TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

"Redmine Teaching Extension: Plugin del sistema de peticiones de Redmine para su uso en docencia"

Curso 2015/2016

Alumno/a:

Mario Merino Navarro

Director/es:

Manuel Torres Gil

Agradecimientos

Antes que nada, me gustaría agradecer el apoyo, supervisión y consejos recibidos por parte de mi tutor Manuel Torres Gil. Su aporte de ideas, tanto en el planteamiento de los objetivos como en el empleo de herramientas y métodos de trabajo adecuados, ha sido fundamental para afrontar con éxito la realización de este proyecto.

A mi familia, por su enorme paciencia conmigo en los malos momentos y por su gran apoyo para seguir avanzando en el transcurso de mis estudios en la Universidad.

A mis amigos, por su aporte de conocimientos y por poder compartir grandísimos momentos junto a ellos, haciendo más amenos estos duros y largos años de carrera.

Por último, agradecer a todo el profesorado del Grado en Ingeniería Informática de la Universidad de Almería por su dedicación para recibir la mejor formación posible y poder cumplir con éxito todos los objetivos a nivel de estudios, así como la realización de este Trabajo Fin de Grado.

Gracias a todos,

Mario Merino Navarro

Índice de contenido

Índice de figuras

Índice de tablas

Listado de abreviaturas

MVC: **M**odel - **V**iew - **C**ontroller (Modelo-Vista-Controlador)

DAFO: **D**ebilidades - **A**menazas - **F**ortalezas - **O**portunidades

DRS: **D**ocumento de **R**equisitos del **S**istema

HTA: **H**ierarchical **T**ask **A**nalysis (Análisis Jerárquico de Tareas)

API: **A**pplication **P**rogramming **I**nterface (Interfaz de Programación de Aplicaciones)

UML: **U**nified **M**odeling **L**anguage (Lenguaje Unificado de Modelado)

CASE: **C**omputer **A**ided **S**oftware **E**ngineering (Ingeniería del Software Asistida por Computadora)

IDE: **I**ntegrated **D**evelopment **E**nvironment (Entorno de Desarrollo Integrado)

HTML: **H**yper**T**ext **M**arkup **L**anguage (Lenguaje de Marcas de Hipertexto)

CSS: **C**ascading **S**tyle **S**heets (Hoja de Estilos en Cascada)

DOM: **D**ocument **O**bject **M**odel (Modelo de Objetos del Documento)

OBJ: **Obj**etivo del sistema

RI: **R**equisito de **I**nformación

RF: **R**equisito **F**uncional

ACT: **Act**or del sistema

RNF: **R**equisito **N**o **F**uncional

1. Introducción

Redmine [1] es una plataforma web para la gestión de proyectos que nos permite almacenar las tareas a realizar dentro de un proyecto, desde los requerimientos de nuevas funcionalidades, hasta la petición y corrección de errores.

Actualmente se está considerando la opción de utilizar *Redmine* como apoyo a la docencia en el Grado en Ingeniería Informática de la Universidad de Almería, de forma que las asignaturas sean concebidas como proyectos, y cada una de las actividades de las asignaturas sean tareas/peticiones de *Redmine*. Se ofrece así un soporte para la gestión de peticiones asignadas, permitiendo tanto a los profesores de una asignatura como a sus alumnos, conocer el grado de evolución del trabajo encargado.

La propuesta de llevar a cabo este proyecto, viene de la necesidad de implementar una mejora en el sistema *Redmine* que permita al profesor definir las peticiones a nivel de asignatura y dichas peticiones se propaguen a los proyectos de cada alumno, concebidos como subproyectos del proyecto padre que es la asignatura en cuestión. En definitiva, la creación de una petición por parte del profesor crearía una tarea idéntica en el proyecto de cada alumno.

Así pues, con la implementación de esta extensión o *plugin* en *Redmine*, se pretende habilitar la mejora anteriormente mencionada, facilitando y agilizando de este modo el sistema de creación de peticiones entre alumnos de una asignatura.

1.1. Motivación para la realización

Una gran motivación personal para llevar a cabo el desarrollo de este proyecto, fue la oportunidad de enfrentarme a la creación de un producto software destinado a ofrecer apoyo a la docencia del Grado en Ingeniería Informática de la Universidad de Almería, concretamente a aquellos docentes que actualmente emplean la plataforma web de gestión de proyectos *Redmine* como herramienta de asignación de tareas a los distintos alumnos de una asignatura.

Además, esto suponía afrontar un concepto nuevo en el campo del desarrollo de software, que es el del desarrollo de plugins o extensiones. El objetivo de un *plugin* es el de aportar una nueva funcionalidad muy específica a otra aplicación. Esta nueva aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la *API* (Application Programming Inteface – Interfaz de Programación de Aplicaciones). Precisamente esto es lo que se pretendía conseguir, dotar a la plataforma *Redmine* de una nueva funcionalidad orientada a ciertas necesidades de docencia del Grado de Ingeniería Informática de la Universidad de Almería.

Por último, y no menos importante, estaba el hecho de enfrentarse al empleo de tecnologías y herramientas de desarrollo nuevas, como es el framework de aplicaciones web *Ruby on Rails* escrito en el lenguaje de programación *Ruby* y que sigue un patrón de arquitectura de software Modelo-Vista-Controlador (*MVC*), mediante el cual está desarrollado el sistema de gestión de proyectos *Redmine*, así como la biblioteca *jQuery* de *JavaScript* que permite simplificar la manera en que se lleva a cabo la interacción entre vistas. Si bien el concepto de *MVC* sí fue aprendido a lo largo del

Grado, el aprendizaje del framework de aplicaciones web *Ruby on Rails*, así como el lenguaje de programación *Ruby* y el uso de la biblioteca *jQuery* de *JavaScript* para implementar las funciones de interacción entre vistas, sí supuso un gran reto.

1.2. Objetivos

El objetivo principal de este proyecto es crear una extensión funcional para el sistema de peticiones de la plataforma de gestión de proyectos *Redmine*, en el ámbito de docencia del Grado en Ingeniería Informática de la Universidad de Almería. Su función es la de facilitar y agilizar al profesorado su tarea de asignación de peticiones/tareas a los alumnos inscritos en sus asignaturas.

Para la consecución exitosa de este proyecto, se debe proceder con el cumplimiento de una serie de subobjetivos clave que permitan el buen avance en el desarrollo del proyecto:

- Estudiar el esquema de la base de datos de la plataforma web *Redmine*, así como su código fuente, para familiarizarse tanto con el funcionamiento de los lenguajes de programación, frameworks y bibliotecas con los que se trabajará (*Ruby, Ruby on Rails*, *JavaScript, jQuery*…) como con el entorno del sistema donde se incluirá la extensión a desarrollar.
- Crear y/o modificar las vistas de *Redmine* que conformarán el *plugin* (*Front-end* de la aplicación web) y que extenderán la ventana correspondiente al tratamiento de peticiones de *Redmine*.
- Crear las funcionalidades que permiten la interacción entre las vistas de *Redmine* y el *plugin*, así como la conexión con la Base de Datos relacional *MySQL* de la que se compone el sistema (*Back-end* de la aplicación web)
- Extender la *API REST* de *Redmine* a partir de los modelos creados para llevar a cabo el desarrollo del *plugin*.
- Almacenar todo el código fuente del desarrollo del *plugin* en un directorio externo, sin necesidad de modificar el código original de *Redmine*. De este modo, se podrá instalar libremente el *plugin* en cualquier plataforma de *Redmine* para su uso.

A su vez, todos estos subobjetivos implican un aprendizaje de las tecnologías nuevas implicadas en el proceso de desarrollo (*Ruby on Rails, jQuery…*), así como una ampliación de conocimientos sobre la interacción con una base de datos siguiendo el patrón de arquitectura software Modelo-Vista-Controlador (*MVC*) estudiados a lo largo de la carrera.

1.3. Análisis DAFO

El *análisis DAFO*, es una metodología de estudio de la situación de un proyecto, analizando sus características internas (Debilidades y Fortalezas) y su situación externa (Amenazas y Oportunidades) en una matriz cuadrada. Se trata de una herramienta para conocer la situación real en la que se encuentra el proyecto a desarrollar y planear así una estrategia de futuro.

1.3.1. Debilidades

- Ninguna experiencia previa en el desarrollo de plugins.
- Empleo de tecnologías y lenguajes de programación poco o nada conocidos, lo cual ralentiza en cierto modo el proceso de desarrollo del *plugin*.

- El *plugin* va dirigido a un sector muy específico de usuarios de *Redmine* (profesorado que use *Redmine* como soporte para la gestión de sus asignaturas), por lo que gran parte del resto de usuarios de *Redmine* que pretendan instalarlo, no le vean gran utilidad o interés en su uso.

1.3.2. Amenazas

- Existe un directorio de plugins de *Redmine* muy extenso donde usuarios experimentados van alojando sus creaciones constantemente. Por ello, es probable que con el tiempo aparezcan alternativas más refinadas a este proyecto (*Redmine Teaching Extension*).
- Aunque es poco probable que ocurra a corto plazo, alguna próxima actualización de versión de *Redmine* podría contener por defecto la funcionalidad propia del *plugin* desarrollado o alguna alternativa similar, con lo cual perdería sentido la existencia del mismo.

1.3.3. Fortalezas

- Conocimiento del funcionamiento de la plataforma *Redmine* debido a su uso en algunas asignaturas del Grado.
- Conocimiento del patrón de arquitectura software Modelo-Vista-Controlador (*MVC*) en el cual se basa *Redmine* y conocimientos básicos de desarrollo web.
- Producto software destinado a facilitar las labores docentes en las asignaturas del Grado en Ingeniería Informática de la Universidad de Almería que cuentan con *Redmine* como plataforma de soporte para la asignación de tareas a los alumnos.
- Al igual que *Redmine*, es un proyecto de software libre y código abierto, puesto a la disposición de cualquier usuario de esta plataforma que desee instalarlo en la misma para satisfacer sus necesidades.
- Al tratarse de una extensión de *Redmine*, es fácilmente actualizable y manipulable por si se pretenden introducir mejoras a lo largo del tiempo (ya sea por el propio creador o por cualquier otro desarrollador), sin afectar por ello al código fuente que constituye la plataforma.
- El consumo de recursos y el espacio que ocupa en memoria es mínimo.

1.3.4. Oportunidades

- Al tratarse de una extensión que agiliza y automatiza una funcionalidad existente en *Redmine*, esto dotará de un ahorro considerable de tiempo al profesorado que la use.
- Es un producto gratuito y fácil de instalar y configurar en el sistema *Redmine*, lo que lo hace muy accesible y atractivo.

1.4. Planificación temporal

El proyecto se ha desarrollado siguiendo una planificación temporal que abarca cada una de las actividades del Ciclo de Vida del software (*Análisis y Planificación, Modelado y Diseño, Implementación, Pruebas y Documentación*).

Fase de Análisis y Planificación

Abarca aproximadamente entre principios de Agosto y mediados de Noviembre de 2015. En esta primera fase, y tras varias reuniones con el tutor para ir obteniendo ideas y definiendo posibles funcionalidades sobre el proyecto a desarrollar, se procedió con el aprendizaje de las tecnologías implicadas en el desarrollo, como son el framework web *Ruby on Rails* (en el cual se basa el desarrollo del sistema *Redmine* y, por tanto, el *plugin* a desarrollar) y la biblioteca *jQuery* de *JavaScript*, entre otros.

Por otro lado, se fueron estudiando las pautas a seguir en el desarrollo de un *plugin* en *Redmine* para tener un modelo a seguir a la hora de la implementación, lo cual se llevó a cabo sobre todo mediante la lectura del libro *Redmine Plugin Extension and Development* [2] que ofrece una documentación muy completa al respecto. A su vez se fue configurando un sistema *Redmine* vacío en una partición del sistema operativo *Ubuntu 14.04* que se empleó como soporte para realizar las distintas comprobaciones durante el desarrollo del proyecto.

Finalmente, y para terminar el período de *Análisis*, se establecieron herramientas de soporte al desarrollo del proyecto como *ScrumDo*, para el seguimiento y control de las tareas basadas en una gestión ágil y *GitHub*, que es un repositorio de código fuente basado en el sistema de control de versiones *Git*. Precisamente, para llevar a cabo la instalación del *plugin* desarrollado en *Redmine*, habrá que descargar la carpeta con el código fuente del proyecto (con las instrucciones de instalación indicadas) desde *GitHub* [3].

Fase de Modelado y Diseño

Abarca varias fechas durante el mes de Diciembre de 2015. En esta fase destaca la elaboración de los mockups o prototipos de la interfaz del *plugin* a desarrollar, así como el modelado y diseño del diagrama de casos de uso del funcionamiento del *plugin* y la definición de los diferentes requisitos funcionales y no funcionales necesarios en el sistema.

Breve fase previa de Documentación (Anteproyecto del TFG)

Aunque es parte de la última fase del Ciclo de Vida, se lleva a cabo el desarrollo del *Anteproyecto del Trabajo Fin de Grado*, que consiste en un resumen preliminar del propósito del proyecto a desarrollar que incluye una introducción del mismo, objetivos, fases de desarrollo, materiales y métodos empleados, y que resulta necesario entregar al comienzo del período de desarrollo del *Trabajo Fin de Grado* para evaluar que se cumplen todas las competencias y contenidos aptos para un *Trabajo Fin de Grado* del *Grado en Ingeniería Informática*.

La entrega del *Anteproyecto* se realiza a finales de Febrero de 2016. Esta demora de dos meses aproximadamente en la planificación temporal se debe a que desde finales del periodo de vacaciones de Diciembre hasta primeros de Marzo se impartió un curso de desarrollo de aplicaciones móviles en la *Universidad de Almería*, lo cual supuso un parón temporal en el desarrollo del *TFG*.

Fase de Implementación

Tras aprobarse el *Anteproyecto del Trabajo Fin de Grado*, se continúa con la fase de *Implementación* desde finales de Marzo hasta mediados de Julio de 2016. Se trata de la fase más larga y compleja, ya que abarca toda la codificación del proyecto. Por ello, para empezar se localizó el código fuente de la *API REST* de *Redmine* y se procedió con una implementación preliminar de parte de la funcionalidad del proyecto, consistente en el desarrollo de una vista de selección de subproyectos (alumnos) de un proyecto raíz (asignatura) para, más adelante y en la última fase de implementación, proceder con el desarrollo de la propagación de una petición entre dichos subproyectos (lo cual corresponde al propósito final del desarrollo de este *plugin*, que es la propagación múltiple de peticiones de *Redmine*).

Al tratarse *Redmine* de una plataforma colaborativa de gestión de proyectos, es necesario establecer unos permisos de usuario para controlar la visibilidad de las peticiones realizadas por los usuarios según el nivel de control que tienen en la plataforma (administrador, usuario registrado, usuario visitante…), por lo que la siguiente fase de codificación sería esta.

Una vez desarrollada la vista de selección de subproyectos (para la posterior propagación de peticiones a los mismos) y definidos los permisos necesarios, se procede a principios de Mayo con la implementación de la lógica de asignación de peticiones a subproyectos desde la ventana de su proyecto padre. En principio, se opta por una asignación de peticiones a nivel individual (1 petición por 1 subproyecto, desde el proyecto padre), ya que la lógica de *Redmine*, a priori, es incompatible con la asignación múltiple y es necesario ir avanzando cuidadosamente solucionando aquellas incompatibilidades que se vayan presentando para dar con la solución final (propagación multiple de una petición).

Para terminar con la fase de *Implementación*, se empieza a principios de Junio con una modificación final de la vista del *plugin*, depurando para ello el aspecto inicial de la vista de la extensión para adaptarla lo mejor posible a la interfaz original de *Redmine*. Una vez finalizado este pequeño arreglo, se procede a codificar el punto clave de este proyecto, que es la función de propagación de peticiones desde un proyecto padre a todos sus subproyectos, lo cual finaliza a mediados de Julio y que da paso a la siguiente fase.

Fase de Pruebas y Documentación

Esta última actividad del Ciclo de Vida del software corresponde con las *Pruebas y Documentación*, fase que se está llevando a cabo durante la redacción de este informe y en donde se realizan diferentes pruebas de testing para comprobar todo tipo de errores que pudieran surgir y verificar así el correcto funcionamiento del *plugin* desarrollado y su integración en *Redmine*. Una vez terminado esto, ya solo quedaría revisar el informe final del proyecto y proceder con la creación de la presentación de la defensa del mismo.

Como resumen de todas las tareas que forman parte de la planificación temporal, se ha diseñado un tablero de tareas o *Scrumboard* en la herramienta web *ScrumDo* para seguir una metodología *Scrum* combinada con *Kanban*, donde se van colocando diferentes tarjetas correspondientes a las tareas que forman parte de la planificación temporal. Cada una de estas tareas/tarjetas, presentan una definición de las mismas, un nivel de dificultad y su estado actual (*Para hacer*, *En progreso* o *Finalizada*).

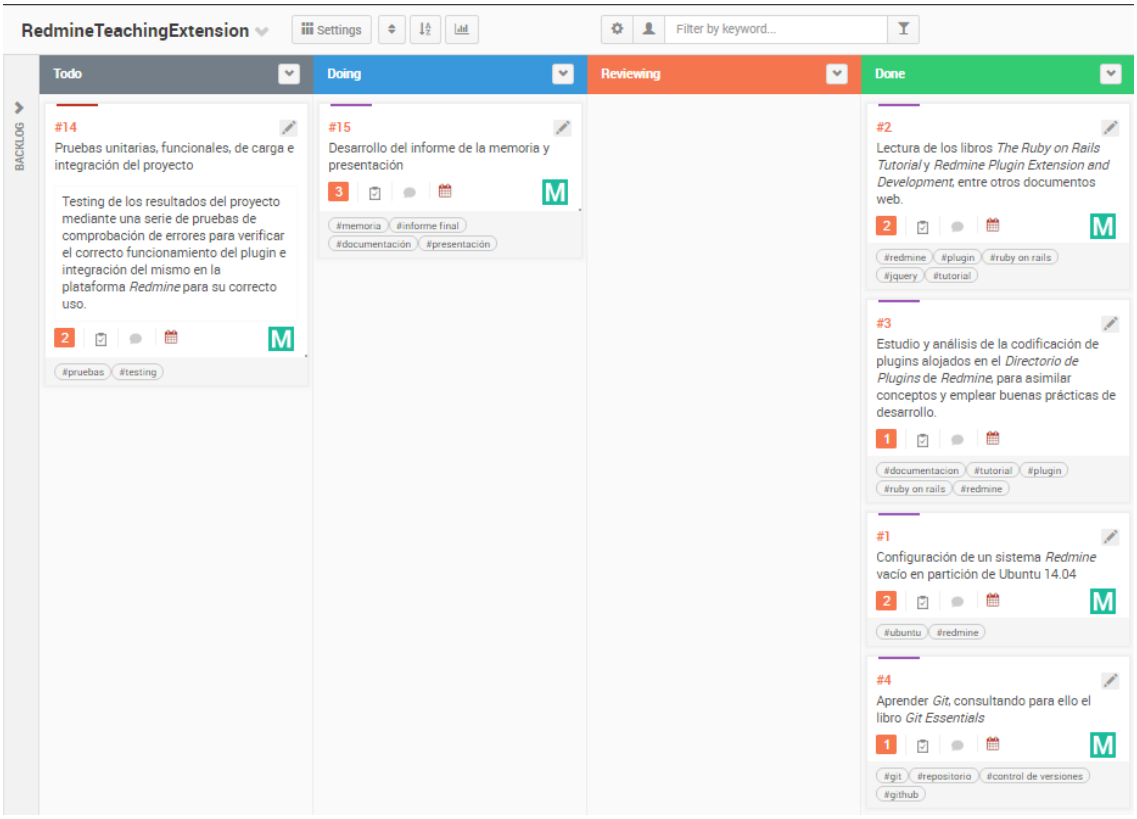

Figura 1. Tablero de tareas (Scrumboard) de ScrumDo

Así pues, el cronograma de toda la planificación temporal con el Ciclo de Vida del software al completo, quedaría de la siguiente manera:

	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1	✓	Análisis y planificación	83 días	lun 03/08/15	mié 25/11/15
2	✓	Modelado y diseño	6 días	lun 14/12/15	lun 21/12/15
3	✓	Documentación del Anteproyecto	7 días?	lun 22/02/16	mar 01/03/16
4	✓	Codificación del proyecto	97 días	jue 17/03/16	vie 29/07/16
5	✓	Pruebas del software	6 días?	lun 15/08/16	lun 22/08/16
6	✓	Documentación	30 días	lun 25/07/16	vie 02/09/16

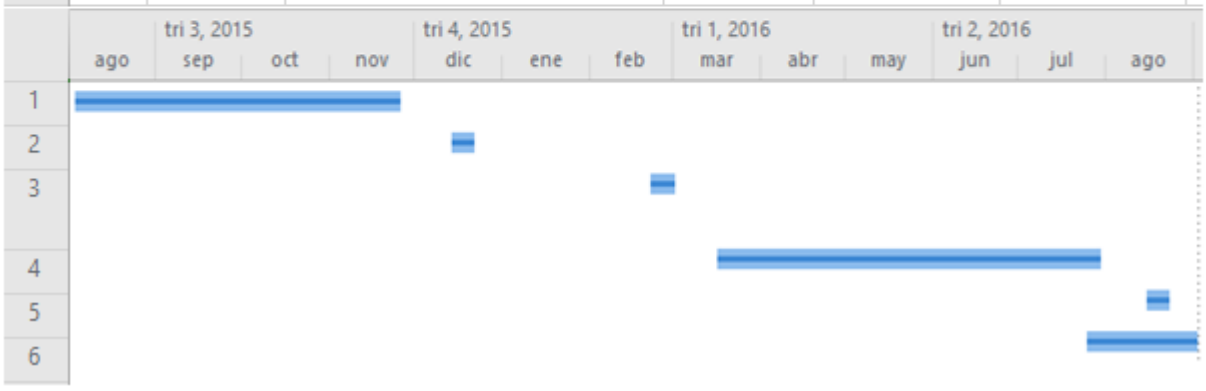

Figura 2. Cronograma de la planificación temporal con Microsoft Project

1.5. Estructura de la memoria

La memoria del *Trabajo Fin de Grado*, presenta la siguiente estructura:

- ✓ **Introducción:** Esta sección presenta un resumen sobre el contenido del proyecto, la motivación para la realización del mismo, un análisis *DAFO* (Debilidades, Amenazas, Fortalezas y Oportunidades) de su situación y la planificación temporal llevada a cabo durante el desarrollo.
- ✓ **Estado del arte:** Breve análisis de los diferentes plugins existentes en el *Directorio de* plugins de *Redmine* y comparativa con el *plugin* que se pretende desarrollar, así como conclusiones finales sobre el propósito del desarrollo del proyecto.
- ✓ **Materiales, herramientas de soporte al desarrollo y tecnologías:** Descripción detallada del conjunto de herramientas y tecnologías de desarrollo software que se han elegido para la correcta consecución del proyecto. Se procede comentando los principales entornos de desarrollo empleados, para luego pasar a las herramientas de soporte al desarrollo y terminar con el conjunto de tecnologías de desarrollo que permiten llevar a cabo la correcta implementación del proyecto.
- ✓ **Redmine Teaching Extension:** Esta sección presenta una introducción del funcionamiento del proyecto y de su arquitectura del software, así como una especificación preliminar del mismo, la descripción del conjunto de objetivos y requisitos funcionales y no funcionales del sistema (*Documento de Requisitos del Sistema – DRS*), la definición de los diagramas de secuencias (*Análisis Jerárquico de Tareas – HTA*), diagrama de clases/diseño del sistema y análisis de la implementación llevada a cabo.
- ✓ **Pruebas:** Realización de las pruebas de testing pertinentes para comprobar todo tipo de errores que pudieran surgir durante la ejecución del proyecto y verificar así el correcto funcionamiento del *plugin* y su integración en *Redmine*.
- ✓ **Resultados:** Muestra ilustrativa de los distintos resultados obtenidos tras ejecutar las funcionalidades implementadas.
- ✓ **Conclusiones y trabajo futuro:** Revisión final de las conclusiones obtenidas tras desarrollar con éxito el proyecto y planteamiento de las posibles mejoras que se pudieran aplicar de cara al futuro.
- ✓ **Bibliografía:** Listado de referencias bibliográficas y webgrafía empleadas durante la consulta de información necesaria para la realización del proyecto.

2. Estado del arte

2.1. Introducción

En informática, una extensión, complemento o *plugin* es una aplicación software que se relaciona con otra para agregarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la *API* [4].

La plataforma de gestión de proyectos *Redmine*, es un software libre y de código abierto y, además, ofrece soporte a plugins para extender su funcionalidad. Por ello, provee de total libertad a los usuarios de la plataforma para que puedan desarrollar y registrar dichos plugins en un directorio que la propia plataforma ha puesto a disposición de todos aquellos que deseen descargarlos e instalarlos para satisfacer las necesidades que no les ofrece *Redmine* de manera predeterminada [5].

Así pues, se van a comentar resumidamente algunos de estos plugins que presentan una apariencia similar al *plugin* a desarrollar en este proyecto y, los cuales, se usarán como modelo a seguir para llevar a cabo una correcta implementación e integración del mismo en la plataforma.

2.2. Redmine Checklists Plugin

Redmine Checklists Plugin [6] es una extensión que proporciona la funcionalidad de agregar un nuevo registro al formulario de peticiones de *Redmine* consistente en una lista de comprobación (*checklist*) de tareas. Estos checklists pueden ser creados desde la página de nueva petición de *Redmine*, editados desde la página de edición de la petición implicada y validados (marcados o desmarcados) desde la página de vista de la petición. Además, los diferentes cambios de los checklists, se almacenan en el registro del historial de peticiones.

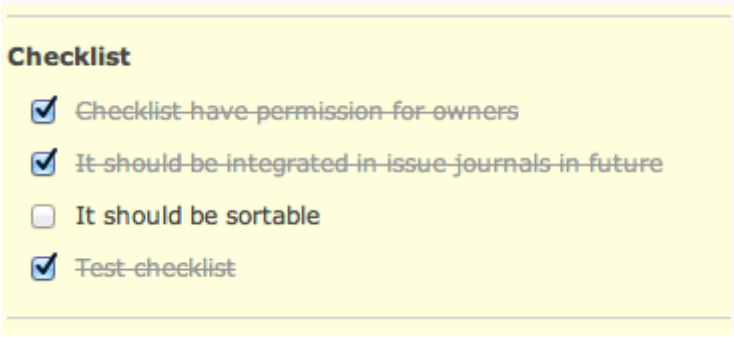

Figura 3. Muestra del plugin Redmine Checklists Plugin desde la página de vista de petición

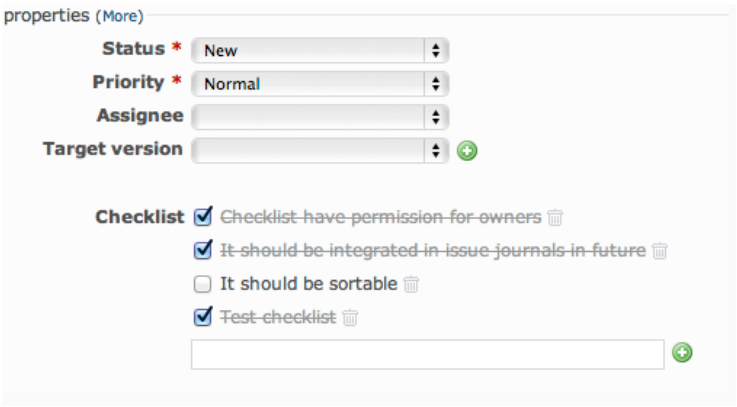

Figura 4. Muestra del plugin Redmine Checklists Plugin desde la página de edición/nueva petición

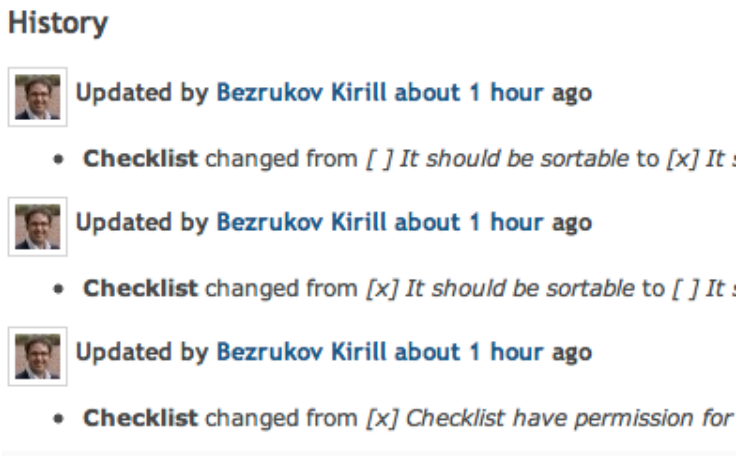

Figura 5. Muestra del plugin Redmine Checklists Plugin desde la página del historial de peticiones

Este *plugin* sirvió de gran ayuda para proceder con el desarrollo del proyecto siguiendo unas buenas prácticas, ya que con *Redmine Checklists Plugin*, la página de edición y nueva petición de *Redmine* presenta modificaciones similares en su interfaz a las que se pretende obtener con el desarrollo de *Redmine Teaching Extension*.

2.3. Redmine Multiprojects Issue Plugin

Redmine Multiprojects Issue Plugin [7] es una extensión que permite al usuario de *Redmine* especificar más de un proyecto por petición realizada. Se basa en el concepto de las peticiones realizadas entre proyectos cruzados (cross-project), las cuales permiten mantener una relación entre sí, aun perteneciendo a proyectos diferentes de *Redmine*. Por ello, estas peticiones pueden ser accesibles por cada uno de los usuarios que tengan suficientes permisos en al menos uno de los proyectos afectados, pero sólo pueden ser editadas/actualizadas por aquellos usuarios que tengan suficientes permisos en el proyecto principal (el proyecto empleado para crear la petición en cuestión).

Como se puede observar, se trata de un concepto cercano al que se pretende reflejar en el desarrollo de *Redmine Teaching Extension*, ya que mientras que uno se basa en la creación de una única petición con varios proyectos asignados en la misma, el otro (*Redmine Teaching Extension*) se basa en la creación de una petición desde un proyecto padre y la posterior propagación/creación múltiple de la misma entre los subproyectos de dicho proyecto padre. Por ello, este *plugin* sirvió de

gran ayuda e inspiración para asimilar conceptos muy útiles y necesarios durante el transcurso del desarrollo de *Redmine Teaching Extension*.

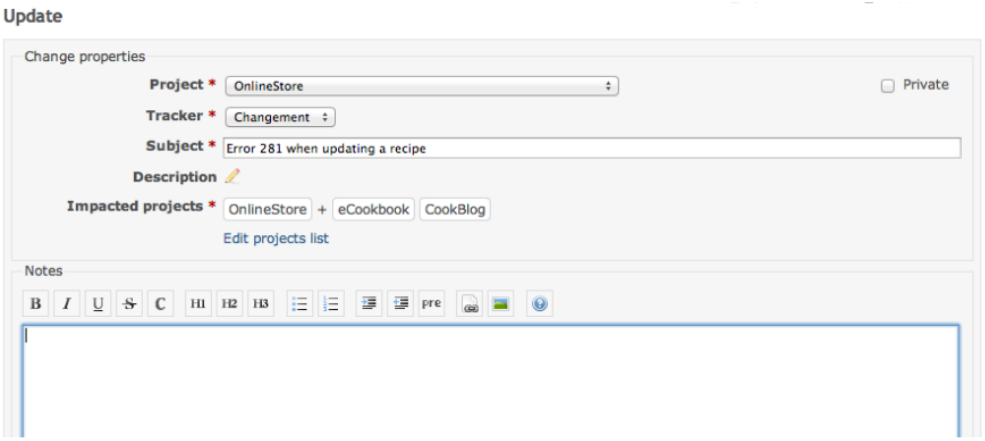

Figura 6. Muestra del plugin Multiprojects Issue Plugin desde la página de edición/nueva petición

2.4. Conclusiones sobre el propósito de Redmine Teaching Extension

Si bien anteriormente se han descrito los principales plugins que han servido de inspiración y modelo para el desarrollo de este proyecto, hay que mencionar que aún no hay información conocida en internet sobre la existencia de algún *plugin* que tenga un propósito similar al de *Redmine Teaching Extension* (crear peticiones en un proyecto raíz y propagarlas simultáneamente entre sus subproyectos), ya que según se ha comprobado en los foros de *Redmine* [8], se trata de una necesidad poco demandada entre la comunidad de *Redmine* por lo que se ha descartado añadir esta funcionalidad en actualizaciones próximas de la plataforma.

Así pues, este hecho hace del proyecto *Redmine Teaching Extension* un producto software único e innovador ya que, aunque va destinado, a priori, a un propósito muy concreto como es el soporte a la docencia y a un público muy reducido como es el personal docente que utiliza *Redmine* en determinadas asignaturas, igualmente no existe un producto registrado oficialmente en el directorio de plugins de *Redmine* que presente una funcionalidad igual.

3. Materiales, herramientas de soporte al desarrollo y tecnologías

3.1. Introducción

En este apartado se van a mostrar el conjunto de materiales imprescindibles para el desarrollo del proyecto (características principales del ordenador personal, herramienta de diseño y modelado, entorno de desarrollo integrado para la implementación del proyecto…), las herramientas de soporte al desarrollo (sistema de control de versiones, metodología ágil para la gestión del proyecto…), así como las tecnologías implicadas en el diseño, modelado y codificación del proyecto.

Este último subapartado correspondiente a las diferentes tecnologías implicadas, abarca desde el lenguaje de modelado del software (*UML*) pasando por el lenguaje de marcas y maquetado de la aplicación (*HTML* y *CSS*), la biblioteca *jQuery* de *JavaScript* que permite llevar a cabo la interacción mediante determinadas funciones con los formularios *HTML* implicados en el contenido del proyecto, hasta el sistema de gestión de base de datos relacional *MySQL* y el framework de aplicaciones web *Ruby on Rails* (que adopta un patrón Modelo-Vista-Controlador (*MVC*)) que conforman la plataforma *Redmine* y cuyos requerimientos hay que respetar para llevar a cabo la correcta y eficiente codificación del *plugin*.

3.2. Materiales

Los materiales agrupan a todas aquellas aplicaciones, entornos y herramientas que permiten llevar a cabo la realización de las distintas fases del Ciclo de Vida del software (*Análisis y Planificación, Modelado y Diseño, Implementación, Pruebas y Documentación*).

3.2.1. Ordenador portátil personal

Como base para todas las herramientas de soporte y desarrollo del proyecto, se ha usado un ordenador portátil personal de gama media con las siguientes características principales:

- Ordenador portátil *ASUS K550CC-XX1173H* de pantalla de 15.6'' 1366 x 768 con tecnología LED
- Disco duro de 500 GB
- Memoria RAM de 4 GB a 1600 MHz con tecnología ddr3 sdram
- Procesador Intel Core i7 3537U a 2 GHz
- Tarjeta gráfica NVIDIA GeForce GT 720M de 2 GB
- Sistema operativo Ubuntu 14.04

3.2.2. Visual Paradigm For UML

Visual Paradigm For UML (*VP4UML*) [9] es una herramienta CASE (Ingeniería del Software Asistida por Computadora) que soporta el lenguaje unificado de modelado *UML*, por lo que su utilización va a ser esencial para abarcar la fase de *Modelado y Diseño* del Ciclo de Vida del software.

Por un lado, va a permitir la creación de los diferentes diagramas de casos de uso del sistema que definen esquemáticamente el comportamiento del sistema desarrollado al afrontar determinados requisitos funcionales. Este aspecto se tratará más delante y detenidamente en el apartado de *Requisitos identificados en el sistema* del *Documento de Requisitos del Sistema* (*DRS*).

Por otro lado, va a permitir la creación de los diferentes diagramas de secuencias, empleados para modelar la interacción entre el conjunto de objetos de un sistema a través del tiempo. Este aspecto se tratará más adelante y detenidamente en el apartado del *Análisis Jerárquico de Tareas* (*HTA*).

Por último, va a permitir la creación del diseño del sistema en términos de clases mediante el diagrama de clases consistente en una estructura estática que describe la estructura del sistema desarrollado mostrando para ello las clases del sistema (en este caso, las clases se definen en términos de modelos, vistas y controladores implicados, ya que se sigue un patrón de arquitectura del software Modelo-Vista-Controlador, como se ha comentado anteriormente). Este tratamiento del diagrama de clases del sistema se tratará más adelante y detenidamente en un apartado propio.

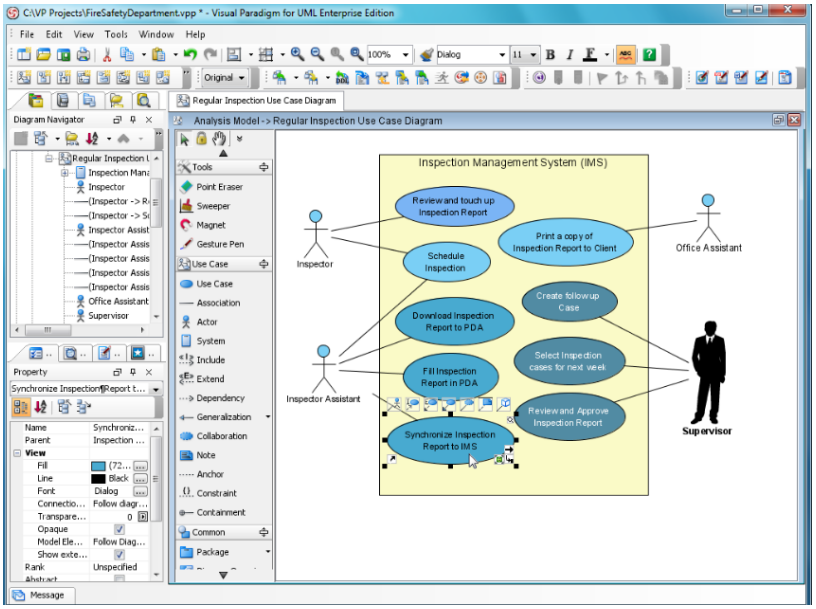

Figura 7. Ejemplo del modelado de un diagrama de casos de uso en VP4UML

3.2.3. Inspector de página de Mozilla Firefox

Durante el desarrollo de las vistas del proyecto se empleó el *Inspector de página de Mozilla Firefox* [10]. Esta herramienta del navegador web *Mozilla Firefox* permite llevar a cabo la edición del *Front-End* (interfaz web) de las aplicaciones web mediante la edición manual de su código fuente en el navegador, permitiendo así visualizar en el mismo instante los cambios implementados en las vistas del proyecto. Así pues, esto se ha empleado como ayuda para mejorar la configuración y correcta integración de la interfaz del *plugin* en el entorno de *Redmine* e ir mejorando así el prototipo final de las vistas del producto final del proyecto.

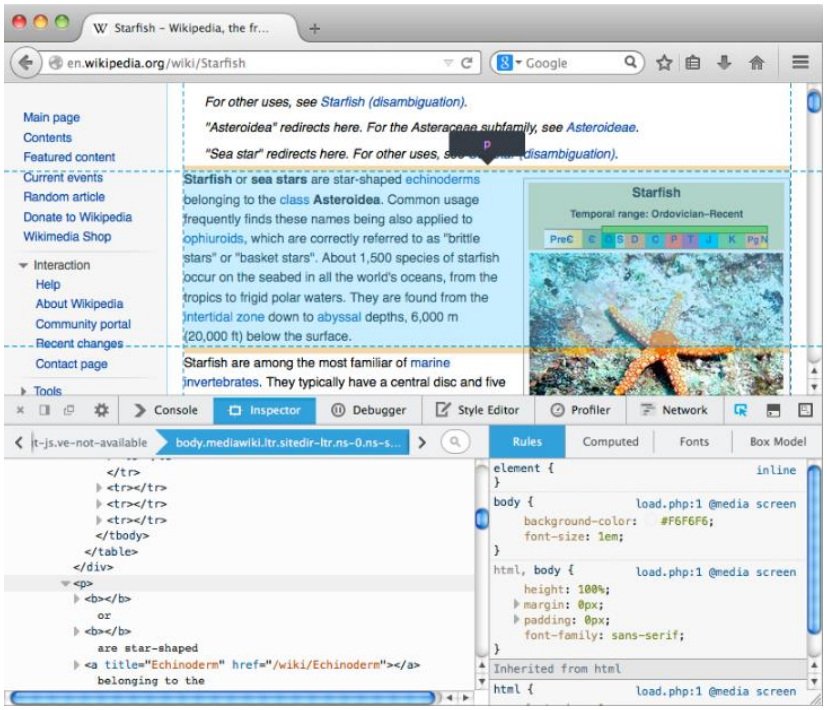

Figura 8. Ejemplo de edición mediante el Inspector de página de Mozilla Firefox

3.2.4. Entorno de desarrollo integrado (IDE) RubyMine

La fase del Ciclo de Vida del software correspondiente a la *Implementación* del proyecto, va a ser llevada a cabo mediante el entorno de desarrollo integrado (*IDE*) *RubyMine* [11].

Se trata del entorno de desarrollo integrado que ofrece el soporte más completo al framework web *Ruby on Rails* (en el cual se basa el desarrollo del sistema *Redmine* y, por tanto, el *plugin* a desarrollar) así como su combinación con el lenguaje *Ruby*, la biblioteca *jQuery* de *JavaScript*, los lenguajes de marcas y maquetado (*HTML* y *CSS*) y el sistema de gestión de bases de datos *MySQL*. Así pues, permite la integración de todas las tecnologías necesarias para llevar a cabo el proceso de codificación del proyecto *Redmine Teaching Extension*.

Figura 9. Logotipo de RubyMine

3.3. Herramientas de soporte al desarrollo

Para facilitar todo el proceso de desarrollo y gestión del producto software que se va a crear, existen herramientas de soporte como son *ScrumDo*, para el seguimiento y control de las tareas mediante una gestión ágil del proyecto y *GitHub*, que es un repositorio de código fuente basado en el sistema de control de versiones *Git*.

3.3.1. Metodología Ágil Scrum mediante ScrumDo

La planificación temporal del proyecto, abarca cada una de las actividades del Ciclo de Vida del software (*Análisis y Planificación, Modelado y Diseño, Implementación, Pruebas y Documentación*). Por ello, resulta necesario llevar a cabo una gestión y planificación de todas las tareas que forman parte de cada una de estas fases del Ciclo de Vida para tener constancia en todo momento de que se cumplen cada una de las iteraciones fijadas para cada tarea en el tiempo previsto.

Como bien se comentó anteriormente en el apartado de *Introducción*, se ha diseñado un tablero de tareas o *Scrumboard* en la herramienta web *ScrumDo* [12], donde se van colocando diferentes tarjetas correspondientes a las tareas que forman parte de la planificación temporal. Cada una de estas tareas/tarjetas, presentan una definición de las mismas, un nivel de dificultad y su estado actual (*Para hacer*, *En progreso* o *Finalizada*).

3.3.2. Sistema de control de versiones mediante GitHub

Para llevar a cabo el control de las versiones del código fuente del proyecto, se ha empleado *GitHub* [13], un sistema de control colaborativo de versiones que hace la función de repositorio del código fuente y que está basado en el software de control de versiones *Git*.

El repositorio presenta una configuración de una única rama de desarrollo, que es la rama principal por defecto, llamada rama *master*. En esta rama, se irán guardando los cambios (*commits*) que se vayan realizando a lo largo del desarrollo del proyecto. A continuación, se va a mostrar una captura de los últimos *commits* realizados durante el desarrollo del proyecto *Redmine Teaching Extension*:

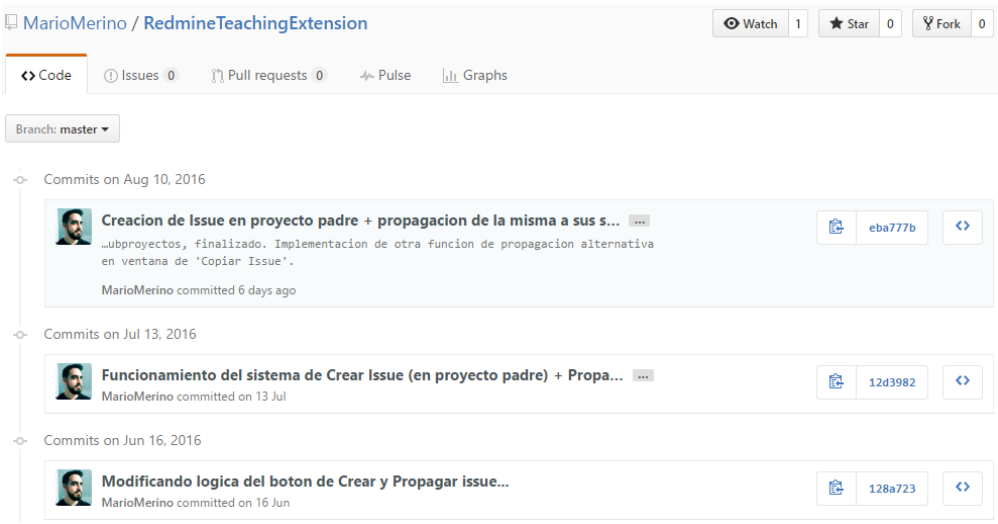

Figura 10. Muestra de últimos commits del proyecto en GitHub

3.4. Tecnologías

En este apartado, se van a explicar detalladamente las distintas tecnologías implicadas en las fases de diseño, modelado y codificación del proyecto.

En primer lugar, se explicará el lenguaje de modelado del software *UML* que se empleará para la creación de los diagramas de casos de uso, clases y secuencias del sistema.

En segundo lugar, se procederá con la explicación del lenguaje de marcas y maquetado de la aplicación (*HTML* y *CSS*) empleado para la creación y maquetación del sistema.

En tercer lugar, se hará una explicación de la biblioteca *jQuery* de *JavaScript* que permite llevar a cabo la interacción (mediante determinadas funciones) con el maquetado del proyecto anteriormente mencionado.

En cuarto lugar, se comentarán los principales aspectos del framework de aplicaciones web *Ruby on Rails*, empleado para poder llevar a cabo la correcta implementación del proyecto.

En quinto lugar, se describirá la *API REST* de *Redmine*, la cual proporciona importantes datos que ofrecen acceso y operaciones básicas *CRUD* (creación, actualización y borrado) para los diferentes recursos y funcionalidades que ofrece la plataforma.

Por último, se explicará el propósito de *MySQL*, el sistema de gestión de bases de datos relacional con el que trabaja *Redmine* y que, por tanto, está implicado en el proyecto a desarrollar.

3.4.1. UML

Lenguaje unificado de modelado (*UML* – Unified Modeling Language) [14] es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Está respaldado por el *OMG* (Object Management Group).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. *UML* ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Hay que remarcar que *UML* es un lenguaje de modelado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. Se trata pues del lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o *RUP*), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, ya que su significado es *Lenguaje Unificado de Modelado*, no es programación, solo se modela la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, la programación orientada a objetos viene siendo un complemento perfecto de *UML*, pero no por eso se toma *UML* sólo para lenguajes orientados a objetos.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

3.4.2. HTML

HTML (*Hypertext Markup Language*) [15] hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código *HTML*) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del *World Wide Web Consortium* (*W3C*) o *Consorcio WWW*, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la *World Wide Web* (*WWW*). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

- Un lenguaje de marcas es un conjunto de etiquetas de marcado.
- Los documentos *HTML* son descritos por las etiquetas *HTML*.
- Cada etiqueta *HTML* describe diferentes contenidos de documentos.

3.4.2.1. Elementos

Los elementos son la estructura básica de *HTML*. Los elementos tienen dos propiedades básicas: atributos y contenido. Cada atributo y contenido tiene ciertas restricciones para que se considere válido al documento *HTML*. Un elemento generalmente tiene una etiqueta de inicio (por ejemplo, `<nombre-de-elemento>`) y una etiqueta de cierre (por ejemplo, `</nombre-de-elemento>`). Los atributos del elemento están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas (por ejemplo, `<nombre-de-elemento atributo=”valor”>Contenido</nombre-de-elemento>`). Algunos elementos, tales como `<br>`, no tienen contenido ni llevan una etiqueta de cierre. Debajo se listan varios tipos de elementos de marcado usados en *HTML*.

El *marcado estructural* describe el propósito del texto. Un ejemplo de ello puede ser `<h2>Golf</h2>` el cual establece «Golf» como un encabezamiento de segundo nivel.

El *marcado presentacional* describe la apariencia del texto, sin importar su función. Por ejemplo, `<b>negrita</b>` indica que los navegadores web visuales deben mostrar el texto en **negrita**, pero no indica qué deben hacer los navegadores web que muestran el contenido de otra manera.

El *marcado hipertextual* se utiliza para enlazar partes del documento con otros documentos o con otras partes del mismo documento. Para crear un enlace es necesario utilizar la etiqueta de ancla `<a>` junto con el atributo `href`, que establecerá la dirección *URL* a la que apunta el enlace. Por ejemplo, un enlace que muestre el texto de *Redmine* y enlace a su página podría ser de la forma `<a href=”http://www.redmine.org”>Redmine</a>`. También se pueden crear enlaces sobre otros objetos, tales como imágenes `<a href=”enlace”><img src=”imagen” /></a>`

3.4.2.2. Atributos

La mayoría de los atributos de un elemento son pares nombre-valor, separados por un signo de igual «=» y escritos en la etiqueta de comienzo de un elemento, después del nombre del elemento. El valor puede estar rodeado por comillas dobles o simples, aunque ciertos tipos de valores pueden estar sin comillas.

Un ejemplo de atributo, puede ser lo explicado anteriormente en el *marcado hipertextual*. En `<a href="http://www.redmine.org">Redmine</a>` el enlace viene definido por la etiqueta `<a>`, mientras que la dirección del enlace viene especificado por el atributo *href*.

Otro ejemplo común, es el de las imágenes en *HTML*, las cuales vienen definidas por la etiqueta `<img>`, el nombre de archivo de la fuente de la imagen viene definido por *src* y el tamaño de la imagen viene definido por *width* (anchura) y *height* (altura). A continuación, se puede ver un ejemplo de ello: `<img src="imagen.jpg" width="104" height="150">`. El tamaño de la imagen viene especificado en píxeles: `width="104"`, hace referencia a 104 píxeles de ancho.

3.4.3. CSS

La hoja de estilo en cascada o *CSS* (*Cascading Style Sheets*) [16] es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en *HTML* o *XML*. El *World Wide Web Consortium* (*W3C*) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

La información de estilo puede ser definida en un documento separado o en el mismo documento *HTML*. En este último caso podrían definirse estilos generales con el elemento *style* o en cada etiqueta particular mediante el atributo *style*.

Cabe destacar que *CSS* ahorra mucho trabajo, ya que puede controlar el diseño de varias páginas web a la vez. Las hojas de estilo externas se almacenan en archivos *CSS* (con extensión *.css*).

3.4.3.1. Sintaxis y selectores

Un conjunto de reglas de *CSS* consta de un selector y un bloque de declaración, como se puede observar en la siguiente figura:

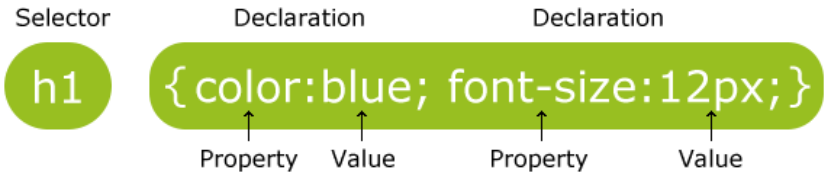

Figura 11. Ejemplo de bloque de declaración de CSS

- El selector apunta al elemento *HTML* al que se desea aplicar el estilo.
- El bloque de declaración contiene una o más declaraciones separadas por punto y coma.
- Cada declaración incluye un nombre de propiedad *CSS* y un valor, separados por dos puntos.

- Una declaración *CSS* siempre termina por punto y coma, y los bloques de declaración están rodeados por llaves.

Los selectores *CSS* son usados para "encontrar" (o seleccionar) los elementos *HTML* en función de su nombre de elemento, id, clase, atributo, entre otros.

Un ejemplo, puede ser la selección de todos los elementos `<p>` de una página, en la cual dichos elementos presentarán el texto alineado en el centro y su fuente será color rojo:

```
p {
    text-align: center;
    color: red;
}
```

Figura 12. Ejemplo 1 de bloque de declaración de CSS

Un caso muy empleado en el proyecto es el del selector *id*. Este selector utiliza el atributo de identificación de un elemento *HTML* para seleccionar un elemento específico. La identificación de un elemento debe ser única dentro de una página, por lo que el selector *id* se emplea para seleccionar un único elemento.

Así pues, para seleccionar un elemento con un *id* específico, hay que escribir una almohadilla (*#*) seguida del *id* del elemento en cuestión, tal y como se muestra en el siguiente ejemplo donde se aplicará el mismo estilo indicado en el ejemplo anterior para aquellos elementos *HTML* con *id="para1"*:

```
#para1 {
    text-align: center;
    color: red;
}
```

Figura 13. Ejemplo 2 de bloque de declaración de CSS

3.4.4. jQuery

jQuery [17] es una biblioteca de *JavaScript*, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos *HTML*, manipular el árbol *DOM* (interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos *HTML*), manejar eventos, desarrollar animaciones y agregar interacción con la técnica *AJAX* a páginas web. Esto hace de *jQuery* la biblioteca de JavaScript más utilizada.

Se trata de software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de *GNU v2*, permitiendo su uso en proyectos libres y privados. *jQuery*, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en *JavaScript* que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Por lo anteriormente comentado, sobra decir que es necesario tener conocimientos básicos previos de *HTML*, *CSS* y *JavaScript* para afrontar *jQuery*.

Figura 14. Logotipo de jQuery

3.4.4.1. Sintaxis y selectores

Con *jQuery* se consultan (*query*) los elementos *HTML* deseados y se realizan "acciones" sobre ellos. Así pues, su sintaxis está hecha a medida para este propósito.

Su sintaxis básica es: `$(selector).action()`

- Un signo `$` para definir/acceder a *jQuery*.
- Un `(selector)` para "consultar (o encontrar)" elementos *HTML*.
- Una `.action()` (acción) que va a ser ejecutada sobre el/los elemento/s.

Al igual que en *CSS*, los selectores son usados para "encontrar (o seleccionar) los elementos *HTML* en función de sus nombres de elemento, id, clases, tipos, atributos, valores de los atributos, entre otros. Por esto, está basado en selectores *CSS* existentes y, además, cuenta con algunos selectores personalizados propios.

Ejemplos de selectores pueden ser los siguientes:

- `$("p")`, para seleccionar todos los elementos `<p>` de una página.
- `$(#test)`, donde la almohadilla (*#*) hace referencia a un elemento con *id* que debería ser único en una página, en este caso, *id="test"*.

3.4.4.2. Eventos

jQuery está hecha a medida para responder a diferentes eventos en una página *HTML*. Todas las diferentes acciones de los visitantes a los que una página web puede responder son llamados eventos. Un evento representa, por tanto, el momento preciso en que algo sucede. Ejemplos de ello pueden ser: mover el ratón sobre un elemento, seleccionar un *radio button*, hacer clic sobre un elemento…

El término *"dispara/disparado"* es a menudo usado con los eventos. Por ejemplo: "El evento de pulsación de tecla es disparado en el momento que se pulsa una tecla".

Aquí se presentan algunos de los eventos más comunes:

Eventos de ratón	Eventos de teclado	Eventos de formulario	Eventos de documento/ventana
click	*keypress*	*submit*	*load*
dblclick	*keydown*	*change*	*resize*
mouseenter	*keyup*	*focus*	*scroll*
mouseleave		*blur*	*unload*

Tabla 1. Eventos comunes de jQuery

En *jQuery*, la mayoría de los eventos tienen un método de *jQuery* equivalente. En el caso de asignar un evento de clic a todos los párrafos en una página, se puede hacer lo siguiente:

```
$("p").click();
```

El siguiente paso es definir lo que ocurriría cuando el evento se dispara. Se debe pasar una función al evento en cuestión:

```
$("p").click(function(){
   // aquí va la acción...
});
```

3.4.5. Ruby on Rails

Ruby on Rails [18], también conocido como *RoR* o *Rails*, es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación *Ruby*, siguiendo el paradigma del patrón Modelo-Vista-Controlador (*MVC*). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación *Ruby* permite la metaprogramación, de la cual *Rails* hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. *Rails* se distribuye a través de *RubyGems*, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones *Ruby*. Estas bibliotecas están destinadas a resolver los problemas más comunes a la hora de desarrollar una aplicación web, para que el programador pueda concentrarse en los aspectos únicos y diferenciales del proyecto en lugar de los problemas recurrentes.

Figura 15. Logotipo de Ruby on Rails

3.4.5.1. Características

Los principios fundamentales de *Ruby on Rails* incluyen el principio *No te repitas* (del inglés *Don't repeat yourself*, *DRY*) y *Convención sobre Configuración* (paradigma de programación de software que busca minimizar el número de decisiones que un desarrollador necesita hacer, ganando así en simplicidad pero no perdiendo flexibilidad por ello).

No te repitas significa que las definiciones deberían hacerse una sola vez. Dado que *Ruby on Rails* es un framework de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Por ejemplo, en *ActiveRecord* (patrón de arquitectura encontrado en aplicaciones que almacenan sus datos en Bases de Datos relacionales), las definiciones de las clases no necesitan especificar los nombres de las columnas. Esto se debe a que *Ruby* puede averiguarlos a partir de la propia base de datos, de forma que definirlos tanto en el código como en el programa sería redundante.

Convención sobre configuración significa que el programador sólo necesita definir aquella configuración que no es convencional. Por ejemplo, si hay una clase `Historia` en el modelo, la tabla correspondiente de la base de datos es `historias`, pero si la tabla no sigue la convención (por ejemplo `blogposts`) debe ser especificada manualmente (`set_table_name "blogposts"`). Así, cuando se diseña una aplicación partiendo de cero sin una base de datos preexistente, el seguir las convenciones de Rails significa usar menos código.

En cuanto a la historia de este framework de aplicaciones web, hay que destacar que fue escrito por David Heinemeier Hansson a partir de su trabajo en *Basecamp*, una herramienta de gestión de proyectos, por la compañía *37signals*. Fue liberado al público por primera vez en julio de 2004 y desde entonces ha sido extendido por el *Rails core team*, más de 2100 colaboradores y soportado por una extensa y activa comunidad.

Sus principales aplicaciones actuales de internet son, además de la nombrada *Basecamp*, *Twitter*, *GitHub*, *Soundcloud*, o la plataforma para la gestión de proyectos *Redmine* en torno a la cual, como ya se ha comentado, se va a basar el desarrollo de este proyecto (*Redmine Teaching Extension*).

Por otro lado, en cuanto a soporte de servidores Web y Bases de Datos, se puede comentar lo siguiente:

Soporte de servidores Web

Para desarrollo y pruebas, se utiliza *Mongrel* o *WEBrick*, incluido con *Ruby*. Para utilizar *Rails* en servidores en producción se está extendiendo el uso de *Passenger* de *Apache* (cabe destacar que es el empleado en el *Redmine* vacío sobre el que se está realizando las diferentes pruebas con *Redmine Teaching Extension*), desarrollado en 2008 por la empresa holandesa *Phusion*. Otras opciones para producción son *Nginx*, *Mongrel*, *Apache*, *Lighttpd* con *FastCGI* o alguna combinación de ambos (por ejemplo utilizando *Apache* como proxy para los procesos *Mongrel*).

Soporte de Bases de Datos

Dado que la arquitectura *Rails* favorece el uso de bases de datos, se recomienda usar un *SGBDR* (*Sistema de Gestión de Bases de Datos Relacionales*) para almacenamiento de datos. *Rails* soporta la biblioteca *SQLite* por defecto. El acceso a la base de datos es totalmente abstracto desde el punto de vista del programador, es decir que es agnóstico a la base de datos, y *Rails* gestiona los accesos a la base de datos automáticamente (aunque, si se necesita, se pueden hacer consultas directas en SQL). *Rails* intenta mantener la neutralidad con respecto a la base de datos, la portatibilidad de la aplicación a diferentes sistemas de base de datos y la reutilización de bases de datos preexistentes. Sin embargo, debido a la diferente naturaleza y prestaciones de los *SGBDRs* el framework no puede garantizar la compatibilidad completa. Se soportan diferentes *SGBDRs*, incluyendo *MySQL* (que será el empleado en el caso de *Redmine* para el trabajo de *Redmine Teaching Extension*), *PostgreSQL*, *SQLite*, *IBM DB2* y *Oracle*.

3.4.5.2. Arquitectura

Ruby on Rails sigue el paradigma del patrón de arquitectura software *Modelo-Vista-Controlador* (*MVC*) [19].

Se trata de un patrón que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello *MVC* propone la construcción de tres componentes distintos que son el *Modelo*, la *Vista* y el *Controlador*, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

Al interactuar este patrón con una aplicación de *Ruby on Rails*, un navegador envía una petición, la cual es recibida por un servidor web y transmitida a al *Controlador* de *Rails*, que es el encargado de qué hacer a continuación. En algunos casos, el *Controlador* inmediatamente representará una *Vista*, que se trata de una plantilla que es convertida a *HTML* y enviada de vuelta al navegador web. Lo más común para sitios web dinámicos, es que el *Controlador* interactúe con un *Modelo*, el cual es un objeto de *Ruby* que representa un elemento del sitio web (como un usuario) y se encarga de la comunicación con la base de datos. Después de invocar al *Modelo*, el *Controlador* representa la *Vista* y devuelve la página web completa al navegador web como *HTML*.

Una representación esquemática de la funcionalidad de la arquitectura *Modelo-Vista-Controlador* (*MVC*), puede ser la siguiente:

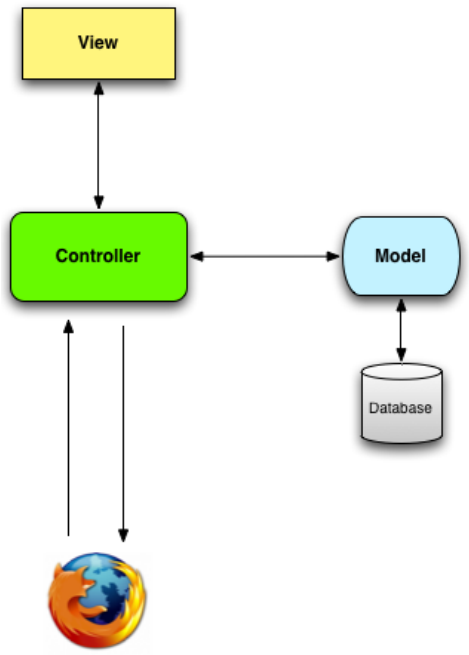

Figura 16. Representación esquemática de la arquitectura MVC

3.4.5.2.1. Modelo

El *Modelo* es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la *Vista* aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al *Modelo* a través del *Controlador*.

Los archivos de *Modelo* en *Ruby on Rails* presentan las siguientes características:

- Pueden contener métodos de manipulación de datos.
- Pueden contener reglas de validación.
- Pueden contener declaraciones de relaciones entre tablas y objetos.
- Son encontrados en el directorio de *Modelos*.
- Son nombrados con el nombre del objeto en singular: *issue.rb*, *project.rb*…

3.4.5.2.2. Vista

La *Vista* presenta el *Modelo* (información y lógica de negocio) en un formato adecuado para interactuar (normalmente la interfaz de usuario) por lo que requiere de dicho *Modelo* la información que debe representar como salida. Cada acción dentro de un *Controlador* puede tener una *Vista* que es un archivo *HTML* (con código *Ruby* incorporado, en el caso de *Ruby on Rails*. En este caso, se tratan de archivos *html.erb*) nombrado después de la acción asociada. Cada *Controlador* puede tener su propia *Vista* nombrada con el plural del objeto al que representa: *issues.html.erb, projects.html.erb*…

Una *Vista* de *Ruby on Rails* puede ser conectada con el *Controlador* o enviada directamente al usuario. Las vistas puras de *Ruby on Rails* no tienen acceso a datos o lógica de negocio en ellas y deberían ser lo más limpias y óptimas posibles. Las *Vistas* pueden también incluir *parciales* que son plantillas *html.erb* para la presentación de fragmentos de datos concretos. Una acción puede también presentar o redireccionar a otra *Vista*.

3.4.5.2.3. Controlador

El *Controlador* responde a eventos (normalmente acciones del usuario) e invoca peticiones al *Modelo* cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede invocar a la *Vista* que tiene asociada si se solicita un cambio en la forma en que se presenta el *Modelo* (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos). Por tanto, se podría decir que el *Controlador* hace de intermediario entre la *Vista* y el *Modelo*.

Cada objeto en una aplicación de *Ruby on Rails* tiene su propio archivo *Controlador*, el cual presenta las siguientes características:

- Contiene acciones (métodos) y lógica de negocio.
- Presenta acciones de unas 3-5 líneas de código idealmente.
- Por lo general, no contiene consultas *SQL*.
- Se encuentra en el directorio de *Controladores*.
- Son nombrados con el nombre del objeto en plural: *issues.rb*, *projects.rb*…

3.4.6. Redmine REST API

REST es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar *HTTP* y que permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP. *Redmine* expone algunos de sus datos a través de una *API REST* [20]. Esta *API* proporciona acceso y operaciones básicas *CRUD* (*crear, actualizar, eliminar*) para los diferentes recursos que se definen en las siguientes tablas y es compatible con los formatos *XML* y *JSON*.

Resource	Status	Notes	Availability
Issues	Stable		1.0
Projects	Stable		1.0
Project Memberships	Alpha		1.4
Users	Stable		1.1
Time Entries	Stable		1.1
News	Prototype	Prototype implementation for `index` only	1.1
Issue Relations	Alpha		1.3
Versions	Alpha		1.3
Wiki Pages	Alpha		2.2

Tabla 2. Recursos de Redmine accesibles por la API REST (1)

Queries	Alpha		1.3
Attachments	Beta	Adding attachments via the API added in 1.4	1.3
Issue Statuses	Alpha	Provides the list of all statuses	1.3
Trackers	Alpha	Provides the list of all trackers	1.3
Enumerations	Alpha	Provides the list of issue priorities and time tracking activities	2.2
Issue Categories	Alpha		1.3
Roles	Alpha		1.4
Groups	Alpha		2.1
Custom Fields	Alpha		2.4
Search	Alpha		3.3

Tabla 3. Recursos de Redmine accesibles por la API REST (2)

Significado de los *Estados* de la tabla:

- *Stable (Estable)*: Característica completa. No hay cambios mayores planeados.
- *Beta*: Útil para integraciones con algunos errores o pérdida mínima de funcionalidad.
- *Alpha*: Funcionalidad mayor en el recurso. Necesita *feedback* de los usuarios de la *API* e integradores.
- *Planned (Planeado)*: Planeado en versiones de *Redmine* futuras, dependiendo de la disponibilidad del desarrollador.

3.4.7. MySQL

MySQL [21] es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. *MySQL* fue inicialmente desarrollado por *MySQL AB* (empresa fundada por David Axmark, Allan Larsson y Michael Widenius). *MySQL AB* fue adquirida por *Sun Microsystems* en 2008, y ésta a su vez fue comprada por *Oracle Corporation* en 2010, la cual ya era dueña desde 2005 de *Innobase Oy*, empresa finlandesa desarrolladora del motor *InnoDB* para *MySQL*. Está desarrollado bajo licencia dual *GPL/Licencia comercial* por *Oracle Corporation* y está considerada como la base datos open source más popular del mundo, y una de las más populares en general junto a *Oracle* y *Microsoft SQL Server*, sobre todo para entornos de desarrollo web.

Al contrario de proyectos como *Apache*, donde el software es desarrollado por una comunidad pública y los derechos de autor del código están en poder del autor individual, *MySQL* es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código. Esto es lo que posibilita el esquema de doble licenciamiento anteriormente mencionado. La base de datos se distribuye en varias versiones, una *Community*, distribuida bajo la *Licencia pública general de GNU*, versión, y varias versiones *Enterprise*, para aquellas empresas que quieran incorporarlo en productos privativos.

Está desarrollado en su mayor parte en *ANSI C* y *C++*. Tradicionalmente se considera uno de los cuatro componentes de la pila de desarrollo *LAMP* y *WAMP*.

MySQL es usado por muchos sitios web grandes y populares, como *Wikipedia*, *Google* (aunque no para búsquedas), *Facebook*, *Twitter*, *Flickr*, *YouTube* y, por supuesto, *Redmine* que es la plataforma que va a servir de soporte para el desarrollo de este proyecto (*Redmine Teaching Extension*).

Figura 17. Logotipo de MySQL

3.4.7.1. Características

Inicialmente, *MySQL* carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas web con contenido dinámico, justamente por su simplicidad.

Poco a poco los elementos de los que carecía *MySQL* están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre. Entre las características disponibles en las últimas versiones se puede destacar:

- Amplio subconjunto del lenguaje *SQL*. Algunas extensiones son incluidas igualmente.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Posibilidad de selección de mecanismos de almacenamiento que ofrecen diferentes velocidades de operación, soporte físico, capacidad, distribución geográfica, transacciones…

- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

MySQL es un sistema de administración de bases de datos. Una base de datos es una colección estructurada de tablas que contienen datos. Para agregar, acceder a y procesar datos guardados en un computador, se necesita un administrador como *MySQL Server*. Dado que los computadores son muy buenos manejando grandes cantidades de información, los administradores de bases de datos juegan un papel central en computación, como aplicaciones independientes o como parte de otras aplicaciones.

MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

MySQL es software de código abierto. Fuente abierta significa que es posible para cualquier persona usarlo y modificarlo. Cualquier persona puede bajar el código fuente de *MySQL* y usarlo sin pagar. Cualquier interesado puede estudiar el código fuente y ajustarlo a sus necesidades. *MySQL* usa el *GPL* (*GNU General Public License*) para definir qué puede hacer y qué no puede hacer con el software en diferentes situaciones.

Por último, es necesario destacar el conjunto de características implementadas únicamente por *MySQL*:

- Permite escoger entre múltiples motores de almacenamiento para cada tabla. En *MySQL 5.0* éstos debían añadirse en tiempo de compilación, a partir de *MySQL 5.1* se pueden añadir dinámicamente en tiempo de ejecución. Estos pueden ser: *nativos*, como *MyISAM*, *Falcon*, *Merge*, *InnoDB*…; *desarrollados por partners*, como *solidDB*, *NitroEDB*, *ScaleDB*, *TokuDB*…; y *desarrollados por la comunidad*, como *memcache*, *httpd*, *PBXT* y *Revision*.
- Agrupación de transacciones, reuniendo múltiples transacciones de varias conexiones para incrementar el número de transacciones por segundo.

4. Redmine Teaching Extension

4.1. Introducción

En este apartado se va a llevar a cabo una explicación detallada del funcionamiento del proyecto *Redmine Teaching Extension*. En primer lugar y a modo introductorio, se comentará la arquitectura software del mismo. Luego, se explicará más a fondo la funcionalidad mediante la redacción de una especificación preliminar de sus diferentes características. Posteriormente, se pasará a la explicación y análisis de las fases de *Modelado y Diseño* e *Implementación* del Ciclo de Vida del software, que abarcan desde la descripción del *Documento de Requisitos del Sistema* (*DRS*) hasta la explicación de la estructura del código desarrollado, pasando por el *Análisis Jerárquico de Tareas* (*HTA*) consistente en los distintos diagramas de secuencias y guía de estilo.

4.1.1. ¿Qué es Redmine Teaching Extension?

Redmine Teaching Extension es un producto software consistente en una extensión funcional o *plugin* para la plataforma de gestión de proyectos *Redmine*. Durante el desarrollo del mismo, se han empleado diferentes tecnologías para conseguir la funcionalidad deseada (descritas detalladamente en apartados anteriores) que se van a recordar a continuación:

- Para la parte correspondiente al *back-end* de la aplicación (parte del lado del servidor que interactúa con la base de datos), se ha empleado el framework web *Ruby on Rails* junto con el sistema de gestión de bases de datos relacional *MySQL*, existente ya en *Redmine* y que ha servido de soporte para verificar el correcto funcionamiento de la aplicación.
- Para la parte correspondiente *front-end* de la aplicación (parte del lado del cliente que aborda la creación de las vistas de la aplicación) se ha empleado el lenguaje *Ruby* combinado con *HTML* y *CSS* (forma parte de un sistema de plantillas de *Ruby Incrustado*, que se usa en *Ruby on Rails* para *Redmine*).

El propósito de *Redmine Teaching Extension*, es la creación un *plugin* para *Redmine* que aborde el sistema de creación y asignación de peticiones a proyectos (y los miembros correspondientes de estos proyectos) y extienda su funcionalidad con la idea de automatizarla. Esta automatización consiste en la propagación de una petición inicial (tras haber sido creada desde un proyecto raíz o padre), entre todos sus subproyectos (cada uno de los cuales pertenece a un usuario). Todo esto va orientado a necesidades específicas de los docentes de aquellas asignaturas del *Grado en Ingeniería Informática* de la *UAL* que empleen *Redmine* en la gestión de tareas y prácticas, donde el emisor de la petición inicial es el docente y los subproyectos son cada uno de los alumnos de la asignatura. Así pues, se pretende que cuando el docente se disponga a mandar aquellas tareas/prácticas/ejercicios comunes para todos los alumnos mediante el uso de *Redmine*, lo haga de manera simultánea a cada uno.

4.1.2. Arquitectura del software

Redmine Teaching Extension presenta una arquitectura del software en la cual se han empleado diversas tecnologías y donde se presenta una clara distinción entre dos partes anteriormente mencionadas: la parte correspondiente al *front-end* de la aplicación (lado del cliente) y la parte correspondiente al *back-end* de la aplicación (lado del servidor). Esto es lo que se conoce como arquitectura *cliente-servidor*.

El lado del cliente abarca todos los dispositivos electrónicos que permiten acceso a internet a través de un navegador Web (*Mozilla Firefox*, *Google Chrome…*) y, por consiguiente, al sistema de gestión de proyectos *Redmine*, donde un usuario *administrador* llevará a cabo la instalación del *plugin Redmine Teaching Extension* para extenderle la nueva funcionalidad implementada y trabajar con la misma. Así pues, cualquier usuario de *Redmine* que tenga a su disposición el *plugin* en cuestión, podrá interactuar con él mediante diferentes llamadas a la *API REST* de *Redmine* (explicada en el apartado *3.4.6*).

El lado del servidor corresponde a las tecnologías que procesan las peticiones de usuario mediante la interpretación de scripts en el servidor web para poder generar las páginas *HTML* dinámicamente como respuesta.

A continuación, se va a representar esquemáticamente la arquitectura del software de *Redmine* (con *Redmine Teaching Extension* incorporado), mostrando la relación que existe entre las tecnologías implicadas en el lado del cliente y el lado del servidor.

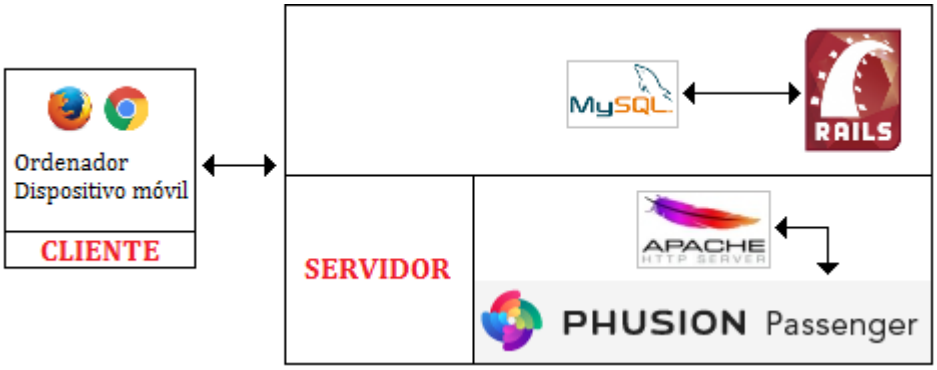

Figura 18. Arquitectura del software de Redmine + Redmine Teaching Extension

Ahora, se va a proceder a describir cada una de estas tecnologías que forman parte de la arquitectura del software del proyecto:

Phusion Passenger + Apache

Phusion Passenger [22] (también conocido como *mod_rails* y *mod_rack* entre la comunidad de *Ruby*) es un servidor web y servidor de aplicaciones gratuito con soporte para *Ruby*, *Python* y *Node.js*. Está diseñado para integrarse con el servidor *Apache* o el servidor web *nginx*, pero también tiene un modo de funcionamiento independiente sin servirse de un servidor web externo. *Phusion Passenger* es compatible con sistemas operativos similares a *Unix* y se encuentra disponible como un *paquete de gemas* o *RubyGems* (que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones *Ruby*), como un archivo comprimido *.tar* o como *paquetes de Linux* nativos.

Originalmente se diseñó para aplicaciones web construidas en el framework *Ruby on Rails* y más tarde se extendió su soporte para frameworks web arbitrarios de *Ruby* por medio de la interfaz web *Rack*.

Como bien se ha dicho en apartados anteriores, *Redmine Teaching Extension* es un *plugin* para el sistema de gestión de proyectos *Redmine*. Por ello, se optó por llevar a cabo la instalación y configuración de *Phusion Passenger* en combinación con el servidor *Apache* durante una de las fases de instalación [23] del sistema *Redmine* vacío (usado como soporte para el desarrollo de *Redmine Teaching Extension*) ya que cumple con todos los aspectos necesarios para posicionarse como una de las opciones más viables y sencillas de configuración de un servidor web (*Redmine* funciona con el framework *Ruby on Rails* y, además, se está trabajando sobre una partición del sistema operativo *Ubuntu 14.04* de *Linux*).

Ruby on Rails + MySQL

Para la creación de *Redmine Teaching Extension*, se va a emplear el framework web *Ruby on Rails*. Como se comentó en apartados anteriores, *Redmine* está programado usando *Ruby on Rails* por lo que para el desarrollo de este *plugin* es necesario basarse también en este framework web para llevar a cabo una correcta y eficiente codificación.

Por otro lado, en combinación con *Ruby on* Rails, se emplea el sistema de gestión de bases de datos relacional *MySQL*, con el cual trabaja *Redmine* y que, por tanto, se verá afectado en cierta medida por el funcionamiento de *Redmine Teaching Extension*.

4.2. Especificación preliminar

El objetivo del proyecto es desarrollar una extensión funcional o *plugin* de la plataforma de gestión de proyectos *Redmine*, destinada al uso docente. Por ello, parte de la especificación preliminar se definirá desde el punto de vista del mismo (considerando *Redmine* y *Redmine Teaching Extension* como un todo) pues, al fin y al cabo, el proyecto a desarrollar es una prolongación de la funcionalidad predeterminada de *Redmine*. Aun así, en ciertos aspectos (como pueden ser el *usuario administrador* o *usuario registrado*, con permisos suficientes para instalar y configurar el *plugin*), se comentará la especificación desde el punto de vista de *Redmine Teaching Extension*.

En *Redmine*, destacan por una parte los ***usuarios visitantes*** (cibernautas sin registrar en el sistema) que, en principio, se limitarán a observar los distintos seguimientos de incidencias registrados por aquellos usuarios registrados en el sistema, siempre y cuando el usuario administrador los haya habilitado como públicos. Estos usuarios visitantes que así lo deseen, podrán registrarse en *Redmine*, introduciendo las credenciales obligatorias para ello que son un identificador o nick, contraseña de 4 caracteres como mínimo, confirmación de contraseña, nombre, apellidos y correo electrónico (utilizado también para la recuperación de la contraseña e inicio de sesión, y que no deberá estar registrado en el sistema). Para iniciar sesión, habrá que introducir el identificador único o nick del usuario y la contraseña.

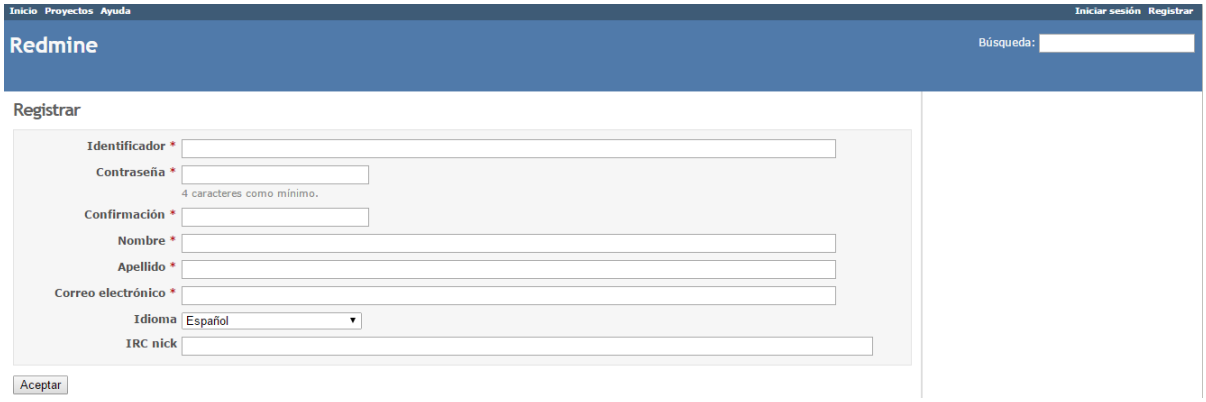

Figura 19. Ventana de Registrar usuario de Redmine

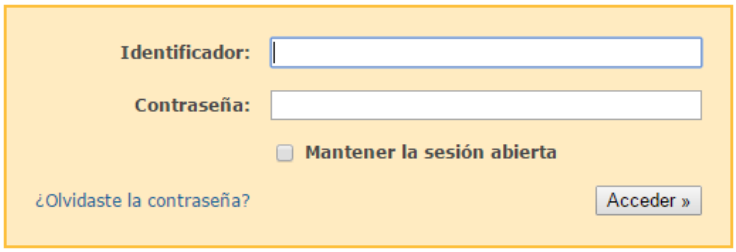

Figura 20. Ventana de Iniciar sesión de Redmine

Por otra parte, están los ***usuarios registrados (no administradores)*** que, una vez hayan iniciado sesión en *Redmine* (como se ha mostrado anteriormente), formarán parte del personal de la plataforma que participa en proyectos con distintos niveles de responsabilidades. En función de las necesidades que se identifiquen, se modelarán distintos perfiles/cargos de usuario atendiendo al papel que desempeñan en los proyectos. Podrán interactuar con las diferentes funcionalidades que ofrece el sistema (según el nivel de permisos y roles otorgados por parte del usuario administrador del sistema). Estos usuarios, podrán llevar a cabo las siguientes acciones (siempre y cuando tengan los permisos suficientes):

- Personalización de la página personal del usuario, añadiendo nuevos bloques de información y organizarlos, así como acceder a información sobre proyectos, tareas, noticias, actividad reciente, informes y notas.
- Visualizar el avance del proyecto del que se forma parte, tomando como punto de partida el esfuerzo estimado y dedicado a nivel de tarea o petición.
- Utilizar un motor de búsqueda, que facilita la localización de información relativa a proyectos, peticiones, usuarios, noticias y documentos.
- Configurar las preferencias de usuario (nombre, apellidos, dirección de correo electrónico, idioma…) a través de la ventana de *Mi cuenta*.
- Acceder a un módulo de calendario que permite consultar las actividades planificadas y asignadas, así como los hitos relevantes del proyecto o cualquier otro tipo de evento, facilitando la tarea de planificación de tareas y reuniones, así como la gestión más eficiente de recursos disponibles.
- Acceder a una sección de *Wiki*, donde se puede centralizar toda la documentación del proyecto y otra información de interés para los miembros del proyecto en cuestión.

- Si el usuario cuenta con el rol de *Jefe de proyecto* o con privilegios suficientes por parte del *Administrador*, podrá crear nuevas peticiones (de tipo *Error*, *Tarea*, *Soporte*…), las cuales podrá asignar a cualquier otro usuario del proyecto actual. En estas peticiones, además de indicar el tipo anteriormente mencionado, el usuario indicará el nombre de la tarea, el estado de la misma (*Nueva*, *En curso*, *Finalizada*, *Rechazada*), la prioridad (*Baja*, *Normal*, *Alta*, *Urgente*) y, a voluntad propia, podrá también asignar la petición a cualquier otro usuario de su equipo de proyecto y definir una descripción de la petición. Además, estas peticiones pueden ser editadas y copiadas, una vez hayan sido creadas.
  - ✓ Como novedad, con *Redmine Teaching Extension* se ha extendido esta funcionalidad, añadiendo a la ventana de *Nueva petición* nuevos componentes que permiten a los usuarios registrados (con los permisos en cuestión) realizar una selección múltiple de los subproyectos del proyecto actual, permitiendo así crear la petición y propagarla entre todos los subproyectos seleccionados. De este modo, se logra la creación de peticiones desde un proyecto actual (proyecto raíz) a otros proyectos externos (subproyectos del proyecto raíz).
  - ✓ Para disfrutar de esta nueva funcionalidad, el usuario deberá llevar a cabo la instalación de la extensión, siguiendo para ello los pasos que proporciona *Redmine* para ello [24]. Una vez instalada, podrá visualizar que se ha integrado correctamente en el sistema, accediendo para ello a la ventana de *Plugins* donde se muestra un listado de todas las extensiones instaladas. Además, algunas extensiones pueden tener habilitado un menú de configuración, para establecer y controlar varios parámetros de las mismas.

A continuación, se van a presentar varias capturas del sistema con la representación, a grandes rasgos, de las acciones anteriormente mencionadas (si bien, los resultados correspondientes a *Redmine Teaching Extension* se mostrarán y explicarán más adelante):

Figura 21. Ventana de la página personal del usuario, con el motor de búsqueda.

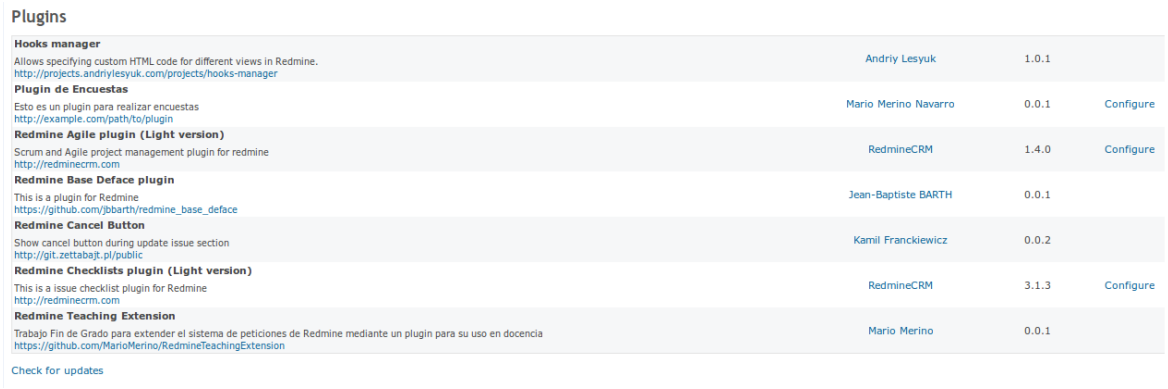

Figura 22. Ventana del listado de plugins instalados en el sistema

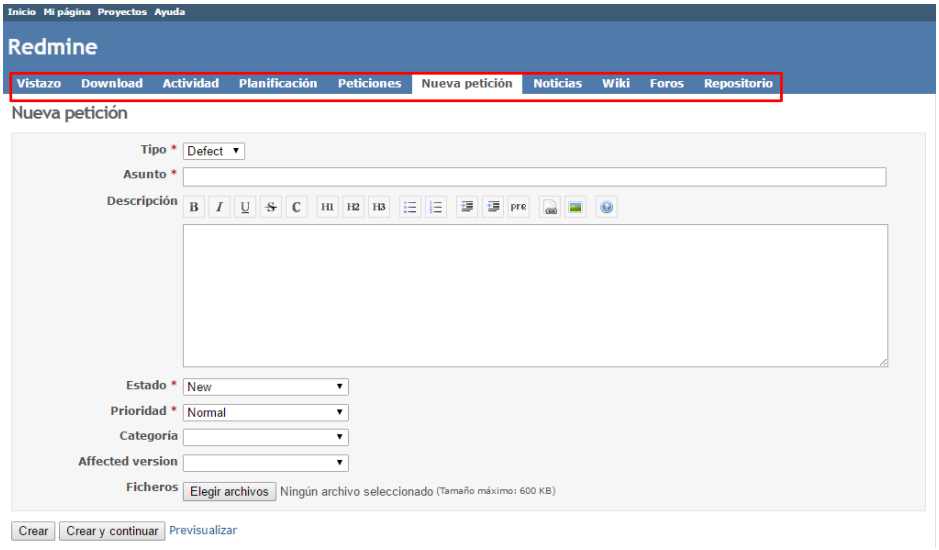

Figura 23. Ventana de Nueva petición e indicación del resto de secciones (marcadas en rojo)

Por último, están los ***usuarios registrados (administradores)***, que además de poder llevar a cabo cualquiera de las acciones propias de un usuario registrado no administrador, dispone de todos los privilegios de gestión de la plataforma (pudiendo controlar las gestiones desde un panel de administrador que tiene habilitado para ello):

- Responsable de la administración general del sistema.
- Dispone de acceso a datos agregados sobre múltiples proyectos, así como cualquier otro indicador mediante el módulo de generación de informes.
- Puede añadir nuevos proyectos y subproyectos.
- Puede gestionar usuarios sin límite.
- Pleno acceso y gestión de parámetros de los proyectos.
- Pleno acceso a la gestión de actividades y subactividades de los proyectos.
- Pleno acceso a la gestión de privilegios y flujos de trabajo asociados a cada tipo de actividad.

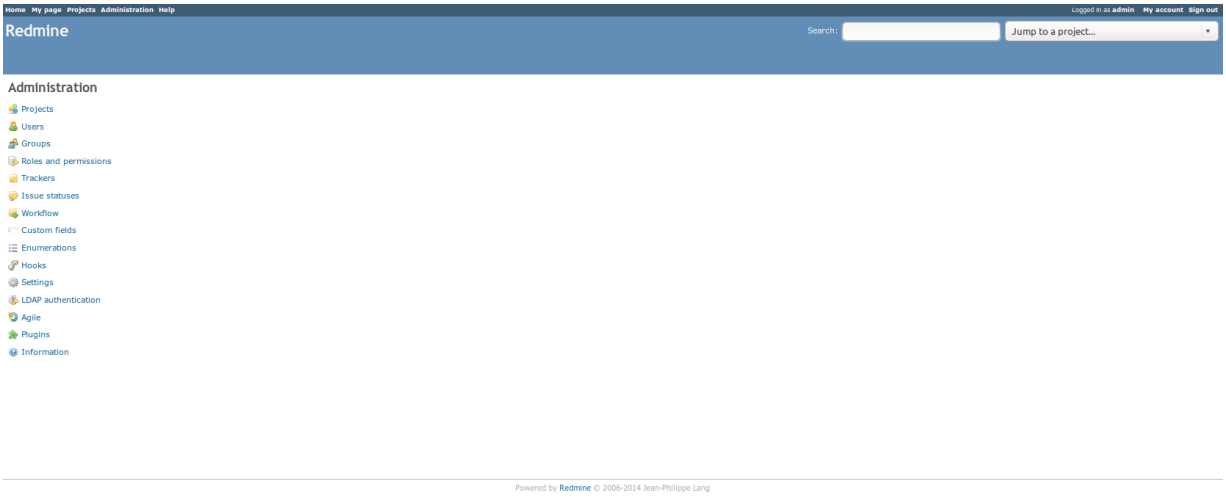

Figura 24. Módulo de administración de Redmine

4.3. Documento de Requisitos del Sistema (DRS)

El propósito del *Documento de Requisitos del Sistema* [25] es llevar a cabo la especificación de todos los requisitos que conforman el producto software. Debe incluir tanto los requerimientos a nivel de usuario para el sistema, como una especificación de los requerimientos informáticos, y se debe organizar de tal forma que pueda ser utilizado tanto por los clientes del sistema como por el desarrollador del software. Así pues, estos requerimientos determinan lo que debe hacer el sistema y definen las restricciones en su funcionamiento.

4.3.1. Descripción del sistema actual

Actualmente, los usuarios del sistema de gestión de proyectos *Redmine* pueden llevar a cabo la creación individual de las tareas/peticiones dentro de un mismo proyecto, y además, sin opción de crear la petición para otro proyecto externo.

Esta técnica definida por *Redmine*, resulta muy lenta e incómoda en el caso concreto de que el usuario *administrador* o con rol de *Jefe de proyecto* gestione un sistema de numerosos subproyectos dentro de un mismo proyecto a los cuales pretende asignar una petición con el mismo contenido (como puede ser a nivel de asignatura, donde el profesor crea las peticiones desde el proyecto principal que es la asignatura y los subproyectos son los alumnos de la misma), ya que tendría que proceder proyecto a proyecto para asignar cada una de las peticiones.

4.3.2. Objetivos del sistema

Lo definido anteriormente en la descripción del sistema actual, conlleva la dedicación de mucho tiempo para realizar esa actividad, por lo que resulta imprescindible la creación de un nuevo sistema que automatice y agilice el proceso.

OBJ-01	Seleccionar múltiples subproyectos para propagarles petición
Descripción	El sistema deberá permitir el acceso (haciendo clic en un botón) a una ventana con un listado de todos los subproyectos del proyecto actual, donde se seleccionarán aquellos implicados en la propagación de la petición en cuestión.
Estabilidad	Alta
Comentarios	Ninguno

Tabla 4. Objetivo del sistema OBJ-01

OBJ-02	Representar listado de subproyectos seleccionados en el formulario de *Nueva petición*
Descripción	El sistema deberá mostrar, en el formulario de *Nueva petición*, un listado de los subproyectos previamente seleccionados para propagarles la petición en cuestión.
Estabilidad	Alta
Comentarios	Ninguno

Tabla 5. Objetivo del sistema OBJ-02

OBJ-03	Propagar petición existente entre los subproyectos seleccionados
Descripción	Una vez seleccionados los subproyectos implicados en la propagación de la petición, el sistema deberá habilitar un botón que permita crear la petición en el proyecto actual y propagarla entre dichos subproyectos.
Estabilidad	Alta
Comentarios	Ninguno

Tabla 6. Objetivo del sistema OBJ-03

4.3.3. Requisitos identificados en el sistema

4.3.3.1. Requisitos de almacenamiento de información

Los *requisitos de almacenamiento de información* recogerán aquella información que va a almacenar el sistema y como se relaciona entre sí.

RI-01	Datos del usuario actual
Versión	1.0
Autores	Mario Merino Navarro
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición
Requisitos asociados	RI-02 – Datos del proyecto del usuario actual RF-01 – Ver proyecto de usuario RF-02 – Ver formulario de *Nueva petición* RF-03 – Seleccionar subproyectos para propagar petición RF-04 – Visualizar el listado de subproyectos seleccionados en el formulario de Nueva petición

	RF-05 – Crear y propagar petición
Descripción	El sistema deberá recoger varios datos del usuario actual, que en concreto son:
Datos específicos	- *ID* o *nick* de usuario - Nombre completo - Contraseña
Intervalo temporal	Presente
Estabilidad	Alta
Comentarios	Ninguno

Tabla 7. Requisito de almacenamiento de información RI-01

RI-02	Datos del proyecto del usuario actual
Versión	1.0
Autores	Mario Merino Navarro
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición
Requisitos asociados	RI-01 – Datos del usuario actual RF-01 – Ver proyecto de usuario RF-02 – Ver formulario de Nueva petición RF-03 – Seleccionar subproyectos para propagar petición RF-04 – Visualizar el listado de subproyectos seleccionados en el formulario de Nueva petición RF-05 – Crear y propagar petición
Descripción	El sistema deberá recoger varios datos del proyecto del usuario actual y de los subproyectos correspondientes, que en concreto son:
Datos específicos	- Nombre del proyecto (asignatura) - Nombre de los subproyectos del proyecto - Nombre de los usuarios miembros (profesor o profesores), asignado/s al proyecto - Nombres de usuarios miembros (alumno y profesor/es), asignados a cada uno de los subproyectos
Intervalo temporal	Presente
Estabilidad	Alta
Comentarios	Ninguno

Tabla 8. Requisito de almacenamiento de información RI-02

4.3.3.2. Requisitos funcionales

Los *requisitos funcionales* describirán una funcionalidad o un servicio del sistema. Las funciones del sistema son descritas como un conjunto de entradas, comportamientos y salidas. Los requerimientos de comportamiento para cada requisito funcional se muestran en los *casos de uso* (que se detallarán más adelante).

RF-01	**Ver proyecto de usuario**	
Versión	1.0	
Autores	Mario Merino Navarro	
Objetivos asociados	Ninguno	
Requisitos asociados	RI-01 – Datos del usuario actual RI-02 – Datos del proyecto del usuario actual	
Descripción	El sistema deberá permitir al usuario visualizar el proyecto del que forme parte, accediendo desde la página personal del usuario	
Precondición	El usuario tiene que estar identificado en el sistema *Redmine* y formar parte de algún proyecto, al que ha sido asignado por parte del *Jefe de proyecto*	
Secuencia normal	**Paso**	**Acción**
	1	El usuario visualiza la sección de proyectos de su página personal de *Redmine*
	2	El usuario accede a su proyecto
	3	El usuario visualiza las distintas opciones con las que puede interactuar en el proyecto (*Vistazo*, *Peticiones*, *Nueva petición*, *Wiki…*)
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si el sistema detecta que el usuario no ha sido asignado al proyecto o no tiene permisos de acceso, posiblemente no pueda visualizar el proyecto
Rendimiento	**Paso**	**Cota de tiempo**
	2	1 segundo
Frecuencia esperada	Se estima que esta función se realice con mucha frecuencia	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 9. Requisito funcional RF-01

RF-02	Ver formulario de *Nueva petición*	
Versión	1.0	
Autores	Mario Merino Navarro	
Objetivos asociados	Ninguno	
Requisitos asociados	RI-02 – Datos del proyecto del usuario actual	
Descripción	El sistema deberá permitir al usuario de un proyecto visualizar la sección de *Nueva petición*, para proceder a rellenar el formulario de creación de la petición, según los campos habilitados para ello	
Precondición	El usuario tiene que estar identificado en el sistema *Redmine* y formar parte de algún proyecto, al que ha sido asignado por parte del *Jefe de proyecto*	
Secuencia normal	**Paso**	**Acción**
	1	El usuario accede al proyecto en cuestión
	2	El usuario busca y selecciona la sección destinada a la creación de una nueva petición
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si el administrador del sistema así lo cree oportuno, puede no concederle el permiso de creación de peticiones al usuario, denegándole el acceso a dicha sección
Rendimiento	**Paso**	**Cota de tiempo**
	2	1 segundo
Frecuencia esperada	Se estima que esta función se realice con mucha frecuencia	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 10. Requisito funcional RF-02

RF-03	Seleccionar subproyectos para propagar petición	
Versión	1.0	
Autores	Mario Merino Navarro	
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición	
Requisitos asociados	RI-02 – Datos del proyecto del usuario actual	
Descripción	El sistema deberá permitir la función de seleccionar de un listado todos aquellos subproyectos del proyecto actual implicados en la posterior propagación de la petición que el usuario se disponga a crear	
Precondición	El usuario tiene que estar identificado en el sistema *Redmine* y formar parte de algún proyecto, al que ha sido asignado por parte del *Jefe de proyecto*, además de haber instalado la extensión *Redmine Teaching Extension*, necesaria para llevar a cabo esta actividad	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la sección destinada a la creación de una nueva petición
	2	El usuario, una vez que haya accedido al formulario de creación de peticiones, hace clic en el botón de selección de subproyectos
	3	El usuario accede a un listado con todos los subproyectos del proyecto actual y selecciona aquellos a los que desea aplicarles la propagación de la petición en cuestión (puede seleccionarlos todos/deshacer selección simultáneamente o seleccionar individualmente)
	4	El usuario aplica los diferentes cambios
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si en el proyecto actual el *Jefe de proyecto* no ha creado ningún subproyecto o si el proyecto actual es un subproyecto, el botón de selección de subproyectos permanecerá deshabilitado

Rendimiento	**Paso**	**Cota de tiempo**
	2	1 segundo
	3	Depende de si el usuario selecciona simultáneamente todos los subproyectos del listado o si hace una selección individual (normalmente, no más de 5 segundos)
	4	1 segundo
Frecuencia esperada	Se estima que esta función se realice con mucha frecuencia	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 11. Requisito funcional RF-03

RF-04	**Mostrar en el formulario de *Nueva petición* el listado de subproyectos seleccionados para propagar petición**	
Versión	1.0	
Autores	Mario Merino Navarro	
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición	
Requisitos asociados	RI-02 – Datos del proyecto del usuario actual RF-03 – Seleccionar subproyectos para propagar petición	
Descripción	El sistema deberá representar el listado de los subproyectos seleccionados en el formulario de creación de una nueva petición	
Precondición	El usuario, además de estar identificado en el sistema *Redmine,* de formar parte de algún proyecto y de haber instalado la extensión *Redmine Teaching Extension* necesaria para llevar a cabo esta actividad, también tiene que haber realizado la selección de subproyectos correspondiente	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace clic en el botón de selección de subproyectos
	2	El usuario accede a un listado con todos los subproyectos del proyecto actual y

		selecciona aquellos a los que desea aplicarles la propagación de la petición en cuestión
	3	El usuario aplica los diferentes cambios
	4	El formulario de *Nueva petición* se refresca y aparece reflejado el listado de subproyectos seleccionado
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	3	Si se aplican cambios sin haber seleccionado algún o algunos subproyectos del listado o se cierra la ventana de selección de subproyectos, el formulario no mostrará ningún listado de subproyectos implicados en el proceso
Rendimiento	**Paso**	**Cota de tiempo**
	4	1 segundo
Frecuencia esperada	Se estima que esta función se realice con mucha frecuencia	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 12. Requisito funcional RF-04

RF-05	Crear y propagar petición
Versión	1.0
Autores	Mario Merino Navarro
Objetivos asociados	OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición OBJ-03 – Propagación de petición entre los subproyectos seleccionados
Requisitos asociados	RI-02 – Datos del proyecto del usuario actual RF-03 – Seleccionar subproyectos para propagar petición RF-04 – Visualizar el listado de subproyectos seleccionados en el formulario de Nueva petición
Descripción	El sistema deberá habilitar un botón que permita la creación de la petición en el proyecto actual y su posterior propagación (creación múltiple) entre los subproyectos del proyecto actual implicados en el proceso

Precondición	El usuario, además de estar identificado en el sistema *Redmine,* de formar parte de algún proyecto y de haber instalado la extensión *Redmine Teaching Extension* necesaria para llevar a cabo esta actividad, también tiene que haber realizado la selección de subproyectos correspondiente y aplicado dichos cambios, para que de este modo se pueda habilitar el botón para crear y propagar la petición	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace clic en el botón de C*rear petición y continuar para propagar*
	2	La petición en cuestión es creada correctamente, por un lado en el proyecto actual y por otro lado en cada uno de los subproyectos del proyecto actual seleccionados para ello
	3	La ventana de *Nueva petición* redirecciona a la ventana de la petición creada en el proyecto principal
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si el usuario actual refresca la ventana de *Nueva petición* del proyecto actual o decide acceder a la ventana de selección de subproyectos para no seleccionar ninguno y luego aplicar cambios, el botón de *Crear petición y continuar para propagar* permanecerá inhabilitado
Rendimiento	**Paso**	**Cota de tiempo**
	2	1-5 segundos aprox. (depende del número de subproyectos implicados en la propagación de la petición)
Frecuencia esperada	Se estima que esta función se realice con mucha frecuencia	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 13. Requisito funcional RF-05

4.3.3.3. Definición de actores

Los *actores* del sistema, especifican un rol jugado por un usuario o cualquier otro sistema que interactúa con el sujeto.

ACT-01	Usuario visitante
Versión	1.0
Autores	Mario Merino Navarro
Descripción	Este actor representa al usuario sin registrar en el sistema que, en principio, se limitará a observar los distintos seguimientos de incidencias registrados por aquellos usuarios registrados en el sistema, siempre y cuando el usuario administrador los haya habilitado como públicos. Si así lo desea, podrá registrarse en el sistema *Redmine*
Comentarios	Ninguno

Tabla 14. Actor del sistema ACT-01

ACT-02	Usuario registrado
Versión	1.0
Autores	Mario Merino Navarro
Descripción	Este actor representa al usuario que, una vez haya iniciado sesión en el sistema *Redmine*, formará parte del personal de la plataforma que participa en proyectos con distintos niveles de responsabilidades. En función de las necesidades que se identifiquen, se modelará un perfil/cargo de usuario atendiendo al papel que desempeña en los proyectos. Podrá interactuar con las diferentes funcionalidades que ofrece el sistema (según el nivel de permisos y roles otorgados por parte del administrador del sistema)
Comentarios	Ninguno

Tabla 15. Actor del sistema ACT-02

ACT-03	Administrador
Versión	1.0
Autores	Mario Merino Navarro
Descripción	Este actor representa al usuario del sistema que, además de poder llevar a cabo cualquiera de las acciones propias de un usuario registrado (no administrador), dispone de todos los privilegios de gestión de la plataforma, pudiendo controlar las gestiones desde un panel de administrador habilitado para ello
Comentarios	Ninguno

Tabla 16. Actor del sistema ACT-03

4.3.3.4. Diagrama de casos de uso

El *diagrama de casos de uso* del sistema, define esquemáticamente el comportamiento del sistema desarrollado al afrontar determinados requisitos funcionales. Cada uno de los casos de uso del diagrama tendrá su equivalente en un requisito funcional.

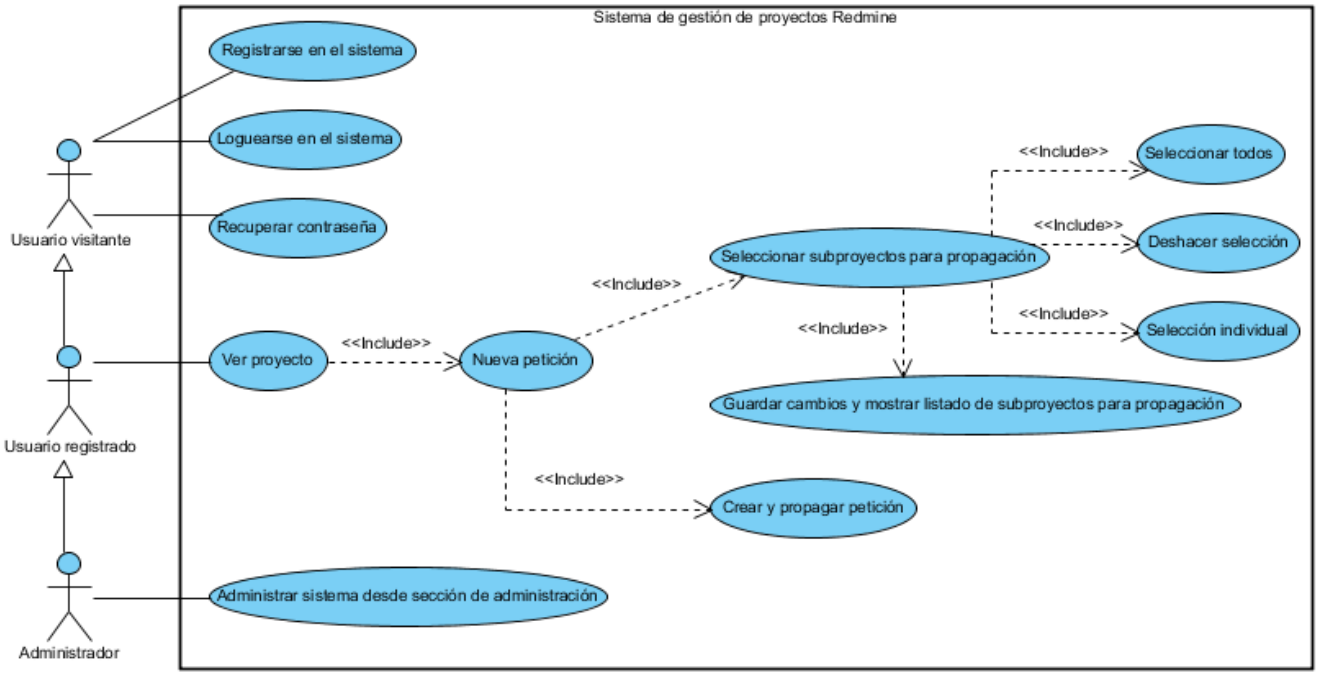

Figura 25. Diagrama de casos de uso del funcionamiento del sistema

4.3.4. Requisitos no funcionales

Los *requisitos no funcionales* o atributos de calidad son aquellos que especifican criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales. Por tanto, se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento.

RNF-01	Versión de *Redmine* compatible con la extensión desarrollada
Versión	1.0
Autores	Mario Merino Navarro
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición OBJ-03 – Propagación de petición entre los subproyectos seleccionados
Requisitos asociados	Ninguno
Descripción	El usuario deberá formar parte de una versión del sistema *Redmine* que cuente con soporte para la extensión desarrollada
Importancia	Alta
Comentarios	En general, esto no es un gran problema, ya que la extensión *Redmine Teaching Extension* se desarrolló para una versión 2.5.3 estable de *Redmine*. De existir alguna incidencia al respecto, sería un problema menor hacer la adaptación de versiones

Tabla 17. Requisito no funcional RNF-01

RNF-02	Entorno de explotación
Versión	1.0
Autores	Mario Merino Navarro
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición OBJ-03 – Propagación de petición entre los subproyectos seleccionados
Requisitos asociados	Ninguno

Descripción	El sistema ha sido desarrollado para su empleo como extensión de la plataforma de gestión de proyectos *Redmine*
Importancia	Alta
Comentarios	Ninguno

Tabla 18. Requisito no funcional RNF-02

RNF-03	Usabilidad
Versión	1.0
Autores	Mario Merino Navarro
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición OBJ-03 – Propagación de petición entre los subproyectos seleccionados
Requisitos asociados	Ninguno
Descripción	El sistema deberá ser sencillo de usar por parte del usuario. Tiene que ofrecer facilidad en su aprendizaje para el buen uso del mismo, ofrecer un buen rendimiento durante su ejecución y ser flexible a nuevas necesidades y cambios que se presenten oportunos
Importancia	Alta
Comentarios	Ninguno

Tabla 19. Requisito no funcional RNF-03

RNF-04	Mantenibilidad
Versión	1.0
Autores	Mario Merino Navarro
Objetivos asociados	OBJ-01 – Seleccionar múltiples subproyectos para propagarles petición OBJ-02 – Ver listado de subproyectos seleccionados en el formulario de Nueva petición OBJ-03 – Propagación de petición entre los subproyectos seleccionados
Requisitos asociados	Ninguno
Descripción	El sistema deberá requerir una cantidad de esfuerzo mínima para conservar su funcionamiento normal o para restituirlo

	cuando se presente algún tipo de error que altere su funcionamiento
Importancia	Alta
Comentarios	Ninguno

Tabla 20. Requisito no funcional RNF-04

4.4. Análisis Jerárquico de Tareas (HTA)

El *Análisis Jerárquico de Tareas* es una de las técnicas de análisis de tareas más conocidas y antiguas. [26]

En *HTA* se realiza una descripción de tareas en términos de operaciones y planes. Las operaciones (descomposición en subtareas) son actividades que realizan las personas para alcanzar un objetivo, y los planes son una descripción de las condiciones que se tienen que dar cuando se realiza cada una de las actividades. Las operaciones se pueden descomponer de forma jerárquica y se asigna un plan a cada una de las subtareas que aparecen. Se define un objetivo como un estado determinado del sistema que puede o quiere alcanzar el usuario.

A la hora de describir la descomposición de tareas en subtareas, se puede representar cuatro tipos de descomposiciones:

- *Secuencia*: Descomposición en un conjunto ordenado temporalmente de una secuencia de tareas.
- *Selección*: Conjunto de tareas de las que se tendrá que elegir una de ellas.
- *Iteración*: Repetición de un subconjunto de tareas.
- *Tarea unitaria*: Actividad indivisible (según el nivel de detalle dado)

El análisis de una tarea implica tres etapas enlazadas: *recogida de información*, *modelado* y *análisis*. Los procedimientos de recogida de información incluyen la revisión de la documentación existente (por ejemplo, manuales de funcionamiento, procedimientos, informes de seguridad, estudios de análisis de tareas previos, diseños, imágenes, prototipos, etc.), que permitan establecer qué hacen las personas en circunstancias específicas (normales y anormales), entrevistas y cuestionarios (descripciones por parte de personas experimentadas de cómo hacen las cosas, que informaciones necesitan y como determinan si la tarea se puede realizar satisfactoriamente).

Así pues, para mostrar el desglosamiento de las tareas implicadas en el sistema con el mayor detalle posible, se va a llevar a cabo una representación mediante *diagramas de secuencias*.

4.4.1. Diagramas de secuencias

El *diagrama de secuencias* [27] es un tipo de diagrama empleado para modelar la interacción entre un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario y mensajes intercambiados entre los objetos.

A continuación se van a definir todos los diagramas de secuencias que forman parte de la funcionalidad del sistema.

Ver proyecto de usuario

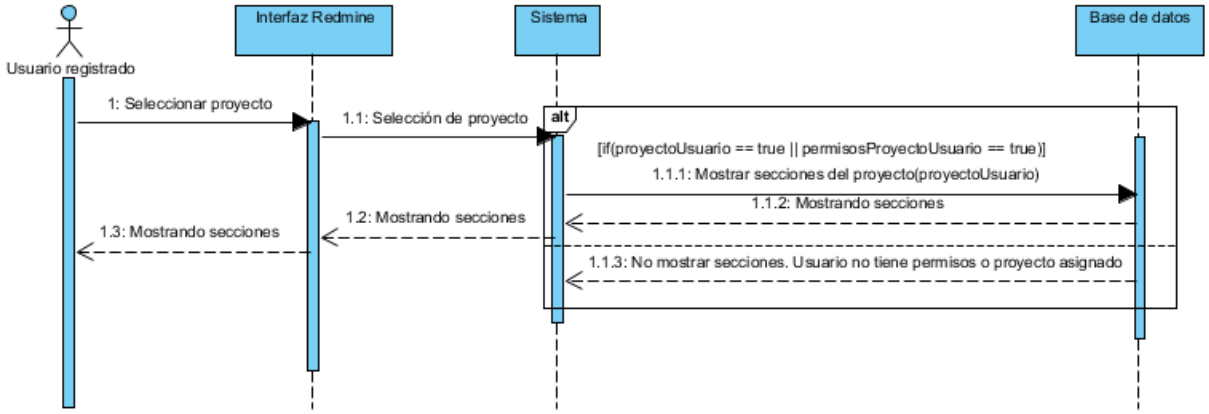

Figura 26. Diagrama de secuencias de visualización del proyecto de usuario

Ver formulario de Nueva petición

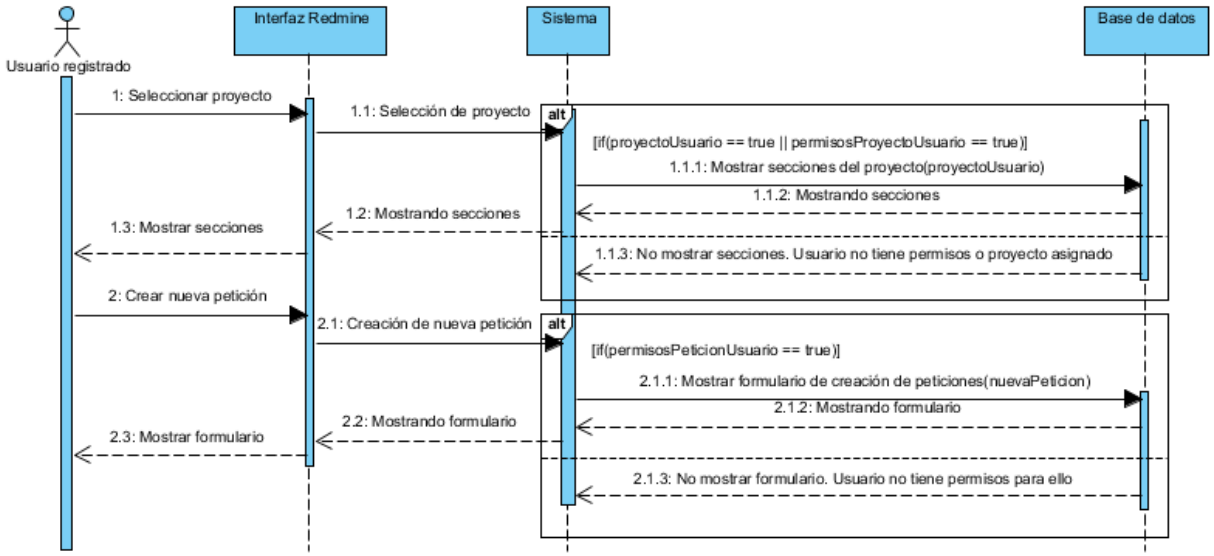

Figura 27. Diagrama de secuencias de visualización de Nueva petición

Seleccionar subproyectos para propagar petición y mostrar selección en el formulario de Nueva petición

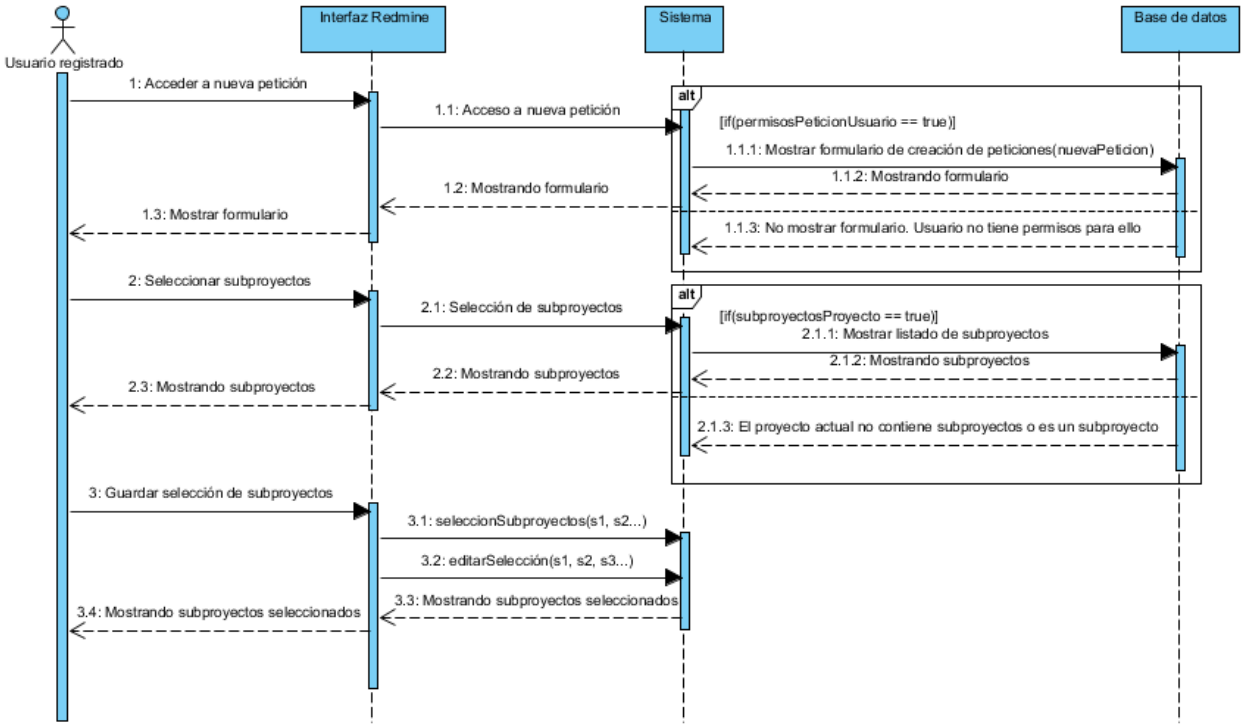

Figura 28. Diagrama de secuencias de selección de subproyectos y su visualización

Crear y propagar petición

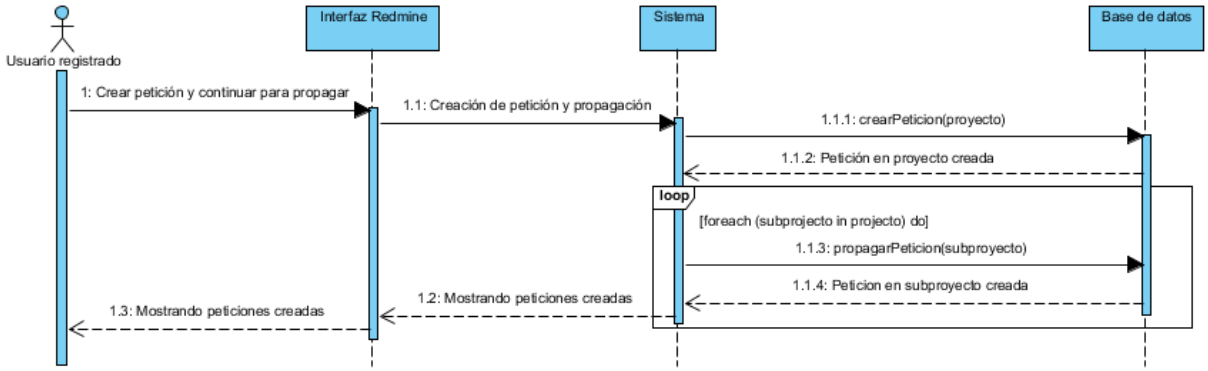

Figura 29. Diagrama de secuencias de creación y propagación de petición

4.5. Diseño del sistema

El *diseño del sistema* [28] es el arte de definir la arquitectura de hardware y software, componentes, módulos y datos de un sistema, a efectos de satisfacer ciertos requerimientos.

Los métodos de análisis y diseño orientado a objetos están siendo los métodos más ampliamente utilizados para el diseño de sistemas. El *UML* se ha vuelto un estándar en el análisis y diseño orientado a objetos, siendo su uso esencial en el diseño del sistema. Por ello, se va a llevar a cabo una representación del sistema en términos de clases mediante un *diagrama de clases*.

4.5.1. Diagrama de clases del sistema

El *diagrama de clases* [29] en *UML* es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.

En este apartado, se va a representar el *diagrama de clases* del sistema desarrollado y posteriormente se detallará una definición de cada una de las clases implicadas en el mismo.

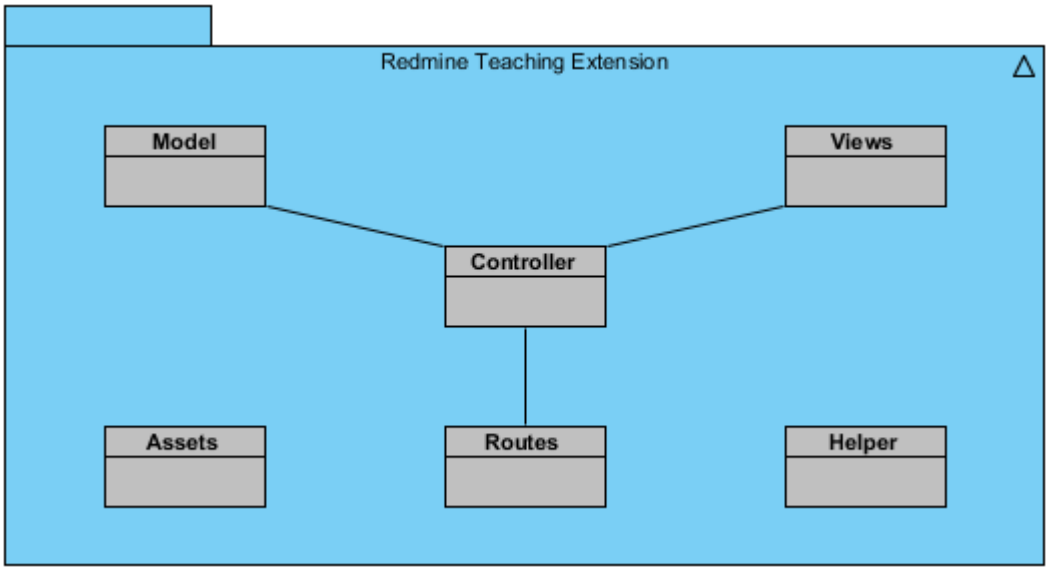

Figura 30. Diagrama de clases del sistema

Model

El *Modelo* agrupa al conjunto de clases que representan a las tablas de la base de datos del sistema.

En *Ruby on Rails*, las clases del *Modelo* son gestionadas por *ActiveRecord*. El desarrollador del sistema lo único que tiene que hacer es heredar una de las clases `ActiveRecord::Base`, y la aplicación averiguará automáticamente qué tabla usar y qué columnas tiene.

Las definiciones de las clases también detallan las relaciones entre clases con sentencias de mapeo objeto relacional.

En definitiva, el *Modelo* puede representar las *tablas de la base de datos*, *migraciones* (expresan cambios en la base de datos) y *observadores*.

Controller

Las clases del *Controlador* responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del *Modelo* y muestra los resultados usando las *Vistas*. Los métodos del *Controlador* son invocados por el usuario usando el navegador web.

La implementación del *Controlador* es manejada por el *ActionPack* de *Rails*, que contiene la clase `ApplicationController`. La aplicación simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web, en la forma `http://aplicacion/ejemplo/metodo`, que invoca a `EjemploController#método`, y presenta los datos (*Vista*) usando el archivo de plantilla `/app/views/ejemplo/método.html.erb`, a no ser que el método redirija a algún otro lugar.

Views

Las *Vistas* forman parte de la lógica de visualización, o cómo se muestran los datos de las clases del *Controlador*.

Existen en la actualidad muchas maneras de gestionar las *Vistas*. El método que se emplea en *Rails* por defecto (y el que se va a emplear en el desarrollo del proyecto *Redmine Teaching Extension*) es usar *Ruby Empotrado* (archivos con formato *archivos.html.erb*), que son básicamente fragmentos de código *HTML* con algo de código en *Ruby*, siguiendo una sintaxis similar a *JSP*.

Es necesario escribir un pequeño fragmento de código en *HTML* para cada método del *Controlador* que necesita mostrar información al usuario. El "maquetado" o distribución de los elementos de la página se describe separadamente de la acción del *Controlador* y los fragmentos pueden invocarse unos a otros.

Routes

Las *Rutas* [30] juegan un rol importante en la existencia de las conexiones entre la petición del usuario y la aplicación.

Cuando un usuario hace clic dentro de algún enlace existente en la aplicación o escribe directamente la *URL* del enlace en cuestión en el navegador, *Rails* lee el archivo de *Rutas* (*routes.rb*), y al encontrarla crea una instancia del C*ontrolador* asociado a esa ruta y ejecuta el método de la acción.

Helper

Un *Helper* [31] es un módulo donde se lleva a cabo la definición de determinadas funciones en *Ruby* cuyo objetivo es ayudar a que el código de las *Vistas* cumpla el principio *DRY* (*Don't Repeat Yourself o No Te Repitas*) encapsulando para ello código repetitivo que se ejecute en las mismas.

Assets

El módulo *Assets* almacena el conjunto de archivos correspondientes a las hojas de estilo *CSS*, los archivos *JavaScript* y las imágenes que se utilizan en el *frontend* de la aplicación para mejorar su aspecto.

4.6. Aspectos destacados de la implementación del sistema

En este apartado se van a tratar los aspectos más importantes en lo que se refiere a la implementación de *Redmine Teaching Extension*, desde la estructura del proyecto que permite llevar a cabo la codificación de una forma ordenada y eficiente hasta la explicación de los fragmentos de código más determinantes y relevantes en la funcionalidad del proyecto.

4.6.1. Jerarquización y organización de archivos del proyecto

El directorio del proyecto *Redmine Teaching Extension* contiene un número de ficheros y carpetas auto-generados que constituyen la estructura de una aplicación en *Ruby on Rails*. Aunque la mayor parte del trabajo de codificación se hará en la carpeta *app*, es necesario demostrar la relación básica entre cada fichero/carpeta y la función que cumple:

Fichero/carpeta	Propósito
redmine_teaching_extension/	Se trata del directorio raíz, que contiene a todos los demás directorios del proyecto, junto con el archivo *init.rb* cuya función es la inicialización del sistema mediante la declaración de información de relevancia (nombre del *plugin*, autor, versión, dependencias…) y el archivo *README.md* que se trata de un breve manual de instrucciones del funcionamiento del proyecto.
app/	Es la carpeta más importante del proyecto, pues contiene los controladores (en la carpeta *controllers*), los modelos (carpeta *models*), las vistas (carpeta *views*) y funciones de ayuda (carpeta *helpers*), fundamentales para el buen funcionamiento de la aplicación.
controllers/	Contiene las clases correspondientes a los *controladores* necesarios para el funcionamiento de la aplicación.
models/	Contiene las clases correspondientes a los *modelos* necesarios para el funcionamiento de la aplicación.
views/	Contiene las clases correspondientes a las *vistas* necesarias para el funcionamiento de la aplicación.
helpers/	Contiene las clases correspondientes a los *helpers* que aportarán determinadas funciones de ayuda para el correcto funcionamiento de la aplicación.
assets/	Contiene el conjunto de archivos correspondientes a las hojas de estilo *CSS* (carpeta *stylesheets*), los archivos JavaScript (carpeta *javascripts*) y las imágenes que se emplean en el *frontend* del sistema para mejorar su apariencia.
config/	Contiene los archivos de configuración de la aplicación, que en este caso son las rutas definidas en el archivo de enrutamiento *routes.rb* y los archivos locales (archivos de traducciones disponibles para la aplicación) contenidos en la carpeta *locales*.
db/	Contiene el actual esquema de la base de datos, así como las migraciones de la misma. En este caso, la carpeta se presenta

	vacía, ya que la aplicación es una extensión de *Redmine* y no requiere el uso de nuevas tablas adicionales de la base de datos, por lo que se trabajará directamente sobre la base de datos de *Redmine*.
lib/	Contiene el conjunto de librerías que permiten la extensión de la aplicación. Destaca la clase *hooks.rb*, que como su nombre indica contiene los elementos *Hook*, que es una *API* que permite que código externo pueda extender determinadas secciones de la aplicación de una manera limpia (en este caso, se emplea un *Hook de Vista* para cargar contenido de la carpeta *assets*).

Tabla 21. Organización y propósito de los archivos del proyecto

4.6.2. Estructuras de código del proyecto más relevantes

Para una mayor comprensión de todos los resultados obtenidos tras finalizar el proceso de *Implementación* del producto software desarrollado, se va a llevar a cabo la explicación de la relación existente entre las estructuras de código fuente que constituyen las funciones más importantes del sistema y las ventanas o componentes resultantes de su desarrollo.

Código fuente de Views (Vistas del proyecto)

Las *Vistas* del proyecto forman parte de la lógica de visualización, o cómo se muestran los datos de las clases del *Controlador*.

- **Clase *new.html.erb***

El fragmento de código de la *Figura 31* representa la implementación, en la ventana *Nueva petición*, del sistema que listará los subproyectos seleccionados previamente para aplicarles la propagación de la petición. Se lleva a cabo una llamada a la función `select_options` que filtrará solo aquellos proyectos clasificados como subproyectos, para ser mostrados en el caso de que hayan sido seleccionados.

Para la representación del listado correspondiente a la selección de subproyectos realizada, se hace uso del *parcial* (plantilla *html.erb* para la representación de datos concretos) *_projects_students_list.html.erb*, cuyo código fuente se detallará más adelante.

```
<!-- Listado de subproyectos seleccionados para llevar a cabo la propagacion de la issue: -->
<% if !@copy_from %>
  <p id="projects_form">
    <% select_options = project_tree_options_for_select([@issue.project] | @issue.project.descendants, :selected =>
       @issue.project) %>
    <%= f.select :project_id, select_options, {:required => false, :include_blank => false, :label => l("
      propagation_subprojects")}, :style => 'display:none;', :multiple => false %>
    <%= render 'projects_students_list' %>
  </p>
<% end %>
```

Figura 31. Fragmento de código del filtrado de subproyectos a propagar

El fragmento de código de la *Figura 32* refleja, por un lado, la creación del nuevo botón de *Crear y propagar* petición (`<%= submit_tag l(:button_create_and_propagate), ... %>`), el cual se habilitará si se cumplen determinadas condiciones, que se comentarán más adelante cuando se analice el código del *Controlador*.

Por otro lado, hay que destacar la creación del enlace/botón que permitirá acceder a la ventana modal con el listado de subproyectos del proyecto actual (`<%= link_to l(:button_propagate), ... %>`), donde se llevará a cabo la selección de aquellos a los que se quiera propagar la petición en cuestión.

```
<%= submit_tag l(:button_create_and_propagate), :name => 'continue', :style => 'display:none;', :id =>
  "createAndPropagate" %>
<%= submit_tag l(:button_create_and_continue), :name => 'continue_issue' %>
<%= preview_link preview_new_issue_path(:project_id => @project), 'issue-form' %>
<!-- Enlace para abrir la ventana de diálogo (showModal()) donde se mostrará la selección de
proyectos  -->
<!-- Se genera la ruta de la ventana de selección de proyectos (...selection_path(id)) al hacer clic
en el enlace, para identificar sobre qué proyecto se está operando:  -->
<% if @issue.project.descendants.any? && !@copy_from %>
  <%= link_to l("button_propagate"), plugin_teaching_extension_load_students_selection_path(:issue_id
    => @issue.id, :project_id => @issue.project.id), remote: true, id: "loadModalStudentsSelection",
    class: "load-modal-students-selection" %>
<% end %>
```

Figura 32. Fragmento de código con los botones de propagación de petición y acceso al listado de selección subproyectos

Para cargar la ventana de diálogo con el listado de subproyectos del proyecto actual, ha sido necesario desarrollar una función mediante la biblioteca *jQuery* de *JavaScript*, que además de permitir el acceso al listado de subproyectos a seleccionar cuando se hace clic en el enlace correspondiente, carga la *URL* de la ruta de esta ventana, permitiendo así identificar desde que proyecto (con sus correspondientes subproyectos) se está ejecutando. En la *Figura 33* se puede observar el desarrollo de esta función.

```
$("#loadModalStudentsSelection").click(function(e) {
  var ids = $('#issue_project_id').val();
  if(ids==null) {
    ids='';
  }
  var sep = ($(this).attr('href').indexOf('?') != -1) ? '&' : '?';
  $(this).attr('href', $(this).attr('href') + sep + 'project_ids=' + ids );
});
```

Figura 33. Función del acceso al listado de selección de subproyectos

Para dotar de funcionalidad al botón de *Crear y propagar petición*, se ha empleado la biblioteca *jQuery*, mediante la cual se ha desarrollado una función a partir de la creación y tratamiento de determinados eventos que interactúan al hacer clic en el botón (`$("#createAndPropagate").click(function(e) {...}`), permitiendo así crear y enviar el formulario de la nueva petición (`form.submit()`) a tantos subproyectos como se hayan seleccionado previamente para la propagación de la petición. En la *Figura 34* se puede observar el desarrollo de esta función.

```
$("#createAndPropagate").click(function(e) {
  var nextSubproject = $('#issue_project_id > option:selected').next('option');
  e.preventDefault();
  if (nextSubproject.length > 0) {
    $('#issue_project_id > option:selected').removeAttr('selected').next('option').attr('selected',
      'selected');
    form.submit();
  } else {
    $("#issue_project_id").val($("#issue_project_id option:first").val());
    form.submit();
    return;
  }
  setTimeout(function(){ document.getElementById('createAndPropagate').click(); }, 100);
});
```

Figura 34. Función de la creación y propagación de peticiones

- **Vista parcial *_projects_students_list.html.erb***

Como se comentó anteriormente en la vista *new.html.erb*, se ha empleado este parcial para la representación en la ventana de *Nueva petición* del listado correspondiente a la selección de subproyectos realizada. Como se puede observar en la *Figura 35*, se ha desarrollado una restricción para que sólo se muestren los subproyectos (descendientes) del proyecto actual (`allowed_projects = @issue.allowed_target_projects & @issue.project.descendants`) agrupados en un contenedor en línea.

```
<%
   other_projects = 0
   visible_projects = 0
   issue_projects = [@issue.project] | @issue.projects
   allowed_projects = @issue.allowed_target_projects & @issue.project.descendants
%>
<!-- Sección del listado de alumnos/subproyectos de la asignatura -->
<span id="my_projects_per_issue">
  <span id="list_of_projects_per_issue">
    <% issue_projects.each do |project| %>
      <% if allowed_projects.include?(project) %>
        <%= content_tag("span", project.name.gsub(/ /," ").html_safe, class: "list_projects_names") %>
        <% visible_projects += 1 %>
      <% else %>
        <% other_projects += 1 %>
      <% end %>
    <% end %>
  </span>
</span>
```

Figura 35. Código de la vista parcial del listado de subproyectos seleccionados

- **Función *load_students_selection.js.erb* que combina *jQuery* con *Rails***

Mediante el desarrollo de esta función se ha conseguido mostrar, en una ventana modal, la vista parcial que contiene el listado de subproyectos a seleccionar. Para ello, ha sido necesario especificar una función *en jQuery* que plasmara el contenido en *HTML* y *Rails* de esta vista parcial en dicha ventana modal, junto con el botón para aplicar los cambios de la selección de subproyectos. En la *Figura 36* se puede observar esta implementación.

```
$(document).ready(function(){
  $('#ajax-modal').html('<%= escape_javascript(render :partial => 'issues/modal_select_students') %>');
  showModal("ajax-modal", "600px");
  $("#button_apply").focus();
});
```

Figura 36. Función para representar la vista del listado de subproyectos en una ventana modal

- **Vista parcial *_modal_select_students.html.erb***

Para cargar correctamente el conjunto de subproyectos del proyecto sobre el cual se está llevando a cabo la creación de la petición en cuestión y, por tanto, realizar las operaciones oportunas, hay que definir una serie de restricciones y funciones.

En primer lugar, se ha realizado una llamada a la función del *helper* `render_subproject_nested_lists(projects)` (esta función del *helper* se explicará en un próximo apartado más detalladamente) que permite visualizar el conjunto de subproyectos anidados dentro del proyecto raíz sobre el que se está operando actualmente. Además, para cada uno de estos subproyectos, se ha definido un conjunto parámetros que permite leer *nombre* e *id* de los mismos para posteriormente mostrarlos con el formato adecuado.

En segundo lugar, se ha implementado el *contenedor* de este listado de subproyectos, que consistirá en un *contenedor de etiquetas* (`content_tag('label',...)`) donde se listará cada subproyecto junto con su *casilla de verificación* o *checkbox* (`check_box_tag('project_ids[]',...)`) para poder marcarlo o desmarcarlo, según esté implicado o no en el proceso de propagación posterior.

En la *Figura 37* se puede observar un fragmento de todo el proceso de codificación anteriormente mencionado.

```
nested_projects_list = render_subproject_nested_lists(allowed_projects|issue_projects) do |project|
  custom_fields_data = {}
  if allowed_projects.include?(project)
    custom_fields.each do |f|
      value = custom_values[project.id][f.id]
      value = value.join(",") if value.is_a?(Array)
      custom_fields_data.merge!(f.name.parameterize => value)
      value.split(",").each do |val|
        options_for_selects[f.name.parameterize] << val unless options_for_selects[f.name.parameterize].include?(val)
            || val.blank?
      end if value.present?
    end
  end
  content_tag('label',
              check_box_tag(
                  'project_ids[]',
                  project.id,
                  @issue != nil && issue_projects.include?(project) && allowed_projects.include?(project),
                  disabled: allowed_projects.include?(project) ? false : true,
                  #:class => ("inactive" unless allowed_projects.include?(project)),
                  data: custom_fields_data
              ) + ' ' + h(project.name), :class => ("inactive" unless allowed_projects.include?(project)), :style =>
                ("display:none;" unless allowed_projects.include?(project))
  )
end
```

Figura 37. Fragmento de código con el proceso de carga y formateo del listado de subproyectos

A la hora de seleccionar en el listado aquellos subproyectos implicados en la propagación de la petición, se han definido dos botones de enlace diferentes para ello: realizar una selección múltiple de todos los subproyectos para ejecutar una propagación total y desmarcar todos los subproyectos seleccionados del listado, además de proceder con una selección individual. La funcionalidad de estos dos botones responde a eventos de *JavaScript* diferentes, que se detallarán más adelante.

El proceso anteriormente mencionado se puede observar en el fragmento de código de la *Figura 38*.

```
<div class="actions_links">
  <%= l("Selection") %>:
  <!-- Botón de selección múltiple de subproyectos del listado: -->
  <%= link_to l("select_all"), '#', id: "link_select_all", :onclick => "select_all(event)" %>
  |
  <!-- Botón de desmarcar selección múltiple de subproyectos del listado: -->
  <%= link_to l("select_none"), '#', id: "link_select_all", :onclick => "select_none(event)" %>
  <!-- Selección individual de algunos de los proyectos de la caja de selección de proyectos: -->
  <hr>
  <% custom_fields.each do |field| %>
      |
      <%= select_tag field.name.parameterize, options_for_select(options_for_selects[field.name.parameterize]),
                    :prompt => field.name, :id => "select_#{field.name.parameterize}",
                    :class => "select_box_custom_field_value",
                    :onchange => "select_from_custom_field(event, '#{field.name.parameterize}')" %>
  <% end %>
</div>
```

Figura 38. Fragmento de código con eventos de selección múltiple, quitar selección y selección individual de subproyectos

Como se comentó en la *Figura 36*, dentro de la ventana de selección de subproyectos se ha creado un botón para aplicar los cambios resultantes de la selección llevada a cabo. Este botón, al igual que los botones de enlace de la *Figura 38*, responde a una función de *JavaScript* que se explicará más adelante. La definición de este botón se puede observar en la *Figura 39*.

```
<!-- Botón para guardar los cambios de la selección de proyectos para propagar Issue: -->
<p class="buttons">
  <%= submit_tag l(:button_apply), :name => nil, :onclick => "updateSelectedProjects();hideModal(this);",
                :type => 'button', :id => 'button_create' %>
</p>
```

Figura 39. Fragmento de código del botón de aplicación de cambios en la selección de subproyectos

Al llevar a cabo la selección individual de subproyectos del listado, se ha definido una estructura de control mediante una función en *jQuery*, donde al seleccionar el subproyecto en cuestión mediante su *casilla de verificación* se inhabilitan las casillas del resto de subproyectos, y al desmarcar dicha casilla se vuelven a habilitar el resto de casillas para continuar con la selección que sea necesaria. Esta función se puede observar en la *Figura 40*.

```
$("input:checkbox[name='project_ids[]']").change(function() {
  if($("input:checkbox[name='project_ids[]']:checked").length == 1) {
    $("input:checkbox:not(:checked)").prop('disabled', true);
  } else {
    $("input:checkbox:not(:checked)").prop('disabled', false);
  }
});
```

Figura 40. Función de la estructura de control de selección/quitar selección de subproyectos

En la *Figura 39* se mencionaba el botón empleado para aplicar los cambios de la selección de subproyectos llevada a cabo. Este botón, al hacer clic, responde a una función de *JavaScript* llamada `updateSelectedProjects()` (`function updateSelectedProjects() { ... }`). Esta función se compone de varias subfunciones internas, cada una de las cuales se encargará de un aspecto diferente a la hora de aplicar los cambios tras seleccionar los subproyectos deseados. A continuación, se van a comentar las características de cada una de estas subfunciones de `updateSelectedProjects()`:

En primer lugar, se obtienen todas las opciones de subproyectos disponibles, pasando el identificador de los mismos, como se puede observar en la *Figura 41*.

```
$("select#issue_project_id option").each(function() {
  this.selected = "";
});
```

Figura 41. Función de selección de subproyectos disponibles, mediante sus Ids

En segundo lugar, y como se puede observar en la *Figura 42*, para cada uno de los subproyectos del listado que es marcado mediante su correspondiente *casilla de verificación* o *checkbox*, se obtiene su valor y su estado de selección se establece a *true* (es seleccionado).

```
$("input:checkbox[name='project_ids[]']:checked").each(function() {
  $("#issue_project_id option[value=" + $(this).val() +"]").prop("selected", true);
});
```

Figura 42. Función de comprobación de subproyectos que hayan sido seleccionados

En tercer lugar, hay que establecer un formato *HTML* para mostrar en la ventana de *Nueva petición* el listado de subproyectos seleccionados. En el parcial *_projects_students_list.html.erb* (*Figura 35*), se definieron las restricciones pertinentes a la hora de mostrar cada uno de estos subproyectos agrupados mediante un contenedor en línea, pero faltaba definir el formato *HTML* necesario para ser representados en este contenedor. De este modo, cada subproyecto queda almacenado mediante un elemento `label` con su nombre. En la *Figura 43* se puede observar esta función.

```
var htmlContent = [];
$("input:checkbox[name='project_ids[]']:checked").each(function() {
  if($(this).val() != <%= @issue.project.id %>){
    htmlContent.push("<span class=\"list_projects_names\">" + $(this).closest('label').text() + "</span>" + "<br />"
      );
  } else {
    htmlContent.push("<span class=\"list_projects_names\" style='display: none'>" + $(this).closest('label').text()
      + "</span>");
  }
});

$("#projects_form #list_of_projects_per_issue").html(htmlContent);
```

Figura 43. Función para aplicar formato HTML a cada subproyecto almacenado en la ventana de Nueva petición

Por último, y para terminar con la explicación de la función `updateSelectedProjects()`, se ha implementado una estructura de control para tratar dos condiciones importantes a la hora de realizar la creación y propagación de la petición entre los subproyectos:

Como primera condición, se comprueba que no existen subproyectos seleccionados o que se ha seleccionado únicamente uno. En este caso, se deshabilita el botón de *Crear y propagar* permitiendo crear la petición únicamente en el proyecto padre (en caso de no seleccionar subproyectos para propagar la petición) o en el subproyecto elegido.

Como segunda condición, se comprueba que existe más de un subproyecto seleccionado, por lo que en este caso se habilita el botón de *Crear y propagar* para proceder con la creación de la petición en el proyecto actual y posterior propagación entre sus subproyectos, deshabilitando por otro lado el botón de creación individual de la petición. Todo el proceso de estas sentencias condicionales se puede observar en la *Figura 44*.

```
if(htmlContent == "" || htmlContent.length == 1) {
  $("#propagate").css("display", "none");
  $('input[value="Create"]').prop('disabled', false);
  $('input[name="continue_issue"]').toggle(true);
  $('input[name="continue"]').toggle(false);
} else if(htmlContent.length > 1) {
  $("#propagate").css("display", "inline");
  $('input[value="Create"]').prop('disabled', true);
  $('input[name="continue_issue"]').toggle(false);
  $('input[name="continue"]').toggle(true);
}
```

Figura 44. Sentencia condicional para habilitar el botón de Crear y propagar si existen subproyectos

En la explicación referente a la *Figura 38*, se comentó que la funcionalidad de los botones de selección múltiple y quitar selección múltiple de la ventana de selección de subproyectos, responden a dos funciones de *JavaScript*, para aplicarles el estado de *seleccionados* o *no seleccionados.* Estas dos funciones se pueden observar en la *Figura 45.*

```
// Función empleada al hacer clic en selección múltiple de subproyectos (todos seleccionados):
function select_all(event){
  event.preventDefault();

  $("input:checkbox[name='project_ids[]']").each(function()
  {
    $(this).attr("checked","checked");
  });
}

// Función empleada al hacer clic en desmarcar selección múltiple de subproyectos (ninguno seleccionado):
function select_none(event){
  event.preventDefault();

  $("input:checkbox[name='project_ids[]']:not(.inactive)").each(function()
  {
    $(this).attr("checked",false);
  });
}
```

Figura 45. Funciones de selección múltiple y quitar selección múltiple de subproyectos del listado

Por otro lado, a la hora de realizar una selección individual de algún subproyecto del listado, también se aplica otra función de *JavaScript*, detallada en la *Figura 46*.

```
function select_from_custom_field(event, id) {
  event.preventDefault();
  select_none(event);

  var selectors, selector, val;
  val = $("#select_"+id).val();
  selectors = [ "='"+val+"'", "^='"+val+"',", "$=',"+val+"'", "*=',"+val+",'" ];
  selector = $.map(selectors, function(e) {
    return "input:checkbox[name='project_ids[]']:checkbox[data-"+id+e+"]"
  }).join(", ");
  $(selector).each(function() {
    $(this).attr("checked","checked") ;
  });

  $(".select_box_custom_field_value").prop('selectedIndex',0);

}
```

Figura 46. Función de selección individual de un subproyecto del listado

Código fuente de Models (Modelos del proyecto)

El *Modelo* puede representar las tablas de la base de datos, migraciones (que expresan cambios en la base de datos) y observadores. Por ello, es necesario definir una serie de funciones mediante *Ruby* que interactúen con la base de datos para controlar aspectos importantes y determinantes de la accesibilidad y uso del proyecto.

- **Clase *issue.rb***

Por un lado, se ha implementado una función que permite controlar la visibilidad de una petición del proyecto actual o de cualquier otro proyecto, según el rol que haya adoptado el usuario logueado actualmente o cualquier otro usuario registrado en el sistema y la relación que mantiene con el proyecto de la petición en cuestión. Este tratamiento se puede observar en la *Figura 47*.

```
def other_project_visible?(usr = nil)
  other_projects = self.projects - [self.project]
  other_projects_visibility = false

  other_projects.each do |project|
    if other_projects_visibility == false
      other_projects_visibility = (usr || User.current).allowed_to?(:view_issues, project) do |role, user|
        if user.logged?
          case role.issues_visibility
          when 'all'
            true
          when 'default'
            !self.is_private? || (self.author == user || user.is_or_belongs_to?(assigned_to))
          when 'own'
            self.author == user || user.is_or_belongs_to?(assigned_to)
          else
            false
          end
        else
          !self.is_private?
        end
      end
    else
      break
    end
  end
  other_projects_visibility
end
```

Figura 47. Función de control de la visibilidad/accesibilidad de una petición según su relación con el usuario y su proyecto

Por otro lado, se ha implementado una función que permite obtener el conjunto de usuarios del sistema (tanto los usuarios del proyecto actual como los de otros proyectos) que deben ser notificados cuando se ven afectados por la asignación de una petición.

En la *Figura 48*, se puede observar el proceso anteriormente comentado, tanto la función de notificación a usuarios del proyecto actual como la función de notificación a usuarios del resto de proyectos del sistema.

```ruby
# Función que devuelve el conjunto de usuarios del sistema que deben ser notificados de una issue asignada:
def notified_users
  notified = issue_notified_users
  notified_from_other_projects = notified_users_from_other_projects - notified
  # Eliminar usuarios que no pueden visualizar la issue:
  notified_from_other_projects.reject { |user| !other_project_visible?(user) }
  notified_from_other_projects | notified
end

def notified_users_from_other_projects
  notified = []
  other_projects = self.projects - [self.project]
  other_projects.each do |pr|
    notified = notified | pr.notified_users
  end
  notified
end
```

Figura 48. Funciones de notificación a usuarios con petición asignada

Código fuente de Controllers (Controladores del proyecto)

Las clases del *Controlador* responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del *Modelo* y muestra los resultados usando las *Vistas*.

- **Clase *issues_controller.rb***

Para comenzar, y como se puede observar en la *Figura 49*, ha sido necesario definir la carga de proyectos en una petición. Para ello, se ha realizado una búsqueda de peticiones creadas y, de no estarlo, proceder a crearlas. Finalmente, se ha asignado cada proyecto a la petición en cuestión.

```ruby
def load_students_selection
  #@issue = Issue.find(params[:id])
  if params[:issue_id]
    @issue = Issue.find(params[:issue_id])
  else
    @issue = Issue.new
  end
  @issue.project = @project
end
```

Figura 49. Acción de carga de proyectos a partir de una petición

Posteriormente, ha sido necesario modificar la función de creación de peticiones (`def create ... end`) del *Controlador* de peticiones (*issues_controller.rb*) existente por defecto en *Redmine*, para adaptarla a los nuevos requerimientos del proyecto *Redmine Teaching Extension*.

Como ya se explicó en el código de la *Figura 32* de la *Vista new.html.erb*, se creó un nuevo botón de *Crear y propagar petición*, para invocar la lógica de creación y propagación del proyecto implementado. Este nuevo botón acompaña a los otros dos botones existentes por defecto de la ventana de *Nueva petición* de *Redmine* que son *Crear petición* y *Crear petición y continuar*.

De este modo, el usuario puede interaccionar de tres modos diferentes con el sistema a la hora de lanzar la creación y/o propagación de la nueva petición. Por esto, se ha implementado una estructura de control en la función de creación de peticiones anteriormente mencionada, para controlar la lógica de estos tres botones.

En la *Figura 50* mostrada a continuación, se puede observar la parte de la interacción del usuario con los botones de *Crear petición y continuar* y *Crear petición y propagar* donde se lleva a cabo una redirección a la *Vista* de *Nueva petición*. Como la lógica de los resultados mostrados al interaccionar con ambos botones es similar en ambos casos, se ha considerado agruparlos en el mismo condicional.

```
if @issue.save
  call_hook(:controller_issues_new_after_save, { :params => params, :issue => @issue})
  respond_to do |format|
    format.html {
      render_attachment_warning_if_needed(@issue)
      flash[:notice] = l(:notice_issue_successful_create, :id => view_context.link_to("##{@issue.id}",
                         issue_path(@issue), :title => @issue.subject))
      if params[:continue] || params[:continue_issue]
        attrs = {:tracker_id => @issue.tracker, :parent_issue_id => @issue.parent_issue_id}.reject {|k,v| v.nil?}
        if @issue.project.parent_id
          redirect_to new_project_issue_path(@issue.project.parent_id, :issue => attrs)
        elsif @issue.project
          redirect_to new_project_issue_path(@issue.project, :issue => attrs)
          #redirect_to project_copy_issue_path(@project, @issue)
        end
```

Figura 50. Fragmento de código de la interacción del usuario con los botones de Crear petición y continuar y Crear petición y propagar

En el caso del botón de *Crear petición*, la interacción del usuario con dicho botón simplemente crearía la petición en cuestión y, posteriormente, se haría una redirección a la *Vista* de la petición ya creada (`redirect_to issue_path(@issue)`)

Código fuente de Helpers (Helpers del proyecto)

Una módulo *Helper* [29] contiene la definición de determinadas funciones cuyo objetivo es ayudar a que el código de las *Vistas* cumpla el principio *DRY* (*Don't Repeat Yourself o No Te Repitas*) encapsulando para ello código repetitivo que se ejecute en las mismas.

- **Módulo *issues_helper.rb***

En este caso, ha sido necesario implementar dos funciones en el módulo *Helper*, que serán utilizadas en la *Vista parcial _modal_select_students.html.erb* a la hora de visualizar el listado de subproyectos a seleccionar.

Por un lado, y como se puede observar en la *Figura 51*, se ha definido una función encargada de mejorar la muestra de los valores de los subproyectos pertenecientes al listado. Esto permite identificar a cada uno de los subproyectos (mediante sus respectivos *id* y *nombre*), para representarlos adecuadamente.

```
def custom_values_by_projects(projects, custom_fields)
  values_by_projects = {}
  projects.each do |project|
    values_by_projects.merge!(project.id => {})
  end
  values = CustomValue.where("customized_type = ? AND customized_id IN (?) AND custom_field_id IN (?)",
                             Project.name.demodulize, projects.map(&:id), custom_fields.map(&:id) )
  values.each do |value|
    values_by_projects[value.customized_id].merge!(value.custom_field_id => value.value)
  end
  values_by_projects
end
```

Figura 51. Función del helper empleada para la identificación y muestra de subproyectos

Por otro lado, y como se puede observar en la *Figura 52*, se ha definido la función que lleva a cabo la construcción del formato empleado a la hora de mostrar los subproyectos del listado de selección, para la posterior propagación de la petición.

El desarrollo de esta estructura se hace identificando, primeramente, el proyecto raíz o proyecto padre desde donde se lleva a cabo la creación y propagación de la petición y, posteriormente, generando un listado ordenado y anidado de todos sus subproyectos.

```
def render_subproject_nested_lists(projects)
  s = ''
  if projects.any?
    ancestors = []
    original_project = @project
    projects.sort_by(&:lft).reverse_each do |project|
      @project = project
      if (ancestors.empty? || project.is_descendant_of?(ancestors.last))
        s << "<ul class='projects #{ ancestors.empty? ? 'root' : nil}'>\n"
      else
        ancestors.pop
        s << "</li>"
        while (ancestors.any? && !project.is_descendant_of?(ancestors.last))
          ancestors.pop
          s << "</ul></li>\n"
        end
      end
      classes = (ancestors.empty? ? 'root' : 'child')
      if(classes == 'root')
        s << "<li class='root'><div class='root'>"
      else
        s << "<li class='child'><div class='child'>"
      end
      #s << "<li class='#{classes}'><div class='#{classes}'>"
      s << h(block_given? ? yield(project) : project.name)
      s << "</div>\n"
      ancestors << project
    end
    s << ("</li></ul>\n" * ancestors.size)
    @project = original_project
  end
  s.html_safe
end
```

Figura 52. Función del helper empleada para la correcta visualización del listado de subproyectos

Código fuente de Config (Archivos de configuración del proyecto)

Los archivos de configuración de la aplicación, agrupan un sistema de enrutamiento mediante el archivo *routes.rb* y los archivos *locales* o archivos de traducciones, contenidos en la carpeta *locales*.

Los archivos *locales* [32], permiten hacer uso de la librería *i18n* de internacionalización cuya finalidad es dotar de múltiples idiomas al contenido del proyecto. Estos archivos particulares, presentan una extensión *.yml* y contienen una serie de instrucciones que permiten definir diferentes traducciones para las distintas cadenas de texto (referenciadas mediante etiquetas) del proyecto.

En el caso del proyecto *Redmine Teaching Extension*, en principio se han definido dos idiomas, el inglés y el español. A continuación en la *Figura 53*, se puede observar un fragmento del archivo *local* con las traducciones al inglés.

```
# English strings go here for Rails i18n
en:
  button_propagate: "Propagate issue to subprojects..."
  button_create_propagation: "Propagate"
  button_create_and_propagate: "Create and propagate"
  click: "* Click"
  propagate_issue: "to propagate the issue to"
  subprojects: "subprojects"
```

Figura 53. Fragmento del archivo en.yml con traducciones al inglés de elementos del proyecto

El *archivo de enrutamiento* o *enrutador routes.rb* permite reconocer *URLs* en el proyecto y enviarlas a su acción correspondiente del *Controlador*. El propósito de cada acción es recoger información para proveérsela a una *Vista*.

En el caso de *Redmine Teaching Extension*, al pulsar el botón que enlaza a la ventana del listado de selección de subproyectos, se dará el caso en el que el sistema reciba una petición entrante que provenga de la operación siguiente:

```
GET / plugin_teaching_extension_load_students_selection/project_ids=…
```

Esta petición entrante, pregunta al *enrutador* que la haga coincidir con una acción del *Controlador*. En este caso, la ruta será definida en el *enrutador routes.rb* de la siguiente manera:

```
get :plugin_teaching_extension_load_students_selection, to: "issues#load_students_selection"
```

Figura 54. Código de la ruta definida para el proyecto

Esta ruta indica que la petición entrante es enviada a la acción *load_students_selection* (explicada en la *Figura 49* y correspondiente a la acción de carga de proyectos en una petición de *Redmine*) del *Controlador issues_controller.rb*.

Código fuente de Lib (Archivos de configuración del proyecto)

La carpeta *lib* contiene un conjunto de librerías que permiten la extensión de la aplicación. Estas librerías se almacenan mediante módulos o clases.

- **Clase *hooks.rb***

Esta clase, como su nombre indica, contiene los elementos *Hook*. Se trata de una *API* que permite que código externo pueda extender determinadas secciones de la aplicación de una manera limpia.

En este caso, y como se puede observar en la *Figura 55*, se emplea un *Hook de Vista* para cargar el contenido de la carpeta *assets*, concretamente el contenido referente a la hoja de estilos *CSS*.

```ruby
def view_layouts_base_html_head(context)
  stylesheet_link_tag("multiusers_issue", :plugin => "redmine_teaching_extension")
end
```

Figura 55. Elemento Hook de Vista empleado para cargar contenido del CSS

Código fuente del Archivo init.rb de inicialización del proyecto

Cuando el sistema *Redmine* carga el proyecto correspondiente al *plugin Redmine Teaching Extension* para su ejecución, realiza primeramente una búsqueda del archivo *init.rb*.

La función de este archivo, escrito en lenguaje *Ruby*, es la inicialización del *plugin* mediante la declaración de información de relevancia (nombre del *plugin*, autor, versión, dependencias…) para registrarlo en el sistema y posteriormente ejecutarlo de forma correcta.

En el caso del archivo *init.rb* del proyecto *Redmine Teaching Extension*, se ha llevado a cabo una distinción entre dos módulos de inicialización. Por una parte, mediante la clase *ActionDispatch::Callbacks*, se han declarado todas las dependencias correspondientes a cada una de las clases (*Modelo, Vistas, Controlador, Helper* y *Hook*) implicadas en el proceso de implementación y que necesitan ser llamadas de este modo para inicializar el proyecto. Este módulo de código del archivo de inicialización se puede observar en la *Figura 56*.

```ruby
ActionDispatch::Callbacks.to_prepare do
  require_dependency 'controller_rte/issues_controller'
  require_dependency 'helper_rte/issues_helper'
  require_dependency 'model_rte/issue'
  require_dependency 'redmine_teaching_extension/hooks'
end
```

Figura 56. Código de devolución de llamadas a dependencias/clases necesarias para inicializar el proyecto

Por otra parte, y como se puede observar en la *Figura 57*, se ha creado una estructura en donde se ha declarado la información importante para proceder con el registro, la identificación y la inicialización del *plugin* (*nombre, autor, descripción, versión del proyecto, sitio web donde se encuentra alojado y sitio web del autor*) en el sistema sobre el que se va a ejecutar.

```ruby
Redmine::Plugin.register :redmine_teaching_extension do
  name 'Redmine Teaching Extension'
  author 'Mario Merino'
  description 'Trabajo Fin de Grado para extender el sistema de peticiones de Redmine mediante un plugin para su uso en docencia'
  version '0.0.1'
  url 'https://github.com/MarioMerino/RedmineTeachingExtension'
  author_url 'http://example.com/about'
  requires_redmine :version_or_higher => '0.0.1'
end
```

Figura 57. Estructura de código con el registro de la información más relevante del proyecto

5. Pruebas

5.1. Introducción

Las *pruebas del software* [33] abarcan la fase final del Ciclo de Vida del software que se define como el proceso de ejercitar o evaluar el sistema, por medios manuales o automáticos, para verificar que satisface los requerimientos o para identificar diferencias entre los resultados esperados y los que produce el sistema.

Estas pruebas son básicamente un conjunto de actividades dentro del desarrollo del software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo. Existen distintos modelos de desarrollo de software, así como modelos de pruebas y a cada uno le corresponde un nivel distinto de involucramiento en las actividades de desarrollo.

El objetivo de las pruebas es presentar información sobre la calidad del producto a las personas responsables de éste. Las pruebas de calidad presentan los siguientes objetivos: encontrar defectos o bugs, aumentar la confianza en el nivel de calidad, facilitar información para la toma de decisiones y evitar la aparición de defectos.

Teniendo esta afirmación en mente, la información que puede ser requerida es de lo más variada. Esto hace que el proceso de *testing* sea completamente dependiente del contexto en el que se desarrolla.

Para el desarrollo de este proyecto, se van a considerar tres niveles de clasificación de pruebas según su finalidad: *pruebas del sistema*, *pruebas de aceptación/validación*, *pruebas de instalación* y *pruebas de rendimiento*.

5.2. Pruebas del sistema

Las *pruebas del sistema* verifican el comportamiento del sistema en su conjunto. Se trata del nivel más adecuado para llevar a cabo la comprobación de los *requisitos no funcionales*. Así pues, se ha optado por plantear *pruebas de usabilidad*, *pruebas de legibilidad* y *pruebas de mantenibilidad* para verificar la calidad del sistema desarrollado.

5.2.1. Pruebas de usabilidad

El sistema deberá ser sencillo de usar por parte del usuario. Tiene que ofrecer facilidad en su aprendizaje para el buen uso del mismo, ofrecer un buen rendimiento durante su ejecución y ser flexible a nuevas necesidades y cambios que se presenten oportunos.

Como ha quedado demostrado durante la definición del propósito de *Redmine Teaching Extension* y la representación de su funcionalidad mediante el *diagrama de casos de uso*, el nivel de dificultad de aprendizaje del mismo es muy bajo pues, al fin y al cabo, se trata de una extensión del sistema *Redmine* lo cual implica llevar a cabo un desarrollo que se adapte e integre lo máximo posible al sistema ya desarrollado para su correcto funcionamiento.

Además, los nuevos componentes de la interfaz que conforman la extensión desarrollada, tienen como objetivo general automatizar y agilizar la funcionalidad existente de creación de peticiones de *Redmine* a subproyectos de un proyecto. Por ello, la navegabilidad llevada a cabo entre componentes del sistema para cumplir esta función, es mínima, y esto facilita la comprensión del proceso.

5.2.2. Pruebas de legibilidad

En esta prueba se evalúa el color de los textos, el contraste de los mismos con el del fondo y el tamaño de la fuente, que debe ser adecuado para su legibilidad por los usuarios del sistema.

Como se ha comprobado en las *pruebas de usabilidad*, el nuevo sistema desarrollado forma parte de una extensión o *plugin* del sistema *Redmine*, por lo que comparten formatos en componentes y ventanas de la interfaz para su adaptación al sistema.

En la *Figura 58* se va a mostrar una representación de la interfaz de *Redmine* con la integración de la nueva ventana de la extensión desarrollada y sus botones, donde se demuestra una buena legibilidad y usabilidad del sistema.

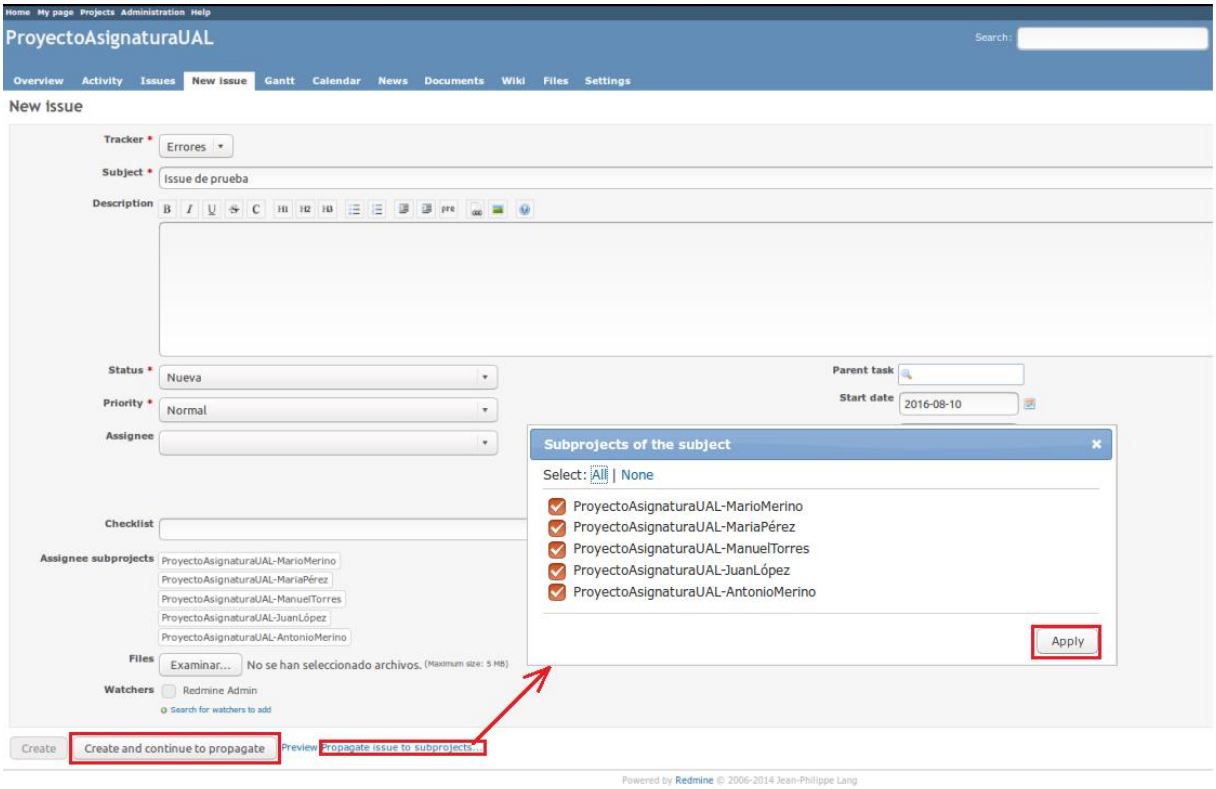

Figura 58. Representación de la integración de la interfaz del proyecto en el sistema Redmine

5.2.3. Pruebas de mantenibilidad

El sistema deberá requerir una cantidad de esfuerzo mínima para conservar su funcionamiento normal o para restituirlo cuando se presente algún tipo de error que altere su funcionamiento.

Mediante las *pruebas de mantenibilidad* se evalúa la facilidad con la que el sistema puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno.

Durante la explicación del código fuente del archivo *init.rb* del apartado *4.6.2*, se comentaron los principales aspectos de este archivo de inicialización del proyecto desarrollado, entre los que destacaba la declaración de la versión del sistema *Redmine* con la que presenta compatibilidad (`requires_redmine :version_or_higher => '0.0.1'`). De este modo, a la hora de aplicar alguna actualización o corrección de fallos en el proyecto o si el sistema *Redmine* es actualizado, simplemente hay que modificar el archivo *init.rb* para adaptarlo a los nuevos requerimientos, dotando al sistema de una buena mantenibilidad.

Una vez realizados los cambios pertinentes, solo quedaría reinstalar el proyecto en el sistema *Redmine*, siguiendo para ello los siguientes 3 pasos:

1) Copiar el directorio de *plugin* del proyecto dentro de la carpeta *plugins* del sistema *Redmine*.
2) Si el *plugin* requiere de una migración de base de datos, ejecutar el comando `bundle exec rake redmine:plugins:migrate RAILS_ENV=production` desde la carpeta raíz de *Redmine* para actualizar su base de datos con el contenido de las nuevas tablas requeridas por el *plugin*.
3) Reiniciar *Redmine* mediante el comando `sudo service apache2 restart`. Una vez reiniciado, se podrá observar el *plugin* instalado en la sección *Administración/Plugins* de *Redmine* (y configurar su funcionalidad, si el *plugin* tiene habilitado el botón de *Configuración*).

5.3. Pruebas de aceptación/validación

Las *pruebas de aceptación/validación*, se centran en las acciones visibles al usuario y salidas reconocibles desde el sistema. Los casos de prueba se obtienen a partir de los casos de uso del sistema (*requisitos funcionales*) y se pueden de forma manual o automatizada.

5.3.1. Automatización de pruebas

Para agilizar el proceso de pruebas, se van a llevar a cabo la automatización de las mismas. Esta automatización implica el uso del proyecto software para implementar las pruebas, controlar la ejecución de las pruebas y comparar los distintos resultados (todo dentro de un entorno de pruebas).

Para la ejecución automatizada se pueden emplear herramientas que permitan ejecutar pruebas de interacción del usuario, como puede ser *Selenium IDE*. Esta herramienta forma parte de la suite de herramientas de pruebas de software *Selenium* y se empleará para llevar a cabo las *pruebas de aceptación/validación* en *Redmine Teaching Extension*.

El conjunto de herramientas de *Selenium* es de código abierto y permite probar aplicaciones web de manera automatizada. Las pruebas de *Selenium* se ejecutan directamente en el navegador web donde esté corriendo el producto software y evitan el trabajo repetitivo de probar una y otra vez lo mismo manualmente.

5.3.2. Selenium IDE

Como se mencionó anteriormente, para realizar las pruebas que se aplicarán al proyecto *Redmine Teaching Extension* con su integración en el sistema *Redmine*, se va a emplear la herramienta *Selenium IDE* de *Selenium* [34].

Selenium IDE es un entorno de desarrollo integrado que está implementado como una extensión del navegador web *Firefox* y permite grabar, editar y depurar una secuencia de acciones sobre la aplicación web en cuestión y luego ejecutarlas de manera automática. Permite ejecutar tanto pruebas individuales como conjuntos de prueba completos mediante una suite de pruebas (colección de pruebas donde se ejecutan todas ellas como un lote continuo de trabajo).

Las suites de pruebas pueden definirse usando un simple archivo *HTML* que consiste en una tabla que define una lista de pruebas, donde cada fila define la ruta del sistema de archivos para cada prueba.

Al ejecutar la grabación de una prueba, se desarrollan automáticamente unos *scripts* en el lenguaje de scripting *Selenese*. De esta manera, se puede editar la grabación manualmente con sentencias y comandos (hacer clic en un enlace, seleccionar de una lista de opciones, verificar la presencia de un texto en particular…) para que su reproducción sea correcta.

La ejecución de las pruebas, ya sean individuales o mediante la suite de pruebas, se realiza desde una ventana integrada en el navegador web *Firefox*, donde se puede activar la función de grabación de las pruebas, visualizar el *log* de los comandos de *Selenese* ejecutados durante la grabación, editar los casos de pruebas, entre otras acciones. En la *Figura 59* mostrada a continuación, se observa la ventana de *Selenium IDE* con la ejecución exitosa de la suite de pruebas en *Redmine Teaching Extension* y un fragmento del *log* de los comandos de *Selenese*.

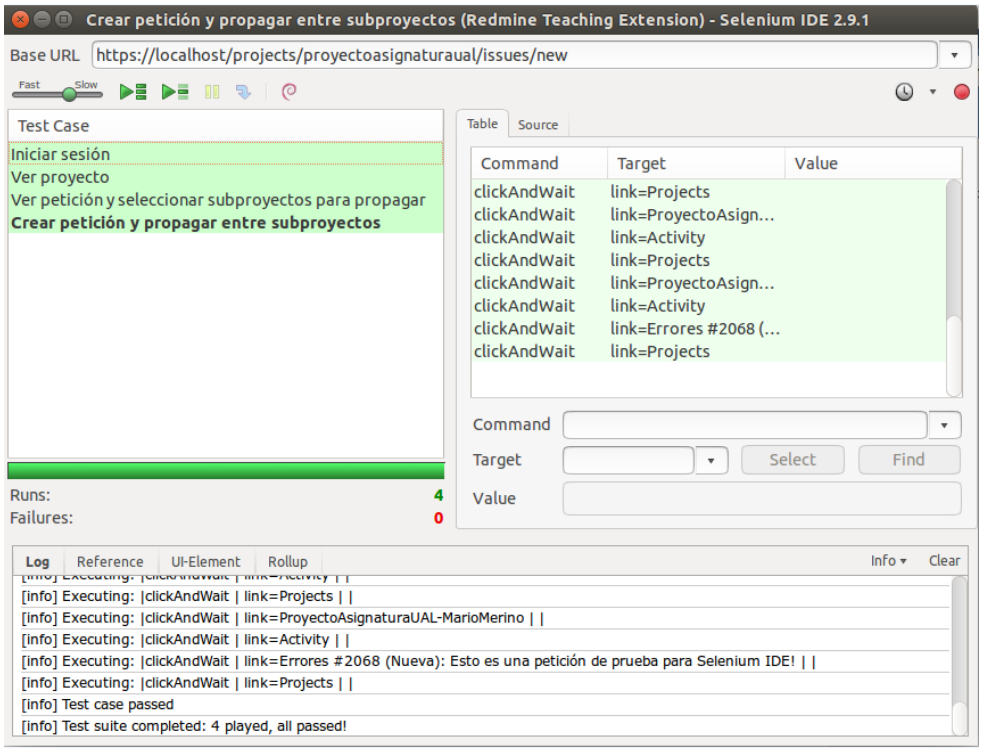

Figura 59. Ventana de Selenium IDE con la ejecución de la suite de pruebas del proyecto

Como se puede comprobar, cada uno de los casos de prueba de la suite de pruebas responde a cada *requisito funcional* del sistema. Además, se ha incluido la acción de *Iniciar sesión* como caso de prueba inicial para así hacer un recorrido completo de la funcionalidad del sistema, considerando que el usuario accede a *Redmine* para probar únicamente el funcionamiento del *plugin Redmine Teaching Extension*. Así pues, para concluir se van a mostrar todas las tablas en *HTML* con la representación de los casos de prueba de la suite de pruebas y sus comandos de *Selenese* resultantes.

Suite de pruebas de Redmine Teaching Extension

Para la suite de pruebas del proyecto *Redmine Teaching Extension*, se han considerado cuatro casos de prueba diferentes: *Iniciar sesión*, *Ver proyecto*, *Ver petición y seleccionar subproyectos para propagar petición* y *Crear petición y propagar entre subproyectos*.

Test Suite
Iniciar sesión
Ver proyecto
Ver petición y seleccionar subproyectos para propagar
Crear petición y propagar entre subproyectos

Tabla 22. Suite de pruebas de Selenium IDE para Redmine Teaching Extension

Caso de prueba de Iniciar sesión en Redmine

El primer caso de prueba consiste en la acción inicial del usuario de iniciar sesión en el sistema *Redmine*. El usuario accede al sitio web de *Redmine*, hace clic en el enlace para acceder a la ventana de *login*, introduce sus credenciales y hace clic en *Iniciar sesión*.

Iniciar sesión		
open	/	
clickAndWait	link=Sign in	
type	id=password	admin
type	id=username	admin
clickAndWait	name=login	

Tabla 23. Caso de prueba de Selenium IDE para Iniciar sesión en Redmine

Caso de prueba de Ver proyecto en Redmine

El segundo caso de prueba consiste en el acceso por parte del usuario a un proyecto de *Redmine* donde se quiere comprobar la funcionalidad del *plugin Redmine Teaching Extension*. El usuario accede a la sección de la página personal de *Redmine* y, desde ahí, hace clic en la sección *Proyectos* para finalmente acceder al proyecto en cuestión donde comprobar el funcionamiento de *Redmine Teaching Extension*.

Ver proyecto	
open	/
clickAndWait	link=My page
clickAndWait	link=Projects
clickAndWait	link=ProyectoAsignaturaUAL

Tabla 24. Caso de prueba de Selenium IDE para Ver proyecto en Redmine

Caso de prueba de Ver petición y seleccionar subproyectos para propagar petición en Redmine

El tercer caso de prueba abarca el proceso de acceder desde el proyecto seleccionado a la sección de crear *Nueva petición*, rellenar los campos pertinentes del formulario de la nueva petición e iniciar la interacción con la extensión de *Redmine Teaching Extension*, accediendo para ello a la ventana de selección de subproyectos del proyecto actual y seleccionando luego aquellos a los que se les desea aplicar la propagación de la petición que se pretende crear. Finalmente, se guardan los cambios y se sale de la ventana de selección de subproyectos.

Ver petición y seleccionar subproyectos para propagar		
open	/projects/proyectoasignaturaual	
clickAndWait	link=New issue	
type	id=issue_subject	Esto es una petición de prueba para Selenium IDE!
click	id=loadModalStudentsSelection	
click	id=link_select_all	
click	link=None	
click	id=project_ids_	
click	id=project_ids_	
click	id=link_select_all	
click	id=button_create	

Tabla 25. Caso de prueba de Selenium IDE para Ver petición y seleccionar subproyectos para propagar petición en Redmine

Caso de prueba de Crear petición y propagar entre subproyectos de Redmine

El último caso de prueba abarca el proceso de creación de la nueva petición y la posterior propagación de la misma entre todos los subproyectos seleccionados en el anterior caso de prueba. Para ello, el usuario hace clic en el nuevo botón habilitado de *Crear y continuar para propagar petición* donde se lleva a cabo la creación de la petición en el proyecto actual y, a su vez, la propagación de esta nueva petición entre cada uno de los subproyectos de forma automática. Finalmente, se accede al *Historial de actividad* de cada subproyecto para verificar que se ha generado correctamente la petición que se pretendía propagar.

Crear petición y propagar entre subproyectos	
clickAndWait	id=createAndPropagate
clickAndWait	link=Projects
clickAndWait	link=ProyectoAsignaturaUAL
clickAndWait	link=Overview
clickAndWait	link=Activity
clickAndWait	link=Errores #2069 (Nueva): Esto es una petición de prueba para Selenium IDE!
clickAndWait	link=Projects
clickAndWait	link=ProyectoAsignaturaUAL-AntonioMerino
clickAndWait	link=Activity
clickAndWait	link=Projects
clickAndWait	link=ProyectoAsignaturaUAL-JuanLópez
clickAndWait	link=Activity
clickAndWait	link=Projects
clickAndWait	link=ProyectoAsignaturaUAL-ManuelTorres
clickAndWait	link=Activity
clickAndWait	link=Projects
clickAndWait	link=ProyectoAsignaturaUAL-MariaPérez
clickAndWait	link=Activity
clickAndWait	link=Projects
clickAndWait	link=ProyectoAsignaturaUAL-MarioMerino
clickAndWait	link=Activity
clickAndWait	link=Errores #2068 (Nueva): Esto es una petición de prueba para Selenium IDE!
clickAndWait	link=Projects

Tabla 26. Caso de prueba de Selenium IDE para Crear petición y propagar entre subproyectos de Redmine

6. Resultados

6.1. Componentes de la interfaz de la aplicación

En este apartado se van a mostrar y explicar todos los elementos resultantes de la ejecución del *plugin Redmine Teaching Extension* y su integración con la interfaz del sistema de gestión de archivos *Redmine*.

En *Figura 60* se pueden observar (marcados en rojo) los dos nuevos componentes que han extendido la ventana de *Nueva petición* de *Redmine*.

Por un lado, se ha creado un nuevo contenedor en el formulario donde se van a reflejar todos aquellos subproyectos seleccionados para llevar a cabo la acción de creación y propagación de la petición (contenedor de *Subproyectos asignados*). Por otro lado, se ha creado un nuevo botón de enlace (*Propagar petición a subproyectos…*) junto al botón de *Vista previa* de la petición, cuya finalidad es abrir una ventana modal donde se va realizar la selección de los subproyectos del proyecto actual que van a estar implicados en el proceso.

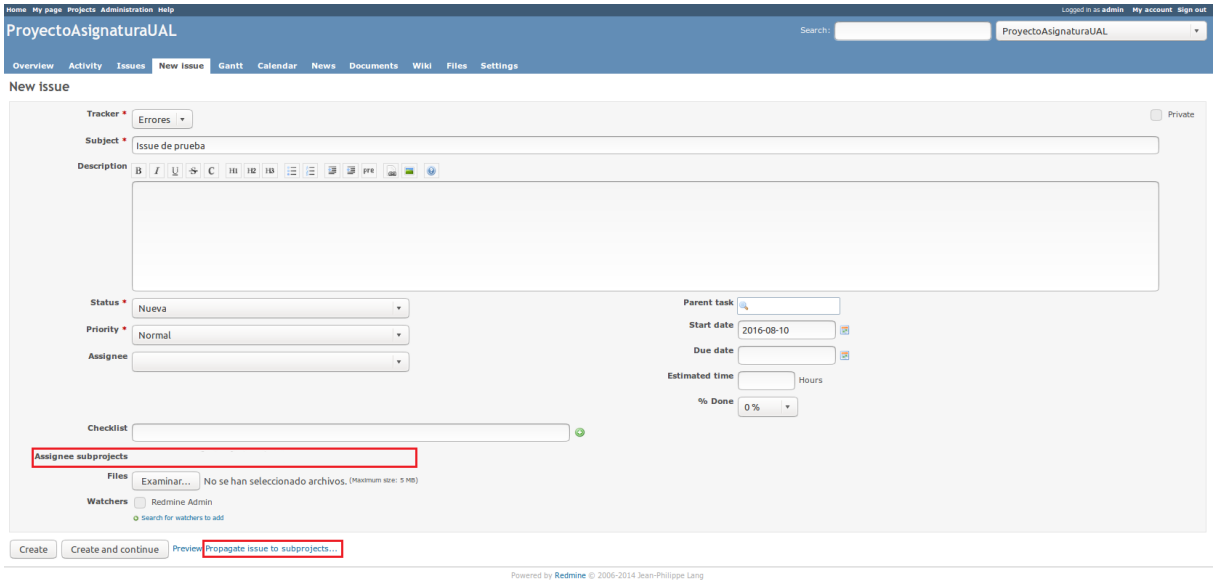

Figura 60. Representación del contenedor de subproyectos y el botón de selección de subproyectos

En la *Figura 61* se pueden observar (marcadas en rojo) las tres partes fundamentales de la ventana modal resultante de hacer clic en el botón de enlace de *Propagar petición a subproyectos*.

Por un lado, se ha creado un listado de todos los subproyectos del proyecto actual junto con sus respectivas casillas de verificación, para marcar o desmarcar aquellos que estarán implicados en el proceso de propagación posterior. Por otro lado, se ha creado un bloque superior con dos botones para, o bien seleccionar automáticamente todos los subproyectos o bien desmarcar todos los seleccionados (*Todos / Ninguno*). Finalmente, en la parte inferior de la ventana se encuentra el botón para guardar y aplicar cambios, que redirecciona al formulario de la petición.

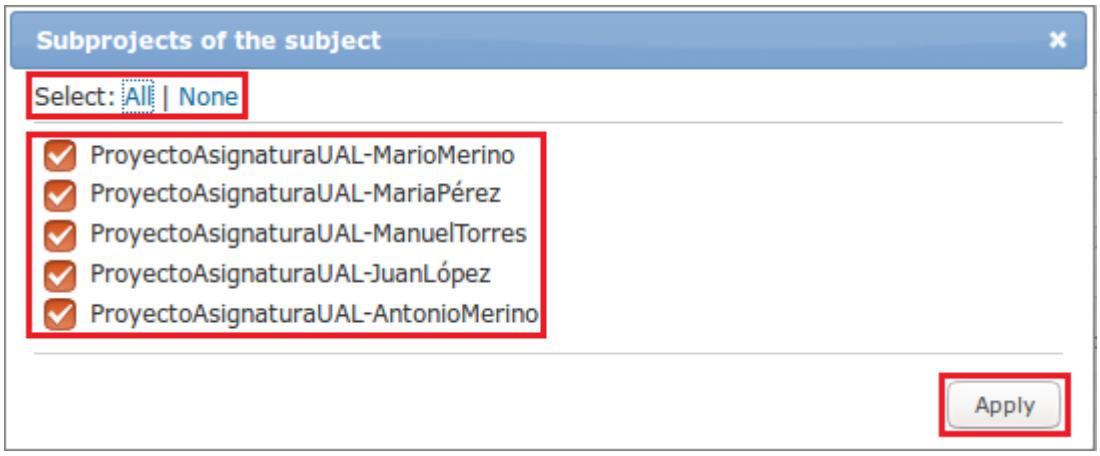

Figura 61. Representación de la ventana modal con el listado de selección de subproyectos

En la *Figura 62* se pueden observar (marcadas en rojo) las dos secciones del formulario de *Nueva petición* resultantes de realizar la selección de subproyectos y guardar los cambios (*Figura 60*).

Por un lado, el contenedor de *Subproyectos asignados* del formulario, se ha actualizado con el listado de los subproyectos que han sido seleccionados para aplicarles la propagación de la petición que se va a crear.

Por otro lado, se ha habilitado un nuevo botón (*Crear y continuar para propagar*) cuya funcionalidad va a consistir en crear la petición en el proyecto actual y propagar esa creación entre todos los subproyectos seleccionados para ello. Este nuevo botón va a sustituir al botón predeterminado de *Redmine* de *Crear y continuar* y, además, el botón predeterminado de *Crear petición* va a cambiar su estado a deshabilitado, ya que la acción que se pretender llevar a cabo es la de crear la petición y propagarla, por lo que no tiene sentido que esté operativo.

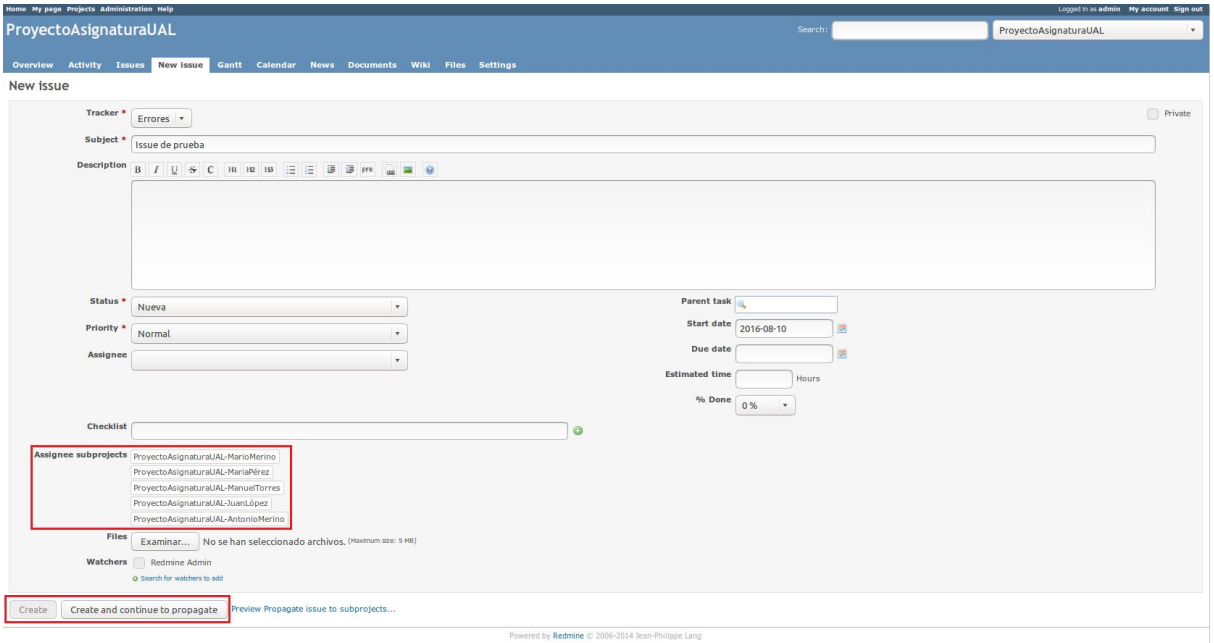

Figura 62. Representación del contenedor con el listado de subproyectos seleccionado y el botón de creación y propagación

Una vez creada y propagada la petición en cuestión, cada uno de los subproyectos afectados (así como el proyecto padre desde donde se creó la petición), tendrán registrada la misma petición (en el mismo instante de tiempo) en el historial de peticiones como resultado de la propagación. Un ejemplo de ello se puede observar en la *Figura 63* (detalles de la creación y propagación de la petición, marcados en rojo).

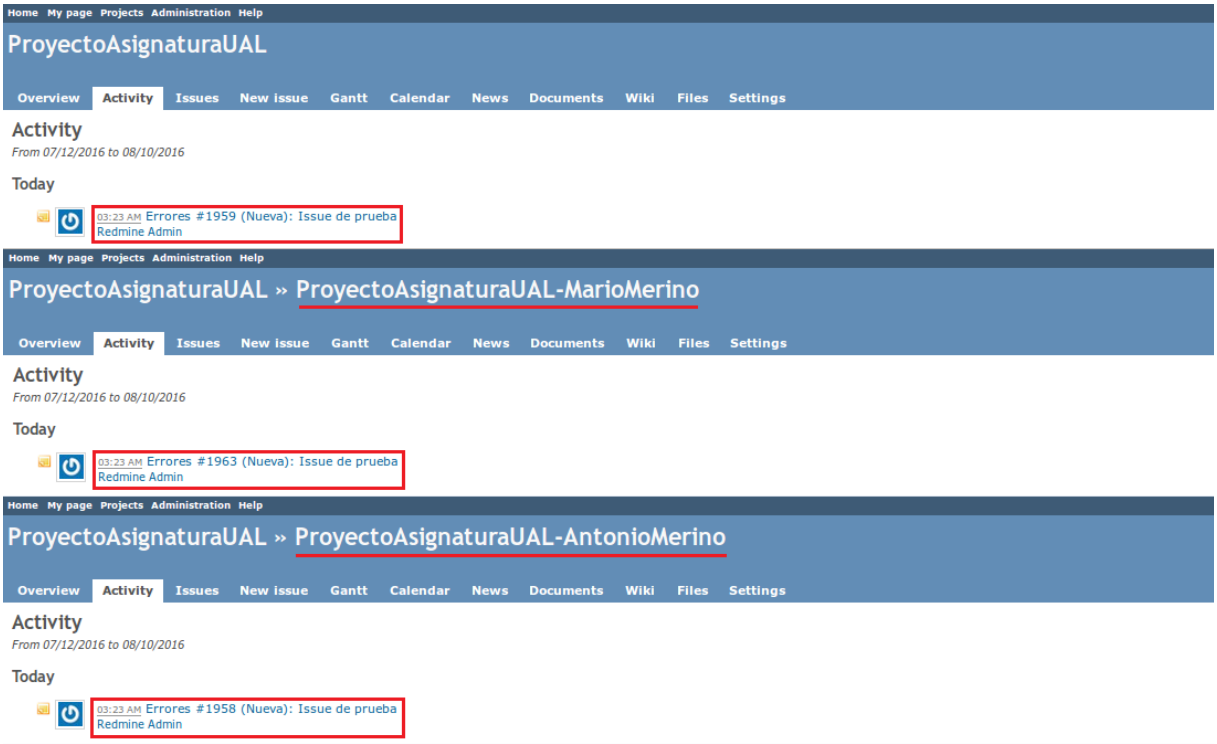

Figura 63. Representación del historial de la petición creada en el proyecto principal y sus subproyectos

Para finalizar, se va a mostrar la ventana de *Plugins* de la sección de *Administración* de *Redmine*, donde aparecen identificadas todas aquellas extensiones de *Redmine* que están actualmente instaladas y operativas en el sistema. Esta ventana se ha representado en la *Figura 64*, donde, en concreto, se puede destacar (en rojo) la fila correspondiente a la instalación de *Redmine Teaching Extension*.

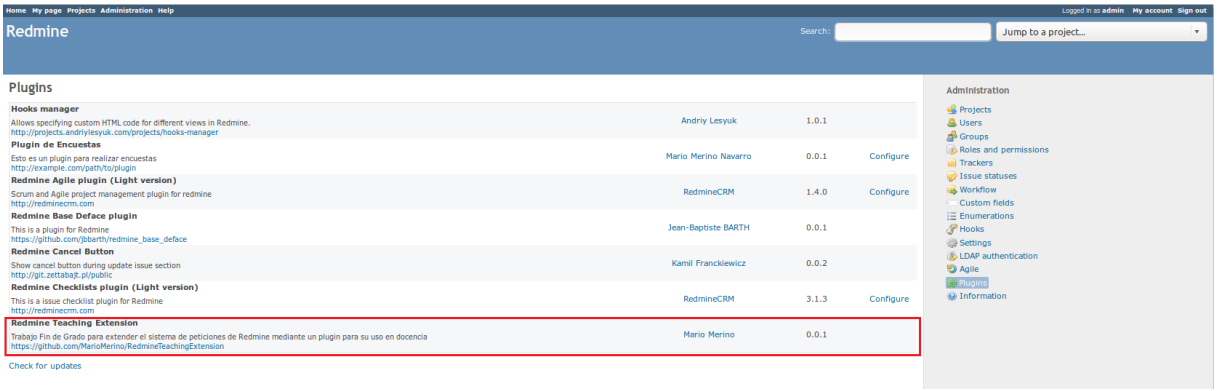

Figura 64. Representación de la ventana del listado de plugins de Redmine, con Redmine Teaching Extension instalado

7. Conclusiones y trabajo futuro

7.1. Conclusiones

Tras realizar este *Trabajo Fin de Grado* se han cumplido importantes objetivos, siendo el más importante la realización de un producto software a gran escala, haciendo uso de los conocimientos adquiridos en el Grado en Ingeniería Informática y cuya finalidad es ser empleado como herramienta de ayuda para los usuarios de la plataforma *Redmine* (orientado especialmente para su uso en la docencia del Grado en Ingeniería Informática de la Universidad de Almería). Además, el autor de este proyecto ha tenido que enfrentarse a numerosas tecnologías web y herramientas desconocidas hasta el momento, siendo destacable el empleo de una metodología de desarrollo basada en el patrón de arquitectura software *Modelo-Vista-Controlador* y el framework web *Ruby on Rails.*

El desarrollo de este proyecto ha supuesto al autor el aporte significativo de conocimientos sobre tecnologías, herramientas y metodologías de desarrollo, siendo destacables los siguientes:

- Aprendizaje de buenas prácticas en el desarrollo de aplicaciones web, en general.
- Aprendizaje del concepto de desarrollo de un producto software empleado como extensión de otra aplicación web (*plugin*).
- Conocimientos sobre el funcionamiento del framework web *Ruby on Rails*, así como ampliación de conocimientos en las tecnologías web *HTML*, *CSS* y *jQuery*.
- Aplicación de los conocimientos adquiridos durante el Grado en Ingeniería Informática, siendo especialmente importante el desarrollo de un producto siguiendo el Ciclo de Vida del software y el empleo de un patrón de arquitectura software como el patrón *Modelo-Vista-Controlador*.
- Conocimientos sobre el uso del sistema de control de versiones *Git*.
- Ampliación de conocimientos sobre el desarrollo de pruebas de un producto software.
- Aprendizaje de buenas prácticas en el proceso de documentación formal de un proyecto.

7.2. Trabajo futuro

Como trabajo futuro del producto desarrollado, se pueden plantear numerosas mejoras dependiendo de la utilización final que se le pretenda dar, ya que al tratarse de una extensión o *plugin* de un producto software existente como es *Redmine*, se puede variar su funcionalidad adaptándolo a nuevos requerimientos que puedan surgir en el sistema.

Ciñéndonos al objetivo del proyecto y su funcionalidad actual, se pueden exponer posibles ampliaciones como pueden ser las siguientes:

- Optimización de la funcionalidad de creación y propagación de tareas, habilitando para ello un nuevo botón en la ventana de *Nueva petición* del proyecto padre que, a golpe de clic, realice la propagación de la petición entre todos los subproyectos del proyecto, sin llevar a cabo un proceso de selección previo.

- Aplicar la función de propagación de peticiones a otras ventanas del sistema, como puede ser la ventana de *Copiar petición*. De este modo, se puede habilitar un sistema similar al ya implementado, permitiendo llevar a cabo la copia múltiple de la petición entre todos los subproyectos.
- Rediseñar los componentes de la interfaz desarrollada, adaptándola a las versiones más recientes del sistema *Redmine*.
- Dotar de múltiples traducciones al contenido del proyecto (aparte del inglés y español), ya que *Redmine* es software libre y de código abierto y su uso está muy extendido por todo el mundo.

8. Bibliografía

Referencias

[1] *Definición de Redmine*. Disponible en: https://es.wikipedia.org/wiki/Redmine. Última consulta: 25 de Julio de 2016.

[2] *Redmine Plugin Extension and Development.* De Alex Bevilacqua. Publicado por: Packt Publishing, el 19 de Marzo de 2014. Última consulta: 17 de Marzo de 2016

[3] *Sitio GitHub de Redmine Teaching Extension*. Disponible en: https://github.com/MarioMerino/RedmineTeachingExtension. Última consulta: 22 de Agosto de 2016

[4] *Definición de Plugin*. Disponible en: https://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica). Última consulta: 27 de Julio de 2016.

[5] *Sitio web del Directorio de Plugins de Redmine*. Disponible en: http://www.redmine.org/plugins. Última consulta: 27 de Julio de 2016.

[6] *Sitio web del plugin Redmine Checklists Plugin de Redmine*. Disponible en: https://www.redminecrm.com/projects/checklist/pages/1. Última consulta: 27 de Julio de 2016.

[7] *Sitio GitHub del plugin Multiprojects Issue Plugin de Redmine*. Disponible en: https://github.com/nanego/redmine_multiprojects_issue. Última consulta: 27 de Julio de 2016.

[8] *Sitio web del foro de plugins de Redmine*. Disponible en: http://www.redmine.org/projects/redmine/boards/3. Última consulta: 28 de Julio de 2016.

[9] *Sitios web de la herramienta Visual Paradigm For UML*. Disponible en: https://en.wikipedia.org/wiki/Visual_Paradigm_for_UML y https://www.visual-paradigm.com/features/. Última consulta: 30 de Julio de 2016.

[10] *Sitio web del Inspector de página de Mozilla Firefox*. Disponible en: https://developer.mozilla.org/es/docs/Tools/Page_Inspector. Última consulta: 30 de Julio de 2016

[11] *Sitio web del entorno de desarrollo integrado Ruby on Rails*. Disponible en: https://www.jetbrains.com/ruby/. Última consulta: 30 de Julio de 2016.

[12] *Sitio web de la herramienta de gestión ágil ScrumDo*. Disponible en: http://www.scrumdo.com/. Última consulta: 26 de Agosto de 2016

[13] *Sitio web del sistema de control de versiones GitHub*. Disponible en: https://github.com/open-source. Última consulta: 10 de Agosto de 2016.

[14] *Definición de UML*. Disponible en: https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado. Última consulta: 30 de Julio de 2016

[15] *Definiciones de HTML*. Disponible en: https://es.wikipedia.org/wiki/HTML y http://www.w3schools.com/html/default.asp. Última consulta: 1 de Agosto de 2016

[16] *Definiciones de CSS*. Disponible en: https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada y http://www.w3schools.com/css/default.asp. Última consulta: 1 de Agosto de 2016

[17] *Definiciones de jQuery*. Disponible en: https://es.wikipedia.org/wiki/JQuery, http://www.w3schools.com/jquery/default.asp y https://jquery.com/. Última consulta: 1 de Agosto de 2016

[18] *Definiciones de Ruby on Rails*. Disponible en: https://es.wikipedia.org/wiki/Ruby_on_Rails y http://www.rubyonrails.org.es/index.html. Última consulta: 2 de Agosto de 2016

[19] *Definiciones del patrón Modelo-Vista-Controlador y su aplicación en Ruby on Rails*. Disponible en: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador y http://www.softwaredeveloper.com/features/intro-to-ruby-on-rails-042507/. Última consulta: 2 de Agosto de 2016

[20] *Conceptos sobre APIs REST y definición de la API REST de Redmine*. Disponible en: http://asiermarques.com/2013/conceptos-sobre-apis-rest/ y http://www.redmine.org/projects/redmine/wiki/Rest_api. Última consulta: 2 de Agosto de 2016

[21] *Definiciones de MySQL*. Disponible en: https://es.wikipedia.org/wiki/MySQL. Última consulta: 3 de Agosto de 2016

[22] *Definiciones de Phusion Passenger*. Disponible en: https://en.wikipedia.org/wiki/Phusion_Passenger. Última consulta: 5 de Agosto de 2016

[23] *Proceso de instalación de Redmine con Apache y Phusion Passenger*. Disponible en: http://martin-denizet.com/install-redmine-2-5-x-with-git-and-subversion-on-debian-with-apache2-rvm-and-passenger/. Última consulta: 25 de Noviembre de 2016

[24] *Proceso de instalación de un plugin en Redmine*. Disponible en: http://www.redmine.org/projects/redmine/wiki/Plugins. Última consulta: 2 de Mayo de 2016.

[25] *Metodología para la Elicitación de Requisitos de Sistemas Software*. De Amador Durán Toro y Beatriz Bernárdez Jiménez. Publicado por: Departamento de Lenguajes y Sistemas Informáticos de la Facultad de Informática y Estadística de la Universidad de Sevilla, en Octubre de 2000. Última consulta: 8 de Agosto de 2016

[26] *La interacción persona-ordenador: El diseño*. De Miguel Gea y Francisco Luis Gutiérrez. Publicado por: Universidad de Granada. Última consulta: 12 de Agosto de 2016

[27] *Definición de diagrama de secuencia*. Disponible en: https://es.wikipedia.org/wiki/Diagrama_de_secuencia. Última consulta: 12 de Agosto de 2016

[28] *Definición de diseño del sistema*. Disponible en: https://es.wikipedia.org/wiki/Dise%C3%B1o_de_sistemas. Última consulta: 15 de Agosto de 2016

[29] *Definición de diagrama de clases*. Disponible en: https://es.wikipedia.org/wiki/Diagrama_de_clases. Última consulta: 15 de Agosto de 2016

[30] *Definición de enrutamiento en Rails*. Disponible en: http://guides.rubyonrails.org/routing.html. Última consulta: 20 de Agosto de 2016

[31] *Definición de helpers de Rails*. Disponible en: http://guides.rubyonrails.org/form_helpers.html. Última consulta: 20 de Agosto de 2016

[32] *Definición de archivos de lenguaje de Rails (archivos locales)*. Disponible en: http://codehero.co/ruby-on-rails-desde-cero-internacionalizacion/. Última consulta: 20 de Agosto de 2016

[33] *Definición de pruebas del software*. Disponible en: https://es.wikipedia.org/wiki/Pruebas_de_software y http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/moreno_a_jl/capitulo5.pdf. Última consulta: 23 de Agosto de 2016

[34] *Definición del IDE de pruebas Selenium*. Disponible en: https://es.wikipedia.org/wiki/Selenium y http://www.seleniumhq.org/. Última consulta: 24 de Agosto de 2016

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

Este documento muestra el desarrollo de un producto software consistente en una extensión o plugin para el sistema de gestión de proyectos *Redmine*. Se basa en un concepto de uso de *Redmine* donde las asignaturas sean concebidas como proyectos, y cada una de las actividades de las asignaturas sean tareas/peticiones de *Redmine*. De este modo, se ofrece un soporte para la gestión de peticiones asignadas, permitiendo tanto a los profesores de una asignatura como a sus alumnos, conocer el grado de evolución del trabajo encargado.

La propuesta de llevar a cabo este proyecto, viene de la necesidad de implementar una mejora en el sistema *Redmine* que permita al profesor definir las peticiones a nivel de asignatura y, posteriormente, dichas peticiones se propaguen automáticamente a los proyectos de cada alumno, concebidos como subproyectos del proyecto padre que es la asignatura en cuestión. En definitiva, se pretende agilizar el sistema de creación de peticiones entre alumnos de una asignatura.

GRADO EN INGENIERÍA INFORMÁTICA, 2015/2016