Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Departamento de Ciencias de la Computación

Proyecto Fin de Carrera

Adaptación y mejora de una herramienta de gestión de SCRUM

para la gestión de los proyectos fin de carrera.

Autor: Eduardo Rodrigo Contreras

Tutor: Jose Francisco Vélez Serrano

Codirector: Ángel Sánchez Calle

Agradecimientos

A mi tutor, Jose, por presentarme este reto, acompañarme y guiarme ante todas las dificultades que se nos han presentado.

A mi amigo, Jesús, por sus consejos, sus puntualizaciones y sus ayudas siempre con una sonrisa.

A Patricia, por intentar entenderme, apoyarme y animarme en todo momento.

A mi familia, por creer en mi.

Ningún hombre digno pedirá que se le agradezca
aquello que nada le cuesta.

Terencio.

Resumen

Este trabajo detalla la mejora de una aplicación para gestionar proyectos fin de carrera utilizando la metodología ***SCRUM*** apoyándose para ello en la Herramienta ***MAVEN*** y usando la técnica de arquitectura de software ***REST***.

Las metodologías ágiles surgieron con la idea de dinamizar la administración y gestión del desarrollo de proyectos de software Scrum. Pero si es importante el tener una correcta metodología a la hora de trabajar, también es importante, tener herramientas que permitan su correcta explotación y de una manera eficiente. Es en este sentido en el que el presente proyecto, centra sus esfuerzos.

Con este objetivo, se detallará como se ha mejorado la eficiencia de la arquitectura de una aplicación ya diseñada aplicando REST e independizando la parte del lado del servidor y la parte cliente apoyándonos en la biblioteca ***G***oogle ***W***eb ***T***oolkit e implementando el sistema Maven para facilitar la gestión de librerías. En cuanto a la interfaz REST se ha implementado un servlet que conectará con la API (encargada de las acciones sobre la Base de Datos) que hará las funciones de servidor devolviendo convenientemente los datos en formato JSON.

Índice general

Índice de figuras

Capítulo 1

Introducción

El desarrollo ágil de software consiste en un conjunto de métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requerimientos y soluciones evolucionan mediante la colaboración de grupos auto organizados y multidisciplinarios (ver Figura 1.1). Existen muchos métodos de desarrollo ágil, la mayoría minimiza riesgos desarrollando software en lapsos cortos. El software se desarrolla en unidades de tiempo llamadas una iteraciones, las cuales deben durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener una "demo" (sin errores) al final de cada iteración donde el equipo vuelve a evaluar las prioridades del proyecto.[13].

En nuestro caso, el método usado es Scrum, modelo de referencia que define un conjunto de prácticas y roles y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. Los roles principales en Scrum son el ScrumMaster, que mantiene los procesos y trabaja de forma similar al director de proyecto, el ProductOwner, que representa a los stakeholders (interesados externos o internos), y el Team que incluye a los desarrolladores.

Durante cada sprint, un periodo entre una y cuatro semanas (la magnitud es definida por el equipo), el equipo crea un incremento de software potencialmente entregable (utilizable). El conjunto de características que forma parte de cada sprint viene del Product Backlog, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar. Los elementos del Product Backlog que forman parte del sprint se determinan durante la reunión de Sprint Planning. Durante esta reunión, el Product Owner identifica los elementos del Product Backlog que quiere ver completados y los

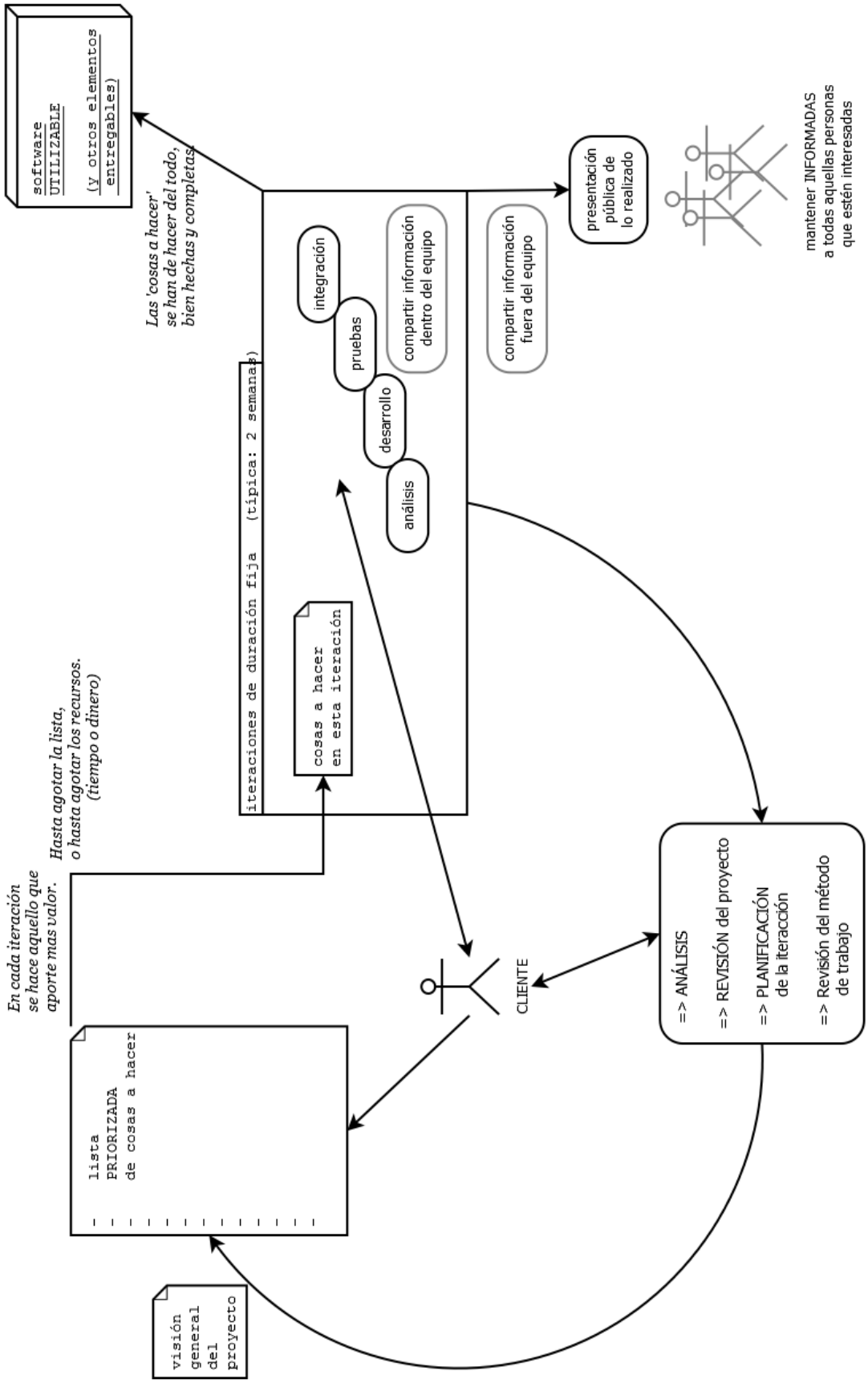

Figura 1.1: Esquema general de metodología ágil.

hace del conocimiento del equipo. Entonces, el equipo determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint en el cual nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el sprint.

Un principio clave de Scrum es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan (a menudo llamado requirements churn), y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, Scrum adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.

Existen varias implementaciones de sistemas para gestionar el proceso de Scrum, que van desde notas amarillas post-it y pizarras hasta paquetes de software. Una de las mayores ventajas de Scrum es que es muy fácil de aprender, y requiere muy poco esfuerzo para comenzarse a utilizar.

1.1. Requisitos Básicos de una herramienta de Gestión de Proyectos Ágiles

Para conseguir una herramienta que gestione proyectos ágiles se deben tener en cuenta ciertos criterios básicos para su concepción:

- Debe permitir trabajar con la metodología Scrum cumpliendo con todas sus necesidades siendo posible:
  - Crear iteraciones.
  - Desglosar las historias en tareas.
  - Llevar un historial de trabajo.
  - Controlar y gestionar la pila de historias.
  - Asignación de personas a trabajos.
  - Permitir el seguimiento de la ejecución de las tareas.
- Por otro lado, para lograr su difusión, se ha considerado que la herramienta debe ser accesible a todo el mundo que desee y para ello, su descarga y correspondiente uso y actualización debe ser totalmente gratuita.

- Además, para facilitar su uso, la herramienta debe ser:
  - Multiplataforma, pudiendo ejecutarse en cualquier tipo de computadora.
  - Tipo web, fácilmente controlable y manejable desde cualquier computadora con navegador web.

1.2. Objetivos

En este proyecto se pretende sentar las bases para el desarrollo de una herramienta que ayude a la gestión del desarrollo y seguimiento de los proyectos fin de carrera en la universidad. Para ello se ha partido de una herramienta, desarrollada en un PFC anterior, que permitía la gestión de proyectos que siguen la metodología Scrum. Así, las tareas desarrolladas se han concentrado en dos aspectos: por un lado mejorar la arquitectura y el funcionamiento interno de la aplicación, y por otro, adaptar el lenguaje y el interfaz de aplicación a su nuevo cometido. Para conseguir estos objetivos se han planteado los siguientes subobjetivos:

1. Mejorar la API de acceso a la Base de datos, dotándola de los métodos necesarios para su nuevo cometido.

2. Crear una capa servidora, que conecte la interfaz de usuario con la API de Base de datos.

3. Diseñar un nuevo interfaz con la nueva nomenclatura de los PFCs e implementarle el acceso a la nueva capa servidora.

Además, dado que hay una cantidad importante de librerías en todo este movimiento de datos entre el cliente y el servidor y de éste último a su vez con la API, se ha creído conveniente la idea de aplicar Maven[12] a todos los proyectos, como herramienta de software para la gestión del proyecto. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de uno o varios repositorios, que proveen acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores.

1.3. Resumen del resto del proyecto

En el capítulo 2 se presentan diferentes aplicaciones de gestión de proyectos que existen en la actualidad, intentando dar una visión global de las diferentes soluciones que hay en el mercado, con sus ventajas e inconvenientes. En el capítulo 3 se analizan los requisitos que se nos plantean una vez que se han fijado los objetivos, dividiendo los mismos en funcionales y no funcionales. En el capítulo 4 se presenta la arquitectura de la aplicación, donde se detalla los tres grandes subsistemas: datos, servicios e interfaz. En el capítulo 5 se puede entender con mayor detalle el diseño realizado en la aplicación lógica y su interfaz así como se explica el diseño del servidor, parte fundamental de este proyecto. En el capítulo 6 se presentan unos informes de resultados obtenidos tanto en Jenkins como en Sonar que nos ayudan a evaluar de manera objetiva el resultado del proyecto. En el capítulo 7 se encuentran diferentes líneas de trabajo futuro que se pueden seguir para la continuación del proyecto y una serie de conclusiones personales respecto al mismo. Al final del proyecto se anexan unos manuales de Uso e Instalación de la aplicación así como se proporcionan diferentes Urls donde se puede evaluar el resultado final del proyecto.

Capítulo 2

Estado del arte

Este capítulo tratará de mostrar las diferentes aplicaciones existentes para la gestión de proyectos en el mercado actual, dando una visión global de los distintos gestores de proyectos.

2.1. Sistemas de gestión de proyectos

En la actualidad existen aplicaciones gratuitas[7] y de pago, las mas significativas son:

- **Gratuitas**:
  - **Xplanner**: Es una Herramientas para Planificación Colaborativa de proyectos y el seguimiento para equipos que utilizan Extreme Programming.
  - **OpenProj**:(Windows,Mac,Linux) OpenProj es un proyecto libre y de código abierto de gestión de programas de software Serena Software Incorporated. El desarrollador afirma que OpenProj es un reemplazo de Microsoft Project y otras soluciones de proyectos comerciales. Ha sido descargado más de 1.250.000 veces en pocos meses desde su lanzamiento y se está utilizando en más de 142 países. Aunque no es igual a MS Project, este programa ofrece varias funciones útiles para la planificación de proyectos, programación y gestión.
  - **OpenWorkbench**:(Windows) Open Workbench ha sido diseñado para la programación y gestión de proyectos. Open Workbench está patrocinado por la División Clarity de CA. Se trata de una aplicación de código abierto que se ejecuta en la plataforma Windows, incluyendo Windows 2000 y XP. El software se puede descargar y utilizar de forma gratuita.

- **GanttProject**: (Windows, Mac, Linux) GanttProject tiene una interfaz fácil de usar, simplemente puedes iniciar la planificación de proyectos de inmediato después de la instalación. Este programa es una herramienta de gestión publicada bajo la licencia GPL. Por tanto, puedes descargarlo y utilizarlo de forma gratuita. Está disponible para múltiples plataformas, incluyendo Windows, Linux y Mac.
- **FusionDeskStarterEdition**: (Windows) FusionDesk Starter Edition proporciona la planificación del proyecto básico y algunas características de gestión. No podrá ser la solución adecuada si estás buscando un programa de gestión de proyectos completo, ya que carece de algunas características de gestión como los diagramas de Gantt y de gestión de recursos.
- **TaskJuggler**:(Linux) TaskJuggler está desarrollado principalmente para los sistemas Linux y Unix-like. El programa proporciona la planificación del proyecto y la solución de seguimiento de manera superior a las herramientas de uso común, como la edición de gráficos de Gantt. Se trata de un programa de gestión de proyectos de software para administradores de proyectos serios. Se trata de una aplicación de código libre y abierto. Puedes descargarlo y utilizarlo de forma gratuita.
- **ScrumWorks**: ScrumWorks es una herramienta de automatización que permite la colaboración en la planificación de un equipo utilizando los conceptos del proceso ágil Scrum.

Aplicaciones Web para la Gestión de Proyectos:

- **Redmine**: Es una de las web más populares basadas en la aplicación de gestión de proyectos. Incluye gráficos de Gantt y calendario para ayudar a la representación visual de los proyectos y sus plazos. Es compatible con múltiples proyectos. Redmine es una aplicación de código abierto y liberado bajo los términos de la GNU General Public License v2 (GPL) que se puede descargar y utilizar de forma gratuita. Es compatible con múltiples bases de datos incluyendo: MySQL, PostgreSQL o SQLite.
- **dotProject**: Se ha lanzado como una aplicación de software de código abierto que está disponible para descargar y utilizar de forma gratuita para cualquier uso. dotProject está programada en PHP y utiliza MySQL para

una base de datos back-end (aunque otras bases de datos como Postgres también podría ser utilizado). La plataforma de desarrollador de servidor recomendado incluye Apache, PHP y MySQL. El programador también os anima a utilizar un sistema operativo como Linux, FreeBSD, OpenBSD. Sin embargo, otros sistemas operativos como Windows, Mac también son compatibles.

- **ProjectPier** : Es una fuente abierta y libre de auto-organizada de aplicaciones PHP para la gestión de tareas, proyectos y equipos a través de una interfaz web intuitiva. El programa está escrito sobre la base de las tecnologías de código abierto PHP y MySQL. ProjectPier es un software de código abierto bajo la licencia pública. Usted no tiene que pagar un canon de licencia, se puede utilizar para cualquier proyecto y puedes modificar el código tu mismo si quieres.
- **Collabtive**: Es un administrador de web de código abierto basado en proyectos y programas de software de colaboración desarrollado por un grupo de los voluntarios. Es un software de código abierto y ofrece una alternativa a herramientas propietarias como Basecamp. Puedes descargarlo y utilizarlo de forma gratuita. El programa está escrito en PHP y JavaScript. Collabtive está dirigido a las pequeñas empresas y medianas empresas y autónomos.
- **Trac**: Es una herramienta de gestión de proyectos de peso ligero que se implementa como una aplicación basada en web, escrita en el lenguaje de programación Python. Ideal para la gestión de los desarrollos de software, es lo suficientemente flexible como para utilizarla para muchos tipos de proyectos. El programa ofrece características de administración de proyectos y de seguimiento de fallos, y más características se pueden agregar mediante el uso de plug-ins disponibles, como el del anti-spam para diagramas de Gantt, gestión de archivos y seguimiento de tiempo.

- **De Pago**:

  - **Jira**: basada en web para el seguimiento de errores, de incidencias y para la gestión operativa de proyectos.
  - **Basecamp**: una de las herramientas más utilizadas en la gestión de proyectos.

- **ActiveCollab**: aplicación web para la administración de proyectos y la gestión de equipos.
- **SourceRepo**: que permite la gestión de Scrum.
- **VersionOne**: soporta distintas metodologías, entre ellas Scrum.

A continuación detallaremos a los principales representantes de cada clase, Xplanner, Jira y VersionOne.

Xplanner

Herramienta GPL que nos permite gestionar proyectos de XP (Extreme Programation) y Scrum. Como características principales se puede destacar su modelo visual bastante simple, permitiendo la asignación de notas a historias y tareas, vistas de la estimación de las iteraciones, exportación del proyecto y contenido a ficheros con formato XML o PDF. Es un programa muy exigente a la hora de solicitar información al usuario para completar las historias. Por otro lado, tiene problemas a la hora de cumplir con ciertos requisitos necesarios en la metodología Scrum, como la falta de herramientas para la gestión de la Pila de Producto. Realiza una clara división de equipos y sus respectivas tareas, o división de tareas para cada equipos/desarrollador. Desgraciadamente, Xplanner es un proyecto que ha sido abandonado y las únicas actualizaciones que aparecen corresponden a soluciones de bugs y son desarrolladas por colaboradores particulares.

Jira

Es una aplicación basada en web para el seguimiento de errores, de incidentes y para la gestión operativa de proyectos. Jira también se utiliza en áreas no técnicas para la administración de tareas. La herramienta fue desarrollada por la empresa australiana Atlassian. Inicialmente Jira se utilizó para el desarrollo de software, sirviendo de apoyo para la gestión de requisitos, seguimiento del estatus y más tarde para el seguimiento de errores. Jira puede ser utilizado para la gestión de procesos y para la mejora de procesos, gracias a sus funciones para la organización de flujos de trabajo. Está basado en Java EE que funciona en varios bancos de datos y sistemas operativos. La herramienta dispone también de paneles de control adaptables, filtros de búsqueda, estadísticas, RSS y función de correo electrónico.

VersionOne

Es una herramienta de gran prestigio y alta calidad, pero con el inconveniente de ser una herramienta “de pago”. Actualmente es la herramienta líder de planificación

Software	Collaborative software	Issue tracking system	Scheduling	Project Portfolio Management	Resource Management	Document Management	Workflow system	Reporting and Analyses
Basecamp	Yes[12]	No	No	No	Yes[4]	Yes	No	No
Collabtive	Yes	No	No	No	No	Yes	No	Yes
GanttProject	No	No	Yes	No	Yes	No	No	No
Microsoft Project	No	No	Yes	No	Yes	No	No	Yes
OpenProj	No	No	Yes	No	Yes	No	No	No
Xplanner	Yes	No	Yes	No	Yes	No	?	?
dotProject	Yes	Yes	No	Yes	Yes	Yes	No	Yes
JIRA	Yes	Yes	Yes	No	No	No	Yes	Yes
Redmine	Yes	Yes	Yes	Yes	No	Yes	Yes	No
Trac	Yes	Yes	No	No	No	No	Yes	No

Figura 2.1: Comparación de Software de Gestión.

y control de proyectos que usan desarrollo ágil de software. Soporta las metodologías ágiles más populares: Scrum, XP, DSM y Ágile UP y dependiendo del uso que se quiera dar, VersionOne puede modificarse para ayudar a trabajar al equipo, permitiendo elegir características de cada una. Permite el trabajo en varios proyectos simultáneos e incorpora también planificación de entregas, planificación de iteraciones y diagramas Burndown. Existe la posibilidad de descargar una versión de prueba o"trial" de 30 días, que ademas limita las funciones disponibles. Asimismo, el precio de las ediciones definitivas es de 36 $ al mes, por usuario o bien de 745 $ más cuotas por mantenimiento y soporte, si se desea comprar de manera perpetua.

Sin embargo, como se puede observar tras analizar estas aplicaciones, ninguna es completa 100 %. Algunas, como XPlanner, debido a que omiten detalles necesarios para una correcta implementación de una metodología. Otras, como VersionOne, porque son herramientas caras. Es por ello, que surge la necesidad de crear una herramienta, que satisfaga las necesidades de ese grupo de usuarios, que por los motivos que fueren, no han encontrado una herramienta para la gestión de sus proyectos Scrum.

Por último en la Figura 2.1 se puede comparar aspectos y características de las herramientas mas significativas.

2.2. Sistema del que se parte

La comunicación actual de la aplicación contaba con una comunicación con la API a través de servicios (RPC) que no era demasiado eficiente y ese ha sido nuestro primer requisito de mejora de la aplicación. Para ello se ha creado, sobre un servlet, toda la gestión de la parte servidora de nuestra aplicación como explicaremos mas adelante.

Otro de los inconvenientes detectado fué la gestión de la gran cantidad de librerías que se usaban por lo que se ha implementado Maven en todos los proyectos.

Para darle una mejor usabilidad, se han renombrado algunos elementos del interfaz, haciendo mas fácil la usabilidad en el entorno en el que se va a usar la aplicación.

Capítulo 3

Requisitos y análisis del problema

Para realizar un correcto análisis, lo primero que debe hacerse es comprender el contexto. En ese sentido, hay que entender el funcionamiento de la metodología Scrum, que se detallará a continuación.

3.1. Introducción a Scrum

En los años 80 los japoneses Takeuchi y Nonaka estudiaron las prácticas de empresas con buenos resultados de rapidez y flexibilidad en la producción. Entre las empresas que estudiaron se puede destacar: Xerox, Canon, Honda, NEC, Epson, Brother, 3M y Hewlett- Packard. De dicho estudio extrajeron las bases de una metodología que llamaron Scrum la cual está especialmente diseñada para ayudar a agilizar todas las tareas del desarrollo de software. Antes de empezar a trabajar con Scrum es necesario conocer diferentes conceptos relativos a:

- Estructura de trabajo.
- Terminología específica de Scrum.
- División que realiza de las fases de desarrollo.

3.1.1. Estructura de trabajo

Equipos pequeños y auto-organizados Los equipos pequeños y formados por miembros de diferentes disciplinas tienen a conseguir excelentes resultados. La clave del éxito es en todos los casos la capacidad de cada equipo de **organizarse** por sí mismo y el hecho de que la **comunicación**es transparente. Esta dos características suponen la forma en que todos los participantes estén comprometidos en el proyecto y se encuentren motivados. Bajo esta ídea de equipo surge Scrum.

Importancia del cliente En la mayoría de las metodologías es el cliente el que tiene un papel determinante a la hora de recoger los requisitos del software. Sin embargo, en Scrum, sus funciones no acaban ahí. En un proyecto Scrum, se necesitará que el cliente interactúe habitualmente en la ordenación de los bloques de trabajo a realizar. Para ello, necesitará priorizar la entrega de unos bloques u otros.

Roles en Scrum Aunque algunos roles han aparecido con anterioridad, conviene agrupar todos y explicar brevemente sus funciones:

- Cliente: Persona interesada en la creación del producto y encargada de definir las historias que necesita su aplicación.
- Dueño del producto: Podrá ser el propio cliente, y su principal función será sustentar económicamente el proyecto, asó como aportar la visión de negocio.
- ScrumMaster: Sera el encargado principal de todo el producto. Es la persona encargada de arbitrar el proceso Scrum, controlar los avances y ayudar en la comunicación entre equipo y cliente. En definitiva, guía al equipo para que éste sea capaz de autogestionarse y solucionar los problemas que le vayan surgiendo.
- Equipo: es decir, los desarrolladores que divididos en grupos de entre 4 y 8 personas se encargan de implementar el software que de soporte a los objetivos previamente definidos.

Reunión Diaría Probablemente la mayor característica de la metodología Scrum es la reunión diaría de todo el equipo, también denominada Daily Scrum. Se realiza de la siguiente forma, tal y como se recoge en [6]:

- La reunión es diaría y se hace siempre a la misma hora. Todos los miembros del equipo deben acudir puntuales.
- Debe durar menos de 15 minutos y se suele realizar de pie, para mantener el máximo de concentración y atención.
- Todos los miembros cuentan qué están haciendo a los demás, así como los problemas que tienen, puesto que estos deben ser hechos públicos en cuanto se detectan.
- En la reunión se realizan las siguientes 4 preguntas clave a cada desarrollador:

1. ¿Qué has hecho desde ayer?
2. ¿Qué tienes planeado hacer hoy?
3. ¿Has encontrado algún problema hasta ahora?
4. ¿Cuánto te queda para terminar lo que estas haciendo?

3.1.2. Terminología básica

Como se ha observado al explicar los roles, Scrum posee su propia terminología. A continuación se detallarán cuáles son los términos más importantes:

- Producto: hace referencia al Proyecto/ Aplicación de software que se desea desarrollar utilizando la metodología ágil Scrum.
- Historia: Resultado de la recogida de requisitos para crear un producto, capturarán las funcionalidades requeridas en el proyecto. Generalmente almacenan un gran número de atributos necesarios para su control, como pueden ser: Id, Nombre, Estimación Inicial, Descripción, etc. Las historias son o deben ser independientes, es decir, deben aportar valor al producto incluso por separado. Generalmente estas historias se plasman en unas tarjetas en las que se recoge de forma concisa y clara, qué es lo que se quiere hacer.
- Pila del producto: el conjunto de estas historias conforma la lista de requisitos del producto, donde cada historia tendrá asignada una prioridad. Esta organización será hecha, como se ha indicado anteriormente, por los clientes
- Sprint (Ejecución de la iteración): de la Pila del producto se extraerán historias, que serán asignadas a Sprints para su desarrollo. Estos sprints de requisitos o "sprint backlog", son en realidad una lista en la que se detalla cómo se van a construir los diferentes requisitos del producto.
- Tareas: Los requisitos del product backlog se dividen en tareas preferiblemente de no más de 16 horas para una persona. Es decir, las tareas son subdivisiones de las historias que se reparten entre los grupos de trabajo de Scrum. Contienen aspectos que el cliente no tiene que preocuparse o siquiera conocer, consistiendo en pequeñas divisiones utilizadas únicamente por los desarrolladores para ayudarles en la implementación de las historias. Comparten con las historias algunas similitudes respecto a los atributos que contienen. En estas tareas, igualmente, se

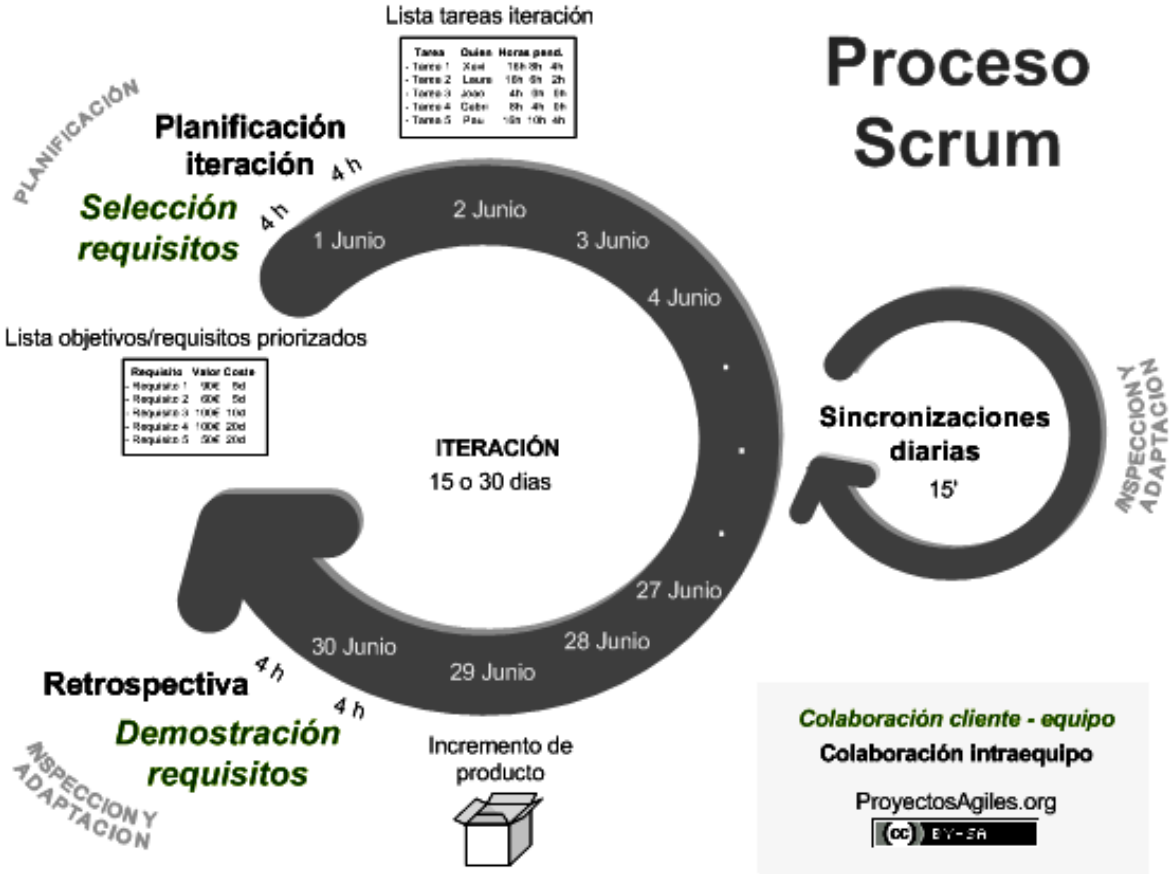

Figura 3.1: Iteraciones en Scrum.[4]

debe estimar el tiempo necesario para realizar cada una de las tareas, y será tarea del equipo intuir cuánto.

3.1.3. Etapas del desarollo

Es importante recordar que hablar de Scrum, es hablar de una metodología ágil, cuya principal característica es su capacidad para incorporar cambios con rapidez y en cualquier fase del proyecto. Para ello es importante trabajar en iteraciones bien planificadas y sincronizadas. (Ver Figura 3.1)

En este sentido, surgen distintas etapas de desarrollo de Scrum, entre las que se encuentran los Sprints y las Releases con sus respectivas funcionalidades.

- Un sprint es un período de tiempo de duración predefinida, suele ser recomendable que duren entre 2 y 4 semanas. Durante este periodo los equipos "esprintan". Como se acaba de mencionar, a cada Sprint se le asignan un grupo de historias que serán las que se implementarán durante dicho Sprint. Los Sprints son aspectos realmente importantes en las metodologías ágiles como Scrum, ya que

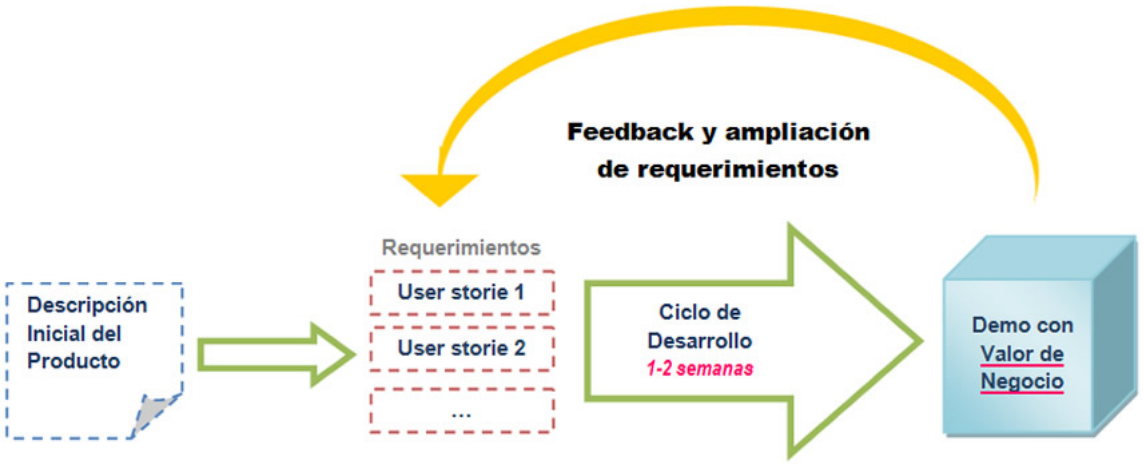

Figura 3.2: Proceso de desarrollo Scrum. [1].

agilizan el trabajo y obligan a los equipos del proyecto a trabajar centrados en un grupo de historias concreto, pudiendo éstos concentrar al máximo sus recursos y energías en vez de perder recursos en el resto de historias del producto. Normalmente cuando se finalizan uno o varios Sprints (dependiendo de la duración de cada uno y de lo que decida del ScrumMaster) se suele producir una entra de Releases al cliente.

- Una entrega de Releases consiste en una versión ejecutable de la aplicación/software que se encuentra en desarrollo. En cada Release se incluye de uno a varios Sprints, dependiendo de lo que el ScrumMaster y el cliente consideren adecuado. Las Releases son de gran importancia tanto para el cliente como para los grupos de desarrollo. Por un lado, para el cliente es esencial ver cómo la aplicación solicitada avanza y va creciendo según las semanas van pasando. Para el grupo de desarrollo significa fechas límites, añadiendo presión y también una sensación de control más exhaustivo de los avances que se van produciendo. Un sprint obliga a finalizar correctamente, con funcionalidad, el mayor número posible de las historias incluidas en cada sprint. Un resumen gráfico se puede ver en la Figura 3.2.

3.1.4. Tablero Scrum

Como se puede observar en la Figura 3.3 el tablero Scrum es un elemento de la metodología Scrum que permite observar de un vistazo el estado general del proyecto que se esta llevando a cabo.

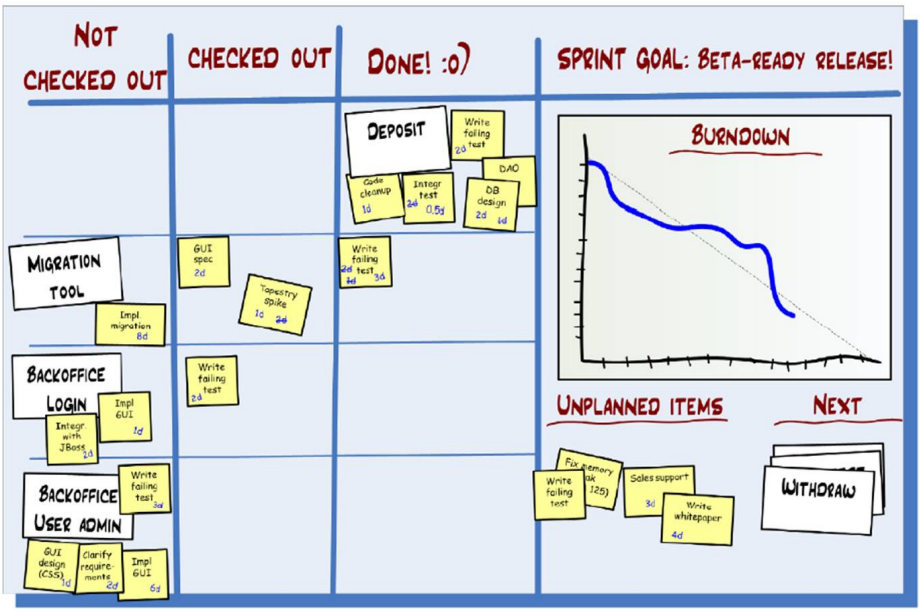

Figura 3.3: Tablero Scrum. [9].

El tablero se encuentra divido en 4 columnas principales, dónde las 3 primeras se dedican a los distintos estados de las historias y en la última se posiciona el diagrama de BurnDown y el resto de historias que no han sido incluidas inicialmente en el Sprint o que surgen durante el desarrollo del mismo. Las 3 primeras columnas son las correspondientes a los estados de: Asignado, Trabajando y Finalizado, que los desarrolladores usaran para mover las historias según se vaya trabajando con cada una. En la última columna se puede observar el gráfico BurnDown. Éste es un indicador del avance que se va realizando mostrado en un gráfico que aúna el tiempo consumido y el tiempo que resta para terminar las tareas del sprint.

Debajo de este gráfico se dispone una lista de Historias pendientes para próximos sprints, o en caso de terminar todas las del sprint actual, continuar trabajando. Por último se encuentra también una lista con historias o tareas que surgen a los desarrolladores durante el desarrollo del sprint.

El tablero en sí es la fuente principal de información. Se encarga de difundir el estado actual de la iteración, por lo que debe ser visible por todos los desarrolladores del equipo. Además de la información detallada hasta ahora, también podría incluir información como: la lista de objetivos priorizada del proyecto (Product Backlog), los diagrama de entidades del dominio, el objetivo de la iteración, un calendario con los principales eventos del mes, etc.

3.2. Requisitos funcionales y no funcionales

En este apartado veremos los diferentes requisitos que se han encontrado para el desarrollo del proyecto. En cuanto a los requisitos funcionales no han variado con respecto al proyecto anterior, respecto a los requisitos no funcionales se han redefinido de nuevo produciéndose aquí los mayores cambios respecto al PFC anterior.

3.2.1. Requisitos funcionales

Los requisitos funcionales se obtendrán de los requisitos necesarios para dar soporte a la metodología Scrum. Así, se tienen:

- Gestión de usuarios: se podrán crear, modificar y borrar los usuarios del sistema. Estos usuarios podrán ser personas físicas, o no. Por ejemplo, podría interesar mantener un control jerárquico de los empleados, a través de grupos. Estos grupos o departamentos, serán a todos los efectos usuarios de la aplicación. Los usuarios podrán acceder a la aplicación, presentando sus credenciales (nombre y contraseña).

- Gestión de productos: se diseñarán productos, que en el caso que nos ocupa serán llamados Proyectos, los cuales podrán ser modificados o borrados. El objetivo principal de los productos será el de crear una estructura que permita crear y definir claramente cada uno de los distintos proyectos que se quieran diseñar. Por tanto, cada producto tendrá sus propias historias, sprints y Releases que deben diferenciarse claramente y a los que se debe tener acceso desde la propia estructura.

- Gestión de Entregas: permitirá almacenar los distintos sprints que comprenden la entrega, accediendo a su creación, borrado y modificación de una manera rápida y cómoda. Permite también la modificación de la descripción y estimación.

- Gestión de Sprints: será otro punto importante en la aplicación, puesto que almacenará las distintas historias que los desarrolladores decidan incluir en dicho sprint. Asimismo permitirá modificar su estimación, trabajo realizado y documentación. Dada su importancia, conviene que se puedan distinguir los sprint de una persona concreta, que trabaje con él. En nuestro caso, mantendrá la información de las reuniones con el profesor.

- Gestión de Historias: cada historia tendrá sus propios atributos de descripción, estimación, planeamiento y asignación. Sera capaz de guardar una relación de las distintas subtareas o tareas que comprenden dicha historia. Con las historias se puede almacenar los distintos objetivos alcanzados en cada sesión.

- Gestión de Tareas: al igual que los predecesores almacenará atributos similares a los anteriores además serán las encargadas de guardar una relación de "etiquetas" de desarrollo, para determinar el avance en la creación del producto. En el caso que nos ocupa, no es necesario detallar tanto un proyecto.

- Gestión de Temas: antes de asignar historias a sprints, las historias estarán organizadas por temas. Desde ellas se podrán rellenar las fichas de las historias antes de pasarlas a los sprints.

- Gestión de Trabajo: supone la relación entre usuarios y productos. Es decir, los usuarios de la aplicación, que desempeñarán algún trabajo en el desarrollo del producto. Se deberá establecer desde la ventana del producto en cuestión. Los trabajos son útiles para que el profesor organice el trabajo antes de dárselo a los alumnos.

- Gestión de Asignación: hace referencia a la relación lógica entre los usuarios y los sprints. En la reunión diaria se determinará qué tareas hará qué persona.

- Gestión de Trabajo realizado: para ello será necesario soportar las reestimaciones de los usuarios con las tareas en las que trabajan. Estas reestimaciones se realizarán por usuarios de la aplicación que la irán creando conforme vayan realizando su trabajo.

3.2.2. Requisitos no funcionales

Una vez mencionados todos los requisitos funcionales de la API, se procede a enumerar los requisitos no funcionales del proyecto:

- División del proyecto: debido a los requisitos funcionales y objetivos anteriormente mencionados es interesante dividir el trabajo en dos capas: lógica y presentación. La parte lógica está formada por la API de ScrumManager, en la que se controla la lógica del proyecto. A su vez, la parte visual, o de presentación, constará de dos partes: la parte servidor y la parte cliente. La parte servidora

será la que maneje la API y la que independizará la lógica de la aplicación respecto al uso que pretenda dar el cliente. Éste, por su parte, podrá utilizar todos los servicios de la API a través de una interfaz que lo hará posible.

- Desarrollo en JAVA: principalmente porque se trata de un lenguaje muy utilizado, que garantiza robustez y, sobre todo, portabilidad. Asimismo, dispone de frameworks que trabajan sobre él, y que ayudarán a construir la interfaz de usuario.

- Lógica del proyecto: la lógica del proyecto será controlada mediante una API totalmente independiente de la parte visual. Este API comprenderá todas las clases y métodos necesarios para la gestión de Scrum.

- Presentación del proyecto: Como se ha mencionado, se mostrará vía web con JavaScript, que sera obtenida gracias a GWT. Por lo tanto, será programada en Java, y permitirá al servidor trabajar con la parte lógica del proyecto con facilidad.

- Comunicación Interface REST: Entre la parte cliente java y la parte servidora java se creará un interface REST de comunicaciones con objetos JSON que transportarán la información de manera transparente entre el interfaz y la base de datos haciendo uso de los distintos métodos de la API. La ventaja más significativa respecto a la anterior arquitectura es que en una única llamada podemos realizar la operación, sin tener que llamar recursivamente a un servicio para recuperar un dato de un objeto. En Subsubsección 4.2.2.1 se puede obtener más información.

- Almacenamiento en Base de Datos: cada una de las operaciones realizadas en la API bien sea de modificación, creación o borrado de cualquier elemento, atributo o valor de ésta, debe ser reflejado y almacenado en una base de datos adjunta a dicho proyecto. Dicha base de datos sera la responsable de respaldar todos los datos con los que se trabaje en la API. Dentro de los diferentes sistemas de Gestión de Bases de Datos se optó por utilizar MySQL, un sistema de gestión simple y eficaz que permite la creación y gestión de bases de datos dinámicas cómodamente. Aunque el diseño prevé el uso de otros Sistemas Gestores de bases de datos. El acceso a esta base de datos deberá ser transparente para el cliente

y, será administrada desde la propia capa lógica del proyecto. Así, a todos los efectos, el único que tendrá noción de la base de datos será el servidor.

3.3. Tecnologías y Herramientas

- Java: Será el lenguaje utilizado tanto en la parte lógica como en la de presentación.
- Base de Datos MySQL: que dará soporte al almacenamiento de la información de la aplicación.
- Driver JDBC: esta tecnología permite el trabajo entre las dos tecnologías más importantes de todo el proyecto, Java y MySQL. Con esta librería se puede trabajar con Java para interactuar con la base de datos, mientras sea de tipo MySQL.
- Servidor Web: para el desarrollo de esta aplicación se trabajará con Apache y Tomcat, aunque será este último el que dará soporte al servicio web.
- JavaScript: es el lenguaje interpretado que se utilizará para diseñar la parte de presentación de la aplicación.
- AJAX: Asynchronous JavScript And XML, técnica de desarrollo web para crear aplicaciones interactivas. Se ejecuta en el lado del cliente, es decir a través del navegador. Permitirá la comunicación cliente - servidor.
- Google Web Toolkit: ofrece a los desarrolladores la posibilidad de crear aplicaciones JavaScript con interfaces complejas, pero de gran rendimiento, y programadas bajo el lenguaje Java.
- Navegadores: que hacen posible la visualización de las distintas páginas de la aplicación. Además, estos navegadores ya llevan incorporados los intérpretes implicados en este proyecto. Concretamente, un intérprete de Javascript, capaz de entender el contenido que recibe de GWT.
- NetBeans: plataforma de Sun Microsystems para el desarrollo de aplicaciones software sobre Java, C++, Perl y Ruby. Es sobre la que se trabajará para las dos capas (lógica y presentación).

Antes de continuar, es necesario explicar con más detalle alguna de las tecnologías utilizadas. De esta forma, se profundizará en MySQL, AJAX, JavaScript, REST, Maven y GWT.

MySQL Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario Se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, aunque también se puede comprar una licencia específica. Está desarrollado en su mayor parte en ANSI C. Existen varias APIs que permiten acceder a las bases de datos MySQL, en distintos lenguajes de programación, como por ejemplo, C, C++, C#, Pascal, Delphi, Eiffel,Smalltalk, Java, Lisp, Perl, PHP, Python, Ruby,Gambas, REALbasic, Harbour, FreeBASIC, y Tcl. También existe una interfaz ODBC, llamado MyODBC que permite a cualquier lenguaje de programación que soporte ODBC comunicarse con las bases de datos MySQL. También se puede acceder desde el sistema SAP, lenguaje ABAP. MySQL es muy utilizado en aplicaciones web, debido a que en las aplicaciones web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

AJAX Asynchronous JavaScript And XML (Ajax), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA. Este tipo de aplicaciones se ejecutan en el lado del cliente, mientras que el servidor actúa de manera independiente. Por lo tanto es el navegador de los usuarios el que mantiene la comunicación asíncrona con el servidor. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones. El hecho de ser asíncrono, se traduce en que los datos recibidos del servidor se procesan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado en el que normalmente se efectúan las funciones de llamada de Ajax. El éxito de AJAX se debe a que los navegadores más importantes han estandarizado el objeto XMLHttpRequest, que permite hacer peticiones al servidor desde la pagina actual sin recargarla. Ademas, la respuesta puede ser tratada desde Javascript, bien sea como texto, bien sea como un objeto DOM (XML), es decir que no significa que el contenido asíncrono deba estar formateado en

XML.En la Figura 3.4 se aprecia las diferencias entre el modelo clásico de aplicación Web y el modelo AJAX de aplicación Web.

JavaScript Al igual que HTML, Javascript es un lenguaje de programación que se puede utilizar para construir sitios Web y para hacerlos más interactivos. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor. La sintaxis que maneja es similar a la utilizada en C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, y aunque es un error muy habitual, Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

REST La Transferencia de Estado Representacional (Representational State Transfer) o REST es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo. Si bien el término REST se refería originalmente a un conjunto de principios de arquitectura, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML o JSON y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP. Es posible diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST de Fielding y también es posible diseñar interfaces XMLHTTP de acuerdo con el estilo de llamada a procedimiento remoto pero sin usar SOAP. Estos dos usos diferentes del término REST causan cierta confusión en las discusiones técnicas, aunque RPC no es un ejemplo de REST. Los sistemas que siguen los principios REST se llaman con frecuencia RESTful.

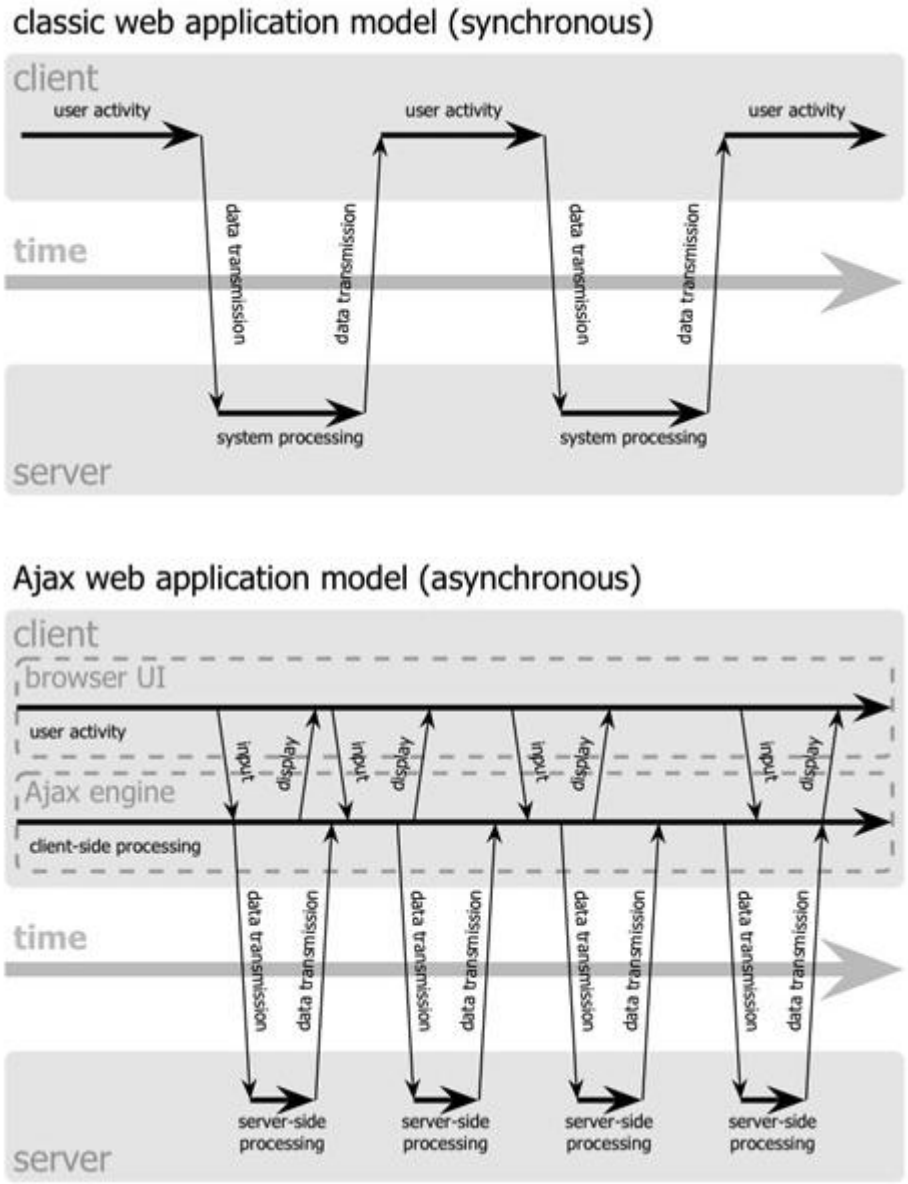

Figura 3.4: Modelo comparativo de aplicaciones Web. [5].

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave: Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST) Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema. Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI. El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

MAVEN Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de

un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio Maven 2, pugnan por ser el mecanismo de facto de distribución de aplicaciones en Java, pero su adopción ha sido muy lenta. Maven provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local. Ciclo de vida Las partes del ciclo de vida principal del proyecto Maven son:

1.compile : Genera los ficheros .class compilando los fuentes .java

2.test : Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.

3.package : Genera el fichero .jar con los .class compilados

4.install : Copia el fichero .jar a un directorio de nuestro ordenador donde Maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos Maven en el mismo ordenador.

5.deploy : Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto Maven con acceso a ese servidor remoto.

Cuando se ejecuta cualquiera de los comandos Maven, por ejemplo, si ejecutamos mvn install, Maven irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando sólo aquellas que no se hayan ejecutado previamente. También existen algunas metas que están fuera del ciclo de vida que pueden ser llamadas, pero Maven asume que estas metas no son parte del ciclo de vida por defecto (no tienen que ser siempre realizadas). Estas metas son:

1. clean : Elimina todos los .class y .jar generados. Después de este comando se puede comenzar un compilado desde cero.
2. assembly: Genera un fichero .zip con todo lo necesario para instalar

nuestro programa java. Se debe configurar previamente en un fichero XML qué se debe incluir en ese zip.

3. site : Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero pom.xml y ficheros .apt separados.
4. site-deploy : Sube el sitio web al servidor que hayamos configurado.

GWT La necesidad de generar aplicaciones Web dinámicas, ha llevado a la creación de una diversidad de frameworks que permiten acelerar el desarrollo de este tipo de aplicaciones. Un ejemplo de ellas es Google Web Toolkit (GWT). Google Web Toolkit es un framework creado por Google que permite ocultar la complejidad de varios aspectos de la tecnología AJAX. Por tanto, permite crear aplicaciones AJAX en el lenguaje de programación Java que son compiladas posteriormente por GWT en código JavaScript ejecutable optimizado que funciona automáticamente en los principales navegadores. Esto ya es un detalle significativo, puesto que cada navegador suele necesitar código específico para lograr un front-end correcto en una aplicación web. GWT construye una interfaz navegador cliente enriquecida con AJAX, sin embargo no puede crear una aplicación completa por sí mismo. Aun así debe tener un almacén de datos en el servidor y una cierta clase de framework para convertir esos objetos java que GWT pueda pasar desde el servidor a sus clientes GWT permite trabajar con Java, que cuando se desee se compilará en archivos JavaScript optimizados independientes. Google Web Toolkit te permite crear fácilmente tanto un artilugio para una página web como una aplicación completa. Sin embargo, GWT no sólo ofrece el compilador, también incluye:

- Un sistema de interfases de usuarios (UI), widgets estándar para trabajar en varios navegadores importantes.

- Un mecanismo de eventos para tomar y responder a eventos completamente en el lado del cliente.
- Un framework para el manejo de llamadas asincrónicas entre la aplicación Web y el servidor.

- Gestión del historial del navegador: Un mecanismo para la creación de estados históricos de navegadores de modo que la aplicación AJAX no sea ensuciada con el comportamiento esperado del botón atrás.
- Un framework test para usar Junit para escribir las pruebas de la aplicación cliente.

Asimismo, una aplicación GWT tiene tres partes:

- Parte cliente: desarrollada en java, luego compilado a JavaScript.
- Parte pública: ficheros públicos: imágenes, sonidos, etc.
- Parte Servidor:
  - Código que se ejecuta en el servidor.
  - Solo necesaria cuando va a haber interacción entre Cliente-Servidor (AJAX).
  - Ejecuta las librerías requerida.

Las aplicaciones GWT pueden ser ejecutadas en dos modos:

- Modo Hosted (Hosted Mode): En modo hosted, la aplicación corre como bytecodes de Java sobre una máquina virtual.
- Modo Web (Web Mode): En modo web, la aplicación corre como HTML + JavaScript sobre un navegador, traducido desde el código fuente Java original con el compilador de GWT. Cuando la aplicación está terminada, lo único que se debe hacer es subirla a un servidor web, y los usuarios finales accederán a ella a través de un navegador en "modo web".

El mayor tiempo de desarrollo de aplicaciones GWT se utilizará en el modo hosted. Por lo tanto, se estará interactuando y probando la aplicación GWT sin que ésta haya sido traducida a JavaScript. Para soportar el modo hosted, GWT cuenta con un navegador especial que está enlazado a la máquina virtual de Java.

SmartGWT SmartGWT es un framework que permite a los usuarios de GWT ampliar los widgets para su aplicación. En concreto, permite usar desde Google Web Toolkit la librería AJAX SmartClient. Los widgets que lo integran son

muy llamativos y abarcan desde un Calendar, animaciones, representaciones de árboles hasta componentes no visuales que ayudan a la comunicación entre la aplicación ejecutada desde el navegador, con el servidor; por ejemplo, un datasource para servicios REST u otro para utilizar los servicios JSON de Yahoo!. Asimismo, tiene un showcase, donde hay ejemplos de todos los componentes. Este producto se distribuye en dos versiones: uno open source bajo la licencia LGPL 3.0 y otro empresarial de pago. Las versiones empresariales se distinguen de la abierta en que incluyen un componente Java para el lado del servidor que permite una integración sencilla con tecnologías como Hibernate o Spring, además de incluir un plugin para IDEs que permite construir las interfaces de forma gráfica.

Capítulo 4

Arquitectura de la aplicación

Como se ha mencionado en el capítulo anterior, para la realización de la aplicación se han diferenciado dos capas : lógica y presentación. Sin embargo, estas dos capas, encapsulan en realidad tres grandes subsistemas: datos, servicios e interfaz.

4.1. Datos

La capa de datos está constituida principalmente por el contenido de la base de datos y por las clases de la API, que permiten su consulta, modificación, inserción y borrado.

La base de datos se puede crear al ejecutar el script ApisScrumManager.sql, de la que se obtienen las correspondientes tablas y relaciones entre ellas. En este sentido, a continuación se detallaran únicamente las relacionadas con Scrum.

- Tabla ScrumManager.Person: Tabla encargada de la descripción y almacenamiento de cada una de las personas involucradas en los proyectos. Este elemento de la API no comparte con el resto su herencia del elemento Item, por tanto su estructura es diferente.

- Tabla ScrumManager.Product: Encargada de la descripción y almacenamiento de los proyectos sobre los que se trabaje. Será necesario almacenar el nombre, la descripción, estimaciones, etc.

- Tabla ScrumManager.Working: Mantiene la relación de trabajo de una persona en un proyecto determinado. Por tanto, bastará que almacene la tupla persona-proyecto.

- Tabla ScrumManager.Assigned: Es la encargada de vincular a una persona con un sprint concreto. Así, bastará con que almacene el par sprint-persona.
- Tabla ScrumManager.Releases: Da soporte a las entradas o versiones que se entregaran al cliente. Al igual que los proyectos necesitará un nombre, las estimaciones, el estado en el que se encuentra, etc.
- Tabla ScrumManager.Spring: Almacenará la agrupación de historias que se realizarán en un periodo acotado de tiempo. Compartirá la mayoría de atributos con las entregas, aunque además, necesitará las fechas de comienzo y el fin.
- Tabla ScrumManager.Story: Tabla que dará soporte al almacenamiento de historias. Guardará el nombre, el estado así como la prioridad que se le da a esa historia.
- Tabla ScrumManager.Task : Dará soporte a la administración de tareas. Tendrá los mismos atributos que los proyectos, entregas, etc..
- Tabla ScrumManager.Theme: Tabla que almacenará los temas que agrupan a un conjunto de historias. Servirá para gestionar las tareas, según una determinada temática. Respecto a sus atributos serán los habituales : nombre, descripción, etc..
- Tabla ScrumManager.WorkDone: Permitirá la gestión de trabajo realizado. Así, almacenará la persona, la fecha y la tarea realizada.
- Tabla ScrumManager.Reestimation: Tabla que permite gestionar las reestimaciones diarias de cada tarea. También necesitará identificar la tarea a reestimar, la fecha, y la estimación.

4.1.1. Estructura de la Base de Datos.

A continuación resumiremos la estructura de la base de datos anexionada al proyecto y que, como se ha comentado anteriormente, se puede encontrar en el fichero ApiScrumManager.sql. En la Figura 4.1 podemos observar las diferentes tablas y las relaciones entre sí.

Dentro de la base de datos denominada ScrumManager se encuentran diversas tablas que dan soporte a la arquitectura creada para la API. A continuación vamos a detallar las tablas mas significativas:

- Tabla ScrumManager.Person:

  - Id: int (11), es un autoincrementable, que será el identificador de la tabla Person. Será la clave primaria de la tabla.
  - ParentId: int(11), que es una clave foránea hacia otra persona. Podrá admitir valores nulos, puesto que como se detalló anteriormente, una persona podrá pertenecer, o no, a otra persona. Sobre el tipo de borrado, por tanto es puesta a nulo. A priori, no interesa que al borrar una persona, se borren todas las que dependen de él.
  - Name: varchar(40), que define el nombre de cada elemento creado. Además, el nombre tiene un valor único, puesto que servirá como credencial para identificarse en la aplicación.
  - Password: varchar(50), que es la contraseña de una persona concreta.
  - Stored: tinyint(1), booleano que indica si el elemento se encuentra almacenado correctamente en la base de datos.

- Tabla ScrumManager.Product:

  - Id: varchar (40), que almacena un código de referencia, que será la clave primaria de la tabla.
  - Number: int(11), para conocer la posición de ordenación.
  - Description: text, permite almacenar una descripción extensa de cada uno de los elementos.
  - InitialEstimation: float, define la estimación inicial que consideran el equipo.
  - CurrentEstimation: float, define la actual fecha de entrega.
  - Assigned: tinyint(4), booleano que indica la asignación del elemento a algún equipo de desarrollo.
  - Planned: tinyint(4), booleano indica la planificación de este producto.
  - Done: tinyint(4), booleano indica la finalización del desarrollo del producto en cuestión.
  - DocUrl: text, permite definir un documento donde se encuentra la documentación sobre el desarrollo.

- Stored: tinyint(4), booleano indica si el elemento se encuentra almacenado correctamente en la base de datos.
- Name: varchar(40), define el nombre de cada elemento creado.

- Tabla ScrumManager.Working:

  - IdProduct: int(11), que es una clave foránea hacia un producto. No admitirá valores nulos.
  - IdPerson: int(11), es una clave foránea hacia una persona. Tampoco podrá admitir valores nulos.

  En este caso, la clave primaria será la tupla formada por ambos identificadores. Esto se debe a que en este dominio, una persona podrá trabajar en varios proyectos y, evidentemente, un proyecto podrá contener a varias personas trabajando en él.

- Tabla ScrumManager.Assigned:

  - IdSprint: int(11), que es una clave foránea hacia un sprint. No admitirá valores nulos.
  - IdPerson: int(11), es una clave foránea hacia una persona. Tampoco podrá admitir valores nulos.

  La clave primaria será la tupla formada por ambos identificadores. Una persona puede ser asignada a varios sprint, y un sprint puede contener a distintas personas.

- Tabla ScrumManager.Theme:

  - Id: varchar (40), que almacena un código de referencia, que será la clave primaria de la tabla.
  - Description: text, permite almacenar una descripción extensa de cada uno de los elementos.
  - Stored: tinyint(4), booleano indica si el elemento se encuentra almacenado correctamente en la base de datos.
  - Name: varchar(40), define el nombre de cada elemento creado.

- ParentId: int(11), que es una clave ajena a un tema. Esto es así, porque un tema puede contener subtemas, que faciliten la administración de las mismas.
- ParentProduct: int(11), clave foránea hacia un producto, que es el contexto de un tema, en caso de no ser un subtema. El borrado y modificación será en cascada.

4.2. Servicios

La capa de Servicios esta formada por toda la lógica de la aplicación, así, por un lado disponemos de la API y por otro lado el servidor.

Se tratará por tanto de toda la lógica relacionada con dar respuesta a las peticiones que pueda realizar el cliente.

En este sentido se necesitaran métodos para poder crear, modificar y borrar datos relacionados con la aplicación.

4.2.1. Servicios API

En la API existe una división entre las clases implementadas para la gestión de los elementos y atributos de la planificación y control de las metodologías cubiertas y otro grupo de clases encargadas de gestionar toda la persistencia de ese continuo movimiento de datos en el proyecto. Para facilitar en gran medida el trabajo y las operaciones con todos los elementos de gestión y planificación se necesitan varios tipos de clases:

- Clases para gestionar los temas y su división en historias: Para el control y gestión de los temas y su "división" o agrupación de historias dentro de un tema son necesarias dos clases principalmente: Theme.java y Story.java.
- Clases para gestionar el desarrollo de los proyectos: Para la gestión de nuevos proyectos en la API es necesario crear una clase que domine y controle todos los productos o proyectos. Para ello será necesario usar otras dos clases como son ScrumManager.java y Product.java.
- Clases para gestionar el tiempo/corte de desarrollo: Respecto a la gestión de costes y tiempo de desarrollo, será responsabilidad de cada desarrollador rellenar

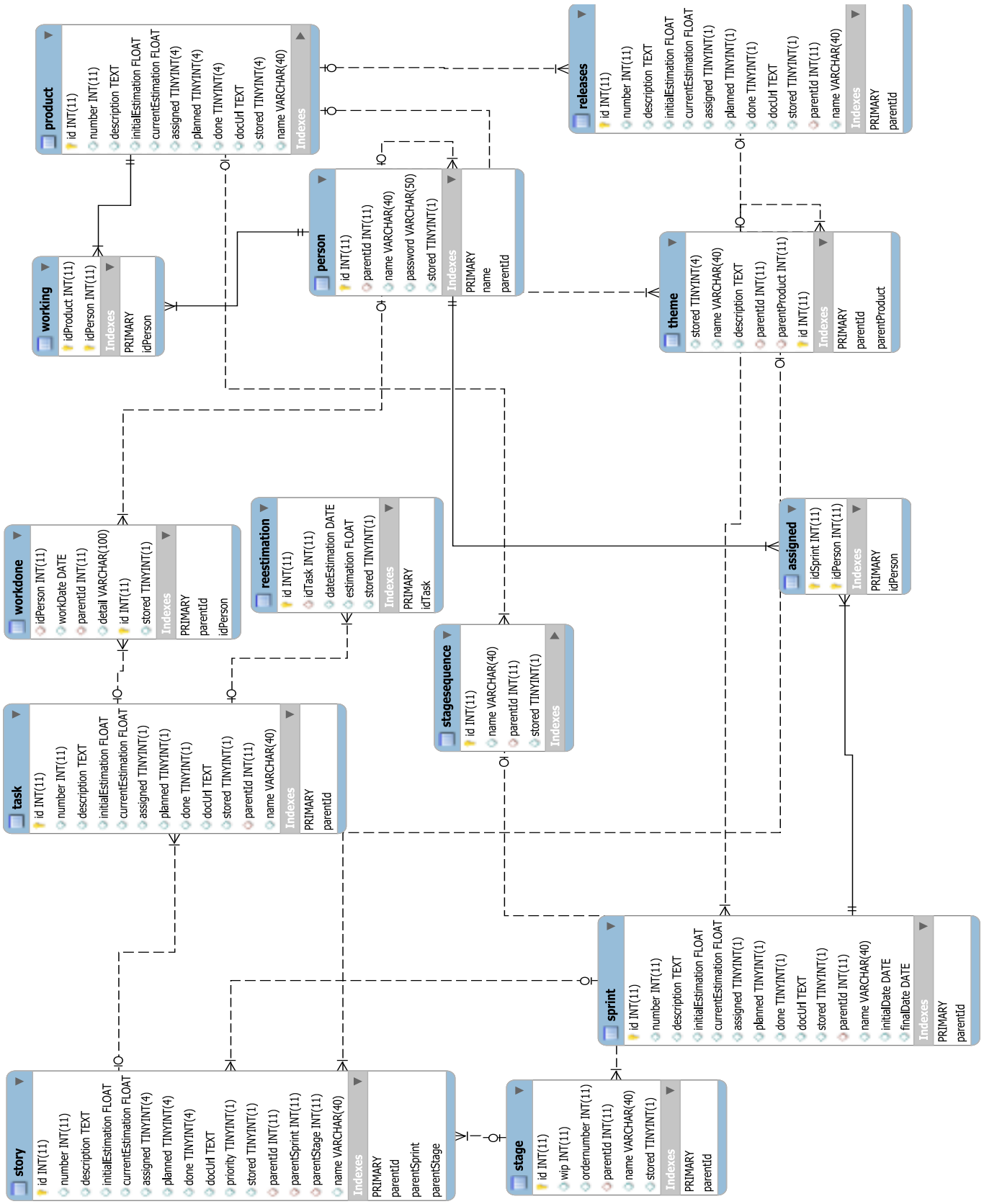

Figura 4.1: Modelo EER.

fichas, en las que indicará brevemente la fecha y una descripción del trabajo realizado. Para llevar a cabo esta gestión será necesario la utilización de las clases Task.java y WorkDone.java.

- Clases para gestionar la pizarra de Scrum: para la gestión de la pizarra de Scrum, sera necesario la utilización de las clases que controlan y gestionan elementos implicados en cada uno de los sprints por tanto, se utilizarán las clases Sprint.java, Story.java y Task.java

4.2.2. Servicios del Servidor

El servidor será el encargado de gestionar la API descrita con anterioridad. El servidor es un servlet creado en Java, que con peticiones a través del cliente, que conviene recordar que utiliza tecnología GWT, y mediante un interface REST es capaz de comunicarse con el cliente recibiendo peticiones y devolviendo la información en formato JSON.

4.2.2.1. Interface REST

Si bien el término REST[14] se refería originalmente a un conjunto de principios de arquitectura, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML (JSON en nuestro caso) y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP.

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones

se equiparan a las operaciones CRUD que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.

- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional. Una aplicación web REST requiere un enfoque de diseño diferente a una aplicación basada en RPC. En RPC, se pone el énfasis en la diversidad de operaciones del protocolo, o verbos; por ejemplo una aplicación RPC podría definir operaciones como:

  - getUser(), addUser(), removeUser(), updateUser(), getLocation(), addLocation(), removeLocation().

En REST, al contrario, el énfasis se pone en la diversidad de recursos, o los nombres; por ejemplo, una aplicación REST podría definir los siguientes tipos de recursos:

- Usuario {}

- Localización {}

Cada recurso tendría su propio identificador. Los clientes trabajarían con estos recursos a través de las operaciones estándar de HTTP, como GET para descargar una copia del recurso. Obsérvese cómo cada objeto tiene su propia URL y puede ser fácilmente cacheado, copiado y guardado como marcador. POST se utiliza por lo general para acciones con efectos laterales, como enviar una orden de compra o añadir ciertos datos a una colección.

La comunicación entre la parte cliente y la parte servidora la hemos resuelto mandado la información en formato JSON[11], acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

A continuación se muestra en la Figura 4.2 un ejemplo simple de definición de barra de menús usando JSON y XML.

JSON:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Figura 4.2: Ejemplo JSON frente a XML

Si bien los defensores de JSON a menudo recalcan que éste es más abreviado que XML, obsérvese que los dos ejemplos tienen unos 190 caracteres cuando se eliminan los espacios en blanco. Además, el uso de compresión GZIP para enviar los datos al navegador puede reducir la diferencia de tamaños entre ambos formatos. De hecho, cuando se usa GZIP sobre los ejemplos anteriores el ejemplo en XML es más pequeño por 6 bytes. Si bien esto no es concluyente, muestra que es necesario experimentar con el conjunto de datos a tratar para determinar qué formato será más eficiente en términos de tamaño. JSON no es siempre más pequeño que XML. El beneficio de JSON, entonces, no es que sea más pequeño a la hora de transmitir, sino que representa mejor la estructura de los datos y requiere menos codificación y procesamiento.

Teniendo todo esto en cuenta, a la hora de crear nuestro servidor, hemos definido dos métodos que pasamos a describir a continuación:

- doGet():Recogerá las peticiones por parte del cliente que no necesitan actuación en la base de datos.

- doPost(): Recogerá las peticiones del cliente que requieran actuar sobre la base de datos.

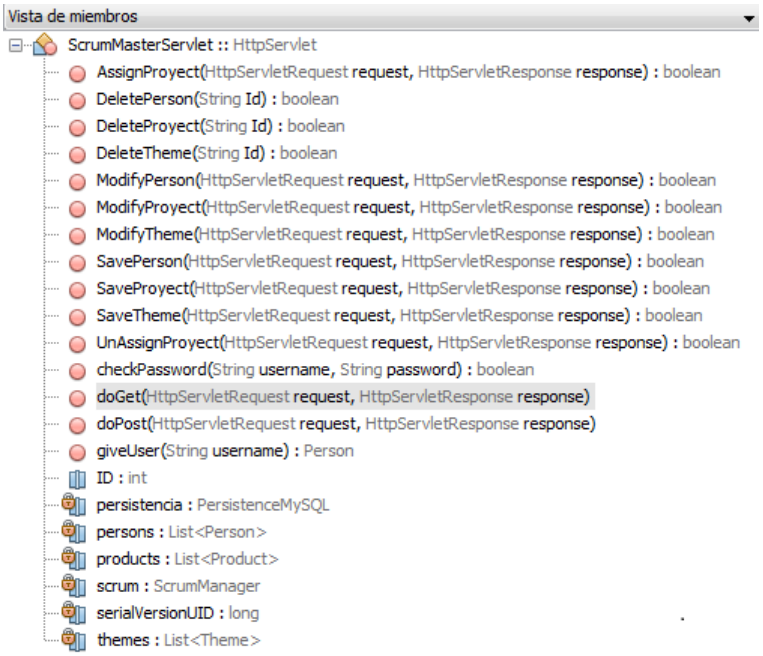

Figura 4.3: Métodos del Servidor

En la Figura 4.3 podemos ver los dos métodos mencionados anteriormente, que en función del tipo de llamada harán uso de los diversos métodos definidos dentro del propio servlet.

Respecto al cliente, hemos comentado con anterioridad que utiliza tecnología GWT. Éste a su vez consta de cuatro componentes principales: Un compilador Java-a-JavaScript, un navegador web "hosted", y dos librerías de clases tal como se ve en Figura 4.4.

Los componentes son[3]:

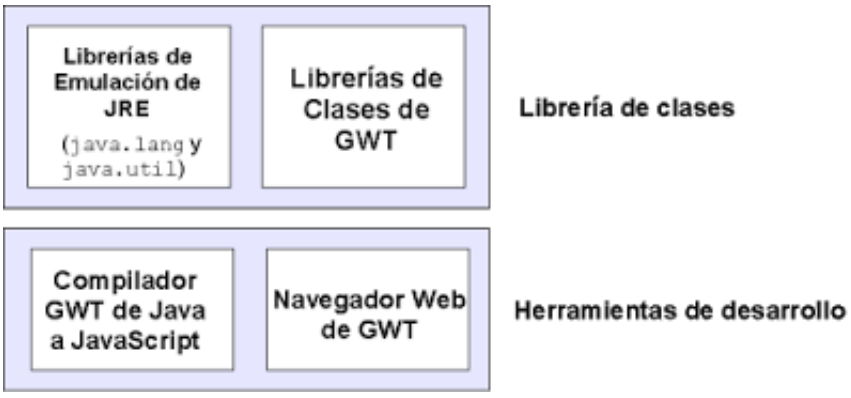

Figura 4.4: Arquitectura GWT.

- Compilador GWT Java-a-JavaScript: traduce del lenguaje de programación Java a JavaScript. El compilador se utiliza para correr la aplicación en modo web.

- Navegador web "Hosted" de Gwt: te permite correr y ejecutar GWT aplicaciones en modo hosted, donde lo que estás corriendo son bytecodes de Java sobre una máquina virtual sin compilarlos a JavaScript. Para lograr esto, el navegador GWT incrusta un controlador de browser especial (un control del Internet Explorer sobre Windows o un control de Gecko/Mozilla sobre Linux) con hooks dentro de la máquina virtual de Java.

- Emulación de librerías JRE: GWT contiene implementaciones en JavaScript de las librerías de clases más usadas en Java, incluyendo la mayoría de las clases del paquete java.lang y un subconjunto de clases del paquete java.util. El resto del estándar de librerías de Java no es soportado nativamente con GWT. Por ejemplo, las clases de los paquetes como java.io no se utilizan en aplicaciones web ya que estas acceden a recursos en la red y al sistema de archivos local.

- Librería de clases de interfaz de usuario de GWT: Las librerías de clases de interfaz de usuario de GWT son un conjunto de interfaces y clases personalizadas que te permiten crear "widgets" para el navegador, como botones, cajas de texto, imágenes, y texto. Éste es el núcleo de las librerías de interfaz de usuario para crear aplicaciones GWT.

Por tanto, el cliente se comunicara con el servidor y éste a su vez con la API que podrá manejar internamente.

4.3. Interfaz

La interfaz de usuario incluye tanto los aspectos relacionados con la presentación como las funciones JavaScript, que acceden y actualizan los datos. Para dar soporte a la aplicación, es necesario una organización de clases de la siguiente manera:

- Clases para administrar usuarios: a través de la interfaz se podrán crear, modificar y borrar los usuarios del sistema. El paquete admin dará soporte a este punto.

- Clases para gestionar el desarrollo de los proyectos: Por un lado, se permitirá acceder a la lista de productos en desarrollo, así como los contenidos relacionados con los mismos. Por otro lado, se podrá crear nuevos proyectos, modificar los

Figura 4.5: Dependencia gráfica ApiScrumManager para Maven.

existentes o borrar antiguos. El paquete product contendrá todas las clases relacionadas con ello.

- Clases para controla el acceso a la aplicación: los usuarios podrán acceder a la aplicación presentando sus credenciales (nombre y contraseña). Además, una vez autenticado, deberá ser posible acceder a la administración de usuarios o de proyectos asignados. El paquete encargado de esto sera init.

- Clases para gestionar los temas: deberá poder administrarse cómodamente los temas, así como sus subtemas. En este caso el paquete implicado sera themes.

4.4. Maven

Ya hemos comentado anteriormente la implicación de Maven en el proyecto, ahora vamos a intentar ilustrar de una manera más gráfica la implicación que tiene en el proyecto esta herramienta. Como bien hemos comentado, la aplicación se divide en tres proyectos; así por un lado tenemos la API, por otro el servidor y finalmente el cliente. Cada proyecto necesita de unas librerías para su correcta ejecución, con esto tenemos que la API requiere de la librería de conexión a la base de datos MySQL como se ve en Figura 4.5. El proyecto ScrumMasterServer necesita a su vez de la API, así que dependerá indirectamente de la librería de conexión de MySQL. Además necesitará de la librería JSON y de la librería propia de java dado que estamos hablando de que el servidor es un Servlet, en la Figura 4.6 podemos observar cómo se relaciona. Por último tenemos el cliente, llamado ScrumMaster, que es un proyecto GWT con lo cual necesita de las librerías gwt-servlet y gwt-user. Además por el tipo de interfaz necesita de la librería smargwt y de la propia de java. Esto se representa en la Figura 4.7.

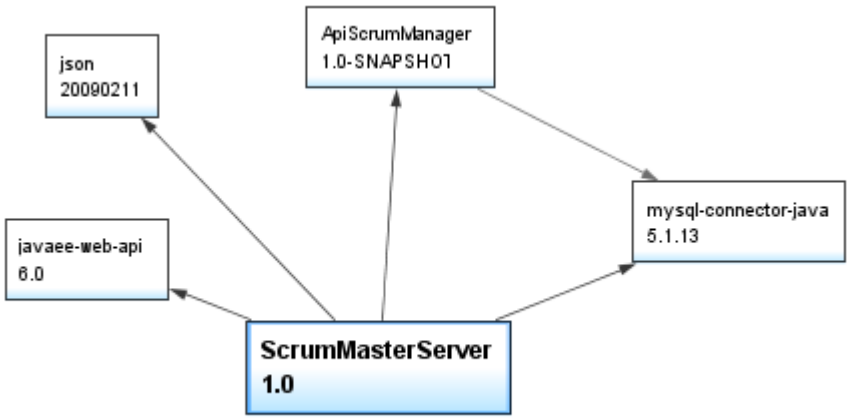

Figura 4.6: Dependencia gráfica de ScrumMasterServer para Maven.

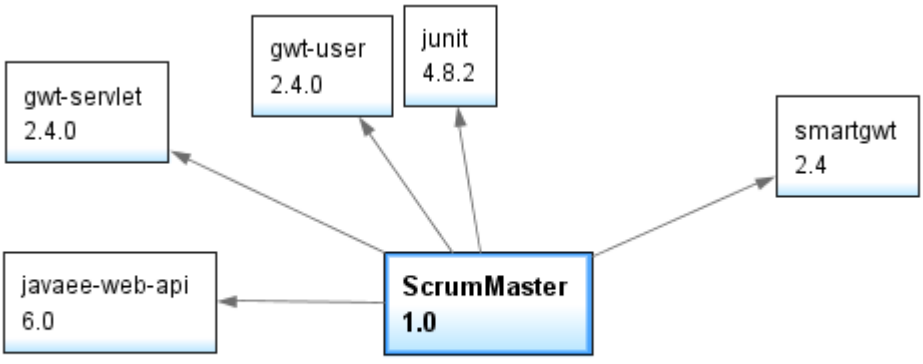

Figura 4.7: Dependencia gráfica de ScrumMaster para Maven.

Capítulo 5

Diseño

Si se disminuye el nivel de abstracción, y el enfoque se centra en más detalles, se obtendrá el diseño detallado de los componentes determinados en los puntos anteriores. De esta forma, conviene centrarse en el diseño de la aplicación lógica y de su interfaz. En este capítulo se expone el diseño necesario para entender la aplicación heredada y el diseño realizado en el servidor que dota al proyecto de los requisitos no funcionales deseados.

5.1. Diseño de la API

La API es una de las partes fundamentales de un software, un proyecto o incluso una empresa; es decir, si conseguimos el objetivo de tener una buena API, nuestro producto será mas fácilmente vendible, comercial y mejorando además la calidad del código. Las características o requisitos de una buena API son los siguientes según [Bloch, 2008[2]]:

- Fácil de aprender.
- Fácil de usar, incluso sin documentación.
- "Debe ser difícil usarla mal".
- El código que use la API, debe ser fácil de leer y mantener.
- Ser capaz de resolver los requisitos necesarios.
- Fácil de extender.

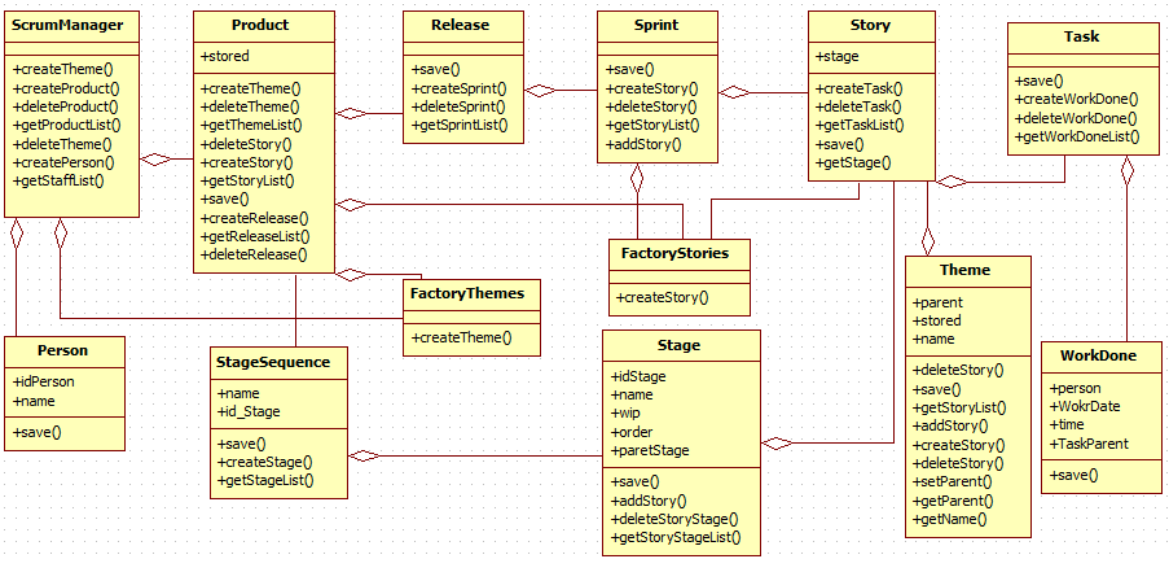

Figura 5.1: Diagrama de clases de las clases principales de la API

5.1.1. Divisiones de clases

Siguiendo las divisiones y elementos ofrecidos por la propia metodología Scrum, se han creado un conjunto de clases que trabajan acorde a la metodología permitiendo controlar todos y cada unos de los distintos elementos de ésta. Para implementar todas las funcionalidades ha sido necesaria la creación de algunas clases extras para albergar todos los métodos necesarios. En la Figura 5.1 podemos observar un diagrama de clases del conjunto que conforma la API de ScrumManager.

A continuación se detallan algunas clases:

- *ScrumManager*: clase principal debido a las funcionalidades de trabajo que permite. Desde aquí podemos lanzar la creación de nuevos Productos, Temas o Pizarras. Es la encargada de guardar la relación de los distintos Productos disponibles y gestionados por la aplicación.

- *Theme*: clase responsable de clasificar Historias dependiendo del tema que ocupen. Permitirá por tanto la gestión de Historias, incluyendo su creación, borrado y añadido. Los temas serán dependientes de cada Producto y será necesaria su creación desde los nuevos Productos.A continuación podemos observar en la Figura 5.2 todas las clases encargadas de la gestión de los nuevos Temas y cómo la clase FactoryThemes es la que se encarga finalmente de crear los nuevos Temas.

- *StageSequence*: clase encargada de la creación de la Pizarra para la clasificación de las Historias según la metodología Kanban. Desde esta clase es posible manejar y

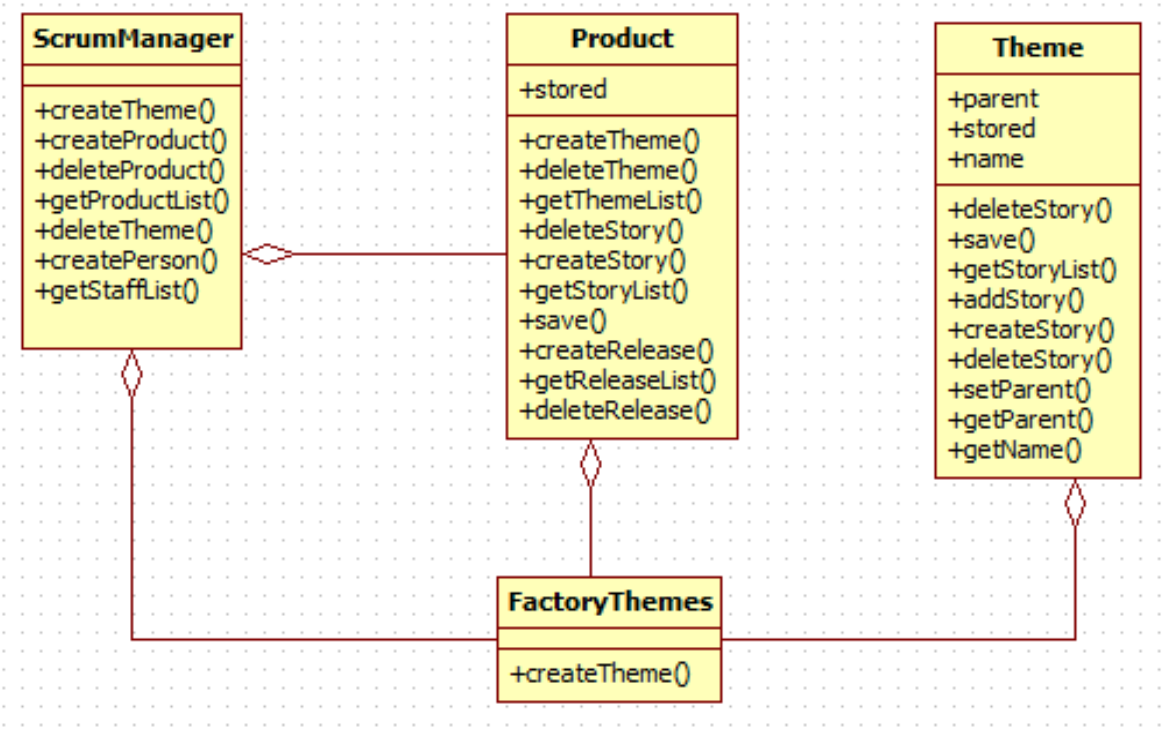

Figura 5.2: Clases responsables creación Temas.

gestionar (crear, borrar y modificar) las distintas Stages que el usuario considere necesarias para su proyecto.

- *Stage*: clase que alberga y clasifica un número variable de Historias siempre menor que el máximo preestablecido WIP. En caso de intentar agregar más Historias de las limitadas por el WIP, el proceso se interrumpe.

- *Item*: clase complementaria de gran relevancia en la implementación realizada. Es una clase que no aparece entre los elementos de Scrum, pero se ha considerado necesaria para la correcta implementación del resto de los elementos de la metodología Scrum. De esta clase heredan clases como Producto, Release, Sprint, Historia y Task como se puede observar en la Figura 5.3. Esta herencia permite compartir cómodamente los mismos atributos para las clases anteriormente mencionadas y sus respectivos métodos Set y Get. Provoca por tanto, una mayor facilidad y agilidad a la hora de la implementación, además de una mayor comodidad para el desarrollador ya que solamente debe recordar los atributos de una clase, y no los distintos atributos e identificadores de cada uno de los elementos. Todo esto complementado con la reutilización de los métodos Get y Set, aligerando aún más el proceso de implementación.

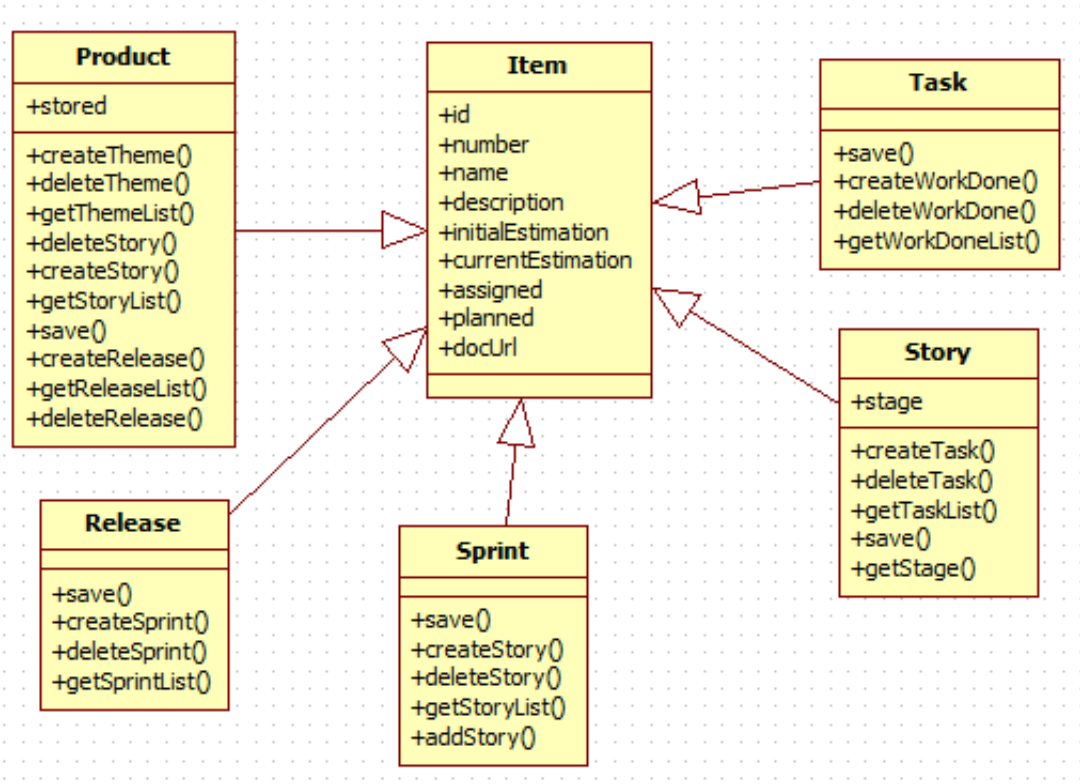

Figura 5.3: Herencia de la clase Item.

- *Product*: clase encargada de controlar el elemento principal de la herramienta, que define el proyecto a desarrollar. Desde aquí se puede controlar gran parte o la mayor parte de los elementos necesarios para Scrum y Kanban. Dispone de métodos para crear, modificar y eliminar elementos como las Releases, Historias, Tareas, Temas, Personas y la Pizarra.En la Figura 5.4 se puede observar el proceso de creación de un nuevo producto.

- *Release*: encargada de gestionar, crear y añadir nuevos Sprints que deberán ser realizados o trabajados durante un determinado periodo, determinado en la fase de diseño del Producto entre el ProductOwner, ScrumMaster y desarrolladores.

- *Sprint*: clase responsable de la gestión de las Historias pertenecientes a dicho Sprint. Por tanto será capaz de crear, añadir y eliminar Historias. Tendrá un periodo de duración máximo en el que se deben finalizar todas las Historias incluidas.

- *Story*: clase encargada de controlar, reflejar y almacenar el trabajo realizado en cada una de todas las Historias que componen los Productos. A su vez gestiona

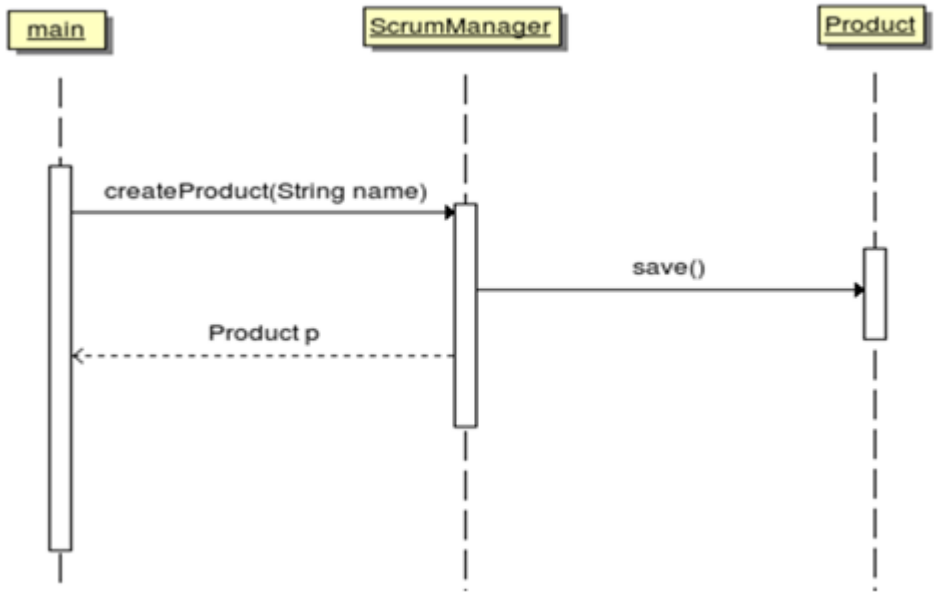

Figura 5.4: Creación de un nuevo producto.

todas las Tareas en las que se subdividen cada Historia, permitiendo su creación y borrado.

- *Task*: responsable de almacenar y controlar cada una de las subtareas en las que se dividen las Historias. Permite la creación y borrado de las etiquetas WorkDone en las que se controla el avance.

- *WorkDone*: clase encargada de reflejar y almacenar el avance producido en cada una de las Historias / Tareas en las que se divide el Producto. Reflejará valores como el usuario, fecha, el tiempo de trabajo y la descripción.

- *FactoryStories*y *FactoryThemes*: clases complementarias a los elementos de las metodologías Scrum y Kanban. Son las encargadas de facilitar el trabajo y la creación de nuevas Historias o Temas, siendo muy utilizadas a lo largo del uso de la API en distintos métodos.

- *Person*: clase encargada de la creación de las personas encargadas o colaboradoras a cada uno de los Productos. Cada vez que un nuevo Producto sea diseñado será necesario dar de alta todos los usuarios/personas que vayan a participar en su desarrollo. Éstos podrán modificar y ampliar el trabajo realizado sobre cada una de las Historias o Tareas según el Historial de trabajo creado al ir añadiendo

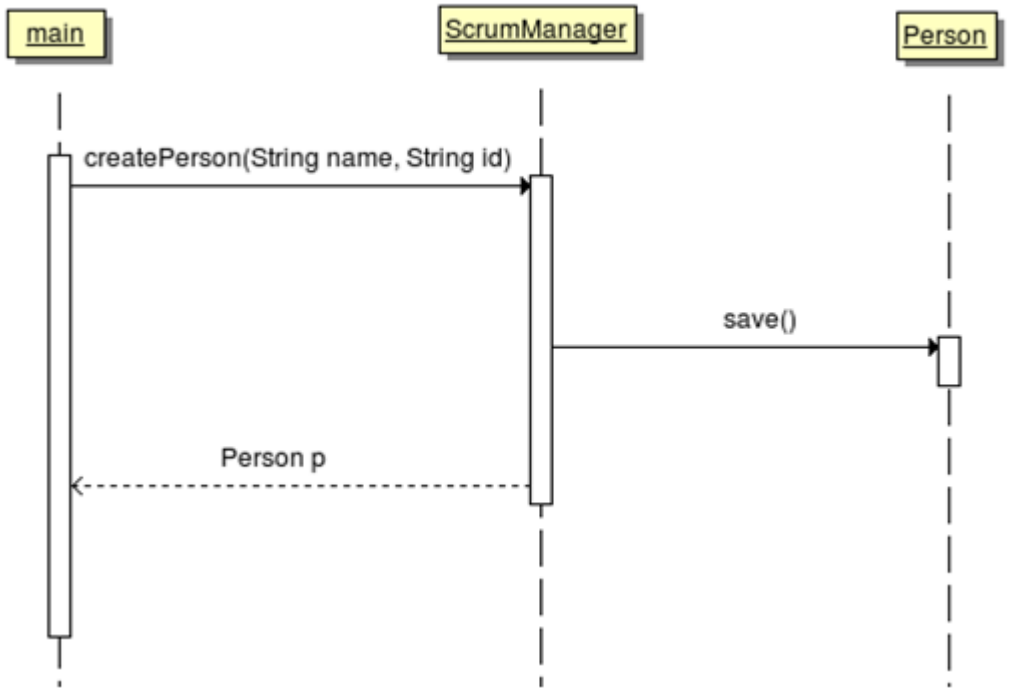

Figura 5.5: Creación de un nuevo usuario.

registros de tipo WorkDone. En la Figura 5.5 se puede observar el proceso de creación de un nuevo usuario.

5.1.2. Persistencia

El sistema de persistencia de la API de ScrumManager está basado principalmente en el uso de la interfaz Persistence. La interfaz Persistence define todos los métodos necesarios para guardar todos los cambios y modificaciones que se realicen en los proyectos gestionados mediante la API. Debe observarse que Persistence sólo define una interfaz que luego podrá implementarse polimórficamente sobre diferentes Bases de Datos, debido a que en la clase Persistence no se hace ninguna suposición relativa a ningún dialecto de SQL en particular. En este PFC se ha implementado un derivado de Persistence, llamado PersistenceMySQL, el cual da soporte para MySQL. La extensión a otros sistemas de gestión de Bases de Datos es trivial.

Como se puede observar en la Figura 5.6, la clase PersistenceMySQL hereda de la interfaz Persistence, implementado todos sus métodos. Esta implementación lleva consigo la creación de todas las querys que se necesitan para cada uno de los métodos, así como la configuración de los datos de conexión a la Base de Datos de tipo MySQL.

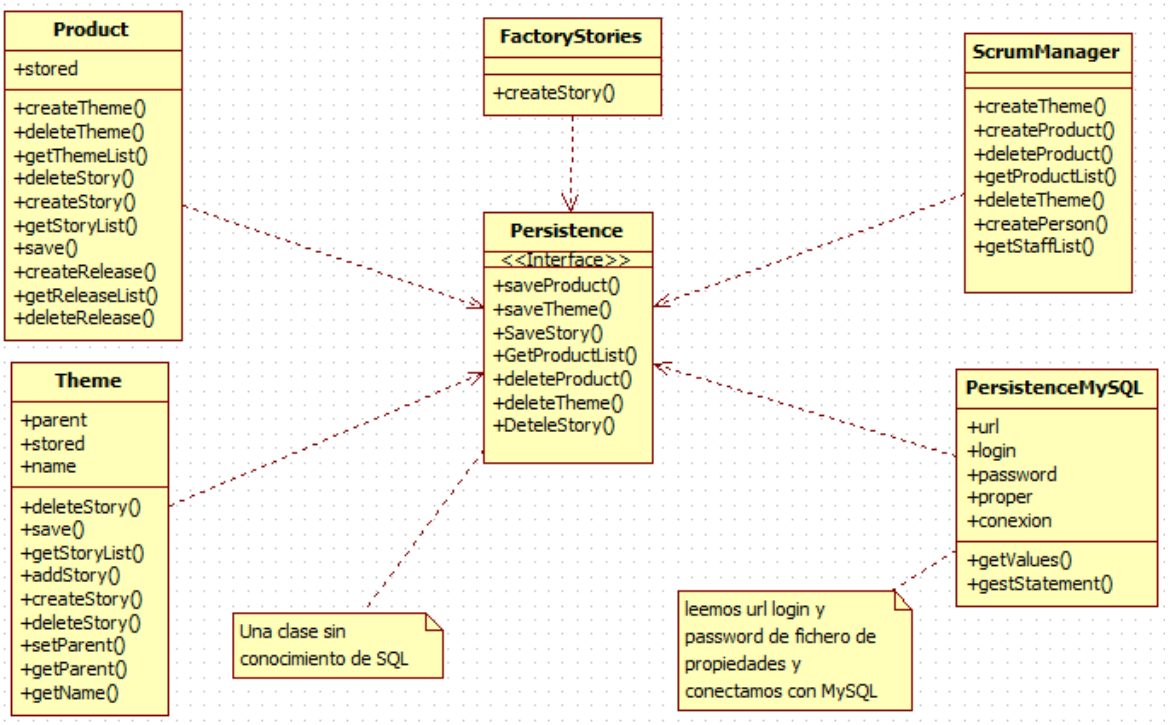

Figura 5.6: Clases que utilizan interfaz Persistence.

Como también podemos observar en la Figura 15, el sistema sería fácilmente ampliable para permitir la persistencia de los datos en sistemas de Bases de Datos distintos como pueden ser Oracle o cualquier otro tipo, simplemente con la implementación de los métodos que se heredan de la interfaz Persistence. La característica principal de este diseño quizás sea su sencillez para lograr la persistencia de los datos mediante el uso de la interfaz Persistence. Ninguna de las clases tienen métodos de Persistencia propios, es decir, siempre que dichas clases guarden o almacenen sus datos, lo harán mediante los métodos existentes en dicha interfaz, la cual es la única responsable de la persistencia de los datos en el conjunto de la API de ScrumManager. Esta característica aporta facilidad a la hora de permitir a los usuarios o desarrolladores intercambiar los sistemas de Gestión de Bases de Datos utilizada en cualquier momento, simplemente modificando una línea de la clase principal de la API ScrumManager.java.

Si se profundiza en la clase PersistenceMySQL se observa cómo se encarga de crear y mantener todas las conexiones necesarias con la Base de Datos de tipo MySQL. La forma en que está implementada obliga a que cada vez que se requiera trabajar con alguno de sus métodos, no sea necesario crear una nueva conexión a la Base de Datos, sino que almacena dicha conexión, acelerando y agilizando el proceso.

Para el correcto funcionamiento de la clase PersistenceMySQL, su correspondien-

te conexión a la Base de Datos, y por tanto, toda la persistencia del proyecto, es necesario que primero se modifiquen los valores del fichero de configuración "configuration.properties" anexo al proyecto, el cual almacena los valores necesarios de url, usuario y contraseña de acceso a la Base de Datos. En el Manual de Instalación anexo (A.1) se puede encontrar más información acerca de la configuración de la conexión a la Base de Datos.

5.2. Diseño de la Interfaz Cliente

La lógica del cliente deberá soportar la gestión de la aplicación de una manera gráfica. Para ello, en primer lugar, se ha definido la estructura general de la aplicación, y se han llegado a una serie de convenios que deberán soportar las clases que intervengan. Estos son:

- La aplicación funcionará bajo un sistema de ventanas. Es decir, cuando se quiera añadir información, se creará la ventana correspondiente, y se añadirá la forma de acceder en la ventana que correspondiera.
- Toda ventana de la aplicación heredará de ScrunManager, que ofrecerá los métodos y los atributos necesarios para mantener la coherencia de la aplicación. Concretamente destacan los siguientes:
  - Identificador y tipo del item seleccionado:idItemSelected, typeItemSelected.
  - Tipo de item sobre el que se trabaja: PRODUCTOS;THEMES;PERSONS..
  - Identificados de la acción que se debe realizar: ADD;MODIFY;SHOW.
  - La variable que permitirá independizar el código del idioma: constants.
  - Métodos para configurar las ventanas de distintas maneras.
- Los menús de opciones aparecerán como menús contextuales, o pop-ups. Estos serán creados en la misma clase en la que después aparecerán.

Explicados estos convenios, vamos a explicar la estructura que hemos seguido para cada tipo de clase de la API.

Así por ejemplo, vamos a analizar la estructura que se ha creado para los productos/proyectos, decidiendo la creación de una clase que implementará las acciones. Siguiendo la Figura 5.7 podemos observar que ProductLisPannel consta de otra clase,

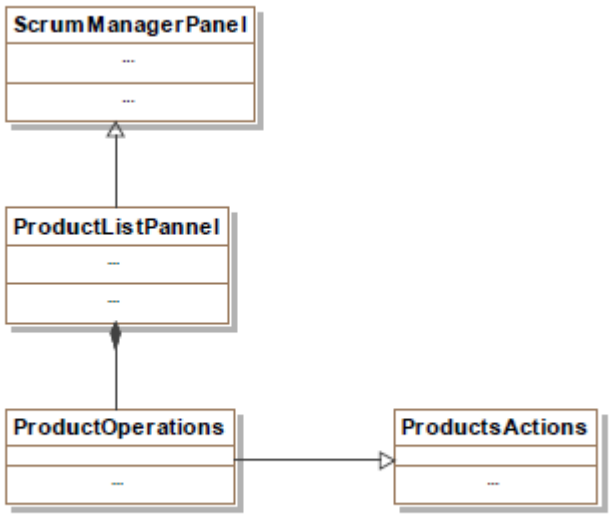

Figura 5.7: Diagrama de clases para productos.

ProductsOperations, que a su vez extiende de ProductsActions. Aunque no es una buena idea este tipo de arquitectura, por que tiene un gran acoplamiento, es la mejor forma de realizar cambios en la interfaz, como respuesta a las acciones que se hagan. El motivo es que GWT funciona con llamadas asíncronas, y por lo tanto, se desconoce cuando se va a realizar la acción, si es que se permite. Esto no tendría mayor problema si no fuera porque dependiendo de que se realice una determinada acción, la clase se comportará de una u otra forma. Por poner un ejemplo, si se desea añadir un proyecto interesa que cuando se haga la inserción se haga en el lado del servidor y después el cliente actualice el listado de productos de su interfaz. Esto lleva a que la clase ProductListPannel, necesite de otra clase con la que comunicarse con el servidor y que además permita modificar su propio listado. Así, surge ProductsOperations, que extenderá de ProductsActions y que modificará el comportamiento de ésta al recibir un producto.

5.2.1. Módulo General

Consta de ciertas clases que permiten iniciar la aplicación ofreciendo una infraestructura sobre la que se sostendrá la aplicación, las clases son las siguientes:

- ScrunManagerPanel: clase base para crear nuevas ventanas en la aplicación. Por tanto, las distintas ventanas de la aplicación extenderán de ella.

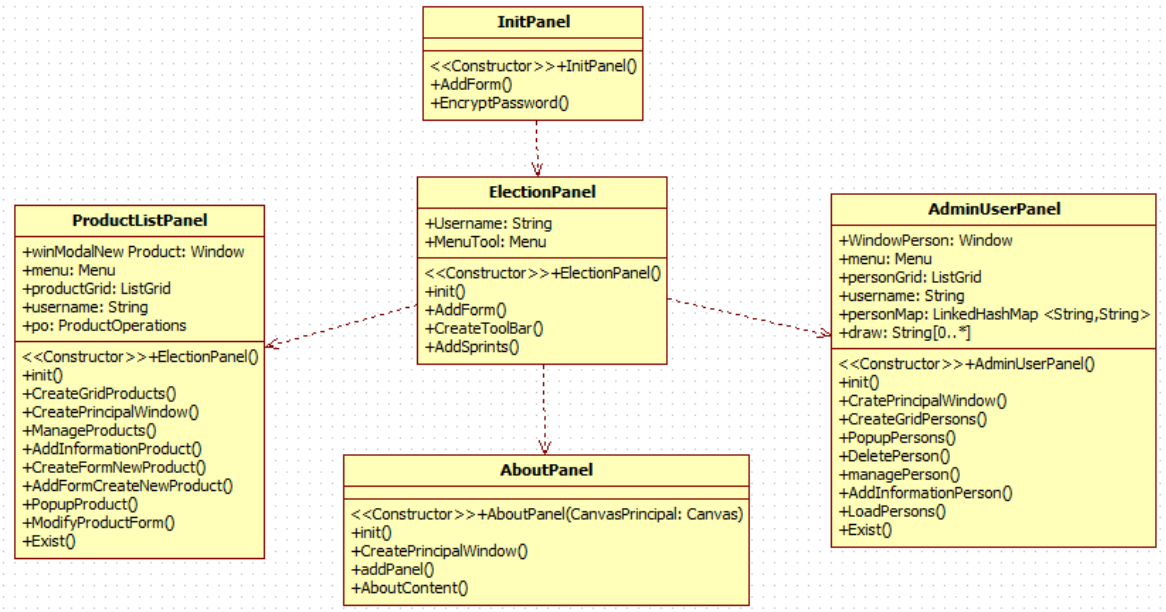

Figura 5.8: Diagrama de clases para módulo Init.

- MainEntryPoint: sera el punto de entrada de la aplicación. Cargará el panel inicial.
- ServerException: sirve para capturar excepciones del servidor.

5.2.2. Módulo Init

Incluye todas las ventanas iniciales al entrar en la aplicación (ver Figura 5.8). Lo configuran, principalmente, las siguientes clases:

- ElectionPanel: ventana que se mostrará tras autenticarse en la aplicación y que permitirá acceder al módulo de usuarios, o de proyectos. Cabe citarse, por otro lado, que será esta clase la que genere el entorno sobre la que se va a trabajar a partir de ese momento. Sobre este lienzo, se añadirán ventanas nuevas y sera compartido por todas las clases, que la recibirán en el constructor.
- InitPannel: ventana que ofrece un formulario que el usuario que desee utilizar la aplicación deberá rellenar con sus credenciales.

5.2.3. Módulo Admin

Esta formado por las clases relacionadas con las personas que intervengan en la aplicación. Permitirá por tanto administrar a éstos usuarios (ver Figura 5.9). Destacan:

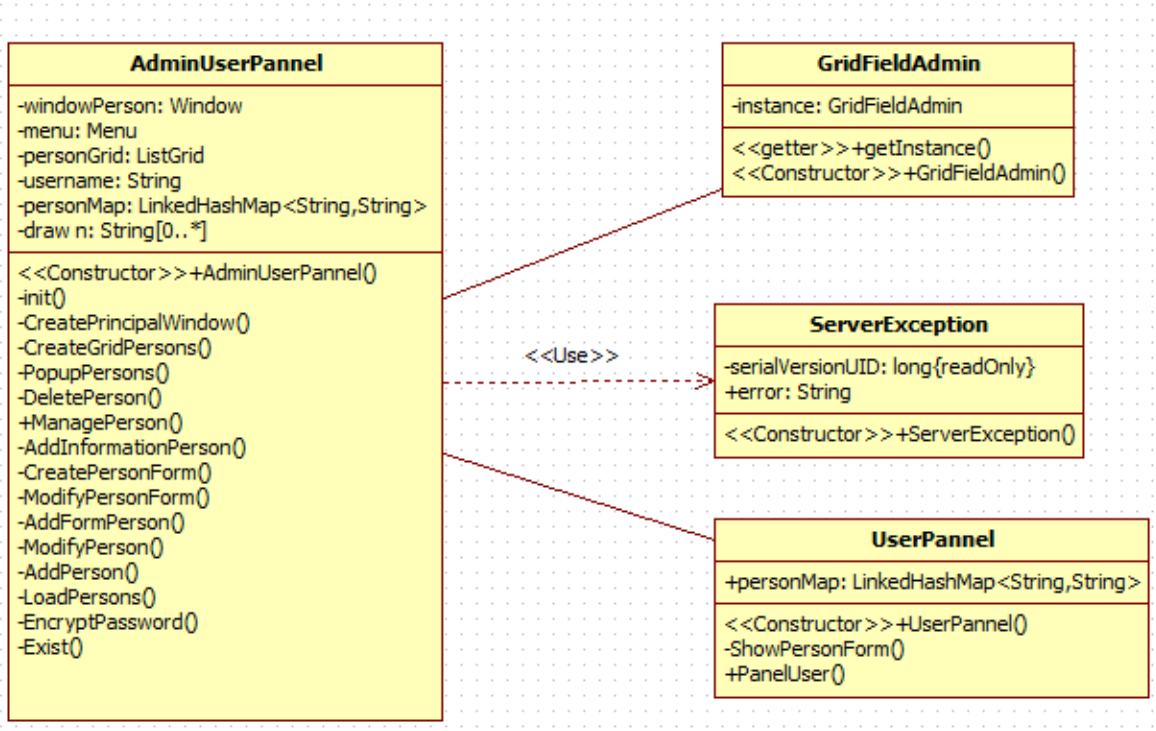

Figura 5.9: Diagrama de clases para módulo Admin.

- AdminUserPannel: mostrará la lista de los usuarios en un grid. A través de un menú contextual o pop-up se podrá gestionar cada usuario, es decir, se podrá acceder a su ficha, se podrá modificar o incluso borrar. Ademas se permite la creación de nuevos usuarios.
- GridFieldAdmin: extiende de DataSource y permite definir la estructura del grid de AdminUserPannel.
- UserPannel: creará una ventana con la información de un usuario concreto.

5.2.4. Módulo Product

Consta de todas las clases relacionadas con los productos. Entre otros:

- GridFieldPrincipal: extiende de DataSource, y permite definir la estructura del grid de ProductLisPannel.
- ProductListPannel: clase que mostrará los productos en un grid. Por cada producto se podrá acceder a información del mismo. Su ficha, los temas de los que consta, o sus entregas. Además podrá modificar o borrar información de un producto concreto, así como crear nuevos.

- ProductPannel: clase que crea una ventana con la información relativa a un producto concreto. Dentro de esta ventana, además de los datos del producto, se podrán asignar personas a ese producto concreto.
- ProductsActions: clase que comunica con el servidor para realizar las acciones relacionadas con los productos. Además, ofrece la estructura de un formulario de un producto
- ProductsOperations: que sera la clase implementada dentro de ProductListPannel y que redefinirá las acciones de ProductsActions.
- TreeNodePerson: como se ha mencionado en ProductPannel, desde ella se puede asignar personas al producto. La estructura en la que muestran las personas libres y las asignadas es un árbol. TreeNodePerson será por lo tanto, una clase que extiende de TreeNode y que tiene la estructura de personas deseada.

Asimismo, para ver el proceso de creación de un nuevo producto, se puede estudiar la Figura 5.10, donde destacamos que a la hora de la creación una vez recogidos los datos, hacemos una llamada al servidor mediante una petición URL al servidor y éste nos devolverá el ID en formato JSON.

5.2.5. Módulo Theme

Incluye toda la gestión de temas. Destacan las siguientes clases:

- ThemePanel: clase que muestra el contenido de un tema en una ventana.
- ThemesActions: gestiona todas las operaciones relativas a los temas.
- ThemesListPannel: lista de temas en forma de árbol a través de una ventana.
- TreeNodeThemes: representa la estructura del árbol de temas.

5.3. Diseño del servidor

Encargado de relacionar las acciones del cliente con la API propiamente dicha. Para ello, implementará una serie de métodos que podrán ser llamados desde el cliente llamando a una url. Existen dos formas de llamar a la url, a través del método GET y a través del POST. Las llamadas al método GET serán para consultar datos, sin

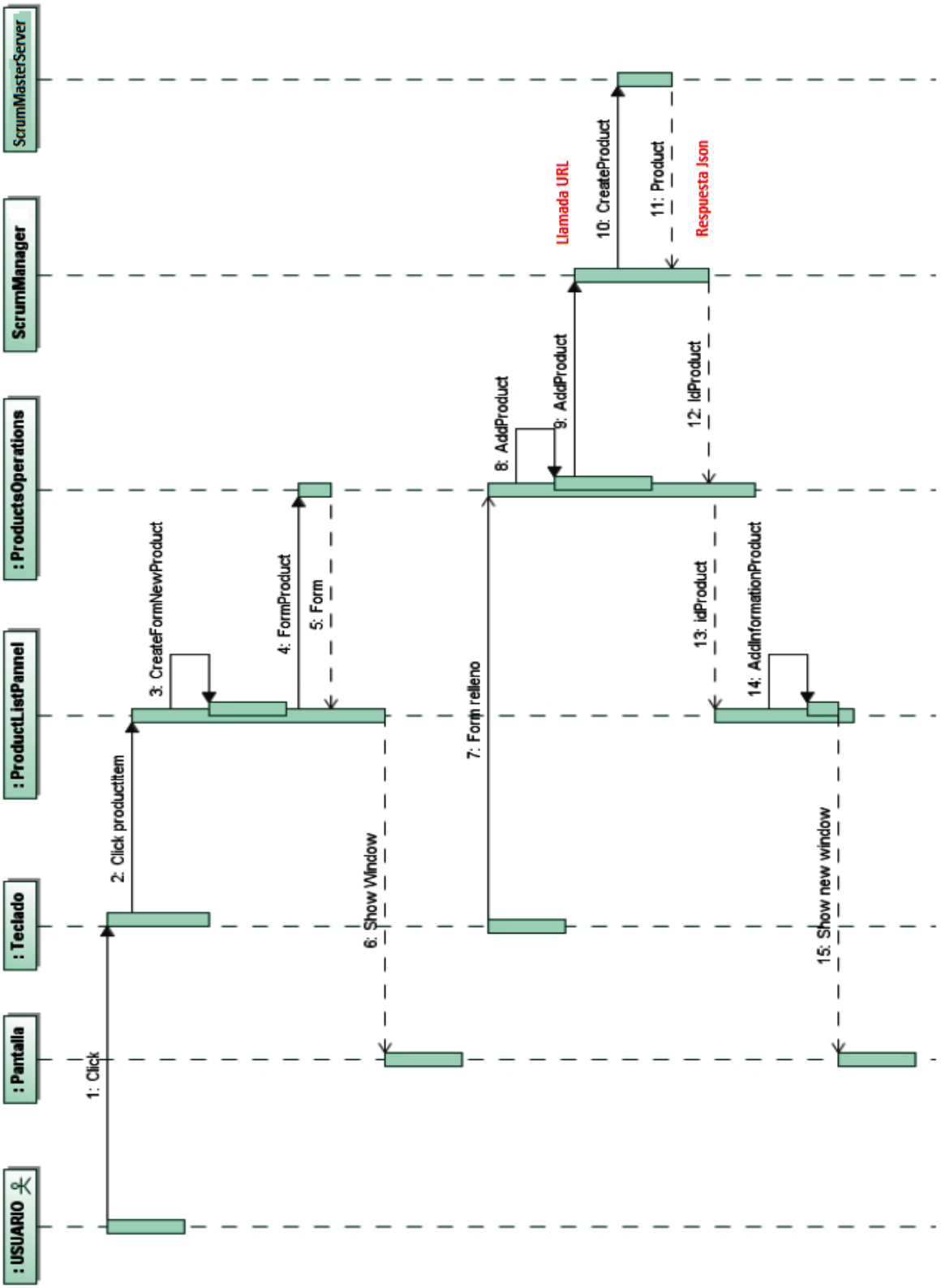

Figura 5.10: Secuencia de creación de un producto.

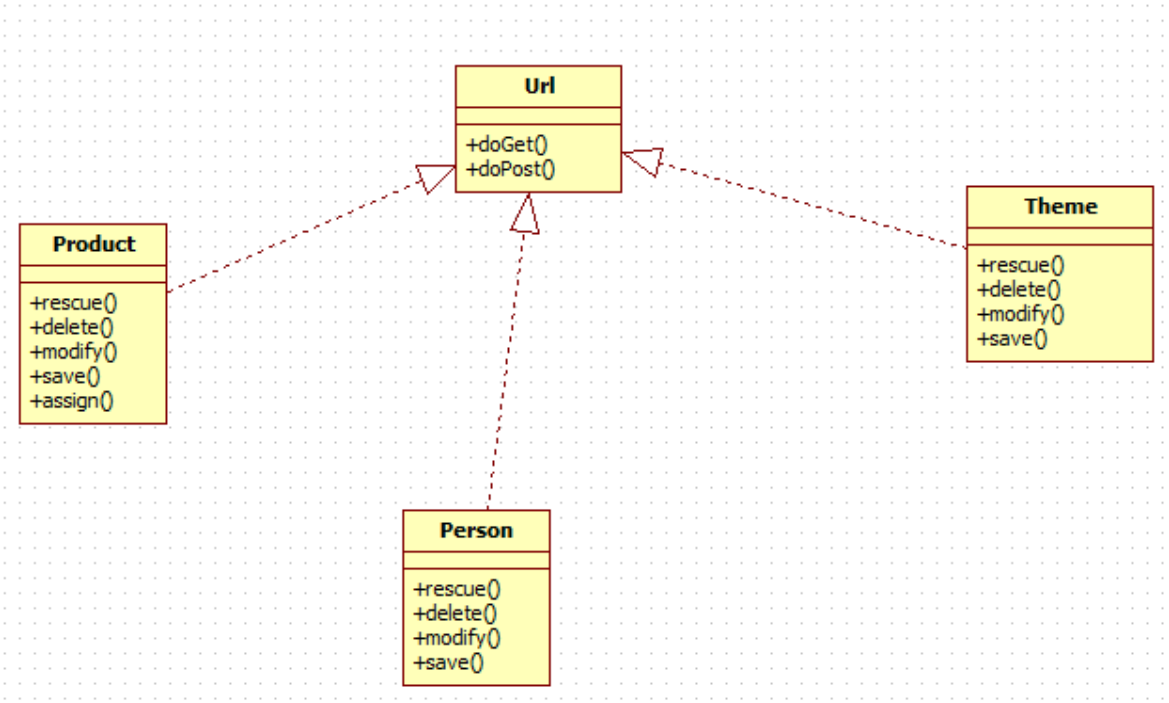

Figura 5.11: Diagrama del Servidor.

intervención en la base de datos mientras que con las llamadas al POST podremos llevar a cabo modificaciones en la base de datos.

En la Figura 5.11 se puede observar los diversos tipos de implementaciones que se han realizado para cada clase. A continuación explicamos la manera de implementar la llamada a la url:

Los parámetros que tiene que recibir la url, separados por el delimitador "&", de manera genérica son:

- user: nombre del usuario con el que se ha accedido a la aplicación.
- password: Contraseña, cifrada en MD-5.
- A continuación se añadirá el "item" sobre el que queremos actuar (product, person, theme, etc..) seguido de:
  - rescue: cuando la acción requerida es una consulta.
  - delete: cuando la acción requerida es un borrado.
  - modify: cuando la acción requerida es una modificación.
  - save: cuando la acción requerida es una inserción.

- assign: cuando la acción requerida implica una asignación.
- unassing: cuando la acción requerida implica una desasignación.

A continuación vamos a detallar para cada elemento sus acciones posibles:

- Product:
  - Para hacer referencia a la consulta de un producto/proyecto, pasándole la palabra "rescue" seguido del id. Por ejemplo: `/ScrumMasterServlet?user=username&password=pwd&product=rescue&id=id`
  - Para hacer referencia al borrado de un producto/proyecto, pasándole la palabra "delete" seguido del id.
  - Para hacer referencia la modificación de un producto/proyecto, pasándole la palabra "modify" seguido del id. Para modificar, pasaremos además las variables requeridas de actualización.
  - Para hacer referencia la creación de un producto/proyecto, pasándole la palabra "save". Para añadir, pasaremos además las variables name, active y pwd con sus datos respectivos. En este caso nos devolverá el id del producto/proyecto creado, en caso de que el resultado haya sido correcto.
  - Para asignar un proyecto, pasándole la palabra "assign". Podemos asignar un proyecto a una persona,pasándole el id de la persona y el id del producto/proyecto. De igual manera podremos desasignarlo usando la palabra "unassign".
- Person:
  - Para hacer referencia a la consulta de un usuario, pasándole la palabra "rescue" seguido del id.
  - Para hacer referencia al borrado de un usuario, pasándole la palabra "delete" seguido del id. Por ejemplo: `/ScrumMasterServlet?user=username&password=pwd&person=rescue&id=id`
  - Para hacer referencia la modificación de un usuario, pasándole la palabra "modify" seguido del id. Para modificar, pasaremos además las variables name y pwd con sus datos respectivos.

- Para hacer referencia la creación de un usuario, pasándole la palabra “save”. Para añadir, pasaremos además las variables name, active y pwd con sus datos respectivos. En este caso, nos devolverá el id del usuario creado, en caso de que se haya podido crear.

- Theme:

  - Para hacer referencia a la consulta de un tema, pasándole la palabra “rescue” seguido del id. En caso de existir mas de un tema a un producto, nos devolverá un array de temas.
  - Para hacer referencia al borrado de un tema, pasándole la palabra “delete” seguido del id. Por ejemplo: `/ScrumMasterServlet?user=username&password=pwd&theme=delete&id=id`
  - Para hacer referencia la modificación de un tema, pasándole la palabra “modify” seguido del id. Para modificar, pasaremos además las variables name y pwd con sus datos respectivos.
  - Para hacer referencia la creación de un tema, paséndole la palabra “save”. Para añadir, pasaremos además las variables name, active y pwd con sus datos respectivos.

Cada vez que se realiza una acción, la respuesta del servidor será en formato JSON y siempre devolverá una variable boolean llamada *paso* que indicará si la llamada ha sido correcta. Además si hemos solicitado unos datos, por ejemplo de un producto/proyecto, nos devolverá los datos en formato JSON. Podemos ver un ejemplo en la Figura 5.12.

El servidor en función de la petición, accederá a un método que hará uso de la API, como se ve en la Figura 5.13 que muestra el código del método en el servidor para modificar los datos de un usuario.

5.4. Diagrama de Despliegue

Lógicamente, al tratarse de una aplicación web, surgen dos partes claramente diferenciadas: cliente y servidor.

- Cliente: estaría compuesta por el usuario y el navegador,ubicado en la estación de trabajo, que llamará mediante la url correspondiente a la aplicación, como se puede ver en la Figura 5.14. Este interpretará comandos JavaScript para construir

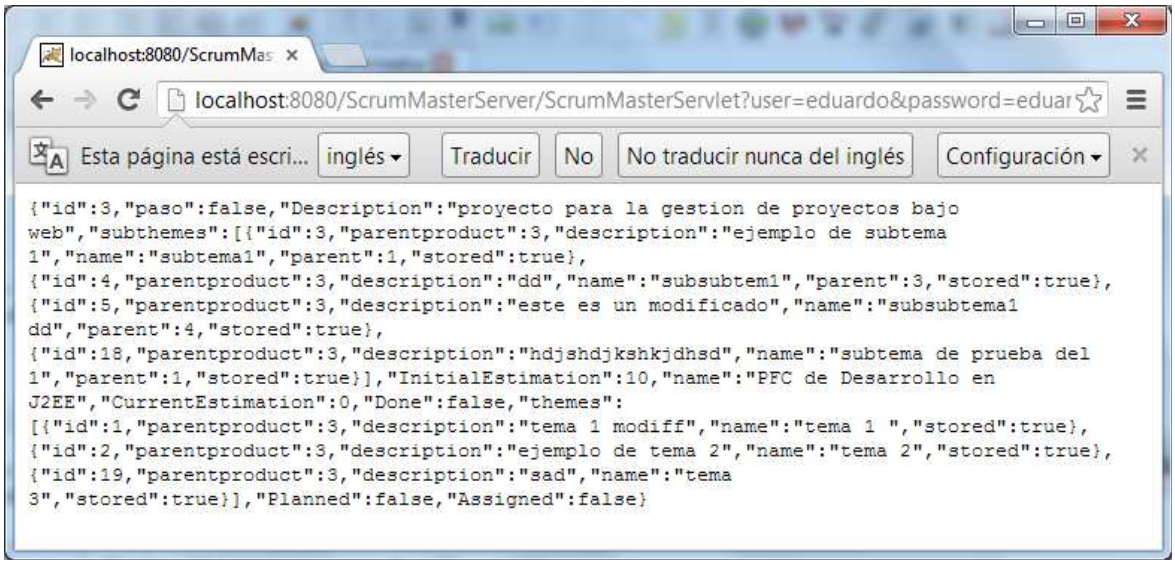

Figura 5.12: Ejemplo de respuesta JSON del servidor.

```
/*
 * modifica los datos de  una persona
 * in:id,name,password
 * out: boolean del resultado obtenido en la actualizacion
 */
public boolean ModifyPerson(HttpServletRequest request,HttpServletResponse response) {
    //modificamos la persona con los datos que recibimos
    String id=request.getParameter("id");
    String name=request.getParameter("name");
    String pwd = request.getParameter("pwd");
    Person p=scrum.getPerson(Integer.valueOf(id));
    p.setPassword(pwd);
    p.setName(name);
    try{
       persistencia.UpdatePerson(p);
       return true;
    }catch (Exception e) {
       return false;
    }
 }
```

Figura 5.13: Código del servidor para modificar un usuario.

el interfaz, mientras por detrás se estará recibiendo con peticiones url información del servidor en formato JSON.

- Servidor: cuyo núcleo será el servidor web Apache Tomcat instalado que contendrá primeramente el servlet ScrumMasterServer, y la API que será usada por el servlet en función de las acciones realizadas por el usuario desde el interfaz. Así, cuando el usuario quiere crear un nuevo proyecto desde el botón "Add new proyect" internamente estaremos haciendo una petición url a ScrumMasterServer pasándole los datos necesarios para la creación de un proyecto. A su vez, una ves recibida la petición desde el método POST, llamaremos a la función de la API encargada de crear un proyecto. Una vez realizada la acción, devolveremos desde el servidor al cliente la respuesta de la acción para que pueda reflejarlo en el interfaz.

  Existen dos métodos en la parte servidora, será a través del GET (solamente para leer datos) y el POST (para leer/escribir) en la base de datos. Estas llamadas son realizadas desde ScrumMaster y no serán visibles para el usuario.

  El servlet ScrumMasterServer hará uso de la ApiScrumManager que será la encargada de acceder a la base de datos MySQL. La API, como se observa, solo será accesible por el servidor, impidiendo que el cliente pueda realizar algún tipo de manipulación con ella. Así pues, al independizarse la API, lo hace también la base de datos del sistema. Ésta, solo será modificada por el sistema, usando para ello la API.

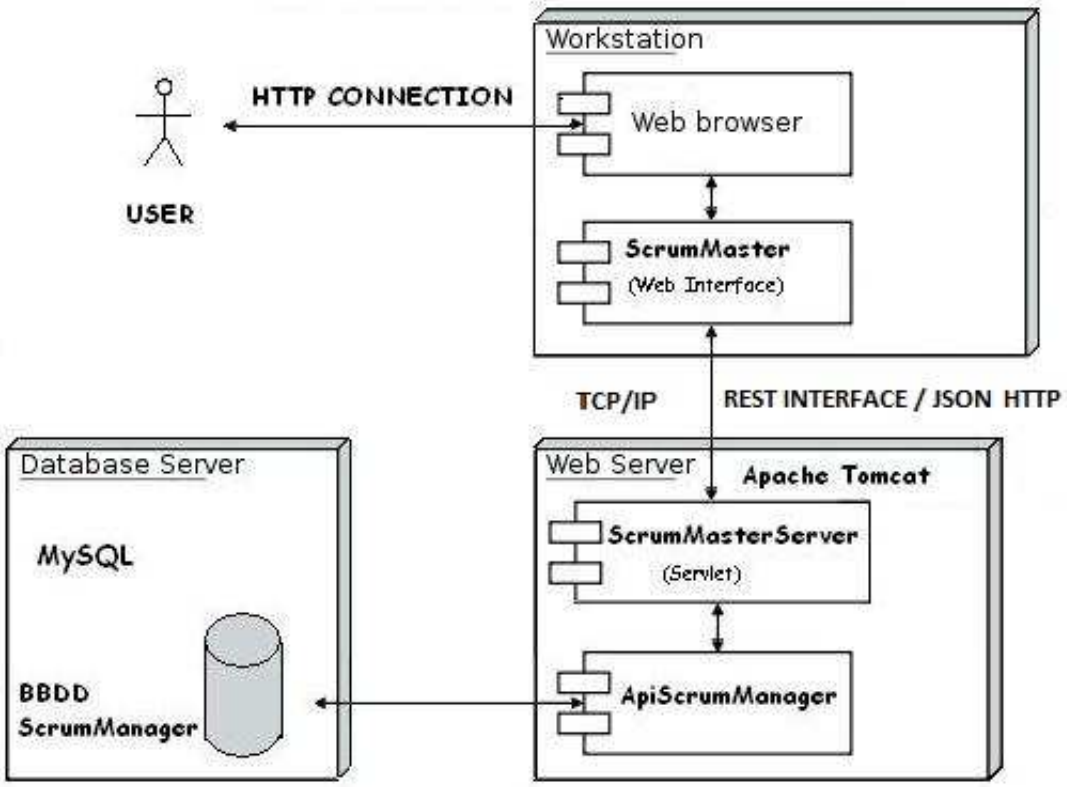

Figura 5.14: Diagrama de Despliegue.

Capítulo 6

Resultados Experimentales

En este capítulo se van a reflejar las informaciones obtenidas a través de Jenkins y de Sonar, una vez que los proyectos han sido mavenizados e introducidos en estos sistemas.

6.1. Resultados en Jenkins

Jenkins[10] es un software de Integración continua open source escrito en Java. Esta basado en el proyecto Hudson y es, dependiendo de la visión, un fork del proyecto o simplemente un cambio de nombre. Jenkins proporciona integración continua para el desarrollo de software. Es un sistema corriendo en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce y Clearcase y puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows. El desarrollador principal es Kohsuke Kawaguchi. Liberado bajo licencia MIT, Jenkins es software libre.

Desde la url `http://larabida.etsii.urjc.es:8000/`, con un usuario con suficientes privilegios, se podrá observar los tres proyectos (API, ScrumMasterServer y ScrumMasterClient) dados de alta que irán al repositorio de subversion a recoger la última versión y compilarla si ha habido cambios tal como se aprecia en Figura 6.1

Desde ahí podremos gestionar las aplicaciones a distintos niveles: Compilación programada, ejecución, integración en Sonar, etc..

6.2. Análisis en Sonar

Para poder controlar la calidad del software hemos utilizado Sonar.

Figura 6.1: Vista principal de Jenkins

Sonar [8]es una plataforma para evaluar código fuente. Sonar, que es software libre usa diversas herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa. Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, posible errores, comentarios y diseño del software. Aunque pensado para Java, acepta otros lenguajes mediante extensiones. Se integra con Maven, Ant y herramientas de integración continua como Atlassian Bamboo, Jenkins y Hudson).

Para poder acceder a Sonar, se ha habilitado una url `http://larabida.etsii.urjc.es:9000/` donde, al igual que en Jenkins, residen los tres proyectos dados de alta con la información detallada de cada uno.

En la Figura 6.2 se observa un total de 2427 líneas de código en los 1266 sentencias dividido en 17 ficheros que componen las 17 clases de las que consta la API en un único paquete con 346 métodos. Se puede observar también que hay un 15 % de comentarios En cuanto a las violaciones, no existe ninguna de tipo “Blocker” ni “Critical” que serían los tipos que inicialmente deberían ser preocupantes.

Ya para terminar, está el último cuadro, que recoge la complejidad ciclomática del código. En este caso, los resultados ofrecidos son: 1,6/ método; 32,6/ clase; 32,8/ archivo. La complejidad ciclomática determina, de forma cuantitativa, la complejidad lógica de un programa, o lo que es lo mismo, la calidad en la que está estructurado el sistema. Y más concretamente, define el número de caminos independientes del código

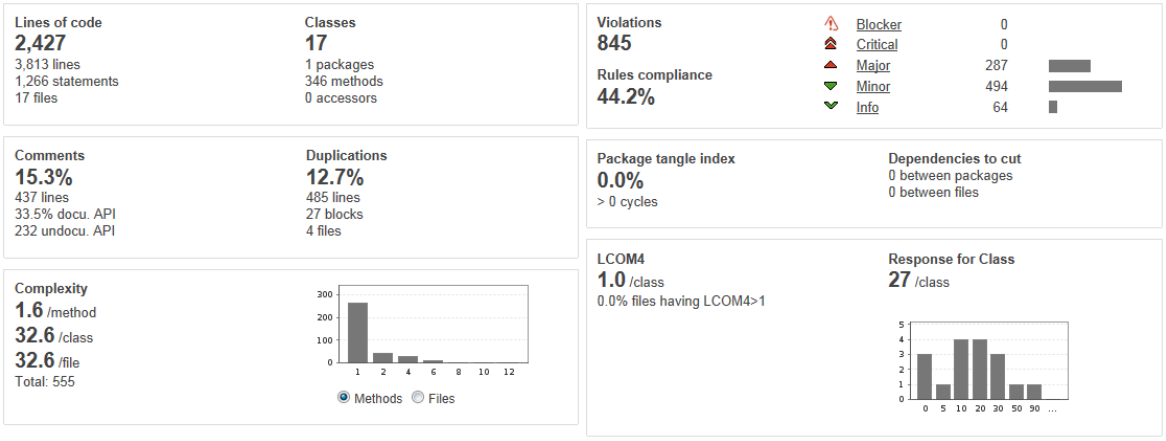

Figura 6.2: Vista del proyecto API en Sonar

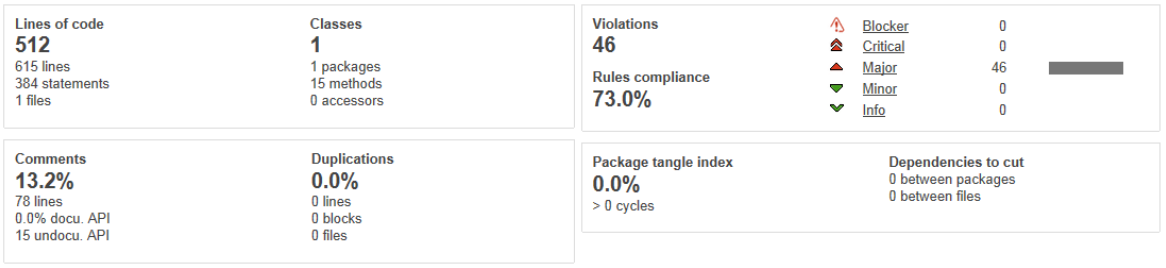

Figura 6.3: Vista del proyecto ScrumMasterServer en Sonar

analizado y determina la cota superior del número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. Así, el resultado de un análisis de la complejidad ciclomática permite determinar el riesgo que supone el código. Para ello, se utiliza la siguiente tabla:

Complejidad ciclomática	**Evaluación del riesgo**
"1-10"	Programa Simple, sin mucho riesgo.
"11-20"	Más complejo, riesgo moderado.
"21-50"	Complejo, Programa de alto riesgo.
"50"	Programa no testeable, Muy alto riesgo.

Así, como se puede observar el código sobre el que se ha hecho las pruebas, entra en los rangos aceptables.

En la Figura 6.3 se observa un total de 512 líneas de código, en un único fichero en una única clase, sin violaciones de interés y con un 13.2 % de comentarios.

En la Figura 6.4 se observa un total de 3558 líneas de código con un total de 1790 sentencias, divididas en 23 ficheros que componen las 29 clases de las que dispone este

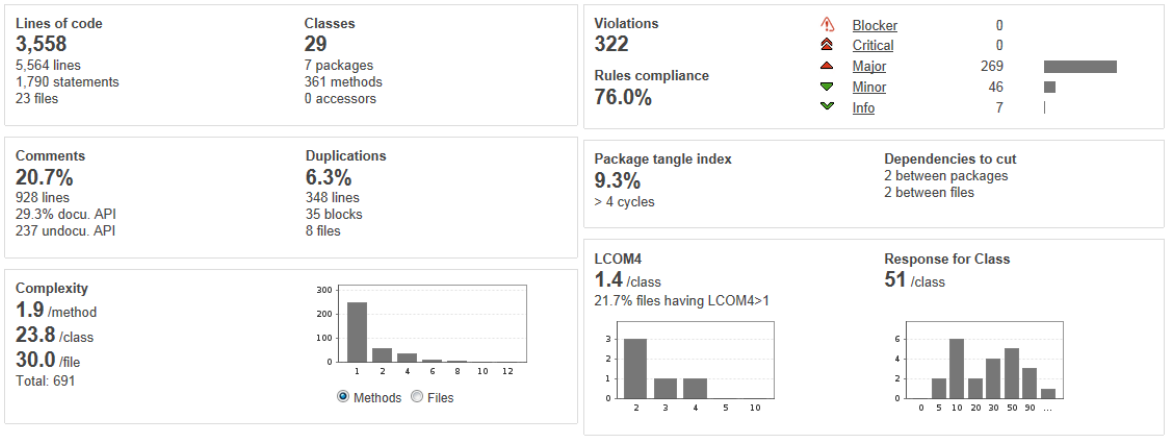

Figura 6.4: Vista del proyecto ScrumMaster en Sonar

proyecto. No existen violaciones digas de mención tal como se aprecia y se observa un 20.7 % de comentarios lo que arroja un total de 928 líneas. En cuanto a la complejidad ciclomática tenemos: 1,9/ método; 23,8/ clase; 30,0/ archivo observando nuevamente que esta entre los rangos esperados.

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se expone un resumen del procedimiento seguido para el desarrollo de este proyecto y se comentan algunas conclusiones al respecto. Además, se detallan diferentes líneas de trabajo que se pueden seguir para la continuación del proyecto, indicando de manera objetiva los puntos en los que se puede mejorar. Finalmente, se exponen algunas conclusiones personales.

7.1. Desarrollo de este proyecto

En los apartados que se han ido tratando en este documento se ha detallado el proceso de creación de una herramienta que cumpliera con los objetivos expuestos al comienzo de este proyecto.

En los primeros apartados se trató de introducir el contexto del problema, analizándolo en detalle con el fin de encontrar una solución que más se acercara al problema real. Así, se llegó al enunciado de los requisitos funcionales, y no funcionales, que supusieron la base sobre la que comenzar el trabajo. Una vez enunciados los requisitos, se pasó a un análisis de la tecnología. En este análisis surgió Google Web Toolkit, que supuso todo un descubrimiento. La idea de crear aplicaciones de interfaz Rica en Java accesibles desde un simple navegador, era algo que generaba un enorme abanico de posibilidades para solucionar el problema anteriormente expuesto. Pero para ello, fue necesario un largo proceso de documentación sobre esta tecnología, así como la generación de ejemplos sencillos que permitieran el aprendizaje, algo a lo que el anterior proyecto me ayudó bastante.

Cuando ya se dispuso de los conocimientos suficientes de la tecnología, se comenzó a analizar cómo se debía estructurar el trabajo, llegando así, a la parte de análisis de

la arquitectura. Así fue como, dividiendo el trabajo en la parte del cliente, y la del servidor,se pudo delimitar qué debía realizar cada una de estas capas. Por tanto, se detallaron las funcionalidades principales de cada uno de las partes. Asimismo, fue en este momento, en el que surgieron las primeras grandes modificaciones de la parte del servidor, quitando todas las llamadas a servicios RPC para crear la interface REST de la que se dispone en la actualidad, permitiendo abrir el abanico de posibilidad al uso del servidor para otras aplicaciones que utilicen estos datos.

Para la realización de la capa servidora se partía completamente de cero,suponiendo el mayor cambio con respecto a la aplicación anterior. Además, fué necesario completar algunas funciones en la API de acceso a datos, pues no contemplaba en su totalidad algunas funciones ahora necesarias. Esto, unido a la nueva gestión de librerías con Maven sitúa tecnológicamente a este proyecto un escalón por encima del proyecto anterior.

Por último, se procedió a la elaboración de esta memoria, que trata de explicar detalladamente cómo se obtuvo la aplicación que diera solución al problema expuesto.

7.2. Futuros trabajos

Como se puede comprobar, la aplicación presentada en este trabajo, aún no recoge todas las funcionalidades necesarias para Scrum. Para poder completar la aplicación se podría desarrollar los siguientes apartados:

- Gestión de Sprints.
- Gestión de Entregas.
- Gestión de Tareas
- Gestión de Historias.

Aunque la falta de estos apartados se puede ver compensada con una gestión de temas y subtemas eficiente que nos daría una funcionalidad análoga.

Además, se puede estudiar la posibilidad de serializar el envió de objetos desde el servidor, algo que en la actualidad no se ha realizado debido a la falta de librerías de deserializacion en GWT.

Puede plantearse un interface con gestión de informes y gráficas que representen las estadísticas de los diversos proyectos realizados.

En cualquier caso, las mejoras podrán estar sujetas a la demanda de los futuros usuarios de la aplicación. Así, las líneas futuras dependerán en gran medida de la acogida de esta herramienta por sus usuarios.

7.3. Conclusiones personales

La primera vez que conocí el proyecto, me impactó la forma de gestionar ventanas a través de una aplicación web. Desarrollar aplicaciones de fácil acceso y con un sinfín de funcionalidades y posibilidades en un entorno web sin necesidad de instalaciones costosas o de requerimientos inasumibles para algunas máquinas, era posible a través de un lenguaje de nueva generación como es GWT.

Una vez conocí la forma en la que funcionaba internamente la aplicación, se abrió un abanico de posibilidades de mejora tomando como primera opción la idea de mejorar el funcionamiento interno de la aplicación. Esto supuso un vuelco total a la hora de gestionar el tratamiento de los datos y las llamadas a la API de gestión. De esta manera pude dividir el proyecto en dos pequeños proyectos, la parte del cliente que dibujará el interfaz y la parte del servidor que gestionará las conexiones con la API y el tratamiento interno de los datos.

Esta segunda parte ha sido la más costosa. La implementación de la interface REST ha significado una mejora cualitativa importante abriendo un nuevo abanico de posibilidades para futuras aplicaciones. Esto unido a la implementación de la gestión de librerías con Maven hace mucho más fácil el futuro mantenimiento de la aplicación.

Toda estas nuevas tecnologías que he aprendido, hacen que el proyecto haya sido un éxito a nivel personal y laboral dado que el conocimiento adquirido me permite desarrollar diferentes ideas de implementación en diversas aplicaciones en mi entorno de trabajo.

Por último, otro de los apartados que he querido mejorar una vez finalizado el proyecto ha sido el relativo a la herencia de este proyecto para futuros alumnos, dejando una documentación más adecuada para el futuro desarrollo de la aplicación, que es algo que yo eché en falta cuando escogí este proyecto.

Apéndice A

Manuales

En este anexo se presentan tres puntos importantes a conocer de la herramienta. En el primer punto se aborda un manual de instalación de la aplicación. En el segundo punto se explica qué pasos seguir para poder continuar el desarrollo de la aplicación y en el tercer y último punto se encuentra un manual de uso de la aplicación.

A.1. Manual de Instalación para uso de la aplicación

A continuación se detallan en una serie de pasos, las acciones que deberán realizarse para manejar esta aplicación.

1. Instalación de MySQL en el equipo donde se desee instalar el servidor de la aplicación.

2. Creación de la estructura de tablas de la base de datos. Para ello, una vez instalado MySQL, bastará ejecutar el script de SQL "ApiScrumManager.sql" adjunto a la aplicación. Además de las tablas, este script generará automáticamente un usuario-administrador, que podrá ser borrado como cualquier otra persona perteneciente a la aplicación, y que permitirá su acceso inicial. El nombre de usuario será ScrumManager, y su contraseña inicial Scrum.

3. Instalación del servidor Apache Tomcat en el equipo servidor.

4. Configuración del archivo config.properties (Figura A.1), que se encuentra bajo el archivo ScrumMaster.war.

    a) Este archivo puede abrirse fácilmente por ejemplo con el programa WinRAR. Una vez abierto, hay que navegar hasta la carpeta lib (WEBINF\ lib),

```
# To change this template, choose Tools | Templates
# and open the template in the editor.

#Login name
login=root
#Keyword for root user
password=root
#Database url
url=jdbc:mysql://localhost:3306/scrummanager
#Fichero de log
log=D:\\fic.log
```

Figura A.1: Fichero config.properties.

donde se encuentran las librerías utilizadas por la aplicación; la que interesa modificar es ApiScrumManager. Utilizando de nuevo WinRAR, se puede abrir este fichero y dentro de él, se encontrará el fichero config.properties buscado.

b) Una vez localizado el archivo, se deberá abrir y configurar los parámetros que en él aparecen. En concreto, se deben ajustar los valores de url, que define la dirección de la Base de Datos. (Un ejemplo podría ser el siguiente: jdbc:mysql://localhost:3306/ScrumManager), el login normalmente usuario (root) y la contraseña o password que permita el acceso a la base de datos, y que se crea en la instalación de MySQL. Además en este mismo fichero se puede configurar donde se ubicará un fichero de log donde se irán registrando los problemas durante el uso de la aplicación.

5. Desplegar la aplicación. Una vez preparada la aplicación, se deberá descomprimir el archivo ScrumMaster.war en la carpeta del servidor Tomcat que sirva aplicaciones. Por ejemplo, en Ubuntu la carpeta será /var/lib/tomcat6/webapps.

6. Seguidamente se despliega el archivo ScrumMasterServer.war en la misma carpeta.

Una vez hechos estos pasos bastará acceder a la aplicación desde un navegador, comportándose a partir de ahora como cliente. Para ello, será necesario poner la dirección configurada para el Tomcat sobre el que se está sirviendo, añadiendo /ScrumMaster. Un ejemplo del tipo de dirección a poner podría ser: http://localhost:8080/ScrumMaster

A.2. Manual de preparación para desarrollo de la Aplicación

A continuación se detallan las herramientas necesarias para el futuro desarrollo de la aplicación. Todas las herramientas necesarias para el desarrollo están incluidas en el DVD anexo a este proyecto. Además existe una url de svn con el contenido del los diferentes proyectos disponible para bajarse directamente con un cliente svn.

Pasos a seguir:

1. Instalación del entorno de desarrollo NetBeans IDE.

2. Instalación del plug-in de desarrollo para GWT llamado GWT4NB.

3. Instalación de Maven.

4. Se abren los proyectos ApiScrumManager, ScrumMaster y ScrumMasterServer. El entorno se encargará de descargar las librerías necesarias referenciadas en los ficheros pom.xml correspondientes a cada proyecto.

5. Instalar un servidor Apache Tomcat dentro del entorno de desarrollo, para una vez que se tienen abiertos los proyectos se puedan lanzar las ejecuciones en local haciendo uso de este servidor.

6. Instalar servidor de Base de datos MySQL. Una vez instalado se debe dar de alta la base de datos tal cual se explica en el Anexo A.

Una vez realizados todos estos pasos, se tiene la posibilidad de tocar el código, compilar y ejecutar en un entorno de pruebas.

A.3. Manual de Usuario

Para ilustrar el uso de esta aplicación, se va a usar una base de datos con datos introducidos de manera aleatoria con el fin de mostrar de una manera mas explícita la forma de manejar la aplicación.

- Acceso a la aplicación. Para acceder a la aplicación basta añadir la dirección donde está el servidor. Ver Figura A.2.

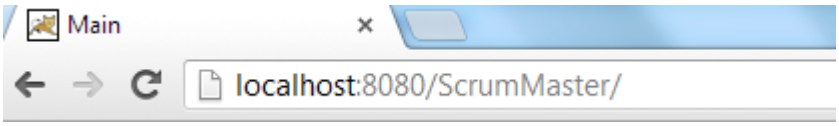

Figura A.2: Url de Acceso (`http://localhost:8080/ScrumMaster/`).

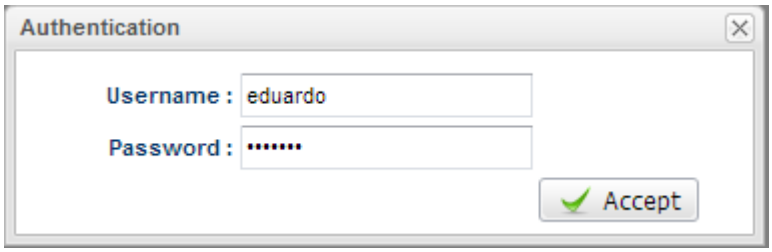

Figura A.3: Autenticación en la aplicación.

- Autenticación. Para poder acceder a la aplicación, será necesario disponer de un usuario y su contraseña.Sobre esta ventana cabe citarse que la contraseña que se introduce aquí, será cifrada mediante MD-5, puesto que en la base de datos no se almacenará, en ningún caso, la contraseña en claro. Así pues, la política de seguridad seguida dotará al cliente de una mayor confiabilidad en la aplicación. Además, no supone ningún inconveniente al cliente, y le es completamente transparente. En caso de que la contraseña introducida no sea válida, se mostrará un mensaje en la propia ventana indicando el motivo.Ver Figura A.3.

- Elección inicial de funcionalidades. Una vez logado en el sistema , aparecerá la que será la ventana principal de la aplicación. Sobre ella, se desplegarán todas las ventanas que se necesitan para manejar la aplicación. Así, se podrá abrir una u otra ventana desde la barra de herramientas que aparecerá en la parte superior de la pantalla.Ver Figura A.4.

- Administración de Usuarios. Pinchando en el botón “Manage Users” , nos aparecerá una ventana con los diferentes usuarios dados de alta en el sistema en el momento actual. Con el menú contextual sobre un usuario, se obtienen las diferentes acciones que se pueden realizar sobre el usuario seleccionado.Ver Figura A.5.

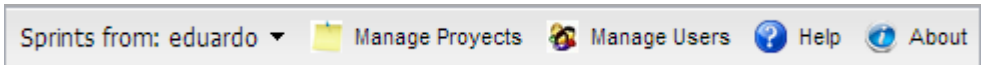

Figura A.4: Funcionalidades de la aplicación.

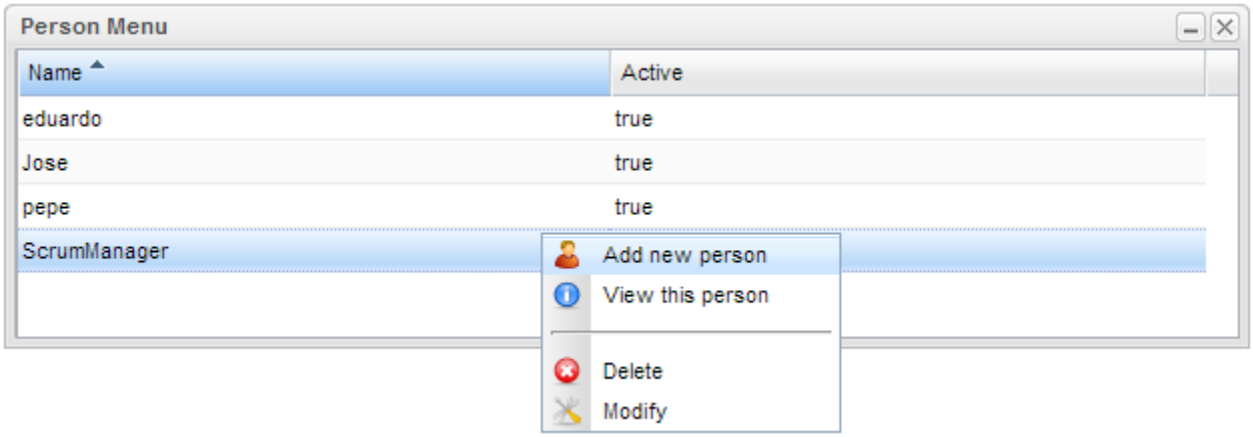

Figura A.5: Administración de Usuarios.

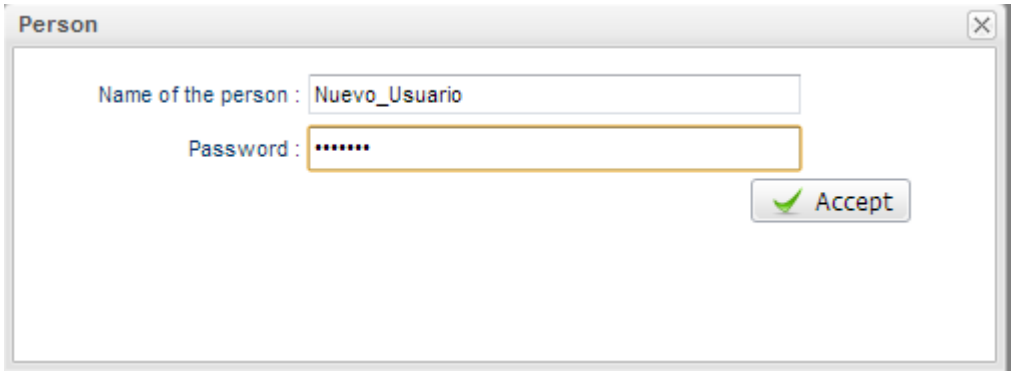

Figura A.6: Creación de nuevo usuario.

Para generar un nuevo usuario se pinchará en "Add new person" y se rellenará con los datos solicitados. Como se vé en la Figura A.6.

- Administración de Proyectos. Volviendo a la barra de herramientas superior, presionando sobre el botón "Manage Proyects" nos mostrará una nueva ventana con todos los proyectos que actualmente se tienen asociados a nuestro usuario. En ningún momento se pueden visualizar ni cambiar ningún proyecto que no se haya asignado. De igual forma, desde el menú contextual con el botón derecho se accede a una serie de operaciones sobre el proyecto seleccionado.Ver Figura A.7.

Así, se puede dar de alta un nuevo proyecto, que automáticamente quedará asignado al usuario con el que se ha dado de alta, o bien poder modificar un proyecto existente. Si bien, con la opción "Modify&Assign" se accede a una nueva ventana con dos pestañas. Una primera llamada "Data" donde se puede modificar los datos relativos al proyecto.Ver Figura A.8. Y otra segunda denominada "People" donde se puede,

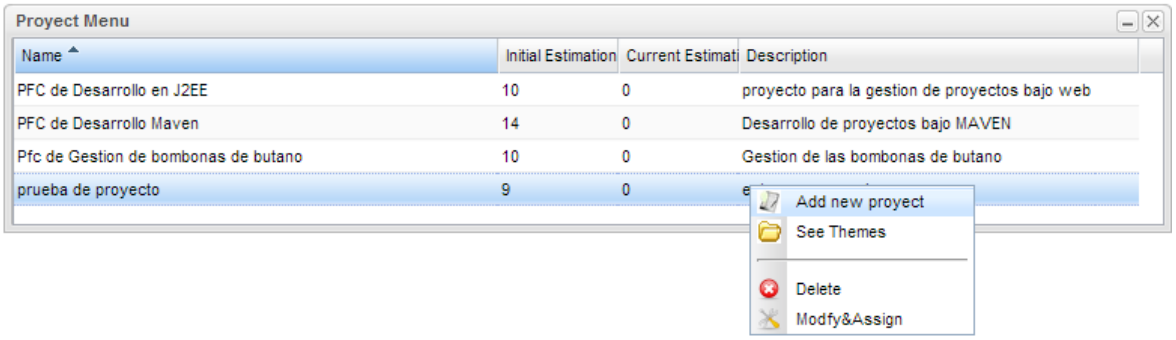

Figura A.7: Gestión de proyectos.

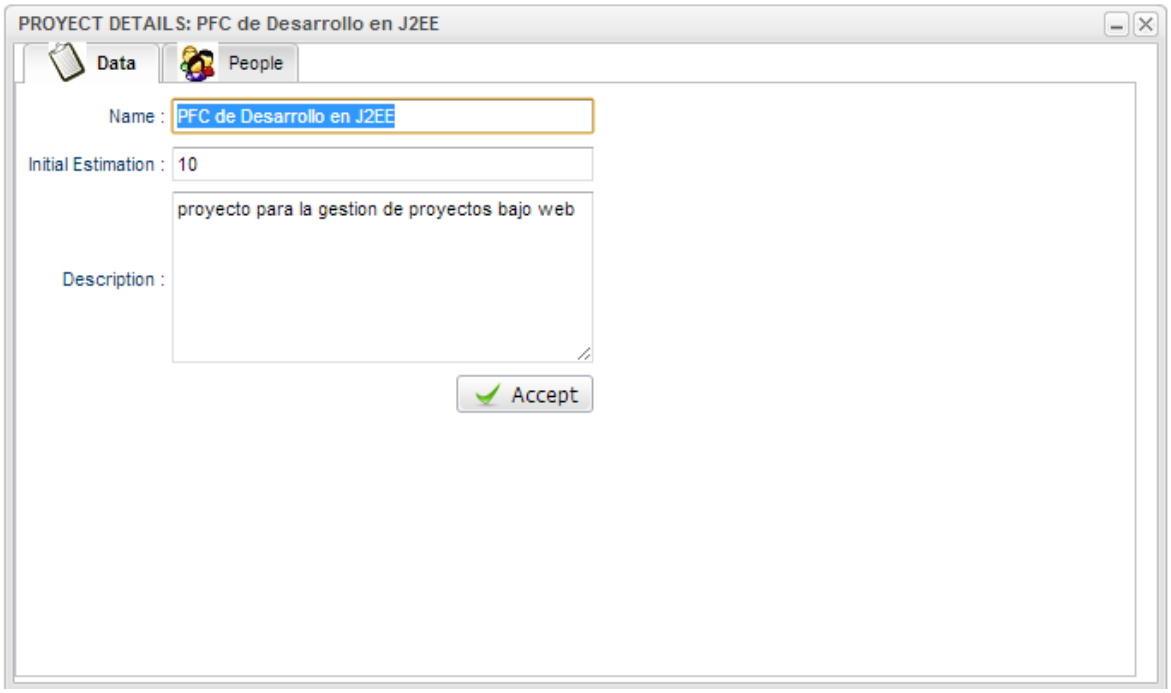

Figura A.8: Modificación de un proyecto.

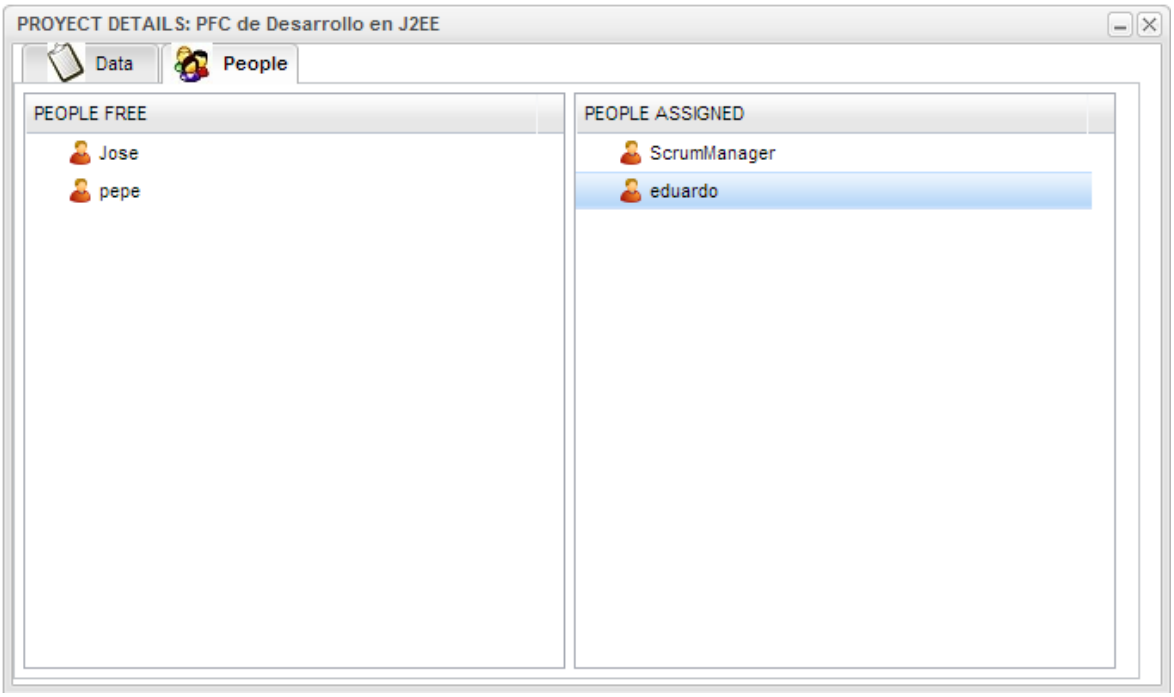

Figura A.9: Asignación de usuarios a proyectos.

cómodamente, asignar y desasignar mas personas a este proyecto. En ningún caso se puede desasignar nuestro propio usuario a un proyecto, se puede eliminar o asignarlo a otra persona para que nos desasigne , pero nunca dejar un proyecto sin asignación. Para pasar un usuario a la lista de asignaciones, bastará con hacer doble click sobre el nombre y de la misma forma se podrá desasignar (ver Figura A.9).

- Administración de Temas. Desde la opción “See Themes” en el menú contextual del proyecto, se puede administrar los temas asociados a el proyecto seleccionado. Aparecerá una nueva ventana con los temas ya creados y de igual manera con el botón derecho se permitirá crear un nuevo tema o modificarlo. Se puede crear un tema sobre un proyecto o sobre un tema, pudiendo crear así subtemas.Ver Figura A.10.

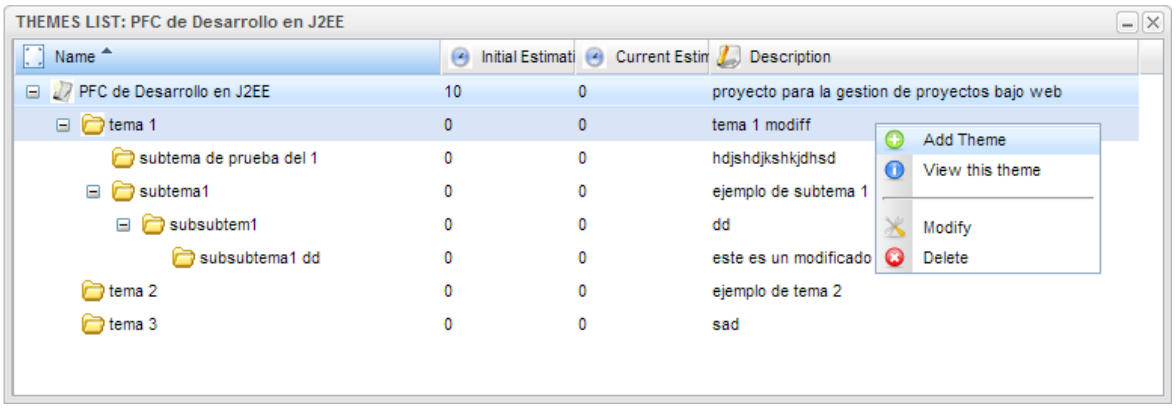

Figura A.10: Administración de temas.

A.4. Urls de Acceso

- El código fuente, memoria y el ejecutable se pueden descargar de:

`https://scrummanager.svn.sourceforge.net/svnroot/scrummanager/trunk`

- La aplicación se puede probar en:`http://mazagon.etsii.urjc.es:8080/ScrumMaster/` con las credenciales
  - Usuario: ScrumManager
  - Password: admin
- El servidor de integración continúa Jenkins es accesible desde la `http://larabida.etsii.urjc.es:8000/` y requiere usuario con privilegios.
- El servidor Sonar es accesible desde `http://larabida.etsii.urjc.es:9000/` y no requiere autenticación.

Bibliografía

[1] AKENDOS. Proceso de desarrollo scrum. http://www.akendos.com/content/el-proceso-de-desarrollo.

[2] Postulado api. http://en.wikipedia.org/wiki/Joshua_Bloch/.

[3] esgooglewebtoolkit. Gwt. http://esgooglewebtoolkit.blogspot.com.es/2007/09/la-arquitectura-de-google-web-toolkit.html.

[4] Iteracion scrum. http://www.info2000.cl/sites/default/files/imagenes/imagen1

[5] Modelo comparativo web. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.

[6] ProyectosAgiles.com. Reunion diaria de sincronizacion del equipo. http://www.proyectosagiles.org/reunion-diaria-de-sincronizacion-scrum-daily-meeting.

[7] La Salle. Software de gestion de proyectos. http://blogs.salleurl.edu/project-management/20-software-gratuitos-para-la-gestion-de-proyectos/.

[8] Sonar. Sonar. http://www.sonarsource.org/.

[9] ThomasWalletBlog. Tablero scrum. http://thomaswallet.blogspot.com.es/2012/02/las-10-mejores-extensiones-para-tunear.html.

[10] Wikipedia. Jenkins. http://es.wikipedia.org/wiki/Jenkins.

[11] Wikipedia. Json. http://es.wikipedia.org/wiki/JSON.

[12] Wikipedia. Maven. http://es.wikipedia.org/wiki/Maven.

[13] Wikipedia. Metodologia agil. http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software, category = SCRUM, key = Agil.

[14] Wikipedia. Rest. http://es.wikipedia.org/wiki/Representational_State_Transfer.

[15] Wikipedia. Scrum. http://es.wikipedia.org/wiki/Scrum.