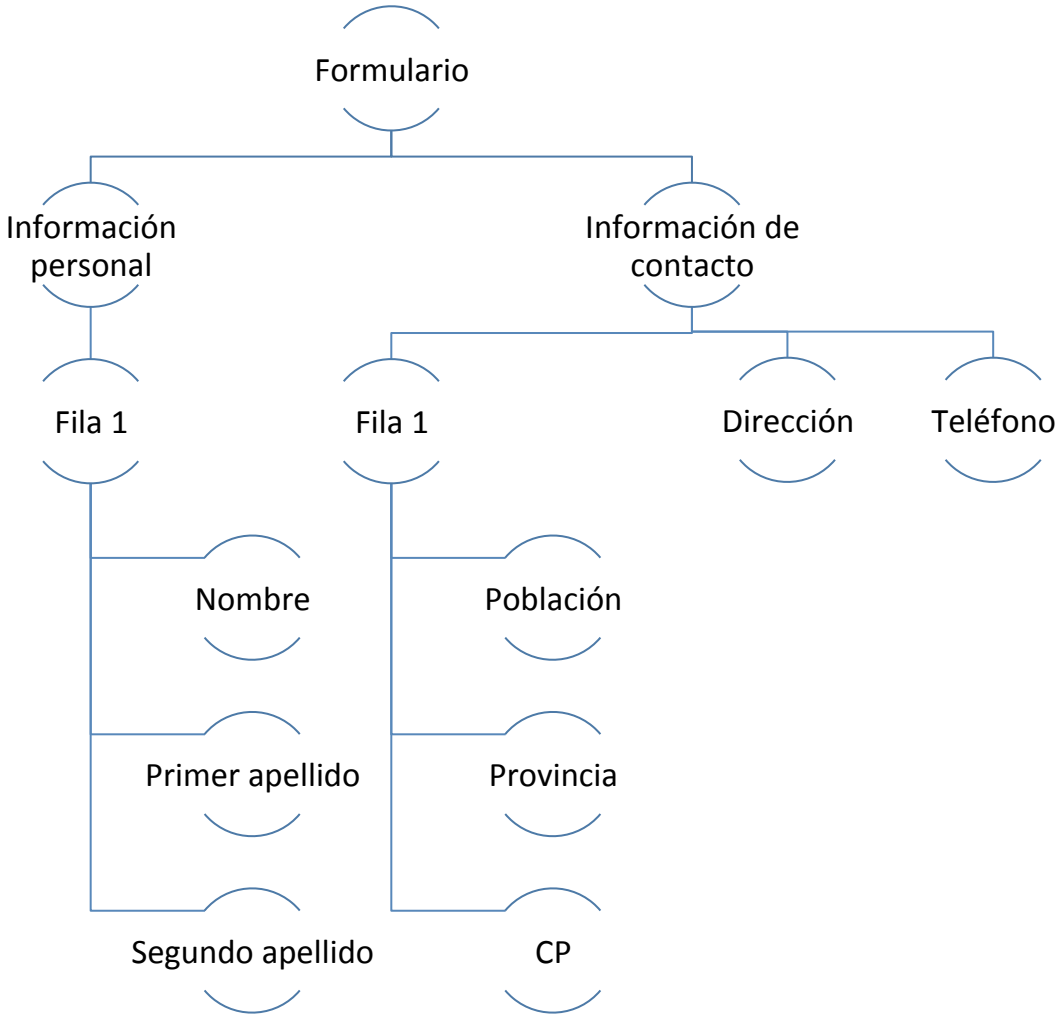

Ilustración 9 - Representación del formulario en forma de árbol

Únicamente serán elementos del formulario, aquellos nodos representados en la *Ilustración 9 - Representación del formulario en forma de árbol*, que no dispongan ningún nodo por debajo de él, todos los demás nodos sirven para agrupar elementos del formulario y que aparezcan de la manera deseada.

Una vez están estructurados los campos del formulario, deben ser transformados en código Java Script para que el navegador sea capaz de interpretarlo. Para que el módulo Devon Forms Web, fuese sostenible y fácil de mantenerse, se optó por utilizar StringTemplate [18] [19], el cual es un motor de plantillas Java para la generación de código fuente, páginas web o cualquier otra salida de texto formateado, como puede ser código Java Script. Muchos programas que emiten código fuente o cualquier otra salida de texto, lo hacen de manera no estructurada con instrucciones de impresión, fomentando el acoplamiento del código fuente del mismo programa con la salida de texto que se va a generar. El objetivo de utilizar StringTemplate, es definir formalmente una salida de texto, creando a través del motor de plantilla la generación de texto estructurado, teniendo visión a futuras ampliaciones sobre Devon Forms

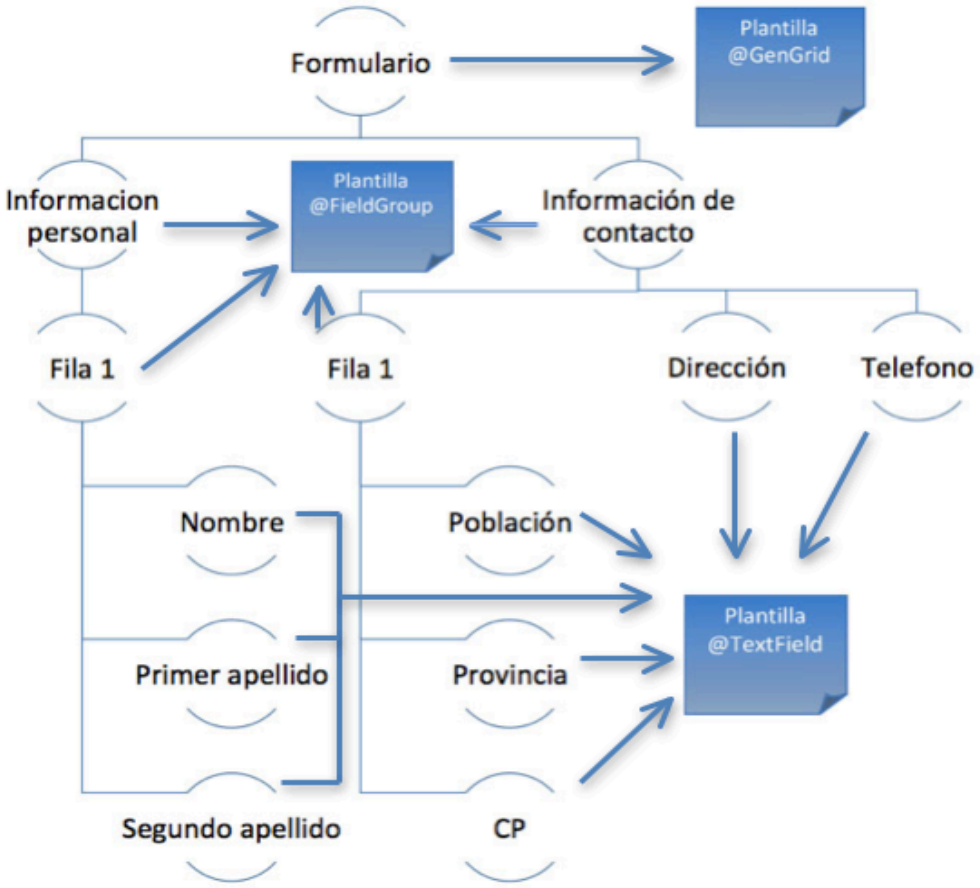

Ilustración 10 - Representación de como utiliza el árbol generado por el formulario las plantillas StringTemplate

Web.

La *Ilustración 10 - Representación de como utiliza el árbol generado por el formulario las plantillas StringTemplate*, es una representación de cómo se realiza la generación del código Java Script, cada tipo de elemento dispone de su propia plantilla, viéndolo desde un alto nivel, una vez se ha generado el árbol a través del formulario, sabiendo de qué tipo de objeto dispone en cada elemento del árbol es seleccionada una plantilla específica, pudiéndose distinguir plantillas que únicamente generan un tipo de componente, como otras que agrupan varios componentes a parte de crear el propio de la anotación. De esta manera, si se quisiese cambiar en cualquier momento la apariencia de alguno de los componentes por requisitos comunes de proyectos futuros, solamente se debería modificar la plantilla, creando una nueva versión del módulo con las nuevas plantillas.

5.2 Servicios Web

Devon Forms Web presenta una serie de servicios web para poder gestionar el formulario, la mayoría de los servicios web ofrecidos por el módulo son servicios web REST (Representational State Transfer) [2] creados a través de Devon, siendo una alternativa más simple y ligera que SOAP y servicios web basados en WSDL. Siguiendo cuatro principios de diseño fundamentales:

- Utiliza los métodos de http de manera explícita, siguiendo el protocolo definido por RFC 2616 [20], haciendo que las peticiones resulten consistentes con la definición del protocolo.
    - POST para crear un recurso en el servidor
    - GET para obtener un recurso
    - PUT para actualizar un recurso
    - DELETE para eliminar un recurso
- No mantienen el estado, siendo escalables para poder satisfacer una demanda en constante crecimiento. Un servicio web REST debe de recibir todos los datos necesarios para poder cumplir su objetivo, todos los servicios REST están orientados hacia la escalabilidad y un cliente no será capaz de distinguir si está realizando una petición directamente al servidor.
- Exponer URIs con forma de directorios, ocultando la tecnología usada en el servidor, siendo estáticas de manera que cuando cambie el recurso o la implementación del servicio, el enlace se mantenga igual.

- Transferencia de XML y/o JSON, reflejando el estado actual del objeto y sus atributos en el momento en que el cliente realiza una petición.

Todos los servicios web expuestos por Devon Forms Web y que requieren de alguna transferencia compuesta de datos utilizan JSON. JSON (JavaScript Object Notation o Notación de Objetos en JavaScript) es un formato ligero de intercambio de datos, basado en un subconjunto del lenguaje de programación JavaScript y el estándar ECMA-262 [21].

JSON está formado por dos estructuras:

- Una colección de pares de nombre/valor: en varios lenguajes de programación esto es conocido como un objeto, estructura o tabla hash.
- Una lista ordenada de valores: en la mayoría de los lenguajes de programación queda representado como un vector, lista o secuencia.

Un objeto JSON tiene el siguiente aspecto:

```
{
  "propiedad1":"valor1",
  "propiedad2":"valor2",
 "objeto1":{
        "propiedadDelObjeto1":"valorDelObjeto1",
        "propiedadDelObjeto2":"valorDelObjeto2",
        "propiedadDelObjeto2":"valorDelObjeto3"

}

 "array1":[
        "propiedadArray1":"valorArray1",
        "propiedadArray1":"valorArray1",
        "propiedadArray1":"valorArray1"

                ]
}
```

Todos los servicios que tratan con objetos que no contengan ningún tipo de binario están creados a través de Devon, en cambio los servicios web que tratan archivos binarios están creados a través de anotaciones Spring MVC, creando servicios RESTful, configurando un contenedor Servlet. Para el programador es totalmente transparente y disponen del mismo camino para acceder al servicio.

5.2.1 Servicios Web Devon Forms Web

A continuación se explican los servicios web ofrecidos por el módulo Devon Forms Web encargados de mantener los formularios, de base se tomará como nombre del host *localhost* y el puerto donde está publicada nuestra aplicación el *8181*.

Antes de todo, se explicarán los objetos Java que se utilizan para transmitir la información desde el cliente al servidor y viceversa. Este tipo de objetos son conocidos como DTOs (Data Transfer Object u Objeto para la Transferencia de Datos) y su función principal es la de transferir datos entre subsistemas. Para comunicarse con el cliente que realiza las peticiones, se optó por utilizar una objeto compuesto y de este modo seguir un patrón en el que se tratan siempre estructuras complejas, siendo más fáciles de refactorizar y escalar.

Response

Todos los DTOs extienden de esta clase, la cual dispone de un atributo que determina si la petición ha ido bien o no (*response*) y de un mensaje detallando el error en caso de que la petición haya sido fallida.

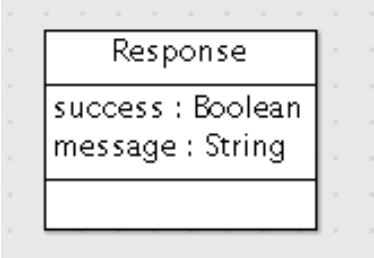

Ilustración 11 – Estructura Response DTO

IDHolder

Este objeto es el proporcionado a los formularios cuando se crea un registro en base de datos e identifica de manera unívoca al formulario, disponiendo únicamente de un atributo que es el identificador del formulario (*id*)

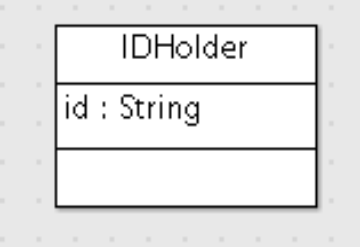

Ilustración 12 - Estructura IDHolder DTO

VersionDocInfo

Objeto que proporciona la información necesaria para listar todas las versiones de un formulario. Dispone del identificador del formulario (*id*), el número de versión (*version*) y de la fecha de creación del registro (*modified*).

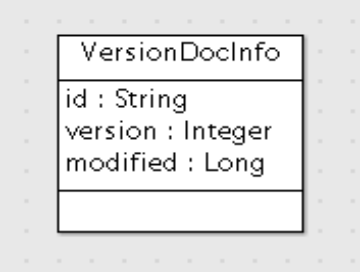

Ilustración 13 - Estructura VersionDocInfo DTO

DocInfo

DocInfo es el DTO que dispone de toda la información de una versión determinada de un formulario, trayendo consigo el identificador del formulario (*DocInfo.id*), el número de versión del formulario (*DocInfo.version*), un campo descriptivo indicando que se trata de un formulario generado (*mimeType*), un listado de segmentos (*DocInfo.docSegments*) y un listado de ficheros adjuntos (*DocInfo.docFileInfo*). Los segmentos (*DocSegment*) disponen de las propiedades básicas para que el formulario pueda disponer de los valores que fueron guardados, estando compuestos por un identificador (*id*), el título que tendrá asignado (*name*), de qué tipo es el segmento (*meta*), el valor que dispondrá el campo (*content*) y el formato del valor (*mimeType*). Por otro lado, respecto a los anexos del formulario (*DocFileInfo*), están dotados de un identificador (*id*), el nombre del fichero (*name*) y el tamaño del mismo (*length*).

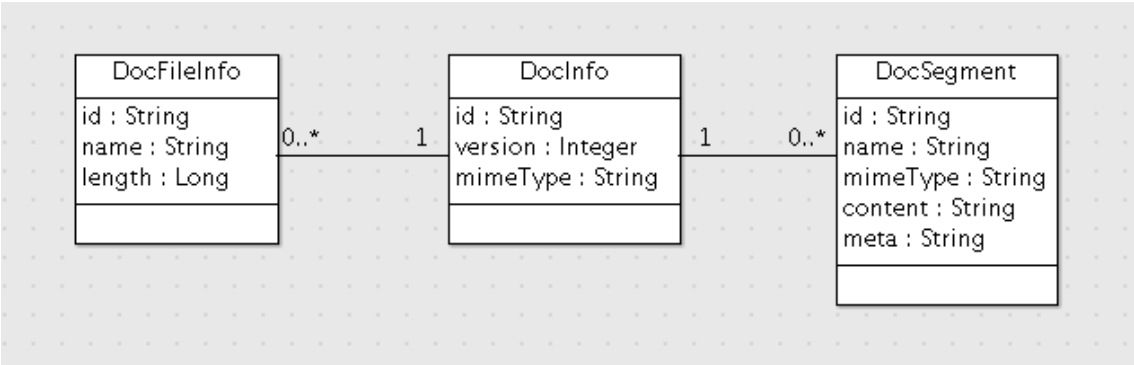

Ilustración 14 - Estructura DocInfo DTO

URL del servicio: *http://www.localhost:8181/devon/forms/newId*

Método HTTP: GET

Parámetros de entrada: -

Parámetro de salida: *IDHolder*

Descripción del servicio:

Servicio web para almacenar un formulario sin ningún campo de entrada ni anexo asociado, cuando se accede a la visualización del formulario, automáticamente se consume este servicio web guardando en base de datos un registro en *DocInfo*, devolviendo un identificador para el formulario y que los guardas del mismo, estén asociadas al registro de *DocInfo* que se ha creado.

URL del servicio: *http://www.localhost:8181/devon/forms/getVersions*

Método HTTP: GET

Parámetros de entrada: Texto identificativo del formulario

Parámetro de salida: *Collection<VersionDocInfo>*

Descripción del servicio:

Servicio web que devuelve todas las versiones de un formulario dado el identificador del registro *DocInfo*, cuando se accede al formulario por primera vez se le asigna automáticamente dicho identificador invocando al servicio *newId*.

URL del servicio: *http://www.localhost:8181/devon/forms/getDoc*

Método HTTP: GET

Parámetros de entrada: Texto identificativo de la versión del formulario

Parámetro de salida: *DocInfo*

Descripción del servicio:

Servicio web que devuelve el formulario asociado al identificador recibido como parámetro de entrada. El identificador dado, es el asociado a una versión concreta del formulario por defecto, el módulo Devon Forms Web dispone de una vista aparte con todas las versiones del formulario, pudiendo cargar cada una de ellas gracias a este servicio.

URL del servicio: *http://www.localhost:8181/devon/forms/getFiles*

Método HTTP: GET

Parámetros de entrada: Texto identificativo de la versión del formulario

Parámetro de salida: *Collection<DocFileInfo>*

Descripción del servicio:

Servicio web que devuelve la meta información de todos los ficheros adjuntos a una versión del formulario.

URL del servicio: *http://www.localhost:8181/devon/forms/saveDoc*

Método HTTP: POST

Parámetros de entrada: DocInfoParams (Objeto estructurado con los datos del formulario)

Parámetro de salida: *IDHolder*

Descripción del servicio:

Servicio web que se encarga de guardar el formulario como una versión. *DocInfoParams* dispone de toda la información necesaria para poder guardar el formulario con todos los parámetros de entrada, ya tengan valor o no, únicamente se almacenarán aquellos que hayan sufrido alguna modificación, debido a que los que tengan el mismo valor que algún formulario de alguna versión anterior, se asociarán a la nueva versión. Para poder comparar los segmentos de las versiones queda guardado el *hashcode* del objeto Java que representa el campo de entrada del formulario.

URL del servicio: *http://www.localhost:8181/devon/forms/deleteDocVersion*

Método HTTP: DEL

Parámetros de entrada: Texto identificativo de la versión del formulario

Parámetro de salida: *Response*

Descripción del servicio:

Servicio web que elimina la versión de un formulario, eliminando los segmentos asociados y anexos a la versión en el caso de que no estén referenciados en ninguna otra versión distinta de un mismo formulario.

URL del servicio: *http://www.localhost:8181/devon/forms/deleteDoc*

Método HTTP: DEL

Parámetros de entrada: Texto identificativo del formulario

Parámetro de salida: *Response*

Descripción del servicio:

Servicio web que elimina todas las versiones de un formulario, eliminando a su vez los segmentos y anexos asociados al mismo.

URL del servicio: *http://www.localhost:8181/devon/forms/deleteFile*

Método HTTP: DEL

Parámetros de entrada: Texto identificativo del fichero adjunto al formulario

Parámetro de salida: *Response*

Descripción del servicio:

Servicio web que elimina el fichero correspondiente al identificador dado para todas las versiones del formulario, eliminando a su vez el binario del fichero. Este servicio por

defecto no está accesible para el control de versiones, pero se da acceso al programador para poder hacer uso del mismo.

URL del servicio: *http://www.localhost:8181/devon/forms/getFile*

Método http: GET

Parámetros de entrada: Identificador del fichero anexo al formulario

Parámetro de salida: Archivo binario

Descripción del servicio:

Servicio web que devuelve el binario del fichero adjunto, permitiendo descargar el fichero que se adjunto al formulario.

URL del servicio: *http://www.localhost:8181/devon/forms/uploadFile*

Método http: POST

Parámetros de entrada: multipart / form-data – Por una parte se dispone de una estructura con meta información del fichero y por otra el binario del fichero.

Parámetro de salida: *IDHolder*

Descripción del servicio:

Servicio web que almacena el fichero adjunto al formulario, creando una nueva versión del formulario.

5.3 Componentes Devon Forms Web

Devon Forms Web se caracteriza por la generación de formularios (*@GenForm*) y de rejillas compuestas (*@GenGrid*). Los formularios son gestionados completamente por el generador, desde la creación de la vista en el cliente como la gestión de los valores introducidos en base de datos. En cambio, las rejillas son gestionadas automática y únicamente en el cliente, aunque se conectan de manera automática al servicio especificado para el rellenado de los datos. Debe ser implementado de manera propia,

ya que los datos que se van a mostrar no pueden abstraerse y suelen estar generados por procesos de terceros.

5.3.1 Estudio del contexto

El objetivo principal de Devon Forms Web es la creación de formularios estándar auto gestionables y de rejillas/listados complejos los cuales permiten al usuario poder crear registros, actualizarlos y eliminarlos, es por ello que se realiza un estudio de cómo son los formularios y listados complejos en distintos proyectos los cuales habían sido desarrollados por Capgemini, viendo que tenían en común y en que diferían cada uno de ellos. También se tuvo en cuenta la opinión de diferentes programadores experimentados que habían participado en proyectos que disponían de estos elementos, viendo cuales eran los requisitos principales que debían de disponer los formularios y listados ofrecidos al cliente final.

5.3.1.1 Estudio de formularios

Uno de los dos objetivos principales de Devon Forms Web es la creación de formularios estándar, por lo que se decidió incluir en ella los componentes más comunes de los formularios aunando las diferencias que existían entre ellos para poder permitir una configuración estándar y a la vez configurable.

En el análisis de estructuración de un formulario, se pudo obtener que la mayoría de ellos agrupan elementos que están relacionados entre sí, teniendo este conjunto un título que describe la agrupación y la opción de poder ocultar cada sección del formulario.

En lo que respecta a los elementos del formulario, los más comunes fueron la entrada de texto libre ya sea en una línea o en varias, un campo seleccionable estilo *combobox*, y campos seleccionables tanto excluyentes como no excluyentes entre ellos. Sobre los elementos del formulario existen diferencias comunes entre cómo se mostraban cada uno de ellos, en lo que respecta a los parámetros de entrada de texto, difiere en donde se muestra el título del campo, estando a la izquierda o encima de éste, el tamaño y la posición del mismo. De todos los elementos de los formularios se incorporaron al módulo Devon Forms Web aquellos parámetros de entrada que se entendían como estándar dentro de los proyectos Devon.

- Un campo de texto que permite introducir una única línea de texto.
- Un campo de texto que permite introducir varias líneas de texto.

- Un parámetro de entrada que permite seleccionar una fecha determinada.
- Un campo seleccionable.
- Una casilla de selección la cual permite seleccionar únicamente una opción de las propuestas.
- Una casilla de selección que permite seleccionar varias opciones de las propuestas.
- Un editor de texto enriquecido.

En lo que respecta a requisitos funcionales de un formulario, por un lado se obtuvo que todos ellos disponían de validación en los campos tanto en el navegador web evitando así envíos innecesarios como en el servidor para así evitar posibles vulnerabilidades, ya que las validaciones en un navegador son fáciles de ser saltadas enviando datos corruptos al servidor. Por otro lado, muchos formularios son guardados en base de datos sin tener que interactuar con terceras partes del sistema, por lo que se decidió crear una serie de servicios web que fuesen capaces de gestionar cualquier tipo de formulario generado a través de Devon Forms Web.

5.3.1.2 Estudio de listados complejos

El segundo objetivo principal de Devon Forms Web es la creación de listados complejos, es por ello que al igual que en los formularios, se realizó un estudio acerca de cómo eran los listados en los proyectos que utilizaba Devon como herramienta de desarrollo, para que así los listados generados sean lo más semejantes posibles a los que suelen ofrecerse al cliente final. Pocos eran los listados en los que se mostrase información que no fuese texto plano, por lo que se optó por crear en la versión inicial de Devon Forms Web, listados cuyos elementos únicamente fuese texto plano.

La gran mayoría de los listados complejos no se limitaban a mostrar la información, sino que se utilizaban como herramientas para gestionar la información permitiendo crear, actualizar y eliminar la información que se mostraba en el listado. Los listados podían considerarse listados con una vista maestro/detalle, ya que la información mostrada en el listado solía ser menor a la información que se mostraba en el formulario para modificar el registro, por lo que habitualmente la vista de modificación del registro representa el detalle del mismo. Debido a que los listados complejos no sólo se basaban en listar información, sino que también se encargaban de gestionar la información. Se optó por que el módulo Devon Forms Web diese soporte a esta gestión, generando formularios estándar.

5.3.2 @GenForm

Uno de los componentes principales es la anotación Genform, indica al contexto de la aplicación que las clases que contengan esta anotación serán transformadas en un formulario.

GenForm está compuesto de cuatro propiedades, las cuales permiten al programador dotarle de un identificador al formulario, agrupar los campos del formulario, permitir añadir adjuntos al formulario y dotarlo de un control de versiones.

- La primera se trata de la propiedad **name**, esta propiedad aparte de identificar al formulario, nos permite crear varias instancias de ella. Para conseguir esto le deberemos asignar tantos nombres como instancias de esta clase queramos. Por ejemplo, si quisiéramos tres instancias del formulario para que éste aparezca en tres sitios diferentes en nuestra aplicación, deberemos asignarle tres nombres:

```
name = "{identificador1, identificador2, identificador3}"
```

Para poder utilizar las tres clases que se generan, se deben importar cada uno de estos formularios, o invocar al servicio sin ningún parámetro de entrada.

- La segunda propiedad es **fieldGroup**, con ella conseguimos agrupar los parámetros de entrada del formulario. Esta propiedad dispone de un listado de @FieldGroup, el cual explicaremos con más detalle en el siguiente punto (@FieldGroup).

Añadiendo *FieldGroup* damos la opción de poder visualizar los campos del formulario en columnas.

- La tercera propiedad es **anexo,** se trata de un booleano el cual indicará si queremos añadir archivos adjuntos al formulario que por defecto, va sin anexo.
- La cuarta propiedad es **version,** se trata de un booleano el cual indicará si queremos dotar de un gestor de versiones al formulario generado, para poder ver los cambios realizados, por defecto, al igual que el anexo, no se añade el control de versiones.

Todos los formularios pueden ser guardados en base de datos, por un lado se dispone la función Java Script *guardarDoc,* la cual tiene tres parámetros de entrada: el primero es una función Java Script, que será ejecutada al terminar de guardar el formulario en base de datos y recibir la respuesta positiva del servidor. El segundo parámetro, es un booleano que indica si se quiere obtener los formularios con los archivos adjuntos

(nunca se traen los formularios adjuntos, ya que sería una carga innecesaria para la red, simplemente se obtiene la información necesaria para poder obtenerlos individualmente, en el caso de querer visualizar alguno de ellos). El tercer y último parámetro, indica el identificador del formulario; la función se encarga de obtener todos los datos del formulario; validar los parámetros de entrada a través de las expresiones regulares que se especifican cuando se está creando el formulario; transformar el formulario en un objeto entendible para el servicio web que ofrece el módulo Devon Forms Web. Cuando se almacena un formulario queda registrado en base de datos como una versión del formulario, ofreciendo así la posibilidad de tener un listado con todos los cambios que se han producido. El gestor de versiones es totalmente transparente para el programador, tanto como si se quiere o no hacer uso de él, no deberá de realizar ningún tipo de trabajo añadido, ya que es un proceso transparente. El servicio que se encarga de almacenar el formulario, ejecuta el método 'validate()' (si existe) de la clase donde se ha definido el formulario, de esta manera se permite la validación de los campos en el servidor, el método debe tener como parámetro de salida un objeto propio del módulo del tipo 'ValidationResult', el cual está compuesto por un booleano que indica si la validación ha sido correcta y un texto descriptivo que se mostrará al usuario indicándole si el formulario ha sido guardado correctamente o el motivo de por qué no ha sido guardado. En caso de que los datos introducidos en el formulario no sean validos, el formulario no se guardará y se mostrará el texto devuelto por el método 'validate'. En el caso contrario, en que los campos si dispongan del formato correcto, el formulario se guardará creando una nueva versión del formulario y se mostrará el texto que ha devuelto el método. En caso de no disponer de una validación en el servidor, al guardar el formulario se mostrará un texto por defecto.

A la hora de trabajar con un formulario, se puede mostrar únicamente el formulario y gestionarlo de manera personalizada u obtener una vista en la cual aparte del formulario, ya vienen definidos el botón de guardado, en el caso de haberse especificado en el formulario, el listado de las versiones guardadas y/o el gestor de documentos anexados. Cada documento anexado al formulario, se guarda automáticamente en base de datos y se crea a su vez, una nueva versión del formulario, ya que ha supuesto un guardado en base de datos. Lo mismo ocurre cuando se elimina un documento anexado, se elimina la referencia de ese documento para la versión que se ha guardado, pero el documento no se elimina de la base de datos.

5.3.2.1 @FieldGroup

FieldGroup es una anotación que se utiliza como parámetro de entrada dentro del atributo 'FieldGroup' de la anotación @GenForm. El objetivo es la creación de un contenedor de entradas del formulario, para así poder agruparlas según el objetivo que tenga cada una. Todas las agrupaciones tienen la posibilidad para el usuario de poder ocultarlas dejando únicamente visible el título y un botón para desplegar de nuevo el contenedor. Cuando el usuario pliega/oculta el contenedor, todas las entradas que están fuera del contenedor y por debajo de él, se ajustan al nuevo espacio que se ha dejado y viceversa. FieldGroup dispone de cuatro propiedades:

- Name: Campo obligatorio que indica el identificador del FieldGroup.
- Title: Título que aparecerá en la agrupación de los campos.
- Order: Orden de aparición del FieldGroup en el formulario, en caso de no especificar un orden, el orden de aparición será el introducido en el listado.
- Border: Indica si se quiere borde en el FieldGroup (en caso de que no aparezca borde, el título del FieldGroup no se visualizará). Por defecto tiene un valor a true.

5.3.2.2 @TextField

TextField es una anotación que representa un parámetro de entrada de texto en un formulario, teniendo que estar asociada a propiedades de la clase que representen un formulario, por lo que la clase deberá de disponer de las anotaciones *@Component* y *@GenForm*.

TextField dispone de siete propiedades las cuales lo convierten en un componente totalmente configurable. Para un mayor entendimiento del componente, vamos a agrupar las propiedades en dos tipos, las propiedades que configuran el propio componente y las propiedades que definen la posición del componente.

- Propiedades que configuran al componente:
  - **name**: Identificador del *TextField*, en caso de omitir este valor el identificador será el nombre de la variable (Se opte o no por darle valor a esta propiedad, el identificador será igual a Name+Identificador de la anotación Genform).
  - **size**: Tamaño total del *Textfield*, su valor por defecto es 150.
  - **title**: Texto que acompaña al *TextField.*

- **regExp:** Expresión regular que será evaluada en el lado del cliente.

- Propiedades que definen la posición del componente:
  - **belongsTo:** Indica el grupo del formulario (FieldGroup) al que va a pertenecer el componente. En caso de omitir esta propiedad, el registro ocupará una línea entera del formulario y no quedara agrupado con ningún otro componente.
  - **order:** Orden de aparición del *TextField* en el formulario. Este valor debe ser único ya que si hay dos componentes con el mismo orden, uno sustituirá al otro (únicamente pueden tener el mismo orden si un componente está dentro de un *FieldGroup* y otro está fuera, ya que en este caso están en niveles diferentes).
  - **column:** Este campo solamente se considera si el *TextField* pertenece a un *FieldGroup* y sirve para poder agrupar varios *TextField* en una misma fila. Esto se consigue poniendo la propiedad column de distintos *TextField* con el mismo valor.

5.3.2.3 @DateField

DateField es una anotación que se encarga de crear entradas en el formulario las cuales representan fechas. Se diferencia de la propiedad @TextField en que no se puede escribir en ella (únicamente se podrá seleccionar una fecha de un calendario emergente que aparecerá al seleccionar el campo) y en algunas de sus propiedades de configuración que están más orientadas (mas) hacia el contexto de fechas. Exceptuando la expresión regular (propiedad *regExp*) , @DateField dispone de todas las propiedades de configuración de la anotación @TextField, además de disponer de unas propiedades enfocadas a las fechas:

- initialValue: Esta propiedad está orientada al calendario que surgirá de manera emergente una vez se quiera seleccionar un valor sobre el campo. Por ello, en esta propiedad se especifica el valor que tomará inicialmente el calendario emergente, puede añadirse una fecha estática o una expresión Java Script que determine la fecha de manera dinámica. Por defecto el valor es la fecha actual.
- from: Determina cual es la fecha mínima que puede tomar el campo. El pop over que muestra las fecha no dará opción a seleccionar una fecha menor a la indicada, al igual que initialValue puede tomar un valor estático o un valor dinámico a través de una expresión Java Script. Por defecto el valor es la fecha actual menos cien años.

- to: Determina cual es la fecha máxima que puede tomar el campo. El pop over que muestra la fecha, no dará opción a seleccionar una fecha mayor a la indicada, pudiendo tomar un valor estático o dinámico a través de una expresión Java Script. Por defecto el valor de la propiedad es la fecha actual más 20 años.
- format: Determina el patrón que seguirá el formato de la fecha (MMddYYYY, MM-dd-YYYY, etc.), por defecto no se toma ningún patrón y siguiendo con el estándar de java el valor esperado es un número que representa los milisegundos pasados desde el 1 de Enero de 1970.
- noBiggerThan: Esta propiedad determina que la fecha seleccionada no sea mayor que una fecha dada, puede indicarse una función Java Script o el identificador de otra fecha que esté en el formulario. Por defecto esta propiedad es vacía.
- NoBiggerThanMessage: Esta propiedad mostrará un mensaje al usuario cuando no se ha cumplido con la validación de la propiedad 'noBiggerThan', debe ser un texto que indique al usuario el por qué no ha cumplido con la validación. Por defecto el valor de esta propiedad es 'la fecha introducida deber ser menor a {la fecha que disponga la propiedad noBiggerThan}'.
- noSmallerThan: Esta propiedad indica lo contrario que la propiedad 'noBiggerThan', por lo que se validará que la fecha que tome el campo no sea menor a la fecha indicada en la propiedad. Al igual que 'noBiggerThan', puede tomar una función Java Script o el identificador de otra fecha en el formulario, por defecto esta propiedad no tiene valor.
- NoSmallerThanMessage: Esta propiedad mostrará un mensaje al usuario cuando no se ha cumplido con la validación de la propiedad 'noSmallerThan', debe ser un texto que indique al usuario el por qué no ha cumplido con la validación. Por defecto el valor de esta propiedad es 'la fecha introducida deber ser menor a {*la fecha que disponga la propiedad noSmallerThan*}'.

Carece de sentido indicar un valor estático a las propiedades 'noBiggerThan' y 'noSmallerThan', ya que si no se quiere tomar un valor más grande o más pequeño de una manera estática puede conseguirse dotándole de un valor a la propiedad 'from' y 'to'. Vamos a ver dos posibles escenarios en los que los valores de las propiedades 'noSmallerThan' y 'noBiggerThan' tienen formato diferente:

1. Supongamos que disponemos de dos parámetros de entrada que solicitan fechas, y dependiendo de la época del año en que nos encontremos, la diferencia entre los campos debe ser de tres días si no es invierno y de cinco si

lo es. Podría ser el caso de una página de un hotel, donde la estancia mínima en depende de la temporada en la que nos encontremos, en este caso, deberíamos crear una función Java Script que haga lo indicado anteriormente.

2. El segundo escenario es más simple, en él seguimos teniendo dos fechas en el formulario identificadas como fechaInicio y fechaFin, por lo que la fecha con identificador fechaInicio no puede ser mayor a la fechaFin y la fechaFin no puede ser menor a la fechaInicio; por lo que la propiedad 'noBiggerThan' de la fecha de inicio dispondrá del identificador 'fechaFin' y la propiedad 'noSmallerThan' de la fecha de fin dispondrá del identificador 'fechaInicio'.

La anotación @DateField debe ir en atributo de una clase Java que sea de tipo *java.util.Date*, se optó por utilizar este tipo básico debido a que los atributos de la clase Java deben de ser consistentes con los atributos que van a ser generados y de esta manera la validación que puede realizar el programador en el método *validate()* de la clase Java explicado en el punto 5.3.2 @GenForm, es consistente con el tipo de campo que se está validando.

5.3.2.4 @ComboBox

Esta anotación convierte el atributo de la clase en un campo seleccionable estilo combobox. En lo que respeta a la configuración del atributo, tiene la misma configuración que el *TextField*, excepto en dos aspectos:

- No dispone de la opción de añadir una expresión regular para que el cliente realice una validación en el cliente.
- Se le pueden asociar datos a través de dos propiedades:
    - **items**: Si los datos que le queremos asociar son estáticos, se puede introducir una lista de texto plano.
    - **provider**: Si los datos están asociados a un origen de datos, se puede asociar un servicio web, el cual debe de tener como respuesta un objeto JSON con el siguiente formato:

      {results: *nº de ítems devueltos* **,options:** *Lista.toString()* **}**

      Cada registro de la *Lista* deber ser un String con el siguiente formato:

{codigo:'*Identificador del item***', descripcion:'***Texto que se mostrará en el combo***'}**

Por defecto, el campo que se selecciona en el combobox es el primer registro del listado.

5.3.2.5 @TextEditor

TextEditor es una anotación que representa un editor de texto enriquecido, en el que el usuario puede darle formato al texto introducido, así como una serie de opciones añadidas como puede ser: introducir la fecha actual, introducir hipervínculos a un texto o introducir imágenes y poder gestionarlas disponiendo de una biblioteca online de las fotos utilizadas. El editor de textos está basado en TinyMCE [9] una plataforma web independiente a Ext-JS, basado en Java Script, que es un sistema de código abierto con licencia GPL. TinyMCE convierte el texto introducido en HTML con el acrónimo WYSIWYG (What You See Is What You Get o Lo que ves es lo que obtienes). En la computación un editor con este acrónimo asegura que el contenido introducido y que ves en ese momento, será el mostrado en cualquier otro sistema, implicando una interfaz de usuario que permite al usuario ver algo muy similar al resultado final.

A lo que respecta a como se ha gestionado la inserción y gestión de imágenes en el editor de textos, Devon Forms Web gestiona las imágenes adjuntas como si se tratasen de documentos adjuntos al formulario. TinyMCE ya incorpora un visualizador de imágenes las cuales necesitan únicamente la URL donde se encuentra la imagen, por lo que cuando se inserta una imagen se le asocia cual será la URL de la imagen. Cada imagen insertada o eliminada del contenedor de imágenes supone una nueva versión en el formulario, por lo que si se insertasen dos imágenes dispondríamos de dos versiones nuevas en el formulario. Al borrar una de las dos imágenes dispondríamos de una tercera versión, aunque en la segunda versión del formulario dispondríamos de las dos imágenes, por lo que cuando se elimina una imagen únicamente supone la generación de una nueva versión sin la imagen eliminada, disponiendo de un historial de todos los cambios realizados con las imágenes siendo totalmente transparente para el usuario.

La anotación @TextEditor es tratada como un campo más del formulario, por lo que debe ir dentro de una clase Java que disponga de la anotación @GenForm y pudiendo incluir tantos formularios como uno desee. Respecto a la configuración que se le puede dar a esta anotación va dirigida a donde se mostrará el editor de texto:

- name: Identificador del editor de texto, en caso de omitir esta propiedad dispondrá del nombre de la variable que contiene la anotación @TextEditor.
- title: Título que se mostrará en la parte superior del editor de texto, si no se introduce ningún valor no dispondrá de ningún título.
- labelSize: Propiedad numérica que define el tamaño de la letra que se mostrará, por defecto el valor es de 12 puntos.
- width: Propiedad numérica que define el ancho del editor de texto, por defecto la anchura del editor es automática ajustándose al tamaño disponible en el formulario.
- height: Propiedad numérica que define el alto del editor de texto, por defecto la altura del editor es de 600 pixels.
- order: Propiedad numérica que indica el orden de aparición en el formulario. Esta propiedad al igual que en @TextField y @ComboBox debe de ser única respecto a todos los demás campos de entrada del formulario, a no ser que el editor de texto disponga de algún valor en la propiedad 'column'. En este caso, debe de ser única respecto a los campos que dispongan del mismo valor en dicha propiedad.
- column: Este campo se considera únicamente si el editor de texto pertenece a un *FieldGroup* agrupando varios editores en una misma fila, al igual que en las anotaciones *@TextEditor* y *@Combobox,* esto se consigue poniendo la propiedad 'column' de distintos *TextField* con el mismo valor.
- belongsTo: Identificador del grupo del formulario (FieldGroup) al que va a pertenecer el componente.

5.3.2.6 @CheckBoxGroup y @RadioButtonGroup

CheckBoxGroup y RadioButtonGroup son anotaciones que representan campos seleccionables. @CheckBoxGroup representa una agrupación de campos seleccionables con selección múltiple, esto quiere decir que se podrán seleccionar varios de ellos, en cambio @RadioButtonGroup representa una agrupación de campos seleccionables con selección única, por lo que sólo se podrá seleccionar una de las opciones ofrecidas. La única diferencia a la hora de declarar ambas propiedades en una clase Java, es que @CheckBoxGroup debe ir como anotación de un listado genérico de *String,* en cambio @RadioButtonGroup debe de ir precedido de un *String*. @CheckBoxGroup y @RadioButtonGroup, comparten la misma configuración ya que el

aspecto visual que ofrecen es prácticamente el mismo, diferenciándolos únicamente en que uno permite seleccionar varios elementos y el otro no:

- items: propiedad que indica los registros que dispondrá de la agrupación de campos seleccionables, cada registro de la propiedad deberá ser del tipo @CheckBox si la propiedad es de @CheckBoxGroup o @RadioButton si la propiedad se trata de la anotación @RadioButtonGroup.
- title: Título que dispondrá la agrupación de los campos, si no indicamos ningún título no aparecerá nada como cabecera de los campos seleccionables ni dispondrán de un aspecto visual que parece agrupar a todos ellos.
- itemsPerLine: Número de campos seleccionables que se quieren por línea, por lo que si la propiedad 'items' dispone de seis campos y la propiedad 'itemsPerLine' es igual a tres, el campo dispondrá de dos líneas con tres campos seleccionables cada una de ellas.
- minimunSelection: Esta propiedad es una validación que se realizará en el cliente antes de guardar el formulario. Indica cuantos campos se han de seleccionar como mínimo de los ofrecidos en la agrupación, en el caso de la anotación @RadioButtonGroup, esta propiedad sólo puede adquirir los valores cero o uno.

Al igual que las anotaciones de su mismo nivel como pueden ser @TextField y @ComboBox, las anotaciones @CheckBoxGroup y @RadioButtonGroup también disponen de las propiedades de configuración 'belongsTo', 'column' y 'order'.

5.3.2.7 @CheckBox y @RadioButton

@CheckBox y @RadioButton permiten crear campos seleccionables dentro de una agrupación de campos. Deberán introducirse dentro de la propiedad 'items' de las anotaciones @CheckBoxGroup o @RadioButtonGroup respectivamente, la cual determina la configuración visual de cómo se mostrará el campo seleccionable en el formulario, por lo que esta anotación únicamente configurará el identificador del campo que obtendrá en el formulario. La etiqueta que se visualizará en el formulario y valor interno del que dispondrá para mandar al servidor:

- name: Esta propiedad es de carácter obligatorio e indica el identificador que dispondrá el registro.
- title: Etiqueta que se mostrará al lado izquierdo del campo seleccionable.

- value: valor que se mandará al servidor cuando el usuario seleccione el campo.

Hay que remarcar que estas anotaciones (@CheckBox y @RadioButton) nunca se declararan por si solas a nivel de un atributo de la clase Java, siempre deben de ir dentro de la propiedad *items* de la anotación @CheckBoxGroup en el caso de que se trate de @CheckBox o de @RadioButtonGroup en el caso de ser @RadioButton.

5.3.3 @GenGrid

Devon Forms Web ofrece al usuario el generar rejillas/listados complejos, para poder realizar las operaciones básicas de mantenimiento sobre registros de base de datos. Una línea en el listado compuesto puede corresponderse con uno o varios registros en base de datos.

@GenGrid dispone de muchas propiedades iguales que @GenForm, haciéndolo totalmente configurable y permitiendo generar varios listados complejos mediante una única clase:

- provider: Servicio web que proporciona los datos al listado para poder completarlo de información y así permitirle realizar el mapeo correctamente. El servicio web debe de devolver un objeto que contenga los atributos con los mismos nombres que se declararon en la clase Java que representa el listado, aparte debe de estar contenido en la siguiente estructura:

  {results: *nº de ítems totales* **,values:** *Listado de objetos que están dentro del límite de la paginación indicada por el listado* **}**

  Respecto a la propiedad de la estructura, aclarar que *results* se refiere al número de registros que están disponibles para el listado complejo. No debe por que casar con el número de registros devueltos en la propiedad *values,* esta propiedad no podrá contener nunca más elementos de los que están especificados en la propiedad *pagination* de @GenGrid, siempre y cuando se haya especificado un valor mayor a cero en la propiedad. En lo que respecta a los parámetros de entrada que debe de contener el servicio web que proporcionará los datos al listado, en el caso de no disponer de paginación no se requiere ningún parámetro de entrada en el servicio. Por otra parte, si el listado dispone de paginación el servicio web debe disponer de dos parámetros de entradas formateados como numéricos enteros, los cuales recibirá por GET. El primero de ellos indica cuál es el número del último registro disponible en la página actual del listado, mientras que el segundo y último parámetro de entrada indica cuantos registros deben ser devueltos por el servicio web. Un

ejemplo de cómo debería de actuar el servicio web con paginación: si nos encontramos en la tercera página de un listado que dispone de una paginación de 20 registros y solicitamos los datos de la siguiente página, el primer parámetro de entrada sería 60 y el segundo parámetros sería 20, por lo que el servicio web debería devolver los registros del listado que ocupan de la posición 60 a la 80.

- name: Al igual que la propiedad *name* de *@GenForm* explicado en el punto 5.3.2 @GenForm, esta propiedad a parte de identificar al formulario, nos permite crear varias instancias de ella. Para conseguir esto, deberemos de asignar a la propiedad tantos nombres como instancias de esta clase queramos.
- store: Identificador que tendrá el *store* en ExtJS donde se almacenan los datos en el cliente, por defecto el identificador es 'store+nombre asociado a @Gengrid'.
- updateProvider: Servicio web que se asociará con la acción de actualizar un registro del listado, deberá de esperar como parámetro de entrada un objeto con los elementos visibles en el formulario de actualización del registro y el método http de petición será POST.
- addProvider: Servicio web que se asociará con la acción de insertar un nuevo registro, en caso de no disponer de valor esta propiedad y sí disponer de valor la propiedad *updateProvider*. Se asociará el servicio de actualización de un registro con la acción de añadir un nuevo registro, en muchos escenarios el servicio de actualizar e insertar un registro suele ser el mismo.
- deleteProvider: Servicio web que se asociará con la acción de eliminar un registro, debido a que en muchos escenarios se requiere más de un parámetro para el borrado del registro, el servicio web recibirá el identificador del registro, o bien el objeto que representa el registro del listado. Si el registro del listado que quiere ser eliminado dispone de una propiedad con el nombre 'id', esta propiedad será mandada al servicio para que sea eliminada. En cambio, si el registro no dispone de esa propiedad, todo el registro será mandado al servicio web.
- paginationLimit: Indica el número de líneas que se muestran por página en el listado, por defecto se mostraran 15 registros por página. En el caso de no querer que el listado disponga de paginación, el valor de la propiedad deberá de ser 0.

Cuando el formulario tiene valor en la propiedad *updateProvider* dispondrá de un formulario para poder actualizar el registro del listado. En el caso de disponer de valor

la propiedad *addProvider* se proporcionará otro formulario para poder insertar un nuevo registro, mientras que en la propiedad *deleteProvider* el listado dispondrá de un botón asociado con el servicio de borrado del registro, invocando al servicio y actualizando la información del listado. Los campos mostrados en el formulario de actualización o inserción de un registro dependerán de los atributos de la clase Java que dispongan de la propiedad @TextColumn, explicada en el siguiente punto. Todos los servicios web asociados a las acciones de mantenimiento del listado, no deben de disponer del dominio del servicio ya que es algo que viene implícito en las aplicaciones web realizadas con Devon, por lo que si un servicio web está publicado por ejemplo en *localhost:8080/serviciosWeb/añadirRegistro*, el servicio web introducido deberá ser '*servicioWeb/añadirRegistro*'.

Todos los servicios web relacionados con la gestión del registro, ya sea borrado, actualización o inserción de un nuevo registro, deberán de devolver como respuesta el objeto Java *Response* propio de Devon Forms Web y explicado en el punto 5.2.1 Servicios Web Devon Forms Web. Recibiendo un valor positivo en la propiedad *success* en el caso de que todo haya ido bien y un valor negativo con un mensaje asociado en la propiedad *message* en el caso de que el servicio web no haya cumplido su cometido.

5.3.3.1 @TextColumn

Esta anotación complementa a @GenGrid y sin ella el listado carecería de sentido, ya que se encarga de indicar las columnas de las que dispondrá el listado de datos. A raíz de @TextColumn se genera la rejilla y el modelo de datos correspondiente en Java Script, el cual se encarga de sincronizarse con el listado y mostrar los datos que están en el modelo. Respecto a la configuración de la anotación, está relacionada con la posición y la cabecera que se mostrará dentro del listado complejo y si el campo será utilizado para actualizar el registro:

- name: Como en todas las anotaciones explicadas, la propiedad name es el identificador del campo y en caso de omitirlo tomará como valor el nombre de la variable Java.
- header: El texto descriptivo con el cual se asociará a la cabecera del campo. En caso de omitir esta propiedad, el campo no aparecerá en el listado, pero si se tendrá en cuenta en el *store*. Los casos en que se añade una variable Java con la propiedad @TextColumn y no dispone de *header*, suele estar relacionado con casos en los que se necesita ese atributo para identificar el registro sin pretender mostrarlo en el listado.

- editable: Propiedad booleana que indica si el campo al que pertenece la anotación será utilizado en el formulario que se genera dinámicamente para actualizar el registro. En caso de que se evalúe a *false* la propiedad, el campo no aparecerá en el formulario de actualización pero si en el de inserción; por defecto el valor de la propiedad es true.
- visible: Propiedad booleana que indica si el campo al que pertenece la anotación será utilizado para la inserción y actualización del registro; por defecto el valor de la propiedad es true.
- order: Campo numérico que indica el orden de aparición en el listado; por defecto el orden es en el que se declaran las variables en la clase Java.
- size: Propiedad que indica el tamaño de la columna dentro del listado, puede representarse a nivel de pixels (px) o de porcentaje (%), la propiedad debe ser de tipo texto ya que se debe de especificar el tipo de medida que se utilizará para el tamaño; por defecto la propiedad es 'auto' poniendo todas las columnas con el mismo ancho.

5.3.3.2 @Filter

Esta anotación permite que los campos del listado puedan ser filtrados y únicamente filtrará aquellos campos que tengan la anotación *@Filter* y *@TextColumn*, generando un formulario en la parte superior del listado que tendrá como objetivo mostrar los datos en el listado que estén dentro de los valores incluidos en los campos de entrada del formulario de filtrado. Todos los campos de entrada del formulario serán de entrada de texto libre, el titulo del campo que acompaña al parámetro de entrada estará situado a la izquierda del campo y su valor será el mismo del título que tiene el encabezado de la columna, pudiendo configurar únicamente la posición de los campos en el formulario a través de las anotaciones 'order' y 'column'.

- order: Propiedad numérica que indica el orden de aparición del campo en el formulario de búsqueda.
- column: Propiedad numérica que indica en qué fila se quiere situar el campo.

Por defecto, cada uno de los campos del formulario de búsqueda ocuparan una fila. El filtrado que se realiza sobre el listado es local, esto quiere decir que no tiene una llamada al servidor para que devuelva los datos que cumplen con la búsqueda establecida, sino que lo realiza sobre el modelo de datos (*Store*) que tiene en el navegador web. Para que esto no sea así y se quiera que los datos mostrados en el

listado sean proporcionados por un servicio web, deberemos dotar a la clase Java de la anotación *@FilterProvider* explicada en el punto siguiente.

5.3.3.3 @FilterProvider

Es una anotación que se declara a nivel de clase, por lo que se encontrará en el mismo nivel que @GenGrid. El objetivo de esta anotación es el dotar al formulario de búsqueda del listado de un enlace con el servidor para que el filtrado en vez de ser local sea a través del servicio web que se indica en la anotación.

- provider: Servicio web que se encargará del filtrado de los datos, esta propiedad es de carácter obligatorio y no debe disponer del dominio del servicio ya que es algo que viene implícito en las aplicaciones web realizadas con Devon. De este modo, si un servicio web está publicado por ejemplo en *localhost:8080/serviciosWeb/filtrarListado*, el servicio web introducido deberá ser '*servicioWeb/filtarListado*'.

El servicio deberá tener las mismas reglas que el servicio que provea los datos sin filtrar al listado explicado en el punto 5.3.3 @GenGrid. De esta manera, si el servicio web que provee los datos al listado soporta la paginación, el servicio de búsqueda también debe de soportar la paginación y del mismo modo el objeto devuelto debe ser el mismo en los dos servicios (en la mayoría de los casos el servicio que devuelve los datos sin filtrar y con filtrado suele ser el mismo).

5.3.4 Personalización en los formularios de creación/actualización en @GenGrid

Cuando se quiere añadir o actualizar un registro del listado generado por la anotación @GenGrid, Devon Forms Web crea un formulario para la gestión del registro con campos de entrada de texto simple si se quiere personalizar la apariencia del formulario y definir otro tipo de campos que no sean de texto, como puede ser un campo seleccionable o simplemente se quiere cambiar la apariencia del formulario para agrupar campos o el tamaño de los mismos, puede hacerse uso de las anotaciones propias de @GenForm. Para que las anotaciones propias del formulario tengan efecto sobre éste, deberán de situarse junto a la anotación @TextColumn, de esta manera se está diciendo al generador, que el registro que representa la columna del listado en el formulario de creación y actualización será gestionado por el programador y no adoptará el tipo de campo que asigna el generador por defecto. Puede optarse por modificar únicamente algunos campos del formulario y dejar el

resto sin ninguna anotación propia del formulario y que sea el generador el encargado de gestionarlo, pero se ha de tener cuidado con ello, ya que puede darse con incongruencias y que no se genere el formulario deseado. Esto puede suceder si el programador hace un mal uso de las anotaciones sin tener en cuenta que el generador añade por un nivel inferior anotaciones sobre los campos. Para que se entienda lo explicado se enseña un ejemplo práctico de las anotaciones que crea el generador en el nivel inferior, en primer lugar creamos un listado que disponga de seis campos que nos mostrará la información personal de los usuarios que tenemos registrados en el sistema.

```
@Component
@GenGrid
public class ListadoUsuariosRegistrados {

    @TextColumn(header = "Nombre")
    public String nombre;
    @TextColumn(header = "Primer apellido")
    public String primerApellido;
    @TextColumn(header = "Segundo apellido")
    public String segundoApellido;
    @TextColumn(header = "Teléfono")
    public String telefono;
    @TextColumn(header = "Población")
    public String poblacion;
    @TextColumn(header = "Provincia")
    public String provincia;
}
```

Se ha dejado la configuración por defecto de todas las propiedades excepto de la propiedad *header,* ya que queremos que todas las propiedades de la clase Java salgan en el listado. Cuando el generador procesa esta clase, internamente trata los atributos que no disponen de ninguna propiedad de @GenForm, como si dispusiesen de la anotación @TextField con la configuración que viene por defecto, modificando únicamente la propiedad *title* de la anotación y dotándole del valor que disponga la propiedad *header* de la anotación @TextColumn. Es por esto que Devon Forms Web a

la hora de generar el listado y el formulario para la gestión del mismo, interpretaría la siguiente clase.

```
@Component
@GenGrid
public class ListadoUsuariosRegistrados {


    @TextField(title = "Nombre")
    @TextColumn(header = "Nombre")
    public String nombre;
    @TextField(title = "Primer apellido")
    @TextColumn(header = "Primer apellido")
    public String primerApellido;
    @TextField(title = "Segundo apellido")
    @TextColumn(header = "Segundo apellido")
    public String segundoApellido;
    @TextField(title = "Teléfono")
    @TextColumn(header = "Teléfono")
    public String telefono;
    @TextField(title = "Población")
    @TextColumn(header = "Población")
    public String poblacion;
    @TextField(title = "Provincia")
    @TextColumn(header = "Provincia")
    public String provincia;
}
```

Al disponer de los valores por defecto, el primer valor de la clase tendrá en el formulario la propiedad de orden igual a uno, el segundo formulario igual a dos y así sucesivamente, por lo que si se decide añadir por cuenta ajena una anotación sobre los campos que tienen la anotación @TextColumn se ha de tener sumo cuidado, ya que si especificamos un orden que ha sido establecido implícitamente por el generador, se solapará y únicamente se visualizara el último campo que haya especificado el orden. De este modo, si se decide modificar el aspecto y/u orden de los campos, no se puede indicar un orden que ha obtenido implícitamente un valor anterior.

```
@Component
@GenGrid
public class ListadoUsuariosRegistrados {


    …
    // 4 Campos restantes
    …
  @TextField(title = "Población")
  @TextColumn(header = "Población")
  public String poblacion;
  @TextField(title = "Provincia", order= 5, size=300)
  @TextColumn(header = "Provincia")
  public String provincia;
}
```

En el ejemplo que tenemos, se ha modificado el orden y el tamaño a la propiedad *provincia* de la clase Java, indicándole un orden que tiene implícitamente la propiedad *población*, por lo que el campo *provincia* sobrescribirá al campo *población* dejando a éste oculto en el formulario. Es por esto que se recomienda al usuario utilizar en todos los atributos de la clase Java anotaciones de *@GenForm* y no únicamente en algunos atributos de la clase Java y en otros no, ya que se puede perder el control y el contexto del por qué no se está generando el formulario de la manera deseada.

6 Creación desde cero de un formulario a través de Devon Forms Web

En este apartado se va a explicar cómo generar formularios a través de una clase en Java, basándose en la extensión Devon Forms Web la cual esta enriquecida con anotaciones propias de la extensión.

En este ejemplo se va a crear un formulario de registro estándar, solicitándonos los datos típicos de registro (nombre, apellidos, teléfono, población…).

Primero definiremos una clase en Java con los parámetros de entrada que queremos que disponga el formulario, cada entrada del formulario debe de corresponderse con un atributo de la clase. Todos los atributos deben de ser de carácter público, ya que el proceso que se encarga de generar el formulario debe tener acceso a éstos.

```
public class FormularioRegistro {
    public String nombre;
    public String primerApellido;
    public String segundoApellido;
    public String telefono;
    public String cp;
    public String direccion;
    public String poblacion;
    public String provincia;
}
```

Una vez definida la clase, indicaremos que propiedades queremos que sean campos. Para ello debemos de dotar a los atributos de la clase con meta información, la cual será representada a través de anotaciones.

Primero se dotará de dos anotaciones a la clase:

@Component: Esta anotación perteneciente de Spring, indica que la clase va a ser un componente.

@GenForm: Esta anotación perteneciente de Devon Forms Web, indica que la clase va a ser un formulario.

Una vez la clase disponga de ambas anotaciones, se debe introducir meta información a las propiedades de la clase. En este ejemplo todos los componentes del formulario van a ser campos de entrada de texto simple.

> @TextField: Esta anotación propia de Devon Forms Web, indica que la propiedad de la clase, va a ser transformada en un campo libre de texto de una única línea.

Añadidas estas tres anotaciones, la clase pasará a tener la siguiente nomenclatura:

```
@Component
@GenForm
public class FormularioRegistro {

    @TextField(title = "Nombre")
    public String nombre;
    @TextField(title = "Primer apellido")
    public String primerApellido;
    @TextField(title = "Segundo apellido")
    public String segundoApellido;
    @TextField(title = "Teléfono")
    public String telefono;
    @TextField(title = "C.P.")
    public String cp;
    @TextField(title = "Dirección")
    public String direccion;
    @TextField(title = "Población")
    public String poblacion;
    @TextField(title = "Provincia")
    public String provincia;
}
```

Por defecto, el identificador del formulario es el nombre de la clase, así como los identificadores de los campos del formulario son los nombres de sus atributos. La

propiedad *title* de la anotación @TextField, indica que el texto que se mostrará al lado del campo en el formulario, en caso de omitirse, no aparecerá ningún texto.

Para poder utilizar la clase como un formulario, deberemos de realizar dos últimos pasos:

1. Importar la clase creada, de la siguiente manera:

```
<script type="text/javascript"
src="../public/renderJavascript.htm?id=FormularioRegistro
script/>
```

   Devon Forms Web proporciona un servicio el cual, dado el identificador del registro, nos devuelve el formulario en sí. En el caso de querer importar todos los formularios que se han creado automáticamente, basta con no indicar nada en el parámetro de entrada "id". En nuestro caso queremos importar únicamente el formulario de la clase es FormularioRegistro.java, por lo que el id=FormularioRegistro. Esto se determina a través de la anotación *GenForm,* su valor por defecto toma como identificador el nombre de clase, como ya se ha explicado anteriormente.

2. Una vez invocado al servicio para importar el formulario, podremos acceder al método java script que proporciona Devon Forms Web para instanciar y visualizar el formulario.

```
app.components.formdefs.FormularioRegistro ();
```

Cuando realicemos estos pasos dispondremos del siguiente formulario, el cual lo podremos incorporar donde queramos.

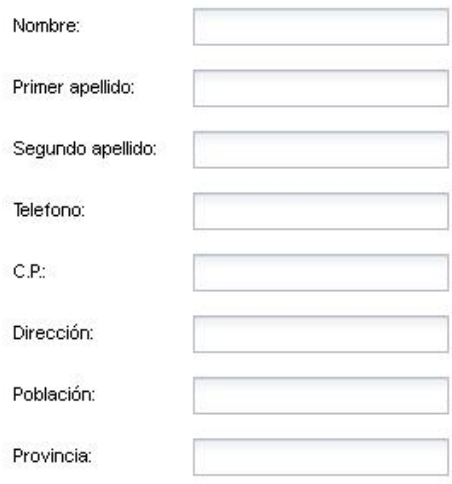

Ilustración 15 - Campos de un formulario generados automáticamente

6.1 Cómo utilizar @TextField y @FieldGroup

A continuación vamos a agrupar el formulario creado en el ejemplo anterior. Por una parte se pedirá la información personal y por otro lado la información de contacto. Para ello, deberemos crear dos *FieldGroup* dentro del *GenForm* y agrupar los *TextField*.

```
@Component

@GenForm(fieldGroup = { @FieldGroup(name = "groupPersonal", title =
"Información personal"), @FieldGroup(name = "groupContacto", title =
"Información de contacto") })

public class FormularioRegistro {


    @TextField(title = "Nombre", belongsTo = "groupPersonal", column = 1)

    public String nombre;


    @TextField(title = "Primer apellido", belongsTo = "groupPersonal", column
= 1)

    public String primerApellido;


    @TextField(title = "Segundo apellido", belongsTo = "groupPersonal", column
= 1)

    public String segundoApellido;


    @TextField(title = "Teléfono", belongsTo = "groupContacto")

    public String telefono;


    @TextField(title = "C.P.", belongsTo = "groupContacto", size = 40, order =
3, column = 1)

    public String cp;


    @TextField(title = "Dirección", belongsTo = "groupContacto", column = 2,
size = 450)

    public String direccion;


    @TextField(title = "Población", belongsTo = "groupContacto", order = 1,
column = 1)

    public String poblacion;
```

```
    @TextField(title = "Provincia", belongsTo = "groupContacto", order = 2,
column = 1)
    public String provincia;
}
```

Los grupos definidos se identificarán como 'groupContacto' y 'groupPersonal', asociados con el título 'Información personal' e 'Información de contacto' respectivamente. Si queremos que el orden de aparición sea el mismo orden en el que declaramos las variables no deberemos especificar nada. En el caso de los grupos no definimos ningún orden, ya que el orden deseado de aparición, es el mismo que se ha declarado en el listado. En cuanto a los campos que pertenecen al grupo relacionado con la información personal, al ser el orden deseado el mismo en que se han declarado en la clase Java no hace falta especificar ningún orden, únicamente los incluimos en la misma columna para que aparezcan todos ellos en una única fila. Respecto al grupo de información de contacto, únicamente especificamos un orden en la primera fila, debido a que existen varios elementos y el orden de aparición en el formulario deseado no es el mismo que en el que se han declarado las variables en la clase Java, cuando se especifica un orden y también se ha especificado a que columna pertenece el campo, el orden es relativo dentro de la columna.

También podemos aumentar o disminuir el tamaño del *TextField* como es el caso del campo *direccion* o *cp* mediante la propiedad *size*.

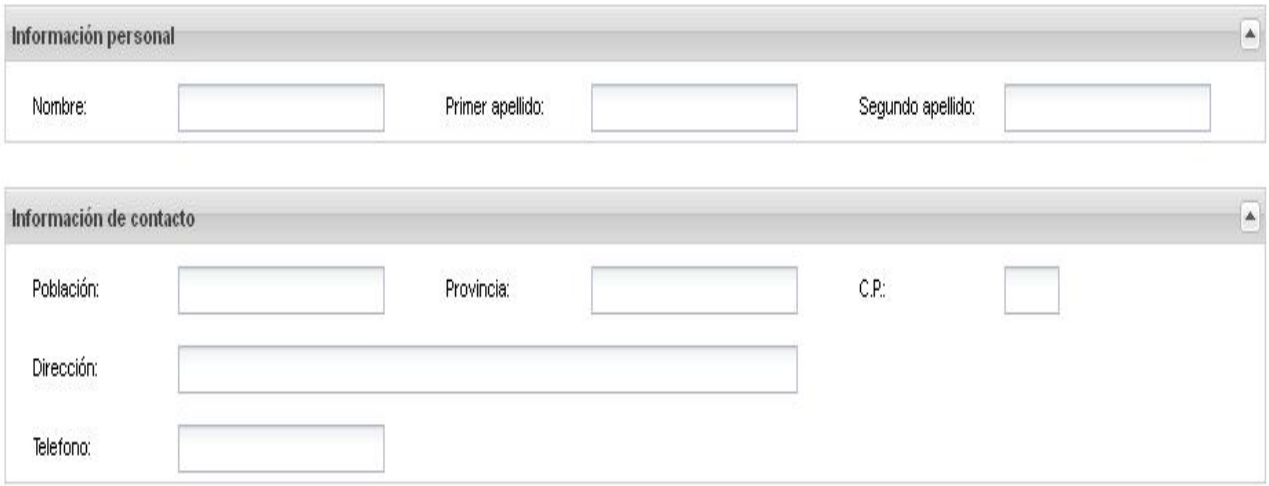

Ilustración 16 - Campos agrupados y generados automáticamente

Si quisiéramos guardar este formulario, *Devon Forms Web* ya incorpora un método Java Script que realiza dicha acción, se trata del método *guardarDoc,* el cual recibe tres parámetros, el primero se trata de *CallbackFunction (*función*)*, el segundo es un booleano que indica si queremos que se muestre un mensaje cuando se ha guardado el documento (si se quiere modificar el mensaje mostrado se deben de añadir a *appMessages.properties*, los mensajes *app.documentoGuardado* y *app.accionCompletada*), el último se trata del nombre del formulario que se quiere guardar (será el nombre que le hemos asignado en @GenForm). El método sería: *guardarDoc(null,true,'FormularioRegistro')*, el primer parámetro es un callback Java Script, en el caso de que quisiésemos que se ejecutase una función Java Script una vez se guarde el formulario, deberíamos de introducir como parámetro de entrada el nombre de la función, en este caso en particular, únicamente queremos que se guarde el formulario.

6.2 Como utilizar @CheckBoxGroup

@CheckBoxGroup supone una agrupación de campos que pueden ser seleccionados a través del chequeo de los mismos, es la única anotación que debe asociarse con un atributo de la clase Java que sea del tipo *List<String>* y no sea de tipo *String*, esto es debido a que pueden seleccionarse más de un campo y por consiguiente tomar más de un valor, como ya se explicó en el punto 5.3.2.6 @CheckBoxGroup y @RadioButtonGroup. En el ejemplo que se va a realizar, se añadirán tres nuevos campos al grupo 'Información personal' creado en el punto 6.1 Cómo utilizar @TextField y @FieldGroup, dando la opción de seleccionar qué aficiones tiene el usuario, como puede darse la opción de seleccionar más de una afición utilizaremos la anotación @CheckBoxGroup. En el caso que queramos que únicamente se quiera seleccionar una única opción de todas las mostradas, se debería de utilizar @RadioButtonGroup y el atributo de la clase Java debería ser de tipo *String*.

```
    @CheckBoxGroup(items = { @CheckBox(name = "informatica", value =
"informatica", title = "Informática"), @CheckBox (name = "deportes", value =
"deportes", title = "Deportes"), @CheckBox (name = "economia", value = "
economia", title = "Economía") }, belongsTo = "groupContacto", column = 2,
itemsPerLine = 3)

    public List<String> aficiones;
```

Por un lado indicamos que existirán tres campos a través de la anotación @CheckBox, este tipo de anotación siempre irá dentro de la propiedad *items* de la anotación @CheckBoxGroup. De las tres propiedades que se han indicado de la anotación @CheckBox, la primera de todas *name* especifica el identificador que tomará el campo Java Script generado, *value* es el valor que se mandará al servidor y que se añadirá al listado declarado en la clase Java y por último la propiedad *title* es el texto que se mostrará al usuario; respecto a las demás propiedades de la anotación @CheckBoxGroup añadimos como en el punto anterior 6.1 Cómo utilizar @TextField y @FieldGroup el identificador del grupo en la propiedad *belongsTo*, especificando que serán mostrados en la segunda fila y que se alojarán hasta tres registros en la fila, como únicamente se han declarado tres registros, todos los nuevo campos aparecerán agrupados en la misma fila; debido a que no se ha dejado el valor por defecto en la propiedad *minimumSelection* de la anotación @CheckBoxGroup no se obliga al usuario a seleccionar ninguno de los campos creados, si se quisiese que se seleccionasen por ejemplo dos de los tres campos creados la propiedad debería tener un valor de dos. Como se puede apreciar en la *Ilustración 17 - Campos creados a través de la anotación @CheckBoxGroup*, se han insertado tres campos en una única fila sin ningún título que pueden chequearse a través del ratón.

Ilustración 17 - Campos creados a través de la anotación @CheckBoxGroup

6.3 Cómo utilizar @ComboBox

En este punto se añade un campo seleccionable en el grupo de información personal definido en el punto anterior, el objetivo de este nuevo atributo es que el usuario pueda indicar el sexo al que pertenece, teniendo como opción 'Hombre' o 'Mujer'. En este caso la información que se muestra al usuario es estática por lo que rellenaremos la propiedad *items* con los valores deseados, en el caso de querer proveer al campo

seleccionable con datos dinámicos se deberá de introducir el servicio web que provee la información en la propiedad *provider*, devolviendo al servicio un objeto con la estructura explicada en el capitulo 5.3.2.4 @ComboBox, se quiere que el campo seleccionable aparezca en una fila debajo de los campos ya introducidos por lo que el valor de la propiedad *column* es igual a dos.

```
    @ComboBox(title = "Sexo:", belongsTo = "groupPersonal", column = 3, items
= { "Hombre", "Mujer" })

    public String sexo;
```

Ilustración 18 - Combobox añadido al grupo 'Información personal'

6.4 Cómo utilizar @DateField

@DateField se encarga de generar campos de entrada en el formulario que solicitan fechas, por lo que en el ejemplo que se va a ver, se va a generar un campo en el formulario de 'Información personal' creado en el punto 6.1 Cómo utilizar @TextField y @FieldGroup, que solicite al usuario la fecha de nacimiento.

```
    @DateField(title = "Fecha de nacimiento:", belongsTo = "groupPersonal",
column = 3, to="function(){ return new Date();}",
noBiggerThan="function(fieldDate){ return fieldDate!= null && fieldDate<=new
Date();}",  )

    public Date fechaNacimiento;
```

Respecto a las propiedades del ejemplo que se ha expuesto en el párrafo anterior, la propiedad *title* indica el texto que se mostrará junto el campo, la propiedad *belongsTo* especifica que la fecha irá incluida en la agrupación 'Información personal' y se colocará en la tercera fila de la agrupación, esto se especifica en la propiedad *column*. Por otra parte no se desea que se pueda seleccionar una fecha mayor a la fecha actual, ya que carece de sentido registrar a alguien con una fecha de nacimiento mayor a la actual, por ello se ha introducido una función Java Script que devuelve la fecha actual, por lo que el calendario nunca permitirá seleccionar una fecha mayor a la actual; para que la fecha sea de carácter obligatorio se ha añadido otra función Java Script que comprueba que la fecha introducida no sea nula y sea a su vez menor que la fecha actual, esto se especifica en la propiedad *noBiggerThan* de la anotación. En lo que respecta al resto de propiedades se han dejado por defecto ya que la configuración con la que dotan al campo es la deseada, el resto de propiedades, así como la configuración que traen por defecto las propiedades de la anotación pueden ser estudiadas en el capítulo 5.3.2.3 @DateField. Como puede apreciarse en la *Ilustración 19 - Fecha de nacimiento añadida a la agrupación 'Información personal'* se ha añadido un campo solicitando la fecha de nacimiento, cuando se selecciona el campo para introducir un valor, aparece un calendario emergente en la parte inferior del campo.

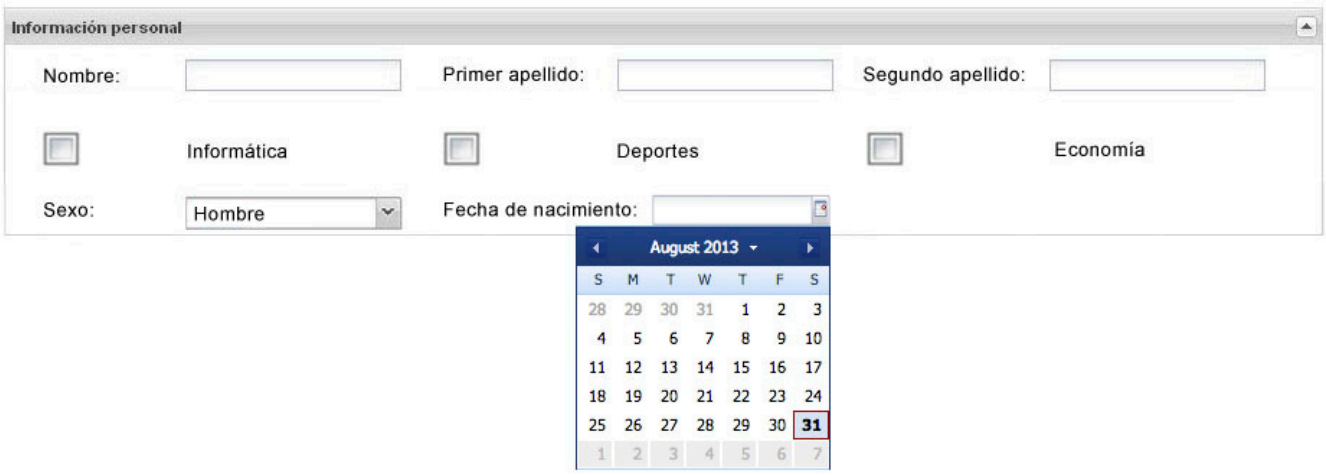

Ilustración 19 - Fecha de nacimiento añadida a la agrupación 'Información personal'

6.5 Cómo utilizar @TextEditor

La anotación @TextEditor genera un editor de texto enriquecido, que permite añadir texto con estilos, permitiendo modificar el tipo de letra, el color, añadir numeraciones, hipervínculos, imágenes sobre el texto introducido, etc. La anotación no requiere de ningún atributo obligatorio, por lo que haciendo referencia a esta, ya podría hacerse uso del editor de texto, en el ejemplo que observaremos a continuación, se pretende que el editor de texto se situé al final del formulario y disponga de un título con el texto 'Acerca de ti', para que así el usuario pueda describir lo que le convenga en el editor.

```
@TextEditor(title = "Acerca de ti")
public String acercaDeTi;
```

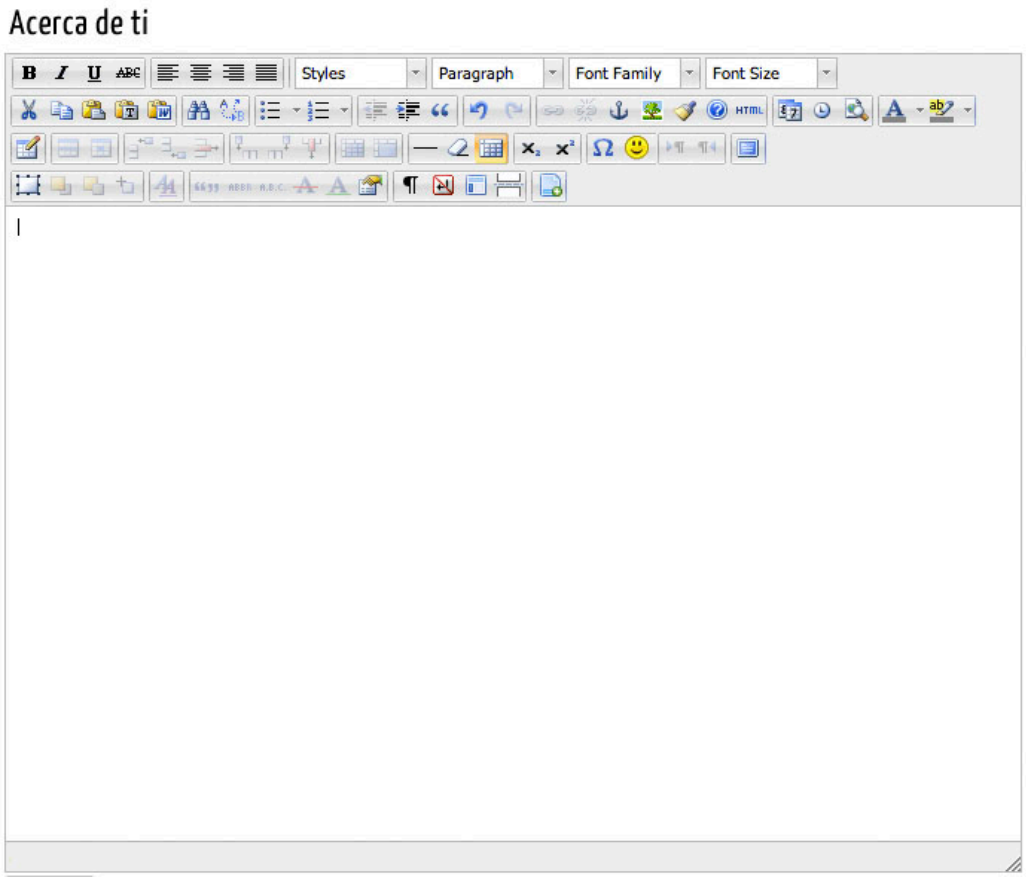

Ilustración 20 - Editor de texto generado por @TextEditor

Al desear que el editor de texto se sitúe al final del formulario, no especificamos ningún orden pero deberá de introducirse el campo 'acercaDeTi', como última propiedad de la clase Java, el identificador del editor de texto, al igual que los demás parámetros del formulario, será el nombre del atributo de la clase Java, ya que no se ha especificado ningún valor en la propiedad *name* de la anotación @TextEditor. Al no querer que pertenezca a ninguno de los dos grupos declarados en el formulario no se especificara ningún valor en la propiedad *belongsTo* de la anotación @TextEditor. Como se aprecia en la anotación escrita del párrafo anterior, únicamente se ha especificado valor en la propiedad *title*, por lo que el editor de texto se visualizará como en la *Ilustración 20 - Editor de texto generado por @TextEditor*, todos los elementos que aparecen en el editor vienen por defecto y no son configurables, únicamente se puede configurar la posición y el tamaño del editor, si se quiere dentro de un cuadro desplegable o que esté agrupado junto con otros parámetros de entrada del formulario.

Todas las casillas que trae el editor de texto son propias del componente utilizado TinyMCE, únicamente se ha personalizado la casilla que permite la inserción de imágenes modificando el componente que trae por defecto TinyMCE. Para acceder al componente se debe de seleccionar el icono que aparece con un círculo rojo de la *Ilustración 21 - Icono para el acceso al gestor de imágenes del editor de texto*.

Ilustración 21 - Icono para el acceso al gestor de imágenes del editor de texto

Cuando se accede al componente se dispondrá de una galería de todas las imágenes que se han ido insertado a través de éste, puede verse como es el componente con una imagen cargada en la *Ilustración 22 - Gestor de imágenes de @TextEditor*; cada guardado o borrado de imagen en el componente supone una nueva versión en el formulario por lo que nunca se perderán las imágenes, cada vez que se accede al componente internamente se invoca al servicio web *devon/forms/getFiles* explicado en el punto 5.2.1 Servicios Web Devon Forms Web, procesando únicamente los adjuntos que se consideran imágenes para el editor de texto. A través del identificador de la imagen se genera dinámicamente la URL para la descarga de la imagen, */devon/forms/getFile* también explicado en el punto 5.2.1 Servicios Web Devon Forms

Web, el componente Java Script es capaz de obtener una visualización en miniatura de la imagen a través de la URL, teniendo internamente los binarios de las imágenes almacenados en el cliente.

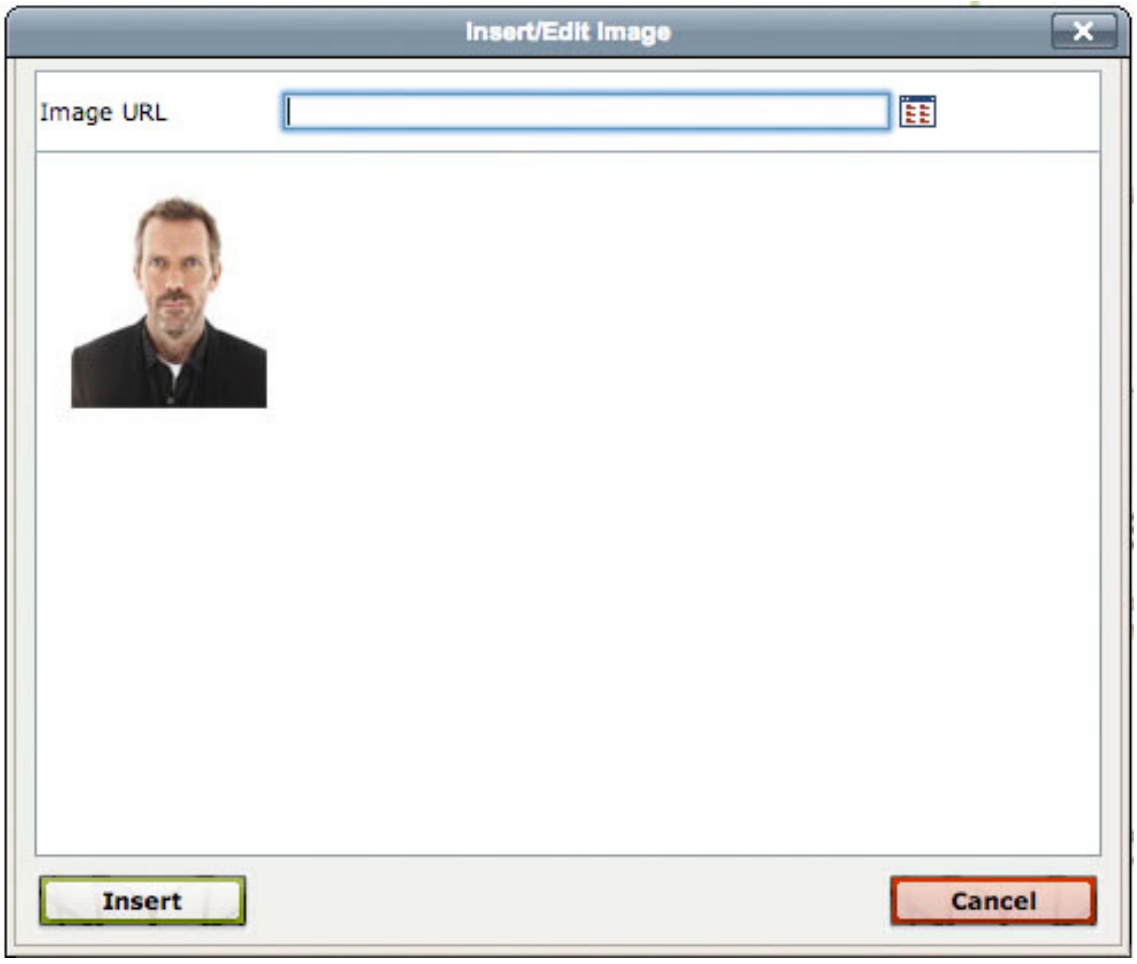

Ilustración 22 - Gestor de imágenes de @TextEditor

7 Creación desde cero de un listado complejo a través de Devon Forms Web

En este capítulo se va a explicar cómo generar listados complejos a través de una clase en Java, basándose en la anotación @Gengrid de la extensión Devon Forms Web.

En este ejemplo se va a crear un listado complejo que registra información acerca de expedientes académicos. Para la generación del listado lo primero es definir una clase en Java con las columnas que queremos que disponga el listado, cada columna debe de corresponderse con un atributo de la clase, al igual que en @GenForm todos los atributos deben de ser de carácter público, ya que el proceso que se encarga de generar el listado debe tener acceso a ellos.

```
public class ListadoExpedientes {
    public String id;
    public String codigoExpendiente;
    public String descripcion;
    public String descripcionCorta;
}
```

Una vez definida la clase indicaremos que propiedades queremos que sean columnas y cuáles sean utilizadas únicamente para identificar el registro. Para ello debemos de dotar a la clase y a los atributos de la clase con anotaciones de Devon Forms Web.

Primero se dotará de dos anotaciones a la clase:

> @Component: Esta anotación perteneciente de Spring, indica que la clase va a ser un componente.
>
> @GenGrid: Esta anotación propia de Devon Forms Web, indica que la clase va a ser un listado.

Una vez la clase disponga de ambas anotaciones, se debe introducir anotaciones a las propiedades de la clase.

> @TextColumn: Esta anotación propia de Devon Forms Web, indica que la propiedad de la clase, va a ser transformada en una columna del listado.

Añadidas estas tres anotaciones, la clase pasará a tener la siguiente nomenclatura:

```
@Component
@GenGrid
public class ListadoExpedientes {
	@TextColumn
	public String id;
	@TextColumn
	public String codigoExpendiente;
	@TextColumn
	public String descripcion;
	@TextColumn
	public String descripcionCorta;
}
```

En el ejemplo mostrado en el párrafo anterior, todas las columnas del listado que se generasen estarían configuradas con sus valores por defecto, por lo que todas las columnas del listado serían visibles para el usuario y el texto que identificaría a cada columna en el encabezamiento del listado sería el mismo que el nombre otorgado al atributo de la clase Java *ListadoExpedientes.*

@GenGrid dispone de un formulario que se genera automáticamente tomando como campos en el formulario todas sus anotaciones @TextColumn, la cual debe de disponer de una configuración determinada que se explicará más adelante en este mismo capítulo. La configuración que trae por defecto @TextColumn permite a @GenGrid que todos los campos sean incluidos en el formulario, tanto de edición como de creación de un registro.

De los cuatro atributos definidos en la clase, queremos que aparezcan todos en el formulario excepto el primero de ellos nombrado como 'id', ya que éste será un identificador autogenerado por el servidor para identificar a los registros unívocamente, por lo que la clase debe pasar a tener la siguiente configuración:

```
@Component
@GenGrid(title = "Listado Tipo Reapertura Expediente")
public class ListadoExpedientes {
	@TextColumn (visible = false)
	public String id;
	@TextColumn(header = "Código", order = 1)
```

```
        public String codigoExpendiente;
        @TextColumn(header = "Descripción", order = 3)
        public String descripcion;
        @TextColumn (header = "Descripción corta", order = 2)
        public String descripcionCorta;
}
```

Con la nueva configuración, el campo 'id' ya no será visible ni en el listado ni en los formularios de edición y creación del registro, por otro lado se ha añadido el título del que dispondrá el listado y el de los encabezados de cada columna, así como el orden que deberán ocupar. El orden que se especifica en la propiedad *order* será válido tanto para la aparición de las columnas en el listado como en el formulario de edición e inserción, al igual que la propiedad *header* será utilizada para el texto descriptivo de su campo correspondiente en el formulario de edición e inserción de registros.

Para poder utilizar la clase como un listado, deberemos realizar los mismos pasos seguidos al importar un formulario, explicados en el capítulo 6 Creación desde cero de un formulario a través de Devon Forms Web:

1. Importar la clase creada, de la siguiente manera:

```
<script type="text/javascript"
src="../public/renderJavascript.htm?id=ListadoExpedientes
script/>
```

2. Una vez invocado al servicio para importar al formulario, se debe invocar al método Java Script que proporciona Devon Forms Web para instanciar y visualizar el formulario.

```
app.components.formdefs.ListadoExpedientes();
```

Una vez seguidos todos los pasos mencionados con anterioridad en este punto, se obtendrá en la pagina donde se ha realizado la llamada Java Script para visualizar un listado como el de la *Ilustración 23 - Listado generado con @GenGrid***Error! Reference source not found.**. Se puede apreciar en la parte inferior del listado la paginación, disponiendo de un botón para actualizar la información del listado y botones para navegar entre las páginas disponibles, así como permitiendo acceder a la primera y última página de una manera directa. Este listado generado no dispone de la posibilidad de gestionar los registros debido a que no se ha asociado ninguna

operación para la gestión de los mismos, tampoco se ha gestionado ningún servicio para proporcionar datos al listado.

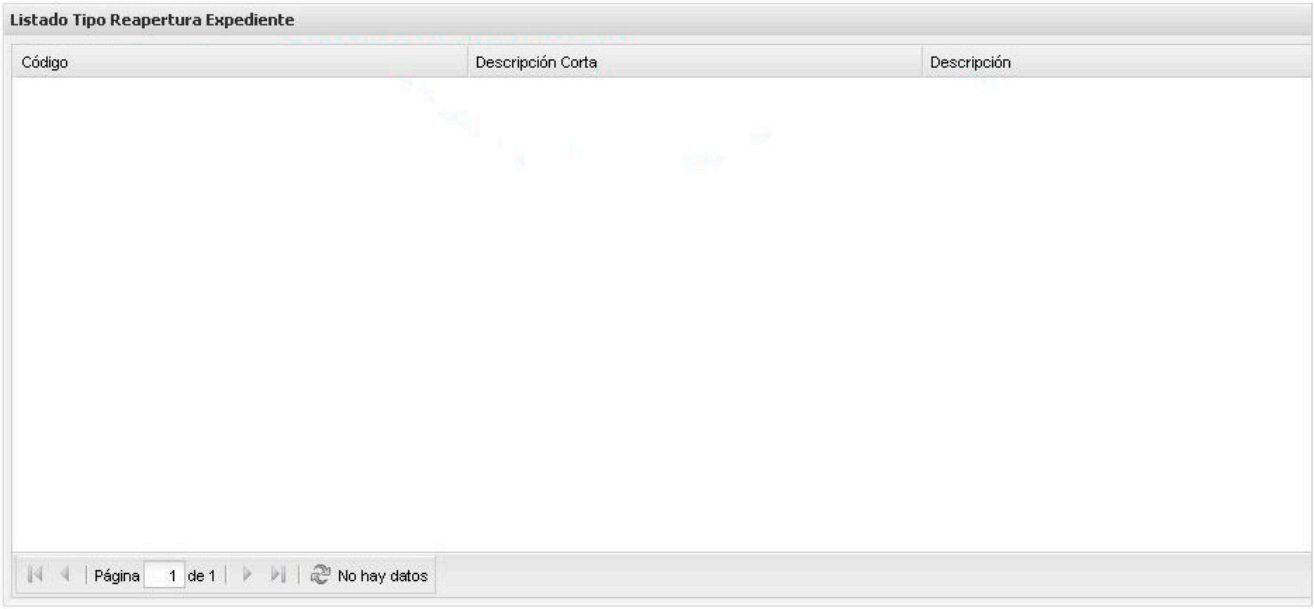

Ilustración 23 - Listado generado con @GenGrid

Para proporcionarle datos al listado generado, se debe de asociar un servicio web que soporte la paginación, para ver que parámetros de entrada y que comportamiento debe tener el servicio web para adaptarse con el listado generado, ver el capítulo 5.3.3 @GenGrid; lo mismo ocurre con los servicios web para la inserción, edición y borrado de los registros, por lo que se debe de asociar tanto el servicio web que proporcionará datos al listado como los servicios que gestionarán los registros, para ello únicamente se debe modificar la anotación @GenGrid. El resto de elementos de la clase no requieren de ningún tipo de cambio.

```
@GenGrid(title = "Listado Tipo Reapertura Expediente", provider =
"expedientes/reapertura/listado", addProvider = "expedientes/reapertura/add",
deleteProvider = "expedientes/reapertura/delete")
```

Al igual que cuando se asocia un servicio web en @GenForm, no es necesario el indicar el host y puerto donde está publicada la aplicación web. Se indica a través de la propiedad 'provider' que el servicio web que proporcionará datos al listado es accesible a través de la uri 'expedientes/reapertura/lista'. Por otro lado al indicar únicamente el servicio web en la propiedad *addProvider*, se asocia el servicio de actualización e inserción del registro sobre el mismo servicio web; *deleteProvider*

indica el servicio web que se encargara de eliminar los registros del formulario. Al asociar los servicios web para la gestión del listado, el listado dispondrá de tres nuevos botones en la parte superior del listado, con los textos 'Agregar', 'Editar' y 'Eliminar', el primero de ellos siempre está habilitado permitiendo agregar en cualquier momento un registro, en cambio los dos restantes únicamente estarán habilitados cuando un registro quede seleccionado, ya que van asociados a un registro determinado del listado. El listado con los botones que permiten gestionar los registros queda reflejado en la *Ilustración 24 - Listado generado con @GenGrid con botones asociados*. Debido a que no se ha añadido ninguna anotación propia de @GenForm sobre los campos que representan las columnas, los campos generados en el formulario de inserción y edición de los registros, dispondrán de entradas de texto simple. En el caso de que se quiera configurar otro tipo de entrada para los formularios que gestionan los registros, se debe añadir la anotación @GenForm sobre la clase Java y las anotaciones propias de la anotación sobre los campos que se quiera, como ya fue explicado en el capítulo 5.3.4 Personalización en los formularios de creación/actualización en @GenGrid. En el ejemplo de este capítulo, los formularios para la edición e inserción de registros será exactamente el mismo, debido a que las propiedades *editable* y *visible* de la anotación @TextColumn han permanecido con su valor por defecto, la explicación de las propiedades puede encontrarse en el capítulo 5.3.3.1 @TextColumn. Únicamente cambiará en que el formulario de edición dispondrá de un registro cargado en el formulario y los campos del formulario de inserción estarán vacíos.

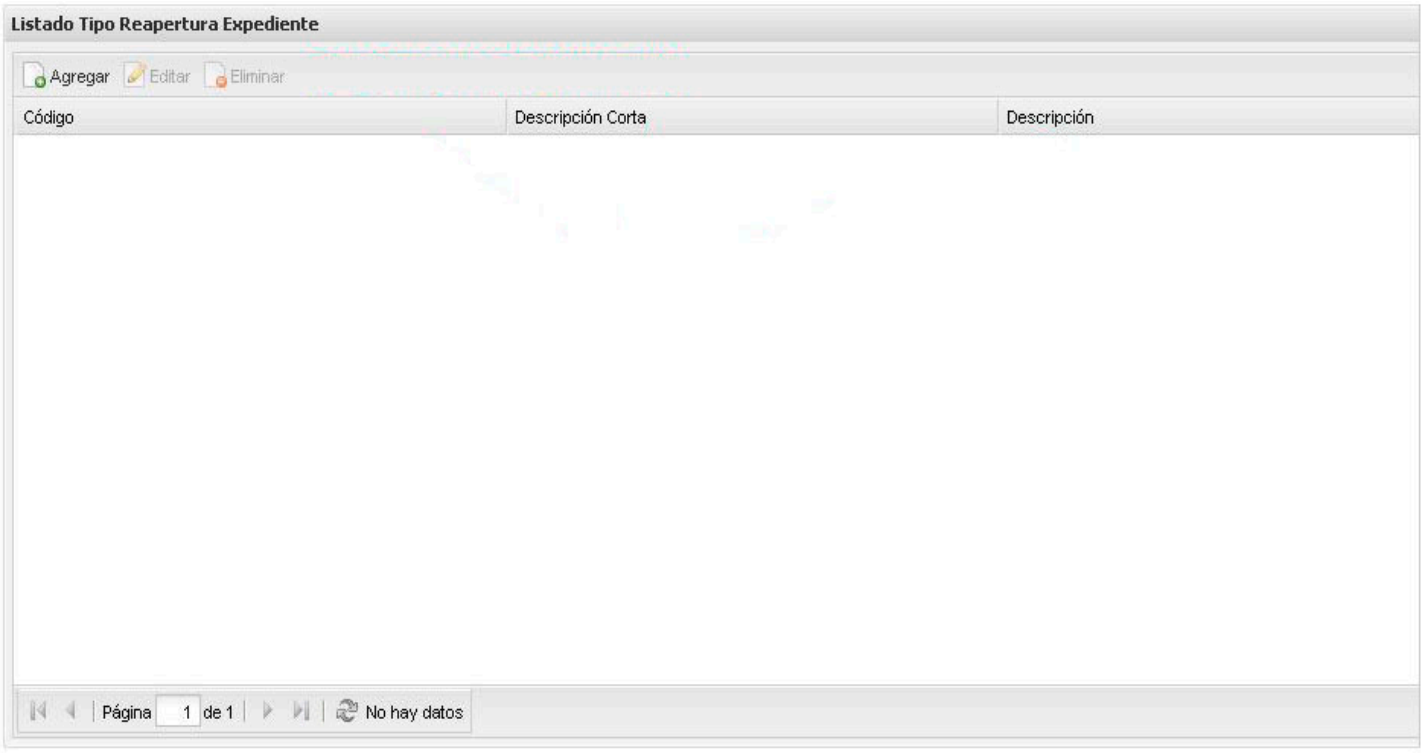

Ilustración 24 - Listado generado con @GenGrid con botones asociados

El listado generado hasta el momento no dispone de ningún tipo de filtrado trayendo consigo siempre los mismos datos, pudiendo resultar para el usuario algo incomodo si los datos que van a ser gestionados disponen de un gran volumen de registros. Tal y como se explicó en el punto 5.3.3.2 @Filter, es posible incorporar un formulario de búsqueda para refinar los datos del formulario.

```
@Component

@GenGrid(title = "Listado Tipo Reapertura Expediente", provider =
"expedientes/reapertura/listado", addProvider = "expedientes/reapertura/add",
deleteProvider = "expedientes/reapertura/delete")

public class ListadoExpedientes {

	@TextColumn (visible = false)

	public String id;


	@Filter(column = 1, order = 1)

	@TextColumn(header = "Código", order = 1)

	public String codigoExpendiente;


	@Filter (column = 1, order = 2)

	@TextColumn(header = "Descripción", order = 3)

	public String descripcion;


	@Filter(column = 2)

	@TextColumn (header = "Descripción corta", order = 2)

	public String descripcionCorta;

}
```

Añadida la anotación @Filter sobre los campos que se pretende realizar la búsqueda, se creará un formulario en la parte superior del listado. El formulario de búsqueda dispondrá de dos filas, en la primera de ellas tendrá alojado los campos *codigoExpediente* y *descripcion,* ya que ambos disponen del mismo valor en la propiedad *column* en @Filter. En la segunda fila únicamente estará el campo *descripcionCorta*, todos los formularios generados por @Filter disponen de dos botones adicionales, uno para realizar la búsqueda y otro para limpiar el formulario y dejar todos los parámetros de entrada vacíos, en la *Ilustración 25 - Listado generado con @GenGrid y formulario de búsqueda creado con @Filter*, se puede apreciar cómo será el listado generado proporcionado por la nueva anotación @Filter.

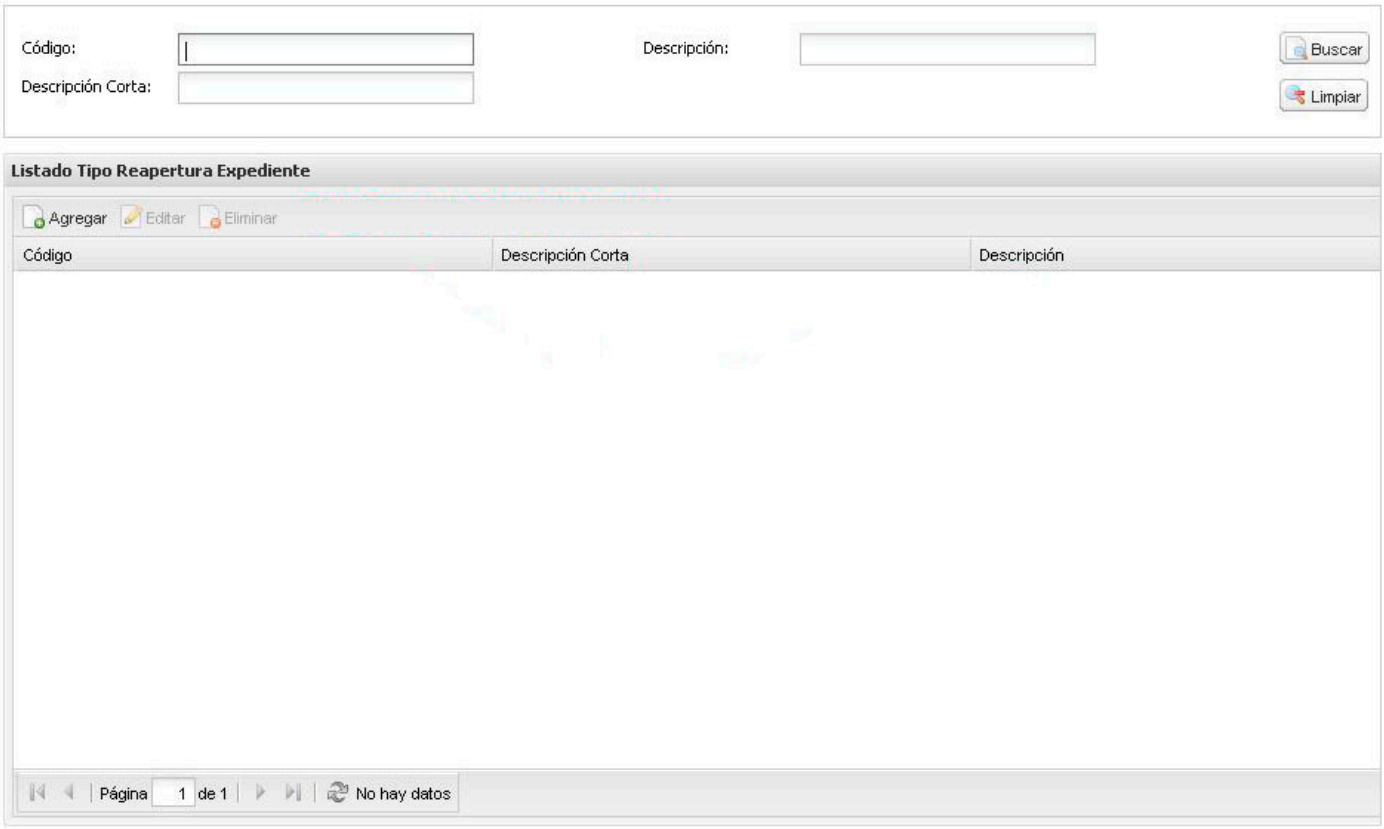

Ilustración 25 - Listado generado con @GenGrid y formulario de búsqueda creado con @Filter

La búsqueda que realice el formulario será de manera local, ya que no se ha asociado ningún servicio web donde solicitar los datos filtrados, por lo que se realizará sobre los datos que se han cargado previamente en el listado. En listados que no disponen de paginación y los datos no son muy cambiantes las búsquedas locales pueden ser el

comportamiento deseado, pero en el ejemplo tratado el listado dispone de paginación, por lo que carece de sentido realizar búsquedas locales ya que se realizan búsquedas solamente de la página que está cargada y no sobre la totalidad de sus registros. Para que la búsqueda se realice con datos servidos directamente por el servidor, se debe de añadir a la clase Java la anotación @FilterProvider asociándole un servicio web. La anotación debe de asociarse a nivel de clase, por lo que se encontrará en el mismo nivel que la anotación @GenGrid. En el siguiente fragmento de código puede verse como quedaría la clase asociándole el nuevo servicio web.

```
@Component

@GenGrid(title = "Listado Tipo Reapertura Expediente", provider =
"expedientes/reapertura/listado", addProvider = "expedientes/reapertura/add",
deleteProvider = "expedientes/reapertura/delete")

@FilterProvider(provider = "expedientes/reapertura/listado")

public class ListadoExpedientes {

	//Mismos atributos con mismas anotaciones que en los ejemplos
	anteriores

}
```

Se han obviado los atributos de la clase y las anotaciones de los mismos ya que no difieren respecto al último ejemplo donde se declaraba el filtrado, el servicio es especificado en *provider*, la única propiedad de la anotación @FilterProvider. En este caso el servicio de búsqueda es el mismo que el servicio que ofrece todos los datos al listado, esto es una práctica habitual, unificando un mismo servicio que ofrezca todos los datos si la búsqueda no dispone de ningún parámetro de entrada, ya que si se realiza la acción de presionar sobre el botón 'buscar' del formulario y no se ha introducido ningún parámetro en la búsqueda, se invocará al servicio de que proporciona los datos recibiendo el servicio web el objeto vacío. Cuando el usuario realiza este tipo de acción, espera obtener todos los resultados sin ningún tipo de filtrado, es por ello por lo que el servicio de filtrado y obtención de los datos suele ser el mismo.

8 Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones obtenidas una vez se finalizó con todo aquello que se quería incluir en el módulo Devon Forms Web, así como las nuevas funcionalidad que se quieren incluir en futuras versiones.

8.1 Conclusiones

Tras la finalización de Devon Forms Web, se apreció el potencial que podía dar un módulo de estas características. A lo que se refiere en tiempo de desarrollo, una vez se entra en contexto con el módulo y se conocen sus anotaciones, el desarrollar algo que puede ser proporcionado a través de Devon Forms Web no puede llevar más de una hora de desarrollo, por lo que supone un gran ahorro en tiempo de desarrollo y ofreciendo a su vez un componente muy fácil de mantenerse frente a futuros cambios. En lo que respecta a los formularios generados, únicamente se ha de tener en cuenta en cómo se quiere visualizar el formulario y que campos de entrada debe de tener, el desarrollador no debe de preocuparse de los servicios web que gestionan el formulario, así como el control de versiones del formulario que trae por defecto, ni del mantenimiento de la base de datos, todo ello es gestionado por Devon Forms Web.

Se asegura que los componentes generados a través de Devon Forms Web, son componentes sin errores, adaptados por un lado al sistema web en el que se ha incorporado, pero abstraído a posibles cambios en el desarrollo del sistema que afecten de alguna manera a los componentes creados. Es común cuando se desarrolla software que el desarrollo de una parte del sistema afecte a otra por algún motivo, estos tipos de errores sólo pueden incurrir si se interactuase con algún componente generado afectando al canal de interacción entre el sistema web que se está desarrollando y los componentes generados, pero nunca deberían afectar internamente al componente.

Los formularios creados a través de Devon Forms Web tienen la ventaja de que todos ellos dispondrán del mismo estilo estandarizado, por lo que el desarrollador no tendrá que involucrarse tanto en el desarrollo de la interfaz de usuario, pudiéndose centrar en otras tareas.

Comparando Devon Forms Web con los Frameworks similares que se presentaron en los puntos 3.1.1 AutoCRUD y 3.1.2 Yii Framework, ambos Frameworks facilitan la labor al desarrollador creando un sistema web base, con más o menos funcionalidades como ya se vio en los puntos citados, pero no disponen de elementos propios que permiten

la creación de elementos gráficos, creados a través de componentes propios del Framework. Por otro lado, ambos están ligados al motor de base de datos, creando el sistema web a través del esquema de base de datos, en cambio Devon Forms Web si va a ser utilizado únicamente para la creación de formularios, el desarrollador no debe de realizar nada en lo que base de datos respecta, para la creación de listados vinculados a servicios web. Si que se debe de realizar la implementación del servicio por parte del desarrollador, pero todos los proyectos realizados con Devon, utilizan Hibernate para la creación y gestión de la base de datos, por lo que nuevamente el desarrollador se abstrae de la base de datos, sin atarse a ningún motor de base de datos.

8.2 Trabajo futuro

Debido al ámbito en el que se encuentra el módulo Devon Forms Web dispone de múltiples ampliaciones. Los sistemas Web tienen la ventaja de que pueden ejecutarse en cualquier navegador web, por lo que cualquier dispositivo con conexión a internet y un navegador puede tener acceso al sistema, es por ello que el sistema web debe de adaptarse dependiendo el dispositivo que va a acceder al mismo. En la versión que se ha presentado en este trabajo, el módulo está diseñado para que el sistema sea visualizado de manera correcta en dispositivos de sobremesa u ordenadores portátiles, aunque se visualice y todo funcione correctamente en los dispositivos móviles (teléfonos móviles y tablets), la interfaz gráfica que se ofrece al usuario no es la apropiada, ya que los componentes que se visualizan en un dispositivo móvil no disponen del tamaño adecuado, siendo demasiado pequeños y difíciles de acceder a ellos.

Debido a que cada vez son más los clientes que solicitan sistemas web para dispositivos móviles, se quiere adaptar Devon Forms Web para que genere los mimos servicios que ofrece actualmente pero adaptados a dispositivos móviles, gracias a la arquitectura y el diseño que se ha seguido para el desarrollo de Devon Forms Web la creación de formularios y listados adaptados a dispositivos móviles será algo muy liviano. La tecnología que se utilizaría sería Sencha Touch, perteneciente al mismo grupo que Ext-JS y que dispone de una nomenclatura muy similar a Ext-JS. Para esta nueva funcionalidad únicamente se tendrán que incorporar Sencha Touch a Devon Forms Web, crear nuevas plantillas StringTemplate utilizando Sencha Touch en vez Ext JS y crear alguna nueva anotación o incluso bastaría con crear simplemente una propiedad de una anotación que indique si los formularios y listados que van a ser creados se emplearán en un dispositivo móvil o de sobremesa.

Otra posible ampliación sería la creación de campos tipados en las columnas que son generadas en el listado complejo, siendo más consistentes con los datos que se están mostrando y siendo consciente el generador del tipo de dato que se está mostrando en el listado, generando de manera automática los campos con el mismo tipo que se muestra en el listado para el formulario de creación y edición de los registros del listado.

Una tercera ampliación sería la creación de nuevas anotaciones que representasen nuevos tipos en los parámetros de entrada del formulario, dando la opción de incluir elementos que no sean utilizados en formularios estándar.

La última ampliación debería estar presente siempre que se actualice Ext JS y Sencha Touch en el caso de que se realzase la ampliación de adaptar el sistema web a plataformas móviles, y es de que disponga Devon Forms Web de la última versión estable de los Frameworks Ext JS y Sencha Touch. La tecnología va actualizándose a un ritmo vertiginoso, mejorando elementos visuales, incorporando nuevos componentes, incrementando la performance de las plataformas, etc. Es por ello que esta última ampliación es una mejora indispensable cuando se realicen cambios significativos en las versiones de alguno de los dos Frameworks.

9 Referencias

[1] Introducción sobre devon, http://www.es.capgemini.com/application-lifecycle-services/desarrollo-acelerado-de-soluciones-devon

[2] Roy Thomas Fielding, Tesis 'Architectural Styles and the Design of Network-based Software Architectures'. CHAPTER 5: Representational State Transfer (REST) http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[3] Craig Walls. *Spring in action third edition*. *Chapter* 11 Giving Spring some REST

[4] Craig Walls. *Spring in action third edition*. *Chapter* 7 Building web applications with Spring MVC

[5] Clarence Ho. *Pro Spring 3*. Chapter 8: Spring JDBC Support

[6] Página oficial Framework Autocrud. http://www.autocrud.sourceforge.net

[7] Página oficial Framework Yiiframework. http://www.yiiframework.com

[8] Página oficial framework Sencha Ext-JS. http://www.sencha.com/products/extjs

[9] Página oficial tinymce. http://www.tinymce.com

[10] Página oficial json, http://www.json.org

[11] Jesus Garcia. *ExtJS in action*. *Chapter* 2 Ext JS Components

[12] Ajit Kumar. *Sencha MVC Architecture*. *Chapter* 1 Sencha MVC Architecture

[13] Página oficial Spring Framework, http://www.springsource.org/spring-framework

[14] Página oficial Hibernate, *http://www.hibernate.org*

[15] James Elliott, Timothy M. O'Brien, Ryan Fowler. *Harnessing Hibernate. Chapter 7 The Annotations Alternative*

[16] James Elliott, Timothy M. O'Brien, Ryan Fowler. *Harnessing Hibernate.* Chapter 13 Put a Spring in your Step: Hibernate with Spring

[17] Clarence Ho. *PRO Spring 3*. Chapter 9: Using Hibernate in Spring

[18] PÁGINA OFICIAL STRINGTEMPLATE. HTTP://WWW.STRINGTEMPLATE.ORG

[19] DOCUMENTACIÓN SOBRE STRINGTEMPLATE 3.0. HTTP://WWW.ANTLR.ORG/WIKI/DISPLAY/ST/STRINGTEMPLATE+3.0+PRINTABLE+DOCUMENTATION

[20] PROTROCOLO TRANSFERENCIA HIPERTEXTO, HTTP://WWW.W3.ORG/PROTOCOLS/RFC2616/RFC2616-SEC3.HTML

[21] ESTÁNDAR ECMA-262, HTTP://WWW.ECMA-INTERNATIONAL.ORG/PUBLICATIONS/FILES/ECMA-ST/ECMA-262.PDF