Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

RED SOCIAL PARA ESCALADORES CON REESTIMACIÓN COLABORATIVA DEL GRADO DE DIFICULTAD DE LAS VÍAS DE ESCALADA

Alumno: Daniel Muñoz Gallardo

Tutores: Prof. D. Antonio Jesús Rueda Ruiz

Prof. D. Carlos Porcel Gallego

Dpto: Departamento de Informática

Junio, 2017

"No es verdad que la gente deja de perseguir sus sueños porque envejece, envejecen porque dejan de perseguir sus sueños."

Gabriel García Márquez

"Si quieres triunfar, no te quedes mirando la escalera. Empieza a subir, escalón por escalón, hasta que llegues arriba."

Anónimo

"Cuando alguien te diga que no eres capaz de conseguir una meta, ya tienes un motivo para empezar."

Daniel Muñoz Gallardo

Índice de contenidos

AGRADECIMIENTOS

En primer lugar estaré eternamente agradecido a la mujer de mi vida, mi madre. Su esfuerzo y sus ganas de vivir vencieron al cáncer y pudo seguir apoyándome con mucho sacrificio a cumplir parte de mi sueño de llegar a ser ingeniero.

Al resto de miembros de mi familia que siempre me han apoyado, especialmente mi hermana y mi cuñado Miguel, mis tíos Lorenzo y Joaquina, mi primo Lorenzo y Tomás, por ayudarme cuando más lo necesitaba durante estos años.

A mi gran amigo y compañero de fatigas, Don Julián Llopis Ruiz, porque hemos formado un gran equipo durante estos años de trabajo que ha ganado muchas batallas… ¡¡y las que queden!!

A esos compañer@s, amig@s y profesor@s que han estado conmigo durante estos duros años a los cuales no olvidaré nunca. Han sido sin duda alguna los mejores años de mi vida.

A mis tutores, Antonio Jesús Rueda Ruiz y Carlos Porcel Gallego, por confiar en mí y darme la oportunidad de realizar un trabajo de estas características.

A José Ramón Balsas Almagro, por su gran ayuda y ánimo de manera totalmente desinteresada en momentos en los que veía las cosas demasiado oscuras.

Quería acordarme también de aquellas personas que dudaron de mí en cualquier momento, pues gracias a ellos cogía fuerzas para seguir adelante.

Espero que esto sólo sea el principio de algo muy grande. Me siento con ganas de comerme el mundo y todo gracias a vosotros. Nunca os olvidaré.

GRACIAS A TODOS

Daniel Muñoz Gallardo

PREÁMBULO

Estructurado en nueve partes, este documento detalla lo que ha sido el proceso de desarrollo de mi Trabajo Fin de Grado. Para facilitar su lectura, se detallan a continuación cada una de ellas. En primer lugar, una pequeña introducción con el problema que se plantea y vamos a resolver.

En segundo lugar, nos metemos de lleno en el estudio del problema en cuestión. Para ello es necesario previamente conocer los aspectos básicos de la escalada, conocer su variante deportiva y los sistemas de gradación de vías empleados. También incluye un pequeño estudio sobre webs y apps existentes actualmente. Seguidamente un tercer apartado donde se incluye una fase de análisis inicial, y se estudiarán los requisitos de la aplicación y las tecnologías empleadas.

Posteriormente, en el cuarto apartado se abordará la estimación inicial del proyecto. Se decidirá la metodología de trabajo a tener en cuenta, las tareas a realizar, así como la estimación de las mismas y se elaborará un presupuesto acorde a ellas. En el quinto apartado se desarrolla el modelo de negocio de la aplicación, correspondiente a la primera parte del *back-end*.

Al finalizar este bloque, en el sexto apartado, se hará una reestimación del proyecto y un nuevo presupuesto en base a un cambio de requisitos determinado en una de las reuniones con el cliente. Se elaborará una nueva estimación de tareas y un presupuesto en base a las mismas.

El séptimo apartado abarca el desarrollo del API RESTFUL que se utilizará para la exportación del servidor por medio de servicios web, indicando en qué consisten cada uno de ellos y la forma de invocarlos.

El cliente web (*front-end*), desarrollado utilizando el servidor anteriormente mencionado, vendrá detallado en el octavo apartado. Por último, un noveno apartado donde se incluyen los enlaces al código de la aplicación y un vídeo de demostración sobre la misma.

1. Introducción

1.1. Motivación

Como colofón a estos duros años de trabajo, me dispongo a la elaboración de mi Trabajo Fin de Grado y la verdad, desde primera hora tenía claro los aspectos técnicos básicos que quería tratar. Durante los dos últimos años he tenido mucha inquietud sobre aspectos relacionados con el mundo web y la programación en Java. Tanto fue así que no dudé en ningún momento en meterme de lleno en un mundo que quizás sea de los más demandados por el mercado laboral actual.

Asignaturas como Desarrollo de Aplicaciones Web o Desarrollo de Aplicaciones Empresariales me hicieron adquirir una serie de conocimientos que me llamaba mucho la atención seguir poniéndolos en práctica, perfeccionarlos y extenderlos a conocimientos nuevos que he desarrollado durante estos meses, como por ejemplo el mundo de JavaScript.

Así pues, el Prof. D. Antonio Jesús Rueda Ruiz me realizó una propuesta de TFG que fue difícil de rechazar ante el gran reto que tenía por delante: elaborar una aplicación real desde cero usando las tecnologías mencionadas anteriormente.

1.2. Ámbito del trabajo. Problema a resolver

Nos situamos en el marco de la escalada deportiva. Tradicionalmente las vías son catalogadas por el escalador que las equipa, en una escala que va desde grado 1 hasta grado 9c+ (hasta el momento). Sin embargo, esta catalogación es discutible y subjetiva. Por ejemplo, una vía en principio de 5a+ puede tener un grado real más cercano al 6a o 6a+, lo que puede ser un problema para muchos escaladores que pueden verse obligados a abandonar la vía a medio camino al no ser capaces de terminarla.

La idea es diseñar una sencilla red social que sirva de diario de escalada, donde los escaladores puedan anotar las vías de escalada completadas y su catalogación subjetiva del grado de la vía, que puede coincidir con el grado oficial o ser muy distinto. El sistema será capaz de ofrecer una gradación más cercana a la real usando la opinión de cada escalador y su trayectoria.

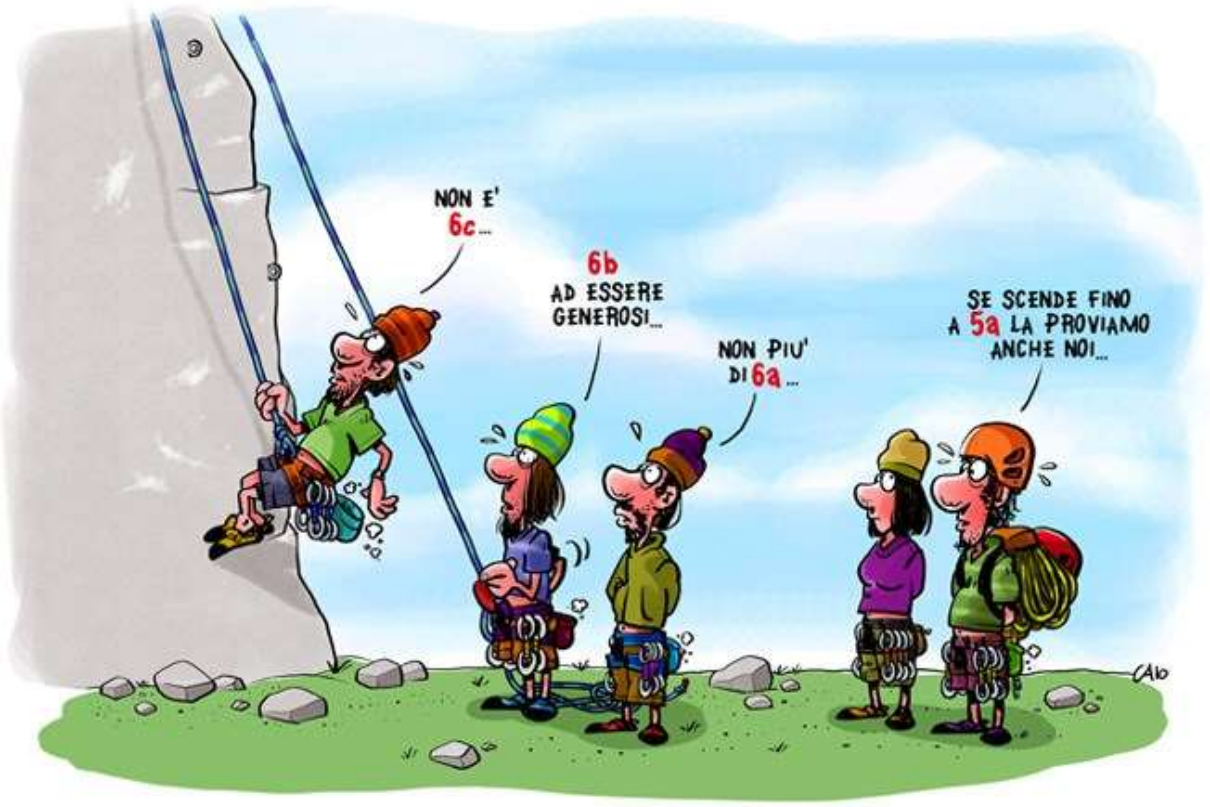

Ilustración 1: Viñeta ilustrativa del problema

La ilustración anterior resume el problema. El que está escalando y se ha caído quedándose colgado piensa que no es un 6c, sino que es más. La persona que asegura ese nivel sin embargo piensa que es un 6b siendo generoso. El amigo que los acompaña piensa que no es más de un 6a y los que llegan para escalar piensan que es un 5a y que van a probar ellos también. En definitiva, un lío que vamos a intentar solucionar. En este interesante artículo [1] se describe perfectamente la situación a la que estamos haciendo referencia.

2. Aspectos básicos de la escalada deportiva

La escalada era un deporte hasta ahora desconocido para muchas personas. Cabe destacar que existen varias modalidades de la misma para su práctica. En este caso, nosotros nos centraremos en una variante llamada “escalada deportiva”, una de las más practicadas y demandadas por los escaladores actuales. A continuación se detalla brevemente en qué consiste y la forma de graduar las vías en base a su dificultad.

2.1. Introducción a la escalada deportiva

La escalada deportiva se trata de un estilo de escalada en el que, como sistema de seguridad, utiliza anclajes previamente fijados a la pared mediante distintos sistemas mecánicos o químicos colocados estratégicamente a lo largo de la vía, lo que permite ampliar las posibilidades de escalada a las placas de roca compacta carentes de aristas o fisuras.

Se caracteriza por reducir notablemente el riesgo del escalador, a cambio de aumentar el nivel de dificultad (grado de la vía). Generalmente, esta modalidad de escalada busca zonas relativamente accesibles y con paredes no necesariamente muy altas, en las que se equipan vías de diferentes grados de dificultad. Por lo general, antes de equiparse, estas vías se “limpian” de maleza y de piedras sueltas o susceptibles de romperse, para ganar en la seguridad del escalador. También existen los llamados rocódromos donde se puede practicar. Esta modalidad de escalada suele buscar la dificultad por sí misma, y la belleza de movimientos. [2]

Ilustración 2: Escalada deportiva en rocódromo

2.2. Escuelas, sectores y vías

Los lugares habilitados para la práctica de escalada deportiva se organizan a tres niveles para facilitar su localización y acceso.

En primer lugar, tenemos en cuenta el concepto de escuela. Se denomina así a una región física donde practicar escalada, es decir, a un lugar que ha sido habilitado para la práctica de la misma de manera segura. Normalmente se asocian a una provincia. [3]

En segundo lugar el concepto de sector. No es más que una división de una zona de la escuela con unas características concretas, como pueden ser la orientación o el tipo de material a utilizar en sus vías.

Por último, las vías. Se trata de cada uno de los “caminos” que pueden seguirse para conseguir el objetivo de escalar en un sector. Cada vía tiene un nivel oficial que es determinado por aquella persona que equipa la vía para permitir su práctica.

2.3. Grados de dificultad de las vías

Una de las formas mundialmente conocidas para saber el grado de dificultad es la escala francesa. Se utilizará esta escala de graduación para determinar el grado de dificultad de las vías. El sistema francés de graduación considera la dificultad total de la escalada teniendo en cuenta la dificultad de los movimientos y la longitud de la vía. Los grados son numerados empezando por 1 (muy fácil) y queda abierta por arriba. Cada grado puede ser subdividido al añadir una letra (a, b ó c). Además se puede añadir un signo + o - para aclarar mejor la dificultad [4]**.** A continuación se presenta una breve comparativa de los distintos sistemas de graduación:

Tabla de conversión de grados en escalada deportiva

Americano	Inglés		Frances	UIAA	Saxon	Australia, NZ & South Africa (Ewbank)	Filandés	Noruego	Brasileño
	Tech/Adj								
2nd class			1	I	I				Isup
3rd class			2	II	II	11			II
4th class			3	III	III	12		3	IIsup
5.0-5.4	4a	VD	4a	IV	IV	12		4	III
5.5		S	4b	IV+	V	13	5–	5–	IIIsup
5.6	4b	HS	4c	V	VI	14	5	5	IV
5.7	4c	VS	5a	V+		15			
5.8		HVS	5b	VI-	VIIa	16	5+	5+	IVsup
5.9	5a		5c	VI	VIIb	17		6–	V
5.10a		E1	6a	VI+	VIIc	18	6–	6–/6	VI
5.10b	5b		6a+	VII-		19		6	VI/VI+
5.10c		E2	6b	VII	VIIIa	20	6	6+	VIsup/VI+
5.10d	5c		6b+	VII+	VIIIb	21		7-	VIsup
5.11a		E3	6c		VIIIc	22	6+	7	7a
5.11b			6c/c+	VIII-		23			7b
5.11c	6a	E4	6c+		IXa	24	7–	7+	7c
5.11d			7a	VIII	IXb				7c
5.12a		E5	7a+	VIII+	IXc	25	7+	7+/8–	8a
5.12b			7b			26	8–	8-	8b
5.12c	6b	E6	7b+	IX–	Xa	27	8	8	8c
5.12d			7c	IX	Xb	28	8+	8/8+	9a
5.13a		E7	7c+	IX+	Xc	29	9–	8+	9b
5.13b	6c		8a				9	9-	9c
5.13c		E8	8a+	X–		30	9+	9–/9	10a
5.13d		E9	8b	X		31	10–	9	10b
5.14a	7a	E10	8b+	X+		32	10	9/9+	10c
5.14b			8c			33	10+	9+	11a
5.14c	7b	E11	8c+	XI–		34	11–		11b
5.14d			9a	XI		35	11		11c
5.15a			9a+	XI+					12a
5.15b			9b						12b

Tabla 1: Comparativa de los distintos sistemas de graduación

Dado el abanico de posibilidades que ofrece la escala francesa y la subjetividad de la misma, utilizaremos los siguientes niveles para nuestro ámbito, ordenados de menor a mayor dificultad y así estandarizar de forma más sencilla: 1, 2, 3, 4, 5, 5+, 6a, 6a+, 6b, 6b+, 6c, 6c+, 7a, 7a+, 7b, 7b+, 7c, 7c+, 8a, 8a+, 8b, 8b+, 8c, 8c+, 9a, 9a+, 9b, 9b+, 9c, 9c+.

Podríamos clasificarlos con una orientación sobre su dificultad en la siguiente tabla:

Nivel	Descripción de dificultad
1	Andar
2	Andar o trepar usando las manos puntualmente
3	Trepada fácil con uso obligatorio de manos
4	Escalada con cierto grado
5	Escalada con cierto grado de dificultad incluyendo pasos complicados no evidentes
6	Escalada difícil que requiere entrenamiento habitual para su superación, incluyendo desplomes y pasos atléticos
7	Escalada técnica de mucha dificultad que sólo puede realizarse con entrenamiento habitual y una buena forma física
8	Escalada técnica de gran dificultad a nivel de élite
9	Escalada de altísima dificultad a nivel de élite mundial

Tabla 2: Orientación de dificultad de los niveles

2.4. Apps y webs existentes actualmente

En la actualidad no existen Apps o webs que se dediquen exclusivamente a este problema. Podemos encontrar numerosas guías de escalada en webs que describen al detalle las escuelas, sectores y vías. También podemos encontrar foros donde se tratan algunos temas, como por ejemplo, los niveles de dificultad, el estado de las escuelas, etc., pero en ningún caso encontramos algo que se acerque al producto que vamos a intentar desarrollar, con lo cual es un aliciente de motivación para el desarrollo de un producto software innovador.

3. Fase de análisis inicial

3.1. Supuesto inicial. Primera reunión de captación de objetivos

Se trata de una primera reunión a primeros del mes de septiembre de 2016 con el que va a ser, el cliente final de la aplicación. En este caso serían mis dos tutores, los profesores D. Antonio Jesús Rueda Ruiz y D. Carlos Gustavo Porcel Gallego, dentro de la cual y tras la misma hago recopilación de una serie de anotaciones en base a los temas que hemos tratado, y que hacen ponerme en una primera situación de lo que va a ser el producto que se va a desarrollar. Los detalles son los que se mencionan a continuación.

El usuario accede a la aplicación por primera vez. Lo primero que debe hacer para poder hacer uso de la misma es registrarse. En el registro se debe facilitar un nombre usuario (a partir de ahora, también *username*), que el sistema comprobará previamente si éste está disponible. También se deberá facilitar una dirección de correo electrónico que irá asociada a la creación de esa cuenta, así como una contraseña de acceso a la misma (a partir de ahora, también *password*). Tras la confirmación del registro la cuenta quedará activa y el usuario podrá acceder a aplicación utilizando su *username* y su *password*.

Tras iniciar sesión la aplicación mostrará su perfil personal, en el que estarán sus datos personales, el máximo nivel que ha llegado a completar como escalador, así como una pequeña imagen o fotografía a gusto del usuario.

En la página personal del perfil también podrá visualizar si tiene alguna petición de amistad pendiente de gestionar. En todo momento el usuario puede gestionar dichas peticiones al acceder a ellas (aceptarlas o rechazarlas).

Aparecerán también las vías que ha escalado hasta la fecha, con un enlace para poder acceder los detalles de las mismas y las valoraciones, así como su lista de amigos, con un enlace a sus perfiles correspondientes.

Se incluirá una sección en el que el usuario pueda buscar tanto usuarios como escuelas de escalada. Para buscar escuelas de escalada el usuario lo hará filtrando por provincias y para buscar usuarios lo hará introduciendo su *username.*

En cualquier momento el usuario podrá añadir una escuela de escalada que no esté, hasta ese momento registrada en el sistema. Análogamente podrá hacer lo mismo con los sectores y las vías.

En lo que respecta a las reglas de reestimación se tendrán en cuenta para ello:

- Nivel oficial de dificultad de la vía.
- Nivel medio de las 10 últimas vías completadas del usuario que valora.
- Fiabilidad en base a su número de amigos.
- Fiabilidad en base a las vías que haya valorado.

La idea es hacer una aplicación web con una interfaz sencilla y agradable, que el usuario pueda usarla de la forma más intuitiva posible y que recoja las características anteriormente mencionadas de forma que acerque al usuario de una manera más sencilla a este maravilloso deporte.

3.2. Casos de uso. Requisitos funcionales de la aplicación

Tras la primera reunión de captación de objetivos, hago análisis de las anotaciones obtenidas y pongo en marcha los casos de uso en base a las mismas. En una breve segunda reunión con el cliente, una semana después, recibo el visto bueno tras detallarle el análisis de funcionalidades acordado, que son las que se detallan en los siguientes apartados.

3.2.1. Diagrama de casos de uso

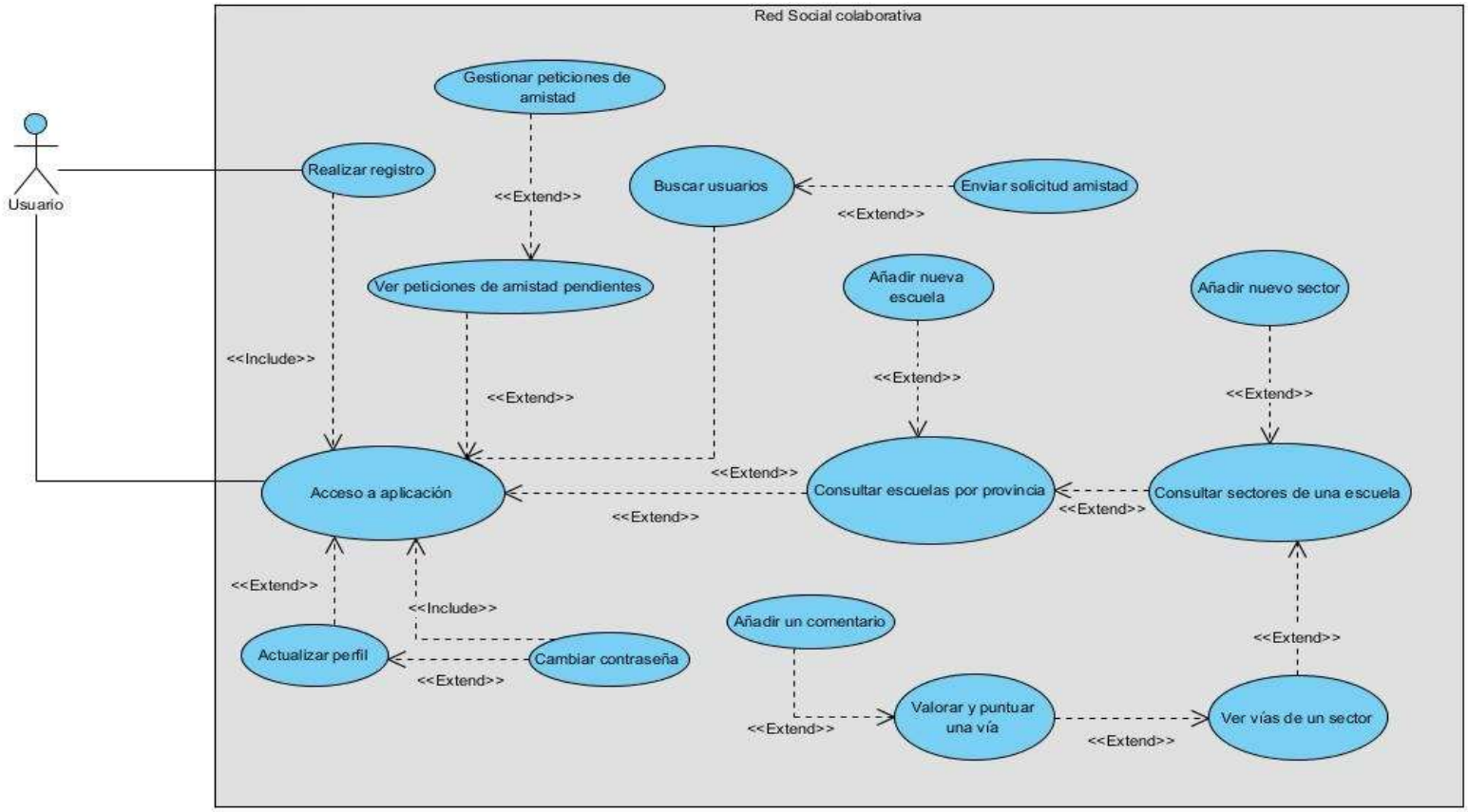

Ilustración 4: Diagrama de casos de uso

3.2.2. Caso de uso: Realizar registro

El registro del usuario en el sistema deberá ser obligatorio. Para ello, se proporcionará un nombre de usuario (*username*), una contraseña y un correo electrónico que irá asociado a su cuenta.

3.2.3. Caso de uso: Acceso a la aplicación

Para acceder a la aplicación el usuario deberá introducir el nombre de usuario (*username*) y la contraseña. El sistema comprobará que es correcto y le redirigirá a su perfil personal.

3.2.4. Caso de uso: Editar perfil de usuario

El usuario podrá editar su perfil personal en todo momento, actualizando la información que considere oportuna. Podrá modificar campos como su nombre y apellidos, y podrá incluir una pequeña foto de perfil.

3.2.5. Caso de uso: Modificar contraseña de acceso

El usuario podrá editar su contraseña en todo momento accediendo a la opción habilitada para ello en los datos de perfil.

3.2.6. Caso de uso: Ver peticiones de amistad pendientes

El usuario podrá ver a través de su página personal, si tiene alguna petición de amistad de otro usuario pendiente de gestionar.

3.2.7. Caso de uso: Gestionar peticiones de amistad pendientes

El usuario verá quien le remite cada una de las peticiones y en cualquier caso, podrá confirmarlas o rechazarlas.

3.2.8. Caso de uso: Consultar las escuelas de una provincia

El usuario podrá consultar las escuelas de una determinada provincia, que previamente habrá seleccionado.

3.2.9. Caso de uso: Publicar una nueva escuela

El usuario podrá añadir una nueva escuela en una determinada provincia, en caso de que ésta aún no haya sido añadida. Deberá introducir los datos de forma correcta, y en cualquier caso el administrador de la aplicación podrá invalidar dichos datos o corregirlos.

3.2.10. Caso de uso: Consultar los sectores de una escuela

El usuario podrá consultar los sectores de una escuela determinada. Para ello, debe acceder previamente a una escuela. Los datos de la misma también deberán estar disponibles.

3.2.11. Caso de uso: Publicar un nuevo sector

El usuario podrá añadir un nuevo sector en una determinada escuela, en caso de que éste aún no haya sido añadido. Deberá introducir los datos de forma correcta, y en cualquier caso el administrador de la aplicación podrá invalidar dichos datos o corregirlos.

3.2.12. Caso de uso: Consultar las vías de un sector

El usuario podrá consultar las vías de un determinado sector al que previamente habrá accedido. Los datos del sector al que pertenecen también serán visibles. Deberá introducir los datos de forma correcta, y en cualquier caso el administrador de la aplicación podrá invalidar dichos datos o corregirlos.

3.2.13. Caso de uso: Publicar una nueva vía

El usuario podrá añadir una nueva vía en un sector determinado en caso de que ésta no haya sido aún añadida. Deberá introducir los datos de forma correcta, y en cualquier caso el administrador de la aplicación podrá invalidar dichos datos o corregirlos.

3.2.14. Caso de uso: Valorar y puntuar una vía

El usuario podrá valorar (dar un grado de dificultad acorde a su opinión) y puntuar (de 1 a 3 estrellas según la belleza de la vía) una vía. Adicionalmente podrá añadir un comentario si así lo desea a su valoración. Las valoraciones y los comentarios estarán visibles al resto de usuarios. Los datos de la vía en cuestión deben aparecer previamente. Al valorar una vía, ésta se añade a la lista de vías realizadas por el usuario.

3.2.15. Caso de uso: Buscar usuarios

El usuario podrá buscar perfiles de usuario en base al nombre de usuario (*username*) y acceder al mismo. Podrá acceder a su perfil completo.

3.2.16. Caso de uso: Enviar solicitudes de amistad a otro usuario

El usuario podrá enviar una solicitud de amistad a cualquier perfil que no se encuentre en su lista de amigos.

3.3. Determinación de los requisitos no funcionales o requisitos de calidad

Volvemos a analizar los detalles de la reunión de captación de objetivos. Es hora de establecer qué requisitos de los que se nos solicita son requisitos no funcionales o requisitos de calidad, así como aquellos que como desarrolladores

podemos aportar de forma que garanticemos la máxima calidad de nuestro producto. Los requisitos no funcionales son los que se detallan a continuación:

- Tras solicitar el registro en la aplicación, el usuario dispondrá de 30 minutos como máximo para validar el mismo.
- La página de perfil personal contará con sus datos personales, el máximo nivel completado hasta la fecha, así como una breve foto que el usuario pondrá a su gusto. También se podrá ver un listado con las últimas vías escaladas, así como un resumen de su lista de amigos.
- Una interfaz sencilla, amigable y usable que simplifique al máximo el uso de la aplicación.

Los requisitos establecidos en este apartado pueden sufrir modificaciones, que se irán detallando a lo largo de las distintas reuniones en las iteraciones durante el desarrollo del producto.

3.4. Modelo de dominio de la aplicación

Se trata de una representación conceptual en base al análisis de requisitos. Evidentemente es una primera aproximación al modelo de datos, que podrá sufrir cambios en etapas posteriores.

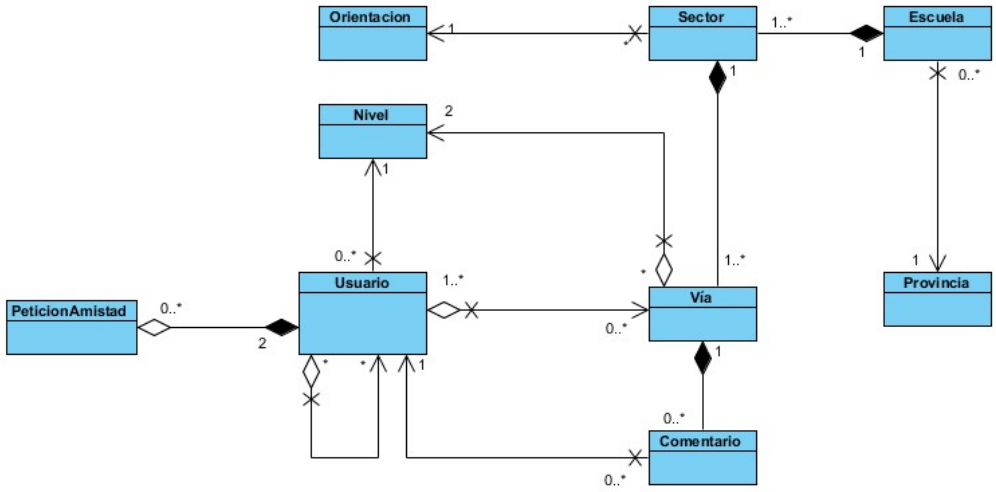

Ilustración 5: Modelo de dominio

3.5. Análisis técnico. Elección de las tecnologías para el desarrollo de la aplicación

Dados los requisitos de la aplicación, es la hora de analizar el aspecto tecnológico a tener en cuenta para realizar nuestro desarrollo. En primer lugar, tenemos que tener en cuenta que el cliente lo que quiere es una aplicación web. Hoy día existen innumerables tecnologías de desarrollo web, así como cantidad de Frameworks tanto *back-end* como *front-end* disponibles. Claro está que se usará la arquitectura cliente-servidor para el desarrollo del producto, por lo cual pasamos a analizar algunas tecnologías disponibles hoy día.

Teniendo en cuenta que a lo largo de mi formación durante estos años he aprendido y perfeccionado el lenguaje Java, vamos a analizar dos Frameworks mundialmente conocidos para la plataforma Java, de cara a poder construir una capa de negocio robusta y empleando tecnologías lo más modernas posibles, así como el uso de herramientas ORM (Mapeo Objeto-Relacional) actuales para la capa de persistencia. También se estudiaremos el SGBD utilizado y se justificará su uso, de cara a tener una capa de persistencia de calidad, consistente, robusta y libre de errores.

3.5.1. Java Enterprise Edition (J2EE). Java Server Faces (JSF)

J2EE [5] se trata de un framework de desarrollo basado en el lenguaje Java, que permite implementar soluciones de *back-end* en varias capas distribuidas. Dispone de numerosas API's independientes, que permiten implementar cada una de las capas del núcleo una aplicación empresarial. Aunque *J2EE* es de propósito general, puede ser utilizado con cualquier tecnología de desarrollo de aplicaciones web, aunque suele ser *Java Server Faces (JSF)* la favorita en este caso.

Lanzado por la empresa *Sun Microsystems* en el año 2001, inicialmente recibió muchas críticas debido a su complejidad. El australiano *Rod Jonhson*, en su libro *One-to-One J2EE Design and Development* propuso en el año 2002 un framework ligero que se opuso a *J2EE* y que posteriormente derivaría en *Spring Framework 1.0* en 2004. Actualmente *Spring Framework* es su mayor competidor en la plataforma Java.

Ilustración 6: Logotipo oficial de J2EE

3.5.2. Spring Framework. Spring MVC web Framework

Spring Framework [6] es un framework de código abierto para el desarrollo de aplicaciones y contenedor IoC (Inversion of Control) para la plataforma Java. Ideal también para la implementación del *back-end* de cualquier aplicación empresarial.

La primera versión fue escrita, tal y como se ha indicado en el apartado anterior, por el australiano *Rod Johnson* en el año 2004, tras el lanzamiento de su libro *One-to-One J2EE Design and Development* como crítica a *J2EE*, ganando así una gran aceptación y popularidad. Posteriores versiones de J2EE han mostrado un rediseño completo del framework siguiendo los principios de Spring Framework.

Al igual que *J2EE*, presenta una gran cantidad de API's y herramientas para la implementación de todas las capas del núcleo de una aplicación empresarial. A pesar de ser de propósito general, también posee una especificación para la implementación de aplicaciones web muy potente y muy utilizada en la actualidad: ***Spring MVC.***

Ilustración 7: Logotipo oficial de Spring Framework

3.5.3. Capa de negocio y capa de servicios: Spring Framework vs J2EE

Hemos visto dos tecnologías muy potentes hoy día para el desarrollo de aplicaciones empresariales, ambas con su especificación para implementación de aplicaciones web tal y como solicita el cliente. Toca responder a algunas preguntas que nos harán decantarnos por una tecnología u otra.

En primer lugar, cabe destacar que *J2EE* es bastante complejo e incluye numerosa funcionalidad no necesaria en la mayoría de aplicaciones, lo cual en primer lugar gana puntos *Spring*.

En segundo lugar, para la implementación de los *Beans* (objetos de negocio, los llamados *Enterprise JavaBeans*), en *J2EE* es necesario la implementación de varias interfaces complejas, así como un servidor de aplicaciones web, como pueden ser *GlashFish*, *Apache Tomcat* ó *JBoss*). Sin embargo, en ese aspecto *Spring Framework* es mucho más sencillo, ligero y modular, ya que sólo se utiliza la funcionalidad que sea necesaria en cada momento, y los *Beans* son implementados a través de POJO's, es decir, clases sencillas sin tener que implementar interfaces. También hay que estar pendientes de qué funcionalidad es y añadirla correctamente, por lo que eso puede ser un punto en contra. Aun así, prima la ligereza y modularidad de *Spring Framework*.

Lo cierto es que las posteriores versiones de *J2EE* han mejorado muchísimo y son realmente potentes, pero aun así en general *Spring Framework* sigue siendo más sencillo, lo que hace decantarme por la utilización de dicha tecnología.

3.5.4. Capa de persistencia: SGBD utilizado y uso de herramientas ORM

La capa de persistencia es aquella parte de la aplicación que permite que los datos de la misma puedan ser guardados en algún lugar de forma segura y que sobrevivan de forma permanente. Los datos son gestionados por la capa de negocio, que se comunica con la capa de persistencia cuando termina algún proceso y hay que guardar los datos de forma permanente.

Evidentemente el mecanismo más sencillo para esto es utilizar una base de datos. En este caso concretamente vamos a utilizar *MySQL* [7]. Algunas de las razones por las que me decanto por este SBDD son las siguientes:

- Se trata de un SGBD multiplataforma.
- Es fácil de utilizar.
- En caso de surgir algún problema su documentación oficial es bastante buena.
- Es una tecnología moderna, mundialmente conocida y utilizada por infinidad de programadores en la actualidad.
- No supone un coste elevado. Instalar un servidor sencillo no tiene coste alguno.
- Soporta transacciones con una calidad bastante elevada. Garantiza las propiedades ACID.

Ilustración 8: Logotipo oficial de MySQL

Por otro lado, cabe destacar el uso de una tecnología también muy utilizada en el desarrollo de la capa de persistencia en muchísimas aplicaciones hoy día, como es el uso de una herramienta de ORM (Mapeo Objeto-Relacional). Utilizaré la última versión de la especificación de *HIBERNATE* que soporta *Spring Framework*, como es la famosa API *JPA* (Java Persistence API), también mundialmente conocida y utilizada. [8]

Ilustración 9: Logotipo oficial de JPA

3.5.5. Capa de presentación: HTML5, CSS3. Twitter Bootstrap.

La capa de presentación es aquella con la que se produce la interacción persona-ordenador con la aplicación. En definitiva, se trata de la interfaz de usuario.

Tengamos muy en cuenta que el cliente nos ha solicitado sencillez y usabilidad a la hora de implementar la interfaz, por lo que en este caso la elección está clara teniendo en cuenta que el cliente debe ser web: uno de los frameworks de HTML5 y CSS3 también mundialmente conocido y utilizado como es *twitter Bootstrap* [9], cuya carta de presentación no es más que la facilidad de acceso a la gran cantidad y calidad de su documentación, la gran cantidad de elementos que tiene y la agradable sensación a la vista que provoca, además de lograr un diseño web responsive (adaptado a dispositivos móviles).

Ilustración 10: Logotipo oficial de Bootstrap

3.5.6. Control de versiones.

La utilización de una herramienta de control de versiones es fundamental para disponer siempre de una versión del código actualizada funcionando, lo que hace que podamos mostrar a nuestro cliente siempre una versión disponible. Independientemente de esto, también por seguridad. Es evidente que en cualquier momento puede ocurrir un contratiempo o un error inesperado y perder parte de nuestro código, por lo que es fundamental el control de versiones.

En nuestro caso, la herramienta de control de versiones elegida es *.git* [10], una de las más utilizadas por los programadores actuales y de gran aceptación a nivel profesional.

Ilustración 11: Logotipo oficial de .git

Como complemento a *.git*, utilizaremos una plataforma de almacenamiento en la nube para almacenar una copia de nuestro repositorio local, garantizando así doblemente la seguridad de nuestro código. Se utilizará la plataforma *Bitbucket* [11] de la empresa australiana *Atlassian*, principalmente porque permite la creación de repositorios privados sin necesidad de tener una cuenta de pago, cosa que por ejemplo no permite *github* [12], a pesar de ser una gran plataforma para el mismo fin que *Bitbucket*.

Ilustración 12: Logotipo oficial de Bitbucket

4. Estimación inicial del proyecto

4.1. Metodologías tradicionales vs metodologías ágiles

Una de las decisiones más difíciles a la hora de afrontar un proyecto software es la metodología a utilizar. Muchas son las ocasiones en las que un proyecto software fracasa debido a una mala elección de la metodología de trabajo o la mala ejecución de la misma, por lo que hay que ser muy cautelosos con la forma de trabajar y la forma en la que se desarrollan las tareas, los posibles cambios en el proyecto, o la implicación del cliente en el proceso del mismo.

Sin duda las metodologías tradicionales han quedado un poco atrás en lo que respecta al dinamismo de un proyecto. Una metodología tradicional es aquella que contempla una serie de fases de forma tradicional, en la que el cliente se despreocupa parcialmente del proyecto al no involucrarse en él directamente, optando únicamente al producto final que el equipo haya elaborado previo a una

serie de especificaciones (requisitos) iniciales. Nos podemos imaginar la serie de riesgos que esto puede conllevar, desde que el producto no sea el que el cliente espera, debido a una falta de entendimiento entre él y los miembros del equipo de desarrollo, o que los plazos de entrega no se cumplan debido a la mala estimación de las tareas a realizar por el gran desconocimiento tras el poco seguimiento que se realiza, así como una gran probabilidad de fracaso y la consecuente pérdida económica. Éste ha sido uno de los grandes quebraderos de cabeza de la Ingeniería del Software.

Hay que tener en cuenta que los equipos de trabajo en la actualidad se enfrentan a proyectos en los que hay constantemente cambios en los requisitos, bien sea por circunstancias ajenas a ellos o bien sea por circunstancias como por ejemplo un cambio de opinión del cliente durante el desarrollo del producto. En ese caso, y aunque sea de una forma un tanto pasiva, nos damos cuenta que el cliente forma parte también del proceso de desarrollo de su producto en el que desempeña un papel fundamental debido al gran dinamismo de un proyecto software. Este aspecto, como podemos observar, hace que el cliente tenga que estar al tanto continuamente de lo que está siendo su producto y como tal, debe ir viendo constantemente el desarrollo del mismo, lo que garantiza finalmente un alto porcentaje de éxito.

Las metodologías ágiles tratan de enfocar al equipo unas técnicas de trabajo de manera que puedan enfrentarse a este tipo de situaciones. El *manifiesto ágil* [13] ya propuso estas prácticas en una reunión celebrada en Utah, Estados Unidos, en la que 17 expertos en desarrollo de software elaboraron un esbozo, sobre cómo desarrollar software de forma colaborativa con el cliente y no sólo adaptándose a los cambios, sino incluso considerándolos bienvenidos en contraposición a las metodologías tradicionales que se caracterizaban por su rigidez y dirigidos por la extensa documentación generada en cada una de las actividades a desarrollar.

Los principios del *manifiesto ágil* [13] son los siguientes:

- **Individuos e interacciones**, sobre procesos y herramientas.
- **Software funcionando**, sobre documentación excesiva.
- **Colaboración con el cliente**, sobre negociación contractual.

- **Respuesta ante el cambio**, sobre seguir un plan.

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

Una vez analizados los principios del desarrollo ágil, nos toca decantarnos por utilizar una metodología tradicional o bien usar una metodología ágil. Teniendo en cuenta que tenemos un cliente con el que tiene intención de reunirse con nosotros periódicamente, y que probablemente en un alto porcentaje los requisitos que ya tenemos preestablecidos podrían cambiar, nos vamos a decantar por usar el desarrollo ágil, concretamente la denominada *Scrum Methodology* que se detallará en el apartado siguiente.

4.2. *Scrum Methodology* como metodología ágil utilizada

Las primeras preguntas que nos podemos hacer es qué es *Scrum* y por qué vamos a ponerla en práctica. La respuesta es sencilla, *Scrum* [14] [15] no es más que el nombre con el que se denomina a los marcos ágiles de desarrollo caracterizados por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación completa del producto.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.
- Solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada.

Se trata de una de las metodologías ágiles por excelencia y más utilizadas en el mundo por centenares de equipos de desarrollo de software. Pensado para trabajar colaborativamente en equipo y obtener resultados pronto. Está especialmente indicado en entornos complejos, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Se trata de una metodología basada en incrementos o *Sprints*, de una duración fija y normalmente de 2 semanas, aunque hay equipos en que duran hasta 4 semanas, donde en cada uno de ellos se desarrollan como microproyectos, con las fases de análisis, diseño, implementación, pruebas y entrega parcial al cliente.

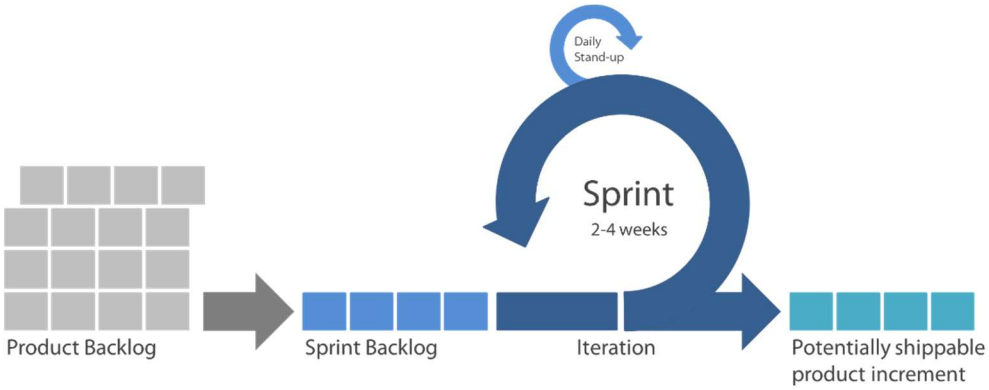

Ilustración 11: Proceso Scrum

Como en toda metodología, existen unos roles establecidos [16] entre los miembros del equipo, así como en los miembros que no forman parte del equipo pero sí forman parte en cierta manera del proceso de desarrollo. Se diferencian curiosamente con los nombres "*Roles Cerdo*" y "*Roles Gallina*", debido a que los primeros están comprometidos con el proyecto, frente a los segundos, que no forman parte directamente del proceso de desarrollo, pero sí hay que tenerlos en cuenta debido a la retroalimentación de cara a la especificación de requisitos de cada Sprint.

Roles Gallina son:

- El propietario del producto (*Product Owner*), que no tiene por qué ser el cliente. Es la persona que representa al cliente. En este caso este rol correspondería con el Prof. D. Antonio Jesús Rueda Ruiz.
- *Scrum Master*, podríamos llamar así al representante de proyecto de los miembros del equipo de desarrollo. En este caso este rol lo desempeñaría yo mismo, puesto que soy el único miembro del equipo de desarrollo.
- El propio equipo de desarrollo.

Roles Cerdo son: Todas aquellas personas que pueden aportarnos retroalimentación de cara a la especificación y puesta en marcha de cada Sprint. Son los llamados *Stakeholders*¸ y serían los potenciales usuarios de la aplicación. Como *Stakeholders* incluiríamos también al Prof. D. Antonio Jesús Rueda Ruiz y al Prof. D. Carlos Porcel Gallego.

En *Scrum* existen una serie de reuniones estandarizadas [17] que llevaremos a cabo durante el desarrollo de las iteraciones que son las siguientes:

- **Reunión de planificación del *Sprint*:** Es la reunión en la que se planifica el trabajo que se va a desarrollar durante lo que dura el *Sprint.*
- **Reunión diaria:** Se realiza por el equipo de desarrollo cada día a primera hora laboral donde se cuenta lo que se hizo el día anterior y se determina el trabajo a realizar durante ese mismo día.
- **Reunión de cierre:** Donde se expone una demo con los requisitos cumplidos del mismo y se realiza una retroalimentación para futuras correcciones en iteraciones posteriores. A esta reunión pueden asistir también los llamados *Stakeholders*.
- **Reunión de retrospectiva:** Podíamos llamarla como una reunión de celebración únicamente del equipo de desarrollo tras la finalización del proceso *Scrum*, en la cual suele haber comida y bebida de por medio, en la que se analiza cómo ha ido el proceso y en el que caso de que fuera necesario, corregir fallos internos del equipo.

4.3. Tareas a desarrollar y planificación ágil

En este apartado especificaremos en qué consiste el *Product Backlog*, la especificación de tareas que componen el mismo así como la estimación ágil de cada una de ellas. Teniendo en cuenta que estamos ante un proceso de desarrollo en el que los requisitos son cambiantes, es difícil predecir un presupuesto cerrado sobre las tareas que vamos a desarrollar. En ese caso, vamos a jugar con un margen de forma que el cliente se haga una idea del alcance del proyecto y que el presupuesto sea lo más acertado posible.

El *Product Backlog* [18] no es más que la lista de objetivos a tener en cuenta a la hora de plasmar los requisitos de la aplicación que hay que tener en cuenta para crear un producto satisfactorio para el cliente. A lo largo del proceso *Scrum* el contenido del *Product Backlog* va viéndose alterado en función del trabajo planificado y realizado en el Sprint. Evidentemente aunque conforme van avanzando los Sprint éste se va a ir reduciendo, también podrán añadirse nuevos elementos al mismo, quedando patente una vez más la dinamicidad del proceso de desarrollo de software. A los elementos que forman el *Product Backlog* se les denomina Historias de Usuario (*Scrum User Stories*) [19].

Una vez tenemos nuestro *Product Backlog* es hora de elaborar un presupuesto. Para iniciar el procedimiento, en primer lugar debemos estimar de forma ágil las historias de usuario que componen el *Product Backlog*. Para ello, emplearemos una estimación ágil en puntos de historia. Un punto de historia no es más que un valor (normalmente suele ser un número dentro de la sucesión de Fibonacci) con el que el equipo de desarrollo determina de forma consensuada la dificultad de una historia de usuario. Podríamos usar una técnica de estimación ágil en equipo como puede ser *Planning Poker* [20] o similar para estimar los puntos de historia, pero es algo que a primera vista podría parecer superfluo debido a que el equipo de desarrollo lo formo únicamente yo.

Aquí surge una pregunta, ¿por qué estimar en historias de usuario en lugar de en horas de trabajo? [21] La respuesta es simple. Imaginemos por un momento que somos un transportista y que tenemos que viajar desde una ciudad A hasta una ciudad B de la que distan 480 kilómetros, son las 8 de la mañana y nos preguntan a qué hora vamos a llegar. Si tenemos en cuenta que nuestra velocidad media es unos 80 km/hora podríamos decir que más o menos llegaríamos a las 14 horas, puesto que a 80 km/hora se tardan 6 horas en recorrer 480 kilómetros. Básicamente con el desarrollo ágil pasa lo mismo. Es decir, sabemos el alcance de dificultad del proyecto en base a nuestro criterio, pero no podemos decir con exactitud cuándo podemos acabar. En base a nuestra experiencia determinaremos la velocidad de trabajo que somos capaces de llevar a cabo de una forma lo más sensata y prudente, que se mide en puntos de historia por iteración.

Pero tenemos un problema, y es que nuestro cliente nos exige una fecha tope de entrega a la hora de elaborar un presupuesto. Pues bien, debemos ser conscientes y acarrear un margen suficiente de error para posibles complicaciones que nos puedan surgir a la hora de realizar las tareas y que puedan implicar retraso del tiempo planificado.

A continuación se detallan de forma muy general las historias de usuario que componen el *Product Backlog* y la dificultad estimada en puntos de historia. Las historias de usuario de cada bloque se irán detallando a lo largo del desarrollo de las iteraciones, puesto que son cambiantes y es difícil determinarlas al inicio del proyecto. Posteriormente determinaremos la velocidad de trabajo a las que seremos capaces de trabajar y estaremos en unas condiciones mínimas de elaboración de una planificación inicial y un presupuesto.

NOTA: Para mayor comodidad en la organización de las historias de usuario se dividirán en dos bloques: las relacionadas al *back-end* y las relacionadas al *front-end*.

Historias de usuario *back-end*	Puntos de historia
Diseño e Implementación del modelo de datos	5
Diseño e implementación de la capa de persistencia	8
Lógica de negocio: Solicitar registro	8
Lógica de negocio: Confirmar registro	13
Lógica de negocio: Cambiar contraseña de acceso	8
Lógica de negocio: Actualizar perfil de usuario	5
Lógica de negocio: Baja de usuario	13

Lógica de negocio: Enviar petición de amistad	13
Lógica de negocio: Peticiones de amistad recibidas	5
Lógica de negocio: Gestionar petición de amistad	8
Lógica de negocio: Añadir nueva escuela	5
Lógica de negocio: Añadir nuevo sector	5
Lógica de negocio: Añadir nueva vía	5
Lógica de negocio: Comentar vía	8
Lógica de negocio: Editar comentario	13
Lógica de negocio: Eliminar comentario	13
Lógica de negocio: Realizar vía	34
Lógica de negocio: Escuelas de una provincia	8
Lógica de negocio: Sectores de una escuela	8
Lógica de negocio: Vías de un sector	8
Lógica de control: Cerrar Sesión	3

Tabla 3: Historias de usuario pertenecientes al *back-end* y estimación de dificultad en puntos de historia

Historias de usuario *front-end*	Puntos de historia
Página de portada	3
Formulario de inicio de sesión	5
Barra de navegación de páginas	8
Página de perfil	21
Formulario de actualizar perfil de usuario	8
Formulario de cambio de contraseña	8
Selector de búsqueda (buscador)	5
Buscador de usuarios	8
Buscador de escuelas	8
Ficha de escuela	13
Ficha de sector	13
Ficha de vía y valoraciones	21
Formulario para valoración de vías	13
Formulario para añadir escuela	8

Formulario para añadir sector	8
Formulario para añadir vía	8

Tabla 4: Historias de usuario pertenecientes al *front-end* y estimación de dificultad en puntos de historia

Historias de usuario documentación y presentación	Puntos de historia
Documentación de la memoria de proyecto	55
Preparación de la presentación final	13

Tabla 5: Historias de usuario pertenecientes al desarrollo de la documentación y preparación de la presentación final

En resumidas cuentas, tenemos un total de 188 puntos de historia en lo que respecta al *back-end* y 158 puntos de historia en lo que respecta al *front-end*, lo que suman un total de 346 puntos de historia en lo que al desarrollo del producto se refiere. Al ser un TFG y tener que redactar una memoria adjunta voy a estimar otros 55 puntos de historia para la realización de la misma y otras 13 en la preparación de la presentación final. En total suman 414 puntos de historia a planificar.

Pues bien, en primer lugar seamos conscientes de la velocidad que somos capaces de llevar a cabo. Bien es cierto que es la primera vez que me enfrento a un proyecto real de estas características, que hasta ahora todo habían sido prácticas como alumno, y que hay que ser extremadamente cautelosos a la hora de determinar la velocidad de trabajo para poder cumplir los plazos y objetivos.

Por tanto, y teniendo en cuenta que me gustaría realizar la entrega a finales de junio de 2017, voy a planificar con suficiente antelación por si surgen cambios durante el desarrollo del proyecto, que ya adelanto que los habrá.

Analizando detenidamente las tareas y la dificultad en puntos de historia de las historias de usuario, determino que acorde a mi experiencia voy a ser capaz de trabajar con una velocidad ágil en torno a las 40 puntos de historia por iteración, lo que me hace planificar inicialmente el trabajo durante 11 iteraciones de dos semanas laborables (10 días de 7 horas de trabajo cada uno). Sólo se tendrá en cuenta la planificación de tiempo estimado en base a los puntos de historia, ya que las tareas seleccionadas para desarrollar en cada iteración se hacen a lo largo del proceso *Scrum*). Si se disponen de 414 puntos de historia entre 40 puntos de historia que somos capaces de llevar a cabo por iteración, son 10.35, lo cual son 11 iteraciones.

No se tendrá en cuenta última semana de 2016 ni la primera de 2017, semanas de Navidad y Año Nuevo respectivamente. La velocidad media de trabajo sería 37.63 puntos de historia por iteración de los 40 que somos capaces de llevar a cabo, por lo que dejo esos 2.37 puntos de historia de margen por iteración. Esto no quiere decir que en cada iteración se llevarán a cabo entregas de 40 puntos de historia, puesto que los números no son exactos y habrá veces que se entreguen menos de 40 puntos de historia, y veces que se entreguen más de 40 puntos de historia. Por ejemplo, supongamos que estamos en la iteración *n* y que faltan dos días para la celebración de la reunión de cierre del *Sprint,* y que la siguiente historia de usuario a realizar tiene una estimación de 21 puntos de historia. Aunque esa historia de usuario se empiece a desarrollar en la iteración *n,* formaría parte de la entrega de la iteración *n+1*, ya que sobrepasa con creces la velocidad que somos capaces de llevar a cabo en una iteración.

Por tanto la planificación ágil de las iteraciones quedaría de la siguiente forma:

Iteración	Plazo de ejecución
1	**19/09/2016 - 30/09/2016**
2	**3/10/2016 – 14/10/2016**

3	**17/10/2016 – 28/10/2016**
4	**31-10-2016 – 11/11/2016**
5	**14/11/2016 – 25/11/2016**
6	**28/11/2016 – 9/12/2016**
7	**12/12/2016 – 23/12/2016**
8	**9/01/2017 – 20/01/2017**
9	**23/01/2017 – 3/02/2017**
10	**6/02/2017 – 17/02/2017**
11	**20/02/2017 – 3/03/2017**

Tabla 5: Planificación ágil de las iteraciones

El siguiente diagrama de quemado [22] representa el número de puntos de historia restantes al final de cada iteración. No es más que la evolución ágil tras la estimación ágil de las historias de usuario y planificación de las iteraciones en base a la velocidad de trabajo que vamos a llevar a cabo, en este caso 40 puntos de historia por iteración, y que se puede ver reflejada en la línea de tendencia de color rojo. Como detalle decir que la última iteración sería algo más breve al ser sólo de 14 puntos de historia, básicamente porque son los últimos 14 a desarrollar.

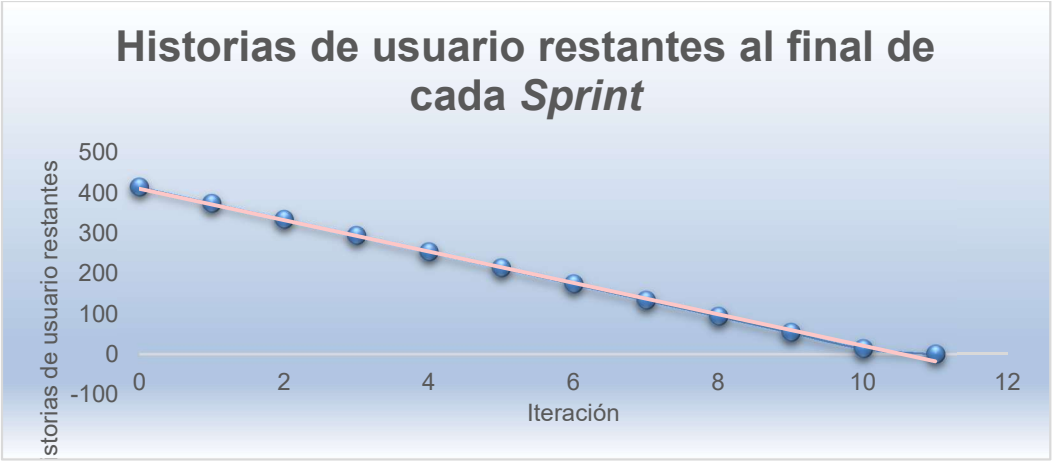

Ilustración 13: Diagrama de quemado tras la planificación ágil inicial

4.4. Presupuesto inicial del proyecto

Para iniciar el presupuesto es necesario tener en cuenta todos los puntos de historia, así como el número de iteraciones empleadas y la duración de cada una de ellas. Sabiendo esto, podemos calcular aproximadamente el tiempo que se empleará en la realización de una historia de usuario.

Tenemos en cuenta que el proyecto está estimado para un total de 11 iteraciones de una duración de dos semanas. Cada iteración son 10 días laborables y en cada día laborable vamos a emplear 7 horas de trabajo. En total suman 770 horas de trabajo. De esta forma podemos calcular el tiempo aproximado empleado por historia de usuario. Si tenemos 414 puntos de historia y empleamos un total de 770 horas de trabajo en completarlos estimamos que cada punto de historia tiene una duración media de 1.86 horas, lo cual se emplea 2 horas de trabajo por historia de usuario. Al cliente se le cobrarán un total de 830 (828) horas de trabajo. Como vemos se ha dado un cierto margen al cliente en base a las tareas estimadas. Por lo tanto, y en resumidas cuentas, desglosamos los conceptos del presupuesto de la siguiente forma:

En esta primera tabla resumimos el coste total del equipo de desarrollo. En este caso es el de una sola persona analista/programador con un sueldo anual de 30000€.

Salario analista/programador	Precio/hora	Total de horas	Total
30.000 € anuales	**16,23 €/hora**	**830 horas**	**13.474,25 €**

Tabla 7: Tabla con la descripción de los costes en salarios

Aquí incluimos la amortización del equipo informático necesario.

Equipo informático	Precio del equipo	Vida útil del equipo	Duración del proyecto	Total
Portátil Acer Aspire ES1-571-507Y, i5 4GB, 500GB	**600 €**	**5 años**	**7 meses**	**70 €**

Tabla 8: Tabla con la descripción de los costes en material informático

Cabe destacar que no hay costes en Software, ya que los entornos de desarrollo y las herramientas de trabajo empleadas, son o bien *open source*, o bien hemos utilizado una cuenta *free* en aquellas herramientas que son comerciales, lo cual hace que no haya que incluir en el presupuesto costes por estos conceptos. Si posteriormente hubiera modificación de requisitos que así lo requiriera, habría que incluir en la planificación y el presupuesto del proyecto dichos costes en base a los nuevos requisitos. En un principio todo parece indicar que esto no sucederá. Sí se incluyen en el presupuesto los conceptos de dominio y alojamiento web con las características de nuestro proyecto y que se detallan en el desglose del presupuesto.

En esta última tabla se resumen todos los conceptos del presupuesto del proyecto.

Concepto	Importe
Costes salariales	**13.474,25 €**
Equipo informático	**70 €**
Dominio web y hosting 1 año [23]	**76 €**
Costes totales…	**13.620,25 €**

Beneficio sobre costes 20 %	**2.724,05 €**
Base imponible	**16.344,30 €**
Impuestos I.V.A. 21 %	**3.432,30 €**
Importe total del proyecto	**19.776,60 €**

Tabla 8: Desglose de conceptos del presupuesto del proyecto

El cliente deberá abonar un 25% del total del presupuesto antes del inicio del proyecto a modo de fianza, y el resto se abonará a la finalización del proyecto.

5. Análisis, diseño e implementación del *back-end*: capa de negocio y capa de persistencia

Se denomina *back-end* a la parte del desarrollo que se encarga de gestionar las entradas de datos desde la interfaz de usuario. Comúnmente se le conoce como la parte de la aplicación que se encuentra en el lado del servidor, y centra el llamado *kernel* de la aplicación.

Es muy importante la buena organización y distribución del mismo ya que gestiona partes muy importantes y clave como son la capa de persistencia, la lógica de negocio o la seguridad de la aplicación. En el siguiente apartado especificaremos la arquitectura empleada en nuestro *back-end* describiendo cada una de las capas que lo forman.

5.1. Arquitectura de 4 capas

En el análisis técnico del proyecto se ha podido observar una breve introducción a la arquitectura del que serán las capas del *back-end* de la aplicación y las tecnologías analizadas y que se emplearán para ello. No obstante vamos a realizar una descripción a fondo de lo que consiste la arquitectura empleada.

5.1.1. Capa del modelo de datos

Es la capa central del *back-end*. En realidad es la base de la capa de negocio. En ella se especifica el modelo de datos y sus relaciones a tener en cuenta para la

realización de las capas de persistencia y de negocio, estableciendo así las entidades primarias que forman nuestra aplicación.

El modelo de datos es una extensión del modelo de dominio, afinando los tipos de datos elegidos y las relaciones entre las distintas entidades que lo forman, de manera que es la base utilizada tanto para implementación de la capa de persistencia como para la implementación de la capa de negocio.

5.1.2. Capa de persistencia

Es aquella que permite que los datos que así lo necesiten puedan guardarse de forma permanente. Esta capa simplifica el acceso a la base de datos eliminando las características particulares de cada SGBD.

5.1.3. Capa de negocio

Implementa la lógica de negocio. No son más que las reglas de negocio en base al modelo de datos con las que trabaja la aplicación y para ello se utilizan los llamados objetos de negocio.

5.1.4. Capa de servicios o capa de control

Se encarga de servir al cliente los datos que se gestionan en la capa de negocio. Para ello normalmente se utilizan protocolos estandarizados como es el caso del protocolo HTTP que es el que vamos a utilizar. En nuestro caso será un controlador *Spring web MVC* quien implemente esta capa. Posteriormente se explicará en qué consiste el patrón MVC usando esta tecnología.

5.2. Iteración 1: del 19/09/2016 al 30/09/2016

5.2.1. Análisis de la iteración 1

En esta iteración vamos a seleccionar las primeras historias de usuario en lo que respecta al *back-end* de la aplicación. Se seleccionan las historias de usuario: ***diseño e implementación del modelo de datos***, ***diseño e implementación de la capa de persistencia, solicitar registro y confirmar registro*** que suman un total de 42 puntos de historia. Antes de proceder al desarrollo de las mismas, es necesario previamente detallar el modelo de datos en base al modelo de dominio y

añadir aquellas entidades que consideremos oportunas. Finalmente se creará el diagrama de clases de diseño.

5.2.2. Diagrama de clases de diseño: diseño e implementación del modelo de datos y la capa de persistencia

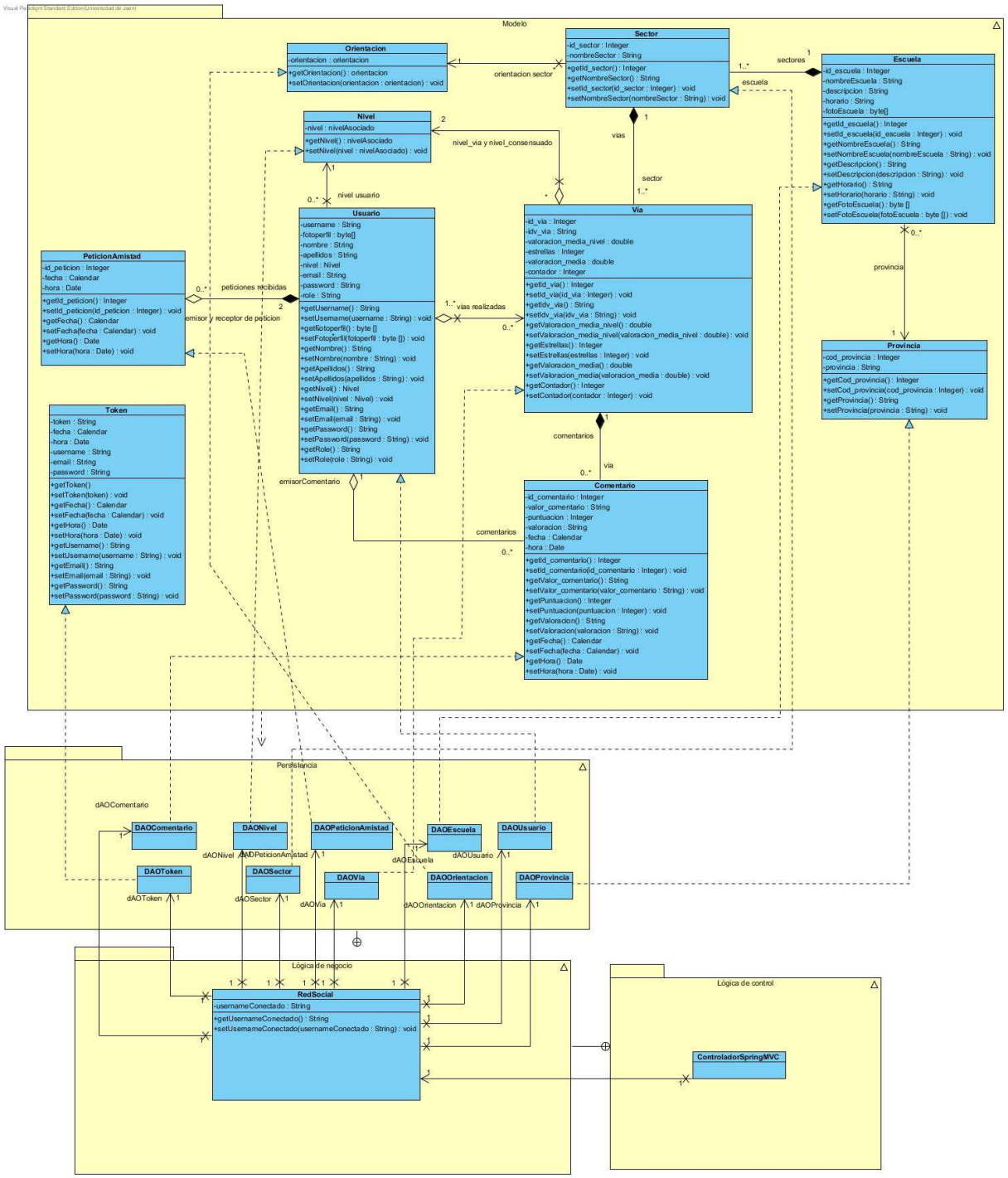

Ilustración 14: Diagrama de clases de diseño

El diagrama de clases de diseño representa las 4 capas. Un primer paquete que representa el modelo de datos de la aplicación. El modelo lo componen 10

entidades cuya principal es la entidad usuario. Ésta tiene una agregación cíclica que representa la colección de amigos que un usuario puede tener. También tiene una agregación con las vías que ha realizado, otra con los comentarios que ha escrito y una relación simple con un nivel asociado.

Hay varias entidades simples que carecen de relaciones, como son el token que nos va a servir para crear un registro con un usuario temporal que sólo se activará si el usuario realiza la confirmación y validación del registro. La entidad nivel también es una entidad simple, pues básicamente su función es representar los distintos niveles de dificultad que se asocian a las vías y a los usuarios. También está la entidad orientación, que nos sirve para representar la orientación de un sector y la entidad provincia, que representa la provincia a la que una escuela puede pertenecer.

La entidad petición de amistad a su vez tiene una agregación de dos usuarios, el emisor y el receptor de dicha petición. La entidad escuela representa las escuelas, y tiene una relación simple con una provincia para determinar su ubicación. También tiene una composición de los sectores que la forman. Esa misma relación en sentido inverso nos da la escuela de un sector. Los sectores también tienen una asociación de las vías que lo forman. Análogamente la relación inversa nos da el sector al que pertenece una vía. Las vías tienen una agregación de 2 niveles que son los que representan el nivel oficial y el nivel consensuado de la vía.

NOTA: Todas las clases que implementan el modelo de datos utilizan la sintaxis oficial de HIBERNATE JPA [24] mediante anotaciones, en base a su diseño, que determinará la construcción de las tablas en la base de datos de forma automática en 3FN. Es la técnica utilizada y llamada ORM directo (Mapeo objeto-relacional). De ahí que se hable de entidades.

A continuación se detalla brevemente la implementación de todas y cada una de las clases que forman las entidades de nuestra aplicación y que dan lugar a la formación de las tablas de la base de datos en 3FN, utilizando la sintaxis JPA para la implementación de la capa de persistencia.

Entidad Usuario:

```
@Entity
public class Usuario implements Serializable
{
    @Id
    private String username;
    @Lob
    private byte[] fotoperfil;
    private String nombre;
    private String apellidos;
    @ManyToOne (fetch = FetchType.EAGER)
    private Nivel nivel;
    private String email;
    private String password;
    private final String role;
    @ManyToMany (fetch = FetchType.EAGER)
    private final List<Usuario> amigos;
    @ManyToMany (fetch = FetchType.EAGER)
    private final List<Via> viasRealizadas;
    @ManyToMany (fetch = FetchType.EAGER)
    private final List<PeticionAmistad> peticionesAmistad;
    @OneToMany (fetch = FetchType.EAGER)
    private final List<Comentario> comentarios;
```

Ilustración 15: Implementación de la entidad Usuario

Entidad Comentario:

```
@Entity
public class Comentario implements Serializable
{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id_comentario;
    private String comentario;
    @ManyToOne (fetch = FetchType.EAGER)
    private Usuario usuario;
    @ManyToOne (fetch = FetchType.EAGER)
    private Via via;
    private Integer puntuación;
    private String valoracion;
    @Temporal(javax.persistence.TemporalType.DATE)
    private final Calendar fecha;
    @Temporal(javax.persistence.TemporalType.TIME)
    private final Date hora;
```

Ilustración 16: Implementación de la entidad Comentario

Entidad Escuela:

```
@Entity
public class Escuela implements Serializable
{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id_escuela;
    private String nombreEscuela;
    private String descripcion;
    private String horario;
    @Lob
    private byte[] fotoEscuela;
    @ManyToOne (fetch = FetchType.EAGER)
    private final Provincia ubicacion;
    @OneToMany (fetch = FetchType.LAZY)
    private final List<Sector> sectores;
```

Ilustración 17: Implementación de la entidad Escuela

Entidad Nivel:

```
@Entity
public class Nivel implements Serializable
{
    public enum
nivelAsociado{_1,_2,_3,_4,_5,_5m,_6a,_6am,_6b,_6bm,_6c,_6cm,_7a,_7am,_7b
,_7bm,_7c,_7cm,_8a,_8am,_8b,_8bm,_8c,_8cm,_9a,_9am,_9b,_9bm,_9c,_9cm};

    @Id
    private nivelAsociado nivel;
```

Ilustración 18: Implementación de la entidad Nivel

Entidad Orientación:

```
@Entity
public class Orientacion implements Serializable
{
    public enum orientacion{N,S,E,O,NE,SE,SO,NO};

    @Id
    private orientacion orientacion;
```

Ilustración 19: Implementación de la entidad Orientación

Entidad PeticionAmistad:

```
@Entity
public class PeticionAmistad implements Serializable
{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id_peticion;
    @ManyToOne (fetch = FetchType.EAGER)
    private Usuario emisor;
    @ManyToOne (fetch = FetchType.EAGER)
    private Usuario receptor;
    @Temporal(javax.persistence.TemporalType.DATE)
    private final Calendar fecha;
    @Temporal(javax.persistence.TemporalType.TIME)
    private final Date hora;
```

Ilustración 20: Implementación de la entidad PeticionAmistad

Entidad Sector:

```
@Entity
public class Sector implements Serializable
{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id_sector;
    private String nombreSector;
    @Lob
    private byte[] fotoSector;
    @ManyToOne (fetch = FetchType.EAGER)
    private Escuela escuela;
    @ManyToOne (fetch = FetchType.EAGER)
    private final Orientacion orientacion;
    @OneToMany (fetch = FetchType.LAZY)
    private final List<Via> vias;
```

Ilustración 21: Implementación de la entidad Sector

Entidad Provincia:

```
@Entity
public class Provincia implements Serializable
{
    @Id
    private Integer cod_provincia;
    private String provincia;
```

Ilustración 22: Implementación de la entidad Provincia

Entidad Token:

```
@Entity
public class Token implements Serializable
{
    @Id
    private String token;
    @Temporal(javax.persistence.TemporalType.DATE)
    private final Calendar fecha;
    @Temporal(javax.persistence.TemporalType.TIME)
    private final Date hora;
    private String username;
    private String email;
    private String password;
```

Ilustración 23: Implementación de la entidad Token

Entidad Vía:

```
@Entity
public class Via implements Serializable
{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id_via;
    private String idv_via;
    private String nombre;
    @ManyToOne (fetch = FetchType.EAGER)
    private Nivel nivel;
    @ManyToOne (fetch = FetchType.EAGER)
    private Nivel nivelConsensuado;
    private double valoracion_media_nivel;
    private Integer estrellas;
    private double valoracion_media;
    @OneToMany (fetch = FetchType.LAZY)
    private final List<Comentario> comentarios;
    @ManyToOne (fetch = FetchType.EAGER)
    private Sector sector;
    private Integer contador;
```

Ilustración 24: Implementación de la entidad Vía

Con esta implementación, evidentemente con la previa configuración en el fichero de contexto de *Spring Framework* con el paquete que contiene las clases a mapear, queda creada automáticamente en 3FN la base de datos que vamos a utilizar para nuestra capa de persistencia.

Por otro lado, en lo que respecta al paquete que representa la capa de persistencia, están presentes unas clases llamadas DAO's (uso del patrón DAO) que

son las encargadas de comunicarse con la base de datos y producir objetos del modelo que se gestionarán en la capa de negocio. Estos objetos (los DAO's) a su vez utilizan el API JPA para hacer las distintas operaciones de acceso a la base de datos, que evidentemente también hay que configurar en el fichero de contexto. Un ejemplo de implementación de DAO que se adjunta es el de usuario. El resto funcionan todos de igual forma:

```
@Repository
@Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
public class DAOUsuario
{
    @PersistenceContext
    EntityManager em;

    /**
     *
     */
    public DAOUsuario(){}

    /**
     *
     * @param _usuario
     */
    @Transactional (propagation = Propagation.REQUIRED, readOnly = false, rollbackFor =
exceptionsDAO.GuardarUsuarioException.class)
    public void guardarUsuario(Usuario _usuario)
    {
        em.persist(_usuario);
    }

    /**
     *
     * @param _usuario
     */
    @Transactional (propagation = Propagation.REQUIRED, readOnly = false, rollbackFor =
exceptionsDAO.ActualizarUsuarioException.class)
    public void actualizarUsuario(Usuario _usuario)
    {
        em.merge(_usuario);
    }

    /**
     *
     * @param _usuario
     * @brief elimina un usuario de la base de datos
     */
    @Transactional (propagation = Propagation.REQUIRED, readOnly = false, rollbackFor =
exceptionsDAO.BorrarUsuarioException.class)
    public void borrarUsuario(Usuario _usuario)
    {
        em.remove(em.merge(_usuario));
    }

    /**
     *
     * @param _username
     * @return devuelve el usuario de la base de datos a partir de su _username
     */
    public Usuario obtenerUsuario(String _username)
    {
        return em.find(Usuario.class, _username);
    }

}
```

Ilustración 25: Implementación del DAO que gestiona la entidad Usuario

En el paquete que representa la lógica de negocio está el objeto de negocio de nuestra aplicación, donde implementaremos la lógica de negocio (historias de usuario del *back-end*) que servirá para posteriormente dar servicios mediante un controlador *SpringMVC*, que se encuentra en el paquete de la lógica de control.

5.2.3. Lógica de negocio: Solicitar registro

Como usuario potencial de la aplicación, el usuario deberá completar un sencillo proceso de registro para poder acceder a la misma. Este proceso consistirá en la elección de un nombre de usuario, un email asociado y una contraseña. Hay que comprobar si ese nombre de usuario ya es utilizado por otra persona, por lo que una sencilla consulta a la base de datos para comprobar si existe un usuario con esas características será suficiente, y lanzar una excepción en caso de que esto ocurra.

La clase Token es la que juega un papel fundamental en esta historia de usuario, puesto que se crea un token con los datos seleccionados y se guarda en la base de datos con fecha y hora actuales. Ese token es enviado al correo que el usuario previamente ha proporcionado y tendrá que confirmarlo durante los próximos 30 minutos (siguiente historia de usuario).

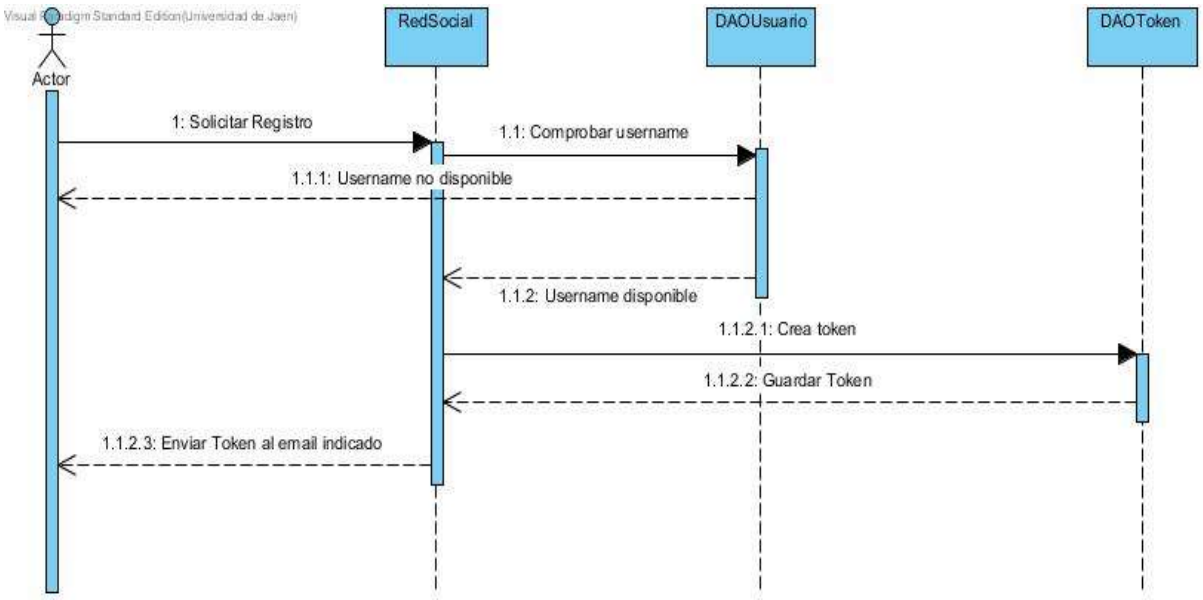

Ilustración 26: Diagrama de secuencia de solicitar registro

5.2.4. Lógica de negocio: Confirmar registro

Cuando el usuario confirma la URL que le llegará a su correo electrónico, ésta llevará implícitamente el valor del token, que será lo que recogerá nuestra lógica de negocio, comprobará si es válido (no tiene un tiempo superior a 30 minutos desde la fecha de creación) y validará el registro, introduciendo ya el usuario en la tabla correspondiente con los datos que le suministre el token.

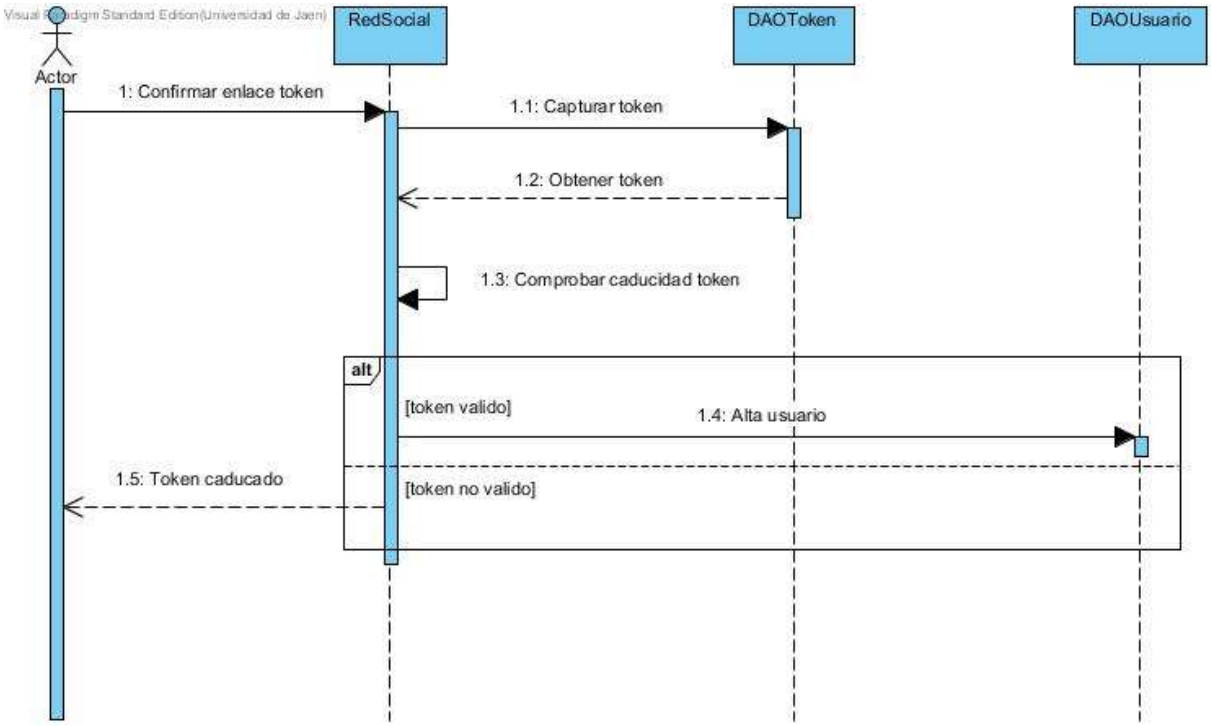

Ilustración 27: Diagrama de secuencia de confirmar registro

5.2.5. Cierre de la iteración 1

Se han desarrollado y presentado un total de 42 historias de usuario en esta primera iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 372 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

5.3. Iteración 2: del 3/10/2016 al 14/10/2016

5.3.1. Análisis de la iteración 2

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Cambiar contraseña: 8 puntos de historia
- Actualizar perfil: 5 puntos de historia
- Baja de usuario: 13 puntos de historia
- Enviar petición de amistad: 13 puntos de historia

Total de puntos de historia seleccionados para la segunda iteración: 39

5.3.2. Lógica de negocio: Cambiar contraseña

Como usuario, se podrá en todo momento cambiar su contraseña siempre y cuando el usuario lo desee. Para ello, es necesario previamente introducir la contraseña actual y comprobar que es correcta. Una vez hecho esto, el la contraseña introducida por el usuario y su confirmación pasarán a ser la contraseña actual.

A efectos de implementación se trata solamente de sustituir el atributo *password* actual por la nueva introducida. Para ello es importante previamente codificar la contraseña con el algoritmo de encriptación *Bcrypt* que es el que será utilizado para cifrar las contraseñas en la base de datos. Una vez sustituido hay que guardar en persistencia los cambios producidos con el DAO de la entidad usuario.

La identificación del usuario (el atributo *username* del objeto de negocio) tendrá lugar en la capa de servicio mediante el uso de *Spring Security* y será detallado más adelante.

5.3.3. Lógica de negocio: Actualizar perfil

Como usuario, es evidente que también se podrán actualizar los datos del perfil. Para ello, lo que se hace es modificar los atributos nombre, apellidos, email y foto de perfil. Destacar que la foto de perfil viene codificada en *base64*, por lo cual hay que descodificar antes de modificar. Una vez hecho todo esto se llama al DAO de usuario que guarda todos los cambios en la base de datos.

5.3.4. Lógica de negocio: Baja de usuario

La baja de usuario es un proceso algo más complejo. Tengamos en cuenta que el usuario tiene amigos, vías realizadas, comentarios y valoraciones, y que éstas están a su vez ligadas a otras entidades. Entonces, hay que “deshacer” todas esas colecciones con las que el usuario está ligado para que no se produzcan fallos de violación de integridad en la base de datos.

En primer lugar hay que eliminar la lista de amigos. Para ello, hay que ir uno por uno y eliminar al usuario de la lista de amigos de cada uno de sus amigos.

En segundo lugar, los comentarios. Tener en cuenta que un comentario no solo tiene un usuario asociado, sino también la vía a la que ha sido relacionado. Por tanto, deshacer la relación de todos los comentarios con la vía y deshacer la relación con el usuario en cuestión para finalmente, eliminar el comentario.

Una vez finalizado este proceso se podrá proceder a borrar las colecciones propias del usuario y una vez hecho esto, ahora sí, podremos eliminar al usuario de la base de datos, y por consiguiente, de la aplicación.

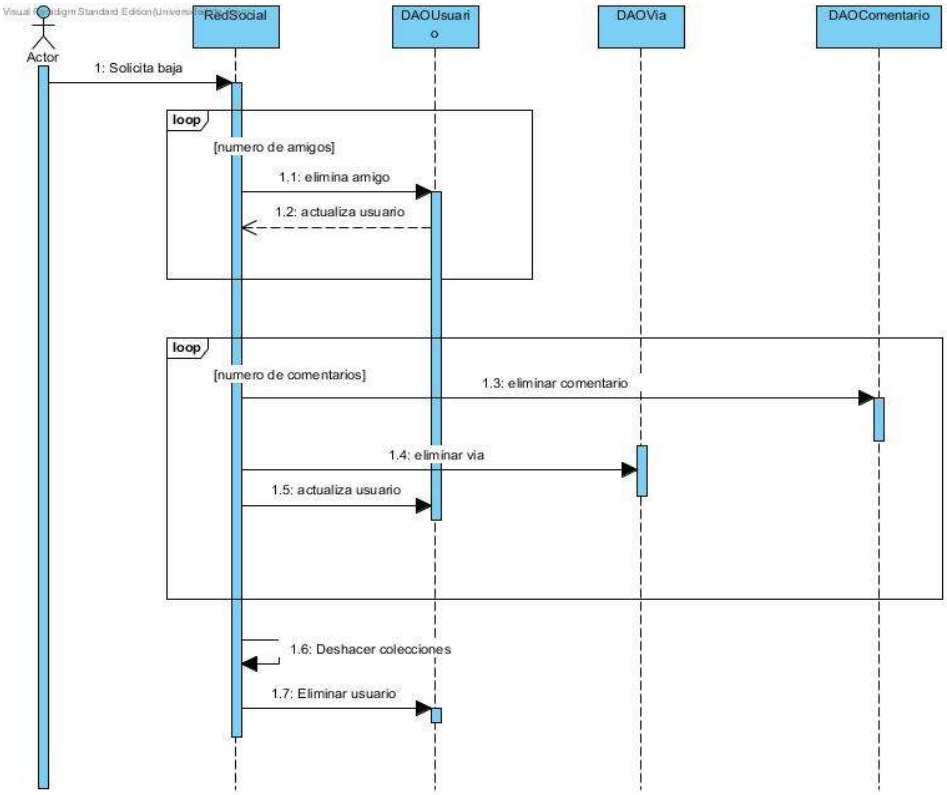

Ilustración 28: Diagrama de secuencia de baja de usuario

5.3.5. Lógica de negocio: Enviar petición de amistad

En esta historia de usuario, como usuario puedo enviar peticiones de amistad a otro perfil. Tras analizar el modelo es más sencillo de lo que puede parecer. Para ello está la colección de peticiones de amistad que cada usuario tiene.

Pues bien, no es más que crear una petición en la lógica de negocio con el emisor (el usuario conectado, el atributo *username* del objeto de negocio identificado por *Spring Security* en la capa de servicios) y el receptor, y añadirlo en la colección del usuario receptor. De esa forma el usuario receptor podrá ver que tiene una petición de amistad pendiente de gestionar al no estar su colección vacía.

Si una misma petición fuera enviada dos veces antes de ser gestionada, entonces se lanzaría una excepción. Esto es, cuando el intentamos introducir una petición en la colección del usuario receptor y el emisor de la misma ya está presente en alguna petición de la colección del receptor que esté sin gestionar.

5.3.6. Cierre de la iteración 2

Se han desarrollado y presentado un total de 39 historias de usuario en esta segunda iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 333 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

5.4. Iteración 3: del 17/10/2016 al 28/10/2016

5.4.1. Análisis de la iteración 3

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Peticiones de amistad pendientes: 5 puntos de historia
- Gestionar petición de amistad: 8 puntos de historia
- Añadir escuela: 5 puntos de historia
- Añadir sector: 5 puntos de historia
- Comentar vía: 8 puntos de historia
- Añadir vía: 5 puntos de historia

Total de puntos de historia seleccionados para la tercera iteración: 36

5.4.2. Lógica de negocio: Peticiones de amistad pendientes

Como usuario, puedo ver las solicitudes de amistad pendientes de gestionar. Es posiblemente una de las historias de usuario más sencillas de implementar. Para ello, basta con devolver la lista que el usuario tiene para tal fin. Evidentemente si esa lista está vacía quiere decir que no hay peticiones de amistad pendientes de gestionar.

5.4.3. Lógica de negocio: Gestionar petición de amistad pendiente

Como usuario, puedo gestionar las peticiones de amistad que tenga pendientes. Para gestionar una, basta con capturar el identificador de la misma e indicar si se acepta o si se rechaza con una variable booleana.

En primer lugar se comprueba si se acepta o se rechaza la petición. En caso de que se acepte hay que añadir al emisor en nuestra lista de amigos y nosotros como receptores y gestores de la petición nos añadimos a la lista de amigos del usuario emisor.

Independientemente de que se acepte o se rechace la petición, ésta debe ser eliminada de la lista de peticiones pendientes del usuario receptor, puesto que ya está gestionada. Posteriormente se elimina de la base de datos para liberar espacio.

Una vez completado todo el proceso se actualiza el usuario con el DAO correspondiente y quedan todos los cambios guardados.

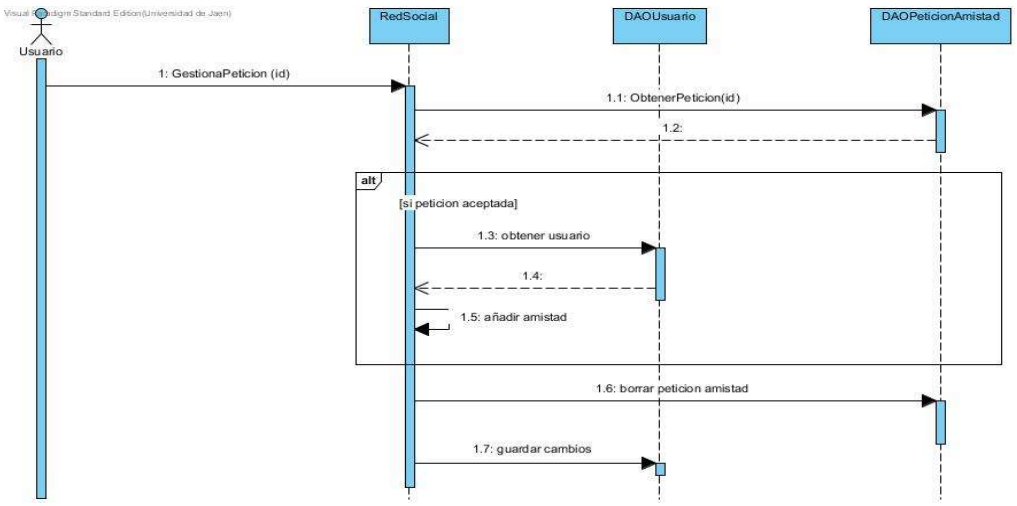

Ilustración 29: Diagrama de secuencia de gestionar petición de amistad pendiente

5.4.4. Lógica de negocio: Añadir nueva escuela

Como usuario puedo añadir una escuela cuando ésta no se encuentre disponible. Añadir una nueva escuela es una transacción sencilla que realiza nuestro objeto de negocio. Para ello, utiliza el DAO de dicha entidad para insertar un objeto de este tipo creado con los parámetros que previamente se ha indicado.

5.4.5. Lógica de negocio: Añadir nuevo sector

Ídem anterior pero con la entidad sector.

5.4.6. Lógica de negocio: Añadir nueva vía

Ídem anterior pero con la entidad vía.

5.4.7. Lógica de negocio: Comentar vía

Como usuario, puedo incluir un comentario con mi valoración y mi puntuación. Para comentar la vía hay que hacer dos pasos. En primer lugar obtener la vía a comentar mediante su identificador. En segundo lugar crear un comentario cuyo emisor es el usuario identificado en ese momento y asignárselo a la vía. Se añaden también los valores de valoración (nivel) y puntuación (estrellas) correspondientes al comentario. Posteriormente guardar tanto el comentario como la actualización de la vía y el usuario en persistencia mediante su DAO.

5.4.8. Cierre de la iteración 3

Se han desarrollado y presentado un total de 36 historias de usuario en esta tercera iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 297 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

5.5. Iteración 4: del 31/10/2016 al 11/11/2016

5.5.1. Análisis de la iteración 4

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Realizar vía: 34 puntos de historia
- Escuelas de una provincia: 8 puntos de historia

- Sectores de una escuela: 8 puntos de historia
- Vías de un sector: 8 puntos de historia

Total de puntos de historia seleccionados para la cuarta iteración: 58

5.5.2. Lógica de negocio: Realizar una vía. Diseño de las reglas de estimación colaborativa

Sin duda la historia de usuario más compleja y significativa hasta ahora que hay que desarrollar en nuestro *back-end,* ya que es el objetivo de nuestra aplicación. Necesitamos diseñar un algoritmo que nos devuelva un grado de dificultad de la vía consensuado en base a las opiniones y valoraciones de la comunidad de usuarios.

Para la puntuación (estrellas) simplemente es obtener una media aritmética de todas las valoraciones. Para el grado de dificultad se han tenido en cuenta los siguientes factores:

- La media de las 10 últimas vías de escalada completadas por el usuario.
- El nivel actual del escalador.
- El nivel consensuado actual de la vía. Inicialmente el nivel consensuado de la vía será el oficial.
- La fidelidad del usuario, que se medirá con el número de vías que el usuario ha valorado, así como premiando esa fidelidad incluyendo el ser más o menos conocido en la aplicación en base al número de amigos que tenga el usuario.

Descripción del algoritmo:

Cuando el usuario valora una vía se recalcula la media de puntuación en base al valor introducido. Eso simplemente es una media aritmética.

Para el cálculo del nivel consensuado, en primer lugar calcularemos el porcentaje de fiabilidad del usuario. Para ello vamos a tener el nivel medio de las últimas 10 vías que ha completado, y calculamos el porcentaje en base a los 30 niveles que hemos establecido, siendo el nivel 1 un 0% y el nivel 9c+ un 100%. Evidentemente cada nivel tiene un ordinal asociado con el fin de saber en qué nivel nos estamos moviendo para poder hacer todos los cálculos:

Nivel	Valor ordinal
1	0
2	1
3	2
4	3
5	4
5+	5
6a	6
6a+	7
6b	8
6b+	9
6c	10
6c+	11
7a	12
7a+	13
7b	14
7b+	15
7c	16
7c+	17
8a	18
8a+	19
8b	20
8b+	21
8c	22
8c+	23
9a	24
9a+	25
9b	26
9b+	27
9c	28
9c+	29

Tabla 9: Ordinal del nivel

El porcentaje de fiabilidad inicial queda de la siguiente forma:

porcentaje fiabilidad = nivel medio 10 ultimas vías (ordinal) x 0.29

Ese porcentaje puede verse incrementado hasta en un 20% extra en función del número de amigos:

Amigos	Extra %
Entre 0 y 10	0
Entre 10 y 20	5
Entre 20 y 50	10
Entre 50 y 100	15
Más de 100	20

Tabla 10: Porcentajes de incremento en base al número de amigos

El porcentaje puede verse incrementado hasta en otro 20% según el número de valoraciones:

Valoraciones	Extra %
Entre 0 y 10	0
Entre 10 y 20	5
Entre 20 y 50	10
Entre 50 y 100	15
Más de 100	20

Tabla 11: Porcentajes de incremento en base al número de valoraciones emitidas

Una vez calculado el porcentaje de fiabilidad en base a los criterios mencionados anteriormente es hora de calcular la media colaborativa (el nivel consensuado). En primer lugar hay que obtener la ponderación total de nuestra puntuación en base al número de opiniones que hay registradas sobre esa vía. Cuando tengamos la ponderación, habrá que reducir esa ponderación en base al porcentaje de fiabilidad, esto es:

ponderación = (1 / numero valoraciones vía) x (porcentaje fiabilidad / 100)

media colaborativa (ordinal) = media actual (ordinal) x (1 - ponderación) + (valoración usuario (ordinal) x ponderación)

El principal inconveniente del algoritmo es que la ponderación de cada opinión se reduce conforme van llegando opiniones nuevas. Teniendo en cuenta que lo ideal sería hacer una media ponderada de todas opiniones, eso implicaría guardar todos los valores de cada una de ellas en la base de datos, con la consecuente ineficiencia de acceso a persistencia para hacer los cálculos. Por eso es preferible guardar un valor actual y actuar sobre él con este criterio, que una vez tenemos un número de valoraciones considerable, resulta efectivo. Es por esto por lo que yo lo llamo una media pseudo-ponderada.

Una posible mejora que no se ha implementado pero sí se contempla es la de descartar los valores extremos, de forma que aún se ajuste mucho más el nivel consensuado que la comunidad de usuarios otorga a una vía.

5.5.3. Lógica de negocio: Obtener las escuelas de una provincia

Como usuario puedo ver las escuelas de escalada filtradas por provincias. Para la implementación de esta historia de usuario, se ha añadido un método extra en el DAO de escuela, que nos devuelve directamente la colección de escuelas filtradas por un código de provincia seleccionado previamente.

5.5.4. Lógica de negocio: Obtener los sectores de una escuela

Como usuario podré acceder a visualizar los sectores de una escuela. En este caso la implementación es sencilla, pues una escuela tiene una colección con los sectores que la componen.

5.5.5. Lógica de negocio: Obtener las vías de un sector

Ídem anterior pero con las vías.

5.5.6. Cierre de la iteración 4

Se han desarrollado y presentado un total de 58 puntos de historia en esta cuarta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Sin

embargo, en al final de esta iteración sí que hay cambios muy importantes que hay que tener en cuenta.

En primer lugar, eliminar las historias de *usuario editar comentario* y *eliminar comentario*, lo cual son 26 puntos de historia menos.

En segundo lugar, en la reunión de cierre de este *Sprint* se ha valorado la posibilidad de ampliar más allá nuestro *back-end*, de forma que podamos construir un servidor totalmente desacoplado para futuras implementaciones de distintos clientes, como por ejemplo *Android* ó *IOS*.

Analizando esta situación, nos damos cuenta de que hasta ahora llevamos hecha la lógica de negocio y la capa de persistencia, y que nos faltaría implementar la parte del *front-end* usando los controladores de *Spring web MVC* hasta ahora sin implementar.

Lo que ocurre a partir de ahora es que usaremos los controladores *Spring web MVC* para la creación de un API RESTFUL que pueda exportar nuestra lógica de negocio a cualquier tipo de cliente a través de servicios web mediante el protocolo HTTP. Es decir, nuestro controlador *Spring web MVC* en lugar de servir al *front-end* de una aplicación web, servirá a cualquier aplicación que se conecte a él a través de peticiones asíncronas.

Tenemos que tener en cuenta y no dejar de lado que aunque desacoplemos el servidor, nuestro producto final será una aplicación web, y por consiguiente hay que elegir una tecnología puntera que consuma de forma asíncrona los servicios web de nuestra API RESTFUL para un cliente web. Una excelente opción es implementar una aplicación web usando el lenguaje JavaScript puro, y más teniendo en cuenta la tendencia actual de las arquitecturas de las aplicaciones web actuales. En un principio el framework elegido para la elaboración de un *front-end* JavaScript puro será AngularJS [25], desarrollado y mantenido por Google.

Esto implica varias cosas, por supuesto. Una de ellas, un retraso notable en el proyecto, una reestimación de las tareas pendientes y un nuevo presupuesto no sólo teniendo en cuenta las tareas extra sino también teniendo en cuenta lo que el cliente ya ha depositado en base al presupuesto inicial.

Tras acordarlo todo con el cliente en esta reunión, acordamos una semana para la elaboración de un nuevo presupuesto y la reestimación del proyecto en base a lo acordado.

6. Reestimación del proyecto en base a nuevos requisitos

Hasta ahora la parte back-end que tenemos desarrollada corresponde con el modelo de datos, la capa de persistencia y la capa de negocio. Al haber cambios de requerimientos los controladores Spring web MVC se implementarán para la construcción de la interfaz RESTFUL, lo cual implica exportar toda nuestra funcionalidad implementada en la capa de negocio.

En primer lugar hay que detallar de nuevo las historias de usuario para completar este trabajo, posteriormente planificaremos de nuevo las iteraciones en base a la modificación de nuestro *Product backlog* y para finalizar elaboraremos un nuevo presupuesto teniendo en cuenta las características del presupuesto anterior. Finalmente en una nueva reunión con el cliente el 18 de Noviembre de 2016 y será quien dé el visto bueno para que nos pongamos manos a la obra.

6.1. Análisis de la velocidad de trabajo

En primer lugar analicemos si nuestra velocidad de trabajo viene siendo buena, para ver si la velocidad determinada al principio fue acertada.

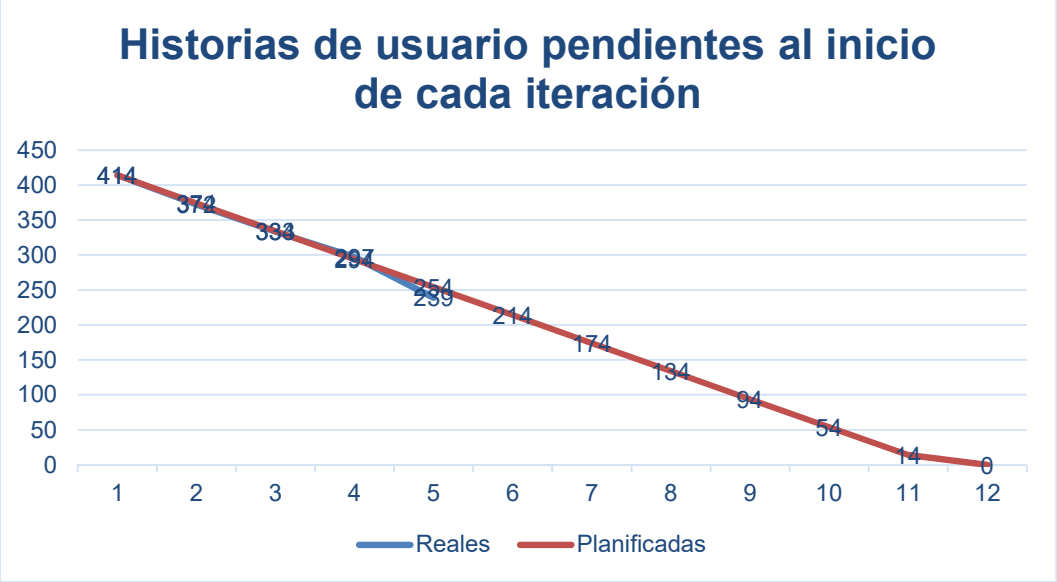

Ilustración 30: Diagrama de quemado que representa la velocidad de trabajo

Podemos observar en el diagrama de quemado que no solamente la velocidad de trabajo es acertada, sino que mejoramos ligeramente los plazos planificados, por lo tanto es algo que no cambia.

6.2. Reestimación de las nuevas historias de usuario

De nuevo tendremos en cuenta las historias de usuario divididas en *back-end* y *front-end*, teniendo en cuenta que nuestro *front-end* será una aplicación JavaScript pura usando *AngularJS* que consumirá de forma asíncrona los recursos prestados por nuestra API RESTFUL.

Historias de usuario *back-end*	Puntos de historia
Solicitud acceso	5
Alta usuario	5
Baja usuario	5
Actualizar usuario	8
Cambio password usuario	8
Datos de perfil	5
Datos de otro perfil	5
Mis amigos	5
Amigos de otro perfil	5
Mis vias realizadas	5
Vias de otro perfil	5
Nº de peticiones pendientes	5
Peticiones pendientes	5
Solicitar amistad	8
Gestionar amistad	8
Obtener usuario conectado	8
Nueva escuela	5
Nuevo sector	5

Nueva vía	5
Escuelas de una provincia	5
Sectores de una escuela	5
Vías de un sector	5
Comentarios vía	5
Realizar vía	13
Obtener provincia	5
Datos escuela	5
Datos sector	5
Datos vía	5
Configuración autenticación	13

Tabla 12: Nuevas historias de usuario back-end

Historias de usuario *front-end*	Puntos de historia
Página de portada	3
Formulario de inicio de sesión	5
Barra de navegación de páginas	8
Página de perfil	21
Formulario de actualizar perfil de usuario	8
Formulario de cambio de contraseña	8
Selector de búsqueda (buscador)	5
Buscador de usuarios	8
Buscador de escuelas	8
Ficha de escuela	13
Ficha de sector	13

Ficha de vía y valoraciones	21
Formulario para valoración de vías	13
Formulario para añadir escuela	8
Formulario para añadir sector	8
Formulario para añadir vía	8

Tabla 12: Nuevas historias de usuario front-end

Historias de usuario documentación y presentación	Puntos de historia
Documentación de la memoria de proyecto	55
Preparación de la presentación final	13

Tabla 13: Historias de usuario pertenecientes al desarrollo de la documentación y preparación de la presentación final

Resumen de puntos de historia:

Historias de usuario	Total puntos de historia
Desarrollados hasta ahora	175
Pertenecientes al back-end	176
Pertenecientes al front-end	158
Pertenecientes al desarrollo de la documentación y presentación final	68

TOTAL PUNTOS DE HISTORIA	577

Tabla 14: Historias de usuario pertenecientes al desarrollo de la documentación y preparación de la presentación final

6.3. Nueva planificación

En un principio establecimos que seríamos capaces de trabajar a 40 puntos de historia por iteración, y recientemente hemos comprobado que efectivamente se trató de una decisión correcta y por lo tanto la velocidad de trabajo la íbamos a mantener en la nueva estimación. Teniendo en cuenta un total de 402 puntos de historia que quedan por desarrollar (tenemos ya desarrollados 175 de los 577), suman un total de 10 iteraciones.

Período de vacaciones a tener en cuenta:

- Navidad: Del 19 de Diciembre de 2016 al 8 de Enero de 2017
- Semana extra de vacaciones: Del 27 de Febrero de 2017 al 6 de Marzo de 2017
- Semana Santa: Del 10 de Abril de 2017 al 17 de Abril de 2017

Iteración	Plazo de ejecución
Iteración 5	**21/11/2016 – 02/12/2016**
Iteración 6	**05/12/2016 – 16/12/2016**
Iteración 7	**09/01/2017 – 20/01/2017**
Iteración 8	**06/02/2017 – 17/02/2017**
Iteración 9	**20/02/2017 – 10/03/2017**
Iteración 10	**13/03/2017 – 24/03/2017**
Iteración 11	**27/03/2016 – 07/04/2017**
Iteración 12	**17/04/2017 – 28/04/2017**
Iteración 13	**01/05/2017 – 12/05/2017**
Iteración 14	**15/05/2017 – 26/05/2017**

Tabla 15: Nueva planificación

Teniendo en cuenta que la entrega final es a mediados del mes de Junio, aún seguimos teniendo un mes de margen para posibles imprevistos y/o cambios en la especificación de requisitos.

6.4. Nuevo presupuesto

En este caso, son un total de 14 iteraciones para desarrollar 577 puntos de historia. Tenemos en cuenta que las iteraciones son de dos semanas (10 días laborables) a 7 horas de trabajo, y que suman un total de 980 horas laborables. Si dividimos 980 horas laborables entre 577 puntos de historia determinamos que tardaremos aproximadamente 1.69 horas. Al igual que la otra vez emplearemos 2 horas de trabajo por punto de historia, garantizándonos así un cierto margen de maniobra:

Salario analista/programador	Precio/hora	Total de horas	Total
30.000 € anuales	**16,23 €/hora**	**1154 horas**	**18.729,42 €**

Tabla 15: Tabla con la descripción de los costes en salarios del nuevo presupuesto

Aquí incluimos la amortización del equipo informático necesario.

Equipo informático	Precio del equipo	Vida útil del equipo	Duración del proyecto	Total
Portátil Acer Aspire ES1-571-507Y, i5 4GB, 500GB	**600 €**	**5 años**	**9 meses**	**90 €**

Tabla 16: Tabla con la descripción de los costes en salarios del nuevo presupuesto

En esta última tabla se resumen todos los conceptos del presupuesto del proyecto.

Concepto	Importe
Costes salariales	**18.729,42 €**
Equipo informático	**90 €**
Dominio web y hosting 1 año [23]	**76 €**
Costes totales…	**18.895,42 €**
Beneficio sobre costes 20 %	**3.779,09 €**
Base imponible	**22.674,50 €**
Impuestos I.V.A. 21 %	**4.761,64 €**
Importe total del proyecto	**27.436,15 €**

Tabla 17: Desglose de conceptos del nuevo presupuesto del proyecto

El cliente deberá abonar un 25% del total del presupuesto antes del inicio del proyecto a modo de fianza, y el resto se abonará a la finalización del proyecto. A la cantidad que debe abonar le restamos la cuantía ya abonada en el anterior presupuesto, que fue de 4.944,15 €, por lo que el cliente deberá abonar únicamente la cantidad de 1.914,89 € para iniciar el trabajo.

7. Diseño e implementación del *back-end:* Capa de servicios. API RESTFUL para exportación de servicios web

7.1. Iteración 5: del 21/11/2016 al 02/12/2016

7.1.1. Análisis de la iteración 5

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Solicitud de acceso: 5 puntos de historia
- Alta usuario: 5 puntos de historia
- Baja de usuario: 5 puntos de historia
- Actualizar usuario: 8 puntos de historia

- Cambio *password* usuario: 8 puntos de historia
- Datos de perfil: 5 puntos de historia
- Datos de otro perfil: 5 puntos de historia

Total de puntos de historia seleccionados para la quinta iteración: 41

NOTA: Aquellos servicios que consuman o produzcan DTO (objetos de transferencia de datos), se especificarán cuáles son sus atributos utilizados necesarios para poder utilizar el API.

7.1.2. Servicio web REST: solicitud de acceso

Con este servicio web estaremos solicitando al servidor el registro en la aplicación. Si cualquiera de los campos no son confirmados (el email o la contraseña) lanzará un código 400 (BAD REQUEST).

URL	/perfil/acceso
Método HTTP	**POST**
Autenticación necesaria	**-**
Consume	**application/json**
Produce	**-**
RequestBody	**NuevoUsuarioDTO**
ResponseBody	**-**
PathVariable	**-**

Tabla 18: REST: solicitud de acceso

NuevoUsuarioDTO:

```
public class NuevoUsuarioDTO
{
private String username;
private String password;
private String confPassword;
private String mail;
private String confMail;
```

Ilustración 31: Implementación de NuevoUsuarioDTO

7.1.3. Servicio web REST: alta usuario

Servicio web que captura el token enviado al cliente a través de la URL y valida el registro.

URL	/confirmacion/{token}
Método HTTP	POST
Autenticación necesaria	-
Consume	-
Produce	-
RequestBody	-
ResponseBody	-
PathVariable	token

Tabla 19: REST: alta usuario

7.1.4. Servicio web REST: baja de usuario

Servicio web que tramita la baja de usuario de la aplicación.

URL	/perfil
Método HTTP	DELETE
Autenticación necesaria	Básica HTTPS
Consume	-
Produce	-
RequestBody	-
ResponseBody	-
PathVariable	-

Tabla 20: REST: baja de usuario

7.1.5. Servicio web REST: actualizar usuario

Servicio web con el que se actualiza el perfil de usuario.

URL	/perfil
Método HTTP	**PUT**
Autenticación necesaria	**Básica HTTPS**
Consume	**application/json**
Produce	**-**
RequestBody	**ActualizarUsuarioDTO**
ResponseBody	**-**
PathVariable	**-**

Tabla 21: REST: actualizar usuario

ActualizarUsuarioDTO:

```
public class ActualizarUsuarioDTO
{
    private String nombre;
    private String apellidos;
    private String mail;
    private String confMail;
    private String _foto; (cadena en base64)
```

Ilustración 32: Implementación de ActualizarUsuarioDTO

7.1.6. Servicio web REST: cambiar *password*

Servicio web con el que se puede cambiar la contraseña de acceso a la aplicación. Si la contraseña actual no fuera introducida lanzaría un código 409 (CONFLICT). Si la contraseña introducida no fuera correcta lanzaría un código 401 (UNAUTHORIZED). Si la nueva contraseña no fuera confirmada se lanzaría un código 400 (BAD REQUEST).

URL	/perfil/password
Método HTTP	**PUT**
Autenticación necesaria	**Básica HTTPS**

Consume	application/json
Produce	-
RequestBody	NewPasswordDTO
ResponseBody	-
PathVariable	-

Tabla 20: REST: cambiar password

NewPasswordDTO:

```
public class NewPasswordDTO
{
    private String passwordActual;
    private String newPassword;
    private String confPassword;
```

Ilustración 33: NewPasswordDTO

7.1.7. Servicio web REST: datos de perfil

Servicio web con el que se obtienen los datos del perfil de usuario.

URL	/perfil
Método HTTP	GET
Autenticación necesaria	Básica HTTPS
Consume	-
Produce	application/json
RequestBody	-
ResponseBody	PerfilDTO
PathVariable	-

Tabla 21: REST: datos de perfil

PerfilDTO:

```
public class PerfilDTO
{
    private String username;
    private String nombre;
    private String apellidos;
    private String nivel;
    private String mail;
    private byte[] foto;
```

Ilustración 33: Implementación PerfilDTO

7.1.8. Servicio web REST: datos de otro perfil

Servicio web con el que se obtienen los datos del perfil de otro usuario.

URL	/perfil/{username}
Método HTTP	**GET**
Autenticación necesaria	**-**
Consume	**-**
Produce	**application/json**
RequestBody	**-**
ResponseBody	**PerfilDTO**
PathVariable	**username**

Tabla 22: REST: datos de otro perfil

NOTA: El objeto utilizado en la respuesta de la petición está anteriormente descrito: se trata igualmente de PerfilDTO.

7.1.9. Cierre de la iteración 5

Se han desarrollado y presentado un total de 41 historias de usuario en esta quinta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 361 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

7.2. Iteración 6: del 05/12/2016 al 16/12/2016

7.2.1. Análisis de la iteración 6

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Mis amigos: 5 puntos de historia
- Amigos de otro perfil: 5 puntos de historia
- Mis vías realizadas: 5 puntos de historia
- Vías de otro perfil: 5 puntos de historia
- Nº peticiones amistad pendientes: 5 puntos de historia
- Peticiones pendientes: 5 puntos de historia
- Solicitar amistad: 8 puntos de historia

Total de puntos de historia seleccionados para la sexta iteración: 38

NOTA: Aquellos servicios que consuman o produzcan DTO (objetos de transferencia de datos), se especificarán cuáles son sus atributos utilizados necesarios para poder utilizar el API.

7.2.2. Servicio web REST: mis amigos

Servicio web con el que se obtienen los amigos que tengo agregados

URL	/perfil/amigos
Método HTTP	GET
Autenticación necesaria	Básica HTTPS
Consume	-
Produce	application/json
RequestBody	-
ResponseBody	AmigosDTO
PathVariable	-

Tabla 23: REST: mis amigos

AmigosDTO:

```
public class AmigosDTO
{
    private String username;
    private String nombre;
```

Ilustración 34: Implementación de AmigosDTO

7.2.3. Servicio web REST: amigos de otro perfil

Servicio web con el que se obtienen los amigos de otro usuario a través de su username.

URL	/perfil/{username}/amigos
Método HTTP	GET
Autenticación necesaria	-
Consume	-
Produce	application/json
RequestBody	-
ResponseBody	AmigosDTO
PathVariable	username

Tabla 24: REST: amigos de otro perfil

NOTA: El objeto utilizado en la respuesta de la petición está anteriormente descrito: se trata igualmente de AmigosDTO.

7.2.4. Servicio web REST: mis vías realizadas

Servicio web con el que se obtienen mis vías realizadas.

URL	/perfil/vias
Método HTTP	GET
Autenticación necesaria	Básica HTTPS
Consume	-
Produce	application/json

RequestBody	**-**
ResponseBody	**ViasDTO**
PathVariable	**-**

Tabla 25: REST: mis vías realizadas

ViasDTO:

```
 public class ViasDTO
{
    private Integer id_via;
    private String id_mapa;
    private String nombre;
    private String sector;
    private String escuela;
    private String nivel_oficial;
    private String nivel_consensuado;
    private Integer contador;
    private Integer estrellas;
    private String provincia;
```

Ilustración 35: Implementación de ViasDTO

7.2.5. Servicio web REST: vías de otro perfil

Servicio web con el que se obtienen las vías realizadas por un usuario.

URL	**/perfil/{username}/vias**
Método HTTP	**GET**
Autenticación necesaria	**-**
Consume	**-**
Produce	**application/json**
RequestBody	**-**
ResponseBody	**ViasDTO**
PathVariable	**username**

Tabla 26: REST: vías de otro perfil

NOTA: El objeto utilizado en la respuesta de la petición está anteriormente descrito: se trata igualmente de ViasDTO.

7.2.6. Servicio web REST: nº peticiones de amistad pendientes

Servicio web con el que se obtienen el número de peticiones de amistad pendientes.

URL	/perfil/numeropeticiones
Método HTTP	GET
Autenticación necesaria	Básica HTTPS
Consume	-
Produce	application/json
RequestBody	-
ResponseBody	NumeroPeticionesDTO
PathVariable	-

Tabla 26: REST: nº peticiones de amistad pendientes

NumeroPeticionesDTO:

```
public class NumeroPeticionesDTO
{
    private Integer numeroPendientes;
```

Ilustración 36: Implementación de NumeroPeticionesDTO

7.2.7. Servicio web REST: peticiones de amistad pendientes

Servicio web con el que se obtiene una lista con las peticiones de amistad pendientes.

URL	/perfil/peticiones
Método HTTP	GET
Autenticación necesaria	Básica HTTPS
Consume	-
Produce	application/json

RequestBody	**-**
ResponseBody	**PeticionDTO**
PathVariable	**-**

Tabla 27: REST: peticiones de amistad pendientes

PeticionDTO:

```
public class PeticionDTO
{
    private Integer idPeticion;
    private String emisor;
```

Ilustración 36: Implementación de PeticionDTO

7.2.8. Servicio web REST: solicitar amistad

Servicio web con el que se envía una petición de amistad a otro usuario.

URL	**/perfil/peticiones**
Método HTTP	**POST**
Autenticación necesaria	**Básica HTTPS**
Consume	**application/json**
Produce	**-**
RequestBody	**UsernameDTO**
ResponseBody	**-**
PathVariable	**-**

Tabla 27: REST: solicitar amistad

UsernameDTO:

```
public class UsernameDTO
{
    private String username;
```

Ilustración 36: Implementación UsernameDTO

7.2.9. Cierre de la iteración 6

Se han desarrollado y presentado un total de 38 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 323 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

7.3. Iteración 7: del 09/01/2017 al 20/01/2017

7.3.1. Análisis de la iteración 7

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Gestionar petición de amistad: 8 puntos de historia
- Obtener usuario conectado: 8 puntos de historia
- Nueva escuela: 5 puntos de historia
- Nuevo sector: 5 puntos de historia
- Nueva vía: 5 puntos de historia
- Escuelas de una provincia: 5 puntos de historia
- Sectores de una escuela: 5 puntos de historia

Total de puntos de historia seleccionados para la séptima iteración: 41

NOTA: Aquellos servicios que consuman o produzcan DTO (objetos de transferencia de datos), se especificarán cuáles son sus atributos utilizados necesarios para poder utilizar el API.

7.3.2. Servicio web REST: gestionar petición de amistad

Servicio web con el que se gestiona una petición de amistad procedente de otro usuario.

URL	/perfil/peticiones
Método HTTP	**PUT**
Autenticación necesaria	**Básica HTTPS**
Consume	**application/json**

Produce	**-**
RequestBody	**GestionarPeticionDTO**
ResponseBody	**-**
PathVariable	**-**

Tabla 28: REST: gestionar petición de amistad

GestionarPeticionDTO:

```
public class GestionarPeticionDTO
{
    private Integer idPeticion;
    private boolean conf;
```

Ilustración 37: Implementación de la entidad Usuario

7.3.3. Servicio web REST: obtener usuario conectado

Servicio web con el que se obtiene el usuario que está autenticado.

URL	**/username**
Método HTTP	**GET**
Autenticación necesaria	**Básica HTTPS**
Consume	**-**
Produce	**application/json**
RequestBody	**-**
ResponseBody	**UsernameDTO**
PathVariable	**-**

Tabla 29: REST: obtener usuario conectado

En este caso resulta interesante ver cómo se accede al API de Spring Security para obtener acceso al usuario autenticado. Cabe destacar que esta es la forma con la que se implementa la asignación de un usuario conectado a cualquier servicio que así lo requiera para personalizar los recursos de la capa de negocio.

```
String usernameConectado = null;
        Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();

        if (principal instanceof UserDetails) {
            usernameConectado = ((UserDetails) principal).getUsername();
        }

        if (usernameConectado == null) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }
```

Ilustración 38: identificación de usuario conectado con Spring Security

7.3.4. Servicio web REST: nueva escuela

Servicio web con el que se registra una nueva escuela en el sistema.

URL	/escuelas
Método HTTP	**POST**
Autenticación necesaria	**-**
Consume	**application/json**
Produce	**-**
RequestBody	**EscuelaDTO**
ResponseBody	**-**
PathVariable	**-**

Tabla 30: REST: nueva escuela

EscuelaDTO:

```
public class EscuelaDTO
{
    private String nombre;
    private String descripcion;
    private String dir_foto;
    private String horario;
    private Integer cod_provincia;;
```

Ilustración 38: Implementación de EscuelaDTO

7.3.5. Servicio web REST: nuevo sector

Servicio web con el que se registra un nuevo sector en una escuela.

URL	/escuelas/{id_escuela}/sectores
Método HTTP	POST
Autenticación necesaria	-
Consume	application/json
Produce	-
RequestBody	SectorDTO
ResponseBody	-
PathVariable	Id_escuela

Tabla 31: REST: nuevo sector

SectorDTO:

```
public class SectorDTO
{
    private String orientacion;
    private String nombre;
    private String dir_foto;
```

Ilustración 39: Implementación de SectorDTO

7.3.6. Servicio web REST: nueva vía

Servicio web con el que se registra una nueva vía dentro de un sector.

URL	/sectores/{id_sector}/vias
Método HTTP	POST
Autenticación necesaria	-
Consume	application/json
Produce	-
RequestBody	NuevaViaDTO

ResponseBody	**-**
PathVariable	**Id_sector**

Tabla 32: REST: nueva vía

NuevaViaDTO:

```
public class NuevaViaDTO
{
    private String nombre;
    private String id_mapa;
    private String nivel_oficial;
```

Ilustración 40: Implementación de NuevaViaDTO

7.3.7. Servicio web REST: escuelas de una provincia

Servicio web con el que se obtiene una lista con las escuelas de una provincia.

URL	**/escuelas/{cod_prov}**
Método HTTP	**GET**
Autenticación necesaria	**-**
Consume	**-**
Produce	**application/json**
RequestBody	**-**
ResponseBody	**EscuelasDTO**
PathVariable	**Cod_provincia**

Tabla 33: REST: escuelas de una provincia

EscuelasDTO:

```
public class EscuelasDTO
{
    private Integer id_escuela;
    private String nombre;
    private String descripcion;
    private byte[] foto;
    private String horario;
    private String provincia;
```

Ilustración 41: Implementación de EscuelasDTO

7.3.8. Servicio web REST: sectores de una escuela

Servicio web con el que se obtiene una lista con los sectores de una escuela.

URL	/sectores/{cod_escuela}
Método HTTP	**GET**
Autenticación necesaria	**-**
Consume	**-**
Produce	**application/json**
RequestBody	**-**
ResponseBody	**SectoresDTO**
PathVariable	**Cod_escuela**

Tabla 34: REST: sectores de una escuela

SectoresDTO:

```
public class SectoresDTO
{
    private Integer id_sector;
    private String orientacion;
    private String nombre;
    private String escuela;
    private byte[] foto;
```

Ilustración 42: Implementación de SectoresDTO

7.3.9. Cierre de la iteración 7

Se han desarrollado y presentado un total de 41 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 282 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

7.4. Iteración 8: del 06/02/2017 al 17/02/2017

7.4.1. Análisis de la iteración 8

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Vías de un sector: 5 puntos de historia
- Añadir comentario a una una vía: 5 puntos de historia
- Realizar, valorar y puntuar una vía: 13 puntos de historia
- Obtener provincia: 5 puntos de historia
- Datos escuela: 5 puntos de historia
- Datos sector: 5 puntos de historia
- Datos vía: 5 puntos de historia
- Configuración de la autenticación y seguridad de la aplicación: 13 puntos de historia

Total de puntos de historia seleccionados para la octava iteración: 56

NOTA: Aquellos servicios que consuman o produzcan DTO (objetos de transferencia de datos), se especificarán cuáles son sus atributos utilizados necesarios para poder utilizar el API.

7.4.2. Servicio web REST: vías de un sector

Servicio web con el que se obtiene una lista con las vías de un sector.

URL	**/vias/{cod_sector}**
Método HTTP	**GET**
Autenticación necesaria	**-**
Consume	**-**

Produce	**application/json**
RequestBody	**-**
ResponseBody	**ViasDTO**
PathVariable	**Cod_sector**

Tabla 35: REST: vías de un sector

VíasDTO:

```
public class ViasDTO
{
    private Integer id_via;
    private String id_mapa;
    private String nombre;
    private String sector;
    private String escuela;
    private String nivel_oficial;
    private String nivel_consensuado;
    private Integer contador;
    private Integer estrellas;
    private String provincia;
```

Ilustración 43: Implementación de ViasDTO

7.4.3. Servicio web REST: añadir un comentario a una vía

Servicio web con el que se puede comentar, puntuar y valorar una vía.

URL	**/comentarios/{cod_via}**
Método HTTP	**POST**
Autenticación necesaria	**HTTPS**
Consume	**application/json**
Produce	**-**
RequestBody	**ComentarioDTO**
ResponseBody	**-**
PathVariable	**Cod_via**

Tabla 36: REST: añadir comentario a una vía

ComentarioDTO:

```
public class ComentarioDTO
{
    private String valor_comentario;
    private Integer puntuacion;
    private String valoracion;
```

Ilustración 44: Implementación de ComentarioDTO

Únicamente este servicio web captura la valoración y puntuación para luego poder mostrarlas al usuario. Para procesarlas está el servicio web del siguiente apartado, que será explicado a continuación. Para devolver los comentarios de una vía (con puntuaciones y valoraciones) está el siguiente servicio web:

URL	/comentarios/{cod_via}
Método HTTP	GET
Autenticación necesaria	HTTPS
Consume	-
Produce	application/json
RequestBody	-
ResponseBody	ComentariosDTO
PathVariable	Cod_via

Tabla 37: REST: obtener comentarios de una vía

ComentariosDTO:

```
public class ComentariosDTO
{
    private String valor_comentario;
    private String username;
    private Integer puntuacion;
    private String valoracion;
    private String fecha;
    private String hora;
```

Ilustración 45: Implementación de ComentariosDTO

7.4.4. Servicio web REST: realizar, valorar y puntuar una vía

Servicio web con el que se valora, puntua una vía. Al hacer esto la vía pasa a nuestro diario de escalada (porque se supone que la hemos hecho).

URL	/perfil/vias/{id_via}
Método HTTP	POST
Autenticación necesaria	HTTPS
Consume	application/json
Produce	-
RequestBody	ValorarViaDTO
ResponseBody	-
PathVariable	Id_via

Tabla 38: REST: realizar, valorar y puntuar una vía

ValorarViaDTO:

```
public class ValorarViaDTO
{
    private String nivel;
    private Integer valoracion;
```

Ilustración 46: Implementación de ValorarViaDTO

7.4.5. Servicio web REST: obtener provincia

Servicio web con el que se obtiene la provincia dado su código.

URL	/provincia/{cod_provincia}
Método HTTP	GET
Autenticación necesaria	-
Consume	-
Produce	application/json
RequestBody	-

ResponseBody	**ProvinciaDTO**
PathVariable	**Cod_provincia**

Tabla 39: REST: obtener provincia

ProvinciaDTO:

```
public class ProvinciaDTO
{
    private String provincia;
```

Ilustración 47: Implementación de ProvinciaDTO

7.4.6. Servicio web REST: datos escuela

Servicio web con el que se obtienen los datos de una escuela.

URL	**/escuela/{cod_escuela}**
Método HTTP	**GET**
Autenticación necesaria	**-**
Consume	**-**
Produce	**application/json**
RequestBody	**-**
ResponseBody	**EscuelasDTO**
PathVariable	**Cod_escuela**

Tabla 40: REST: datos escuela

EscuelasDTO:

```
public class EscuelasDTO
{
    private Integer id_escuela;
    private String nombre;
    private String descripcion;
    private byte[] foto;
    private String horario;
    private String provincia;
```

Ilustración 48: Implementación de EscuelasDTO

7.4.7. Servicio web REST: datos sector

Servicio web con el que se obtiene una lista con las escuelas de una provincia.

URL	/sector/{cod_sector}
Método HTTP	GET
Autenticación necesaria	-
Consume	-
Produce	application/json
RequestBody	-
ResponseBody	SectoresDTO
PathVariable	Cod_sector

Tabla 41: REST: datos sector

SectoresDTO:

```
public class SectoresDTO
{
    private Integer id_sector;
    private String orientacion;
    private String nombre;
    private String escuela;
    private byte[] foto;
```

Ilustración 49: Implementación de SectoresDTO

7.4.8. Servicio web REST: datos vía

Servicio web con el que se obtiene una lista con las escuelas de una provincia.

URL	/via/{cod_via}
Método HTTP	GET
Autenticación necesaria	-
Consume	-
Produce	application/json
RequestBody	-

ResponseBody	ViasDTO
PathVariable	Cod_via

Tabla 42: REST: datos vía

ViasDTO:

```
public class ViasDTO
{
    private Integer id_via;
    private String id_mapa;
    private String nombre;
    private String sector;
    private String escuela;
    private String nivel_oficial;
    private String nivel_consensuado;
    private Integer contador;
    private Integer estrellas;
    private String provincia;
```

Ilustración 50: Implementación de VíasDTO

7.4.9. Configuración de la autenticación y Seguridad de la aplicación

La autenticación es la parte de la aplicación que permite detectar qué usuario es el que está accediendo a un determinado recurso mediante el uso de credenciales de acceso (*username* y *password*).

La autenticación HTTP básica ha sido uno de los métodos estudiados para controlar el acceso de usuarios a la aplicación. Este tipo de autenticación, tal y como su propio nombre indica, es la forma más básica de lograr una comunicación cliente-servidor mediante credenciales de acceso. Éstas viajan a través de la red a usando la cabecera “Authorization” codificadas en base64 con el formato *usuario:clave*. Esto puede suponer una gran inseguridad, puesto que un atacante que se interponga en la red entre el cliente y el servidor puede descifrar fácilmente el usuario y la contraseña, ya que lo único que tiene que hacer es descodificar la cadena.

La autenticación Digest soluciona en parte este problema. En este caso, utiliza un algoritmo de encriptación MD5 para codificar la información que se envía al servidor. A priori puede parecer una solución bastante buena pero ahora veremos que no es así.

No solamente hay que tener en cuenta el tipo de autenticación elegida, sino cómo se guardan las contraseñas en la base de datos, ya que si un usuario consigue acceder a nuestro servidor y localizar la tabla donde estén las contraseñas, estaríamos inmersos en un serio problema. Entonces la solución parece clara, codificar tanto la información que se envía al servidor como las contraseñas en la base de datos.

Sin embargo, tras investigar varias fuentes de información [26] [27], éstas me confirman que la autenticación Digest no es compatible con guardar las claves codificadas en la base de datos, cosa que por ejemplo sí lo permite la autenticación HTTP básica.

Esa semana confirmo varias visitas a varios profesores del departamento para explicar mi problema, como son el Prof. D. Manuel José Lucena López y el Prof. D. José Ramón Balsas Almagro. Éste último me propone lo que venía suponiendo tras la lectura de las distintas fuentes de información, y no es más que la creación de un certificado SSL auto-firmado [28], de forma que no solamente garantizo que la información viaje cifrada de cliente a servidor (algoritmo SHA-256), sino que también me permite codificar las contraseñas en la base de datos con un algoritmo bastante potente y muy recomendado para tal fin: el algoritmo Brcypt [29]. Es por esto por lo que el protocolo que utilizan los servicios web que requieren de autenticación es HTTP**S.**

Una vez tratado y solucionado este aspecto, me dispongo a configurar con autenticación básica las peticiones de mi servidor web, protegiendo las URL que así sean necesarias e indicando los parámetros que indican cuales son las fuentes de datos para realizar la autenticación. Para la configuración es necesario la creación de un fichero *.xml* en la configuración de *Spring* y luego añadírselo al *web.xml*, que previamente debe ser configurado con el filtro que utiliza *Spring Security*.

```
<?xml version="1.0" encoding="UTF-8"?>
<b:beans xmlns="http://www.springframework.org/schema/security"
         xmlns:b="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/security
            http://www.springframework.org/schema/security/spring-
security.xsd">

    <http auto-config="true">
        <csrf disabled="true" />

        <intercept-url pattern="/username" access="hasRole('USER')"
requires-channel="https"/>
        <intercept-url pattern="/perfil/**" method="GET"
access="hasRole('USER')" requires-channel="https"/>
        <intercept-url pattern="/comentarios/**"
access="hasRole('USER')" requires-channel="https"/>
        <logout logout-url="/logout" />
        <form-login  authentication-failure-url="/error"/>
    </http>

    <authentication-manager erase-credentials="false">
        <authentication-provider user-service-ref="database" >
            <password-encoder hash="bcrypt"/>
        </authentication-provider>
    </authentication-manager>

    <jdbc-user-service id="database" data-source-ref="dataSource"
    users-by-username-query="select username, password, true from
usuario where username=?"
    authorities-by-username-query="select username, role from usuario
where username=?" />

</b:beans>
```

Ilustración 51: Fichero de configuración de Spring Security

A continuación se detallará la implementación del fichero *web.xml* mencionado anteriormente no sin antes hablar de CORS [30], otro de los mayores quebraderos de cabeza de este apartado.

Realizando pruebas mediante llamadas AJAX me doy cuenta de que las llamadas a mi servidor no se realizan correctamente, devolviendo éstas un código 404 (NOT FOUND). Sin embargo, no tengo ese problema en mi RESTCLIENT donde funcionan a la perfección.

De nuevo confirmo varias visitas tanto con mi tutor el Prof. D. Antonio Jesús Rueda Ruiz como con el Prof. D. José Ramón Balsas Almagro y les vuelvo a

explicar mi problema. Anteriormente buscando en varias fuentes doy con lo que es la base del problema, y es la llamada política del mismo origen, la cual no te permite consumir recursos cuyo origen no sea el mismo con el que el navegador esté actuando, de ahí que el fallo sea del cliente y no del servidor. [31]

Por tanto, la solución a esto es configurar un filtro CORS [32] que nos permita adaptar las características de exportación del servidor, tanto para las cabeceras como para los métodos HTTP utilizados.

```
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>CorsFilter</filter-name>
        <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
        <init-param>
            <param-name>cors.allowed.headers</param-name>
            <param-value>Content-Type,X-Requested-With,accept,Origin,Access-Control-
Request-Method,Access-Control-Request-Headers,Authorization</param-value>
        </init-param>
        <init-param>
            <param-name>cors.allowed.origins</param-name>
            <param-value>*</param-value>
        </init-param>
        <init-param>
            <param-name>cors.allowed.methods</param-name>
            <param-value>GET,POST,HEAD,OPTIONS,PUT,DELETE</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>CorsFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Ilustración 52: Fragmento del fichero web.xml donde se especifica el filtro de Spring Security y Filtro CORS

7.4.10. Cierre de la iteración 8

Se han desarrollado y presentado un total de 56 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 226 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

8. Diseño e implementación del *front-end*

Se denomina *front-end* a la parte de implementación de la aplicación en la que se produce la interacción usuario-ordenador. Comúnmente es llamada programación en el lado del cliente y no es más que la implementación de la interfaz de usuario y su comunicación con el *back-end*.

En nuestro caso emplearemos la tecnología JavaScript pura realizando llamadas asíncronas a nuestro servidor utilizando el framework AngularJS de Google.

8.1. Iteración 9: del 20/02/2017 al 10/03/2017

8.1.1. Análisis de la iteración 9

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Página de portada: 3 puntos de historia
- Formulario de inicio de sesión: 5 puntos de historia
- Barra de navegación de páginas: 8 puntos de historia
- Página de perfil: 21 puntos de historia
- Formulario de actualizar perfil de usuario: 8 puntos de historia

Total de puntos de historia seleccionados para la novena iteración: 45

8.1.2. Página de portada

Simplemente se trata de una página de portada en la que aparecen dos secciones. La primera con dos botones, uno para registrarse y otro para iniciar sesión. Se incluye también una sección con el enlace a Play Store y App Store para cuando tengamos disponibles nuestras versiones de la aplicación para dispositivos móviles. Se añade también un pequeño botón que permite movernos por las dos secciones.

8.1.3. Formulario de inicio de sesión

Cuando en la página de portada pulsamos sobre el botón de iniciar sesión, nos redirige a este formulario en el que introducimos los credenciales de acceso a la aplicación.

AngularJS captura los valores de los credenciales y los traspasa al Scope. Con estos valores se realiza una llamada al servicio web GET */username* y comprueba la autenticación e identificación de esas credenciales. En caso de error muestra el mensaje correspondiente. Si se produce de forma satisfactoria se crea la cookie de sesión que identifica al usuario cada vez que se utilicen servicios web que necesiten autenticación en esa sesión.

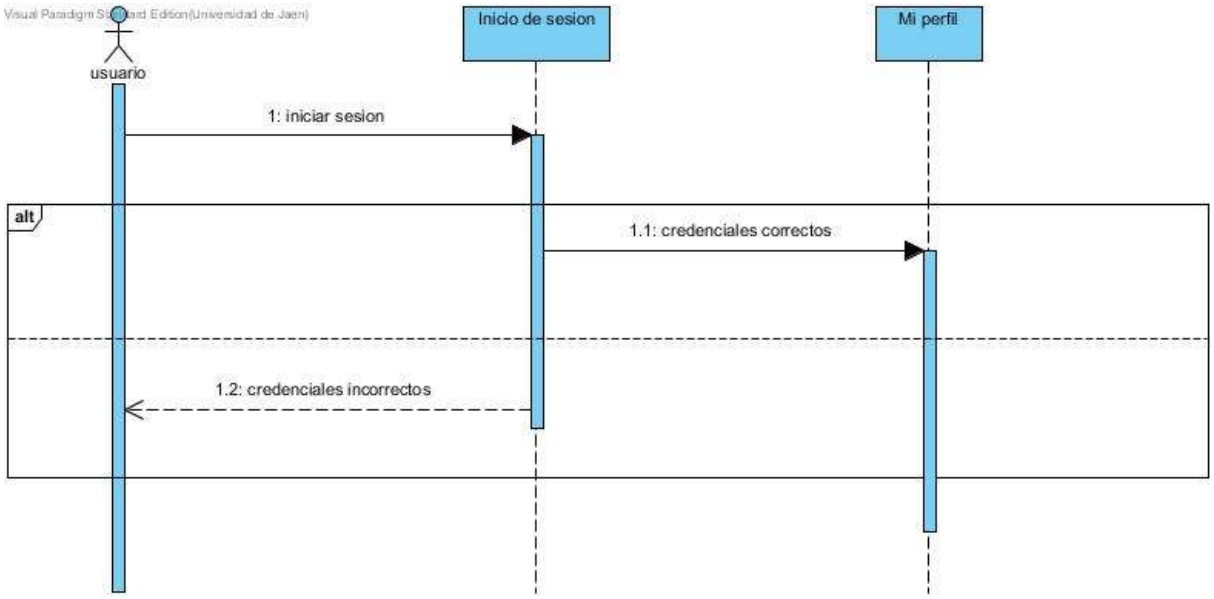

Ilustración 53: Diagrama de secuencia de inicio de sesión

8.1.4. Barra de navegación de páginas

Es un *nav* común a todas las páginas a partir del inicio de sesión que nos permite movernos por las secciones de la aplicación. Incluyen los botones *mi perfil,* que redirige a la página de perfil personal, el botón *buscador*, que redirige al buscador donde buscar escuelas, sectores y vías, y por último un botón *cerrar sesión* que llama al servicio GET */logout* implementado la configuración de *Spring Security,* el cual elimina la sesión borrando la cookie de sesión y redirige a la página de portada.

8.1.5. Página de perfil

Es la primera página que el usuario se encuentra tras iniciar sesión de forma satisfactoria. En ella se pueden ver sus datos de perfil, su máximo nivel completado así como una pequeña fotografía a gusto del usuario. También se pueden ver los

amigos así como un enlace para acceder a sus perfiles. De igual manera ocurre con las vías realizadas.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- GET */perfil*, para obtener los datos de perfil.
- GET */perfil/amigos,* para obtener la lista de amigos.
- GET */perfil/vias*, para obtener las vías realizadas por un usuario.

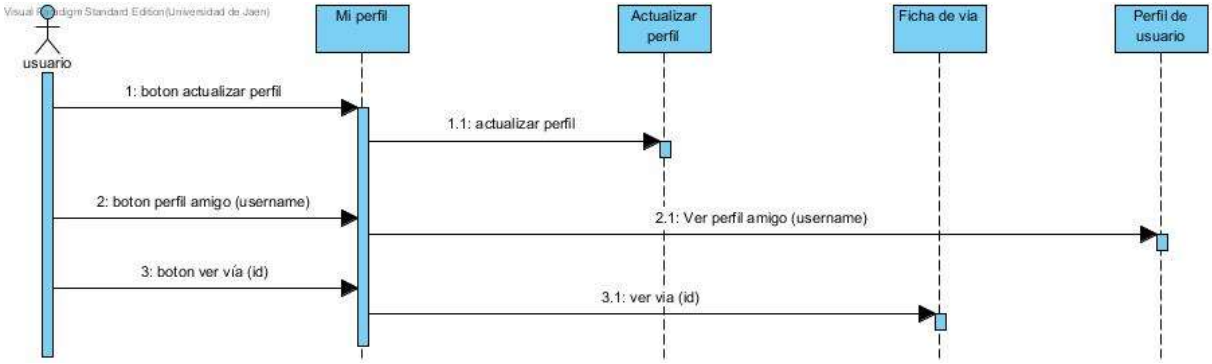

Ilustración 54: Diagrama de secuencia de página de perfil

8.1.6. Formulario de actualizar perfil de usuario

Es una página que contiene el formulario donde el usuario puede actualizar la información personal que estime oportuna. Se accede desde la página de perfil personal a través de un botón habilitado para ello. Cuando el usuario confirma se recarga y le muestra los datos ya actualizados. Si hubiera algún error se muestra con el mensaje correspondiente.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- GET */perfil*, para obtener los datos de perfil.
- POST */perfil*, para actualizar los datos de perfil.

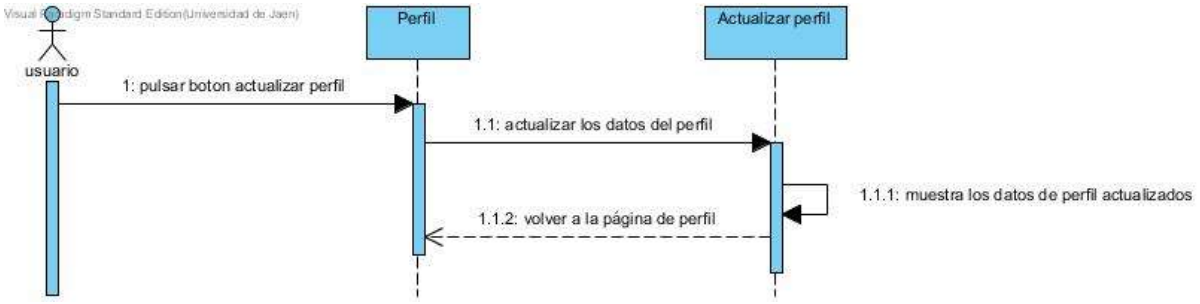

Ilustración 55: Diagrama de secuencia de formulario de actualizar perfil de usuario

8.1.7. Cierre de la iteración 9

Se han desarrollado y presentado un total de 45 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 181 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

Analizando brevemente nuestra velocidad de trabajo, comprobamos que aún nos faltan 5 iteraciones para la finalización el proyecto. La velocidad de trabajo estimada y planificada fue de 40 puntos de historia por iteración. Teniendo en cuenta que únicamente nos quedan 181, quiere decir que vamos cumpliendo los plazos planificados.

8.2. Iteración 10: del 13/03/2017 al 24/03/2017

8.2.1. Análisis de la iteración 10

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Formulario de cambio de contraseña: 8 puntos de historia
- Selector de búsqueda (buscador): 5 puntos de historia
- Buscador de usuarios: 8 puntos de historia
- Buscador de escuelas: 8 puntos de historia
- Ficha de escuela: 13 puntos de historia

Total de puntos de historia seleccionados para la décima iteración: 42

8.2.2. Formulario de cambio de contraseña

Para acceder al formulario de cambio de contraseña se hace accediendo desde el formulario de actualizar perfil, mediante el botón habilitado para tal fin. En primer lugar hay que introducir la contraseña actual para poder cambiarla. En caso de error se muestra al usuario mediante el mensaje correspondiente.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- PUT */perfil/password*, para obtener la contraseña de usuario.

8.2.3. Selector de búsqueda: buscador

En la sección del buscador, este selector es aquel que te permite elegir entre buscar usuarios mediante *username*o buscar escuelas de escalada por provincia.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.

8.2.4. Buscador de usuarios

Una vez elegimos el tipo de búsqueda que queremos realizar, en el caso de querer buscar usuarios nos pedirá el *username* de la persona que queremos buscar. Si el sistema identifica un perfil con esos datos nos mostrará un pop-up con una opción por si queremos acceder al perfil de ese usuario.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- GET */perfil/{username}*

Se actuará en función del código HTTP que devuelva la petición para saber si el usuario está registrado en el sistema.

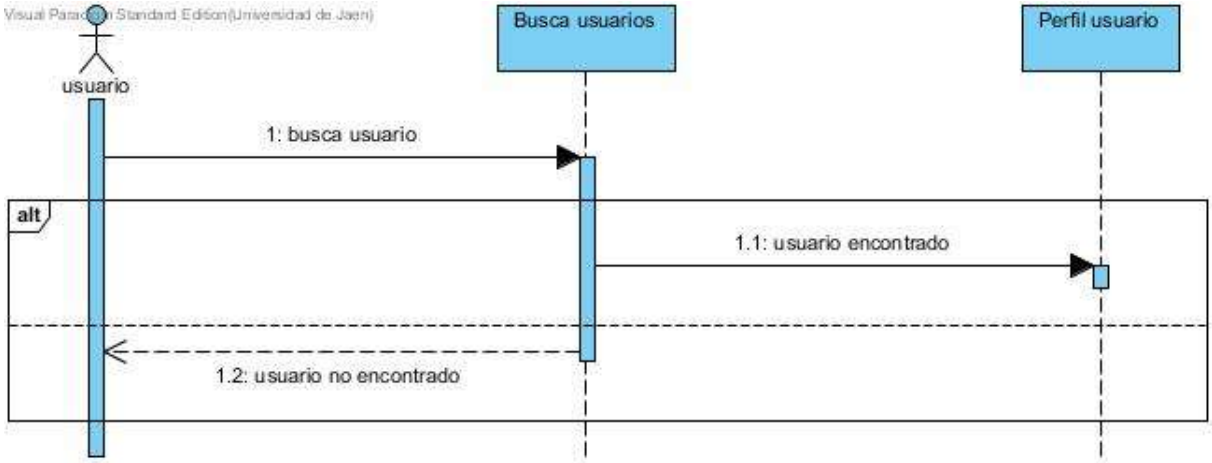

Ilustración 56: Diagrama de secuencia de buscador de usuarios

8.2.5. Buscador de escuelas

En este caso la búsqueda, tal y como sabemos por la especificación de requisitos, se hace por provincias. Para ello se muestra un selector con las provincias para que el usuario pueda elegir. Una vez elegida la provincia, el usuario confirmará y se le redirigirá a la página de las escuelas seleccionadas. El servicio web que inicia la búsqueda vendrá en esta página en función de una variable pasada a través de la URL con el código de provincia.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.

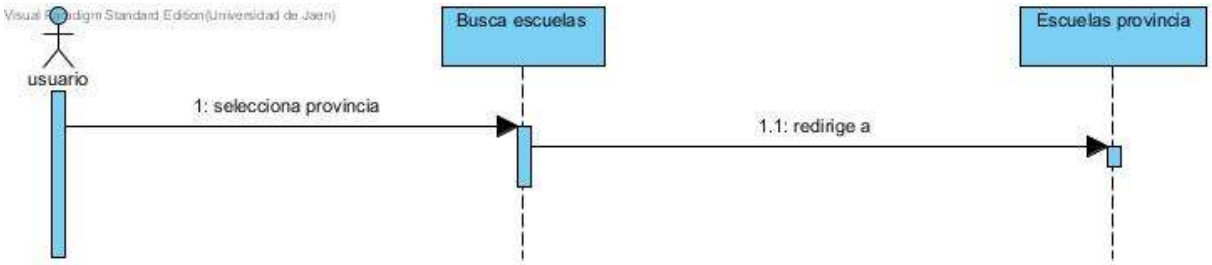

Ilustración 57: Diagrama de secuencia de buscador de escuelas

8.2.6. Escuelas de una provincia y acceso a ficha de una escuela

En esta historia de usuario un tanto compleja desarrollaremos dos funcionalidades. La primera, es la muestra de las escuelas previamente seleccionadas en la historia de usuario anterior. La segunda funcionalidad, y no por ello menos importante, es el acceso a los datos de una escuela y la muestra de no sólo de sus datos, sino de sus sectores.

Por tanto, en primer lugar, la página que presenta las escuelas tal y como hemos comentado captura el código de provincia de la URL y llama al servicio web correspondiente para obtener los datos.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- GET */provincia,* para obtener la provincia seleccionada y poder mostarla.
- GET */escuelas/{cod_provincia}*, para obtener las escuelas de esa provincia.

En segundo lugar, los datos de una escuela concreta y sus sectores. De la misma forma cuando se pulsa el botón para acceder a una escuela concreta estamos pasando a través de la URL una variable con el código de la escuela que el recurso web capturará para obtener los datos que estamos solicitando.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- GET */escuela/{cod_escuela},* para obtener los datos de una escuela.
- GET */sectores/{cod_escuela},* para obtener los sectores de una escuela.

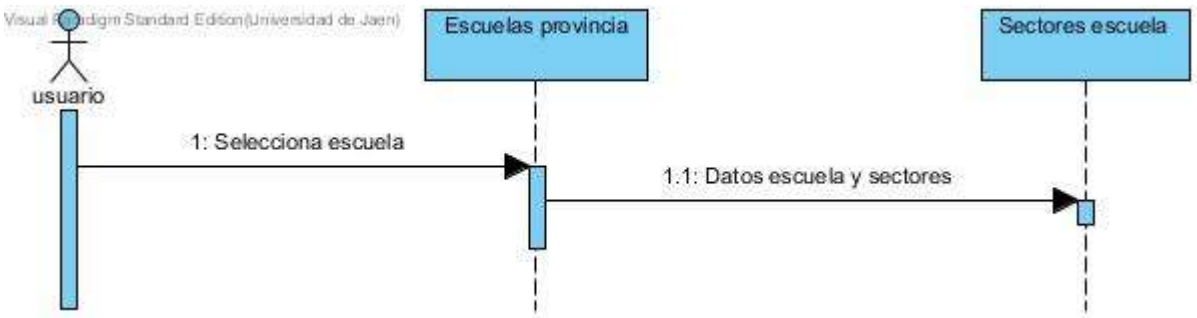

Ilustración 58: Diagrama de secuencia de escuelas de una provincia y acceso a ficha de una escuela

8.2.7. Cierre de la iteración 10

Se han desarrollado y presentado un total de 42 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 139 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

8.3. Iteración 11: del 27/03/2017 al 07/04/2017

8.3.1. Análisis de la iteración 11

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Ficha de sector: 13 puntos de historia
- Ficha de vía y valoraciones: 21+13 puntos de historia

Total de puntos de historia seleccionados para la décima iteración: 47

8.3.2. Ficha de sector

En esta historia de usuario se ha desarrollado la página que muestra los datos de un sector. Al igual que para las escuelas, se envía por la URL la variable con el código que es capturada por el servicio web y conseguir los datos, tanto del sector como de las vías que lo componen.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- GET */sector/{cod_sector},* para obtener los datos de un sector.
- GET */vias/{cod_sector},* para obtener las vías de un sector.

8.3.3. Ficha de vía y valoraciones

Esta historia de usuario centra la funcionalidad principal de la aplicación, pues es la página desde la cual se pueden ver los datos de la vía y las valoraciones, puntuaciones y comentarios añadidos hasta ahora. Para añadir una nueva opinión/valoración y opcionalmente un comentario, hay un botón habilitado para ello. Se muestra un pop-up con el perceptivo formulario para introducir los datos y enviarlos al servidor a través del servicio web correspondiente.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- GET */via/{cod_via},* para obtener los datos de una vía.
- GET */comentarios/{cod_via},* para obtener los comentarios/valoraciones.

- POST */comentarios/{cod_via},* para añadir un nuevo comentario/valoración.
- POST */perfil/vías/{cod_via}*, para introducir la valoración y tenerla en cuenta con el algoritmo de reestimación.

8.3.4. Cierre de la iteración 11

Se han desarrollado y presentado un total de 47 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 92 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

8.4. Iteración 12: del 17/04/2017 al 28/04/2017

8.4.1. Análisis de la iteración 12

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Formulario para añadir escuela: 8 puntos de historia
- Formulario para añadir sector: 8 puntos de historia
- Formulario para añadir vía: 8 puntos de historia

Total de puntos de historia seleccionados para la décima iteración: 24

8.4.2. Formulario para añadir escuela

Se trata de un formulario disponible para el usuario cuando no encuentra la escuela que busca dentro de una provincia, por lo que se le da la posibilidad de añadirla él mismo.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- POST */escuelas,* para enviar al servidor los datos de una escuela.

8.4.3. Formulario para añadir sector

Ídem anterior pero con los sectores.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- POST */sectores,* para enviar al servidor los datos de un sector.

8.4.4. Formulario para añadir vía

Ídem anterior pero con las vías.

Servicios web utilizados:

- GET */username*, para obtener el usuario conectado.
- POST */vias,* para enviar al servidor los datos de una vía.

8.4.5. Cierre de la iteración 12

Se han desarrollado y presentado un total de 24 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 81 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

8.5. Iteración 13: del 02/05/2017 al 12/05/2017

8.5.1. Análisis de la iteración 13

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Desarrollo de la documentación del proyecto: 55 puntos de historia

Total de puntos de historia seleccionados para la décima iteración: 55

8.5.2. Cierre de la iteración 13

Se han desarrollado y presentado un total de 55 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito. Para finalizar el proyecto nos restan 13 historias de usuario. En esta iteración no se añade ni se elimina ninguna historia de usuario, por lo que no hay que reestimar.

8.6. Iteración 14: del 15/05/2017 al 26/05/2017

8.6.1. Análisis de la iteración 14

Las historias de usuario seleccionadas para esta iteración son las siguientes:

- Preparación de la presentación final: 13 puntos de historia

Total de puntos de historia seleccionados para la décima iteración: 13

8.6.3. Cierre de la iteración 14

Se han desarrollado y presentado un total de 13 historias de usuario en esta sexta iteración de forma correcta. La iteración se ha llevado a cabo con éxito, por lo que damos por concluido el proyecto. Posteriormente se celebrará la reunión de retrospectiva.

9. Funcionamiento

9.1. Vídeo de demostración. Manual de usuario virtual

Aquí se puede ver una pequeño vídeo de demostración del funcionamiento de la aplicación [33]. Para ver el vídeo pulsar aquí.

9.2. Código del proyecto

Una vez finalizado el proyecto para su entrega voy a subir el código a un repositorio público en la plataforma *github* para facilitar su acceso.

- Código del servidor [34]: Para acceder al código pulsar aquí.
- Código del cliente [35]: Para acceder al código pulsar aquí.

10. Conclusiones y líneas futuras

Una de las principales conclusiones que obtengo a la finalización de este trabajo es sobre cómo enfocar proyectos próximos. En este caso, era la primera vez que me lanzaba en el desarrollo de un producto real, y al ser mi Trabajo Fin de Grado, he tenido que realizarlo en solitario. Creo que en este tipo de trabajos formar un buen equipo de profesionales es algo fundamental. Cuestiones relativamente sencillas que me han supuesto gran cantidad de tiempo y quebraderos de cabeza hubiesen sido más sencillas de llevar a cabo. Sin duda, el no asistir a clases y tener que trabajar en solitario es lo que peor he llevado este año.

Quiero expresar también mi satisfacción en lo que aprendizaje autónomo se refiere. Tocar tecnologías JavaScript como AngularJS que hasta ahora desconocía por completo me ha abierto un mundo de posibilidades sobre el desarrollo de aplicaciones web. Perfeccionar el tema de los servicios web REST ha sido otra de mis grandes satisfacciones, así como el uso de un SGBD que hasta ahora no había utilizado. Por otro lado, conocimientos sobre SSL y configuración CORS de Tomcat habían sido conceptos que hasta hace unos meses no conocía. En resumidas cuentas estoy muy satisfecho con las tecnologías que he aprendido y perfeccionado.

Decir que ha sido un proyecto en el que, a pesar de las numerosas dificultades que me he ido encontrando a lo largo del camino, he disfrutado muchísimo con las tecnologías empleadas y espero poder seguir mejorando, incluyendo también el abrirme a nuevas tecnologías y aprendizajes a raíz del trabajo realizado.

Por otro lado, me gustaría añadir unas mejoras que realmente me hubiera gustado implementar, pero que simplemente por falta de tiempo no he podido estudiar a fondo e implementarlas. Por ejemplo, en lo que respecta al *back-end*, yo incluiría de primera mano la recuperación de contraseña en caso de olvido. Esto se podría implementar fácilmente con un token que enviara la contraseña al correo electrónico de la misma forma que se envía la confirmación de registro. Por otro lado, revisar los métodos HTTP e incluir o quitar, o incluso rediseñar, algunos de los atributos de las clases de los datos que se manejan. Incluiría también unas coordenadas de ubicación para las escuelas, de forma que pudiéramos implementar

una obtención de escuelas en base a la ubicación, y más con un servidor que puede ser utilizado en la implementación de clientes móviles.

En lo que respecta al *front-end*, la conclusión está clara. No es más que la implementación de clientes para dispositivos móviles dada la tecnología actual. Prácticamente cualquier persona tiene acceso a internet y lleva un Smartphone en el bolsillo, lo cual hace fundamental, bajo mi punto de vista y a pesar del diseño web responsive, una aplicación específica para *Android* o *IOS*, por ejemplo.

Referencias

[1] «LOS GRADOS EN LA ESCALADA ¿QUIÉN LOS ESTABLECE?,» [En línea]. Available: http://escaladagranada.es/graduacion-en-la-escalada.

[2] «Qué es la escalada. Aspectos básicos de la escalada deportiva.,» [En línea]. Available: https://es.wikipedia.org/wiki/Escalada.

[3] «Escuela de escalada,» [En línea]. Available: https://es.wikipedia.org/wiki/Escuela_de_escalada.

[4] «Graduación de dificultad,» [En línea]. Available: https://es.wikipedia.org/wiki/Graduaci%C3%B3n_de_dificultad#Franc.C3.A9s.

[5] «J2EE official Site,» Oracle, [En línea]. Available: http://www.oracle.com/technetwork/java/javaee/documentation/index.html.

[6] «Spring Framework, official site,» [En línea]. Available: https://spring.io/.

[7] «MySQL, The world's most popular open source database,» [En línea]. Available: https://www.mysql.com/.

[8] «Spring Data JPA - Reference Documentation,» [En línea]. Available: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

[9] «Bootstrap: Sleek, intuitive, and powerful front-end framework for faster and easier web development,» [En línea]. Available: http://getbootstrap.com/2.3.2/.

[10] «Control de versiones .git,» [En línea]. Available: https://git-scm.com/.

[11] «Bitbucket, The git solution for professional teams,» [En línea]. Available: https://bitbucket.org/product.

[12] «Github, Learn Git and GitHub without any code!,» [En línea]. Available: https://github.com/.

[13] «Manifiesto Ágil,» 2001. [En línea]. Available: http://agilemanifesto.org/iso/es/manifesto.html.

[14] «Metodología Scrum,» [En línea]. Available: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software).

[15] «Qué es Scrum,» [En línea]. Available: https://proyectosagiles.org/que-es-scrum/.

[16] «Roles Scrum,» [En línea]. Available:

] https://www.pymesyautonomos.com/management/cerdos-y-gallinas-metodologia-scrum.

[17 «Tipos de reuniones Scrum,» [En línea]. Available: http://www.obs-edu.com/es/blog-
] project-management/scrum/cuales-son-los-tipos-de-reuniones-del-metodo-scrum.

[18 «Product Backlog,» [En línea]. Available: https://proyectosagiles.org/lista-requisitos-
] priorizada-product-backlog/.

[19 «Historia de usuario Scrum,» [En línea]. Available:
] http://www.scrummanager.net/bok/index.php?title=Historia_de_usuario.

[20 «Planning Poker - Sprints made simple. Estimates made easy.,» [En línea]. Available:
] https://www.planningpoker.com/.

[21 «¿Por qué utilizamos puntos de historia para estimar y no horas?,» [En línea]. Available:
] http://www.javiergarzas.com/2015/06/puntos-historia-para-estimar-y-no-horas.html.

[22 «Qué son los diagramas de quemado y guía de uso,» [En línea]. Available:
] http://managementplaza.es/blog/guia-uso-paso-paso-los-diagrama-quemado/.

[23 «Dominio web y alojamiento por un año,» [En línea]. Available:
] https://www.anw.es/alojamiento-web/alojamiento-hosting-java.html.

[24 «Official User Guide Hibernate ORM,» [En línea]. Available:
] http://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.ht
ml.

[25 «AngularJS, Superheroic JavaScript MVW Framework,» [En línea]. Available:
] https://angularjs.org/.

[26 «HTTP Basic auth password storage more secure than Digest auth,» [En línea].
] Available: https://security.stackexchange.com/questions/18626/http-basic-auth-
password-storage-more-secure-than-digest-auth.

[27 «Encrypted password in database and browser digest auth,» [En línea]. Available:
] https://stackoverflow.com/questions/18551954/encrypted-password-in-database-and-
browser-digest-auth.

[28 «Securing Your Tomcat App with SSL and Spring Security,» [En línea]. Available:
] https://dzone.com/articles/securing-your-tomcat-app-ssl.

[29 «Do any security experts recommend bcrypt for password storage?,» [En línea].
] Available: https://security.stackexchange.com/questions/4781/do-any-security-experts-
recommend-bcrypt-for-password-storage.

[30 «Control de acceso HTTP (CORS),» [En línea]. Available:
] https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS.

[31 «Política del mismo origen,» [En línea]. Available:

] https://es.wikipedia.org/wiki/Pol%C3%ADtica_del_mismo_origen.

[32 «CORS filter,» [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-
] doc/config/filter.html#CORS_Filter.

[33 «Vídeo de demostración de la aplicación,» [En línea]. Available:
] https://www.youtube.com/watch?v=6wSwdCKSYFc.

[34 «Código del servidor,» [En línea]. Available:
] https://github.com/dmg00024/redsocialcolaborativarestful.

[35 «Código del cliente web usando AngularJS,» [En línea]. Available:
] https://github.com/dmg00024/redsocialcolaborativaclientangularjs.