Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA UN CLUB DE LECTURA

Alumno: Antonio Cejudo Cintas

Tutor: Prof. D. José Ramón Balsas Almagro

Dpto: Departamento de Informática

Septiembre, 2016

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don JOSÉ RAMÓN BALSAS ALMAGRO , tutor del Proyecto Fin de Carrera titulado: ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA UN CLUB DE LECTURA, que presenta ANTONIO CEJUDO CINTAS, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, SEPTIEMBRE de 2016

El alumno: Los tutores:

ANTONIO CEJUDO CINTAS JOSÉ RAMÓN BALSAS ALMAGRO

Índice

1. INTRODUCCIÓN

Esta primera sección se presenta a modo de introducción al TFG. Se comentan la motivación y propósito para proponer un trabajo de este tipo, los objetivos a alcanzar, las metodologías que se emplearán, y por último, una explicación sobre la estructura que seguirá este documento.

1.1. Motivación

En este apartado, se expone de forma general el proyecto a realizar, del contexto en el que se desarrollará el sistema y la motivación que lleva a proponer un trabajo de estas características.

La propuesta de este trabajo, surge de la necesidad de un grupo de aficionados a la lectura que quieren poder gestionar una biblioteca personal on-line, poder asignar estados a sus lecturas actuales y poder consultar la de los demás.

También tienen la necesidad de poder crear un grupo de lectura privado donde decidir el próximo libro a leer del grupo, establecer plazos de lectura e intercambiar impresiones sobre el mismo.

Además, se desea que a largo plazo, esta solución sea integrable con lectores de libros iOS y Android para realizar sincronización de la biblioteca y que permita, por ejemplo, actualizarla al añadir un nuevo libro al lector de libros o actualizar automáticamente el progreso en la lectura de un libro… Evidentemente, estas características requieren un tiempo de desarrollo adicional que excede los límites estipulados para un TFG, pero se menciona para dar una mejor visión de la motivación que lleva a proponer este proyecto.

1.2. Objetivos

En este apartado se enumeran los objetivos plasmados en la propuesta de trabajo:

- Realizar un estudio de necesidades y soluciones para la creación y gestión de una biblioteca personal y de clubs/grupos de lectura online.
- Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas y metodologías de desarrollo web.
- Implementar un prototipo de aplicación web usando tecnologías REST basadas en Spring Framework en el lado del servidor, y una interfaz adaptativa en el cliente usando HTML5 y AngularJS.

1.3. Metodología

En este apartado se enumeran los conceptos básicos de metodología a desarrollar durante el proyecto:

- Estudio de los requisitos del sistema a desarrollar.
- Recopilación bibliográfica sobre temática y tecnologías relacionadas.
- Metodología ágil de desarrollo SCRUM para realizar estimación temporal y planificación de desarrollo.
- Modelado UML.
- Diseño del sistema utilizando metodología de orientación a objeto, haciendo uso de patrones de diseño y arquitecturas basadas en técnicas de Ingeniería del Software.
- Implementación del prototipo del sistema.

1.4. Estructura del documento

El presente documento se ha organizado de la siguiente manera:

En el apartado 2 de análisis trataremos:

- Análisis preliminar donde hablaremos de características, problemática y necesidades del proyecto. También se analizarán soluciones similares existentes y se introducirán los frameworks a utilizar.

- Propuesta de solución donde se da una visión general del sistema a desarrollar.
- Requisitos funcionales y no funcionales del sistema.
- Historias de usuario, ya que se usará una metodología SCRUM.
- Descomposición de historias de usuario en tareas y planificación temporal.
- Modelo de dominio.

En el apartado 3 de diseños se verán:

- Diagrama Entidad-Relación.
- Diagrama de clases de cliente y servidor.
- Diagramas de secuencia de los aspectos más relevantes del proyecto.

En el apartado 4 de implementación se desarrollarán los siguientes apartados:

- Arquitectura.
- Detalles sobre implementación: aspectos metodológicos y técnicos, documentación, recursos y bibliotecas externas utilizadas, frameworks, versiones, dependencias…

En el apartado 5 trataremos:

- Conclusiones.
- Mejoras y trabajos futuros.

En el apartado 6 se listan las referencias bibliográficas en orden de aparición.

Apéndices:

- Apéndice I: Manual de instalación del sistema.
- Apéndice II: Manual de usuario.
- Apéndice III: Descripción de contenidos suministrados.

2. ANÁLISIS

En la fase de análisis se intentará que el lector adquiera un conocimiento cada vez mayor del sistema a desarrollar de una forma gradual. Esta sección se divide de la siguiente forma:

Análisis preliminar donde se explica el sistema o contexto del que partimos y para el que se realiza el desarrollo. Hablaremos de sus características, problemáticas, necesidades… Se estudiarán algunas soluciones que existan actualmente en la web, sobre las que se intentarán obtener ideas o aspectos positivos para nuestro proyecto y aspectos que no sean adecuados o simplemente, mejorables con nuestra propuesta.

También se hace un análisis preliminar, a modo de introducción sobre las tecnologías o frameworks a estudiar: características generales, historia, evolución, licencia, tipología de proyectos para los que son adecuados…

A continuación, se aporta una propuesta de solución, que pretende dar una visión global del sistema a desarrollar y de las metodología y tecnologías a emplear.

En el apartado siguiente, se detallan los requisitos funcionales y no funcionales del sistema.

Los siguientes apartados están condicionados por la metodología de desarrollo ágil escogida (SCRUM [1]). En primer lugar se exponen las historias de usuario[2] consideradas que nos proporcionarían una funcionalidad completa del sistema.

Y por último se realiza una descomposición de estas historias de usuario en tareas y se muestra una planificación temporal inicial para el desarrollo, contemplando información sobre puntos de esfuerzo por tareas, prioridad, duración de sprint…

2.1. Análisis preliminar

En este apartado, se realiza una descripción del sistema o contexto del que partimos y para el que se realizará el desarrollo. Hablaremos de características, problemáticas, necesidades…

A continuación, realizaremos un pequeño estudio de soluciones similares existentes e intentaremos obtener puntos fuertes, débiles y posibles mejoras.

Por último, se realiza una introducción a los frameworks seleccionados, donde se expondrán, características generales, tipos de proyectos para los que son adecuados…

2.1.1. Análisis preliminar del sistema.

Se intentará crear un sistema que dé soluciones al propósito y objetivos descritos en el apartado de introducción, para ello el sistema debería:

Permitir a los usuarios obtener información de libros, añadirlos a una estantería personal donde poder realizar un seguimiento del estado de lectura de cada libro, asignarles puntuaciones y comentarios, ver información de autores y libros escritos, ver la estantería de otros usuarios, realizar búsquedas de libros, autores y usuarios…

Además, se intentará que un usuario pueda crear grupos privados de lectura, invitar a otros miembros y publicar mensajes dentro del grupo para debatir el siguiente libro a leer por el grupo, consensuar plazos para su lectura y plasmar opiniones durante la lectura del libro, o comentar capítulos concretos.

2.1.2. Estudio de soluciones existentes similares.

Se pueden encontrar varias aproximaciones a los objetivos aquí planteados en la web, con sitios tanto en español como en inglés, aunque prácticamente todos tienen un enfoque a red social de lectura, haciendo del

sistema de recomendaciones automático uno de los mayores atractivos de estos sitios.

Vamos a empezar con una mención a la red social de lectura más usada a nivel mundial, se trata de Good Reads (https://www.goodreads.com/), en la siguiente imágen se pude ver su portada.

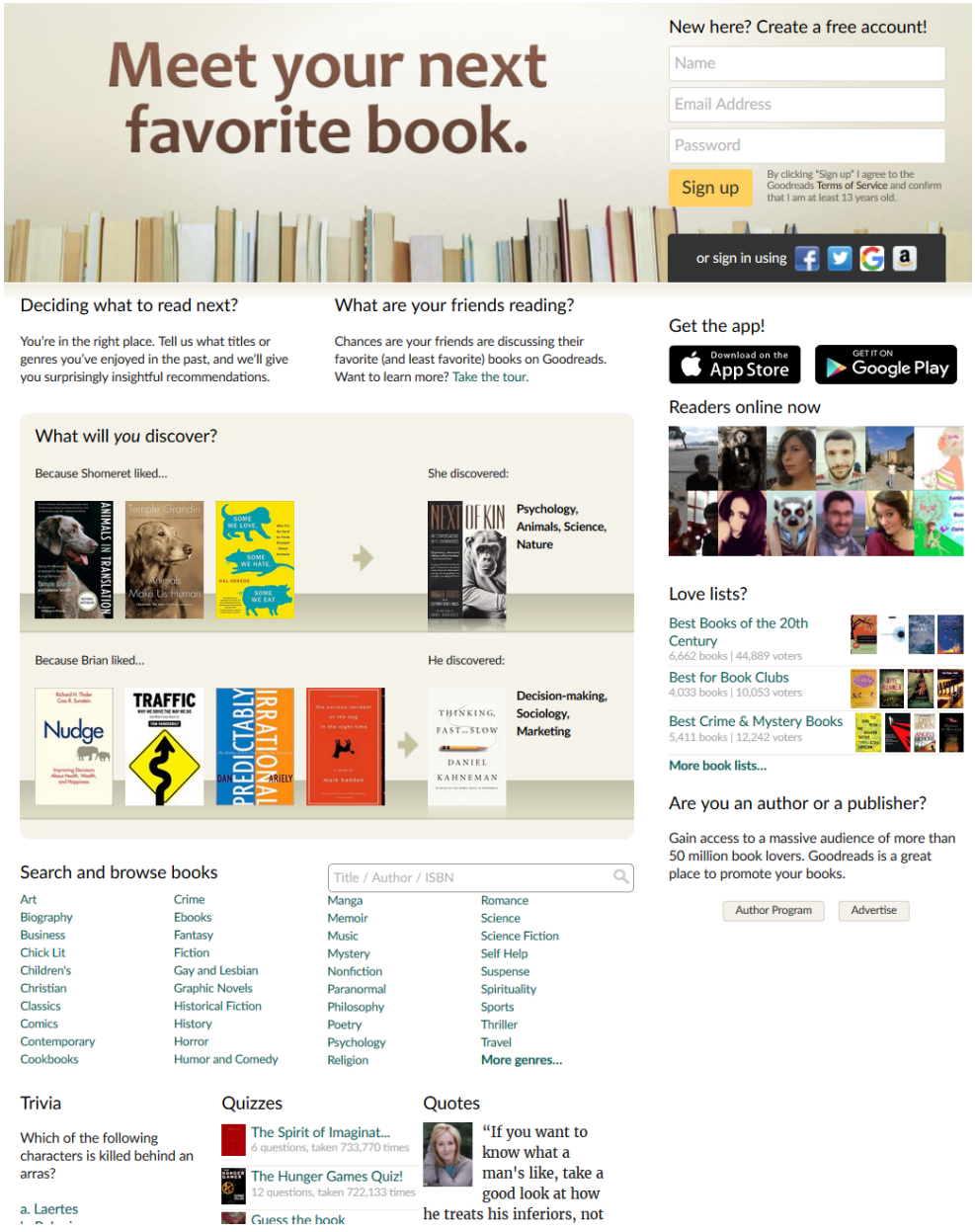

Ilustración 1.1 Portada GoodReads

Esta web ofrece una enorme cantidad de libros, y una funcionalidad completa para gestionar una estantería personal, realizar puntuaciones, comentarios, ver las de otros usuarios… Además, está completamente enfocada a la recomendación de libros, de ahí el eslogan: *"Meet your next favorite book"*. Esto se consigue mediante el uso de algoritmos de

recomendación, por lo tanto, si quieres obtener alguna recomendación primero debes añadir a tu biblioteca y puntuar un número mínimo de libros. Tras realizar este prerrequisito, puedes acabar con una recomendación de un libro, con una lectura y una nota de cinco, sobre un libro con 100000 lecturas y una puntuación de 4,5…

Además, posee también la opción de crear grupos, aunque son públicos. En la siguiente imágen se pude ver como ofrece la descripción de los libros.

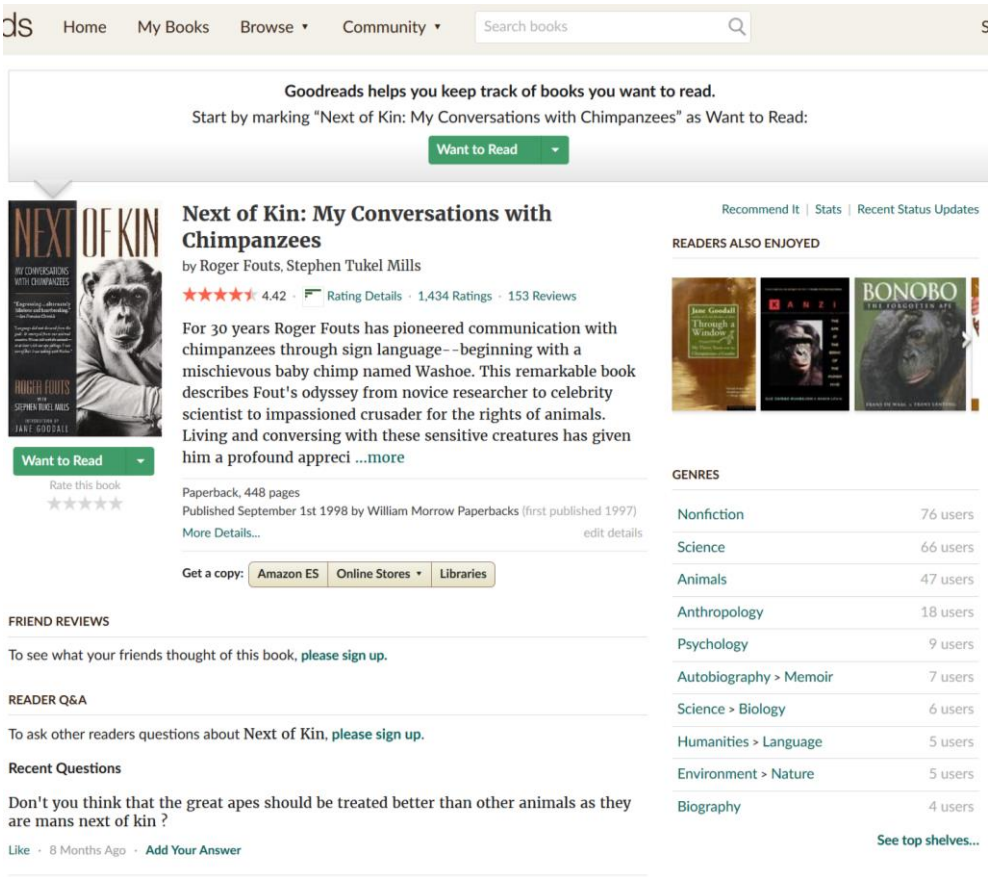

Ilustración 2.2

La web fue creada en el 2006 y a día de hoy el aspecto visual puede parecer anticuado, ya que no se usa ningún framework front-end moderno. Además, la página principal da la sensación de estar sobrecargada de elementos, texto y publicidad, en vez de llevar a una página de login sencilla o a la estantería de libros del usuario.

Aspectos mejorables:

- Sólo disponible en inglés.
- Sólo dispone de ediciones de libros publicadas en inglés (incluso para autores españoles).
- No permite establecer un idioma por defecto para listar los comentarios realizados por usuarios de un libro.
- Sistema de recomendación mediante algoritmos. No se establece ningún mecanismo de recomendación entre usuarios.
- Se permite modificar información de libros y autores a cualquier usuario.
- No está claro si se verifica la veracidad de los datos introducidos de alguna forma.
- Aspecto visual anticuado.
- No tiene ningún tipo de integración con lectores de libros en Android o iOS.
- Aspecto sobrecargado y con poco contenido del elemento básico ¡los libros!

2.1.3. Introducción a los frameworks seleccionados.

2.1.3.1.- Programación en el servidor en Java: Spring Framework

Ilustración 2.3 Spring Framework logo

Spring [3], [4] es un framework de desarrollo de código abierto que proporciona un modelo completo de programación y configuración para aplicaciones empresariales basadas en Java. Permite que el desarrollo se centre en la lógica de negocio de la aplicación, en vez de en ajustar el proyecto a un entorno de desarrollo determinado.

Propuestas tecnológicas que ofrece:

- Inyección de dependencias [5].
- Framework organizado por módulos para poder usar la funcionalidad necesaria como se muestra en [6].

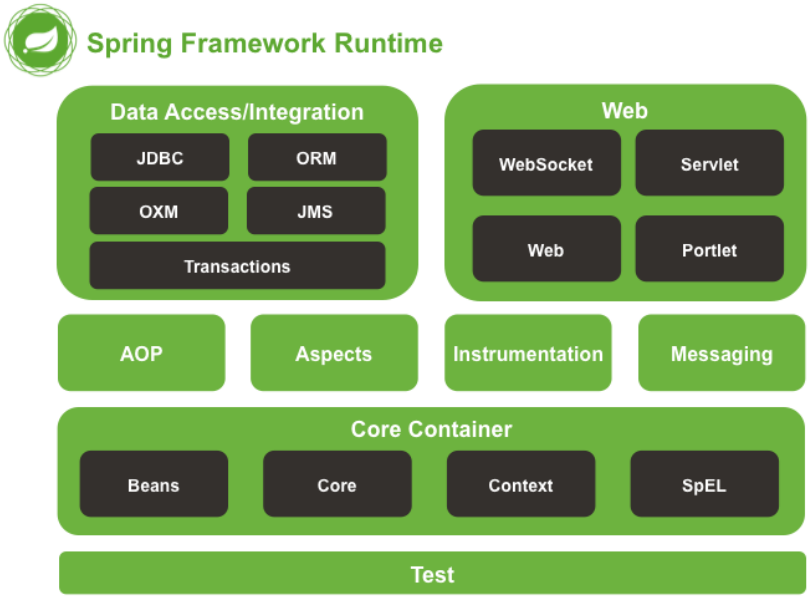

Ilustración 2.4 http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html#overview-modules

- Soporte de acceso a datos: JDBC, JPA, JMS [7].
- Soporte para aplicaciones web MVC y servicios web RESTful [8].

Como podemos ver en [9], fue lanzado por Rod Johnson en 2003 bajo licencia Apache 2.0[10]. En la actualidad es inminente la salida de la versión 5.0, la cual está prevista para el último trimestre de este año.

2.1.3.2.- Programación en cliente web: AngularJS

Ilustración 2.5 AngularJS logo

AngularJS o simplemente Angular [11], [12], [13], es un framework de código abierto desarrollado y mantenido por Google basado en JavaScript que permite crear y mantener *Single Page Web Applications[14].*

El framework aumenta la potencia de HTML y lo amplía para poder crear páginas web dinámicas.

Angular implementa un patrón MVC para separar, datos, lógica y componentes.

Sus principales propuestas tecnológicas son:

- Data binding [15] que permite realizar una sincronización bidireccional automática de valores tanto en modelo como en vistas.

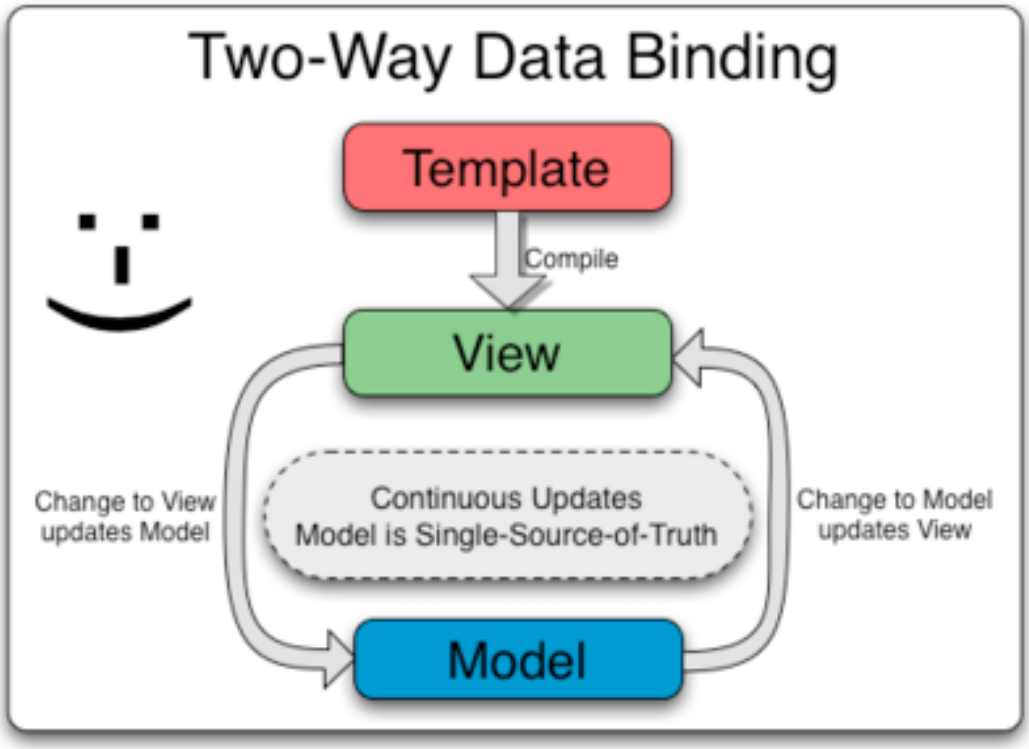

Ilustración 2.6 [15]

- Directivas[16] permiten extender la funcionalidad HTML y dar un determinado comportamiento a diferentes elementos.
- Filtros [16] permiten manipular el modo en el que se presenta la información de las vistas.
- Servicios [16] se encargan de la comunicación con otros sistemas, por ejemplo, una API REST, y de proporcionar la información externa necesaria a los controladores.

2.2.- Propuesta de solución

En esta sección se intenta dar una visión general del sistema que se pretende desarrollar, y de las metodologías y tecnologías a emplear durante el proceso.

2.2.1.- Visión general del sistema a desarrollar.

Se pretende realizar un sistema que permita al usuario obtener información sobre libros y autores, gestionar una estantería o biblioteca personal donde añadir libros y darles un estado actual de lectura, puntuación y comentarios que sean visibles por el resto de usuarios.

También se pretende que los usuarios puedan relacionarse entre ellos, mediante la creación de grupos privados de lectura, donde un usuario puede crear grupos, invitar a otros usuarios, y darle el uso que se desee, por ejemplo, debatir el próximo libro a leer, establecer fechas entre las cuales se debería de terminar el libro para comentarlo o debatirlo, comentar el progreso individual en la lectura, realizar comentarios sobre capítulos determinados, hacer recomendaciones sobre libros que el usuario haya leído y que puedan gustar a otros miembros…

Para ello se propone la utilización de los frameworks y metodologías analizados en el apartado anterior, que evidentemente permiten cumplir con los objetivos, con los que crearemos un sistema descompuesto en dos subsistemas, servidor y cliente web estableciendo una comunicación a través de una API REST.

2.3.- Requisitos del sistema

En este apartado se amplía la información ofrecida sobre el sistema de la sección anterior y se detallarán los requisitos funcionales y no funcionalidades con los que el sistema debe cumplir. Los requisitos funcionales aquí expuestos serán detallados de una forma más exhaustiva en la sección siguiente de la memoria.

- Funcionales

En este apartado se intenta plasmar las funcionalidades generales y restricciones que se espera que contemple el desarrollo.

  - Para que el sistema sea manejable por los usuarios, debe soportar una serie de funcionalidades básicas comunes a cualquier sitio web, como registro, login, ver el perfil de usuario, añadir imagen de perfil, logout…

- Una vez que el usuario puede acceder al sitio web debemos proveer funcionalidades básicas relacionadas con la temática del proyecto, como pueden ser: añadir nuevos libros y autores, listado de libros disponibles, listado de autores, ver libros escritos por un autor y toda la información relevante sobre ellos (nota media, comentarios realizados, número de lecturas, etc..). También es necesario ofrecer funcionalidades de búsqueda por título de libros y nombres de autores.
- Además, debe existir el perfil de Administrador del sitio web, que además de poseer las mismas funcionalidades de los demás usuarios, se pueda encargar de tareas de revisión y moderación, por ejemplo, modificar información errónea de libros, actualizar imágenes de portada en caso de que no fueran proporcionadas cuando el usuario los creó, eliminar libros que en realidad no existen o contienen información o imágenes inapropiadas…
- El siguiente paso, sería permitir que los usuarios interactúen con los libros, permitiéndoles añadir libros a su estantería personal (la cual sería la página principal del usuario), y desde ahí poder establecer el estado en el que se encuentra la lectura, (Quiero leerlo, Leyendo, Leído), la puntuación que se asigna al libro una vez leído y un comentario o valoración sobre el mismo.
- Y por último, permitir que los usuarios interactúen entre ellos permitiéndoles: ver lista de usuarios y realizar búsquedas por nombre, consultar la estantería de un usuario y ver estado de sus lecturas, comentarios y puntuaciones…

- Una parte fundamental en la relación entre usuarios y que realmente aporta algo novedoso sobre los sitios similares existentes, será la posibilidad de crear grupos privados de lectura, donde un usuario puede crear grupos, invitar a otros usuarios, y publicar mensajes dentro del grupo con la finalidad que se desee,

además se debe dar la posibilidad a un usuario de que edite o borre sus mensajes publicados.

- No funcionales

Aquí se tratarán aspectos que vengan impuestos por el contexto o por un supuesto “cliente”, para obtener un funcionamiento satisfactorio del sistema.

  - El sistema debe ser seguro, y garantizar que un usuario sin privilegios no puede acceder a las funcionalidades de administrador.
  - El sistema debe proporcionar una respuesta rápida frente a cualquier petición.
  - La interfaz del sitio web debe ser atractiva visualmente y proporcionar una interacción los más intuitiva posible.
  - El contenido de la web debe adaptarse a diferentes tamaños de ventana de visualización.

2.4.- Historias de Usuario

Debido al uso de metodología SCRUM, en este apartado se describirán las diferentes historias de usuario[17] con el mayor detalle posible que tratarán de alcanzarse durante el desarrollo.

Se utilizará un formato como (rol) quiero (algo) para (beneficio), como el propuesto en [18], [19] y que es de uso habitual para modelar historias de usuario.

Se elige como métrica para la estimación de historias de usuario los puntos de historia sobre días de trabajo ideales, las razones para esta elección se pueden encontrar en [20]. Es importante destacar, que los puntos de historia representan “tamaño” de una historia y no el tiempo que se tardará en completar.

Con el uso de puntos de historia conseguiremos:

- Métrica relativa.
- Planificación de entrega.
- La precisión es menos importante.
- Eliminar el sesgo.
- Imposibilidad de comparación con otros equipos.

Para las unidades de estimación se usará la serie: 1,2,3,5,8,13,20,40,100. Aunque intentaremos usar medidas en rango 1-8 como se recomienda en [21].

Se utilizará el sistema MoSCoW[22] para realizar la priorización de historias de usuario. Las prioridades asignables serán: Must have, Should have, Could have, Would not have.

La información se presenta en la siguiente tabla:

Índice	Título	Historia de Usuario	Estimación	Prioridad
1	Registro	Como usuario, quiero poder registrarme en la web e introducir información personal e imagen para tener acceso al sistema.	8	M
2	Modificar datos de usuario	Como usuario, quiero poder ver y modificar mis datos de acceso e imagen de perfil.	2	C
3	Borrado de usuario	Como usuario, quiero poder eliminar mi cuenta en cualquier momento para que mi información sea eliminada.	3	C
4	Login	Como usuario, quiero poder acceder al sistema con los datos indicados en el registro para acceder a al sitio web.	5	M
5	Creación de libro	Como usuario, quiero poder dar de alta un libro, introducir su información e imagen, para que aparezca listado en la web.	2	M
6	Listado de libros	Como usuario, quiero consultar el listado de libros disponibles en la web para	2	M

		añadirlos a mi estantería.		
7	Ver información de libro	Como usuario, quiero poder ver la información completa de un libro y las lecturas que tiene por otros usuarios.	2	S
8	Modificar libro	Como administrador, quiero añadir y modificar datos de un libro existente.	2	C
9	Borrar libro	Como administrador, quiero borrar un libro falso, o con contenido inapropiado.	2	C
10	Añadir libro a estantería	Como usuario, quiero añadir libros a mi estantería.	5	M
11	Estantería	Como usuario, quiero tener toda la información de los libros de mi estantería, y el estado actual de las lecturas de cada uno.	5	M
12	Modificar estado de lectura	Como usuario, quiero modificar el estado actual en el que se encuentra la lectura de un libro de mi estantería.	3	M
13	Modificar puntuación de lectura	Como usuario, quiero puntuar libros de mi estantería	2	M
14	Modificar comentario	Como usuario, quiero realizar un comentario sobre un libro de mi estantería que haya leído.	2	M
15	Añadir autor	Como usuario, quiero poder añadir autores y especificar su información e imagen para que puedan ser listados.	2	S
16	Listado de autores	Como usuario, quiero ver un listado de todos los autores disponibles.	2	S
17	Ver información de autor	Como usuario, quiero ver información pormenorizada de autores y listado de libros que han escrito.	2	C
18	Modificar autor	Como administrador, quiero modificar información e imagen de un autor	2	C
19	Borrar autor	Como administrador, quiero borrar información de autores falsos.	1	C
20	Búsqueda de libros	Como usuario, quiero buscar libros por título.	3	S

21	Búsqueda de autores	Como usuario, quiero buscar autores por nombre.	3	S
22	Búsqueda de usuarios	Como usuario, quiero buscar usuarios por nombre.	3	S
23	Creación grupos de lectura	Como usuario, quiero crear grupos de lectura privados.	5	M
24	Añadir miembros	Como usuario, quiero añadir miembros a un grupo de lectura.	5	M
25	Publicar mensajes	Como usuario, quiero publicar mensajes en un grupo de lectura, para poder comunicarme con los demás miembros.	3	M
26	Modificar mensajes	Como usuario, quiero modificar mis mensajes publicados en el grupo de lectura.	2	C
27	Borrar Mensajes	Como usuario, quiero borrar mis mensajes publicados en el grupo de lectura.	1	C

Tabla 2.1 Historias de usuario

2.5.- Planificación inicial de tareas

El siguiente paso para continuar con la metodología SCRUM elegida, consiste en hacer una descomposición en tareas de cada historia de usuario. También se realizará una propuesta de iteraciones necesarias, tareas a realizar en cada una de ellas, puntos de esfuerzo por tarea, y se fijará la duración de los sprints.

Tras realizar estas labores, obtendremos una estimación inicial del tiempo de desarrollo necesario, que evidentemente debe de estar en consonancia con el tiempo estimado de horas de trabajo para la realización de un TFG, aproximadamente 300 horas.

Partimos con un total de 77 puntos de historia a realizar y establecemos el tiempo de entrega en 4 meses, es decir 17-18 semanas aproximadamente.

Establecemos la duración del sprint en 3 semanas y realizaremos 6 sprints hasta llegar al plazo fijado, con 50 horas de trabajo real aproximadamente para cada sprint.

Las historias de usuario para cada sprint serán escogidas de mayor prioridad a menor, y teniendo en cuenta que otras historias necesarias para su funcionamiento ya hayan sido terminadas. Por ejemplo, la modificación del texto de un mensaje publicado en un grupo de lectura, no será realizada hasta que el grupo pueda ser creado, se puedan invitar a otros miembros y publicar el mensaje.

Como no tenemos datos históricos de la velocidad a la que completamos puntos de historia ni se han completado proyectos similares que puedan servir de aproximación, podemos establecer la velocidad inicial de dos modos: realizar un sprint y tomar la velocidad real, o realizar una estimación de la velocidad. En este caso vamos a realizar una estimación “optimista” considerando que la velocidad se corresponde con los puntos de historia planificados por sprint. Como tenemos 77 puntos de historia y 6 sprints para completarlos, en cada sprint habría que realizar 13 puntos de historia aproximadamente, por lo tanto, se toma 13 puntos de historia por sprint como velocidad inicial.

Por último, es importante comentar que las estimaciones de las tareas en las que se descompone cada historia de usuario, no se estiman en puntos de historia, sino en horas de trabajo ideales. Por facilitar el visionado del lector las siguientes tablas se muestran con posicionamiento de página horizontal.

Primer sprint:

Índice HU	Título	Tareas	Estimación horas ideales	Estimación Inicial PH	Prioridad Inicial
1	Registro	• Configuración inicial servidor (bases de datos, configuración de Spring, estructura…) • Crear clase de modelo de Usuario en servidor + mapeo BBDD. • Crear clases y métodos de repositorio/servicio/controlador. • Configuración inicial web (index.html, routing, estructura…) • Función de DataService para envío de datos al servidor desde el cliente. • Controlador web de registro. • Vista de registro con formulario.	• 4 • 2 • 4 • 4 • 2 • 2 • 2	8	M
4	Login	• Estudio de mecanismos de autenticación en API's REST. • Implementación de autenticación en servidor. • Métodos en repositorio/servicio/controlador para comprobar que el usuario existe. • Función de DataService para solicitar login y tratar token devuelto. • Controlador web de Login. • Vista de login con formulario con tratamiento de errores.	• 5 • 5 • 3 • 3 • 2 • 3	5	M

Tabla 2.2

Consideraciones sobre la planificación del primer sprint:

Como podemos ver, estamos en la etapa de inicialización del proyecto, lo cual implica invertir un número de horas de trabajo importantes en la inicialización de los proyectos, configuración, creación y conexión con BBDD y estudio del tipo de autenticación que usaremos para controlar el acceso a los métodos de la API.

También hay que establecer la comunicación de los dos subsistemas, API REST y cliente web, creando las estructuras y clases necesarias para ello.

Se escogen las historias de usuario “Registro” y “Login” que son básicas para cualquier tipo de funcionalidad posterior, evidentemente, ambas tenían el máximo nivel de prioridad posible.

Se estructura la descomposición en tareas de cada historia de usuario de forma que vayamos del nivel más “interno” del servidor que será la clase que implementa el modelo, hasta el controlador, que será el encargado de realizar la comunicación con el cliente web. Igual aproximación usamos en el cliente, en primer lugar, se implementan los métodos del DataService para comunicación con la API, luego el controlador que está asociado con una vista, y por último la vista.

Se completan 13 puntos de historia, lo cual, según la planificación inicial, es adecuado y un total de 41 horas de trabajo ideal, lo cual es apropiado, teniendo en cuenta que se parten de 50 horas reales aproximadamente, por sprint.

Segundo sprint:

Índice HU	Título	Tareas	Estimación H.I.	Estimación Inicial	Prioridad Inicial
5	Creación de libro	• Crear clase de modelo de Libro en servidor + mapeo. • Crear clases y métodos de repositorio/servicio/controlador. • Función web DataService para el envío de información al servidor. • Crear controlador web para creación de libro. • Crear vista con formulario de creación de libro.	• 2 • 2 • 1 • 2 • 2	2	M
6	Listado de libros	• Crear métodos de repositorio/servicio/controlador en servidor. • Función web DataService para realizar la petición al servidor. • Crear controlador de libros. • Crear vista con el listado de libros.	• 3 • 2 • 2 • 4	2	M
10	Añadir libro a estantería	• Crear clase de modelo de Lectura en servidor. • Crear array de Lecturas en la clase de modelo Usuario. • Crear métodos para añadir una lectura a un usuario. • Clases y métodos de repositorio/servicio/controlador para dar soporte a la adición de lecturas de libros por usuarios. • Función web en DataService para realizar la petición al servidor. • Función en controlador de libros para agregar libro a biblioteca para libros que no estén ya. • Botón en vista de libros para añadir un libro, si el libro no está en la estantería ya.	• 3 • 1 • 1 • 3 • 1 • 3 • 2	5	M
11	Estantería	• Crear métodos para obtener las lecturas de un usuario. • Métodos en clase de modelo Usuario para obtener número de lecturas, y puntuación media asignada a libros. • Métodos de repositorio/servicio/controlador para obtener lecturas de usuario. • Función en web DataService para solicitar lecturas de un usuario. • Crear controlador web de estantería. • Estudio de directivas para realizar y mostrar puntuaciones mediante estrellas, e integración. • Vista de estantería.	• 1 • 1 • 3 • 1 • 2 • 5 • 3	5	M

Tabla 2.3

Consideraciones sobre la planificación del segundo sprint:

Como podemos ver, una vez que se han realizado los trabajos iniciales de estudio y configuración, pasamos a tener una carga de trabajo orientada a la implementación de nuevas funcionalidades.

Se escogen las historias de usuario “Creación de libro”, “Listado de libros”, “Añadir libro a estantería” y “Estantería” que ya implementan funcionalidades importantes para el usuario.

Al igual que en el sprint anterior, se estructura la descomposición en tareas de cada historia de usuario de forma que vayamos del nivel más “interno” del servidor que será la clase que implementa el modelo, hasta el controlador, que será el encargado de realizar la comunicación con el cliente web. Igual aproximación usamos en el cliente, en primer lugar, se implementan los métodos del DataService para comunicación con la API, luego el controlador que está asociado con una vista, y por último la vista.

Se completan 14 puntos de historia, lo que significa que es muy probable que la última historia no se pueda completar al 100%, pasando el trabajo restante al siguiente sprint.

En cuanto al número de horas de trabajo ideales, podemos ver que se consideran 50. Este resultado también está en consonancia con lo mencionado en el párrafo anterior, puesto que es casi imposible que en 50 horas de trabajo real se realicen 50 horas de trabajo ideales, por lo que habrá que pasar carga de trabajo al sprint siguiente.

Para las historias de usuario restantes la descomposición en tareas consistiría en implementar métodos en capa de servicio y controlador, crear métodos en DataService, crear métodos en el controlador del cliente y asociarlo con la acción del botón de la vista. Se muestran las tablas de los sprints ordenadas por orden de prioridad a realizar.

Tercer sprint:

Índice	Título	Historia de Usuario	Estimación	Prioridad
12	Modificar estado de lectura	Como usuario, quiero modificar el estado actual en el que se encuentra la lectura de un libro de mi estantería.	3	M
13	Modificar puntuación de lectura	Como usuario, quiero puntuar libros de mi estantería	2	M
14	Modificar comentario	Como usuario, quiero realizar un comentario sobre un libro de mi estantería que haya leído.	2	M
23	Creación grupos de lectura	Como usuario, quiero crear grupos de lectura privados.	5	M

Cuarto sprint:

Índice	Título	Historia de Usuario	Estimación	Prioridad
24	Añadir miembros	Como usuario, quiero añadir miembros a un grupo de lectura.	5	M
25	Publicar mensajes	Como usuario, quiero publicar mensajes en un grupo de lectura, para poder comunicarme con los demás miembros.	3	M
7	Ver información de libro	Como usuario, quiero poder ver la información completa de un libro y las lecturas que tiene por otros usuarios.	2	S
15	Añadir autor	Como usuario, quiero poder añadir autores y especificar su información e imagen para que puedan ser listados.	2	S

Quinto sprint:

Índice	Título	Historia de Usuario	Estimación	Prioridad
16	Listado de autores	Como usuario, quiero ver un listado de todos los autores disponibles.	2	S
20	Búsqueda de libros	Como usuario, quiero buscar libros por título.	3	S
21	Búsqueda de autores	Como usuario, quiero buscar autores por nombre.	3	S
22	Búsqueda de usuarios	Como usuario, quiero buscar usuarios por nombre.	3	S
2	Modificar datos de usuario	Como usuario, quiero poder ver y modificar mis datos de acceso e imagen de perfil.	2	C

Sexto sprint:

Índice	Título	Historia de Usuario	Estimación	Prioridad
3	Borrado de usuario	Como usuario, quiero poder eliminar mi cuenta en cualquier momento para que mi información sea eliminada.	3	C
8	Modificar libro	Como administrador, quiero añadir y modificar datos de un libro existente.	2	C
9	Borrar libro	Como administrador, quiero borrar un libro falso, o con contenido inapropiado.	2	C
17	Ver información de autor	Como usuario, quiero ver información pormenorizada de autores y listado de libros que han escrito.	2	C
18	Modificar autor	Como administrador, quiero modificar información e imagen de un autor	2	C
19	Borrar autor	Como administrador, quiero borrar información de autores falsos.	1	C
26	Modificar mensajes	Como usuario, quiero modificar mis mensajes publicados en el grupo de lectura.	2	C
	Borrar	Como usuario, quiero borrar		

27	Mensajes	mis mensajes publicados en el grupo de lectura.	1	C

2.6.- Modelo de dominio

En esta sección se intentará dar una visión general de las clases que compondrán el modelo del servidor y como se relacionan entre ellas, mediante la presentación del modelo de dominio[23].

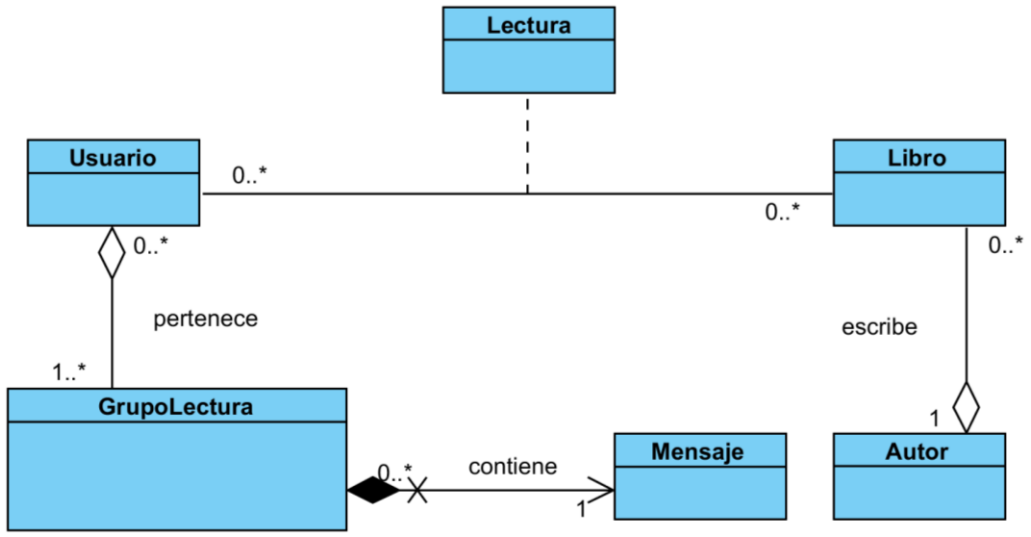

Ilustración 2.7

Aspectos que se han tenido en cuenta para llegar a este modelado:

En el diagrama aportado se presentan las clases del modelo sobre las que construiremos la aplicación. Se presentan su nombre, relaciones con otras clases, multiplicidad y navegabilidad. Es importante mencionar que en la sección de diseño se realiza un estudio más exhaustivo sobre las relaciones existentes, sobre un diagrama de clases de diseño más desarrollado.

Partiremos de la clase Usuario en la cual que pueden encontrar dos relaciones importantes:

Usuario – Libro: evidentemente un usuario debe poder relacionarse con los libros de alguna forma. Además, para esta relación se deben guardar

valores como el estado actual de la lectura del libro por el usuario, la puntuación que el usuario da al libro, y el comentario que realiza sobre el mismo. Estas características provocan que tengamos que usar una clase de asociación [24] que llamaremos “Lectura”. Un usuario puede realizar lecturas de cero o muchos libros, al igual que un libro puede ser leído por cero o muchos usuarios, pero una lectura sólo relaciona a un usuario con un libro en concreto. La navegabilidad se considera bidireccional. Una descripción mucho más desarrollada sobre esta asociación se dará en la parte de diseño.

Usuario – GrupoLectura: Un usuario puede pertenecer a cero o muchos grupos de lectura, pero un grupo de lectura al menos tiene que tener un usuario. Podemos ver que el usuario agrega grupos, por lo tanto no es el usuario quien posee la lógica de creación de los mismos.

Sobre la clase GrupoLectura destacamos: en este caso, la relación con la clase “Mensaje” es de composición, lo cual indica que el grupo es el encargado de la creación de mensajes. Un grupo puede tener cero o muchos mensajes. Además, en este caso, la navegabilidad es unidireccional, por lo que el Grupo tiene acceso a sus mensajes, pero desde un mensaje no podemos acceder al grupo que lo contiene.

Sobre Autores y su relación con libro podemos destacar: Un autor puede tener cero o muchos libros, y un libro cero o muchos autores, la navegabilidad es bidireccional y como se muestra, se relacionan mediante agregación, con lo cual no se fuerza a que el autor del libro exista previamente para poder crear un libro.

3.- DISEÑO

En este apartado se detallan los aspectos relacionados con la parte de diseño del proyecto. Los diagramas mostrados son el resultado final de las iteraciones realizadas mediante metodología SCRUM.

En primer lugar, se expone el diagrama entidad-relación obtenido desde el modelo de dominio expuesto en el apartado anterior.

A continuación, se exponen los diagramas de clases de diseño realizados tanto para el servidor como para el cliente web y se comentan los aspectos más relevantes, como estructura de los sistemas, decisiones tomadas durante el diseño, patrones aplicados…

El siguiente aspecto a tratar son los diagramas de secuencia de algunas de las historias de usuario que se han considerado más representativas y que muestran de una forma global y generalizada el funcionamiento interno del sistema.

Por último, se muestra un diseño preliminar de las pantallas de usuario del sistema cliente y su interacción mediante storyboards de las historias de usuario más relevantes.

3.1.- Diagrama Entidad-Relación

El diagrama Entidad-Relación [25] ha sido obtenido mediante Visual Paradigm desde el modelo de dominio.

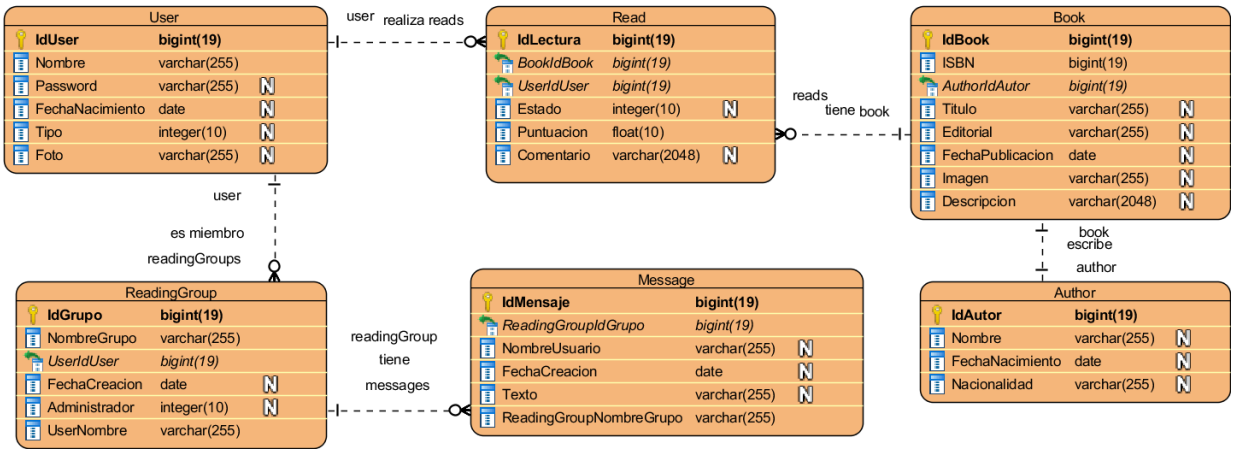

Ilustración 3.1

- Resumen de los aspectos más relevantes:
- La existencia o falta de claves foráneas definen la navegabilidad entre clases.
- La parte de la relación marcada con una línea indica multiplicidad 1.
- La parte de la relación marcada con un círculo indica multiplicidad 0 … *.
- Fechas de nacimiento y publicación son de tipo date (no contiene información de hora), fechas de creación de grupo de lectura y de mensaje son de tipo timestamp (información de fecha y hora).
- Id’s de tipo Long autogenerados.

Es importante mencionar que estas tablas, relaciones e id's serán generadas automáticamente por el ORM utilizado (sobre el cual entraremos en detalle más adelante) a partir de las clases del modelo. Aun así, se ha verificado que esté normalizado hasta la Tercera Forma Normal, basándonos en el concepto de dependencia funcional. Para ello:

- Se eliminan valores calculados y atributos derivados.
- Se crea la relación de forma no normalizada.
- Se aplica la Primera Forma Normal (todas las intersecciones de filas con columnas contienen valores indivisibles).

- Se aplica la Segunda Forma Normal (cada uno de los atributos dependen funcionalmente de la clave principal).
- Se aplica la Tercera Forma Normal (cada atributo depende sólo de la clave principal y no de ningún otro atributo no clave).

Por último, se comentan algunas consideraciones que se han tenido en cuenta:

Las relaciones más relevantes son las de lectura de un libro por un usuario, esta relación obliga a la creación de una tabla intermedia para poder almacenar estado, puntuación y comentario del usuario sobre el libro. Evidentemente, un usuario puede tener muchas lecturas al igual que un libro, pero una lectura sólo puede relacionar a un libro y usuario concretos, por lo tanto, tendremos relaciones de muchos a uno entre User y Read y uno a muchos entre Read y Book. Además, un usuario debe conocer los libros de sus lecturas, al igual que un libro debe conocer los usuarios que han realizado lecturas, lo cual implica relaciones bidireccionales entre las tablas.

Otra relación fundamental es la existente entre usuarios y grupos de lectura, donde un usuario puede pertenecer a varios grupos y un grupo, evidentemente, puede tener muchos usuarios. La relación resultante es bidireccional y con multiplicidad muchos a muchos.

Un aspecto importante a destacar es el tratamiento de borrado de entidades y cuándo se producirán borrados en cascada o no. Por ejemplo, al borrar un usuario, serán eliminadas en cascada todas sus lecturas, y para mantener la integridad referencial, se deben eliminar las relaciones de libros a estas. Por otra parte, la eliminación de un usuario no implica el borrado de los grupos de lectura a los que pertenece, pero sí el borrado del usuario de la lista de miembros, y de todos los mensajes que haya publicado.

3.2.- Diagrama de Clases

En esta sección veremos los diagramas de clases que dan una visión más detallada de la aplicación web. A modo de introducción, es interesante resaltar que la aplicación está formada en realidad por dos sub-aplicaciones.

En el lado del servidor, es donde se realiza toda la lógica de negocio, consultas y modificaciones sobre los datos. Toda la funcionalidad implementada queda expuesta mediante una API REST.

Por otra parte, tenemos una aplicación web cliente que se comunica con la API REST para obtener los datos a mostrar y las modificaciones a realizar.

Para ambas se ha utilizado una arquitectura MVC, que más adelante se materializará con SpringMVC/Spring y AngularJS, como se detallará más adelante en la fase de implementación.

Para facilitar el visionado de los diagramas por parte del lector se ha optado por incluir cada uno de ellos en una página independiente con alineado horizontal. En la página siguiente a cada diagrama, se realizan los comentarios e indicaciones que se han considerado oportunas para cada uno de ellos.

Además, todos los diagramas que se incluyen en este apartado pueden encontrarse en el archivo de Visual Paradigm que se adjunta en el soporte físico de entrega del trabajo.

3.2.1.- Diagrama de clases del sistema servidor.

3.2.1.1.- Paquete *Model*:

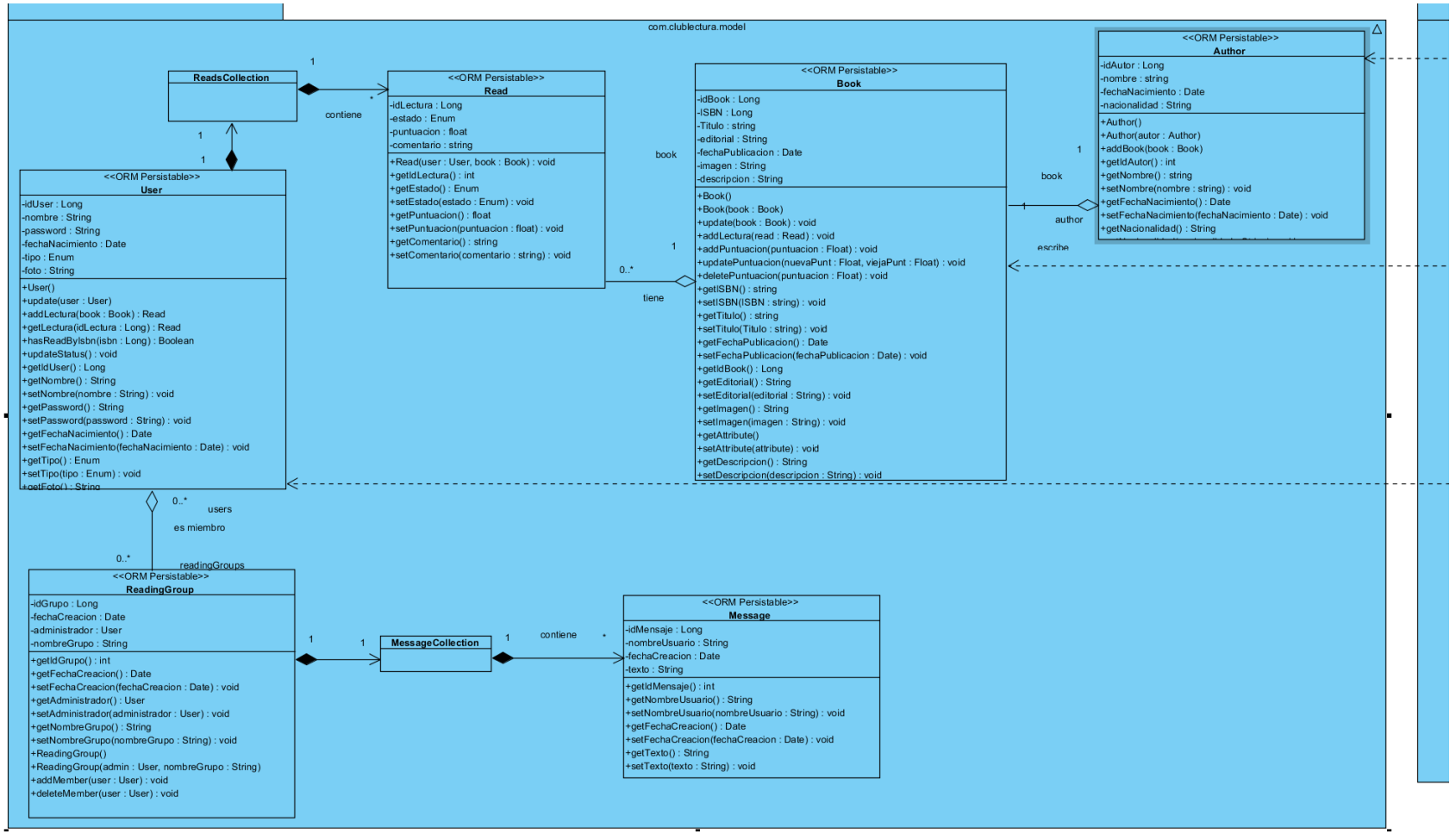

Ilustración 3.2

Este paquete representa las clases de objetos base sobre las que funciona el sistema y sus relaciones, se obtiene de una forma casi directa desde el modelo de dominio expuesto en el apartado de análisis, aunque se han realizado las siguientes transformaciones para obtener el paquete de modelo actual:

- Añadidos constructores utilizados para cada clase.
- Queda definida la navegabilidad, la multiplicidad en los extremos, y el nombre de asociación o rol en, al menos, el destino.
- Definición de asociaciones en agregación o composición.
- Las asociaciones uno a muchos incluyen una colección de objetos en el lado del que parte la relación. Son clases especializadas en gestionar la colección objetos de un tipo determinado encargándose de adición, eliminación, búsqueda, recorrido …
- La clase de asociación *Read* entre la asociación muchos a muchos de *User* y *Book* no está directamente soportada por los lenguajes de programación orientados a objetos habituales. Se sustituye la clase de asociación por una clase y se usa una combinación de agregación y composición para representar la semántica de la clase de asociación.

3.2.1.2.- Paquetes *Repository* y *Service*.

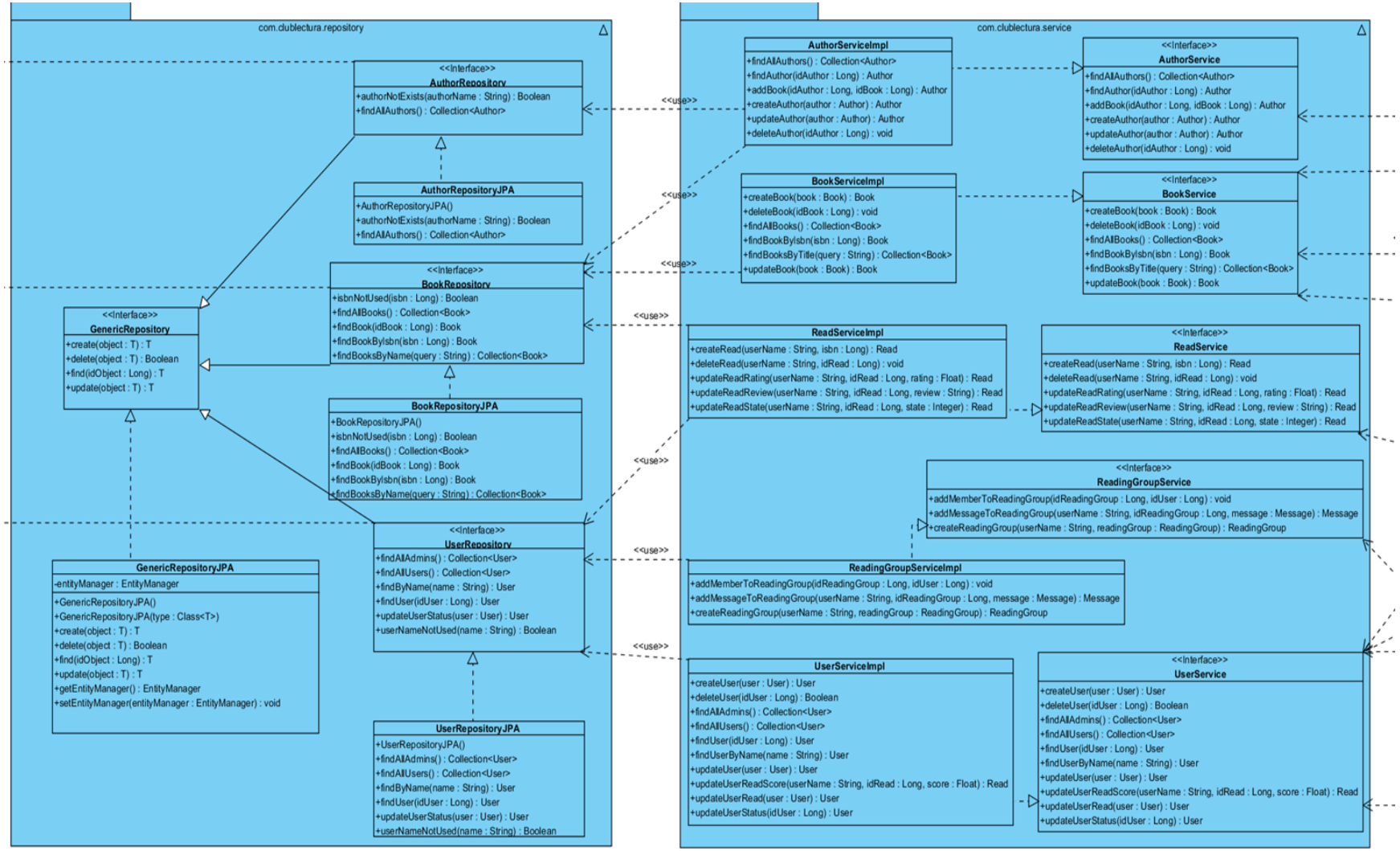

Ilustración 3.3

Algunos de los aspectos destacados del diagrama de estos dos paquetes son:

El paquete *Repository* incluye toda la interacción necesaria con la base de datos.

Se usa una interfaz genérica con métodos comunes a todos los repositorios existentes, de la cual heredan las interfaces de cada clase. Se incluyen implementaciones de estas interfaces orientadas a bases de datos relacionales usando JPA. Es importante hacer notar que, con esta estructura, podríamos realizar un cambio a una base de datos orientada a objetos, con sólo incluir las implementaciones necesarias de estas interfaces con *Hibernate*, por ejemplo. Las clases y métodos definidos en este paquete son usados por el paquete *Service* de forma exclusiva.

El paquete *Service* realiza funciones típicas de la capa de servicio de cualquier aplicación.

Usa las interfaces de repositorio para realizar cambios en el modelo y provee métodos que serán utilizados por la capa de controladores, proporcionándoles una interfaz lo más sencilla posible y abstrayéndolos de la parte más importante de la lógica de negocio, la cual se realizará en este paquete.

3.2.1.3.- Paquete *Controller*.

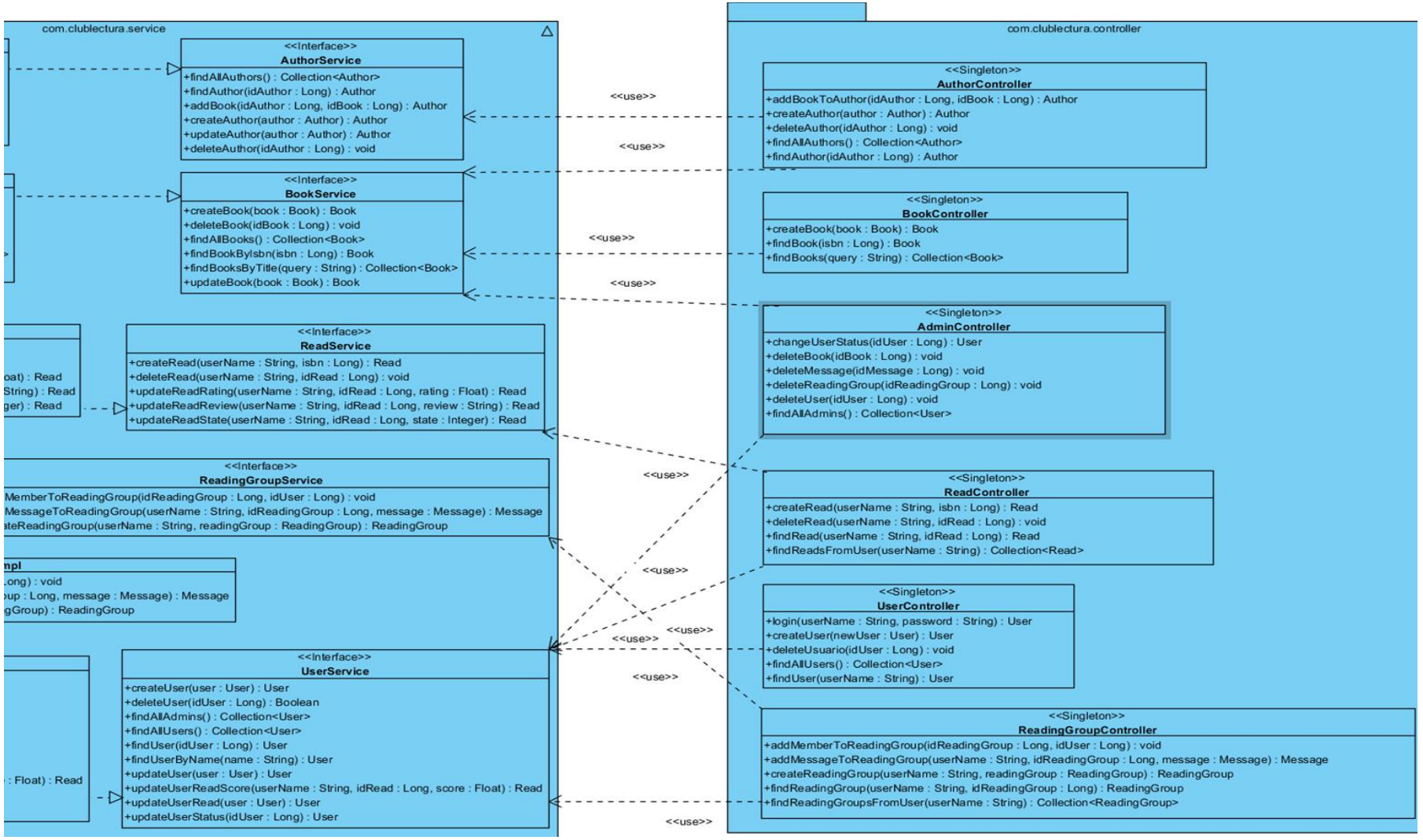

Ilustración 3.4

Algunas de las consideraciones sobre este paquete son:

Las clases del paquete controlador implementan el punto de entrada a la API REST desde el exterior.

Cada controlador define una ruta de entrada común, y con cada método implementado se especifican partes de ruta adicionales y el método para acceder, también establecen la información y formato que deben recibir y devolver.

Estos métodos son utilizados por el *framework* para atender las peticiones asíncronas que pueden ser realizadas desde cualquier cliente y servir los resultados de forma transparente.

Además, todas las funcionalidades de validación de la información entrante en cada uno de los métodos expuestos, se delegan al *framework* y se explicarán con detalle en el apartado de implementación.

3.2.2.- Diagrama de clases del sistema cliente.

Pasamos al diagrama del cliente web, donde se utiliza también el patrón MVC, soportado en este caso por el framework AngularJS.

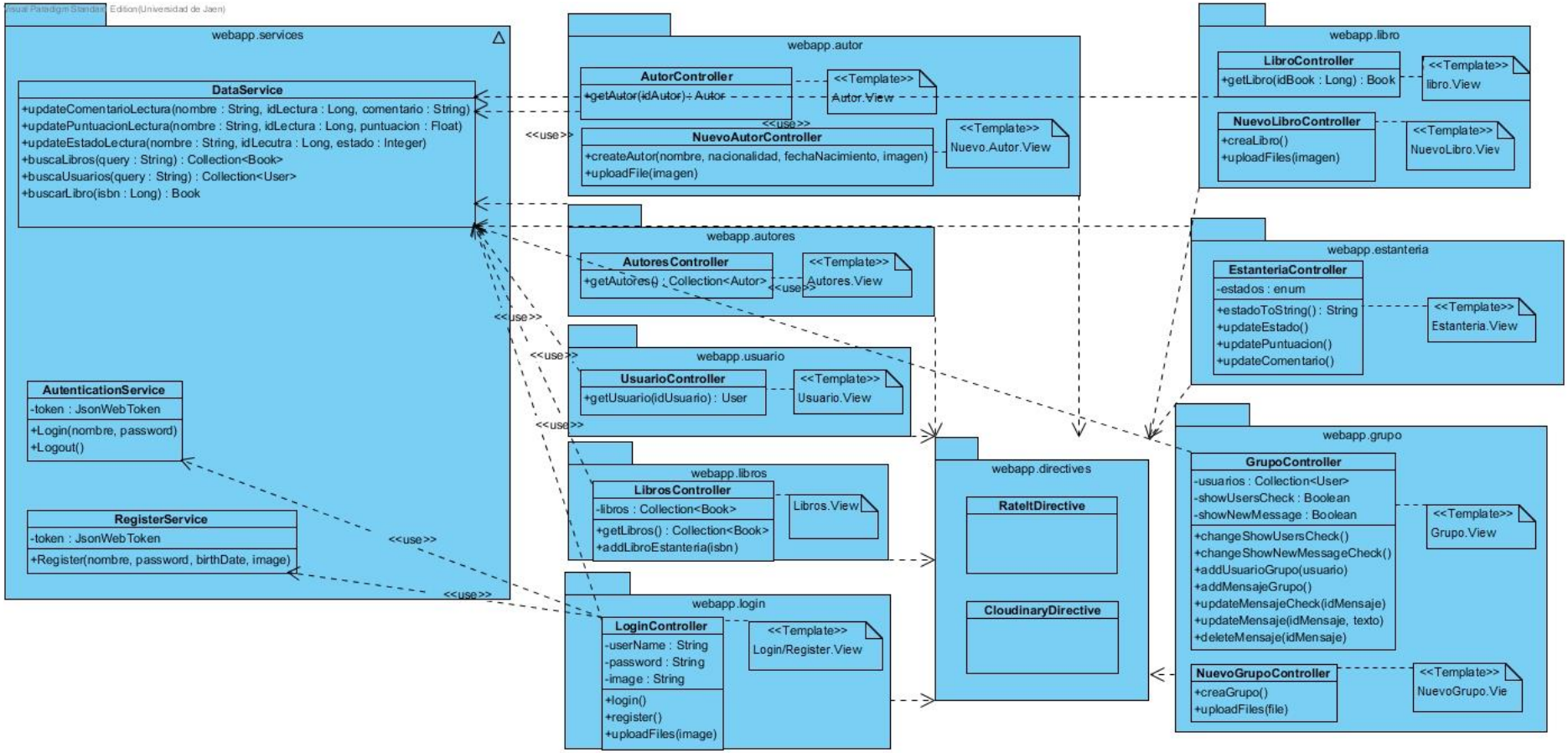

Ilustración 3.5

Aspectos relevantes sobre el diagrama:

En primer lugar, es necesario mencionar que se ha elegido una práctica habitual en proyectos de este tipo para estructurar la forma en la que se agrupan controladores y vistas. Normalmente, un controlador se asocia con una vista únicamente y estos se guardan juntos en una carpeta identificativa de la funcionalidad que otorgan. Esto provoca una estructura limpia, intuitiva y más fácilmente navegable que si se agrupasen todos los controladores por un lado y todas las vistas por otro.

El paquete webapp.services es el encargado de la comunicación con la API REST del servidor mediante llamadas asíncronas. Obtiene todos los datos necesarios para los controladores y comunica modificaciones a realizar sobre el modelo al servidor. El *DataService* debe contemplar que las peticiones provienen de un cliente autorizado, para ello se encarga de gestionar información adicional en cada petición para realizar la autorización: un *token* de acceso devuelto por el servidor al hacer login o registrar un usuario. Este mecanismo de autenticación se tratará con profundidad en la parte de implementación.

Los controladores están asociados con alguna vista y precargan la información que la vista necesita. También se encargan de propagar hacia el DataService las modificaciones a realizar en el modelo, así como de hacer todo el tratamiento de datos necesario que la vista demande.

Las vistas muestran la información al usuario mediante archivos HTML que usan directivas de angular, y otras definidas en el paquete directivas para realizar una funcionalidad concreta en la web.

El paquete de directivas tiene la definición de funcionalidades auxiliares para el funcionamiento de la web. Por ejemplo, Rate-it y Cloudinary que se encargan de mostrar y captar puntuaciones sobre inputs gráficos como estrellas, y de subir imágenes a la nube en el formato apropiado, respectivamente. Estas funcionalidades serán explicadas con detalle en el apartado de implementación.

3.3.- Diagramas de secuencia

El objetivo de este apartado es comprender el funcionamiento del sistema modelado en los diagramas de clases anteriores. Se han escogido los diagramas más representativos y que dan una visión lo más extensa posible, de tal forma que el comportamiento del resto de funcionalidades se puede deducir fácilmente a partir de los presentados.

Para facilitar el visionado de los diagramas por parte del lector se ha optado por incluir cada uno de ellos en una página independiente con alineado horizontal. En la página siguiente a cada diagrama, se realizan los comentarios e indicaciones que se han considerado oportunas para cada uno de ellos.

Además, todos los diagramas que se incluyen en este apartado pueden encontrarse en el archivo de Visual Paradigm que se adjunta en el formato físico de entrega del trabajo.

3.3.1 Login

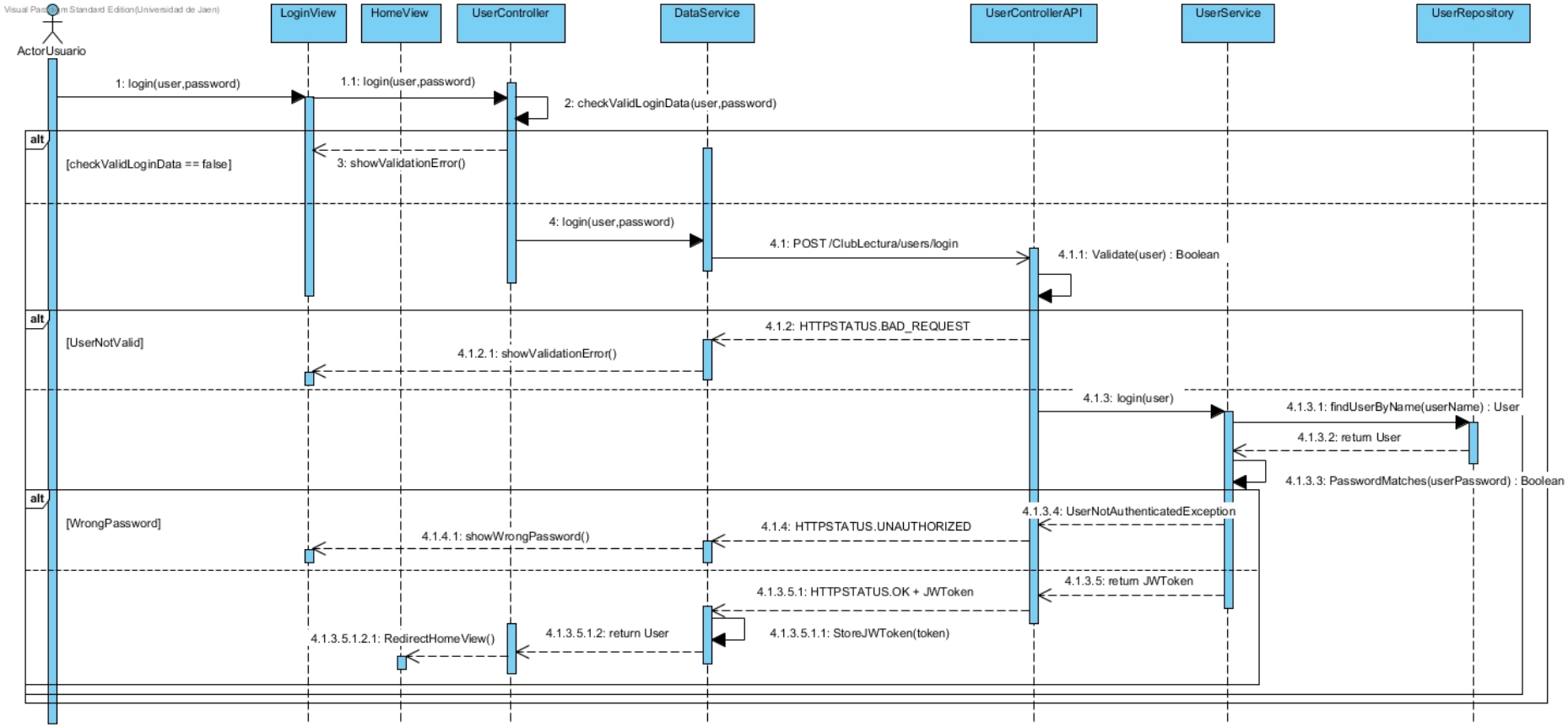

Ilustración 3.6

Algunas consideraciones sobre el diagrama presentado:

Cuando un usuario intenta acceder a la aplicación web se le redirige a la página de login, desde la cual se solicitan los datos de acceso.

El usuario introduce *username* y *password*, si tienen el formato apropiado la llamada de login se propaga hasta el DataService donde se realiza una llamada asíncrona al servidor para que realice la autenticación.

El servidor realiza una validación del contenido json recibido, y llama a la capa de servicio para que realice la autenticación, genere y devuelva el Json Web Token (usado para autenticar al emisor de llamadas, se verá con más detalle en la parte de implementación) generado si hay éxito. Se traslada al cliente un código http de éxito, y el token de acceso para sucesivas llamadas.

El DataService almacena el token para poder ser usado en sucesivas llamadas a modo de autenticación, y actualiza la vista del navegador a la página principal del sitio web donde el usuario puede ver los libros que hay añadidos en su estantería.

3.3.2 Creación de libro

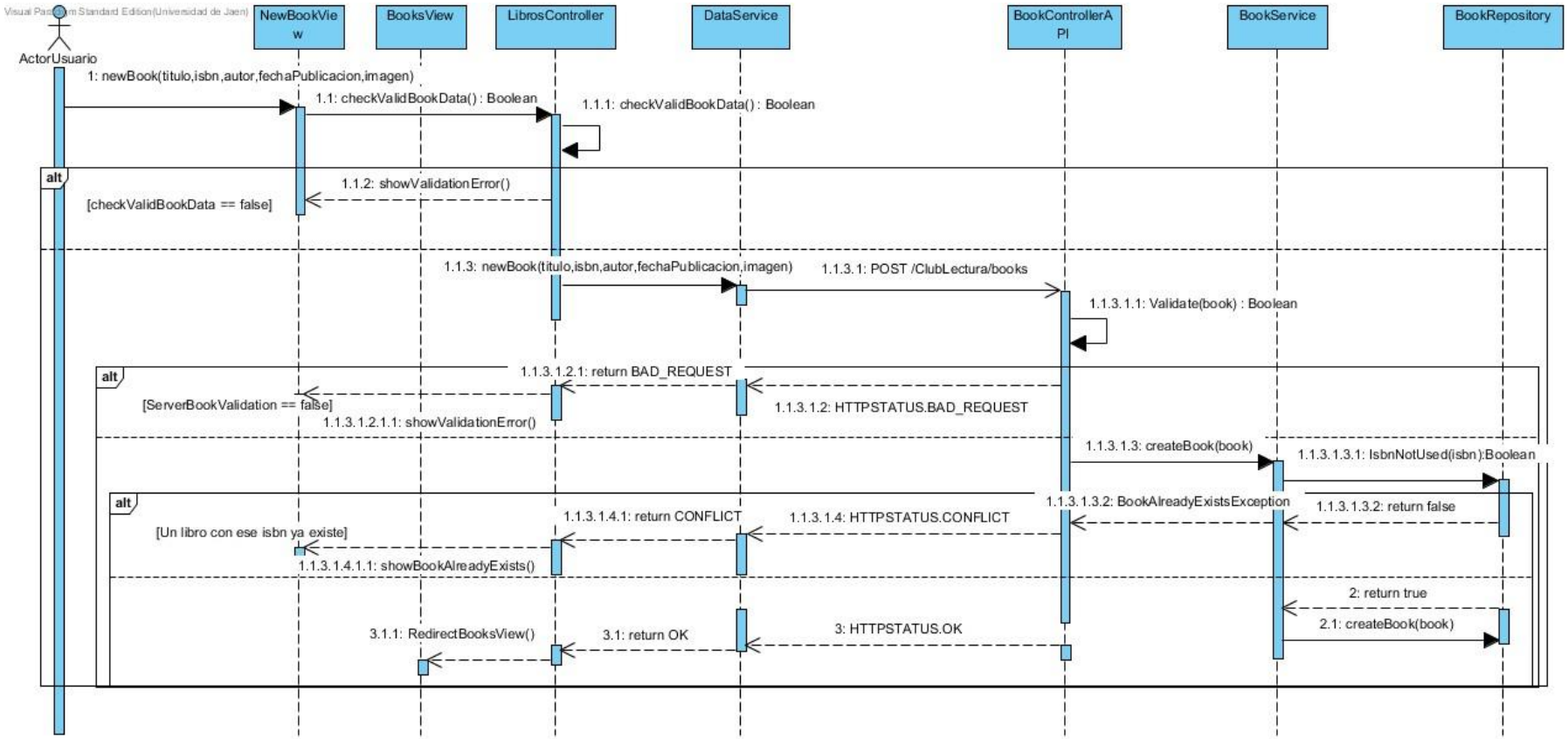

Ilustración 3.7

Algunos comentarios sobre el diagrama:

El usuario introduce la información del libro: título, isbn, fecha de publicación e imagen de portada.

Tras una validación inicial de los datos en el controlador, se llama a un método del DataService al que se le pasan los parámetros introducidos, que se encarga de realizar la comunicación con la API REST. En el controlador del servidor, una vez validada la información, se procede a su almacenamiento en la BBDD. En caso de que ya exista un libro con ese mismo isbn se lanzará la excepción apropiada que se convertirá a un código http status y llegará hasta el controlador de libros del cliente web para mostrar el mensaje de error apropiado en la vista para que el usuario pueda revisar los datos introducidos.

En caso de éxito en la creación del libro, se devolverá un http status de OK y se redirigirá a la vista de libros donde ya aparecerá el libro que acabamos de introducir.

En este caso, no tiene sentido capturar y tratar en el cliente web el libro creado en el servidor, puesto que al actualizar la vista de libros hay que volver a pedir los datos al servidor, ya que se han podido producir modificaciones sobre los libros existentes o pueden haberse creado otros al mismo tiempo.

3.3.3. Añadir lectura de un libro.

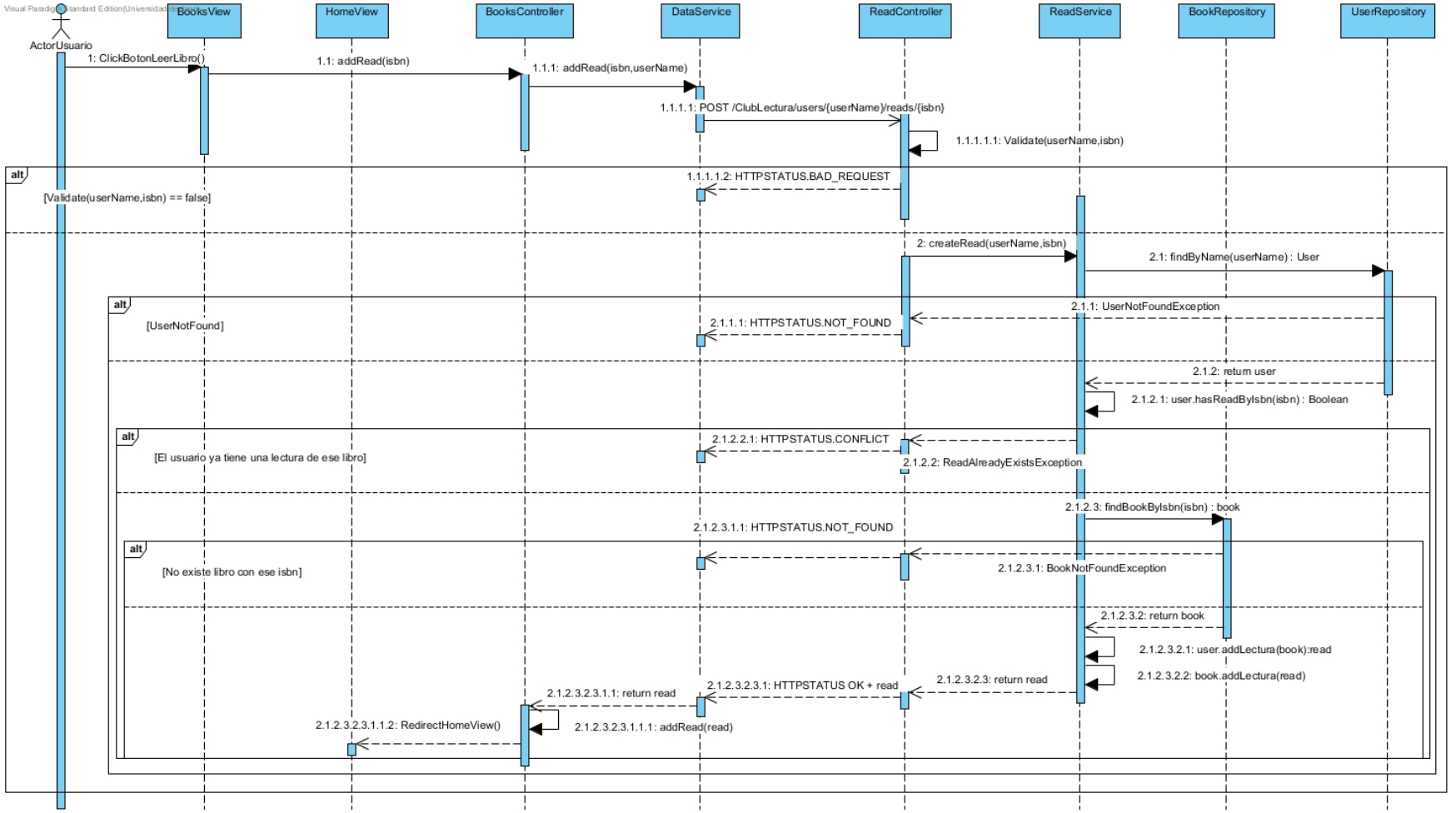

Ilustración 3.8

Algunos comentarios sobre el diagrama:

Como vemos, en este caso prácticamente toda la carga de trabajo recae sobre el servidor. La acción se desencadena cuando un usuario pulsa el botón de “Añadir a mi estantería” de un libro cuando se encuentra en la vista en la que se muestran los libros existentes. Es importante mencionar que si el libro ya se encuentra en la estantería del usuario este botón no se mostrará.

El controlador de libros del cliente llama a un método del DataService pasándole la información del usuario que realiza la petición y del isbn del libro que quiere añadir. El DataService realizará una llamada asíncrona al servidor para que la petición sea procesada.

Al llegar al servidor la información será validada para saber si tienen el formato apropiado, lanzando una excepción con el http status apropiado en caso contrario. A continuación, se pasa la petición al ReadService donde se encuentra la lógica de negocio necesaria para tratarla.

En primer lugar, se intenta obtener el usuario cuyo nombre se ha pasado, lanzando una excepción con su correspondiente http status en caso de no encontrarse. Después se comprueba que para ese usuario no exista ya una lectura del libro pasado, lanzando de nuevo excepción y http status correspondiente en caso de que ya exista.

A continuación, se intenta obtener los datos del libro cuyo isbn se ha pasado, controlando también que exista y tomando las acciones adecuadas para informar al cliente en caso contrario.

Una vez que sabemos que la operación se puede realizar, creamos una lectura del usuario para el libro indicado, y añadimos la lectura creada a las lecturas del libro.

Desde el controlador del servidor, se lanza un código http status de éxito y se pasa también la información de la lectura creada.

En el cliente se recoge la lectura creada para añadirla a la lista de lecturas del usuario, una vez hecho, se redirige a la estantería del usuario donde ya se podrá ver la nueva lectura creada.

En este caso, no es necesario realizar una petición nueva desde el cliente cada vez que se quiera ver la estantería, pues las lecturas de un usuario sólo son modificables por él mismo, y se garantiza que no hay modificaciones o nuevas lecturas que se hayan podido hacer desde la última consulta y que no hayan sido adecuadamente tratadas en el cliente.

También es importante hacer notar que en el servidor se comprueban y lanzan todos los posibles estados anómalos por los que no se pueda realizar la operación, sin embargo, en el cliente no se tratan estos posibles inputs, debido a que se garantiza que un usuario sólo pueda realizar operaciones sobre libros que existen, y sobre los que no existe ya una lectura.

3.3.4. Añadir miembros a grupo de lectura.

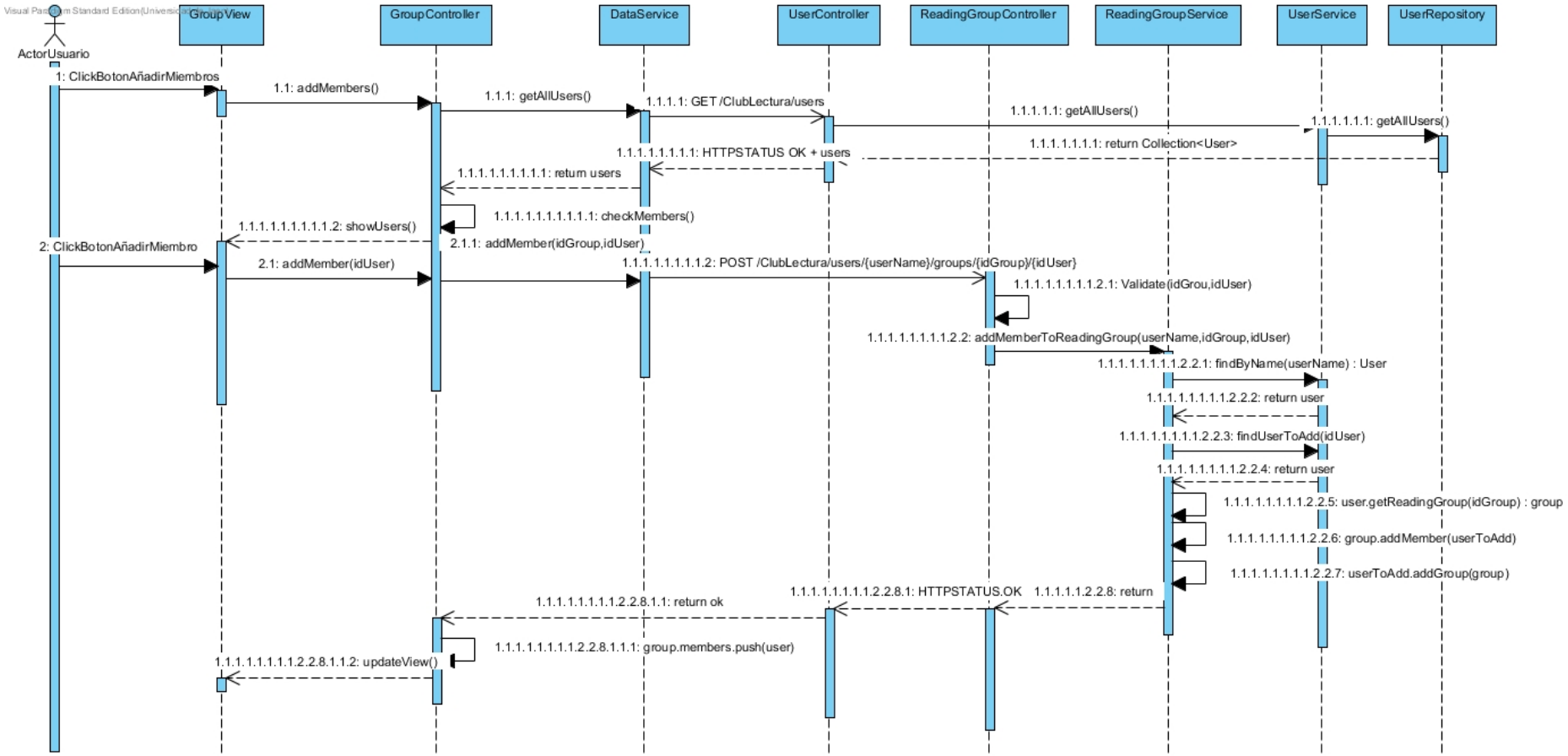

Ilustración 3.9

Algunas consideraciones sobre el diagrama:

En primer lugar, aclarar que se han obviado los retornos de códigos de error del servidor al cliente para los casos de que los usuarios o el grupo no existan y de validación incorrecta, ya que tienen un tratamiento similar al diagrama anterior, y de esta forma podemos obtener una vista más limpia del flujo de ejecución.

La ejecución comienza cuando un usuario pulsa el botón “Añadir miembros” desde la vista del grupo. En ese momento, se realiza una consulta al servidor para obtener una lista completa de los usuarios existentes.

Al llegar la información de los usuarios al controlador de la vista, se comprueban los usuarios que ya pertenecen al grupo, para los cuales el botón de “Añadir al grupo” no estará disponible.

A continuación, la vista del grupo se actualiza mostrando los usuarios que pueden ser añadidos. El usuario ya puede pulsar el botón de “Añadir al grupo” que aparecerá tras la información del usuario.

La petición de añadir un usuario al grupo se propaga hasta la capa de servicio correspondiente del servidor, donde se comprueba que los usuarios, y el grupo de lectura existen y que el usuario que realiza la petición, pertenece al grupo sobre el cual quiere añadir otro usuario.

Una vez comprobado que se puede realizar la operación, se añade el usuario al grupo, y se añade el grupo a la lista de grupos a los que pertenece el usuario a añadir.

4.- IMPLEMENTACIÓN

4.1.- Arquitectura

Diagrama arquitectónico del sistema desarrollado, donde se observen los elementos constituyentes, ya sean desarrollados o externos, y la relación entre ellos, explicando brevemente la función de cada uno de ellos.

En este apartado se intentan aportar diagramas donde se observe la arquitectura utilizada en cada uno de los subsistemas, los diferentes elementos que los componen, su función y la relación existente entre ellos.

Como ya se ha comentado en apartados anteriores, realmente el sistema está compuesto por dos subsistemas independientes. Veremos en primer lugar el servidor, después el cliente web, y por último, un breve comentario sobre la comunicación entre ambos.

4.1.1 Servidor.

En el apartado 3 de diseño de clases, vimos cómo se estructuran internamente las diferentes capas de tal forma que cada una de ellas se comunicaba única y exclusivamente con su capa superior e inferior. Se puede considerar que las capas de modelo, repositorio y servicio se encargan de las modificaciones sobre el modelo y de la lógica interna de negocio. La capa más externa de controladores se encarga de la comunicación con otros sistemas e internamente Spring MVC la implementa de la siguiente forma:

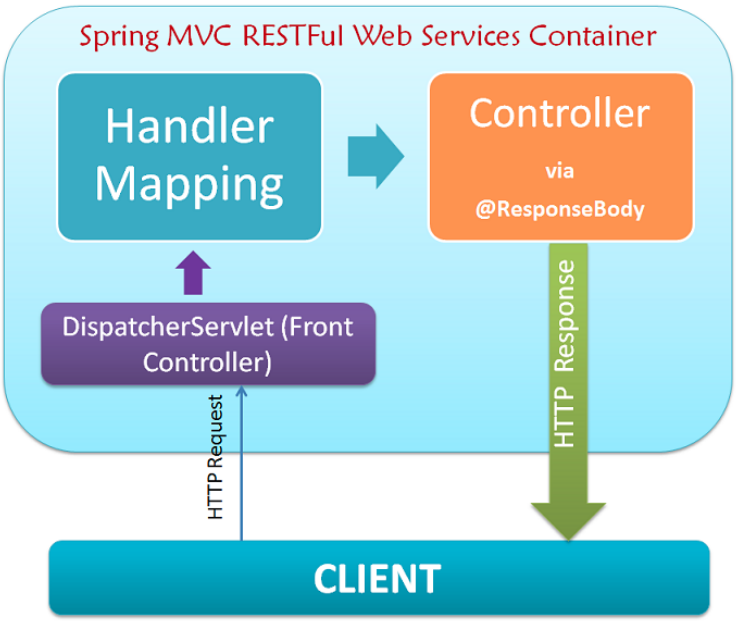

Ilustración 4.1 http://www.programming-free.com/2014/01/spring-mvc-40-restful-web-services.html

El flujo de ejecución sería:

El dispatcher servlet recibe una petición desde un servicio AngularJS, por ejemplo, y el handler mapping es el encargado de seleccionar el método del controlador apropiado para atenderla. El controlador realizará las llamadas necesarias a la capa de servicio para procesar la petición y devolver la información solicitada.

En nuestro caso, no necesitamos realizar la carga de la vista a mostrar, puesto que exponemos un servicio REST, por eso, generamos una respuesta HTTP directamente con el contenido Json apropiado cuando corresponda, acompañado por el código resultante de la operación.

4.1.2.- Arquitectura cliente web.

AngularJS está también basada en una arquitectura MVC [26] que implementa de la siguiente forma:

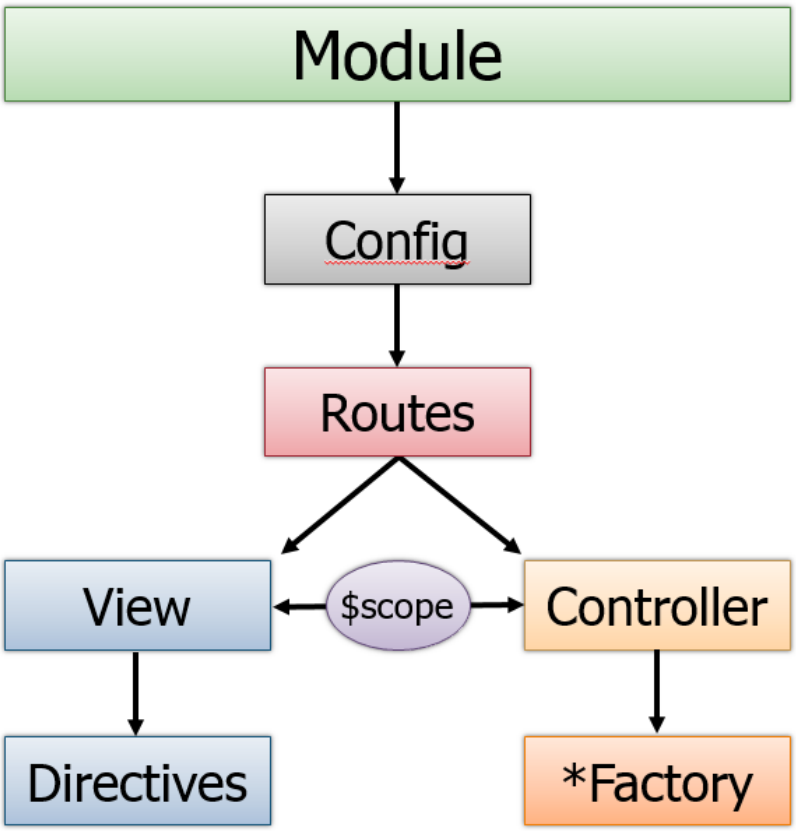

Ilustración 4.2 http://weblogs.asp.net/dwahlin/video-tutorial-angularjs-fundamentals-in-60-ish-minutes

Para poder utilizar una aplicación web con Angular tenemos que definir el módulo de nuestra aplicación, sobre el cual podemos aplicar diferentes tipos de configuración y además, hay que definir las rutas que se utilizarán. En este paso, es donde se establecen las asociaciones entre vistas y controladores

Una factoría de Angular, en nuestro caso un servicio, se encargan de obtener el modelo de datos sobre el cual trabajará la aplicación mediante llamadas asíncronas. Estos datos estarán disponibles para los controladores que realizarán el tratamiento oportuno sobre ellos, permitiendo que sean representados en las vistas. Además, mediante el $scope se implementa un patrón observador bidireccional que propaga los cambios en el modelo entre vista y controlador independientemente de dónde se modifiquen.

La comunicación entre ambos se realiza mediante un servicio REST que define el recurso al que se accede mediante una ruta, un método (GET, POST, PUT, DELETE) y opcionalmente, en los casos que sea necesario un cuerpo de petición en

formato JSON. En el apartado siguiente desarrollamos en detalle cómo queda definida esta comunicación.

4.2.- Detalles sobre implementación

En este apartado, serán comentados aspectos metodológicos y técnicos sobre la implementación, recursos empleados, bibliotecas y servicios externos usados. Se tratarán aspectos concretos de los mismos como versiones utilizadas, dependencias, configuración… También se mostrarán ejemplos extraídos del propio código del proyecto para explicar cómo se realiza la implementación de aspectos relevantes del diseño.

4.2.1.- Implementación en el servidor.

La implementación del lado del servidor está basada en Java EE 7 Web [27], [28] sobre la cual se puede encontrar información exhaustiva de sus componentes en [29]. Utilizaremos NetBeans 8.1 y un proyecto MAVEN que nos permitirá construir la aplicación y controlar dependencias y versiones usadas de una forma potente. La versión de Spring sobre la que trabajaremos será la 4.0.1. Una vez que se ha expuesto la base sobra la que se construye el proyecto, vamos a entrar en detalle en diferentes aspectos de la implementación.

4.2.1.1.- Persistencia.

Para realizar la persistencia de la aplicación nos basaremos en el uso de JPA [30] 2.1 sobre el ORM Hibernate [31], [32] en su versión 4.3.6, para conseguir la autogeneración de tablas sobre una base de datos relacional, a partir del código de las clases Java que escribamos. Esto nos permite poder cambiar entre diferentes bases de datos relacionales, con sólo cambiar unas líneas de configuración de Spring. La forma en la que se realiza el mapeo se puede manipular mediante el uso de anotaciones de JPA sobre los atributos o métodos de las clases Java.

```java
@Entity
@JsonIdentityInfo(generator=ObjectIdGenerators.PropertyGenerator.class, property="idBook")
public class Book implements Serializable {

    @Id
    @SequenceGenerator(name = "bookId", sequenceName = "book_Id")
    @GeneratedValue(strategy = GenerationType.AUTO, generator="bookId")
    private final long idBook;
    @NotNull
    private long ISBN;
    @NotEmpty
    private String titulo;
    private String autor;
    private String editorial;
    @Temporal(TemporalType.DATE)
    private Date fechaPublicacion;
    @OneToMany(fetch = FetchType.EAGER, orphanRemoval = true)
    private Map<String,Read> lecturas;
    private String imagen;
    @Lob
    @Column(length = 2048)
    private String descripcion;
    @Version
    int version;
```

Ilustración 4.3

Ejemplo mapeo JPA clase Book

Algunas de las anotaciones que se pueden ver en el extracto del código son:

- @Entity especifica que la clase debe persistirse en BBDD.
- @Id especifica la clave primaria de la tabla que se generará.
- @SequenceGenerator y @GeneratedValue, crean una secuencia en la base de datos para asignar valores automáticamente a esta clave primaria.
- @Temporal especifica el mapeo de una fecha en BBDD y los valores a considerar, en este caso se especifica TemporalType.DATE, es decir, no se guarda información de hora.
- @OneToMany se usa para el mapeo de relaciones entre objetos del modelo. En este caso tenemos una relación uno a muchos implementada mediante un mapa. Se puede establecer el tipo de carga a realizar de los objetos relacionados mediante FetchType.EAGER, usado por defecto (se cargan siempre), o FetchType.LAZY (los objetos se recuperan bajo demanda, realizando una carga perezosa). Usaremos

cada uno de ellos dependiendo de que nos interese realizar un tratamiento sobre la colección siempre o sólo en algunos casos.

- @Lob y @Column especifican la forma en la se debe guardar un valor en BBDD. En este caso se fuerza a que el campo de la tabla sea de tipo large object byte y se especifica un tamaño máximo permitido de 2048.

Hay que realizar una configuración específica para cada tipo de BBDD que se utilice en el applicationContext.xml de Spring, como se puede ver en la siguiente imagen se proporcionan configuraciones para Derby y Mysql pudiendo cambiar de una BBDD con sólo aplicar la configuración:

```
<!-- Configuración para BBDD Derby-->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.apache.derby.jdbc.ClientDriver" />
    <property name="url" value="jdbc:derby://localhost:1527/clublectura" />
    <property name="username" value="root" />
    <property name="password" value="root" />
</bean>

<bean id="jpaAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="database" value="DERBY" />
    <property name="databasePlatform" value="org.hibernate.dialect.DerbyTenSevenDialect" />
    <!-- Sólo en fase de desarrollo -->
    <property name="showSql" value="true" />
    <property name="generateDdl" value="true" />
</bean>

<!-- Configuración para BBDD Mysql -->
<!--
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/QtnMiddleware?zeroDateTimeBehavior=convertToNull" />
    <property name="username" value="root" />
    <property name="password" value="root" />
</bean>
<bean id="jpaAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="database" value="MYSQL" />
    <property name="databasePlatform" value="org.hibernate.dialect.MySQL5Dialect" />
    <property name="showSql" value="true" />
    <property name="generateDdl" value="true" />
</bean>
-->
```

Ilustración 4.4

4.2.1.2.- Beans

Un Bean [33] es una clase Java gestionada por el contenedor de Beans de spring y que son utilizadas para realizar inyección de dependencias [34], es decir cada uno de los objetos definen las dependencias que necesitan para que le sean suministradas por Spring. En nuestro caso hemos usado una configuración de Spring que auto detecta los Beans a partir de un path mediante ciertas anotaciones

en nuestras clases Java usando <context:component-scan base-package="com.clublectura" />:

Marcamos las clases a gestionar con anotaciones @Service y @Repository y realizamos la inyección de dependencias en las clases que usen estos Beans mediante @Autowired como se muestra a continuación:

```
public class ReadController {

    @Autowired
    UserService userService;

    @Autowired
    ReadService readService;
```

Ilustración 4.5

4.2.1.3.- Controladores

Se encargan de definir la comunicación de la API REST que implementamos. La clase Java deberá marcarse como @Controller para ser manejada por el Dispatcher Servlet, adicionalmente se puede especificar la ruta de entrada al controlador con @RequestMapping(“path”).

```
@RequestMapping(value = "/{idAuthor}/books/{idBook}", method = RequestMethod.POST, produces = "application/json; charset=utf8")
@ResponseStatus(HttpStatus.CREATED)
@ResponseBody
public Author addBookToAuthor(
        @PathVariable Long idAuthor,
        @PathVariable Long idBook) throws AuthorNotFound, BookNotFound, IdAuthorBadRequest, IdBookBadRequest {

    if(idAuthor==null){
        throw new IdAuthorBadRequest();
    }

    if(idBook==null){
        throw new IdBookBadRequest();
    }

    return authorService.addBook(idAuthor,idBook);
}
```

Ilustración 4.6

En la imagen anterior se muestra un ejemplo de método de un controlador. Con @RequestMapping podemos especificar la ruta a la que hay que hacer la

petición para ejecutar el método, la operación, en este caso POST que es usada para añadir un recurso, y el formato de respuesta del método: “application/json”.

Se puede establecer el estado de respuesta en caso de éxito mediante @ResponseStatus(HttpStatus.xxx) e indicar que el contenido que se sirve no es una vista e irá en el cuerpo de la respuesta con @ResponseBody.

Adicionalmente se pueden obtener parámetros del path, para ser usados en el procesamiento de la petición mediante @PathVariable. También se puede indicar que se recibe un objeto en formato Json con @RequestBody e indicar un objeto donde se almacenará como se observa en la imagen de ejemplo del apartado siguiente.

4.2.1.4.- Validación

Se usará la dependencia javax.validation [35] para tareas de validación de objetos. En primer lugar, mediante el uso de anotaciones en la clase java se establecen las restricciones que pueden ser de cualquier tipo, por ejemplo @NotNull delante de cada atributo.

Se usará principalmente para validar los objetos recuperados a partir de los Json de entrada en los métodos de los controladores, usando la anotación @Valid, como se muestra en:

```
@RequestMapping(method = RequestMethod.POST, consumes = "application/json; charset=utf8",
                                             produces = "application/json; charset=utf8")
@ResponseStatus(HttpStatus.CREATED)
@ResponseBody
public Book createBook(
        @Valid @RequestBody Book book,
        BindingResult result) throws LibroIncorrecto, BookAlreadyExists {

    if (result.hasErrors()) {
        throw new LibroIncorrecto();
    }

    return bookService.createBook(book);
}
```

Ilustración 4.7

4.2.1.5.- Autenticación

Se ha optado por incluir un sistema de autenticación para poder acceder a los métodos restringidos de administrador de la API basado en Json Web Token [36], en concreto usando una implementación en java del mismo que se encontrar, junto a toda la documentación relevante y ejemplos de uso en https://github.com/jwtk/jjwt.

Usando este mecanismo, se genera un token encriptado usando una clave privada que puede contener la información que nosotros establezcamos, por ejemplo el rol del usuario, y se le envía al cliente. El cliente debe almacenar el token y reenviarlo junto a cada petición que realice en el campo de la cabecera *Authorization* con el formato: Bearer token. El token se recupera en el servidor, desencripta y se obtiene información del rol del emisor para comprobar si tiene permisos de administrador para acceder a operaciones restringidas.

4.2.1.6.- Conversión de excepciones a HTTPSTATUS.

Un aspecto fundamental a tratar al implementar una API REST, es la información que se le suministra al cliente en caso de que se produzcan errores al atender sus peticiones. Estos deben ser lo más descriptivos posibles y proveer toda la información necesaria para que el usuario conozca qué ha sucedido y cómo reenviar la petición para recibir una respuesta exitosa.

Por defecto, cualquier error o excepción lanzada se comunicará con un error 500 y una traza del error, lo cual resulta poco informativo para el cliente y potencialmente peligroso, ya que se expone mucha información de la estructura del proyecto como nombres de clases, métodos y trazas de ejecución.

La solución empleada en este proyecto se basa en la creación de excepciones propias para los diferentes tipos de error que podamos controlar y realizar un mapeo de estas excepciones a códigos http, añadiendo también un mensaje informativo personalizable.

Además, se consigue realizar este manejo de conversiones de forma unificada en una sola clase usando la anotación @ControllerAdvice [37] y creando una clase Global Exception Handler, de tal forma que todas las excepciones que se propaguen

desde los controladores serán manejadas por este. A continuación, se muestra un ejemplo:

```
@ResponseStatus(value = HttpStatus.CONFLICT, reason = "El recurso ya existe.")
@ExceptionHandler({AuthorAlreadyExists.class, BookAlreadyExists.class, ReadAlreadyExists.class,
    UserAlreadyExists.class, UserNameAlreadyExists.class})
public void handlerParametroExistente() {
}
```

Ilustración 4.8

En este caso se mapean todas las excepciones propias que representan que un recurso que intenta ser creado ya existe, por ejemplo, crear un libro cuyo isbn ya está registrado, a un HttpStatus.CONFLICT y se añade un mensaje informativo.

4.2.2.- Implementación en el cliente.

La implementación del lado del cliente se utiliza AngularJS basado en JavaScript. Se utilizará también NetBeans 8.1 como IDE de desarrollo y un proyecto HTML5 básico. Se usará la versión 1.5.3 de Angular y misma versión de sus dependencias básicas: angular-animate, angular-route, angular-aria y angular-messages. En fase de desarrollo se han utilizado las versiones normales para obtener mensajes de error lo más descriptivos posibles, aunque siempre es buena idea usar versiones minificadas (.min.js) en producción para realizar descargas más rápidas. A continuación, se comentan los aspectos más destacados:

4.2.2.1.- Enrutamiento mediante ui-router.

Se ha considerado oportuno prescindir del mecanismo de enrutamiento (donde se asocian vistas con controladores) estándar de Angular y sustituirlo por ui-router en su versión 0.2.18, que añade vistas anidadas y tratamiento de estados al mecanismo de enrutamiento básico ngRoute, cuya documentación y código está disponible en https://github.com/angular-ui/ui-router.

4.2.2.2.- Angular Material

En el apartado de la interfaz de usuario se ha decidido usar Angular Material [38] en su versión 1.1.0-cr4, un framework que proporciona una serie de componentes basados en Material Design (cuya filosofía se describe en https://material.google.com/#)

de Google, para conseguir un aspecto visual moderno y lo más atractivo posible.

4.2.2.3.- Servidor de almacenamiento de imágenes en la nube.

Como solución para el tratamiento de imágenes de todo el proyecto (imágenes de perfil de usuario, portadas de libros, fotos de autores), se ha optado por el uso de Cloudinary, un servicio privado de almacenamiento en la nube con múltiples utilidades, como conversión de formatos de imagen, manipulación de dimensiones, optimización del peso de la imagen dependiendo de la pantalla en la que se visualiza la web, recorte de imágenes, centrar la imagen en la cara, y además, sirve las imágenes vía CDN. Más información en http://cloudinary.com/.

En concreto se usa un repositorio específico de cloudinary para angular que se puede encontrar en https://github.com/cloudinary/cloudinary_angular que permite realizar las descargas de imágenes especificando las transformaciones que necesitemos, mediante directivas de Angular.

4.2.2.4.- Angular-rateit

Es simplemente, una directiva que permite realizar y mostrar puntuaciones mediante iconos, generalmente estrellas, totalmente adaptable a las necesidades del proyecto, que mejorará el aspecto visual de la web y hará más intuitivo el uso por parte de los usuarios. El proyecto y su documentación se pueden encontrar en https://github.com/akempes/angular-rateit.

5.- CONCLUSIONES

En este apartado, realizaremos una serie de justificaciones y reflexiones, sobre el grado de cumplimiento de los objetivos iniciales. Impresiones sobre el enfoque, tecnologías empleadas y ventajas e inconvenientes presentados. También se dará una pequeña reflexión personal sobre lo que ha aportado el proyecto como punto final a los conocimientos adquiridos durante el grado.

5.1.- Cumplimiento de los objetivos iniciales.

Partíamos con los objetivos de realizar un estudio de necesidades y soluciones para la creación y gestión de bibliotecas personales y grupos de lectura on-line, lo cual queda desarrollado en el apartado de análisis, donde estudiamos requisitos funcionales y no funcionales, se dio una visión general del sistema a desarrollar y se presentaron frameworks y herramientas para conseguir un sistema como el requerido.

Se planteaba también como objetivo, el diseño de un sistema especialmente orientado a la utilización de arquitectura y metodología de desarrollo web. Este diseño se desarrolla con detalle en la sección 3 de diseño de esta memoria y se materializa con la posterior implementación

Por último se planteaba el objetivo de implementar un prototipo de aplicación web usando tecnologías REST basadas en Spring Framework en el servidor y una interfaz adaptativa en el cliente usando HTML5 y AngularJS. Este apartado se explica detalladamente en la sección 4 de implementación de la memoria, donde se detalla la forma en la que se ha trabajado con cada framework y como se consigue exponer un servicio REST mediante SpringMVC y como este es utilizado por un cliente web para el que se ha usado HTML5 y AngularJS.

Por lo tanto, se consideran cumplidos los objetivos que dieron pie a la realización de este proyecto.

5.2.- Algunas reflexiones sobre el proyecto:

Considero acertadas las tecnologías usadas para la realización de este proyecto, ya que permiten cumplir con los requisitos sin ningún tipo de problema, además, se ha podido comprobar como Spring Framework permite (una vez que se sabe utilizar) un desarrollo de back-end rápido, donde puedes dedicar tu tiempo a la lógica de negocio y no a la interconexión de diferentes sistemas, bases de datos, mapeo de objetos java a formato json, o con gestión de transacciones y consultas SQL.

Además, la impresión de la que para mí ha sido el primer contacto con AngularJS ha sido muy positiva también. Aunque la curva de aprendizaje al principio fue un poco lenta, una vez que se dominan los conceptos básicos del framework el desarrollo es rápido, y algunas de sus características como el data binding bidireccional hacen aún más fácil y rápido el desarrollo que en otros frameworks utilizados para el mismo fin.

La impresión que tengo tras realizar este TFG es que se ha conseguido por primera vez, utilizar en un mismo sitio, y con una aplicación práctica real, una gran parte de contenidos estudiados en diferentes asignaturas, ver cómo se relacionan entre ellas y por qué son realmente importantes en un proyecto de diseño y desarrollo de software real.

Además considero muy positiva la elección inicial de frameworks y tecnologías a utilizar, ya que son ampliamente utilizadas en el mundo empresarial, y me hacen sentir más capacitado a nivel técnico para desarrollar trabajos en esta área.

5.1.- Mejoras y trabajos futuros

Evidentemente, el trabajo realizado dista mucho de ser perfecto. Existen varias líneas de mejora, empezando simplemente por un mejor manejo con los propios lenguajes de programación que produzcan un código lo más limpio y legible posible para otros desarrolladores.

Una de las características del proyecto, en la parte del servidor que quedan pendientes es la implementación de sistemas de cacheo de información, para la cual ofrece un manejo muy potente y que puede ser explotado para reducir el número de consultas que se realizan a la BBDD, pero por falta de tiempo y considerar prioritarios otros aspectos que afectan directamente a la funcionalidad, no ha podido ser implementada.

Otra característica que afecta tanto a servidor como a cliente web es la implementación de paginación en las consultas a la BBDD. Si bien, esta

característica puede ser implementada de una forma fácil en el cliente web, no ofrece toda su potencia sin consultas a la BBDD que requieran sólo el contenido a mostrar por cada página que se muestra al usuario. El objetivo inicial era implementar ambas al mismo tiempo, y por la propia gestión del tiempo de trabajo y dificultades encontradas en otros aspectos, no ha sido posible.

Otro punto sobre el que me gustaría hacer mención y que es muy mejorable, es el aspecto visual y layout de la aplicación web. Evidentemente el tiempo es finito y se ha priorizado siempre en funcionalidad antes que en estética.

6. BIBLIOGRAFÍA

[1] *Scrum Desarrollo Software*. Wikipedia. [Online] Disponible en https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software) . Acceso: Ago, 2016

[2] Historias de Usuario. Wikipedia. [Online] disponible en https://es.wikipedia.org/wiki/Historias_de_usuario . Acceso Ago, 2016

[3] W. Wheeler and J.White ,*Spring in Practice*. Safari, 2012. [Online]. Disponible en Recursos Electrónicos de la Universidad de Jaén

[4] Spring.io. [Online] Disponible en https://projects.spring.io/spring-framework/. Acceso Ago, 2016

[5] Overview Dependency Injection. Spring.io. [Online] disponible en http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html#overview-dependency-injection. Acceso Ago, 2016

[6] Overview Modules. Spring.io. [Online] disponible en http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html#overview-modules . Acceso Ago, 2016

[7] Overview Data Access. Spring.io. [Online] disponible en http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html#overview-data-access . Acceso Ago, 2016

[8] Overview Web. Spring.io. [Online] disponible en http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html#overview-web . Acceso Ago, 2016

[9] Spring Framework. Wikipedia. [Online] disponible en https://es.wikipedia.org/wiki/ Spring_Framework . Acceso Ago, 2016

[10] License 2.0. Apache.org [Online] disponible en http://www.apache.org/licenses/LICENSE-2.0. Acceso Ago, 2016

[11] Brad Green, Shyam Seshadri. O'Reilly Media, Inc. 2013. [Online]. Disponible en Recursos Electrónicos de la Universidad de Jaén

[12] Angularjs. [Online] disponible en https://www.angularjs.org/

[13] Introduction. Angularjs.org. [Online] disponible en https://docs.angularjs.org/guide/introduction. Acceso Ago, 2016

[14] Spi Manifesto. Sourceforge.net [Online] disponible en http://itsnat.sourceforge.net/php/spim/spi_manifesto_es.php. Acceso Ago, 2016

[15] Data Binding. Angularjs.org. [Online] disponible en https://docs.angularjs.org/guide/databinding. Acceso Ago, 2016

[16] Data Binding. Angularjs.org. [Online] disponible en https://docs.angularjs.org/api. Acceso Ago, 2016

[17] Historias de usuario. Wikipedia. [Online] disponible en https://es.wikipedia.org/wiki/ Historias_de_usuario . Acceso Ago, 2016

[18] Mike Cohn, "User Stories Applied", 2004, Addison Wesley, ISBN 0-321-20568-5

[19] Mike Cohn, Agile Estimating and Planning, 2006, Prentice Hall http://0-proquestcombo.safaribooksonline.com.avalos.ujaen.es/9780137126347

[20] Mike Cohn, Agile Estimating and Planning, 2006, Prentice Hall http://0-proquestcombo.safaribooksonline.com.avalos.ujaen.es/9780137126347/ch08

[21] Mike Cohn, Agile Estimating and Planning, 2006, Prentice Hall http://0-proquestcombo.safaribooksonline.com.avalos.ujaen.es/9780137126347/ch08#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODAxMzcxMjYzNDclMkZjaDA2JnF1ZXJ5PQ==

[22] MoSCoW Method. Wipikedia [Online] https://en.wikipedia.org/wiki/MoSCoW_method

[23] Modelo de dominio. Wipikedia [Online] https://es.wikipedia.org/wiki/Modelo_de_dominio

[24] Association Class. Etutorials.org [Online] http://etutorials.org/Programming/UML/Chapter+6.+Class+Diagrams+Advanced+Concepts/Association+Class/

[25] Modelo entidad relación. Wipikedia [Online] https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n

[26] *Modelo Vista Controlador*. Wikipedia. [Online] Disponible en: http://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador. Acceso: Jul, 2014

[27] Digital Java EE 7 Web Application Development. Peter Pilgrim. Packt Publishing 2015 [Online] disponible en http://0-proquestcombo.safaribooksonline.com.avalos.ujaen.es/9781782176640

[28] Java EE Overview. Oracle.com [Online] disponible en http://www.oracle.com/technetwork/java/javaee/overview/index.html

[29] Web Profile. Java.net [Online] disponible en https://java.net/downloads/javaee-spec/WebProfile.pdf

[30] Pro JPA 2, Second Edition. Mike Keith, Merrick Schincariol. Apress 2013. [Online] disponible en http://0-proquestcombo.safaribooksonline.com.avalos.ujaen.es/9781430249269

[31] Mapeo objeto relacional. Wikipedia. [Online] disponible en https://es.wikipedia.org/wiki/Mapeo_objeto-relacional

[32] Hibernate. [Online] disponible en http://hibernate.org/orm/

[33] Beans definition. Spring.io. [Online] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-definition

[34] Beans. Spring.io. [Online] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans

[35] Package description. Oracle.com. [Online] https://docs.oracle.com/javaee/7/api/javax/validation/package-summary.html#package.description

[36] Json Web Token. [Online] https://jwt.io/introduction/

[37] Controller advice clases. Spring.io [Online] https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc#using-controlleradvice-classes

[38] Angular Material [Online] https://material.angularjs.org/latest/

7. APÉNDICES

Apéndice I. Manual de Instalación del sistema

Dentro de la carpeta de código fuente suministrada hay dos subcarpetas con los proyectos de servidor y cliente, en cada una de ellas se puede encontrar el proyecto con el código fuente comprimido en formato .zip que puede ser importado directamente a NetBeans o descomprimido en la carpeta de trabajo del framework.

Como se comenta en el apartado de implementación se ha realizado la configuración necesaria para utilizar tanto BBDD Derby o Mysql. Por comodidad para el lector, se ha dejado por defecto la configuración Derby, ya que se puede crear y utilizar desde NetBeans en pocos segundos de la siguiente forma: primero hay que abrir la pestaña de Prestaciones, depegar Bases de Datos y sobre Java DB, click en botón derecho y crear base de datos. Los datos a introducir son: Nombre de

BBDD “clublectura”, nombre de usuario “root” y contraseña “root”. Pulsamos aceptar y ya está creada.

El siguiente paso es arrancarla, lo cual se puede hacer con doble click sobre la BBDD creada. Ahora ya podemos compilar y ejecutar el servidor. En el proceso de ejecución se nos preguntará que servidor usar y se puede seleccionar Tomcat o Glassfish, se han realizado pruebas con ambos y no se deberían encontrar ningún tipo de problema al usar uno o otro.

Ya tenemos el servidor arrancado, pudiendo lanzar la aplicación web sobre el navegador preferido (se ha comprobado compatibilidad total con Chrome y Firefox).

Claro está, el estado inicial de la BBDD es vacío. Para que el lector pueda realizar simulaciones con datos de prueba se provee de un método de la API que introduce datos de prueba. Simplemente en el navegador se introduce la siguiente dirección: http://localhost:8080/ClubLectura/init y se recibirá un mensaje de confirmación al terminar el proceso.

Si volvemos a lanzar la aplicación web podemos acceder con cuenta de administrador (user: “Admin”, password: “Admin”) o con cuenta de usuario (nombre “Aura Garrido”, password “Aura”), evidentemente también se puede registrar una nueva cuenta.

Apéndice II. Descripción de contenidos suministrados

La entrega de este TFG se realiza en soporte de almacenamiento físico CD con la siguiente estructura de carpetas:

Memoria: que contiene la memoria en formato PDF.

Código Fuente: que contiene dos subcarpetas, Servidor y Cliente Web. Dentro de cada una de ellas se puede encontrar el proyecto con el código fuente comprimido en formato .zip que puede ser importado directamente a NetBeans o descomprimido en la carpeta de trabajo del framework.

Diagramas: contiene todos los diagramas que se exponen en esta memoria en un archivo de Visual Paradigm.