Trabajo de Fin de Grado

Aplicación para la Centralización y Automatización del Proceso de Release de aplicaciones web

Junio 2014 – Las Palmas de Gran Canaria

Alumna:

Maylin Leal Vilariño

Grado en Ingeniería Informática (Ingeniería del Software)

Tutor:

Dr. Alexis Quesada Arencibia

Ciencias de la Computación e Inteligencia Artificial

Resumen

Esta aplicación surgió de la necesidad real del equipo de desarrolladores encargados de llevar a cabo el proceso de despliegue de aplicaciones web en empresas que siguen modelos de desarrollo iterativos haciendo uso de grandes procedimientos que deben ser aplicados con bastante frecuencia.

El objetivo fundamental del proyecto es proporcionar a estos usuarios una aplicación que sirva de ayuda en el proceso de despliegue y centralice toda la información importante y útil que debe ser comprobada de manera constante y automatice todas aquellas tareas triviales que constituyen una pérdida de tiempo para los encargados del proceso de despliegue y disminuyen la concentración en el propio proceso y los detalles importantes y, por lo tanto, la calidad final del proceso de despliegue.

La aplicación desarrollada será útil para la organización y control del proceso y además ofrecerá beneficios en cuanto a comunicación entre aquellos interesados en la nueva versión a desplegar. De modo general, la aplicación ofrecerá una forma de monitorizar los distintos estados de la nueva versión y mantendrá un sistema de rastreo de todos los eventos sucedidos durante la etapa de despliegue e información relacionada, además de automatizar tantas tareas como sea posible.

En este proyecto se gestiona la creación de esta aplicación y se llevan a cabo fases de elicitación de requisitos, análisis, diseño, implementación y pruebas, con el fin de concluir con una versión operativa final de la aplicación que cubra las necesidades iniciales de los usuarios afectados.

Summary

The main goal of this Project is to develop a tool that will make easier the process of realeasing a new version of a web application by automating different tasks and organizing all the important and useful information in only one place.

The developed tool will be useful for the users that have to carry out the Release in order to organize and control the process. This application offres a way to monitor the different states of the new version to be deployed and keeps tracking of everything that happens during the involved period of time.

With the use of this tool the users will lose less focus into important process by having all the needed information in a centralized tool and all the common tasks will be automated in order to reduce the dummy work for the Release managers and increase the quality of the Release process.

This project covers everything related to the creation of this tool, including Management, Specification of Requirements, Analysis, Design, Implementation of all required functionalities and final Testing.

Agradecimientos

A mi familia por todo el apoyo, los ánimos y la confianza depositada en mí, no sólo durante la realización de este proyecto sino durante cada año de estudio e incluso antes, en cada momento de mi vida y mi formación personal y profesional. En especial, un agradecimiento enorme a mi madre por todo su sacrificio, sin la que este proyecto no hubiera sido posible y a la que debo cada una de las grandes oportunidades que he tenido y me han llevado a donde estoy hoy.

A mis amigos, familiares y personas más allegadas por soportar mis cambios de humor y estrés y aún así continuar ofreciéndome todo su apoyo. Por los consejos, los momentos de desconexión y simplemente, por estar ahí cuando los he necesitado y cuando no.

Un agradecimiento especial a la empresa donde este proyecto ha sido realizado y a todas las personas que han estado involucradas de un modo u otro en su realización. A mi equipo “42” por todo el apoyo, los ánimos y los excelentes consejos técnicos recibidos y a todos los miembros de los equipos de desarrollo de la empresa en general, el equipo de Recursos Humanos y a mi jefe, CTO de la empresa, por su gran trabajo, profesionalidad y ayuda, y principalmente por permitir que la realización de este proyecto fuera posible. Gracias por la magnífica oportunidad y experiencia.

Esta sección no podría terminar sin un gran agradecimiento a la Universidad y a todos los profesores que de un modo u otro han marcado una parte de mi vida y me han provisionado con enseñanzas técnicas y enseñanzas de vida, sentando las bases para mi desarrollo profesional. Un agradecimiento especial a mi tutor Alexis Quesada por aceptar ser mi tutor y permitir la gran aventura que ha sido la realización de este Trabajo de Fin de Grado. Sin tu consentimiento, ánimos y guía este proyecto no hubiera llegado a existir.

Índice

Índice de ilustraciones

Introducción

En los tiempos actuales cada día se desarrollan más proyectos informáticos y de forma más eficiente. Sin importar la naturaleza de los proyectos, estos se han convertido en la principal herramienta de solución de la gran mayoría de los problemas de la humanidad. Todos los proyectos tienen una razón por las que han sido concebidos: algunos buscan proporcionar entretenimiento, otros mejorar la calidad de vida en variadas formas. De cualquier manera, absolutamente todos buscan satisfacer un problema orginal que tiene el cliente que los solicita o un grupo de usuarios.

Cada día existen más elementos automatizados y nos hemos vuelto más dependientes de los equipos informáticos y los sistemas software. Las nuevas tecnologías nos inclinan al uso de nuevas herramientas software y a su inclusión en nuestra vida cotidiana. Si bien es cierto que muchos proyectos software no ofrecen un valor imprescindible para el ser humano, también es verdad que muchísimas tareas se vuelven más agradables, eficientes y fiables gracias a su automatización o al uso de proyectos software para gestionarlas.

Este proyecto nace como solución y respuesta a un problema real de los usuarios encargados de ejecutar procesos de despliegue de aplicaciones web. El proyecto pretende facilitar el desarrollo y gestión de dicha tarea mediante diversas estrategias que serán comentadas en puntos posteriores.

1. Capítulo 1: Estado actual y objetivos generales

En la actualidad muchas empresas tienen procesos de despliegue muy pesados, con muchísimos procesos incorporados y demasiadas herramientas de monoitoreo que deben ser sincronizadas.

Además, cada día las aplicaciones están más soportadas por múltiples servidores con el fin de ofrecer una mayor disponibilidad y tolerancia a fallos. Este incremento de servidores provoca que los encargados de realizar el despliegue de una aplicación tengan que estar pendientes de más servidores y de sus estados en cada momento del proceso.

Otro punto a observar es el considerable incremento de la aplicación de metodologías de desarrollo ágil al desarrollo de proyectos en las distintas compañías sofware alrededor del mundo. Es bien sabido que una de las principales características de estas metodologías es el desarrollo del proyecto basado en iteraciones, con lo cual el número de despliegues a realizar se incrementa y con ello se incrementa también todo el trabajo que conlleva este proceso. Otro efecto secundario de tener tantas iteraciones para las que hacer despliegues es el hecho de que se debe dejar constancia de las nuevas características incorporadas en cada iteración, así como de los problemas encontrados y demás cuestiones interesantes que la empresa de desarrollo considere oportuno.

De forma general, la principal motivación de este proyecto es la elaboración de una herramienta que facilite el proceso de despliegue en las empresas de desarrollo, desde el punto de vista organizativo.

1.1. Descripción del Proyecto

Esta idea surgió al reflexionar sobre la gran cantidad de tareas que se deben realizar durante el proceso de Release de aplicaciones web en grandes empresas. En general, las personas encargadas de dicho proceso deben estar pendientes de muchísimas cosas que son independientes entre sí, siguiendo una lista de instrucciones para no pasar por alto ninguno de los pasos importantes.

La idea principal es crear una aplicación web que centralice todos los procesos e información necesarios para llevar a cabo una Release sin inconvenientes y reduciendo al mínimo el esfuerzo humano que implica mantener en coordinación y bajo control tantas herramientas, información y procesos distintos.

El proyecto se plantea para ser construido desde los cimientos, incorporando además de las etapas de análisis, diseño e implementación, todas aquellas relacionadas con la administración, manejo y gestión de proyectos técnicos.

Es un proyecto rico para aplicar todos los conocimientos de Ingeniería y Arquitectura de Software adquiridos y para crecer como manager de proyectos informáticos. Además, este proyecto es interesante pues puede ser muy flexible si se construye de forma modularizada y basada en eventos y además tiene enormes posibilidades de crecer. A todo lo anterior hay que

añadir que está enfocado a la solución de un problema real, lo cual aporta bastante motivación.

Para desarrollar el Proyecto se utilizarán metodologías de desarrollo ágil para basarlo en iteraciones y tener siempre una versión operativa disponible, de modo que, a forma de anillos, se pueda ir creando desde el núcleo, con las funcionalidades más básicas, e ir incrementando el tamaño con cada iteración.

1.2.Objetivos

La finalidad es servir de soporte a los usuarios encargados del proceso de Release para ahorrar esfuerzo humano y aumentar la calidad del análisis, seguimiento y organización, ya sea porque las distintas tareas se han automatizado o porque la información para desarrollarlas se encuentra más organizada y accesible.

Los principales objetivos son:

- Centralizar toda la información en un único sitio de acceso.
- Automatizar tantas tareas relacionadas con el proceso de Release como sean posibles.

1.3.Situación actual

En este momento, el proceso de Release de aplicaciones web en grandes empresas que sigan desarrollo ágil basado en iteraciones puede abarca unos 3 días y realizarse cada 2 semanas. Existe un grupo de personas implicadas en llevar a cabo todo el proceso (de ahora en adelante, Release Crew) y suele haber una serie de pasos establecidos de antemano que sirven de guía para el procedimiento.

Durante el proceso de Release, la Release Crew debe, entre otras cosas, enviar emails de notificación a los desarrolladores y otros Departamentos, crear y notificar distintas Meetings para mantener sincronización con los desarrolladores e informar de progresos y posibles errores, estar pendiente de las distintas herramientas de monitoreo del Sistema (Ficheros Log, Sistema de manejo de errores, Herramientas para chequear Performance, etc.), conocer el estado del testeo del Sistema, el estado en el que se encuentran los distintos Servidores y por supuesto llevar la cuenta mental sobre qué tareas de la Lista preestablecida ya se han realizado y cuáles están aún pendientes.

Para controlar el “estado del testeo” del Sistema es necesario controlar el deploy en los servidores, saber qué versión se encuentra operativa y estar pendiente de posibles nuevos errores, además del estado de los Test Cases del equipo de Quality Assurance.

Al terminar, es necesario dejar constancia en algún sitio de todo el proceso llevado a cabo: fechas de deploy de las distintas versiones, problemas encontrados, soluciones, características incluídas en la nueva entrega, etc.

Conclusión: Organización, preparación, comunicación, supervisión de ejecución de migraciones en base de datos, monitoreo del sistema, múltiples deploy de la nueva versión y atención rigurosa al testeo. Todo ello atendiendo en cada momento herramientas distintas.

1.4.Situación posterior a la realización del Proyecto

La Release Crew tendrá acceso a una aplicación web que centralizará todos los procesos citados anteriormente, creando la sensación para el usuario de dominio absoluto sobre el estado de la Release y toda la información necesaria para desarrollar la tarea sin inconvenientes y reduciendo al mínimo el esfuerzo humano que implica mantener en coordinación y bajo control tantas herramientas y procesos distintos durante estos días.

- Mediante esta herramienta se podrá tener acceso directo a la información relevante que se necesite, ofreciendo la posibilidad de personalizar el entorno de trabajo atendiendo al rol que juegue el usuario en el proceso de Release.
- Se ofrece la posibilidad de automatizar tantos procesos como sean posibles, por ejemplo: el envío de ciertos correos habituales de información, la creación de las meetings habituales, selección de la próxima Release-Crew (basado en un proceso cíclico), elaboración de un prototipo de la página final de información acerca del proceso de Release (atendiendo a las fechas de deploy y los errores vistos), envío de información a otros Departamentos en momentos puntuales, sugerencia de asignación de bugs durante los días de testeo basando la decisión en la información obtenida del error y posibles tickets relacionados con el error y el nombre de un desarrollador, etc.
- Mantener un tracking de las acciones de la Release-Crew.
- Almacenar estadísticas acerca de qué procesos son más costosos en cuestión de tiempo y esfuerzo humano y cuáles suelen tener más problemas con el fin de mejorar el proceso general.
- Integración de los distintos servicios usados en la compañía para sacar el mayor provecho posible a la herramienta, avanzando un paso más y no utilizándola solamente como distribuidor de información. (Servicios: icinga, errbit, bamboo, calendar, email, internal chat...)
- Utilizar una “Checklist” dinámica que te permita ver qué tareas ya han sido realizadas, cuáles están pendientes y cuáles, en el momento actual, deberían estarse llevando a cabo.

1.5.Mejoras que supone el uso de la nueva aplicación

De forma general, estas son las principales mejoras y beneficios que ofrece el uso de esta aplicación:

- Organización y control sobre todo el proceso de Release.
- Acceso a estadísticas, análisis y cualquier tipo de información acerca de lo acontecido durante el proceso.
- Considerable disminución del esfuerzo humano invertido en la preparación y realización del proceso y por lo tanto, incremento de su calidad.
- Mejoras en la comunicación.
- Facilita la visión global del proceso y la toma de decisiones.

Para tener una idea visual de las diferencias que supondría un Proyecto así, se ha creado dos Diagramas de Secuencia muy generales, que sólo pretenden ofrecer con mayor claridad una imagen global, aunque no completa, del proceso de Release antes y despues de la elaboración de la Aplicación. Estos diagramas se encuentran como anexos en este documento.

2. Capítulo 2: Competencias del Proyecto

2.1.Comunes a la Ingeniería Informática

2.1.1. CII08

Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.

Se ha realizado la elección de las distintas herramientas y tecnologías atendiendo a criterios de calidad, facilidad de uso y actualización. Se utilizan tecnologías potentes, avanzadas y punteras, como el lenguaje de desarrollo y framework web elegidos: Groovy & Grails. Las etapas de análisis y diseño han sido bien desarrolladas, con bastante tiempo invertido y siempre bajo control, planificación y manteniendo una conexión entre las diferentes etapas, ya que inevitablemente se encuentran relacionadas y la salida de una debe ser el punto de entrada de la siguiente.

2.1.2. CII011

Conocimiento y aplicación de las características, funcionalidades y estructura de los Sistemas Distribuidos, las Redes de Computadores e Internet y diseñar e implementar aplicaciones basadas en ellas.

El proyecto es una aplicación web que a su vez integra distintos servicios web para los que hay que implementar comunicación a través de distintos tipos de APIs con tecnologías y modos de acceso distintos. Además, esta aplicación está pensada para ser ejecutada en una red interna por lo que el manejo de redes VPN es necesario, reforzando de esta forma los conocimientos sobre Redes de Computadores. Finalmente, se realiza la integración con algún servicio de email, con lo cual es necesario poder en práctica todo el conocimiento sobre servidores email y su correcta configuración.

2.1.3. CII012

Conocimiento y aplicación de las características, funcionalidades y estructura de las bases de datos, que permitan su adecuado uso, y el diseño y el análisis e implementación de aplicaciones basadas en ellos.

La aplicación tiene una base de datos en la que se guarda la información importante y básica. Es necesario la ejecución de queries para el acceso a los datos y el conociemiento del funcionamiento de las bases de datos para su configuración inicial y posterior manejo. Como extra ha de mencionarse que se realizó un análisis comparativo entre las distintas opciones de base de datos a usar antes de la selección final.

2.1.4. CII013

Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los Sistemas de información, incluidos los basados en web.

Se realiza la integración de distintos servicios web por medio de las APIs que ofrecen para solicitar información o para automatizar distintas tareas relacionadas con ellos. Para ello ha sido necesario realizar una investigación acerca de los métodos ofrecidos por las APIs y para las conecciones se ha debido utilizar distintas técnicas y herramientas como el WebClient de Java y la librería HtmlUtils, además del uso del formato JSON y su correspondiente "parser" para realizar correctamente la comunicación con los servicios externos integrados. Específicamente se ha realizado la integración con los servicios Bamboo, Errbit, Jira y con los servidores.

2.1.5. CII016

Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.

Se ha seguido distintas metodologías de desarrollo ágil y se ha aplicado las distintas fases que estas metodologías proponen. De la misma forma, se ha generado los artefactos propuestos por dichas metodologías, en la medida en que eran útiles para el desarrollo del proyecto. Principios como "DRY", "KISS", "Open-close" y "Single Responsability" han sido aplicados de forma global a lo largo de todo el proceso de desarrollo y diseño. El conocimiento sobre el ciclo de vida del software se hace evidente en el manejo de las distintas fases del proyecto. Asimismo, el conocimiento general sobre los distintos principios y metodologías, así como su aplicación, se puede observar en apartados posteriores de descripción de estrategias de gestión del proyecto a seguir y apartados de análisis y diseño del sistema.

2.1.6. CII017

Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.

Se ha diseñado completamente la interfaz de usuario de la aplicación en base a criterios de usabilidad. Como se verá en el capítulo oportuno, la primera versión de interfaz de usuario fue deshechada para garantizar la flexibilidad de añadir nuevas áreas a la aplicación y para ofrecer al usuario la posibilidad de acceder en todo momento a cualquiera de las áreas principales de la aplicación, sabiendo siempre en qué lugar se encuentra y evitando así el malestar que provoca la sensación de estar perdido dentro de una aplicación software. Además, el estilo de la interfaz de usuario sigue principios de simplicidad para ofrecer con mayor claridad el contenido importante.

2.1.7. CII18

Conocimiento de la normativa y la regulación de a informática en los ámbitos nacional, europeo e internacional.

Se ha realizado un trabajo investigativo para incluir en la memoria del TFG un capítulo entero que abarque este tema, señalando las normativas más importantes y su

impacto sobre este proyecto, así como las distintas leyes y acuerdos nacionales e internacionales de mayor relevancia.

2.1.8. TFG01

Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

En el presente documento y todos los demás relacionados con este proyecto se exponen estrategias, modelos y técnicas utilizadas para el desarrollo del mismo, así como el uso de múltiples herramientas que, junto a las debidas explicaciones y justificaciones de uso, ponen de manifiesto la aplicación de todos los conocimientos y competencias adquiridas durante la carrera en este proyecto final.

2.2.Relativos a la especialidad Ingeniería del Software

2.2.1. IS01

Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

Se ha llevado a cabo una intensa y completa planificación de todo el proceso para cumplir con las expectativas del usuario, entregar software de calidad y mantener el desarrollo en el presupuesto de tiempo acordado. Se ha aplicado muchos principios de diseño software, patrones de diseño, patrones arquitectónicos (MVC) y estilos arquitectónicos (Layered Architecture) para garantizar una robusta y flexible arquitecture del software. Además, se ha aplicado las buenas prácticas de la ingeniería del software: seguir una metodología de desarrollo, estándares de especificación, realizar modelados, prototipados de interfaz de usuario, etc.

2.2.2. IS02

Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones.

Se ha realizado reuniones con personas claves para determinar requisitos y definir el dominio de la aplicación. Se ha dividido los requisitos en base a prioridades y por lo tanto, ha existido la necesidad de llegar a acuerdos con el cliente acerca de cuáles son los requisitos que potecialmente no serán incluídos en la versión final del producto para así mantenernos dentro del presupuesto de tiempo. Además, se realizaron acuerdos para identificar cuáles servicios serán integrados en la aplicación y cuáles se mantendrán como futuras mejoras, nuevamente para garantizar el mantenernos dentro del presupuesto.

2.2.3. IS03

Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles.

La aplicación se ha desarrollado en iteraciones que necesitan ser integradas al finalizar cada iteración. Como principal medio de control de versiones se ha usado GIT para utilizar un sistema de branching que ayude en el procedimiento. Además, la herramienta web Stash también ha servido de ayuda en este aspecto.

2.2.4. IS04

Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

Se ha seguido procedimientos básicos a la hora de comenzar la planificación y manejo de proyectos software. En este caso se usan principalmente estrategias de metodologías de desarrollo ágiles y distintas técnicas de modelado mediante diagramas UML para garantizar un óptimo resultado, una buena documentación y facilitar el desarrollo de las distintas etapas de vida del software. Se declara esta competencia satisfecha al haber desarrollado con éxito cada una de las etapas mencionadas y al haber, en cada uno de ellas, haber analizado las distintas soluciones software a aplicar. Estos procesos se hacen evidentes durante el desarrollo de los distintos apartados de esta memoria.

2.2.5. IS05

Capacidad de identificar, evaluar y gestionar los riesgos potenciales asociados que pudieran presentarse.

Se ha identificado los mayores riesgos de este proyecto en etapas iniciales del mismo. Durante todo el proceso de desarrollo se ha prestado especial atención a estos riesgos y se ha comunicado su existencia a las distintas partes interesadas. Asimismo, se han adoptado medidas para mitigar los riesgos el máximos posible. Los mayores riesgos identificados son los siguientes:

- Las APIs de los servicios integrados pueden cambiar en cualquier momento y algunos elementos del software se verían comprometidos. Se ha asilado y encapsulado los distintos elementos que intervienen en la comunicación con servicios externos para reducir el impacto que tendrán dichos cambios sobre la aplicación y se ha desarrollado las distintas funcionalidades asociadas de forma que un fallo de comunicación sea un escenario probable y la respuesta no quibre el sistema.

- Hay demasiados requisitos funcionales para un presupuesto de tiempo tan limitado. Inevitablemente no todos los requisitos serán implementados, por ello la lista de requisitos se ha dividido en distintos grupos atendiendo a criterios de prioridad, riesgos y facilidad de implementación.

2.3.Relativos a la especialidad Tecnologías de la Información

2.3.1. TI06

Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.

El proyecto es una aplicación web, basada en un framework de desarrollo web que hace uso de un servidor web y además se encuentra integrado con varios servicios web externos.

3. Capítulo 3: Aportaciones

3.1.Aportación al entorno socio-económico

El desarrollo de esta aplicación ofrecerá a los usuarios de la misma un medio para controlar todo el proceso de Release de aplicaciones web de una forma organizada, con lo que se conseguirá mejorar la calidad del proceso y finalmente invertir menos tiempo en el mismo y/o garantizar que el proceso se lleve a cabo con éxito en un alto porcentaje. Esto repercute directamente en el entorno económico, ya que el momento de Release de una nueva entrega es crucial desde el punto de vista económico y administrativo de una empresa pues los clientes están esperando las nuevas características incluídas en la entrega.

Además, este proyecto ofrece al grupo de management de la empresa una herramienta con la que medir y evaluar la calidad del proceso de Release y con ello la facilidad de mejorar el proceso atendiendo a los aspectos más críticos. El proceso se puede evaluar haciendo uso de las estadísticas que se inluirán acerca del tiempo invertido en el proceso y la cantidad de errores nuevos que aparecen y están directamente relacionados a una nueva entrega.

Un segundo y crucial aporte en este entorno es el modo en que se ha llevado la administración y control del proyecto a lo largo del ciclo de vida. Se ha aplicado distintas técnicas y mezclado varias metodologías de desarrollo de software con el objetivo de garantizar la calidad del proceso y facilitar el desarrollo del mismo. Esta mezcla de procesos y por consiguiente, la creación de un nuevo proceso de desarrollo de software, o metodología, constituye la mayor aportación de este proyecto al entorno socio-económico, pero también al entorno científico-investigativo que busca cada día el desarrollo de mejores e innovadoras metodologías de desarrollo.

En este documento, se ha creado un capítulo especial dedicado a la descripción del proceso seguido para realizar la administración y planificación del proyecto y que constituye una de las mayores aportaciones de este proyecto.

3.2.Aportación al entorno técnico

La aplicación ofrece una arquitectura flexible y lista para integrar diferentes servicios. Desde el punto de vista técnico se puede ver cómo es posible integrar una gran variedad de herramienta y servicios de forma independiente sin afectar a los existentes y además aplicar las distintas herramientas que ofrece el mercado en el beneficio del desarrollo de una sola aplicación.

3.3.Aportación a nivel personal

Personalmente este ha sido un reto increíblemente grande pues se trata de un proyecto que involucra el uso de muchísimas herramientas software, el trabajo en áreas de desarrollo muy variadas y por supuesto, la toma de decisiones y la aplicación de metodologías de desarrollo y toda la responsabilidad, planificación, administración y procesos que ello implica.

He visto aplicado en este proyecto cada uno de los conocimientos adquiridos durante la carrera y a medida que he avanzado he podido mejorar la mayoría de ellos y adquirir en el camino nuevos conocimientos y muchísima experiencia, pues este ha sido el primer gran

proyecto del que soy totalmente responsable y todas las deciciones de desarrollo y sobretodo, las de planificación, han sido llevadas a cabo a nivel personal.

Ha sido un reto desde el punto de vista administrativo y un conjunto de enseñanzas desde el punto de vista técnico.

Desde el punto de vista técnico, se ha incrementado el conocimiento sobre el frontend que se aplica a los proyectos: plugins, templates, librerías, últimas tecnologías y tendencias así como el conocimiento de buenas prácticas, estándares y el refuerzo del conocimiento sobre el uso de html, css y javascript. Otros aspectos son el diseño de una arquitectura completa y modularizada, el uso de diversas tecnologías backend y la comunicación con diversos servicios externos, punteros en la actualidad, mediante APIs. El reto de la comunicación basada en APIs radica en que cada servicio integrado en la aplicación ofrece una API distinta y la necesidad de un modo distinto de acceso, por lo tanto el reto y la enseñanza son siempre nuevos.

Desde el punto de vista organizativo, el proyecto ha constituido una fuente valiosísima de información y experiencia. He podido aplicar distintas y nuevas técnicas y metodologías de planificación, organización y dirección de proyectos técnicos y he podido comprobar el impacto de las mismas en el desarrollo del proyecto. De esta forma tengo una idea clara de cómo enfocar los proyectos en el futuro y llevar a cabo su planificación y manejo, sabiendo el impacto y consecuencia de ciertas decisiones y sabiendo de antemano cómo reaccionar a ciertas alertas que, a ojos de quien no ha realizado un proyecto como este, podrían pasar desapercibidas.

4. Capítulo 4: Normativa y Legislación

En este capítulo se recogen las principales leyes y normativas que afectan a los proyectos software, en especial aquellas relacionadas con el tipo de aplicación que se ha desarrollado como TFG.

En el ámbito de la Informáica y el desarrollo software es necesario aplicar normativas y leyes con la misma fuerza que se aplican en ámbitos no virtuales y digitales debido a la creciente utilización de productos software, la digitalización de la información y la automatización de la mayoría de las tareas que realiza el ser humano.

Almacenamiento de información personal, transacciones, distribución de información y productos a través de la web, son ejemplos de acciones propensas al crimen y la estafa que deberían ser controladas y normalizadas por distintas leyes y por organismos legales de poder.

Algunos de los delitos más recurrentes son el acoso vía web, el cibercrime, delitos contra la intimidad, fraudes y estafas y ventas ilegales.

En 1990 la Organización de las Naciones Unidas (ONU) en el Octavo Congreso sobre Prevención del Delito y Justicia Penal, celebrado en La Habana, Cuba, se dijo que la delincuencia relacionada con la informática era consecuencia del mayor empleo del proceso de datos en las economías y burocracias de los distintos países y que por ello se había difundido la comisión de actos delictivos.

A continuación se detallan brevemente algunas de las leyes nacionales e internacionales más relevantes, que se encuentren directa o indirectamente relacionadas con el producto software desarrollado bajo los términos de este TFG.

4.1.Generales

4.1.1. Licencia de software

Es necesario tener una licencia válida de todo software utilizado para el desarrollo, soporte y mantenimiento del producto software construido y cumplir con sus requerimientos. En especial, se debe utilizar una licencia válida de todos aquellos programas que sean premium o de pago. El hecho de utilizar material pirata para el desarrollo del software constituye un delito por sí mismo y repercute en la validez y legitimidad del software final realizado.

¿Qué es una licencia software?

Estas licencias básicamente son un contrato entre el autor del programa y el usuario, y comprenden una serie de términos y cláusulas que el usuario deberá cumplir para usar el mismo.

Existen muchos tipos de licencias software y variantes de las existentes. A continuación se describen brevemente aquellas correspondientes a los programas utilizados para la realización de este proyecto y aquellas que forman las base de las licencias finalmente utilizadas.

4.1.1.1.Conceptos base

Copyleft

La mayoría de las licencias usadas en la publicación de software libre permite que los programas sean modificados y redistribuidos. Estas prácticas están generalmente prohibidas por la legislación internacional de copyright, que intenta impedir que alteraciones y copias sean efectuadas sin la autorización del o los autores. Las licencias que acompañan al software libre hacen uso de la legislación de copyright para impedir la utilización no autorizada, pero estas licencias definen clara y explícitamente las condiciones bajo las cuales pueden realizarse copias, modificaciones y redistribuciones, con el fin de garantizar las libertades de modificar y redistribuir el software registrado. A esta versión de copyright, se le da el nombre de copyleft.

Debian

La licencia Debian es parte del contrato realizado entre Debian y la comunidad de usuarios de software libre, y se denomina Debian Free Software Guidelines (DFSG). En esencia, esta licencia contiene criterios para la distribución que incluyen, además de la exigencia de publicación del código fuente: (a) la redistribución libre ; (b) el código fuente debe ser incluido y debe poder ser redistribuido; (c) todo trabajo derivado debe poder ser redistribuido bajo la misma licencia del original; (d) puede haber restricciones en cuanto a la redistribución del código fuente, si el original fue modificado; (e) la licencia no puede discriminar a ninguna persona o grupo de personas, así como tampoco ninguna forma de utilización del software; (f) los derechos otorgados no dependen del sitio en el que el software se encuentra; y (g) la licencia no puede 'contaminar' a otro software.

Open Source

La licencia de Open Source Initiative deriva de Debian.

4.1.1.2.Licencias utilizadas

Software Propietario

El Software propietario es aquel cuya copia, redistribución o modificación están, en alguna medida, prohibidos por su propietario. Para usar, copiar o redistribuir, se debe solicitar permiso al propietario o pagar.

Aplicaciones:

- IntelliJ IDEA 12 Premium
- Stash
- Jira
- Bamboo
- Errbit
- Microsoft Office
- Microsoft Windows

GPL

La Licencia Pública General GNU (GNU General Public License GPL) es la licencia que acompaña los paquetes distribuidos por el Proyecto GNU, más una gran varidad de software que incluye el núcleo del sistema operativo Linux. La GPL se basa en la legislación internacional de copyright, lo que debe garantizar cobertura legal para el software licenciado con GPL.

Aplicaciones:

- Ubuntu

- TramperData
- Dia
- StartUML (GPL modificado)

Licencia Apache 2.0

La licencia Apache 2.0 es una licencia de software libre creada por la Apache Software Foundation (ASF), que requiere la conservación del aviso de copyright y el disclaimer, pero no es una licencia copyleft, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.

Aplicaciones:

- Grails
- Librería HtmlUnit
- Plugins de Grails
    - Mail
    - Cache
    - Quartz
    - Resources
    - Calendar

Licencia BSD

Es una licencia de software libre permisiva como la licencia de OpenSSL o la MIT License, no se considera copyleft y tiene restricciones en comparación con otras como la GPL.

Aplicaciones:

- Firebug
- HttpRequester

Licencia MIT

Es una licencia de software libre premisiva originada en el Instituto de Tecnología de Massachusetts (MIT). No es considerada copyleft y no tiene copyright, lo que permite su modificación. Es muy parecida a la licencia BSD en cuanto a efectos.

Aplicaciones:

- Foundation (framework)
- Plugin “Noty”

Licencia PostgreSQL

PostgreSQL utiliza una licencia open source liberal, similar a las licencias BSD o MIT.

Licencia pública de Mozilla (MPL)

Mozilla Firefox utiliza esta licencia de código abierto y software libre.

Creative Attribution 2.5 License

La librería de íconos famfamfam utiliza esta licencia que establece que es necesario incluir en el trabajo un link a la Licencia y mencionar al autor de igual forma. Se declara que ningún cambio ha sido realizado a la librería de iconos durante el desarrollo del proyecto.

4.1.2. Seguridad de los datos

La Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal, (LOPD), es una Ley Orgánica española que tiene por objeto garantizar y proteger las libertades públicas y los derechos fundamentales de personas físicas, como el honor, intimidad y privacidad tanto familiar como personal, en todo lo referente al tratamiento de los datos personales.

Según esta Ley, en esta aplicación se debe garantizar la protección de los datos personales que figuren en todos aquellos elementos relacionados, principalmente aquella información guardada en base de datos. Esta ley afecta a todos los datos que hacen referencia a personas físicas registradas sobre cualquier soporte informático.

En esta aplicación no se mantienen datos personales en base de datos ni en cualquier otro sitio del sistema, más allá del nombre de los usuarios que tomen parte en la gestión del proceso de Release. La información sobre los nombres es almacenada cuidadosamente en base de datos y sólo tiene acceso a dicha información el grupo de gestores de la aplicación pues esta base de datos se encuentra protegida además de ser sólo accesible dentro de la red interna de la empresa donde se despliegue la aplicación.

Para la distribución de emails no se utiliza en ningún momento las direcciones de correo personales o profesionales de los usuarios finales. En lugar de ello, se hace uso de listas de distribución de emails generales. El sitio donde se guarda la información final acerca de los usuarios y sus direcciones de email es de responsabilidad de la empresa donde se despliegue la aplicación.

De todas formas, solamente los empleados de la empresa podrán acceder a esta información privada ya que, como se ha dicho, la aplicación se utilizará de forma interna y es innaccesible desde el exterior. Sería conveniente implementar en cualquier caso un sistema de autenticación que permita a los usuarios acreditar con sus credenciales el derecho de acceso a la información interna, con el fin de reforzar la seguridad de los datos.

4.2.Internacionales

4.2.1. Europa (general)

21 de noviembre del 2001 (Budapest) – Convenio sobre la Ciberdelincuencia

El fin de este convenioes la regulación de todos los delitos relacionados con la Informática y el establecimiento de distintas sanciones así como la especificación de los distintos términos tenidos en cuenta, como “sistema informático” y “datos informáticos”.

En este convenio se tienen en cuenta delitos contra la confidencialidad, integridad y disponibilidad de los datos y sistemas informáticos por distintas vías:

- Acceso ilícito
- Interceptación ilícita
- Ataques a la integridad de datos
- Ataques a la integridad del sistema
- Abuso de dispositivos

Además, se mencionan los distintos tipos de delitos informático. Entre ellos podemos encontrar:

- Falsificación informática

- Fraude informático
- Delitos sobre pornografía infantil
- Delitos relacionados a las infracciones de la propiedad intelectual

Se detallan además las condiciones y normativas aplicables al almacenamiento y distribución de datos informáticos.

4.2.2. Estados Unidos

1994 – Acta Federal de Abuso Computacional, modificada por el Acta de Fraude y Abuso Computacional de 1986

Esta ley está directamente en contra de la transmisión de virus informáticos y proporciona definiciones y sanciones relacionadas con este tema.

En el Estado de California, en 1992 se adoptó la Ley de Privacidad en la que se contemplan los delitos informáticos pero en menor grado que los delitos relacionados con la intimidad que constituyen el objetivo principal de esta Ley.

4.2.3. Alemania

15 de mayo de 1986 – Segunda Ley contra la Criminialidad Económica

Esta Ley reforma el Código Penal para incluir sanciones contra los siguientes delitos:

- Estafa informática
- Espionaje, alteración y falsificación de datos
- Sabotaje informático en general.
- Destrucción de datos de especial significado, así como la tentativa de realizarlo.
- Cualquier acción informática que concluya en perjuicio patrimonial para terceros.
- Utilización abusiva de cheques o tarjetas de crédito.

4.2.4. Austria

22 de diciembre de 1987 - Ley de reforma del Código Penal

Esta Ley incluye sanciones para aquellos que causen perjuicio patrimonial a un tercero a través de la manipulación de datos automáticos (estafa informática), contemplando sanciones para quienes incurren en este acto utilizando su propia profesión. Además, se especifican sanciones relacionadas con la destrucción de datos, tanto personales como no personales y los referentes a programas.

4.2.5. Reino Unido de Gran Bretaña e Irlanda del Norte

1991 – Ley de Abusos Informáticos (Computer Misuse Act)

Surge a raíz de un caso de hacking en 1991 y penaliza los intentos de alteración de datos informáticos, independientemente de su éxito, y además la modificación de datos sin autorización donde se incluyen los virus.

4.2.6. Holanda

1 de marzo de 1993 – Ley de Delitos Informáticos

Sanciona los delitos relacionados con el hacking, preacking (utilización de servicios de telecomunicaciones evitando el pago en cualquier medida del servicio), ingeniería social con malas intenciones y la distribución de virus.

4.2.7. Francia

5 de enero de 1988 – Ley número 88-19 sobre el fraude informático

Se sancionan delitos relacionados con:

- Acceso fraudulento a un sistema de elaboración de datos.
- Sabotaje informático
- Destrucción de datos
- Falsificación de documentos informatizados
- Uso de documentos informatizado falsos

4.2.8. Venezuela

2001 – Ley Especial contra los delitos informáticos

En esta Ley se tienen en cuenta los siguientes delitos:

- Delitos contra los sistemas que utilizan tecnologías de información.
- Delitos contra la propiedad.
- Delitos contra la privavidad de las personas y las comunicaciones.
- Delitos contra niños, niñas o adolescentes.
- Delitos contra el orden económico.

5. Capítulo 5: Herramientas, estándares, metodologías y tecnologías

5.1. UML

El Lenguaje Unificado de Modelado es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidadEs un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

En este proyecto se ha hecho gran uso de esta herramienta para describir el sistema y sus funcionalidades en las etapas más tempanas del proyecto, para documentar el propio sistema y su arquitectura y para mostrar de forma visual cualquier característica, funcionalidad o mecanismo que necesitara clarificación o negociación con los stakeholders.

5.2. Groovy

Groovy es un lenguaje de programación orientado a objetos implementado sobre la plataforma Java. Tiene características similares a Python, Ruby, Perl y Smalltalk. La especificación JSR 241 se encarga de su estandarización para una futura inclusión como componente oficial de la plataforma Java.

Groovy usa una sintaxis muy parecida a Java, comparte el mismo modelo de objetos, de hilos y de seguridad. Desde Groovy se puede acceder directamente a todas las API existentes en Java. El bytecode generado en el proceso de compilación es totalmente compatible con el generado por el lenguaje Java para la Java Virtual Machine (JVM), por tanto puede usarse directamente en cualquier aplicación Java. Todo lo anterior unido a que la mayor parte de código escrito en Java es totalmente válido en Groovy hacen que este lenguaje sea de muy fácil adopción para programadores Java; la curva de aprendizaje se reduce mucho en comparación con otros lenguajes que generan bytecode para la JVM, tales como Jython o JRuby.

Groovy 1.0 apareció el 2 de enero de 2007. Después de varias versiones beta, el 7 de diciembre de 2007 apareció la versión Groovy 1.1 que finalmente fue renombrada a Groovy 1.5 con el fin de notar la gran cantidad de cambios que ha sufrido con respecto a la versión 1.0. En diciembre de 2009 se publicó la versión 1.7.

"Groovy is like a super version of Java. It can leverage Java's enterprise capabilities but also has cool productivity features like closures, builders and dynamic typing. If you are a developer, tester or script guru, you have to love Groovy." – Groovy founders

Por toda la potencia que ofrece, la facilidad de aprendizaje que supone el adpotar este lenguaje siendo fundamentalmente una programadora de Java y la estrecha relación que tiene Groovy con el framework web que se quiere adpotar, este ha sido el lenguaje de programación seleccionado para desarrollar el proyecto. En este proyecto se trabaja con la versión 2.0 de Groovy ya que es la versión incluída en la versión utilizada del framework de desarrollo web. Esta versión incluye, entre otras cosas, características estáticas para el lenguaje con comprobación de tipos estáticos y compilación estática.

5.3.Grails

5.3.1. Historia y Descripción del framework

Grails es un framework para aplicaciones web libre desarrollado sobre el lenguaje de programación Groovy (el cual a su vez se basa en plataforma Java). Grails pretende ser un marco de trabajo altamente productivo siguiendo paradigmas tales como convención sobre configuración o no te repitas (DRY), proporcionando un entorno de desarrollo estandarizado y ocultando gran parte de los detalles de configuración al programador.

Grails ha sido impulsado principalmente por la empresa G2One, la cual fue adquirida por la desarrolladora de software libre SpringSource en noviembre de 2008 y en agosto de 2009 SpringSource fue a su vez adquirida por VMWare, empresa especializada en virtualización de sistemas.

Grails fue conocido como 'Groovy on Rails' (el nombre cambió en respuesta a la petición de David Heinemeier Hansson, fundador de Ruby on Rails). Se inició en julio de 2005, con la versión 0.1. En este momento, la última versión estable es la 2.3.7, sacada al mercado el 18 de febrero del 2014.

Características:

- Framework web de alta productividad para la plataforma Java.
- Reutiliza tecnologías Java ya probadas como Hibernate y Spring bajo una interfaz simple y consistente.
- Ofrece un framework consistente que reduce la confusión y que es fácil de aprender.
- Ofrece documentación para las partes del framework relevantes para sus usuarios.
- Proporciona lo que los usuarios necesitan en áreas que a menudo son complejas e inconsistentes:
    - Framework de persistencia potente y consistente.
    - Patrones de visualización potentes y fáciles de usar con GSP (Groovy Server Pages).
    - Bibliotecas de etiquetas dinámicas para crear fácilmente componentes web.
    - Buen soporte de Ajax que es fácil de extender y personalizar.
- Proporciona un entorno de desarrollo orientado a pruebas.
- Proporciona un entorno completo de desarrollo, incluyendo un servidor web y recarga automática de recursos.

Grails se ha diseñado para ser fácil de aprender, fácil para desarrollar aplicaciones y extensible. Intenta ofrecer el equilibrio adecuado entre consistencia y funcionalidades potentes.

Grails es un framwork diseñado para ofrecer potencia a medida que hace la vida del programador más sencilla y agradable. Al trabajar con el concepto de “Convención sobre Configuración”, este framework ofrece la posibilidad de generar una aplicación que funcione con muy poco esfuerzo y configuración y desde los primeros momentos de la creación. La intexistencia de configuración XML es un gran punto a favor de esta rapidez de iniciación que Grails garantiza. Además, Grails tiene un servidor web integrado preparado para desplegar la

aplicación desde el primer momento y todas las librerías requeridas son parte de la distribución de Grails y están preparadas para ser desplegadas automáticamente.

Grails proporciona método dinámicos basándose en el tipo de clase. Por ejemplo, la clases de dominio tienen métodos para automatizar operaciones de persistencia, como save para salvar, delete para borrar y find para buscar.

5.3.2. MVC

El Modelo Vista Controlador es un patrón arquitectónico de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Los tres componentes son el modelo, la vista y el controlador, siendo el primero el encargado de la representación de la información, el segundo se ocupa de la interacción con el usuario y, finalemente, el controlador se encarga de llevar a cabo toda la lógica de negocio del sistema.

Este patrón de diseño se aplica incesantemente en el desarrollo de softwares pues se basa en ideas de reutilización de código y separación de conceptos, lo cual concluye en buenas prácticas de desarrollo y un código organizado con menos probabilidades de errores y más fácil de mantener.

Muchas empresas aplican el patrón MVC a sus proyectos, sin embargo, este framework web se ha diseñado sobre el paradigma Modelo Vista Controlador y lo incluye de forma nativa.

MVC en Grails

Grails define una estructura de Proyecto que lo divide claramente en distintos tipos de elementos para seguir el diseño MVC y aplicar una gran cantidad de convenciones. Por ejemplo, los objetos colocados en la carpeta “domain” son mapeados automáticamente a base de datos y los servicios (todos aquellas clases que concluyan con dicha palabra) son inyectados automáticamente por Spring en cualquier otra clase de tipo servicio. Por lo tanto, el programador no se tiene que preocupar en este punto por temas como la inyección de dependencias, pues con Grails esto viene manejado por el propio framework.

- `grails-app` - top level directory for Groovy sources
  - `conf` - Configuration sources.
  - `controllers` - Web controllers - The C in MVC.
  - `domain` - The application domain.
  - `i18n` - Support for internationalization (i18n).
  - `services` - The service layer.
  - `taglib` - Tag libraries.
  - `utils` - Grails specific utilities.
  - `views` - Groovy Server Pages - The V in MVC.
- `scripts` - Gant scripts.
- `src` - Supporting sources
  - `groovy` - Other Groovy sources
  - `java` - Other Java sources
- `test` - Unit and integration tests.

Ilustración 1. Grails - Estructura del proyecto por defecto

Controladores

Grails usa controladores para implementar el comportamiento de las páginas web

Vistas

Grails soporta JSP y GSP y proporciona un gran número de tag libraries. De todas maneras se pueden crear y reutilizar librerías de tags fácilmente.

Persistencia en el Modelo

El modelo de datos en Grails se guarda en la base de datos utilizando GORM (Grails Object Relational Mapping). Grails se encarga de la gestión de la base de datos en todo momento, con lo cual solamente es necesario centrarse en el diseño del dominio de la aplicación y dejar todo lo relacionado con la base de datos a Grails. El mecanismo de persistencia de GORM está implementado mediante Hibernate. De todas formas, también es posible ejecutar sentencias HQL para acceder a base de datos y evitar el uso de GORM.

HQL es el potente lenguaje de consulta de Hibernate, que se parece a SQL pero, en comparación, HQL es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. En Grails es totalmente posible realizar consultas a base de datos haciendo uso de esta sintaxis.

Grails soporta scaffolding para implementar operaciones CRUD (Create, Read, Update, Delete) sobre clases de dominio, lo cual influye directa y positivamente en la rapidez con la que se puede generar una aplicación particular.

5.3.3. Notas finales

Para trabajar con un framework que tiene tantas convenciones por defecto es necesario, evidentemente, tener conocimiento acerca de dichas convenciones y el funcionamiento de la herramienta. Para ello se ha hecho uso del libro oficial de Grails y la documentación técnica online.

Este framework ha sido seleccionado debido a su gran potencia, facilidad de uso, agilidad, las útiles herramientas y conceptos que incorpora por defecto y porque es uno de los frameworks más actuales que sigue las últimas tendencias tecnológicas. Además, al haber tenido experiencia en su utilización puedo ahorrar todo el tiempo que conlleva realizar una introducción a un nuevo framework de desarrollo y leer toda la documentación necesaria para trabajar correctamente con la herramienta y sacar todo el provecho posible.

La versión de Grails utilizada en este proyecto es la 2.2.4, que incluye la distribución 2.0 de Groovy.

5.4.Metodologías de desarrollo ágil

Este proyecto se ha desarrollado bajo un enfoque completamente ágil, combinando diferentes técnicas de distintas metodologías ágiles de desarrollo software e incorporando en el proceso nuevas técnicas nacidas de la experiencia y el criterio personal. En este trabajo se ha dedico un capítulo entero a describir con detalle cada proceso y rasgo de la **Metodología Propia** de Desarrollo utilizada.

Es necesario remarcar que como principal fuente de apoyo se ha utilizado la metodología SCRUM y sus innovadores principios. Esta ha sido la metodología base debido a la experiencia adquirida en el desarrollo de la misma y por su reconocida eficiencia y revolucionarias ideas.

Además, se ha incorporado elementos de XP tales como el intentar proporcionar código que funcione, pensado solamente para los requisitos actuales y en el futuro hacer refactorizaciones en caso de que fuera necesario. De esta forma, no se pierde tiempo en el momento actual pensando en futuros escensarios que probablemente ni siquiera lleguen a ocurrir.

5.4.1. SCRUM

A principios de 1990, Ken Schwaber usó en su compañía lo que se volvería Scrum, los Métodos Avanzados de Desarrollo, y Jeff Sutherland, junto a John Scumniotales y Jeff McKenna, desarrolló un enfoque similar en la Corporación Easel, y fueron los primeros en referirse a este con una única palabra, Scrum. En 1995, Sutherland y Schwaber presentaron juntos un trabajo describiendo la metodología Scrum por primera vez. Schwaber y Sutherland colaboraron los siguientes años para combinar los escritos anteriores, sus experiencias y las mejores prácticas de la industria en lo que hoy se conoce como Scrum.

Las características más marcadas que se logran notar en Scrum serían: gestión regular de las expectativas del cliente, resultados anticipados, flexibilidad y adaptación, retorno de inversión, mitigación de riesgos, productividad y calidad, alineamiento entre cliente y equipo, por último equipo motivado.

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.

Scrum es una Metodología ágil enfocada al desarrollo de un producto en iteraciones y al buen trabajo en equipo. Ofrece la posibilidad de trabajar de forma organizada, manteniendo la

comunicación entre todas las personas interesadas (roles de Scrum) y permite a los equipos responder a los "feedbacks" tempranos y los cambios en los requisitos sin que tengan demasiado impacto en la gestión y planificación del proyecto, para construir exactamente lo que el usuario final necesita.

En este proyecto no se ha podido aplicar todos los elementos de Scrum ya que no ha habido un equipo de desarrollo. Sin embargo, algunos elementos han sido aplicables y se enumeran a continuación:

- Desarrollo en iteraciones
- Desarrollo basado en Sprints: con planifcación de Sprint, seguimiento y estimación parcial de las distintas tareas a desarrollar.
- Backlog de tareas: Se ha mantenido un Backlog donde se encuentran todos los requisitos a desarrollar y cada vez que se ha querido incluir una nueva característica, esta ha sido añadida a la lista para ser planificada e incluída en un Sprint, según propone SCRUM.
- El Backlog de tareas y las tareas propias del Sprint están priorizadas de acuerdo a urgencia, dependencias o valor para el usuario.
- Se mantienen las Daily-Meetings para mantener al resto de personas implicadas en el Proyecto al tanto de los progresos.
- Se mantienen las Retrospective-Meetings para evaluar la productividad del Sprint, los objetivos alcanzados y visualizar los futuros, así como para recibir feedbacks por parte de las personas interesadas en el proyecto.
- Se utilizan elementos visuales de ayuda como post-its y una pizarra para la organización del Proyecto general y del Sprint vigente (ver anexos).

5.4.2. Extreme Programming (XP)

La programación extrema o eXtreme Programming (XP) es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Su enfoque es más bien práctico por lo que las buenas prácticas que ofrece suelen usarse en conjunto con otras metodologías enfocadas a la gestión del proyecto.

XP tiene 5 principios fundamentales: simplicidad, comunicación, feedback, coraje y respeto. Algunos se han aplicado en la medida en que han sido necesarios y/o posibles:

- Se aplica simplicidad sobre el código al seguir el principio KISS (Keep it simple and stupid). Un código simple ofrece una gran ventaja a la hora de compartir código, comunicarnos con otros desarrolladores o simplemente realizar cambios futuros o refactorizaciones. Además, un código simple, limpio y autodocumentado es la mejor forma de documentación y comunicación dentro de un equipo de desarrollo.
- Se entiende que en gran medida la simplicidad y la comunicación son principios complementarios, de acuerdo con lo que propone esta metodología.

- Se aplica la utilización de feedbacks al generar iteraciones en poco tiempo y mostrar a los usuarios los avances luego de la generación de cada iteración. Esto es especialmente útil para detectar errores o aclarar malos entendidos antes de volver a caer en el mismo problema en un requisto futuro.
- Se aplica coraje al intentar diseñar y programar para hoy y no para mañana. Esto significa que nos centramos en el problema actual y el requisito que queremos implementar, sin tener en cuenta distintos escenarios y/o requisitos posibles que podrían aparecer en el futuro. Se genera un código flexible capaz de amortiguar cambios, sin embargo se diseña y programa para el problema actual pues esos posibles requisitos del futuro tal vez ni siquiera lleguen a aparecer. “La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo”.Otro ejemplo de la aplicación de valentía es cuando debemos refactorizar partes del programa para ofrecer mayor flexibilidad o simplificar el código. En este proyecto, cada vez que se ha detectado que una parte de la arquitectura o código podría ser mejorada, se ha realizado una refactorización para mejorar la comunicación y legibilidad del código, así como su diseño.

5.4.3. Proceso Unificado de Desarrollo (PUD)

El Proceso Unificado de Desarrollo Software o simplemente Proceso Unificado es un marco de desarrollo de software que se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental.

Las disciplinas de esta metodología recuerdan al ciclo de vida clásico de la displina “cascada”. Sin embargo, se han usado algunos elementos del PUD (no todos) como ayuda para este proyecto y sin entrar demasiado en la metodología del mismo.

Entre los elementos usados encontramos la utilización del modelado de casos de usos para la extracción de requisitos funcionales y el uso del Lenguaje Unificado de Modelado (UML) para definir los distintos modelos del sistema y realizar el diseño inicial de la arquitectura.

UML se ha aplicado principalmente porque ayuda a visualizar el sistema en toda su extensión y a comprender ciertos comportamientos o características del dominio que podrían pasar desapercibidas. En otras palabras, UML se ha utilizado principalmente para la comprensión del dominio del Proyecto y su posterior diseño a grandes rasgos. Además, los diagramas generados se utilizarán como parte de la documentación del proyecto ya que describen el sistema, se pueden utilizar para una apropiada comunicación entre stakeholders y desmuestran la aplicación de uno de los principios de la Ingeniería: el modelado de sistemas.

En este punto es necesario aclarar que los diagramas UML han sido utilizados en beneficio propio, siempre que fueran útiles para conseguir expresar una idea o dominio, o simplemente como medio de comunicación. Algunos diagramas no siguen los criterios estrictos establecidos de elaboración, pues se siguen criterios de simplicidad y usabilidad: si cumple con su cometido lo consideramos válido. [no sé si dejar esto]

El PUD consta de 4 fases de desarrollo: Inicio, Elaboración, Construcción y Transición. En este proyecto se ha aplicado los elementos de la fase de Elaboración en su mayoría, sin embargo a

partir de ese punto se continúa con otro enfoque más cercano a metodologías de desarrollo como SCRUM. Las tareas básicas de la fase de Inicio van enfocadas a definir el negocio: facilidad de realizar el proyecto, presentación de un modelo, visión, metas, deseos del usuario, plazos, costos y viabilidad.

5.4.4. Agile Modeling (AM)

Se aplican algunos aspectos del Modelado Ágil como:

- La creación de modelos simples, fáciles de crear y mantener a lo largo de la vida del software.
- La creación de modelos que cumplan un objetivo pre-establecido y tengan una función específica.
- La creación de distintos tipos de modelos para tener una mejor visión del sistema en su totalidad.
- Lla exploración de técnicas de modelado en proyectos software a través de enfoques ágiles como XP y SCRUM.
- La exploración de la mejora del modelado bajo procesos prescriptivos como PUD donde el modelado es realmente extenso y muchas veces no responde a un objetivo funcional y de valor para el equipo de desarrollo (falta de agilidad).
- La demostración de la posibilidad de realizar un modelado ligero efectivo.

5.5.IntelliJ – IDE de desarrollo

IntelliJ IDEA es una IDE Java creada por JetBrains y disponible en versiones comerciales y gratuitas bajo la licencia de Apache 2. La primera versión de IntelliJ IDEA fue desplegada en enero de 2001, y en su momento fue uno de las primeros IDE de Java disponibles con capacidades integradas de refactorización de código y navegación avanzada. IntelliJ ofrece un gran set de características e integración con las tecnologías y frameworks modernos más importantes para empresas y desarrollo web.

La herramienta ofrece muchísimas facilidades al programador, es realmente fácil de utilizar y soporta una gran variedad de lenguajes, tecnologías, servidores y sistemas de control de revisión y versiones de software. Además de su potencia y el ahorro de tiempo que supone el uso de esta herramienta debido a sus características, el principal motivo de selección es que, en su versión comercial, incluye el framework de desarrollo web “Grails” de forma nativa, lo cual es una ventaja sobre otras IDEs donde sea necesario importar Grails como un plugin externo, ya que el riesgo de errores e incompatibilidades con ciertos elementos existentes en la herramienta y otros plugins a integrar se dispara.

Facilidades:

- La IDE reconoce tu código y forma de trabajar y ayuda proporcionando relevantes e inteligentes sugerencias en todos los contextos.
- Se automatizan tareas de desarrollo tediosas y repetitivas para que los desarrolladores se mantengan concentrados en lo que realmente importa y la productividad final sea elevada.
- Integra muchísimas herramientas útiles como sistemas de control de versiones integrados y una gran variedad de lenguajes de programación.

Características:

- Frameworks de pruebas unitarias con agradable interfaz y cobertura de código.
- Incluye sistemas de control de versiones populares con una interfaz agradable y visual para cada una de sus funcionalidades.
- Inluye un diseñador UML.
- Incluye tecnologías avanzadas incluyendo Java EE, Spring, GWT, Struts, Play, Grails, Hibernate, Google App Engine y OSGi.
- Herramientas de despliegue y depuración para la mayoría de servidores de aplicación.
- Desarrollo web simplificado con editores inteligentes para HTML, CSS y otros lenguajes y tecnologías como Sass, LESS, TypeScript, CoffeeScript y Node.js.
- Soporte extenso para PHP, Ruby on Rails y Python/Django.

5.6.PostgreSQL

Se ha seleccionado PostgreSQL como sistema de gestión de base de datos por sus amplias características, potencia, fiabilidad, por la buena documentación ofrecida y porque es un producto que además puede ser utilizado de forma gratuita.

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional con código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y ofrece características a la altura de otras bases de datos comerciales. Utilizado un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema, lo cual constituye un gran punto a su favor pues a la hora de desarrollar software con acceso a base da datos, la estabilidad, consistencia y disponibilidad de la base de datos son puntos cruciales que deben ser garantizados.

Es una de las bases de datos más potentes y robustas del mercado gracias a sus características técnicas: estabilidad, robustez, facilidad de administración e implementación de estándares. Además, funciona bien para trabajar con alta cantidad de datos y concurrencia de usuarios.

A lo citado anteriormente hay que añadir la gran cantidad de opciones de programación y desarrollo que ofrece y la riqueza en todo lo referente a SQL.

En este proyecto se ha creado una base de datos en PosgreSQL para almacenar toda la información persistente del sistema. Para ello se han realizado las configuraciones pertinentes en el proyecto Grails, por medio de la IDE IntelliJ y haciendo uso del driver JDBC para realizar la conexión entre el software y la base de datos. Los pasos para realizar la configuración serán discutidos en el capítulo sobre las Configuraciones básicas del proyecto.

5.7.GIT – Stash

Durante la realización de un gran proyecto es necesario llevar a cabo un control de versions para mantener bajo supervisión todos los cambios realizados a lo largo del período de implementación. Si además el proyecto va a ser gestionado bajo metodologías de desarrollo ágil iterativas, cobra especial importancia el hecho de tener la posibilidad de controlar cada versión correspondiente a una iteración, con sus nuevas características y cambios realizados al código y configuraciones existentes.

Git es un software de control de versiones diseñado por Linus Torvalds, libre y de código abierto, diseñado para manejar todo desde pequeños hasta realemente grandes proyectos de forma rápida y eficiente. Es una maravillosa herramienta que facilita el trabajo el equipo y ayuda a realizar un control sobre las distintas iteraciones y/o versiones del producto software a lo largo del proceso de implementación y mantenimiento. Gracias a ella se puede organizar cada cambio realizado en el código y las configuraciones mediante comentarios, fechas, identificación de autores, etc., proporcionando un completo control sobre las distintas etapas por las que pasa nuestro producto y permitiendo además la vuelta a un estado anterior de la aplicación en cualquier momento, o, simplemente, el uso de múltiples versiones del mismo sistema software al mismo tiempo.

Como este proyecto ha sido realizado de manera individual, no se ha podido obtener un beneficio de las increíbles ventajas que esta herramienta ofrece para trabajar en equipo y desarrollar distintas características en versiones diferentes para luego proceder a la integración de todas ellas en una única versión general. Sin embargo, se han utilizado otras características que han contribuido a un mejor y más organizado desarrollo de la aplicación. Estas características son, principalmente, el uso de toda la información descrita anteriormente que se proporciona cada vez que se realiza un cambio en el código o las configuraciones, el uso de las áreas de staging para deshacer cambios de forma temporal y, especialmente, el uso del sistema de ramas (branching) que proporciona GIT.

Este sistema de branching ha hecho posible la implementación de cada grupo de requisitos en una rama distinta, que permite testear y probar la aplicación en cierto modo de isolación sin permitir que los cambios introducidos afecten a aquellos que se hayan declarado como estables. La estrategia de branching que se ha seguido se describe en el capítulo Configuraciones del proyecto.

Entre las características más relevantes se encuentran:

- Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal.
- Gestión distribuida. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.
- Los repositorios Subversion y svk se pueden usar directamente con git-svn.
- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.
- Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio (denominado autenticación criptográfica de historial).
- Los renombrados se trabajan basándose en similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre con base en supuestos nombres únicos de nodos de sistema de ficheros, lo que evita

posibles, y posiblemente desastrosas, coincidencias de ficheros diferentes en un único nombre.

5.8.jQuery

Esta es una rápida, pequeña y enriquecida librería de JavaScript que permite que la manipulación de documentos HTML, manejo de eventos, animaciones y llamadas Ajax sean mucho más simples mediante una API fácil de usar que es funcional en multitud de navegadores. Gracias a su versatilidad y extensibilidad, jQuery ha ido cambiando a través de los años el modo en que las personas escriben JavaScript

Esta librería viene incluida en Grails por defecto como parte de la gran cantidad de herramientas que lo componen. Es necesario dedicar una sección de este documento a jQuery debido a la gran importancia que ha tenido dentro de la realización del proyecto y al hecho de que es sin duda una de las grandes librerías Frontend de nuestros días, que aporta grandes ventajas en términos tecnológicos a cualquier proyecto al que sea aplicada.

Todas las funcionalidades de Javascript implementadas en este proyecto han sido realizadas por medio de jQuery debido a la gran simplicidad y facilidad de uso que proporciona y a su vez a la potencia de las funciones ofrecidas. En especial, se he hecho uso de las funciones relacionadas con AJAX.

5.9.Foundation

Este Frontend Framework fue encontrado por recomendación directa de un experto Frontend. Además, ofrece muchísimas ventajas sobre otros frameworks web (por ejemplo, ofrece un diseño “responsive”) y trabaja con las últimas tecnologías web a la vez que se ofrece un diseño simple, mínimo y ligero que se adapta perfectamente a las necesidades del proyecto.

Con este framework se pretende generar una Interfaz de Usuario de forma sencilla y rápida, con un diseño y template previamente establecidos por el propio framework y a su vez hacer uso de todas las funcionalidades de javascript que viene por defecto ofrecidas.

Por supuesto, toda funcionalidad o estilo extra deseado será añadido en el propio proyecto para alcanzar de esta forma la Interfaz de Usuario pensada y reflejada en los Sketches generados para la aplicación y cumplir de esta manera con las expectativas de los usuario finales.

5.9.1. Breve historia

Foundation fue creado como un proyecto ZURB para desarrollar código frontend más rápido y mejor. En octubre del 2011 se despliega Foundation 2.0 como proyecto open-source bajo la licencia MIT. En junio del 2012 se despliega Foundation 3.0 y en febrero del 2013 Foundation 4.0. Finalmente, en noviembre del 2013 se saca al mercado la versión actual, utilizada para el desarrollo de este proyecto: Foundation 5.0.

Desde la versión 2.0 Foundation soporta el diseño “responsive”, lo que significa que el diseño gráfico de las webs se ajusta dinámicamente teniendo en consideración las características del dispositivo usado (pc, table, móvil). Este fue el primer framework frontend “responsive” y se mantiene como tal en la actualidad.

5.9.2. Características principales

En su página oficial, Foundation se declara “The most advanced responsive front-end framework in the world” (El más avanzado framework frontend “responsive” en el mundo).

Además de ser un framework “responsive” pensado para implementar una correcta visualización de la UI de la aplicación en todo tipo de dispositivos y dar soporte a los dispositivos móviles, se encuentra las siguientes características:

- Altamente customizable
- Rápido y ligero
- Bien documentado con la inclusión de ejemplos prácticos en la web
- Fácil de comprender y utilizar
- Aplicable a cualquier tecnología back-end
- Contiene elementos y diseños sencillos y bonitos, con un estilo moderno.

Además de ofrecer estilos, ofrece distintos templates listos para incluir en las vistas existentes en el proyecto y una gran variedad de funcionalidades útiles e interesantes basadas en javascript, perfectamente documentadas y listas para utilizar de forma sencilla.

Por supuesto es necesario destacar que, a pesar de la gran variedad de beneficios y tecnologías que aporta este framework y lo convierten en una potente herramienta que cualquier desarrollador desearía utilizar, este framework ofrece el 100% de sus características a los usuarios de forma gratuita y se mantiene como producto open-source.

5.10. Otros Plugins

5.10.1. Noty plugin

Noty es un plugin de javascript sencillo y útil que ofrece una gran variedad de formas para mostrar notificaciones al usuario. Por su diversidad, la facilidad de integración en el proyecto y la gran vida que aporta a la aplicación, ha sido seleccionado como el plugin a utilizar para realizar todas las notificaciones de la aplicación.

Hay que destacar que Foundation también incluye un sistema de notificaciones, sin embargo dichas notificaciones son muy básicas y el framework no ofrece una gran variedad de colores, formatos o sitios de aparición de la notificación. Además, usando este plugin el usuario puede interactuar con la notificación recibida y es posible incluse desencadenar funciones javascript a partir de dicha interacción. Todo esto convierte a Noty en un plugin flexible, que ofrece muchas posibilidades de configuración y que aporta una gran dinámica y vida al proyecto.

5.10.2. Mail plugin

Este plugin de Grails ofrece de forma sencilla y eficaz las funcionalidades necesarias para enviar emails por medio de la aplicación. Se trata de un plugin sencillo, bien documentado con ejemplos prácticos en el sitio oficial de Grails, que encapsula la parte más técnica y complicada del procedimiento de envío de emails y ofrece a los desarrolladores una interfaz de uso simplificada y comprensible.

Es un plugin extremadamente sencillo de utilizar para el cual es necesario solamente hacer unas simples configuraciones y permite el envío de correos utilizando el servidor que se ha

indicado en la configuración. Dicha configuración debe ser realizada en el fichero "Config.groovy" del proyecto Grails.

```
grails {
   mail {
     host = "smtp.live.com"
     port = 587
     username = "youracount@live.com"
     password = "yourpassword"
     props = ["mail.smtp.starttls.enable":"true",
                  "mail.smtp.port":"587"]
  }
}
```

Ilustración 2. Mail plugin - Configuración

Este plugin añade un método llamado "sendMail" por defecto a todos los controladores del proyecto y añade un nuevo servicio llamado "MailService" que contiene el mismo método y puede ser utilizado desde cualquier servicio de la aplicación, haciendo uso de la inyección de servicios automática ofrecida por Grails / Spring. El método "sendMail" es realmente básico pero al mismo tiempo ofrece una gran potencia pues permite incluir no solamente los elementos bases mostrados en la figura, sino copias CC, CCO, documentos adjuntos y el texto puede ser enviado como código html, con un formato mucho más atractivo.

Ejemplo de utilización:

```
sendMail {
  to "fred@g2one.com"
  subject "Hello Fred"
  body 'How are you?'
}
```

Ilustración 3. Mail plugin - Utilización

5.10.3. Calendar plugin

Otro plugin de Grails que ofrece todas las funcionalidades para trabajar con un calendario desde la Interfaz de Usuario de una forma más agradable y moderna. Grails incorpora por defecto la utilización de selectores de fecha desde los elementos de vista, sin embargo dichos selectores tienen un estilo primitivo, poco agradable y definitivamente no cumplen con los criterios de usabilidad que queremos mantener en esta aplicación. Por ello se ha decidido utilizar una herramienta que permita el uso efectivo de calendarios de un modo agradable y, por su sencillez, facilidad de uso y buena documentación, Calendar ha sido seleccionado para formar parte de este proyecto.

Calendar es un plugin ligero, customizable y multiplataforma, basado en jscalendar y sin dependencias con librerías JavaScript externas. Permite incluir seleccionadores de fechas en las vistas para una sencilla selección desde la Interfaz de Usuario.

5.10.4. Famfamfam plugin

Famfamfam es otro plugin de Grails que recoge los famosos iconos "famfamfam" para su incorporación directa a cualquier proyecto grails. Este plugin ofrece un medio único de acceso a los iconos y además una tag para incluir cualquier icono de la lista ofrecida orginalmente por famfamfam en las distintas vistas de la aplicación.

6. Capítulo 6: Gestión y planificación del proyecto

Este capítulo está destino a mostrar y analizar todas las decisiones y estrategias tomadas desde el punto de vista de gestión ("management") del proyecto, lo cual incluye por supuesto la planificación y división en distintas etapas del mismo.

Este proyecto se ha desarrollado con la aplicación de Metologías de Desarrollo Ágil. Concretamente se utilizan elementos del PUD y SCRUM según sean convenientes para el correcto y eficiente desarrollo del proyecto. Se ha mezclado diversas estrategias con el fin de adaptarlas a la naturaleza del Proyecto y utilizarlas en beneficio propio, creando una nueva forma de trabajo, proceso o metodología a medida que se avanzaba. La ideología a seguir es ofrecer flexibilidad y calidad mientras nos mantenemos abiertos a amortiguar posibles cambios durante el tiempo de ejecución y controlar los riesgos y principalmente mantenernos dentro del presupuesto de tiempo asignado, asegurando la creación de al menos un grupo básico de requisitos.

Para conseguir los objetivos globales anteriormente descritos es necesario llevar a cabo una correcta planificación y priorización de las tareas a realizar. Por ello, los primeros pasos en el comienzo de este proyecto han sido la creación de un plan de sprints, con deadlines y objetivos por periodo de tiempo y la especificación de manera global de los distintos requisitos del sistema, funcionales y no funcionales, manteniendo siempre un enfoque ágil sobre los mismos. Estos 2 elementos serán claves durante todo el proceso de desarrollo.

A pesar de que habitualmente las Metodologías de Desarrollo Ágil no siguen un plan, para este proyecto se ha diseñado uno que describe los pasos a seguir de manera global, o sea, se realiza una macro-planificación. Internamente, cada Sprint y grupo de Requisitos es planificado de forma ágil y el momento de la planificación corresponde a fechas próximas al comienzo de dicho Sprint. Para mantenernos flexibles podemos planificar de forma detallada las próximas iteraciones pero no todas las que corresponden al proyecto.

6.1. Plan de Sprints

¿Por qué un plan de Spints? ¿En qué nos beneficia?

Un plan de Sprints ayuda a planificar el trabajo global, ofreciendo una idea general de las tareas a realizar y del tiempo disponible. La estimación de la cantidad de tareas a incluir en cada Sprint no está perfectamente estimada: en algunos casos puede ser que no todas las tareas puedan ser realizadas y en otras ocasiones sobrará tiempo, pues en este punto no estamos estimando de forma certera y tampoco estamos interesados en ello.

La duración de los Sprints no tiene que ser siempre la misma aunque ello vaya en contra de los principios de SCRUM. Como se ha dicho, las metodologías se utilizarán en nuestro beneficio y serán amoldadas a nuestras necesidades, creando un proceso de desarrollo totalmente ágil y flexible. La duración de los Sprints debe estar establecida de forma "lógica" por los objetivos a conseguir. Por ejemplo, en las primeras semanas estaremos planificando el proyecto y generando muchísimos artefactos (como los Modelos UML). Es lógico entonces pensar que un Sprint puede estar dedicado a las tareas del **Dominio** y otro a las tareas de la **Arquitectura** del

Sistema. Sin embargo, a la hora de empezar a incorporar funcionalidades en la aplicación, los Sprints pueden tener una mayor duración para incluir en cada uno, por ejemplo, un grupo de requisitos.

Un plan de Sprints es útil principalmente en 2 aspectos:

Organización de las tareas semanales / por Sprint

Útil para la organización de los desarrolladores en cada Sprint. Es realmente sencillo saber **qué tareas** se han de realizar en la semana/Sprint actual y **cuáles son los objetivos** a alcanzar para el final de la jornada.

Gestión del Proyecto

Terminada una jornada de trabajo (Sprint), se puede analizar qué tareas se han realizado y cuántos de los objetivos pre-establecidos en el Plan de Sprints se han alcanzado. Con esta información, es posible realizar un análisis crítico del progreso y observar si todo marcha acorde al plan establecido o si hay riesgos de salirse del plan. Los riesgos deben ser analizados observando no solamente un Sprint. Dado que la estimación de las tareas no es certera, pudiera darse el caso de que en un Sprint no todos los objetivos sean alcanzado, sin embargo en el siguiente alcancemos todos los que quedaban más los propio del Sprint. De esta forma, al final del segundo Sprint seguiríamos estando dentro del plan establecido aunque haya habido una alarma de riesgos en el camino.

El no alcanzar los objetivos de un Sprint es una **temprana alarma** para el manager del Proyecto que debe ser inspeccionada. Esto es especialmente útil para realizar una re-planificación del Proyecto y tareas a realizar en una etapa temprana del desarrollo y mantenernos de esta forma dentro del presupuesto de tiempo.

6.2.Lista de Requisitos

La lista de Requisitos debe contener todas las posibles funcionalidades y aspectos a desarrollar y tener en cuenta durante el Proyecto. Una lista completa nos dará una visión global acerca del tamaño de la aplicación a desarrollar, complejidad, riesgos y definitivamente es un punto clave para especificar los objetivos de cada Sprint en el Plan citado anteriormente.

La lista de Requisitos ha sido dividida en grupos distintos, atendiendo a criterios de prioridad y nivel de importancia para la aplicación. Esto es especialmente útil cuando tenemos muchos requisitos y un presupuesto limitado de tiempo, como es el caso actual. Existe claramente el riesgo de no incluir muchos de los requisitos establecidos, por lo tanto, usando este mecanismo estamos al tanto de este hecho y podemos decidir, desde la primera etapa del proyecto, cuáles son los requisitos que no serán incluidos en la entrega final en caso de no ser posible.

En el caso de desarrollar una aplicación personalizada para una Empresa específica, esto además ofrece al resto de shareholders una visión clara y real de los posibles requisitos a incluir.

En este proyecto, se ha dividido la lista en 4 grupos: básicos, siguientes, finales y extras.

Por supuesto, el grupo de requisitos considerados como básicos será el primero a incluir en la aplicación y el grupo de requisitos extra son aquellos que potencialmente no serán incluídos en la entrega final por cuestiones de tiempo. Esta división nos permite desarrollar cada grupo en una iteración sin tener problemas de dependencia entre distintos requisitos.

Como se puede observar hasta este punto, con estos dos elementos podemos tener en cada momento una idea del progreso de la aplicación, riesgos y funcionalidades a incluir.

6.3.Manejo de Sprints

6.3.1. Primer Sprint del Proyecto: Dominio

Los objetivos para el primer Sprint están enfocados a adquirir todo el conocimiento posible acerca del dominio de la aplicación y planificar el futuro desarrollo. En este punto es desarrollado el Modelo de Dominio y se realizan distintas reuniones con personas claves directamente relacionadas con el proyecto para conocer qué esperan los potenciales usuarios de la aplicación y para finalmente entender los requerimientos generales del sistema.

Elementos generados:

- Plan de Sprints
- Lista de Requisitos
- Modelo de Dominio
- Modelo de Casos de Uso

Al terminar este Sprint se tiene una visión global del dominio del sistema y suficiente conocimiento sobre **qué** se quiere realizar.

6.3.2. Segundo Sprint del Proyevto: Arquitectura / Lógica de negocio

En este punto se toman las principales deciciones de diseño y se genera la primera propuesta de arquitectura del sistema. Es necesario invertir bastante tiempo tomando estas decisiones y entendiendo la lógica general del sistema.

Elementos generados:

- Especificación de algunos Casos de Uso
- Modelo de Negocio
- Diseño de la Arquitectura
- Grupo de sketches básicos para la Interfaz de Usuario

Al terminar este Sprint se tiene una visión técnica del sistema: sabemos **qué** queremos ofrecer y **cómo** hacerlo.

6.3.3. Tercer Sprint del Proyecto: Configuración básia de la aplicación

Es importante dedicar tiempo a la configuración de la aplicación. Este es el momento de pensar en profundidad qué tecnologías se van a utilizar, plugins, herramientas, qué procedimientos vamos a seguir, patrones, convenciones, etc. Además, se deben crear los elementos básicos de la arquitectura de la aplicación así como las interfaces de sus métodos básicos.

En este proyecto se utilizará Groovy & Grails y la IDE IntelliJ para desarrollar la aplicación.

Además, se hará uso de GIT para realizar un control de versiones pues se quiere generar diferentes iteraciones e irlas integrando a medida que se vayan realizando. La base de datos será postgreSQL y además haremos uso de la herramienta Stash.

Se realiza la generación de la estructura básica de paquetes y módulos del sistema en el recién creado proyecto. Especialmente en Grails es importante y útil el generar una buena estructura interna pues muchas utilidades pueden ser utilizadas entonces por convención.

Se realiza la generación de los elementos básicos de la aplicación, siempre enfocando mayoritariamente al primer grupo de requisitos a incluir. Se crean todas las interfaces, clases abstractas y elementos que con gran seguridad sabemos que habrá que generar en el futuro. Asimismo, se establecen los atributos de las clases de dominio y las interfaces de los métodos principales de los servicios y controladores, documento siempre la finalidad de cada método y la responsabilidad principal de cada clase. Todo ello desde un punto de vista abstracto, sin ir demasiado en detalles de implementación. Más adelante, a la hora de implementar funcionalidades, esto será útil para evitar perder la concentración durante importantes procesos de decisión e implementación.

Al terminar este Sprint sabemos qué herramientas usar y la configuración básica está realizada: todo listo para empezar a implementar funcionalidades.

Desde este momento, los Sprints estarán enfocados al desarrollo de iteraciones que incluyan grupos de requisitos.

6.3.4. Fases de desarrollo de requisitos

¿Cómo manejarlas?

Al planificar la implementación de requisitos es necesario pensar no sólo en el tiempo que se dedicará al desarrollo de la funcionalidad, sino el tiempo también dedicado al testeo y limpieza de la funcionalidad desarrollada. Este proceso mencionado muy brevemente es lo que yo suelo llamar “Cleaning / Testing / Fixing” (proceso CTF).

Proceso CTF

¿En qué consiste el proceso CTF?

Cuando una funcionalidad ha sido desarollada es necesario dedicar cierto tiempo, posiblemente mayor al propio de implementación, al testeo y limpieza del sistema.

Etapas que incluye el proceso:

Cleaning (limpieza de código)

Revisar el código realizado y aplicar criterios de calidad que no se hayan aplicado en un comienzo, ya sea porque no se tuvieron en cuenta o simplemente porque interferían en la concentración del desarrollador mientras programaba. Aplicar DRY donde aún no se haya hecho, renombrar clases y variables si se considera oportuno y realizar todas las modificaciones necesarias para garantizar que el código introducido es mínimo, comprensible y sencillo, y por lo tanto fácil de mantener o modificar en el futuro.

Testing

Es necesario testear todas las funcionalidades, bajo diferentes escenarios y en distintas posibles situaciones para observar las respuestas del sistema y detectar posibles bugs en una etapa temprana.

Fixing

Durante las 2 etapas anteriores se puede observar que un arreglo o cambio drástico en la implementación de la funcionalidad son necesarios, ya sea proque la funcionalidad no responde como se esperaba o porque se ha descubierto que el código es inaceptablemente rígido y cerrado a cambios.

¿Por qué es necesario aplicar CTF?

Los desarrolladores intentan la mayoría de las veces poder tener una funcionalidad operativa lo más pronto posible, de esta forma se puede decir que la funcionalidad está implementada y que el sistema se comporta como se espera. Para muchas personas y metodologías software ágil, lo más importante es tener software que funcione (principio básico del manifiesto de SCRUM). Siguiendo este criterio, aunque los desarrolladores apliquen criterios de calidad y buenas prácticas durante la etapa del desarrollo, es muy probable que al final de la jornada sean necesarios unos retoques, para garantizar calidad.

Los programadores pueden aplicar todas las buenas prácticas que conozcan de corazón, sin preocuparse demasiado por los detalles y por alcanzar un "código perfecto". Por ejemplo: no se debe emplear demasiado tiempo en elegir el nombre de una variable pues esto provoca una desconcentración del problema real que se quiere resolver (funcionalidad a implementar) y al final es sólo ruido. Mejor dejar estos detalles y refinamiento para el proceso de CTF.

¿Cuándo aplicar CTF?

Para ser coherentes con el resto del documento y ser fieles a la idea de "desarrollo y planificación ágil", CTF debe ser aplicado cuando sea conveniente y se considere necesario y de valor para el Proyecto.

Por lo general, debe ser aplicado luego de implementar cierta cantidad de código que esté relacionado entre sí. De todas formas, el proceso se adapta a las necesidades humanas y del Proyecto.

- Si se considera oportuno se puede aplicar CTF a cada funcionalidad independiente para garantizar máxima calidad y se dispone de un amplio presupuesto de tiempo.
- Otro enfoque es aplicar CTF a grupos de funcionalidades que pertenecen al mismo dominio, que tienen relación entre sí y constituyen un paquete cerrado de funcionalidades.
- Si el dominio abarca demasiadas funcionalidades, no es una buena práctica esperar al final de toda la implementación para aplicar CTF. En estos casos se puede realizar una práctica híbrida y aplicar CTF al generar algunas funcionalidades específicas o en cierto punto del proceso, al tener un subdominio implementado.

En el caso de este Proyecto, CTF se aplica luego de implementar un grupo de funcionalidades que pertenecen al mismo dominio, siempre y cuando el código generado no sea demasiado grande. En caso de tener grandes cantidades de código, CTF se aplica cuando se considera oportuno.

6.4.Planificación y Organización de Requisitos

La planificación de los requisitos también incluye un proceso constante de clarificación de los requisitos, utilizando como principal fuente de información a los usuarios finales y teniendo siempre en cuenta el presupuesto del proyecto y el Plan incial de Sprints.

A medida que se implementan los requisitos principales, pueden aparecer nuevas características "nice-to-have" muy relacionadas a los requisitos que estamos implementando. Esto ocurre debido a que no se ha realizado una especificación completa y detallada de los requisitos, ya que trabajamos de forma ágil para evitar perder demasiado tiempo durante

etapas de especificación y mantenernos abiertos a requisitos nuevos. La idea es la siguiente: si podemos reaccionar positivamente a requisitos nuevos y cambiantes durante el desarrollo del Proyecto, probablemente construyamos una arquitectura flexible que nos permita reaccionar de la misma manera al finalizar el Proyecto, garantizando mantenibilidad.

Cuando aparecen nuevas características “nice-to-have”, se debe decidir si va a ser incluída en la entrega final del producto o no.

Si se preveé que la nueva característica requerirá mucho tiempo y recursos, entonces se debe poner en la lista de Requisitos, en una posición dependiente de la prioridad que le querramos asignar o simplemente en una nueva sección “nice-to-have” y realizar la priorización en el próximo re-planning de requisitos. Durante la realización del proyecto deben realizarse al menos 3 planificaciones de requisitos. Éstas ocurrirán básicamente cuando especifiquemos la lista de Requisitos a incluir en cada gran iteración

En el caso de este Proyecto, se ha realizado un profundo análisis de la lista de Requisitos y sus prioridades al definir la primera lista de Requisitos a implementar en la primera iteración (requisitos básicos). Al terminar con la implementación de todos esos requisitos básicos, es necesario analizar el estado del Proyecto, la velocidad con la que se está trabajando y cómo nos estamos ajustando al Plan de Sprints pre-establecido. En este punto ya deberíamos tener una temprana alarma atendiendo principalmente a un punto: ¿vamos en tiempo? ¿necesitamos más tiempo del estimado?

En dependencia de la respuesta, se debe llevar a cabo la próxima planificación de requisitos de una forma u otra. Esta planificación se realizará para definir la Segunda Lista de Requisitos a implementar en la próxima gran iteración, habiendo organizado de nuevo los requisitos restantes de la Lista Principal de Requisitos.

¿Qué ha cambiado para realizar otra planificación?

- Probablemente haya nuevos requisitos que se quieran incluir o cambios en los existentes.
- Se tiene más información del dominio del sistema y la cantidad de recursos necesarios para implementar ciertas funcionalidades así como información adicional sobre elementos relacionados, como la facilidad de conexión con ciertas APIs de servicios externos. Probablemente esta información no existía en la primera planificación, por lo tanto la estimación ahora puede ser más certera.
- Es necesario atender a las alarmas obtenidas del Plan de Sprint y tomar decisiones en consecuencia. Si se tiene suficiente tiempo puede que se incluyan más requisitos de la lista. En caso contrario algunos deben ser descartados y es necesario llevar a cabo esta decisión mediante una apropiada planificación.

Si la característica se preveé sencilla y puramente relacionada con uno de los requisitos a implementar en el Sprint actual, se puede tener en cuenta y decidir implementarla, siempre y cuando, según el Plan de Sprints, se oberve que todo marcha según lo previsto o incluso mejor de lo esperado.

Es posible decidir descartar la nueva característica inmediatamente si no se considera de valor para el Proyecto o los usuarios finales. Esta decisión debe ser tomada en mutuo acuerdo con los usuarios finales.

Nota: Incluir nuevas características relacionadas con las existentes puede ser utilizado para testear la Arquitectura del Sistema. Si se pueden realizar cambios e incluir nuevas funcionalidades de una forma eficiente, sencilla y siguiendo la línea base de la Arquitectura,

entonces probablemente la Arquitectura esté bien diseñada. En este proyecto se ha hecho uso de esta idea.

6.5.Seguimiento y metodología general

En este apartado se exponen las estrategias principales de organización, planificación y manejo del proyecto llevadas a cabo y además se hacen evidentes los elementos incorporados de las distintas metodologías de desarrollo de software.

- Se ha trabajado con enfoque en distintos Sprints.
- Se mantiene una daily-meeting para informar al grupo cercano de compañeros acerca del estado actual del proyecto, problemas encontrados, soporte necesitado (para la creación de cuentas en los servicios internos, por ejemplo), etc.
- Al finalizar cada Sprint se ha tenido una pequeña reunión con el cliente para mostrar las funcionalidades incluidas en la nueva iteración, informar sobre los progresos del proyecto y obtener feedbacks. Además, en este punto se negocia y establece los nuevos requisitos a incluir en la próxima iteración.
- Además, cada 2 Sprints se ha tenido una reunión con un mayor número de stakeholders con el fin de mantenerlos informados acerca de los progresos del proyecto desde un punto de vista más enfocado a la gestión y management del mismo.
- Al final de cada Sprint se analiza el estado del proyecto, la velocidad de avance, si se han cumplido todos los objetivos del Sprint o no y las causas de los resultados obtenidos. Se toma nota de todas las recomendaciones y feedbacks obtenidos de la reunión con el cliente y se prepara el nuevo Sprint atendiendo a los criterios de prioridad sobre los requisitos establecidos en la misma reunión.

6.6.Pizarra de trabajo

Este proceso de seguimiento, control y gestión llevado a cabo se refleja en la pizarra del proyecto. Desde el comienzo del proyecto se ha hecho uso de una pizarra, con piezas magnéticas para mantener a vista los documentos más importantes y diversos post-its, pensado para garantizar una apropiada organización durante la elaboración del proyecto y de esta forma nunca perder la concentración u olvidar el punto de desarrollo en el que nos encontramos, lo que ya hemos desarrollado y hacia donde nos dirigimos.

En los documentos anexos se puede observar la evolución de la pizarra de trabajo a lo largo del desarrollo del proyecto. La siguiente imagen ha sido seleccionada del grupo de imágenes como muestra de la visión general de la pizarra de trabajo.

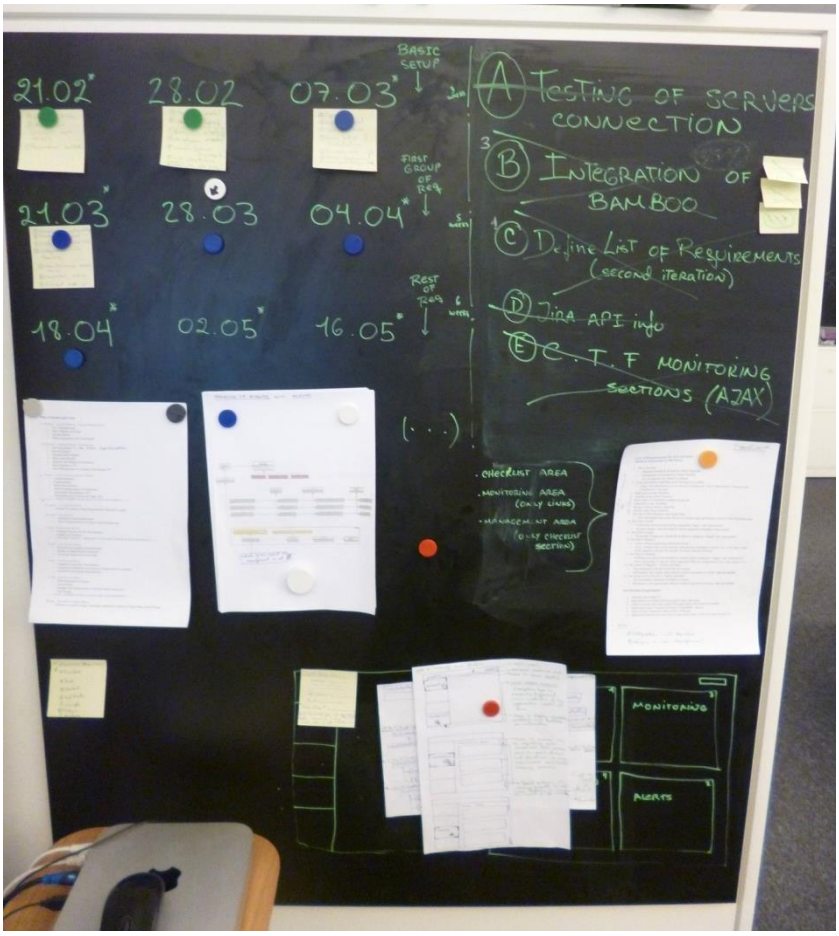


Ilustración 4. Pizarra de trabajo - visión global

A continuación se describe la distribución de la pizarra de trabajo pues se considera un elemento de crucial importancia en el desarrollo de este proyecto.

6.6.1. Esquina superior izquierda

Se ha creado un modelo de calendario personal con las deadlines correspondientes a los finales de Sprint. Se mantiende una figura magnética con la imagen de una flecha indicando en qué Sprint se está trabajando en el momento actual. Al terminar un Sprint, un post-it se crea y pone bajo la deadline correspondiente donde se escribe de manera general todo lo que se ha conseguido (objetivos) y/o artefactos generados. El color de los imanes utilizados también tiene un significado: los verdes indican la etapa de inicio (procesos de análisis y modelado del sistema), los azules indican etapas de implementación de funcionalidades y finalmente el blanco vendría a indicar la etapa de limpieza y testeo final para terminar ofreciendo la versión final de la aplicación.

Ilustración 4. Pizarra de trabajo - área de deadlines

6.6.2. Centro-izquierda

En este sitio se encuentra el Plan de Sprints. Este documento debe mantenerse en la pizarra en todo momento como elemento clave de la gestión del proyecto que es. Es fundamental poder ver a simple vista y de manera rápida la relación entre el calendario de trabajo, los objetivos reales conseguidos en cada Sprint y el Plan de Sprints original para obtener una rápida impresión del progreso y estado del proyecto.

6.6.3. Centro

Se colocan todos los documentos importantes considerados como documentación oficial del proyecto: modelo del dominio, modelo del negocio, casos de uso, modelo de análisis, etc. Si uno de los documentos es actualizado durante el desarrollo del proyecto, en la pizarra debe mantenerse solamente la última versión. El motivo es simple: tener rápido acceso a información física y actualizada que permita analizar el sistema en un momento dado y realizar reuniones rápidas de clarificación de requisitos sin necesidad de acceder a documentos digitales.

6.6.4. Centro-derecha

En este sitio se encuentra la Lista de Requisitos funcionales a incluir en la siguiente iteración. Esta lista está expresada en modo formal y contiene información detallada acerca de lo que se quiere implementar.

Ilustración 5. Pizarra de trabajo - área central

6.6.5. Esquina superior derecha

Se listan los objetivos para el Sprint actual (o para la semana actual en caso de que haya demasiados requisitos para el Sprint y no quepa la lista de manera apropiada en el espacio reservado para ello). A lo largo del Sprint los elementos de la lista serán tachados o comentados con pequeños post-its para reflejar de manera clara el estado del Sprint. A

diferencia de la Lista de Requisitos funcionales, esta lista de objetivos está pensada para una jornada más corta de trabajo y la descripción de los objetivos se realiza de forma abstracta, utilizando palabras claves para saber de forma sencilla y rápida en el punto de desarrollo que nos encontramos durante cualquier día de trabajo.

Ilustración 6. Pizarra de trabajo - área de objetivos de Sprint

6.6.6. Esquina inferior derecha

Reservada para todo lo que esté relacionado con la Interfaz de Usuario. Se dibujan las distintas opciones y se colocan los Sketches básicos a seguir así como cualquier diseño pensado.

Ilustración 7. Pizarra de trabajo - UI área

7. Capítulo 7: Requisitos

7.1.Elicitación de Requisitos

La elicitación de requisitos se ha llevado a cabo utilizando distintas herramientas y técnicas. Se ha empleado la técnica de la entrevista con stakeholders relevantes del sistema y algunos usuarios finales para entender sus necesidades y conocer las expectativas individuales sobre la aplicación. Esta técnica es particularmente reveladora ya que permite entender en detalle las necesidades de los usuarios y ganar conocimiento sobre la prioridad de ciertos requisitos, lo cual será bastante útil a la hora de priorizar su implementación. Además, por medio de esta técnica se puede observar qué requisitos son más inestables y propensos a cambiar en el futuro. Si el usuario final no sabe realmente lo que necesita de un requisito pre-establecido, probablemente este requisito esté mal definido o no se corresponda con una necesidad de valor para el usuario.

Asimismo, los stakeholders son capaces de proporcionar información ajena al equipo de desarrollo: ejemplo, se preveé el cambio de un servicio por otro, un cambio en el procedimiento, se sabe que ciertas necesidades de los usuarios serán cubiertas por algún otro proyecto, etc. Nuevamente, todo influirá en la priorización de los requisitos y la gestión de riesgos.

Las fuentes de los Requisitos han sido:

- Personal (propia)
- Stakeholders
- Modelos de Casos de Uso

Más adelante se hablará de la generación del modelo de casos de uso. En este punto basta con decir que dichos modelos ayudaron a la identificación de requisitos poco visibles y la total comprensión de algunos de ellos.

En general, la especificación de los requisitos se trata de un proceso iterativo donde hay diversas fuentes y cada una proporciona entradas y puntos de discusión en la otra. Ejemplo: un descubrimiento de requisitos durante el modelado de casos de uso puede dar lugar a una discusión detallada con un stakeholder específico o cierta información de un stakeholder puede hacer que analicemos desde otro ángulo nuestro modelo de casos de uso.

Para definir los Requisitos se ha usado una Lista de Requisitos donde se han agrupado todos con una breve descripción y una enumeración. Sencillo, simple y comprensible. Cada requisito se expresa como una necesidad general sin ser detallada. Cada uno de ellos puede ser una funcionalidad a implementar o un grupo de ellas. La agrupación se realiza pensando en el dominio del “problema”, dominio del “requisito” y lo que queremos ofrecer, sin llegar a especificar a nivel atómico.

Dicha lista se encuentra dividia en varias secciones para garantizar un enfoque ágil sobre los Requisitos.

Las técnicas de Ingeniería de Requisitos utilizadas para el proceso de elicitación de requisitos se detallan a continuación. Estas técnicas se han utilizado de forma paralela pues en muchas

ocasiones unas constituyen la fuente de alimentación de otras y en realidad todas contribuyen de una forma u otra a la realización de las demás. Sólo por citar algunos ejemplos, cuando definimos casos de uso u observamos el sistema, las dudas generadas serán clarificadas mediante entrevitas, y muchas veces durante las entrevistas obtenemos información detallada que implica un cambio en los casos de uso o nos lleva a realizar una observación detallada de ciertos procesos.

7.1.1. Entrevistas

Se han utilizado los 2 tipos de entrevistas que propone el proceso de Elicitación de Ingeniería de Requisitos: charlas abiertas y cerradas. Para las entrevistas cerradas se ha preparado una serie de preguntas sobre aquellos requisitos más complejos y difíciles de entender y se ha seleccionado a personas claves con suficiente conocimiento sobre el dominio de la aplicación con el fin de ganar tiempo y agilizar esta etapa.

Las entrevistas abiertas han sido más abundates por el gran valor que proporcionan. Como no se ha realizado un proceso exhaustivo de especificación de requisitos, puede que pasen desapercibidos algunos requisitos con valor para distintos tipos de usuarios. Al ser conscientes de esta realidad, se ha decidido que la mejor forma de cubrir la detección del mayor número de requisitos y cubrir las expectativas del usuario es mediante la utilización de entrevistas abiertas en las que cada cual expresa qué espera de la aplicación. Esto ha sido especialmente útil para identificar requisitos ocultos, sincronizar a todos los interesados en el proyecto con la misma visión global de la aplicación mediante la discusión directa a modo de charla de sus características y, además, para determinar cuán prioritarios son ciertos requisitos para los usuarios: un requisito que a simple vista puede parecer de poco valor pudiera adquirir prioridad si los usuarios finales declaran que tiene un alto valor de negocio. Los implicados en este tipo de charla han sido mayoritariamente los usuarios finales pues queremos ver qué esperan conseguir exactamente con el uso de la aplicación.

7.1.2. Observación y análisis

Un método efectivo de adquirir requisitos y una comprensión global del dominio del sistema es la observación y el análisis del mismo, especialmente si la aplicación a construir va a replicar de un modo automatizado y mediante herramientas software un proceso real existente.

Se ha prestado especial atención al modo de operar de los usuarios encargados del proceso de Release durante los días de despliegue de las nuevas versiones. Cada vez que una Release se ha puesto en marcha se ha seguido de cerca el proceso, observando comportamientos, detectando los fallos del proceso, las mejoras que pudieran ser introducidas por la aplicación y los puntos más importantes a implementar que podrían hacer más eficiente el proceso de despliegue. Además se ha detectado cuáles son los procesos más propensos a cambiar, lo cual influye directamente en el nivel de riesgo de los requisitos relaciones, y distintas cuestiones han surgido como resultado de todo este proceso de observación y análisis.

7.1.3. Casos de uso

El uso de esta técnica y los elementos generados se describe extensamente en el capítulo dedicado a los Casos de Uso del sistema. A pesar de ser una ténica de la Ingeniría de Requisitos formal y formar parte de las metodologías de desarrollo más conservativas, en este proyecto

se ha utilizado bajo criterios de agilidad, aprovechando los beneficios que ofrece pero sin invertir demasiados recursos en ello.

7.2.Especificación de Requisitos

En este proyecto no se ha realizado una especificación de requisitos formal que siga los criterios de Ingeniería de Requisitos ya que se considera una práctica demasiado formal, rígida y extensa, tal vez efectiva para grandes proyectos donde se espera que los requisitos no cambien demasiado con el tiempo, pero totalmente obsoleta en el manejo de proyectos bajo metodologías de desarrollo ágil.

Se ha obtado por manejar los requisitos bajo un enfoque ágil, haciendo uso de los elementos citados anteriormente y describiendo los requisitos deseados en un lenguaje informal y sin demasiadas especificaciones, simplemente tendiendo una idea clara de qué se quiere conseguir, qué valor aporta el requisito al usuario y la prioridad del mismo.

Por lo tanto, las estimaciones iniciales se entiende que no sean precisas, ya que de primera mano no se sabe la magnitud real que puede tener un requisito específico. Se pretende realizar una especificación más detallada, en términos de búsqueda de información, clarificación con usuarios y clientes, etc., cuando llegue el momento en que el requisito vaya a ser incluído en una iteración y sea necesario planear dicha iteración con más detalles.

Por esta razón, es totalmente normal, aceptado y esperado que se deban especificar y clarificar los requisitos durante etapas de implementación durnate cada fase de iteración. Esto ayuda al enfoque ágil que queremos garantizar sobre el proyecto, además de ahorrar tiempo en especificar completa y formalmente un requisito al inicio del proyecto y tener que volver a hacerlo antes de implementarlo debido a algún cambio de especificación.

Obviamente es necesario conocer los requisitos que queremos incluir en una aplicación al comienzo del proyecto para saber su magnitud y realizar una planificación inicial. Por este motivo se ha dedicado un tiempo especial, mayor que cualquier tiempo que se le pueda dedicar durante las estapas de implementación, y se ha utilizado las técnicas descritas anteriormente para llevar a cabo la “Ingeniería Ágil de Requisitos”.

7.3.Gestión de Requisitos

Todos los procedimientos de negociación, documentación y gestión de Requisitos se han llevado a cabo mediante el uso de la lista de Requisitos creada (Pila de Producto, en SCRUM).

Como Anexo se encuentra la lista de Requisitos original, dividida en los distintos grupos que se ha mencionado en secciones anteriores. Consecutivamente, se puede observar la evolución final de la lista de Requisitos, por supuesto manteniendo la misma división que la lista original.

La diferencia radica en el hecho de que los requisitos van cambiando y evolucionando a lo largo del Proyecto y la prioridad de los mismos también sufre sus cambios. Hay requisitos nuevos que se añaden en etapas posteriores, requisitos descubiertos durante la implementación, etc.

La prioridad de los requisitos de la lista es revisada antes de la elaboración de cada lista de Requisitos a incluir en una nueva iteración. En cada momento en que planificamos una

iteración tenemos una visión diferente de los requisitos, las necesidades de los usuarios y, probablemente, más información acerca de los requisitos: dificultad de implementación, usuarios afectados directamente, valor real que proporciona el requisito para el usuario, etc. Todo ello provoca que podamos y debamos llevar a cabo un análisis prioritario sobre los requisitos restantes antes de establecer la próxima lista de requisitos a incluir en una iteración.

La Lista de Requisitos es una variante propia de la Pila de Producto que propone la Metodología SCRUM. Es un documento vivo que cambia con el tiempo y contiene todas aquellas características que el usuario desea que se implementen y compongan el producto final. Esta lista admite nuevos elementos, modificación de los existentes, sigue criterios de agilidad y simplicidad pues sólo se utiliza una breve descripción de la funcionalidad que se quiere implementar, y está organizada atendiendo a criterios de prioridad que pueden variar en el tiempo.

Las listas de requisitos por iteración se asemejan asimismo a la Pila de Sprint que propone SCRUM. Se crean a partir de la Lista de Requisitos y está organizada atendiendo a criterios de prioridad. El contenido de dicha lista está generalmente estimado y se preveé que pueda ser implementado en el tiempo pensado de duración de la iteración. Por supuesto, esta estimación puede ser completamente errónea y es en este punto donde las estrategias de gestión descritas en el Capítulo 6 entran en juego para mitigar riesgos.

8. Capítulo 8: Análisis y Diseño

8.1.Especificaciones generales

Se ha aplicado criterios de agilidad a la hora de generar documentos de especificación que contribuyan al análisis y el diseño de la aplicación, como las listas de requisitos, diagramas de casos de uso y modelos UML en general. Dichos criterios de agilidad son los siguientes:

- Los diagramas servirán principalmente como medio de comunicación entre los distintos stakeholders del proyecto.
- Los diagramas serán simples, sencillos y legibles incluso para aquellos menos familiarizados con los conceptos de ingeniería de software aplicados.
- Los diagramas se dividirán en distintos bloques según su dominio para facilitar su comprensión.
- Los diagramas sólo contendrán aquellos elementos de la arquitectura del software que sean considerados lo más relevantes para la comprensión del sistema.
- La actualización de los diagramas será llevada a cabo cuando se considere necesario. Sólo se considera obligatoria una actualización al finalizar el proyecto para incluir los diagramas como parte de la documentación oficial del mismo.

En resumen, el ideal a seguir es considerar los diagramas UML como una herramienta que nos ayude a comunicarnos y obtener una visión general del sistema y su alcance. En ningún momento se requiere una especificación detallada de los componentes, atributos o métodos de las clases: sólo se especifica aquello que se considere relevante. Los diagramas UML no deben en ningún momento convertirse en una carga pesada que mantener. Por ello las actualizaciones sólo deben ser realizadas cuando se considere oportuno.

Como este proyecto se ha realizado bajo el título de Trabajo de Fin de Grado, se ha considerado oportuno incluir los diagramas UML generados como parte de la documentación oficial del proyecto. La versión de los diagramas corresponde a la actualización final de los mismos, que refleja la estructura de la aplicación en el momento final del desarrollo.

8.2.Modelo del Dominio

El modelado del dominio se ha llevado a cabo utilizando distintas herramientas y técnicas. Se ha empleado la técnica de la entrevista con personas cercanas al problema que se quiere resolver: usuarios finales, personas afectadas y aquellos que comprenden como funcionan los distintos elementos que se han nombrado durante la definición del Proyecto.

De cada uno de ellos se extrajo información útil que ayudó a comprender a un nivel elevado el dominio del sistema que se quiere implementar.

Una vez comprendido el dominio es necesario modelarlo para tener una guía durante las etapas venideras de desarrollo y contar con un elemento que sirva de comunicación entre la parte de desarrollo y el resto de stakeholders y permita evitar malinterpretaciones. Es extremadamente importante que todas las partes involucradas en el proyecto tengan la misma visión del dominio del sistema. Esto permitirá mantener una mejor comunicación, evitar confusiones y generar funcionalidades que respondan a los problemas que se quieren resolver.

8.2.1. Diagramas

Se ha utilizado UML como herramienta para el modelado del dominio. El modelo generado corresponde a la primera versión, creada en la etapa inicial del Proyecto, y como es de esperar ha cambiado ligeramente a lo largo del desarrollo del mismo, aunque manteniendo la línea y conceptos bases capturados en la primera versión.

Asimismo se puede observar que el modelo de dominio no es demasiado extenso. Esto se debe a que la aplicación tiene un carácter funcional. Esta aplicación está pensada para ofrecer servicios e información en tiempo real y realizar la integración con servicios externos de manera dinámica y sin generar demasiados elementos de dominio. Además, a la hora de modelar se han seguido criterios de simplicidad: se pretende ofrecer una idea completa del dominio del sistema pero sin entrar en demasiados detalles acerca de su implementación, pues el enfoque del modelado es puramente ágil.

El modelo del dominio refleja todas aquellas entidades que constituyen la capa de "Modelo" de la aplicación. La gran mayoría de las clases reflejadas son persistentes en base de datos y algunas, sin embargo, sirven como elemento contenedor de información para ser enviado de un servicio a otro o simplemente para ser mostrado al usuario. En Grails, por convención, todas las clases consideradas como Modelo se deben colocar bajo la carpeta "domain" y son persistentes en base de datos por defecto. Sin embargo, en la realidad sabemos que la definición de una clase de tipo modelo/dominio no incluye características de persistencia, por lo que es totalmente posible tener clases de Modelo que no sean necesariamente persistentes en base de datos, si lo pensamos desde un punto de vista abstracto y ajeno a las tecnologías utilizadas. Dichas clases de modelo que no son persistentes deben ser colocados en otro sitio del proyecto para que no sean persistentes por defecto, sin embargo, siguen considerándose como clases de dominio.

El modelo de dominio está claramente diferenciado en distintos grupos pues constituyen grupos con un dominio y problemas independientes del resto. Esta separación nos permite pensar desde un primer momento en una posible división por paquetes, módulos o plugins.

El grupo de clases que forman el concepto de Checklist es realmente mínimo. Se han aplicado criterios de simplicidad para obtener esta versión final pues en su diseño existen muchísimas posibilidades. Por ejemplo, podríamos diseñar la Checklist para tener distintos tipos de categorías y agrupar los elementos de distintas formas. Sin embargo, se ha optado por reflejar el problema actual representando la Checklist exactamente en el modo en que los usuarios la utilizan. Siguiendo criterios de agilidad y simplicidad de XP se diseña la Checklist bajo el escenario existente y sin pensar en posible futuros escenarios que quizás nunca lleguen a suceder. El diagrama muestra que una Checklist en el sistema, como concepto general, está compuesta por pequeñas listas diarias, cada una con su propia información y objetivos (checkpoints) a completar.

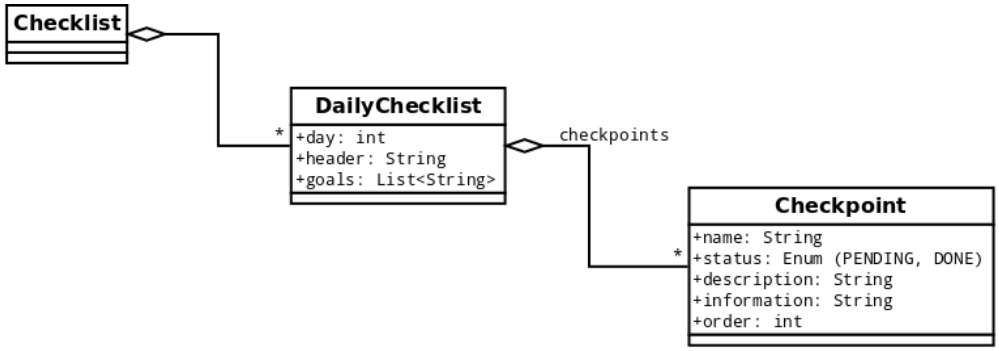


Ilustración 8. Modelo del dominio - Checklist

Existe otro grupo de clases independientes para reflejar los templates del sistema. Se observa un sistema de jerarquía basado en el principio "Open-Close" para implementar su representación. Como es totalmente posible tener distintos tipos de template en el sistema y se preveé que en el futuro se puedan incorporar otros nuevos, tenemos un tipo general y abstracto que contiene las funcionalidades y características básicas de todos los templates y sirve para agrupar a todos los templates del sistema y al mismo tiempo de base de extensión para los nuevos que se incorporen en el futuro. La aplicación del principio "Open-Close" nos permite evitar la modificación de los elementos existentes por la llegada de uno nuevo mientras nos mantenemos flexibles a la incorporación de más templates, garantizando que seguirán las convenciones pre-establecidas, compartirán las mismas características base que los otros templates y podrán ser agrupados bajo el mismo concepto.

Se considerará como template todos aquellos elementos que contengan información estática que deba ser usada siempre de la misma forma. Hay ciertos tipos de emails, meetings y tickets para los que siempre utilizaremos el mismo tipo de información: asunto, mensaje, destinatarios, etc. Por ello, estos elementos constituyen los templates potenciales de la aplicación.

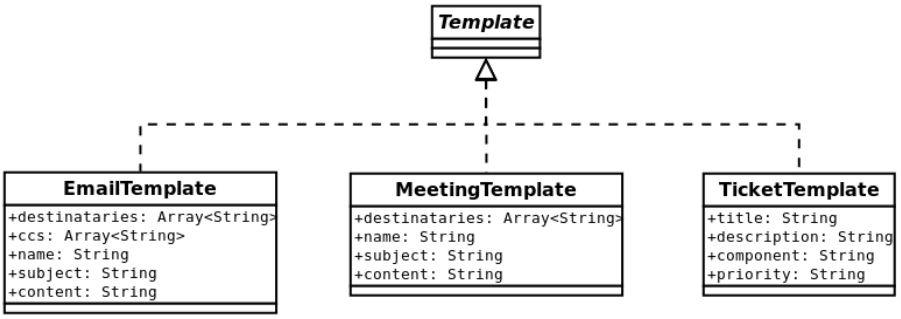


Ilustración 9. Modelo del dominio - Templates

La mayor parte del modelo de dominio está enfocada a todos aquellos elementos relacionados con servicios externos y por supuesto, con la Release. En la aplicación se manejará distintos tipos de eventos provenientes de distintas fuentes que serán principalmente servicios externos. En la versión final de la aplicación no se ha incorporado todos los eventos reflejados en el diagrama. Se asume que al incorporar nuevos tipos de eventos, nuevas categorías

pueden aparecer para agruparlos de una forma más lógica y eficiente. Por ahora, se mantiene un diseño simple y comprensible.

El modelado general de una Release detecta 2 componentes funddamentales: ReleaseTeam y ReleaseLocations. Estos componentes se han extraído directamente del concepto real de Release con el que trabajan los usuarios. Además, se asume que una Release tiene BreakingChanges, que son cambios introducidos por el usuario durante el desarrollo de la nueva versión a desplegar y que no constituyen eventos. Inicialmente se relacionaba el concepto de Release con los eventos: “una Release tiene eventos”, es la forma natural de pensar. Sin embargo, no se considera necesario ni recomensable establecer una relación directa y fija entre una Release y los eventos que ocurran en la aplicación. La relación se mantiene pero se mueve a un nivel más dinámico: una Release se relacionada con aquellos eventos que hayan sucedido en el período en que la Release se encontraba activa. Las fechas de activación y creación jugarán un papel fundamental en este aspecto.

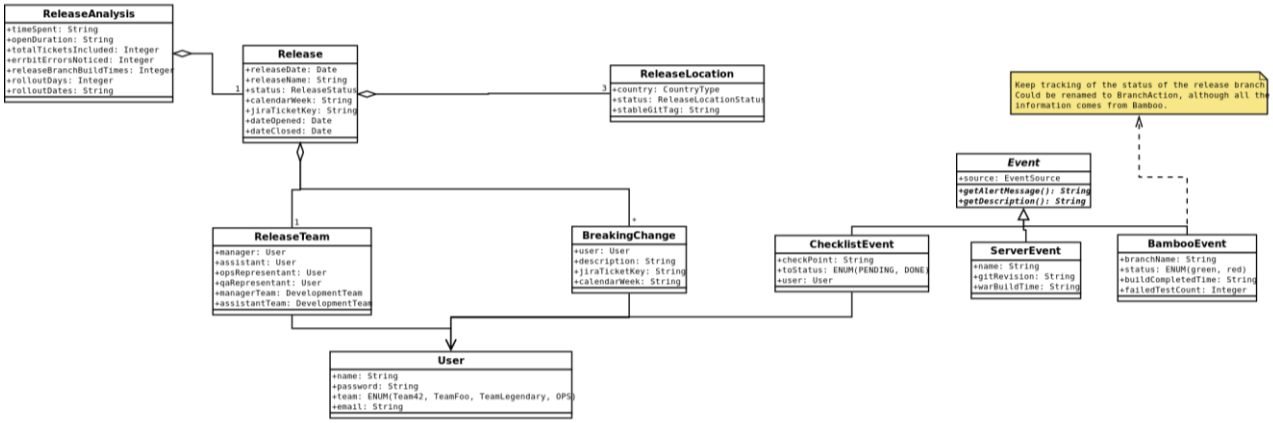


Ilustración 10. Modelo del dominio - Tracking

Otros elementos, son utilizados para representar un conjunto de información que ha de ser recopilada por algún servicio de la aplicación y que no se relacionan con otros componentes de tipo “modelo” del sistema. Estas clases son consideradas “ViewModels” y se utilizan mayoritariamente para mostrar información al usuario.

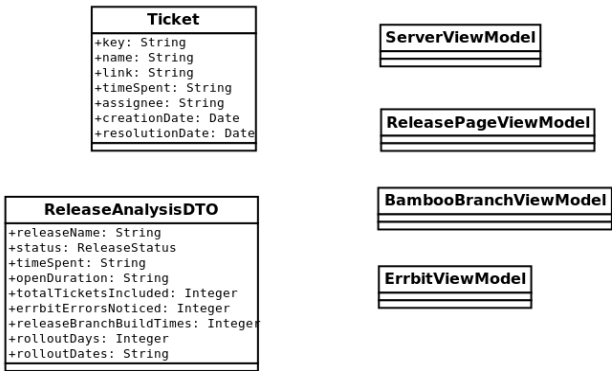


Ilustración 11. Modelo del dominio – ViewModels

8.3.Casos de Uso

El modelo de casos de uso se ha dividido en 3 paquetes principales que representan la división primera de los Requisitos en distintos grupos independientes entre sí, que pueden ser desarrollados en iteraciones distintas y se organizan atendiendo a criterios de prioridades. De esta forma, el primer modelo de casos de uso corresponderá a los requisitos más básicos y así sucesivamente.

8.3.1. Actores

En este sistema aparecen 2 actores fundamentalmente que representan los roles de los usuarios: Release-User y Release-Process Manager.

Release-User

Es el usuario interesado en el proceso de Release que se está llevando a cabo en la empresa. Quiere utilizar la aplicación para conocer el estado actual del proceso, monitorear el estado general de los elementos implicados (elementos de infraestructura o servicios externos de ayuda) y realizar ciertas tareas relacionadas con el proceso de Release de forma automática mediante la aplicación.

Release-Process Manager

Es el usuario que establece cómo se va a realizar el proceso y tiene los privilegios de realizar cambios de configuración en la aplicación. Este usuario decide cuáles serán los pasos de la Checklist a seguir, elaborará los templates que por defecto será usados por el otro tipo de actor y realizará todas las tareas relacionadas con el management, por ejemplo, el manejo de usuarios.

Además, existen otros actores del sistema que representan a Servicios Externos, con los cuales el sistema establece comunicación e interactúa:

Servidor

Contiene una versión de la aplicación para la que se quiere hacer Release. La versión puede corresponder a la Release actual o ser una anterior. Proporciona información acerca del estado de la versión desplegada.

Bamboo

Servicio de integración automático. Proporciona información acerca del estado de construcción de las ramas de Release. Básicamente, nos informa si la rama se ha construido exitosamente y todos los test han pasado o no.

Errbit

Servicio de captura y almacenado de errores. Proporciona información sobre los errores ocurridos en las versiones desplegadas en los servidores de interés.

Icinga

Servicio de monitoreo del rendimiento y captura de errores de infraestructura. Proporciona información sobre los errores referentes a los servidores y base de datos de interés.

TestRails

Servicio de manejo y gestión de TestCases. Proporciona información acerca del estado actual y progreso de la ejecución de TestCases en las versiones nuevas desplegadas.

Zimbra

Servicio de mensajería. Permite el envío de emails a usuarios de la red de la empresa, así como la creación de las meetings requeridas por el sistema.

Jira

Servicio de gestión y organización de tareas. Proporciona información sobre las tareas que se realizan a cabo en la empresa, el tiempo loggeado por los usuarios en distintos tickets y otros tipos de información relevante para las esferas más administrativas. Permite, además, la creación de tickets y la manipulación de los mismos.

8.3.2. Diagramas

A continuación se mostrarán y describirán los distintos grupos que componen el modelo de casos de uso. Dichos modelos han sido divididos en 3 grupos para mantener los diagramas inteligibles y sencillos. En cierto modo, cada grupo de casos de uso se corresponde con el grupo de funcionalidades deseadas a incluir en cada gran iteración individual.

De todas formas, los modelos de casos de uso se utilizarán como una herramienta para comprender el dominio del sistema, especificar requisitos y favorecer la comunicación con usuarios finales y clientes. En el futuro pudiera darse el caso que algún caso de uso se implemente en una iteración previa o posterior a la inicialmente pensada.

Básicos

En este paquete se encuentran todos los casos de uso relacionados con la Checklist del sistema. Se recuerda que la Checklist servirá de ayuda para saber qué actividades hay que realizar durante el proceso de Release, cuáles de ellas han sido realizadas y cuáles están aún pendientes. El actor “Release-User” será capaz de realizar todas las funcionalidades básicas relacionadas con la Checklist. Este actor, que está interesado en realizar un seguimiento del proceso de Release, podrá visualizar la Checklist y todos sus componentes, así como realizar el marcado y desmarcado de los puntos de la misma. Una funcionalidad extra es la Limpieza de la Checklist. Este caso de uso está pensado para facilitar el trabajo de los usuarios cuando quieran resetear el estado de la Checklist y comenzar el proceso de Release nuevamente. Dicha funcionalidad desmarcará todos los puntos de la lista general.

Además, existe un caso de uso que no es ejecutado directamente por ningún usuario pero se encuentra incluído en algunas funcionalidades: el tracking de las acciones del usuario. Este caso de uso existe ya que queremos dejar constancia en base de datos de todas las acciones realizadas por cada usuario.

Por otro lado, el actor “Release-process Manager” podrá realizar todas las tareas relacionadas con la gestión de la Checklist y sus componentes.

Además, en este paquete se encuentran todos los casos de uso relacionados con las acciones

de monitoreo mediante la conexión con servicios externos. En este caso, no existe ningún caso de uso que implique la gestión de algún elemento por lo que solamente aparece el actor "Release User" como interesado en monitorizar el estado general del sistema y sus componentes.

Básicamente, el usuario podrá monitorear el estado del sistema de forma general y recibir alertas cuando se produzca algún cambio en el estado de algún elemento relevante. El monitoreo del estado del sistema incluye la visualización de los errores captados en los servicios externos "Icinga" y "Errbit", el estado del servicio de integración "Bamboo", el estado / progreso de los TestCases en "TestRails" y por supuesto, el estado de los servidores.

Para tener una visión real y acertada del estado del sistema, es necesario actualizar el estado de cada uno de los elementos mencionados anteriormente cada vez que se quiera visualizar la información referente a ellos.

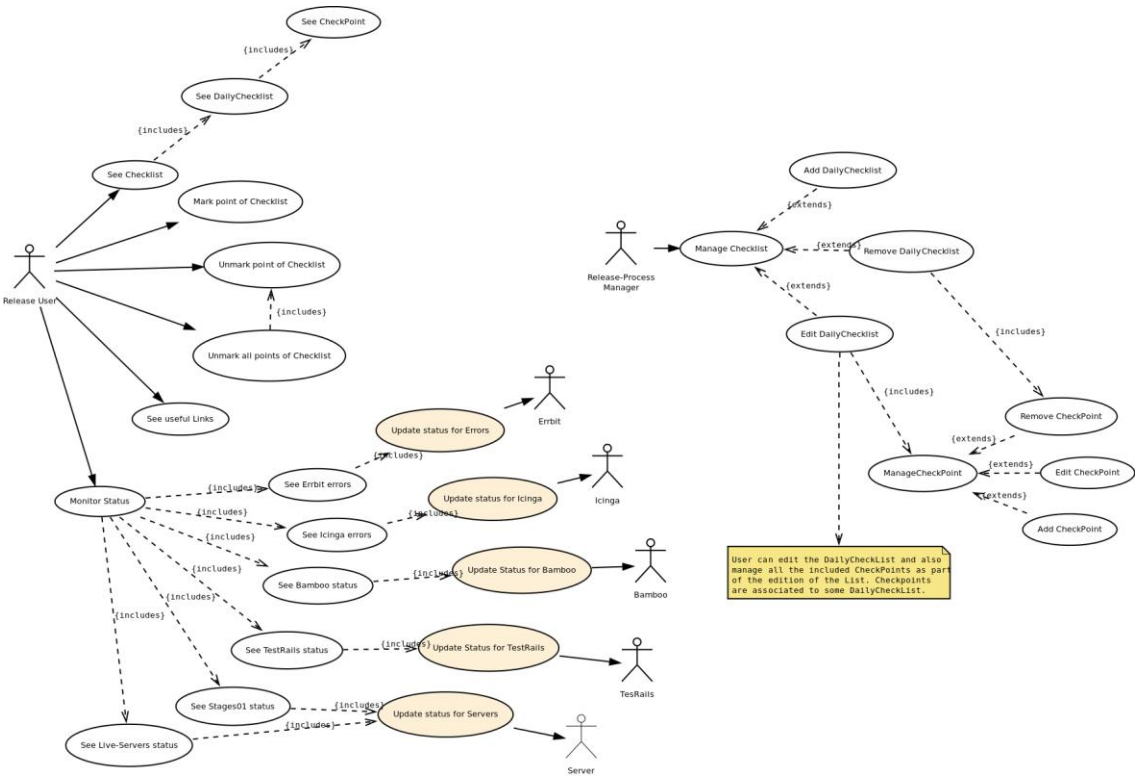


Ilustración 12. Modelo de Casos de Uso - Básicos

Segundo grupo

En este paquete se encuentran todos los casos de uso correspondientes a las distintas tareas que se pretende automatizar. Todos ellos proporcionan valor y están disponibles para el actor "Release User" que, como se ha dicho en el apartado de descripción de actores, tiene interés en utilizar la aplicación para realizar ciertas tareas referentes a la Release que se han automatizado.

Se destaca que el caso de uso "Add Breaking Change" no corresponde al área de gestión pues Breaking Changes son aquellos cambios realizados por los desarrolladores que se están incluyendo en la nueva Release y pueden ocasionar problemas. El actor "Release User" desea dejar constancia en base de datos de estos cambios, por lo que este constituye un caso de uso de su interés. Además, esta acción no provoca un cambio en el proceso de Release; por ello el ejecutor es el actor "Release User" y no "Release-process Manager".

Como se puede observar, algunos casos de uso desencadenan la ejecución de otros, aunque la gran mayoría son ejecutables directamente por el actor "Release User" de forma

independiente, incluso aunque puedan ser ejecutados también por medio de otro caso de uso. Los únicos casos de uso que se quedan fuera del alcance directo del usuario son los relacionados con el sistema de alerta, ya que este sistema se manejará de forma automática e interna en dependencia del estado de la aplicación y de la ejecución de ciertos casos de uso especiales.

En este paquete se incluyen además aquello casos de uso que no se consideran básicos y se pretenden implementar en una iteración posterior a la correspondiente a los casos de uso básicos.

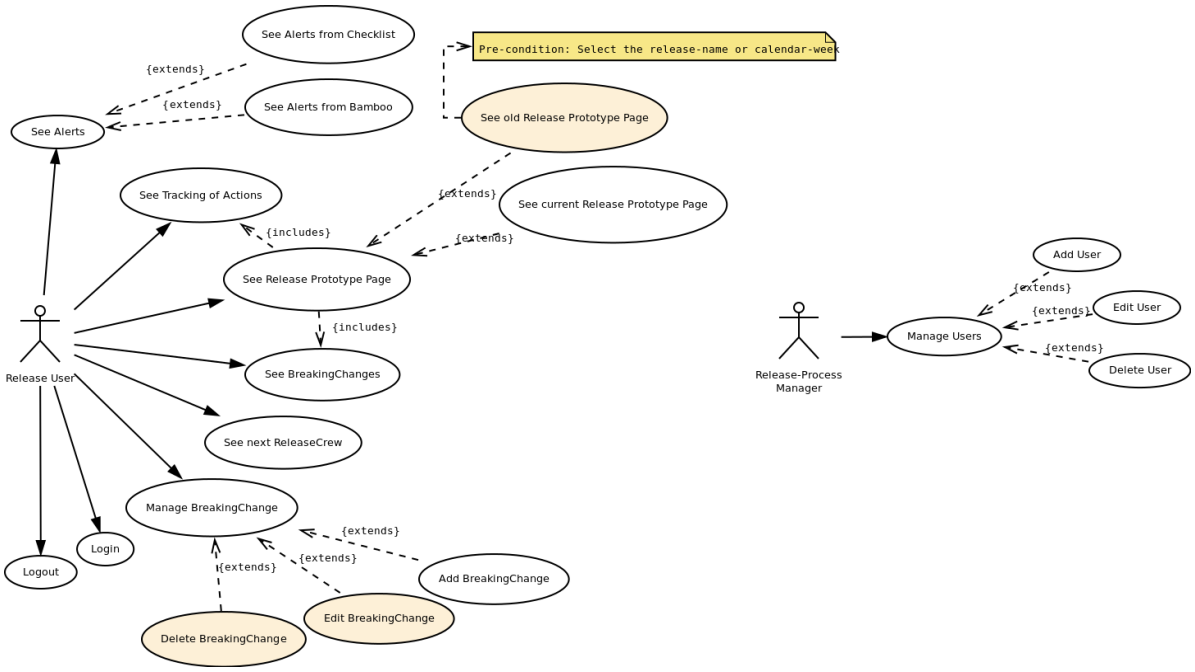


Ilustración 13. Modelo de Casos de Uso - Segundo grupo

Tercer grupo

Finalmente, el tercer grupo de casos de uso está compuesto también por tareas que se van a automatizar, sin embargo se agrupan de forma separada ya que se consideran los menos prioritarios en este punto del desarrollo del proyecto.

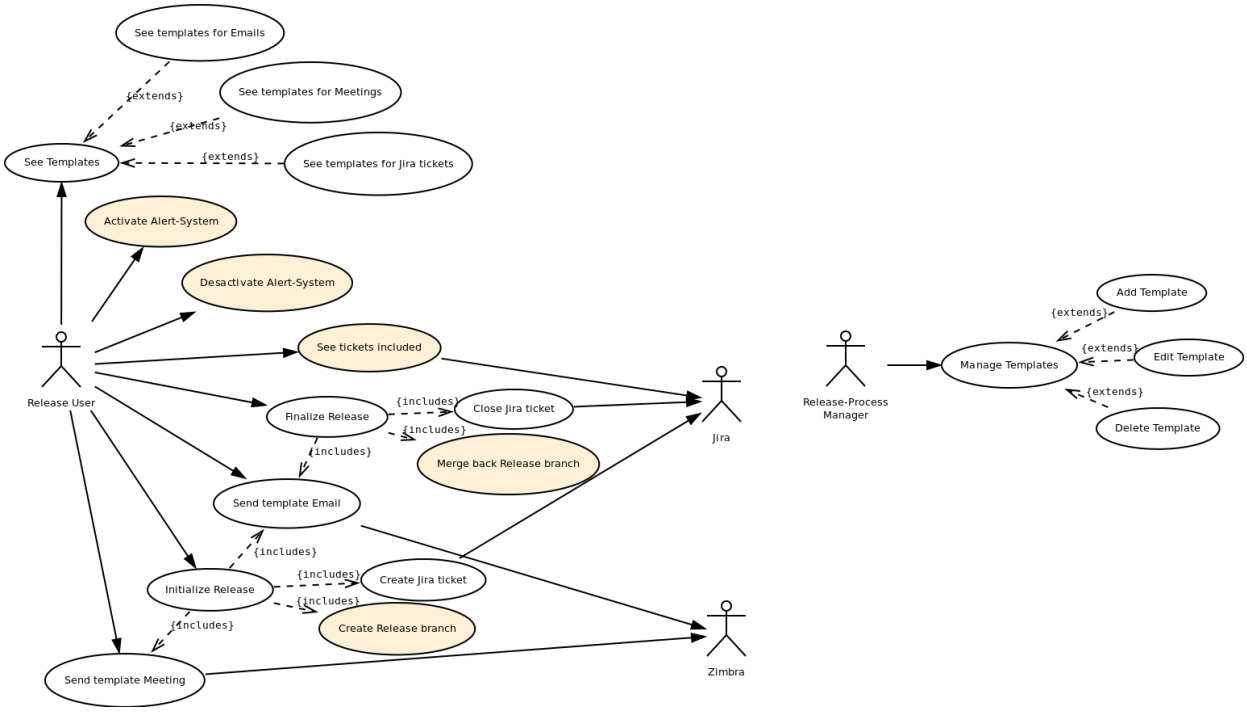


Ilustración 14. Modelo de Casos de Uso - tercer grupo

8.3.3. *Especificación*

Se ha realizado la especificación de los casos de uso considerados como los más relevantes para la arquitectura del sistema. De todas formas, aún sin haber especificado todos los casos de uso, la conclusión final es que este proceso convella una gran cantidad de tiempo y en la gran mayoría de las ocasiones los beneficios no compensan el tiempo invertido.

Si bien algunos requisitos y aspectos escondidos han salido a la luz en esta fase y se ha aclarado el comportamiento de algunas funcionalidades, esto sólo se ha obtenido en Casos de Uso puntuales.

Por ello, es recomendable escoger con gran cuidado aquellos casos de uso a especificar ya que podemos sacar grandes beneficios en caso de haber escogido los indicados o invertir tiempo en formalidades que no ofrecen luz sobre los requisitos en caso de escoger casos de uso poco relevantes. La selección depende en gran medida de la intuición y experiencia del especificador de casos de uso y/o persona responsable del proyecto.

La especificación en este caso se ha llevado a cabo de manera ágil y sencilla, sin incorporar todos los elementos y componentes que intervienen en una especificación formal de un caso de uso. Según sugiere el PUD, por ejemplo, la especificación de un caso de uso incluye una descripción formal y distintos diagramas de secuencia, estado y actividades. En este proyecto solamente se ha realizado una descripción formal, utilizando un template estándar. La razón de esta reducción de elementos implicados en la especificación, como se ha dicho, es agilizar el proceso y no invertir demasiado tiempo en formalidades. Se utilizan los elementos de Ingeniería que ayuden a comprender el sistema, especificarlo y propiciar una buena comunicación sin interferir con los criterios de agilidad deseados. Una vez más, se utilizan los componentes de las distintas Metodologías que aporten algún valor en beneficio del proyecto y según encajen con el procedimiento propio elegido por el responsable del proyecto.

A continuación se muestran los casos de uso seleccionados con su breve especificación.

Marcar punto de la Checklist

Use Case Marcar punto de la Checklist
Description El usuario quiere marcar un punto de la Checklist.
Precondition La Checklist es visible junto con todos sus checkpoints. El checkpoint no se encuentra aún marcado (Estado = “pendiente”). El usuario está loggeado en el sistema.
Primary flow 1. El usuario marca un punto de la Checklist. 2. El estado del punto seleccionado se cambia a “realizado”. 3. Se salva el nuevo estado del punto en base de datos. 4. Se ejecuta el Caso de Uso “Track Action” para dejar constancia de la acción realizada.
Postcondition La Checklist es visible junto con todos sus checkpoints. El estado del punto seleccionado es “realizado”.
Alternative flow
Exceptions Error intentando salvar cambios en la base de datos.
Includes Track Action

Añadir Checkpoint

Use Case Añadir Checkpoint
Description El usuario quiere añadir un nuevo punto a una existente DailyChecklist. Extiende de “Administrar Checkpoint”.
Precondition El usuario se encuentra en el área de edición de una DailyChecklist. Una DailyChecklist se encuentra seleccionada. El usuario está loggeado en el sistema. El usuario tiene privilegios administrativos (rol = “release-process manager”)
Primary flow 1. El usuario selecciona la opción para añadir Checkpoint. 2. Se muestra al usuario un cuadro de diálogo para introducir la información necesaria. 3. El usuario introduce la información requerida. 4. Se crea un nuevo Checkpoint con la información introducida y estado “pendiente”: 5. El Checkpoint se asocia a la DailyChecklist pre-seleccionada. 6. Se salvan los cambios en base de datos.
Postcondition El usuario se encuentra en el área de edición de una DailyChecklist.

La misma DailyChecklist se encuentra aún como seleccionada.
Alternative flow 4. Si la información introducida no es válida, se vuelve al punto 2.
Exceptions Error durante la interacción con base de datos.
Includes

Ver estado de servidores

Use Case Ver estado de servidores
Description El usuario quiere ver el estado actual de todos los servidores de stage y producción.
Precondition El usuario se encuentra en el área de Monitoreo.
Primary flow 1. Se busca la url de cada servidor de Stage. 2. Se busca la url de cada servidor de Producción. 3. Se ejecuta el Caso de Uso “Actualizar estado de servidor” para cada servidor. 4. Se renderiza la información referente a cada servidor en un formato apropiado.
Postcondition Estado de los Servidores actualizado. Estado de los Servidores visible.
Alternative flow -
Exceptions Excepción durante la conexión al Servidor.
Includes Actualizar estado de servidor.

Crear ReleasePage

Use Case Crear ReleasePage
Description El usuario quiere abrir una nueva ReleasePage donde dejar constancia de la ejecución y estado del proceso de Release.
Precondition El usuario está loggeado en el sistema. No hay ninguna Release activa.
Primary flow 1. Se muestra un cuadro de diálogo al usuario para que introduzca información básica sobre la nueva Release (nombre del manager, nombre del asistente, etc.) 2. El usuario introduce la información requerida. 3. Se crea una nueva ReleasePage con la información introducida por el usuario

anteriormente. 4. La ReleasePage es salvada en base de datos.
Postcondition
Alternative flow 3. Si la información introducida por el usuario no es válida, se vuelve al punto 1.
Exceptions Excepción durante la interacción con la base de datos.
Includes

Comenzar Release

Use Case Comenzar Release
Description El usuario quiere abrir una nueva Release y ejecutar un grupo de acciones automatizadas asociadas a este proceso de comienzo.
Precondition El usuario está loggeado en el sistema. No hay ninguna Release activa-abierta.
Primary flow 5.4 Se ejecuta el Caso de Uso “Crear ReleasePage”. 5.5 Se ejecuta el Caso de Uso “Crear Jira-ticket”. 5.6 Se ejecuta el Caso de Uso “Enviar template-email”. 5.7 Se ejecuta el Caso de Uso “Enviar template-meeting”. 5.8 Se marca la Release como activa. 5.9 Se ejecuta el Caso de Uso “Activar sistema de alertas”.
Postcondition El sistema de alertas está activado. Una Release está ahora activa-abierta.
Alternative flow
Exceptions
Includes Crear ReleasePage Crear Jira-ticket Enviar template-email Enviar template-meeting Activar sistema de alertas

8.4.Modelo de Negocio

Los diagramas para el modelo de negocio han sido divididos en distintos grupos para facilitar la representación de los componentes, la comprensión y garantizar la simplicidad. Cada grupo de diagramas está creado de forma que los elementos que lo componen tengan cohesión entre sí y respondan a un dominio del sistema que pueda ser analizado de forma independiente al resto.

A pesar de dividir los diagramas en grupos independientes, a veces es inevitable incluir en algunos grupos elementos que corresponden de forma natural a otros grupos, pues su relación es indispensable para la comprensión general del sistema y el modelo de negocio. Para solucionar esta situación se ha obtado por realizar una diferenciación mediante colores, de forma que todos aquellos elementos representados en color rosa oscuro son componentes que pertencen de forma natural a otro grupo y se encuentran duplicados en el modelo de negocio general.

Además, se ha marcado con color marrón claro a las clases de dominio que se han representado en el modelo de negocio. Como se ha dicho anteriormente, para garantizar la agilidad de los modelos no todos los elementos y relaciones son representadas en los diagramas, solamente aquellos más importantes para la comprensión general del sistema.

De forma general se explicarán las principales decisiones de diseño aplicadas durante la elaboración del modelo de negocio y seguidamente se verá de forma resumida las características más relevantes de cada uno de los diagramas de modelo de negocio generados.

8.4.1. Principales decisiones de diseño

El modelado del negocio se ha realizado teniendo en cuenta todos los requisitos funcionales y no funcionales que deben ser implementados y además, los criterios y principios fundamentales de la Ingeniería de Software para desarrollar un producto basado en buenas prácticas y estándares y por lo tanto, que cumpla con los criterios de calidad esperados.

A continuación se enumeran las distintas deciciones tomadas durante el modelado del negocio de la aplicación y su diseño. Algunas de ellas corresponden a principios generales de Ingeniería de Software, otras a buenas prácticas y algunas de ellas han sido aplicadas bajo criterio personal. Las razones para la aplicación de cada una de estas decisiones son expuestas en los puntos indicados.

KISS Principle (“Keep it simple and stupid”)

Este principio es uno de los clásicos a aplicar durante cualquier etapa del ciclo de vida de un software. Si además el software se está desarrollando bajo metodologías de desarrollo ágil, este principio cobra especial importancia pues todo aquello que es simple es fácil de entender, facilita la comunicación, posiblemente la implementación y las pruebas y, finalmente, contribuye a la agilidad con la que se desarrolle la aplicación.

Existen muchas opciones válidas para diseñar una misma funcionalidad en un sistema, ya sea en el modelado del negocio, del dominio o la creación del diseño de la arquitectura. La decisión es realizar el diseño más sencillo que permita obtener los resultados deseados, enfocándonos en el problema actual que queremos resolver y sin tener demasiado en consideración posibles escenarios que quizás nunca lleguen a suceder (aplicación directa de los principios generales de XP).

Open-close Principle

Se ha aplicado otro principio fundamental de diseño software: “Cerrado a cambios, abierto a extensiones”. Se pretende constuir una aplicación lo suficientemente flexible como para que sea posible su extensión en el futuro y los cambios necesarios puedan ser realizados de

manera sencilla y sin afectar al código existente que sea totalmente independiente de dichos cambios a realizar.

Por ello, este principio cobra fundamental importancia y se ha aplicado cada vez que ha sido posible para garantizar la construcción de una arquitectura robusta, con niveles de jerarquía que agrupen a los elementos con las mismas características y permitan la inclusión de nuevos elementos que sigan los patrones y diseños previamente establecidos.

Con esto se consigue mantener homogeneidad entre los elementos del sistema no sólo en las primeras etapas de desarrollo sino en el futuro, cuando se incorporen nuevas características y componentes. Es bien sabido que en el futuro se incorporarán nuevos Servicios y Componentes (como Templates), por lo que debemos mantenernos flexibles a este requisito de extensión y garantizar la estabilidad y cohesión de los elementos de la aplicación en el futuro.

Por ejemplo, todos los Servicios que conectan con alguna API se agruparán bajo la misma clase para garantizar homogeneidad en el modo de conexión con sistemas externos y permitir la inclusión de nuevos servicios que realicen dicha actividad. Además, se sabe que dichos componentes serán más propensos a cambios que el resto pues dependen de un servicio externo y deben cambiar si la API de conexión cambia (alto riesgo de cambio). Por ello, en estas clases debe mantenerse toda la lógica relativa y dependiente de conexiones extenas.

Single Responsibility Principle

Se ha intentado modelar y diseñar el sistema manteniendo al máximo posible criterios de Responsabilidad Única. Se pretende crear componentes que cumplan con una única labor, de forma que si dicho componente debe ser modificado, esto suceda solamente debido a una razón. Esto no se puede aplicar el 100% de las veces porque sencillamente en ocasiones este criterio perfecto escapa un poco de la realidad y hay casos en los que, por otro tipo de cuestions, un componente posee más de una responsabilidad.

Además de conseguir que un componente sólo sea modificado bajo una única causa, el principio de responsabilidad única permite a los desarrolladores y arquitectos del sistema saber con exactitud dónde colocar una función determinada o llevar a cabo la implementación de una nueva característica, pues si cada componente tiene una responsabilidad específica y bien documentada es realemente sencillo saber **quién hace qué** dentro de la aplicación.

Otras razones para la aplicación de este principio son la facilidad para testear, el encapsulamiento de información y la facilidad de trabajo con cada componente, por ejemplo, para crear Observers y Observables: es realmente sencillo configurar un componente como uno de estos tipos cuando solamente tiene una responsabilidad y no debemos preocuparnos de otros detalles.

Si diseñamos pensando en una responsabilidad principal corremos menos riesgo de tener acoplamiento en componentes individuales o de cometer grandes errores de diseño, pues cada componente tiene su propia razón de ser.

Aplicación del patrón MVP a las secciones

Cada sección tendrá la posibilidad de ser distinta a las demás y por lo tanto será manejada de forma independiente por un Presenter propio. Esto constituye un punto fundamental en la idea de ofrecer flexibilidad para futuras integraciones de servicios en el sistema.

8.4.2. Modelos

8.4.2.1. Especificaciones generales

- La máxima a la hora de realizar los diagramas de clases y el diseño general de la aplicación es diseñar guiados por responsabilidad ("Responsibility Driven Design"). Se diseña pensando en las responsabilidades finales de las clases e intenando que cada clase tenga la mínima cantidad de responsabilidades posibles, siempre manteniendo criterios de cohesión. De esta forma los cambios en la aplicación se pueden aislar completamente en pequeños componentes, la implementación y testeo se vuelven más sencillos y el riesgo de error al diseñar disminuye considerablemente pues las decisiones tomadas y clases creadas se encuentran dotadas de alta carga semántica.
- Los controladores se mantienen mínimos y sin acciones pesadas de lógica. Sus funcionalidades son la verificación de los parámetros de entrada, redirecciones, llamas a servicios para la ejecución de alguna funcionalidad lógica y renderizado de las distintas vistas con mensajes apropiados si fuera necesario.
- Toda la lógica de la aplicación se sitúa en los Servicios.
- Existe un tipo de Servicios especiales con la clave "DataAccessService" al final del nombre del servicio. Estos servicios se encargan de todas aquellas operaciones que requieren ejecutar peticiones a base de datos. En general, cada servicio de este tipo se encuentra asociado con una clase de dominio.
- Las funcionalidades que deben comunicarse con un servicio externo se encuentran generalmente agrupadas en una misma clase de servicio. Existe una clase de servicio para comunicación por cada servicio externo integrado. Su nombre suele incluir la clave "RequesterService" y el objetivo detrás de esta estrategia es centralizar las funcionalidades que se comunican con un servicio externo tanto como sea posible para garantizar un mayor control pues éstas tienen alto riesgo de cambio, ya que dependen en el modo de comunicación del servicio al que acceden.

8.4.2.2. Características y aspectos más relevantes

Áreas y Secciones

Cada área de la aplicación será totalmente independiente de las otras y tendrá su propio controlador general encargado de la visualización, el contenido inicial y las funcionalidades ofrecidas en dicha área. A su vez, si el área está compuesta por distintos grupos con dominios diferentes y se considera que es realmente extensa, se dividirá en distintas secciones. De esta forma el contenido será agrupado de una forma más intuitiva y además se facilita la carga de cada sección de forma independiente mediante AJAX ofreciendo 2 ventajas fundamentales: los errores de una sección no afectan a las demás y el tiempo de carga de una sección no afecta a las demás, lo que se traduce en que el usuario no deba esperar a que la página completa esté cargada para poder ver contenido de interés.

Cada sección a su vez consta de un controlador principal y algún servicio específico en caso de ser necesito para desarrollar las acciones lógicas.

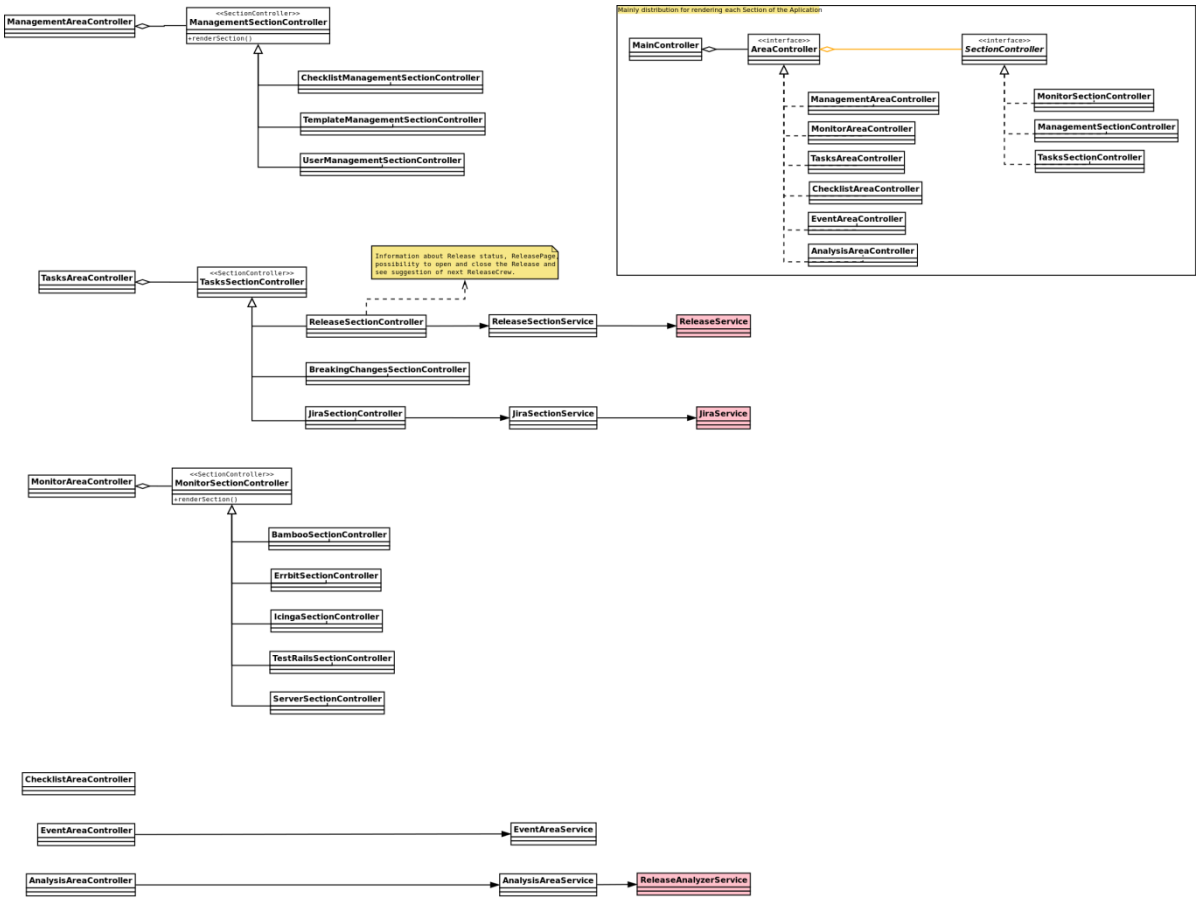


Ilustración 15.Modelo del Negocio - Áreas y Secciones

Checklist

Este diagrama es bastante básico y muestra cómo se implementarán los distintos componentes involucrados en el manejo de una Checklist. Lo más relevante de este diagrama es el seguimiento de las convenciones de Grails sobre las clases de dominio: se crea un servicio y un controlador para manejar cada clase de dominio. En este caso se puede observar además la presencia de los servicios “DataAccess” para cada clase de dominio.

Otro detalle importante es la división del controlador propuesto teóricamente por Grails para una clase de dominio en 2 controladores distintos: uno se encargará de las funciones de gestión y otro del resto. Por supuesto, cada controlador tendrá un servicio propio con el objetivo de simplificar y reducir las líneas de código en una misma clase. Sin embargo, se mantiene sólo un servicio de acceso a base de datos por clase de dominio. La razón de ello es que se quiere centralizar las funcionalidades que se comunican con un servicio externo tanto como sea posible para garantizar una mejor organización.

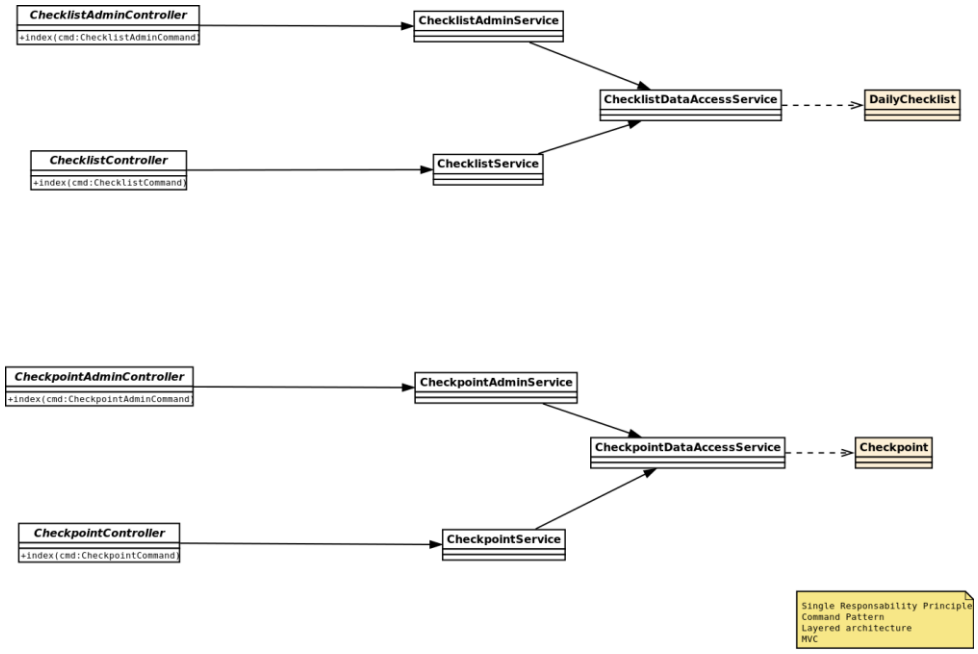


Ilustración 16. Modelo del Negocio – Checklist

Communicación y Tareas

Se muestra una vez más la idea de mantener un controlador y un servicio general por cada clase de dominio. Un comportamiento especial se refleja en la clase "ReleaseDataAccessService" que agrupa todas aquellas funcionalidades que acceden a base de datos para interacturar no sólo con Releases, sino con ReleaseTeams y ReleaseLocations. Se ha decidido mantener una única clase de acceso a base de datos en lugar de 3 distintas ya que los conceptos se encuentra estrechamente relacionados y no se necesitan demasiados métodos, por lo que el servicio sigue siendo legible y simple.

Por otra parte, algo interesante es el modo en que se maneja todo lo relacionado a los servicios externos integrados. Como se ha dicho anteriormente, para cada servicio externo se crea una clase "Requester" encargada de la comunicación con dicho servicio. Todas estas clases tienen por supuesto una base común que garantice la estabilidad y homogeneidad del sistema. Cada servicio externo es considerado realmente como un sistema externo de datos, similar a una base de datos desde el punto de vista abstracto de la aplicación. Los "Requesters" se encuentran al mismo nivel que las clases "DataAcess" que creamos para cada clase de dominio de la aplicación. Siguiendo la misma lógica, es necesario tener una clase de servicio general para cada servicio externo integrado y, cuando sea necesario, un controlador.

Finalmente, se aprecia el diseño aplicado al sistema de comunicación por emails de la aplicación. Esta parte del diagrama fue añadida en etapas posteriores de implementación pues en los momentos iniciales de análisis aún no se sabía exactamente cuáles eran los requerimientos ni que se añadiría el plugin "Mail" de Grails para realizar el envío de emails. La idea preconcebida era realizar el envío de emails mediante la integración con el servicio externo "Zimbra".

El plugin "Mail" añade al proyecto una clase llamada "MailService" con todos los métodos necesarios para enviar emails y provisiona a todos los controladores de la aplicación con un

método básico para realizar estas operaciones. Se ha decidido aislar el uso de estas funcionalidades y acceder a ellas desde un único punto de la aplicación que constituya la barrera entre la aplicación y dicho plugin. De esta forma si se produce algún cambio en el plugin sabemos que los cambios deben ser realizados en un único lugar y esto nos proporciona un mayor control por reducción de riesgos debido a dependencias externas.

La clase propia "EmailService" será la encarga de ofrecer todas las funcionalidades relacionadas con envíos de emails al resto de servicios de la aplicación. Esta clase utiliza un lenguaje propio del dominio del sistema y hace uso de la clase "MailService" para realizar el envío final de emails. Se ha decidido que la clase "EmailService" implemente distintas interfaces en dependencia de las necesidades de las clases cliente. Con ello, cada interfaz especifica métodos a ser implementados y, a pesar de que los métodos especificados por todas las interfaces son implementados en la misma clase, los clientes de cada interfaz trabajarán y verán solamente aquellos métodos de su interés, por lo tanto se garantiza encapsulación, limpieza de código y menos acoplamiento.

Dos estrategias fueron tenidas en cuenta para el sistema de emails:

- Creación de distintas interfaces e implementación en 1 clase final.
- Creación de distintas clases que implementen los métodos necesitados y extiendan de una misma clase abstracta que especifique detalles y tenga funcionalidades comunes, por la aplicación del patrón de diseño "Template".

Se ha decidido aplicar la primera estrategia por su sencillez y rapidez de implementación, porque los métodos a implementar no son complejos ya que la verdadera carga del envío de emails se encuentra implementada en la clase ofrecida por el plugin "Mail" y porque de esta forma establecemos un único punto de uso de este plugin para así aislarlo ya que constituye una dependencia externa y puede introducir riesgos y errores.

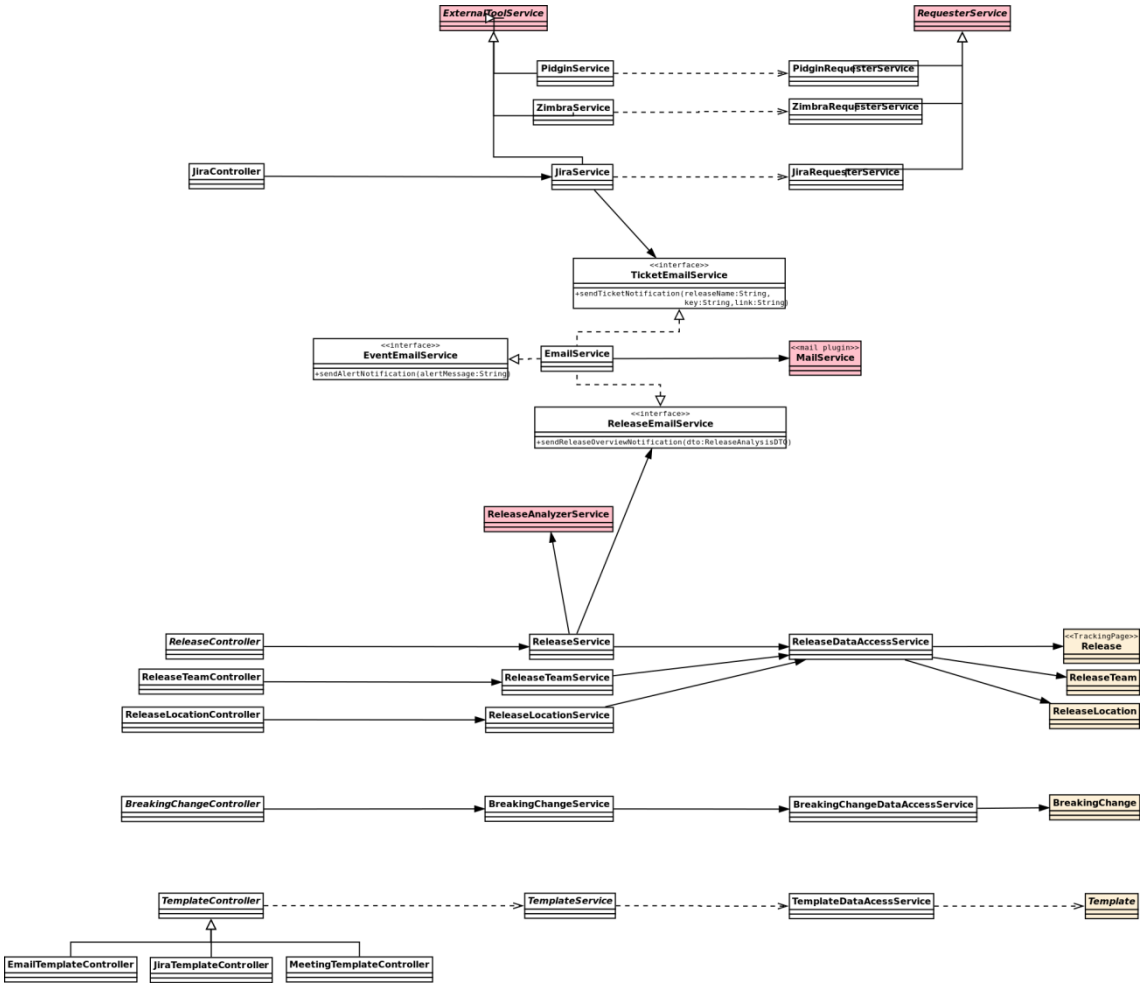


Ilustración 17. Modelo del Negocio - Comunicación y Tareas

Monitoreo

En este diagrama se muestra como se une el controlador del área de monitoreo con los controladores de las distintas secciones y los demás componentes necesarios para dar vida a esta parte de la aplicación.

Se observa que para cada controlador de sección existe un servicio encargado de realizar todas las operaciones lógicas necesarias para monitorear la información referente a un servicio externo. Cabe destacar que en el área de monitoreo se muestra información obtenida de servicios externos que han sido integrados. Por ello, además de los servicios creados para los controladores, existe un servicio "Requester" para cada uno de los servicios integrados por los motivos descritos previamente.

Se enfatiza el uso del patrón de diseño "Template" sobre la clase "ServerService". En la aplicación se consideran 2 tipos de servidores: servidores de stage y servidores de producción. Por ello es necesario tener servicios distintos para cada tipo, sin embargo ambos tienen características comunes y necesitan ciertos métodos por igual, con lo cual es perfectamente aplicable el patrón de diseño "Template" estableciendo que tenemos un servicio principal para los servidores con los métodos y atributos básicos, y luego cada servicio específico implementa aquellos métodos y detalles necesarios y aplicables a cada tipo específico de servidor.

Otro punto interesante es la distinción de los 2 tipos de servicios externos que se integran en la aplicación. Tenemos servicios de monitoreo y otros para realizar tareas automáticas y de comunicación. Ambos tipos extienden de las mismas clases base pues finalmente son todos servicios externos y queremos mantener el mismo modo de integración y manejo para todos ellos. Este diagrama se enfoca principalmente en aquellos relacionados con el monitoreo y el resto se hace más visible en el diagrama relacionado con Comunicación y Tareas.

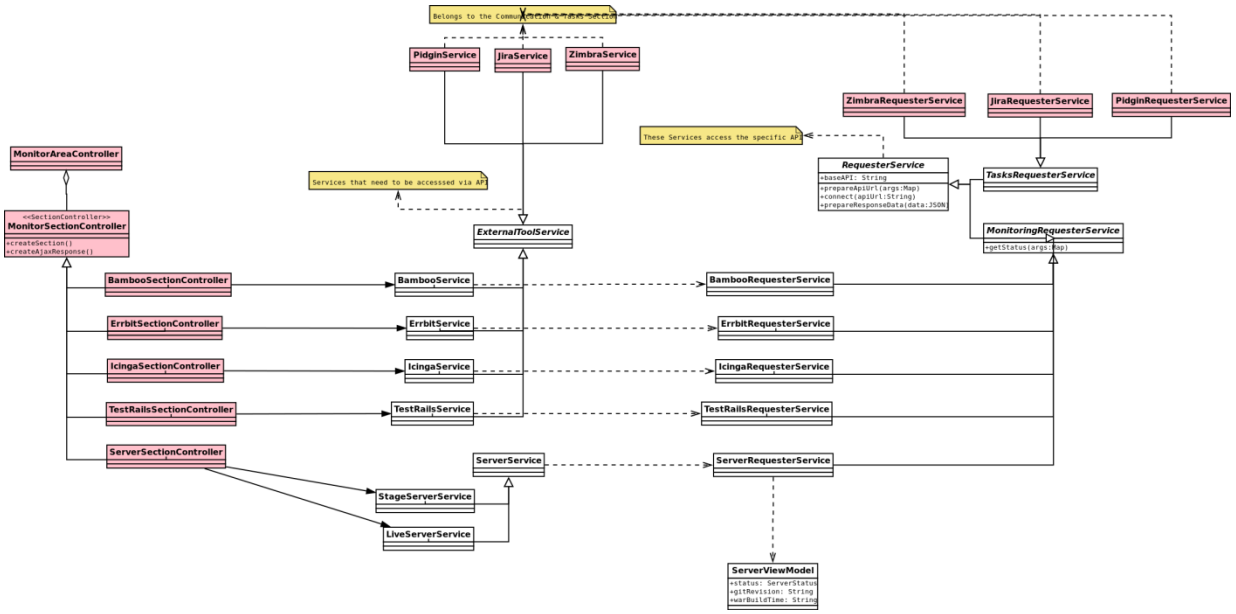


Ilustración 18. Modelo del Negocio – Monitoreo

Eventos (sistema de rastreo)

Para la generación de eventos e implementación del sistema de rastreo se ha realizado una mezcla de técnicas que termina mostrando un diagrama de clases totalmente sencillo y legible a la vez que potente. Se pretende mantener la máxima preestablecida de diseñar atendiendo a responsabilidades.

Se aplica el criterio seguido por los más innovadores desarrolladores “menos DRY (Don't Repeat Yourself principle) en favor de menos acoplamiento”. Cuando aplicamos DRY reducimos la cantidad de código redundante y duplicado y por lo tanto los cambios a una funcionalidad determinada sólo requieren modificación en un sitio específico. Sin embargo, cuando muchas funcionalidades distintas hacen uso de una misma funcionalidad independiente, el nivel de acoplamiento crece inmediatamente en la aplicación. Los cambios realizados en dicha funcionalidad básica utilizada por tantas otras funcionalidades afectarán por igual a todas ellas. Finalmente se puede decir que dichas funcionalidades que podrían ser totalmente independientes, no lo son ya que comparten código común.

Además, si utilizáramos los servicios de monitoreo existentes para generar eventos, podría darse el caso de que ocurra una excepción o fallo en la zona de código dedicada al manejo de eventos y de esta forma el área de monitoreo se vería afectada. Como funcionalidades distintas que son, deben permanecer independientes sin tener impacto la una sobre la otra.

Por ello en este sistema se ha duplicado mínimanente código referente a la petición de información sobre los servicios externos de monitoreo. Así se separan completamente las funcionalidades dedicadas al **renderizado de esta información en la UI** de las funcionalidades relativas a la **producción de eventos**.

La creación de la clase abstracta "EventService" agrupa todos los servicios de eventos y proporcionar el mismo tipo de interfaz general a los clientes de estos servicios. Esta clase se encuentra relacionada con la interfaz "eventEmailService" y permite el envío de notificaciones. Gracias a esta clase general abstracta el código es más limpio y la funcionalidad de notificación se ofrece a todos servicios de eventos futuros por defecto. En este caso se aplica el principio DRY de modo ventajoso.

Estrategias utilizadas en general:

- Se mantiene la misma estructura reflejada anteriormente para el manejo de clases de dominio: se crea un servicio de acceso a base de datos y un servicio general para cada clase de dominio.
- Se crea un nuevo tipo de servicios llamandos "Handlers" (manejadores) que se encargan de detectar si un evento es significativo y llamar al servicio oportuno para que ejecute las tareas establecidas para el tipo de evento específico. Existe un manejador para cada tipo de evento.
- Se crea un tipo de servicios cuya función es generar eventos, sean significantes o no, mediante la conexión a los distintos servicios externos. En este punto es donde se aplica el criterio "menos DRY en favor de menos acoplamiento" pues no se utilizan las funcionalidades previamente desarrolladas por los servicios generales, sino que se utilizan las funcionalidades más básicas de conexión para acceder a la información requerida.
- Se aplica el patrón de diseño "Observer" para establecer la comunicación independiente entre los generadores de eventos y los manejadores de eventos.
- Se utilizan los Jobs de Grails para ejecutar las tareas de los servicios generadores de eventos cada cierto tiempo mediante una expresión de cron.

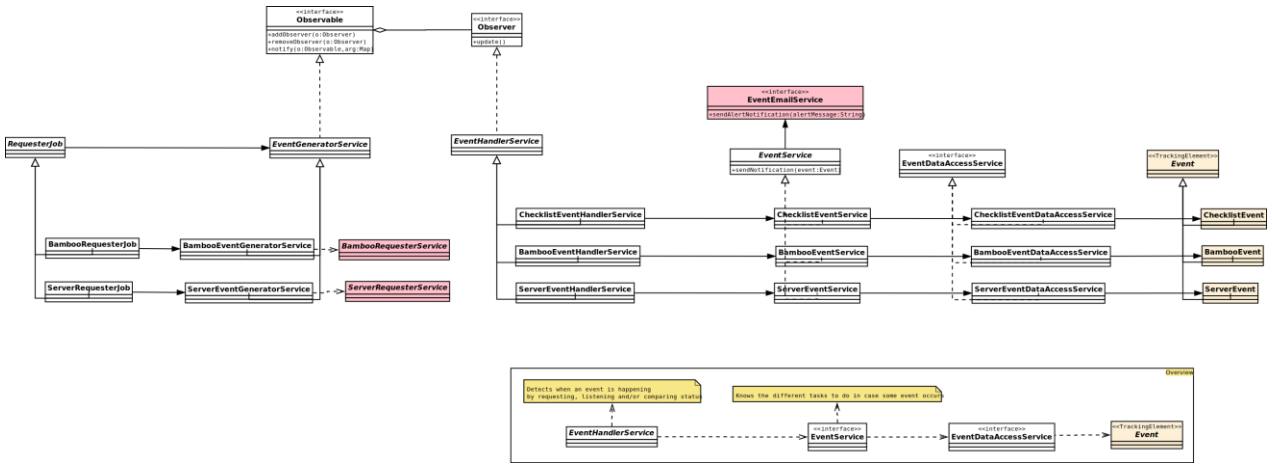


Ilustración 19. Modelo del Negocio - Eventos

8.5.Arquitectura

Se ha optado por aplicar el estilo arquitectónico en capas “Layered” ya que se ajusta perfectamente a los elementos modelados en los diagramas anteriores y debido a su conocida fama y uso como buen estilo arquitectónico para aplicaciones web.

Se considera que los servicios externos a integrar forman parte de la capa de recursos (“Resource Layer”) ya que proporciona utilidades e información externa a la aplicación y es necesario establecer mecanismos de comunicación y al mismo tiempo de reducción de riesgos para evitar que errores en dichos servicios afecten a la aplicación. El mismo criterio se aplica a los elementos contenidos en base de datos.

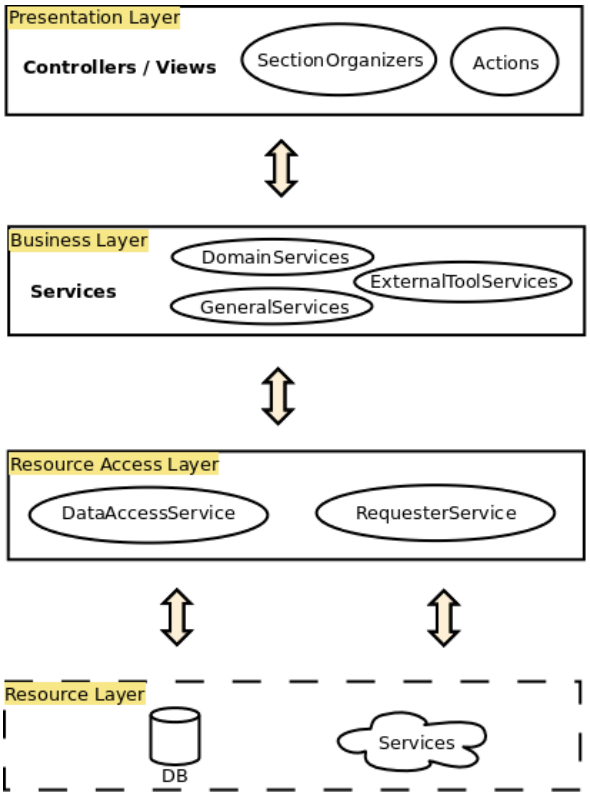


Ilustración 20. Estilo de arquitectura en capas

9. Capítulo 9: Implementación

9.1.Configuraciones básicas

9.1.1. Infraestructura

El proyecto va a ser desplegado en producción en una máquina virtual previamente creada por el equipo de Operaciones y donde se utilizará un servidor Tomcat y una base de datos Postgres local.

Durante etapas de desarrollo y pruebas, el proyecto fue ejecutado en el servidor Tomcat embebido proporcionado por Grails/IntelliJ y la base de datos utilizado corresponde a una creada al comienzo del desarrollo en un servidor interno, utilizando Postgres como gestor de base de datos.

Para realizar la configuración de la base de datos postgres se ha modificado el archivo del proyecto Grails "DataSource.groovy". Además se ha modificado la configuración por defecto del archivo del proyecto Grails "BuildConfig.groovy" para incluir el plugin para postgreSQL que realizará la conexión final.

9.1.2. Estructura del proyecto

El proyecto se encuentra compuesto por una aplicación base y múltiples plugins y librerías que sirven de ayuda e incorporan funcionalidades y estilos extra. La aplicación base a su vez se encuentra formada por distintos módulos internos, visibles en la estructura del proyecto. Esta distribución permitirá en el futuro dividir los distintos módulos en plugins independientes de una forma menos compleja. Los módulos del proyecto son los siguientes:

- Checklist
- Monitoring
- Management
- Tasks
- Events
- Analysis

Como se puede apreciar, cada módulo se corresponde con una de las áreas básicas de la aplicación. El objetivo final es crear áreas totalmente independientes que no tengan impacto en las demás. Las clases generales y básicas de la aplicación, como Interfaces generales de Servicios o contraladores y servicios generales, se mantendrán en el directorio base del proyecto y en el futuro formarán lo que será el Core de la aplicación.

9.2.Especificaciones técnicas

9.2.1. Hardware

Este proyecto ha sido desarrollado en una laptop LENOVO (Think-Pad) con las siguientes especificaciones técnicas:

- Sistema Operativo: Ubuntu 12.04 LTS (64 bits)
- Procesador: Intel Core i7 – 4700MQ @ 2.40GHz
- Memoria RAM: 8GB

9.2.2. Software

A continuación se listan los distintos elementos software básicos que han dado soporte a la creación del producto final y toda la documentación relativa. En el capítulo dedica a las herramientas y tecnologías utilizadas se describe en detalle cada una de ellas, su utilidad, motivos de selección y una breve historia. En este apartado solamente se espefican las versiones utilizadas junto con una pequeña descripción general acerca de la principal si se considera necesario.

Herramientas generales

- Paquete Microsoft Office 2010.
- Libre Office (3.5)
- Dia (0.97.2)
  Programa para dibujar diagramas estructurados, utilizado para crear los distintos diagramas UML (http://live.gnome.org/Dia)
- Mozilla Firefox (27.0.1)
  - Firebug (1.12.8)
  - TamperData (11.0.1)
  - HttpRequest (1.0.5)
- IntelliJ IDEA 12.1.1
- Postgres 9.2
- PgAdmin III (v1.14.0)

Dependencias directas del proyecto

- JDK 1.6 (1.6.0_30)
- Grails 2.2.4
- Groovy 2
- Foundation5 (5.2.1)
  - HTML 5
  - CSS 3
- jQuery (1.8.3)
- Mail plugin (1.0.5)
- Calendar plugin (1.2.1)
- Noty plugin (v2.2.2)
- Stash (v2.12.13)
- Famfamfam (1.0.1)
- Quartz plugin (1.0.1)
- Hibernate y Tomcat de acuerdo a la versión de Grails.
- Cache plugin (1.0.1)
- Resources plugin (1.2)
- HtmlUnit library (2.15)

Software externo

Los descritos en esta sección son aquellos softwares que constituyen servicios externos con los que la aplicación interactúa y se comunica. Es necesario mencionar las distintas versiones utilizadas ya que la comunicación se establece mediante API y esta puede ser completamente distinta en dependencia de la versión del servicio al que conectamos. Se prevé que estas

versiones cambien y por ello habrá que proporcionar un mantenimiento activo a la aplicación final para adaptar el modo de comunicación a los cambios introducidos por nuevas versiones. Sin embargo, a continuación se listan las versiones actuales en el momento de la entrega de este proyecto:

- Bamboo (v4.4.4)
- Jira (v5.2)
- Errbit (v0.2)

9.3.Iteraciones desarrolladas

El proyecto se ha dividido en 3 grandes iteraciones donde cada una debe conseguir ciertos resultados. Desde el comienzo de la aplicación se previó la posible necesidad de una última iteración extra, también descrita posteriormente. Esta última iteración es llamada "extra" y no de forma numérica para marcar la diferencia entre ambas, pues esta iteración no estaba completamente planificada.

9.3.1. Primera iteración

Se genera la Lista de Requisitos a incluir en la primera iteración funcional. Estos son los requisitos en los que se trabajará en la siguiente iteración y para los que se debe realizar el setup básico en mayor profundidad.

Se genera una aplicación que contiene las configuraciones básicas del proyecto con la implementación del esqueleto de la Arquitectura del Software y contiene los elementos esenciales de todo el proyecto y la especificación de los métodos y atributos de aquellos relacionados con el primer gran grupo de requisitos a incluir.

Todas las configuraciones se han realizado previamente, de modo que la aplicación pueda arrancar aunque aún no se haya implementado ninguna funcionalidad. La base de datos y su conexión se han creado y configurado, el repositorio GIT se ha creado y el sistema de "ramas" se ha pensado, de modo que ya hemos creado las primeras ramas básicas y generales de la aplicación y se ha establecido el formato a seguir para crear nuevas ramas.

Toda la documentación previa se encuentra accesible en un formato apropiado. Los elementos generados en este punto son los siguientes:

- Lista de Requisitos (funcionales y no funcionales)
- Modelo de Dominio general
- Modelo de Negocio general
- Modelo de Análisis general
- Diseño de la Arquitectura
- Lista de Requisitos a incluir en la primera iteración funcional

9.3.2. Segunda iteración

Se incluye el primer grupo de requisitos. La aplicación ofrece las funcionalidades básicas más priorizadas y la interfaz de usuario se corresponde con los Sketches básicos generados en etapas tempranas del proyecto.

El cliente puede en este momento evaluar el producto y tomar deciciones sobre el mismo. Asimismo, se puede generar una evaluación sobre la velocidad de trabajo, la calidad y la planificación del proyecto. Se observa el estado general del proyecto y caso necesario se toman deciciones que afecten de forma positiva el desarrollo del proyecto basadas en las pruebas que proporcionan esta primera iteración funcional.

En este momento se realiza una re-estimación del Plan de Proyecto si fuera necesaria y se realiza una re-planificación y organización de la Lista de Requisitos principal (Backlog) atendiendo a las nuevas necesidades surgidas durante la realización de esta segunda iteración, a las nuevas prioridades que el cliente establezca y a la información colectada durante este tiempo de desarrollo.

Se crea la Lista de Requisitos a incluir en la segunda iteración funcional.

9.3.3. Tercera iteración

En esta iteración se ofrece el grupo final de requisitos seleccionados por los clientes y se incluyen todos los arreglos señalados por los propios clientes para obtener una versión final de la aplicación que cumpla con las expectativas iniciales y satisfaga criterios de calidad.

Las funcionalidades incluidas en esta iteración son más avanzadas y se apoyan sobre las funcionalidades construidas previamente. Utilizando la aplicación con los requisitos implementados para la segunda iteración el usuario podía hacer uso de funcionalidades útiles, sin embargo la aplicación se sentía incompleta. En este momento, se hace uso de toda la configuración y estructura preestablecida y se trabaja con mayor velocidad, implementando funcionalidades más complejas y a un nivel más abstracto, a la vez que se ha incluido la integración con otros servicios de interés.

Esta es la iteración necesaria para ofrecer una versión operativa, útil y completa al usuario de la aplicación deseada en sus inicios.

9.3.4. Iteración extra

Se ha dedicado tiempo a realizar una última iteración extra para reforzar la aplicación. Después de mostrar la versión resultante de la tercera iteración a los clientes se ha podido colectar feedbacks referentes a funcionalidades y cambios deseadas, y además se ha encontrado algunos errores funcionales que debía ser corregidos antes de realizar la entrega final de la aplicación.

Acorde al Plan global del proyecto, esta iteración estaba pensada para ser realizada en las últimas semanas del presupuesto temporal en caso de ser necesaria. Finalmente, ha sido necesario llevarla a cabo y se han añadido algunas funcionalidades pequeñas, correcciones de funcionalidades existentes, correcciones en el estilo de la Interfaz de Usuario y sobretodo, se ha utilizado esta iteración para testear a fondo toda la aplicación.

10. Capítulo 10: Testeo y Validación

10.1. Testeo de desarrollador

Como parte del método "CTF" aplicado, se ha realizado un testeo profundo de cada una de las funcionalidades implementadas. Se ha elegido aplicar CTF a dominios, es decir, a grupos de funcionalidades que compartan un dominio entre sí, siempre y cuando dichas funcionalidades no hubieran generado demasiadas líneas de código. En el caso de implementar funcionalidades con una gran extensión de codigo, CTF se ha aplicado solamente a dicha funcionalidad con el objetivo de garantizar la calidad final de la aplicación.

Este testeo se ha realizado desde el lado del desarrollador, probando la funcionalidad en cuestión, cada posible escenario al que se debiera enfrentar en el futuro y las conexiones a los distintos sistemas externos con los que se interactúa si se diera el caso.

Este proceso de testeo sirve para validar la funcionalidad en cuestión y para detectar los distintos problemas existintes en etapas muy tempranas de la implementación, pues el método CTF se suele aplicar con no más de 1 semana de retraso sobre la implementación de la funcionalidad a testear.

De forma general, antes de finalizar una gran iteración todas las funcionalidades incluidas deben haber sido testeadas previamente. El proceso CTF puede ser aplicado varias veces dentro del tiempo válido de una iteración.

10.2. Retroalimentación de usuarios finales

Otro medio para testear la aplicación y la implementación de los distintos requisitos ha sido la retroalimentación por parte de los usuarios finales de la aplicación.

Después de la realización de cada Sprint (semanales/quincenales) se ha realizado una reunión con un grupo reducido de usuarios finales de la aplicación que han seguido la realización del proyecto desde sus comienzos y han sido usados, mayoritariamente, para detectar requisitos y validarlos o modificarlos, gracias a la retroalimentación proporcionada.

En cada una de las reuniones se ha expuesto cuáles han sido las funcionalidades incluidas y se ha mostrado una versión operativa de la aplicación, al mismo tiempo que se ha explicado cómo se utilizan dichas funcionalidades y cuáles son las ventajas y propósitos de su uso. Una vez los usuarios finales han observado la presentación del producto, se procede a comentar distintos aspectos de la aplicación, valorar las funcionalidades y su utilidad, expresar opiniones de forma abierta y generar críticas, tanto positivas como negativas, que permiten anotar futuras mejoras y modificaciones a realizar, descubrir la posición del usuario respecto a la aplicación y finalmente, validar las funcionalidades creadas.

Se podría decir que los requisitos se validan gracias a un prototipado evolucionario de la aplicación. Este término es habitual cuando se aplican metodologías de desarrollo ágil basadas en iteraciones, pues cada iteración constituye por sí misma un prototipo de la aplicación, sólo que se define como avanzado y evolucionario ya que incluye funcionalidades reales y responde a requisitos de usuario completados.

10.3. Retroalimentación del cliente

Del mismo modo, se ha llevado a cabo distintas reuniones de validación de requisitos con los clientes del proyecto. Estas reuniones sin embargo han sido menos frecuentes y en total se han realizado 2 reuniones cuyos objetivos han sido **validar los requisitos implementados** y **mostrar el progreso del proyecto**, y una 1 reunión inicial para mostrar el procedimiento y metodología de desarrollo y gestión del proyecto a seguir así como los objetivos y funcionalidades a alcanzar.

En estas reuniones se ha seguido el mismo procedimiento de las reuniones anteriores. Se ha realizado una presentación de la aplicación, las nuevas funcionalidades y requisitos incluidos, el modo de uso y los motivos que han conducido a un diseño específico o a distintos cambios realizados. Después de la presentación, cada cliente es libre de expresar su opinión, sugerencias y críticas y finalmente, de validar o no las funcionalidades incluidas en la nueva entrega que ha sido presentada.

Nuevamente, se está usando la técnica de validación de requisitos por prototipado evolucionario.

Se han realizado 3 reuniones a lo largo del desarrollo del proyecto. Las 2 últimas tenían como objetivo validar la implementación de las distintas funcionalides de la aplicación. La primera de las reuniones de validación se realizó sobre a mediados del desarrollo con el principal objetivo de tener un acercamiento con los clientes en etapas tempranas del desarrollo del proyecto, conocer su posición, opinión y grado de satisfacción con respecto al progreso del mismo y validar los requisitos implementados hasta el momento. La segunda reunión de validación tuvo lugar al finalizar el proyecto con el objetivo de mostrar a los clientes el producto obtenido y todas las funcionalidades incorporadas. En este momento el cliente puede apreciar un mayor número de funcionalidades desarrolladas y obtiene una idea más completa acerca del producto y su finalidad. Asimismo, en este momento el cliente valida la implementación de los requisitos, dando a conocer su grado de satisfacción con el producto entregado.

Esta última reunión se considera la validación definitiva de este Trabajo de Fin de Grado, ya que como se ha dicho anteriormente, este proyecto se ha desarrollado por iteraciones y existen muchísimos requisitos que quedan por implementar en futuras iteraciones. En el momento de la última reunión con el cliente se tiene una versión operativa y final de la aplicación desarrollada, que constituye la aplicación desarrollada bajo los términos de este TFG. Por medio de la presentación de la aplicación, la explicación de las distintas funcionalidades, su uso, sus objetivos y beneficios, y una demostración en vivo de la propia aplicación y todas sus funcionalidades, el cliente ha validado y aceptado el producto final.

10.4. Pruebas de seguimiento en vivo

En etapas finales del desarrollo se ha realizado un conjunto de pruebas de validación y ejecución “en vivo” con el objetivo de comprobar las distintas funcionalidades implementadas, detectar posibles errores y probar el funcionamiento global de la aplicación en entornos similares al que será el entorno de producción de la aplicación.

Además, se ha buscado la validación y aceptación de las distintas herramientas ofrecidas por la aplicación por parte de aquellos usuarios que la utilizarán en el futuro: el equipo de Release.

En las 2 últimas Releases que se han llevado a cabo en la empresa se ha realizado un seguimiento en vivo mediante el uso de la aplicación desarrollada, haciendo uso de todas las funcionalidades que ofrece como si la aplicación se encontrara ya en producción. De esta forma se puede observar si la aplicación responde finalmente a las necesidades de los usuarios que llevan a cabo los procesos de Release y por lo tanto validamos los requisitos iniciales que dieron lugar a la creación de este proyecto.

Además, se obtiene una retroalimentación valiosísima de mano de los usuarios finales que en ese preciso momento forman parte del equipo de Release y cuyas necesidades de organización y comunicación cubre la aplicación desarrollada.

Para realizar este proceso de prueba, se ha ejecutado la aplicación en una máquina distinta a la máquina de desarollo de forma constante e ininterrumpida durante todo el proceso de Release, para cada una de las Releases testeadas. Se ha hablado con los integrantes del equipo de Release y se les ha dado acceso a la aplicación en ejecución para su utilización. Como parte de este procedimiento ha sido necesario realizar una introducción al uso de la aplicación para todas aquellas personas implicadas. De forma personal también se ha hecho un seguimiento cercano de todo lo acontencido y el estado del sistema bajo prueba. Durante estos períodos se ha podido detectar fallos y/o puntos de mejora que han sido reparados inmediatamente o se han agregado a la Lista de Requisitos a implementar y quedan recogidos en este TFG como mejoras de futuro. Al finalizar estos periodos se ha obtenido una información muy valiosa por parte de aquellos usuarios finales que han testeado la aplicación mediante su uso durante los procesos de Release. Todas las sugerencias obtenidas han sido tenidas en cuenta y este proceso ha servido para validar las funcionalidades implementadas.

Uno de los mayores beneficios de este paso de prueba es la ejecución de la aplicación en un entorno distinto al entorno de desarrollo y su ejecución prolongada. Esto sirve para testear y validar a su vez elementos hardware y de “performance” que pudieran pasar desapercibidos y que sean sólo visibles a través de este tipo de pruebas.

10.5. Pruebas en distintos entornos de desarrollo

Para realizar este tipo de pruebas se ha obtado por distintas estrategias que permitan cubrir el máximo posible de escenarios y por lo tanto tener una mayor cobertura de pruebas. Se ha ejecutado y probado la aplicación en distintos entornos de desarrollo para garantizar la completitud del proyecto y su estabilidad en distintas máquinas.

10.5.1. Personal

De forma personal, se ha ejecutado el proyecto en el portátil personal (que no es el ordenador de desarrollo del mismo), descargando el mismo desde el repositorio GIT y probando cuidadosamente cada una de las funcionalidades desde una red distinta a aquella utilizada en todo momento durante etapas de desarrollo y utilizando una conexión VPN para poder acceder a aquellos servicios que se han integrado de la red interna de la empresa.

10.5.2. Colectivo

Se ha pedido a personas claves que descarguen el proyecto en sus ordenadores de trabajo, lo ejecuten y lo prueben mínimamente. Por supuesto, este proceso ha sido realizado bajo supervisión.

10.5.3. Testeo en vivo por tiempo prolongado

Se ha realizado un testeo enfocado a las situaciones de ejecución real, donde se ha dejado la aplicación operativa de forma ininterrumpida por unos 4 días (descrito anteriormente). La aplicación ha sido ejecutada en una máquina totalmente ajena a aquella en la que se ha desarrollado.

10.5.4. Producción

Se ha desplegado y ejecutado la aplicación en el que será en entorno de producción con la finalidad de comprobar las distintas configuraciones y el comportamiento de la aplicación en el entorno de producción, atendiendo especialmente a los tiempos de respuesta bajo el acceso de forma concurrente de varios usuarios, los errores log generados y la estabilidad del sistema.

10.6. Prueba final en Producción

En la última semana de desarrollo del proyecto ha sido proporcionada una Máquina Virtual donde se ejecutará finalmente la aplicación cuando se considere oportuno ponerla en producción. No obstante y modo de testeo la aplicación ha sido desplegada en dicha Máquina Virtual para realizar pruebas de testeo finales en el entorno de producción.

Se considera que no hay riesgo alguno al realizar estas pruebas ya que la aplicación será visible solamente de forma interna y el hecho de estar en producción no provoca cambio alguno a las características previamente testeadas.

Para desplegar la aplicación en producción se ha creado por primera vez un war de la misma y se ha podido comprobar la validez del procedimiento de despliegue. Durante esta etapa se detectaron errores de configuración que fueron corregidos inmediatamente y que, sin esta prueba final, nunca hubieran sido detectados y por lo tanto no se podría jamás considerar que la aplicación está lista para ser utilizada en producción.

Una vez la aplicación se encontró opertaiva y visible para toda la red interna, se informó a personas claves acerca de la misma y se requirió de manera cordial un testeo final de las distintas funcionalidades con el fin de probar la respuesta del sistema en el entorno final de producción y bajo el acceso de distintos usuarios de forma concurrente. De forma personal también se llevó a cabo este proceso de prueba, verificando no sólo las funcionalidades sino las conexiones con los sistemas externos, tiempos de respuesta y errores de logs generados.

Una vez se ha desplegado, ejecutado y testeado la aplicación de forma existosa en el entorno de producción, se considera totalmente válida desde el punto de vista del desarrollador.

10.7. Notas finales

Se han realizado distintas pruebas de verificación y validación a lo largo del desarrollo de la aplicación para detectar distintos fallos, errores de especificación de requisitos y/o la aceptación de las funcionalidades del sistema. Sin embargo, se marcan como cruciales 2 puntos que han servido para validar la aplicación y sus funcionalidades completamente y de forma final:

Testeo en el entorno de producción

Esta última fase de testeo en el entorno de producción ha servido para verificar todas las configuraciones, conexiones externas y funcionalidades del sistema en un entorno y

condiciones totalmente distintas a las habituales durante la etapa de desarrollo de la aplicación. Además, se ha podido verfiicar asuntos relacionados a la concurrencia de usuarios y de performance por lo que se considera el último paso de testeo de la aplicación que verifica completamente sus funcionalidades.

Última reunión con el cliente

La presentación final del producto durante esta reunión ha servido para validar completamente todos los requisitos implementados en la versión final mostrada a los clientes debido a su aceptación del producto.

11. Capítulo 11: Interfaz de Usuario

11.1. Análisis y Requisitos

Se ha requerido una Interfaz de Usuario sencilla y amigable, fácil de utilizar e intuitiva, que ayude al usuario a desempeñar de un modo más ameno las distintas tareas relacionadas con la Release.

El diseño debe ser, por lo tanto, mínimo, sencillo y agradable. Por supuesto, también debe ofrecer flexibilidad a futuros cambios: visualización de nuevos Servicios incorporados, nuevas áreas de trabajo, etc.

El usuario debe saber en todo momento en que sitio de la aplicación se encuentra; asimismo debe ser capaz de acceder a las funcionalidades y áreas principales de la aplicación de forma directa.

11.2. Principales decisiones

Se ha cambiado el diseño que se tenía originalmente, donde se dividía la pantalla en los distintos cuadrantes principales, ya que este diseño no ofrece flexibilidad a la inclusión de nuevas áreas de trabajo. El nuevo diseño, sin embargo, ofrece una alta flexibilidad con su barra lateral a la vez que proporciona la sensación de ubicación buscada para que el usuario nunca se sienta perdido en la aplicación y siempre pueda acceder a las áreas principales de la misma.

11.2.1. Tecnologías seleccionadas

El framework frontend utilizado ha sido Foundation, en combinación con el plugin “noty” para realizar las notificaciones visuales al usuario, el paquete de íconos globalmente conocido “famfamfam” debido a su versatilidad y compatibilidad y la librería de Javascript jQuery con el uso reiterado de la tecnología AJAX. Además, se ha hecho uso de templates de foundation y las tecnologías de estilos utilizadas han sido HTML5 y CSS3.

Foundation ha sido seleccionado como framework frontend ya que ofrece una tecnología robusta y puntera hoy en día, además de un diseño limpio y mínimo con funcionalidades sencillas y fáciles de utilizar y comprender a la vez de potentes. Además, es uno de los pocos frameworks frontend que incorpora “responsive design”, con lo cual podremos adaptar la aplicación a todo tipo de monitores y dispositivos de visualización que puedan utilizar nuestros usuarios.

Además, se han utilizado otros plugins como Calendar y Noty, siendo este último un plugin puramente Javascript.

11.2.2. Colores

Se van a utilizar los colores que vienen por defecto con el plugin “foundation”: azul-cielo, blanco y gris. El color principal será el azul-cielo y esta combinación ha sido seleccionada ya que ofrece una sensación de tranquilidad y relajación para el usuario; algo que encaja perfectamente con el propósito fundamental de esta aplicación.

11.2.3. Diseño

Se mantiene la decisión de optar por un layout que sea visible siempre y contenga los links directos a las áreas principales con el objetivo de ofrecer al usuario sensación de seguridad y de ubicación en todo momento. De esta forma, tendremos una barra en el tope donde se colocará el logo de la aplicación y links para acceder el home-page y realizar el login/logout.

Además, colocaremos 4 paneles principales para acceder a cada una de las grandes áreas de la aplicación: Checklist, Monitoring, Communication & Tasks y Management.

La tercera área de la aplicación está reservada para la parte "móvil". Esta parte será el área de contenido principal.

11.2.4. Alertas a utilizar

Se usarán las alertas en la esquina superior derecha para mostrar errores o alertas que requieren la atención urgente del usuario.

Se utilizarán las alertas en la parte superior para mostrar notificaciones generales o información interesante para el usuario.

Se utilizarán las aletas en el medio de la pantalla para dejar constancia de acciones realizadas (creación de algún elemento, etc.).

11.3. Sketches básicos

Como parte de las primeras etapas del proyecto, se elaboraron los Sketches básicos de la aplicación para validar junto con los clientes y usuarios el diseño a implementar, criterios de usabilidad y obtener una idea general sobre el estilo final que mostrará la aplicación.

Nuevamente, no se han seguido técnicas avanzadas de diseño y prototipado con el fin de garantizar la agilidad del proyecto y para no invertir demasiado tiempo en la utilización de nuevas herramientas de diseño de Interfaz de Usuarios pues se sabe de antemano que los requisitos de Interfaz de Usuario no son prioritarios: se pretende ofrecer una herramienta para utilizar de forma interna que aporte funcionalidades de valor y un mínimo de criterios de usabilidad en la UI, sin muchos requisitos específicos a implementar. Por este motivo, los Stekches han sido creados de forma manual para obtener una primera impresión acerca de la UI antes de comenzar a implementar funcionalidades. No obstante, en cada iteración con la inclusión de nuevas funcionalidades, la UI generada, distribución de los elementos y estilos aplicados son revisados en conjunto con los stakeholders para recibir feedbacks y establecer nuevos requisitos de Interfaz de Usuario y/o aplicar correcciones en caso de ser necesario.

Nuevamente, estamos siendo guiados por criterios de metodologías ágiles que nos permiten mantenernos abiertos a cambios, mantener la estrecha relación con los stakeholders y producir software funcional de forma más rápida al no dedicar demasiado tiempo a aquello que consideramos no es de crucial importancia para nuestro proyecto.

Los Sketches básicos se encuentran como documento anexo en esta memoria.

12. Capítulo 12: Principales problemas encontrados

12.1. Duplicación de eventos

Uno de los más grandes problemas encontrados durante el desarrollo de la aplicación ha sido la duplicación de eventos en base de datos. Los eventos son generados mediante la combinación de Jobs de Grails, clases generadoras de eventos y handlers que se encargan de determinar si un evento es significante o no y continuar con la persistencia del nuevo evento en caso de que sea significante.

Luego de implementar dicho sistema generador de eventos, se ha detectado que en base de datos aparecen eventos duplicados, generados en un corto periodo de tiempo.

Este ha sido el bug más grande al que la aplicación se ha enfrentado. Se ha buscado una solución por distintas vías y mediante la comprobación de variados aspectos:

- Comprobación de la efectividad del patrón Observer: se ha comprobado que no se esté añadiendo más de un Observer a una clase en un momento dado y se ha incluido un método de prevención que lance una excepción en caso de que querramos añadir la misma clase como observer 2 veces al mismo elemento obsevable.
- Comprobación de los Cron-Job de Grails: se ha comprobado el funcionamiento de los jobs de Grails, que se ejecutan bajo una expresión de cron y son los encargados de desencadenar las acciones que generan los eventos. Se ha descubierto que los Jobs de Grails poseen un atributo que en este caso es necesario especificar: concurrent. Cuando dicho atributo se establece a falso se impide que más de una instancia del Job se ejecute al mismo tiempo.
- Se establece una comprobación en base de datos justo antes de salvar el nuevo evento para impedir salvar información duplicada.
- El último método accedido para salvar un evento, donde se hace la comprobación de base de datos para impedir duplicaciones, se ha establecido comosynchronized. Esta probablemente sea lo que más influya para evitar las molestas duplicaciones ya que, independientemente de la razón por la que se obtienen 2 eventos significantes que son el mismo, a la hora de salvar los eventos se realiza una comprobación en base de datos, y estas acciones se hacen de forma sincronizada por lo que se consideran atómicas.
- Evidentemente, cuando se salva un evento en base de datos se realiza flush instantáneo de la sesión para garantizar que en ese momento el estado de la base de datos es totalmente confiable para realizar las comprobaciones de duplicación mencionadas anteriormente.
- Se ha comprobado que el elemento tomado de base de datos para realizar la comprobación de duplicidad sea siempre el último evento almacenado (del mismo tipo del evento inspeccionado).

12.2. Manejo de peticiones AJAX

Grails proporciona un fácil manejo de peticiones AJAX, sin embargo se ha debido invertir bastante tiempo en comprender completamente dichas funcionalidades para usarlas de la mejor forma posible y tomando ventaja de todas las posibilidades que ofrecen.

De todas formas, el problema fundamental ha sido la inicialización al mundo de AJAX: la colección de información y la comprensión del modo en que las peticiones AJAX funcionan. Esta información ha sido necesaria para crear peticiones AJAX propias, más allá de las funcionalidades básicas que Grails ofrece. Ha sido necesario crear propias peticiones, mediante el uso de jQuery, para realizar el renderizado de cada sección en distintas áreas de la aplicación de forma independiente y de esta forma evitar largos tiempos de carga de las páginas en el navegador. Además, se garantiza que los fallos en unas secciones no afecten a otras. En adición, ha sido necesario trabajar con peticiones AJAX periódicas.

En conclusión, se ha necesitado bastante tiempo dedicado a AJAX debido a distintos problemas en la aplicación que necesitaban ser resueltos por el uso de esta tecnología y además debido a otros problemas generados por su propio uso.

12.3. Conflictos frontend

Funcionalidades del framework que no funcionan sin motivo aparente, estilos del framework en conflicto con estilos provinientes de plugins (como calendar), etc.

En repetidas ocasiones se ha tenido problemas debido a conflictos de frontend. En esta aplicación se utiliza un framework frontend que incluye estilos y funcionalidades, se hace uso de estilos y funcionalidades propias y además se incluyen plugins que incorporan funcionalidades javascript en algunas ocasiones y estilos propios para los elementos incorporados en otras.

Con toda esta variedad, es de esperar que algunos conflictos ocurran durante la etapa de implementación.

12.4. Modificación del framework

Por error y falta de experiencia se modificó la hoja de estilos del framework frontend incorporado, en etapas inciales de desarrollo. Evidentemente, esto no se puede realizar ya que los cambios realizados sólo son válidos en la máquina de desarrollo y no se quedan de forma permanente en el propio plugin.

El error fue detectado al integrar la rama de desarrollo de la primera integración con la rama de continuosIntegration. Una vez realizada la integración y haciendo uso de la rama continuosIntegration, se detectó rápidamente que los estilos añadidos al framework habían desaparecido. Es de destacar la utilidad del trabajo con sistemas de control de versiones y la integración continua de los cambios realizados en una rama general.

Dicho problema fue solucionado rápidamente ya que los cambios no eran demasiado radicales ni abundantes. Para las soluciones se usó la técnica de especificar estilos propios para un componente a un mayor nivel de especificación, utilizando ids por ejemplo.

12.5. Autenticación via API con token de seguridad

Para interactuar con la API de Errbit es necesario estar loggeado en el servicio y la API no ofrece ningún medio oficial para realizar el loggin en el sistema, por lo tanto es necesario realizar la autenticación mediante el crawling de la página. Para realizar la autenticación, es necesario proporcionar el token de seguridad que se asigna al cliente en la sesión de loggeo iniciada. Este es un modo no muy elegante para realizar la autenticación pero hasta el momento es la única vía existente debido al poco soporte que Errbit ofrece para la API.

Se ha dedicado bastante tiempo a esta autenticación para comprender completamente el modo en que funciona la autenticación por medio de un token de seguridad, para identificar el tiempo de validez de dicho token y las condiciones a mantener para que no expire y sobretodo para intentar encontrar otra forma más elegante y robusta de realizar dicha autenticación.

Finalmente, se ha realizado el crawling de la página para realizar la autenticación con el token de seguridad proporcionado por Errbit mediante el uso de un cliente de java (HttpClient), realizando 2 requests: una para obtener el token y otra (tipo POST) para realizar el loggin con la información requerida.

12.6. Problemas con la API de Jira para realizar algunas acciones

Para ejecutar acciones como la creación de un ticket y la transición de un estado a otro de los tickets (para cerrarlos) hubo muchísimos problemas debido a rechazos del servidor.

La API de Jira está perfectamente documentada y es simple. Las peticiones generadas fueron revisadas múltiples veces e información extra fue buscada en distintos blogs técnicos de internet y en el propio foro de Jira. Sin embargo, todo apuntaba a que las peticiones estaban correctamente elaboradas y por lo tanto nada debería fallar.

Sin tener idea alguna acerca de la causa de este extraño comportamiento, se decidió obviar de momento dichas funcionalidades y no dedicar mucho más tiempo a su investigación ya que se había invertido bastante tiempo hasta el momento. Para la creación de tickets se utilizó otra llamada de la API menos elegante y menos potente pero funcional y para las transiciones se perdió la esperanza de un futuro arreglo luego de encontrar un ticket del propio equipo de desarrollo de Jira, donde se comentaba exactamente dicho problema y además se decía que sería arreglado en una versión futura de la aplicación, no desplegada aún.

Durante la etapa de CTF para Jira, aplicada luego de incluir un gran grupo de funcionalidades, se detectó el error. Al detectarlo se supo de forma inmediata que ésta era la causa de los errores: para todas las peticiones API de Jira es necesario incluir la cookie obtenida. En las peticiones de tipo GET se estaba incluyendo la cookie, sin embargo en las peticiones de tipo POST no se realizaba. Evidentemente las funcionalidades afectadas hacían uso de peticiones POST.

Con este ejemplo se puede observar como un error de desarrollo totalmente básico y sencillo, no detectado durante el tiempo de implementación, puede ser fácilmente identificado durante etapas de limpieza de código. Además, se observa cómo la simplicidad, limpieza, autodocumentación y minimalidad del código ayuda a generar software de mayor calidad e identificar errores en etapas tempranas del desarrollo.

12.7. Cambios en la configuración de los servidores

Los servidores cms cambiaron su configuración y ahora no ofrecen las peticiones API de forma abierta (sin realizar un loggin) y no es posible obtener información sobre el estado de dichos servidores. Además, se comenzó a utilizar un certificado de autenticación después de haberse implementado la funcionalidad relacionada, lo cual implica que fue necesario un arreglo de dicha funcionalidad en etapas más avanzadas del desarrollo.

Esto demuestra la dependencia de la aplicación ante los servicios externos que integra y la razón fundamental por la que se ha intentado que cada componente sea totalmente independiente del resto y evitar de esta forma que los fallos de unas áreas afecten a otras.

En este proyecto se han implementado las funcionalidades bajo la perspectiva de que las dependencias externas pueden fallar y debemos estar preparados para responder de forma efectiva en esas situaciones.

12.8. Demasiados requisitos funcionales

Los procesos de planificación para cada iteración se vuelven mucho más complicados cuando tenemos una gran cantidad de requisitos para desarrollar. Además, es necesario negociar la prioridad y futura inclusión o no de estos requisitos con los clientes un mayor número de veces. Esto hace que los procesos de gestión del proyecto sean más críticos e importantes, por lo tanto mucha más atención debe ser prestada a los mismos y cada decisión relativa a los requisitos debe estar correctamente fundamentada.

Además, el tener demasiados requisitos hace difícil el cierre de la etapa de implementación en un momento dado: siempre queremos incluir algo más. Desde el punto de vista del TFG, donde nos debemos ajustar al presupuesto de tiempo establecido, es necesario establecer el punto en el que ya no se implementarán más requisitos y decidir cuáles quedarán fuera de esta primera entrega oficial. Todo ello indica grandes procesos de negociación y decisión.

Por último, destacar que muchos requisitos a implementar son un indicio de una aplicación grande a desarrollar. Es bien conocida la cantidad de trabajo y procesos que convella la implementación de un único requisito funcional, incluso trabajando con metodologías de desarrollo ágil: es necesario implementar, revisar y limpiar código, testear, pensar en posibles escenarios que rompan la aplicación, documentar cuando es requerido, negociar con el cliente, validar, etc.

Esta aplicación ha resultado ser mucho más grande de lo esperado al comienzo del proyecto. Ha sido crucial y de extremada ayuda la aplicación de criterios de agilidad sobre los requisitos. Primeramente, porque hubiera sido necesario invertir demasiado tiempo en ellos si hubiéramos seguido criterios clásicos de Ingeniería de Requisitos. En segundo lugar, porque la mayoría de ellos han cambiado con el tiempo y/o ha sido más fácil especificarlos en etapas más avanzadas, contando con un mayor conocimiento global del dominio del sistema. Finalmente, porque muchísimos requisitos más han surgido durante las etapas de desarrollo.

12.9. Poco tiempo dedicado al testeo del sistema

A pesar de haber dedicado tiempo a testear las funcionalidades implementadas, se considera que no ha sido suficiente y que la aplicación debe ser aún testeada de forma apropiada desde el punto de vista de desarrollo y Quality Assurance. Se garantiza que la aplicación cumple criterios de calidad mínimos, sin embargo cada funcionalidad implementada debería ser testeada completamente, prestando atención a los detalles y los escenarios extremos que puedan ocurrir para garantizar de esta forma una aplicación totalmente robusta.

Además, se considera necesario la creación de tests unitarios y de integración al menos para las funcionalidades más críticas de la aplicación. De esta forma, en el futuro se podrá detectar fácilmente cuando la inclusión de una nueva funcionalidad esté afectando a otra existente.

12.10. Trabajo con frontend en etapas iniciales

Uno de los principales problemas o punto que ha realentizado el desarrollo es el trabajo con la Interfaz de Usuario: frontend y sus tecnologías. Previamente a este proyecto no se ha tenido demasiada experiencia con el desarrollo frontend, por lo que cada vez que ha sido necesario

trabajar con alguna tecnología o funcionalidad frontend se ha invertido más tiempo de lo habitualmente dedicado para un trabajo proporcional de backend.

Esto ha constituido una excelente oportunidad para desarrollar conocimientos y adquirir experiencia en el mundo de desarrollo frontend, sin embargo se considera que debe ser reflejado como uno de los puntos en esta sección debido a la gran cantidad de tiempo invertido y a los múltiples problemas que el trabajo con frontend ha ocasionado. Algunos han sido:

- Un efecto de javascript que no funciona como se espera.
- Estilos que no ofrecen la vista deseada.
- Acostumbrarse al uso de las vistas en una aplicación web.
- Acostumbrarse a la depuración de javascript desde el navegador y demás.
- Herramientas ofrecidas para detectar errores frontend.

12.11. Problema con la API de Errbit

Además del problema de autenticación, la API de Errbit ha ofrecido distintos problemas relativos a la veracidad de los datos que proporciona. Se ha comprobado que no todos los datos obtenidos via API son consistentes con los valores visualizados en el propio Servicio. Parecería que se trata de un problema interno del desarrollo de la API, sin embargo se continúa investigando si el problema radica en alguna configuración propia realizada sobre el servicio Errbit y ajena a este proyecto. Este es sin duda alguna uno de los puntos a seguir trabajando en el futuro de la aplicación.

12.12. Cambio de entorno de desarrollo

El proyecto se ha desarrollado desde sus comienzos en un ordenador portátil con unas características y configuración determinadas. Al concluir el proyecto ha sido necesario cambiar el entorno de desarrollo para mostrar la aplicación funcionando en un nuevo ordenador portátil. Por ello ha sido necesario invertir tiempo en la configuración de todas las características necesarias en el nuevo portátil: Sistema Operativo, JDK, instalación de programas utilizados como pgAdmin3 y Dia, configuración de la conexión VPN, etc.

Todo ello ha consumido tiempo ya que además se han encontrado algunos problemas durante la configuración. El principal problema ha sido la necesidad de una conexión Ethernet para realizar todas las descargas e instalaciones de forma sencilla en Ubuntu, incluyendo la instalación de los drivers de la tarjeta de red inhalámbrica.

12.13. Configuración del Contexto

En etapas finales se ha tenido problemas con llamadas AJAX a urls parciales debido a la configuración del Context. Por defecto, en Grails el root context toma el nombre del proyecto que se está ejecutando. Cuando este valor es configurado de una forma distinta en servidores de producción, entonces se generan problemas de inconsistencia que afectan principalmente a todos aquellos elementos que dependen en una especificación parcial de alguna url.

En este caso, las llamadas AJAX a la url parcial “/mvp/section/ajax/${controllerName}” fallan si mantenemos el nombre del proyecto como valor de root context. Ello implica que ninguna sección cargada con AJAX se resuelva de manera satisfactoria y constituye un gran problema

para la aplicación ya que las áreas más interesantes (monitoreo y tareas automáticas) son cargadas de esta forma.

13. Capítulo 13: Conclusiones y trabajo futuro

13.1. Conclusiones

Se ha desarrollado un proyecto bastante extenso que ha servido para reforzar los conocimientos adquiridos durante la carrera casi en su totalidad y a su vez ha constituido una gran oportunidad para adquirir conocimientos nuevos sobre otras áreas de la Informática y el desarrollo web y descubrir nuevas áreas de interés personal que nunca habían sido exploradas con anterioridad como la gestión de proyectos informáticos, con la planificación, análisis y toma de decisiones que conlleva.

Este proyecto ha servido para adquirir una gran experiencia en muchos niveles. Su impacto se refleja a nivel técnico, constituyendo una avalancha de conocimiento y experiencia pues se ha trabajado con nuevas y punteras tecnologías, con lo cual ha sido necesario buscar información acerca del estado actual de las más avanzadas herramientas y aprender correctamente su uso con el fin de obtener las mayores ventajas que ofrecen. Además, se ha trabajado junto a profesionales que han ofrecido su asesoramiento y de los cuales se ha obtenido muchísima información que ha servido para crecer en experiencia a nivel técnico.

Otro de los mayores impactos de este proyecto ha sido a nivel personal: autonomía, determinación, responsabilidad y capacidad de llevar a cabo un proyecto de principio a fin dentro de presupuestos preestablecidos. En este aspecto, el proyecto ha constituido sin duda alguna un tremendo reto pues durante la carrera nunca nos enfrentamos a la responsabilidad que conlleva el desarrollo completo de un proyecto informático, teniendo en cuenta todas las fases del ciclo de vida del software. Una vez concluido el proyecto se puede decir con gran seguridad que el alumno se encuentra preparado para gestionar de forma autónoma, eficiente y profesional cualquier tipo de proyecto requerido por futuras empresas.

Finalmente, la versatilidad del proyecto ha tenido gran impacto en el modo de enfrentar proyectos en el futuro, en la expansión de los conocimientos previamente adquiridos y ha permitido visualizar otras áreas estrechamente relacionadas con la Ingeniería de Software que no habían sido exploradas hasta el momento. Se considera que este proyecto ha abarcado muchísimas áreas de interés y es completo, pues cubre prácticamente todas las fases del desarrollo software, es rico a nivel técnico y ha necesitado un gran proceso de gestión y planificación en todo momento. Se trata de una aplicación para la cual ha sido necesario especificar todos los requisitos, negociar con los clientes, entrevistar a los usuarios, planificar el proyecto en cada una de sus fases, analizar el dominio del sistema, implementar, testear y analizar las funcionalidades requeridas y por supuesto, documentar cada una de las decisiones tomadas para su posterior presentación como Trabajo de Fin de Grado. Durante las distintas etapas de desarrollo se ha debido jugar distintos roles acorde a las necesidades del momento: arquitecto, modelador, gestor del proyecto, “product owner” (término directo de SCRUM), desarrollador, tester, etc. Cada uno de los roles ha proporcionado una rica experiencia a nivel profesional que permite ver el desarrollo de un proyecto software de forma global con todos los elementos, fases y roles que intervienen.

Ha sido especialmente revelador el trabajo como gestor del proyecto. Una de las principales conclusiones obtenidas a raíz de la realización de este proyecto es lo interesante, complejo y apasionante que es el mundo de la gestión de software. Las personas encargadas de este proceso deben elegir las metodologías de desarrollo a seguir, planificar el proyecto, elegir estrategias para concluir el proyecto con éxito, tomar decisiones, interactuar con muchísimos stakeholders y por supuesto tener un alto conocimento a nivel técnico para poder llevar a cabo todas las tareas anteriores de forma eficiente y exitosa. Las capacidades de comunicación, compromiso, responsabilidad y organización son por supuesto primordiales.

Antes de la realización del proyecto nunca había podido experimentar el rol de gestor de proyecto en toda su complejidad y con este proyecto, que ha necesitado muchísima gestión por su gran tamaño, he podido dedicarme en su mayoría a explorar este nuevo territorio, descubriendo que es una de mis más grandes vocaciones.

Finalmente, como principales conclusiones se puede decir que el proyecto ha constituido una herramienta de altísimo valor para el desarrollo a nivel personal y profesional en cuanto a conocimiento, aspectos técnicos, gestión y responsabilidad, entre otros. Desde el punto de vista personal ha servido para crecer en aspectos relativos a la responsabilidad de estar a cargo de un gran proyecto, en cuanto a la organización, superación personal al encontrar tantos obstáculos durante el desarrollo y ver que con esfuerzo y la actitud apropiada éstos pueden ser vencidos, gestión del tiempo propio e incontables más. Desde el punto de vista profesional el proyecto ha sido la formación final definitiva para comenzar una carrera profesional. Ha proporcionado una gran cantidad de conocimiento y la adopción natural de técnicas profesionales para manejar un proyecto, gestionarlo y llevarlo a cabo de principio a fin dentro del presupuesto de tiempo establecido, manteniendo al cliente involucrado y satisfecho con el producto final, pues se ha entregado una versión operativa que incluye una gran cantidad de funcionalidades que responden a las necesidades principales del cliente y los usuarios, y proporcionando un producto de calidad. Además, ha servido no sólo para adquirir muchísimo conocimiento y experiencia, sino para reforzar y afianzar conceptos de autonomía que permitirán en el futuro continuar aprendiendo nuevas técnicas y tecnologías de modo eficiente y profesional.

Sintiéndonos autónomos, con suficiente conocimiento técnico y con la actitud apropiada para hacernos responsables de la realización de cualquier proyecto informático, considero que estamos totalmente preparados para hacer nuestra entrada al mundo profesional. Esta ha sido sin duda alguna la más grande de las aportaciones de este proyecto a nivel personal.

13.2. Líneas de trabajo futuro

El proyecto tiene un gran futuro y un largo camino por delante no sólo desde el punto de vista de mejoras sino de nuevas funcionalidades. Como se ha mencionado anteriormente, se trata de un gran proyecto con muchísimos requisitos funcionales a implementar. Evidentemente no se ha podido implementar todos los requisitos anotados ya que el presupuesto temporal de este TFG abarca solamente 300 horas. Por ello se decidió desarrollar el proyecto bajo metodologías de desarrollo ágil, ofreciendo distintas iteraciones a lo largo del ciclo de vida del proyecto con versiones operativas de la aplicación final. Además, se ha involucrado lo

suficientemente al cliente como para que decidiera qué requisitos se implementarían en el presupuesto de tiempo y cuáles se quedarían anotados para versiones futuras del proyecto.

A continuación se describen algunas de las mejoras / nuevas funcionalidades más importantes para el futuro. Sin embargo, como Anexo se encuentra una lista de Requisitos con aquellos requisitos funcionales que no han sido implementados y se quedan, por supuesto, como requisitos para versiones futuras.

- Hacer la configuración del sistema persistente en base de datos y que el usuario la pueda modificar en cualquier momento: destinatarios de emails, nombres de usuario y contraseñas de APIs, etc.
- Escritura automática de la ReleasePage en una página de Confluence.
- Marcar checkpoints automáticamente cuando un evento relacionado es realizado.
- Implementar un loggin que incluya roles para los distintos tipos de usuario y asignar permisos de ejecución de tareas a cada rol.
- Realizar la integración con Pidgin para el envío de mensajes y alertas vía chat.
- Realizar la integración con Icinga.
- Realizar la integración con TestRails.
- Detectar y solucionar el problema relacionado con Errbit que impide obtener datos 100% consistentes y confiables.
- Refactorización del código mediante revisiones conjuntas en pair-programming para aplicar criterios de mejora.
- Mejoras frontend en general.
    - Refactorización de la parte frontend por un experto frontend. Durante la realización del proyecto se ha enfocado la limpieza del código principalmente al backend, quedando el código frontend en un plano secundario y por lo tanto, necesita atención para garantizar una futura mantenibilidad y simplicidad.
    - Mejora de la Interfaz de Usuario: colores, íconos, distribución, efectos, etc.

14. Capítulo 14: Fuentes de información

En este proyecto se ha hecho uso de variadas fuentes de información, desde artículos y libros profesionales hasta expertos de materia.

Existen muchas fuentes de información que consultar cuando se lleva a cabo un proyecto de fin de carrera. La mayoría de ellas son accesibles por medio de Internet, lo cual facilita mucho la tarea de búsqueda de información pues encontramos la mayor cantidad de información en nuestro navegador y además tenemos buscadores como "Google" que nos facilitan la tarea de búsqueda.

Entre las fuentes de información utilizadas con mayor frecuencia para solucionar problemas técnicos y de gestión del proyecto se encuentran:

- Libros oficiales y técnicos de las herramientas.
- Artículos profesionales.
- Foros de internet.
- Blogs personales de desarrolladores y managers de proyectos.
- Documentación oficial de las herramientas.
- Sitio oficial de las distintas herramientas y tecnologías utilizadas.
- Prefesionales y expertos en determinadas áreas de conocimiento.

Evidentemente, las fuentes de información para solucionar problemas de requisitos, especificaciones, validaciones, etc. han sido los potenciales usuarios finales, los clientes y distintos desarrolladores y expertos experimentados en el trabajo con grandes proyectos y especificación de requisitos. En este punto se ha obtenido información especialmente útil de personas con títulos de Product Owner y Product Manager.

14.1. Fuentes físicas

Ha sido especialmente útil y enriquecedor el hecho de desarrollar el proyecto en una Empresa de Desarrollo real ya que ha sido posible acceder a la biblioteca interna y consultar de forma física todos los libros técnicos necesarios. Además, ha sido posible realizar consultas puntuales a profesionales de desarrollo software de distintas áreas y dominios. Dichos profesionales son personas en su mayoría con más de 10 de años de experiencia, que trabajan en el día a día con los dominios de los problemas que han surgido durante el desarrollo del proyecto. Por lo tanto, dichos profesionales son una fuente directa, rápida y fiable de información.

Aquellas fuentes de información "no físicas" se encuentran descritas de forma apropiada en el siguiente apartado.

14.2. Bibliografía

1. **Jeff Scott Brown, Graeme Rocher.** *The definitive guide to Grails 2.* s.l. : Apress, 2013. p. 360.

2. **Jacobson, Ivar.** *El proceso unificado de desarrollo de software.* s.l. : Addison Wesley Publishing Company, 2000.

3. **Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra.** *Head First Design Patterns.* s.l. : O'Reilly Media, 2004. pág. 678.

4. Foundation. [En línea] ZURM. Disponible en: http://foundation.zurb.com/. http://foundation.zurb.com/.

5. Groovy. [En línea] Disponible en: http://groovy.codehaus.org/. http://groovy.codehaus.org/.

6. **Graeme Rocher, Peter Ledbrook, Marc Palmer, Jeff Brown, Luke Daley, Burt Beckwith.** The Grails Framework - Reference Documentation. [En línea] Disponible en: http://grails.org/doc/2.2.4/. http://grails.org/doc/2.2.4/guide/single.html.

7. Noty. [En línea] http://ned.im/noty/.

8. Groovy. *Using Enums.* [En línea] http://groovy.codehaus.org/Using+Enums.

9. **Heilmann, Christian.** Tutsplus. *Create a sticky note effect in 5 easy steps with css3 and html5.* [En línea] 8 de diciembre de 2011. http://code.tutsplus.com/tutorials/create-a-sticky-note-effect-in-5-easy-steps-with-css3-and-html5--net-13934.

10. **Kocz, Matheusz.** Radiating Star. *Distribute divs or images equaly in line.* [En línea] 26 de junio de 2012. http://radiatingstar.com/distribute-divs-images-equaly-line.

11. **James, Mark.** Famfamfam. *Silk icons.* [En línea] http://www.famfamfam.com/lab/icons/silk/previews/index_abc.png.

12. W3schools. *Html links.* [En línea] http://www.w3schools.com/html/html_links.asp.

13. Jira. *Instance oddity on forms.* [En línea] 5 de diciembre de 2013. https://jira.grails.org/browse/GRAILS-10883.

14. Stackoverflow. *Form inside a table.* [En línea] 2011. http://stackoverflow.com/questions/5967564/form-inside-a-table.

15. **Grails Plugin Collective, Craig Andrews, Luke Daley, Peter Ledbrook, Jeff Brown, Graeme Rocher, Marc.** Grails. *Mail Plugin.* [En línea] 8 de mayo de 2014. http://grails.org/plugin/mail .

16. Grails. *Calendar Plugin.* [En línea] http://grails.org/plugin/calendar.

17. **Sergey Nebolsin, Graeme Rocher, Ryan Vanderwerf.** Grails. *Quartz Plugin.* [En línea] 5 de noviembre de 2013. http://grails.org/plugin/quartz.

18. Jira. *g:remotelink , before option for confirmation dialog on a Delete link call causes the renderer to misbehave.* [En línea] 2 de enero de 2013. https://jira.grails.org/browse/GRAILS-3910.

19. **Devijver, Steven.** Javalobby. *Thread-safe webapps using Spring.* [En línea] http://www.javalobby.org/articles/thread-safe/.

20. **Bramley, Robin.** Lean Java Engineering. *Using JMS in Grails.* [En línea] 12 de octubre de 2012. http://leanjavaengineering.wordpress.com/2011/10/12/using-jms-in-grails/.

21. Stackoverflow. *Java date format pattern.* [En línea] 2012. http://stackoverflow.com/questions/8911358/java-date-format-pattern.

22. **Jovanovic, Janco.** Dzone. *Css message boxes for different message types.* [En línea] 26 de mayo de 2008. http://css.dzone.com/news/css-message-boxes-different-me.

23. Atlassian. *Jira REST API documentation.* [En línea] https://docs.atlassian.com/jira/REST/latest/.

24. Atlassian Developers. *Bamboo REST Resources.* [En línea] 2013. https://developer.atlassian.com/display/BAMBOODEV/Bamboo+REST+Resources.

25. Atlassian Developers. *Jira REST API Tutorials.* [En línea] 2013. https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+Tutorials.

26. **Neves, Arthur Nogueira.** Github. *Errbit API.* [En línea] 22 de marzo de 2014. https://github.com/errbit/errbit/tree/master/app/controllers/api/v1.

27. Stackoverflow. *How to call the errbit API.* [En línea] 2013. http://stackoverflow.com/questions/16483689/how-to-call-the-errbit-api.

28. Zimbra. *Zimbra Soap API Reference.* [En línea] 2012. http://files.zimbra.com/docs/soap_api/8.0/soapapi-zimbra-doc/api-reference/index.html.

29. Pidgin. *Notify API How-To.* [En línea] 2013. https://developer.pidgin.im/wiki/CHowTo/NotifyAPIHowTo.

30. Gurock. *TestRails API.* [En línea] http://docs.gurock.com/testrail-api/accessing.

31. **Larman, Craig.** *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* s.l. : Prentice Hall, 2004. pág. 736.

32. Hibernate - Community Documentation. *Capítulo 15. HQL: El lenguaje de consulta de Hibernate.* [En línea] 2004. http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/html/queryhql.html.

33. Wikipedia. *Foundation (framework).* [En línea] 13 de mayo de 2014. http://en.wikipedia.org/wiki/Foundation_(framework).

34. Wikipedia. *Scrum.* [En línea] 13 de mayo de 2014. http://es.wikipedia.org/wiki/Scrum.

35. Wikipedia. *Programación extrema.* [En línea] 23 de mayo de 2014. http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema.

36. Wikipedia. *Proceso Unificado.* [En línea] 17 de abril de 2014. http://es.wikipedia.org/wiki/Proceso_Unificado.

37. **Aris Burgos Pintos, Silvia María Garbarino de la Rosa.** WikiCE. *Agile modeling.* [En línea] http://osl2.uca.es/wikiCE/index.php/Agile_modeling.

38. JetBrains. *IntelliJ IDEA.* [En línea] http://www.jetbrains.com/idea/.

39. PostgreSQL. *Sobre postgreSQL.* [En línea] 2 de octubre de 2010. http://www.postgresql.org.es/sobre_postgresql.

40. **Group, The PostgreSQL Global Development.** PostgreSQL. *PostgreSQL Documentation.* [En línea] http://www.postgresql.org/docs/9.2/static/index.html.

41. Wikipedia. *Git.* [En línea] 24 de marzo de 2014. http://es.wikipedia.org/wiki/Git.

42. Git. [En línea] Software Freedom Conservancy. http://git-scm.com/.

43. JQuery. [En línea] http://jquery.com/.

44. Informática Hoy. *Tipos de Licencia de Software.* [En línea] http://www.informatica-hoy.com.ar/software-libre-gnu/Tipos-de-licencia-de-Software.php.

45. Wikipedia. *Apache License.* [En línea] 19 de enero de 2014. http://es.wikipedia.org/wiki/Apache_License.

46. Wikipedia. *Licencia BSD.* [En línea] 12 de marzo de 2013. http://es.wikipedia.org/wiki/Licencia_BSD.

47. Wikipedia. *Licencia MIT.* [En línea] 20 de abril de 2014. http://es.wikipedia.org/wiki/Licencia_MIT.

48. Creative Commons. *License 2.5.* [En línea] http://creativecommons.org/licenses/by/2.5/.

49. Wikipedia. *Ley Orgánica de Protección de Datos de Carácter Personal de España.* [En línea] 29 de mayo de 2014. http://es.wikipedia.org/wiki/Ley_Org%C3%A1nica_de_Protecci%C3%B3n_de_Datos_de_Car%C3%A1cter_Personal_de_Espa%C3%B1a.

50. AGPD. *Convenio sobre la Ciberdelincuencia.* [En línea] 23 de noviembre de 2001. http://www.agpd.es/portalwebAGPD/canaldocumentacion/legislacion/consejo_europa/convenios/common/pdfs/Convenio_Ciberdelincuencia.pdf.

51. **Egil Emilio Ramírez Bejerano, Ana Rosa Aguilera Rodríguez.** EUMED. *Los delitos Informáticos. Tratamiento internacional.* [En línea] mayo de 2009. http://www.eumed.net/rev/cccss/04/rbar2.htm.

52. **Borghello, Cristian Fabian.** Segu-Info. *Legislación y Delitos Informáticos - Alemania.* [En línea] http://www.segu-info.com.ar/delitos/alemania.htm.

53. **Álvarez, José Cuervo.** Informática Jurídica. *Ley Federal Alemana sobre Protección de Datos de 27 de enero.* [En línea] http://www.informatica-juridica.com/anexos/Ley_Federal_Alemana_Protecci%C3%B3n_Datos_27_enero.asp.

54. Wikipedia. *Lenguaje Unificado de Modelado .* [En línea] 12 de mayo de 2014. http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado.

55. **Carroll, Nick.** Nick Carroll. *Configuring the Grails Root Application Context.* [En línea] 27 de marzo de 2009. http://nickcarroll.me/2009/03/27/configuring-the-grails-root-application-context/.

15. Capítulo 15: Anexos

15.1. Anexo 1: Glosario

En este anexo se ofrece un glosario mínimo con el objetivo de servir de ayuda al lector de este documento en la comprensión del dominio del problema. Los términos aquí descritos son una parte fundamental del proyecto y se utilizan con gran frecuencia durante toda la memoria.

Release

Proceso de despliegue de una nueva versión de un software en un sistema de producción.

Release-Crew / Release-Team

Equipo de desarrolladores y testers principalmente que se encarga de supervisar y llevar a cabo todos los procesos necesarios para hacer release de una nueva versión.

Bamboo

Servicio web que se encarga de la integración continua del software y la ejecución de pruebas unitarias y de integración.

Jira

Servicio web que permite la gestión y manejo de las distintas actividades de desarrollo en una compañía que sigue metodologías de desarrollo ágil.

Errbit

Servicio web donde se almacenan en un formato legible y ameno todos los errores del sistema (logs del sistema de nivel “error”).

Icinga

Servicio web de monitoreo en tiempo real de la carga de los servidores, tráfico, estado de bases de datos y todos aquellos elementos relativos a la infraestructura en general.

TestRails

Servicio web para ejecutar y documentar los distintos Test-Cases (Pruebas de Aceptación) del sistema. Este servicio es principalmente útil para los testers.

15.2. Anexo 2: Diagramas de secuencia

Este anexo está pensado para mostrar el impacto que tendrá la herramienta elaborada en una empresa que tiene el dominio descrito. Estos diagramas son parte de la descripción del dominio de la aplicación y ofrecen una visión clara de los objetivos de este proyecto.

En la primera imagen se muestra el estado del dominio del sistema en la situación previa a la realización del proyecto. En la segunda imagen se muestra el estado del dominio del sistema una vez realizado el proyecto y utilizando la herramienta como parte del proceso de despliegue.

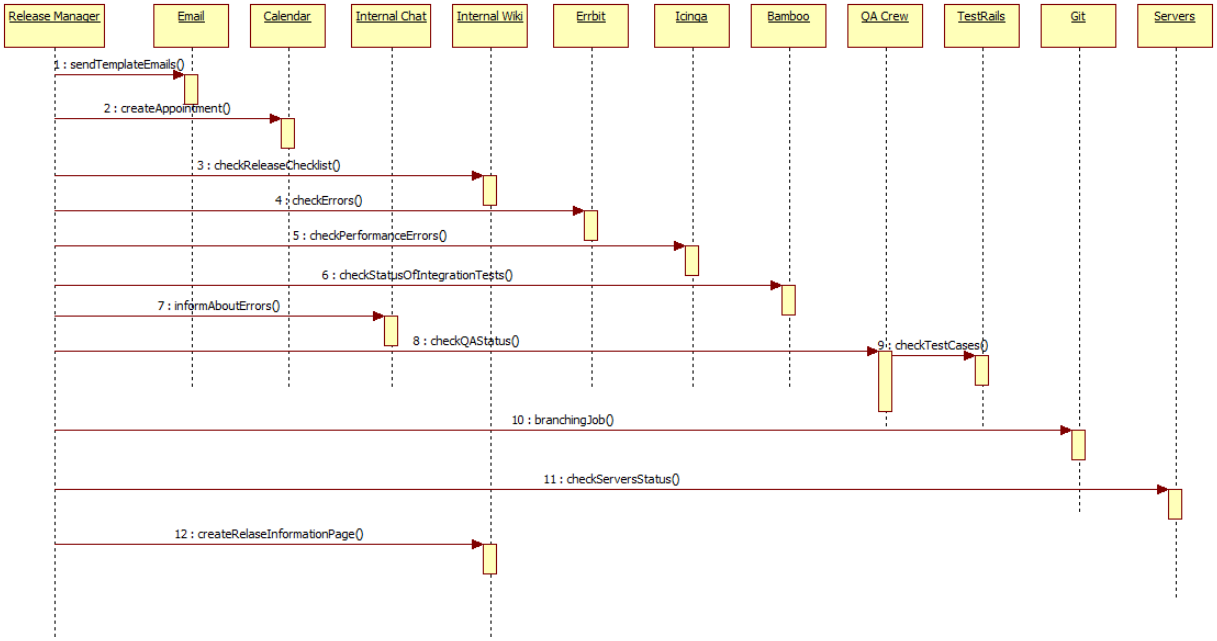


Ilustración 21. Diagrama de secuencia - Dominio del sistema antes de utilizar la aplicación

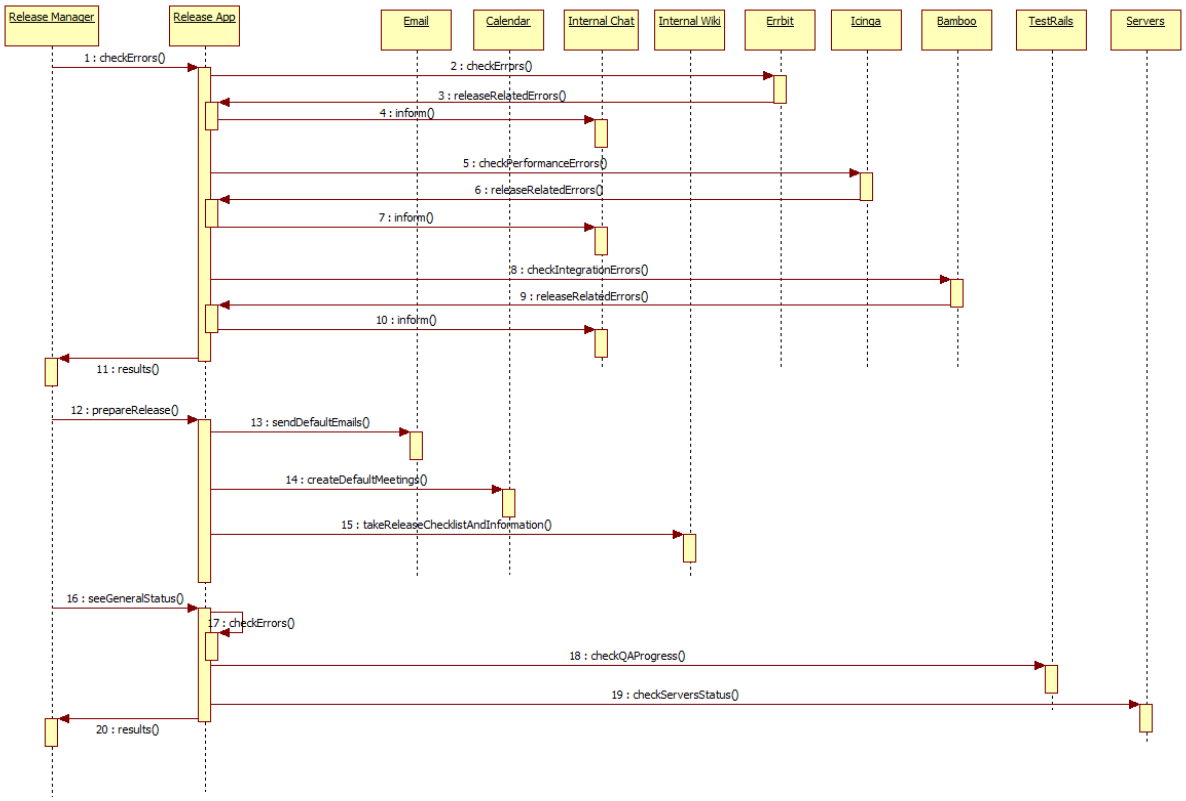
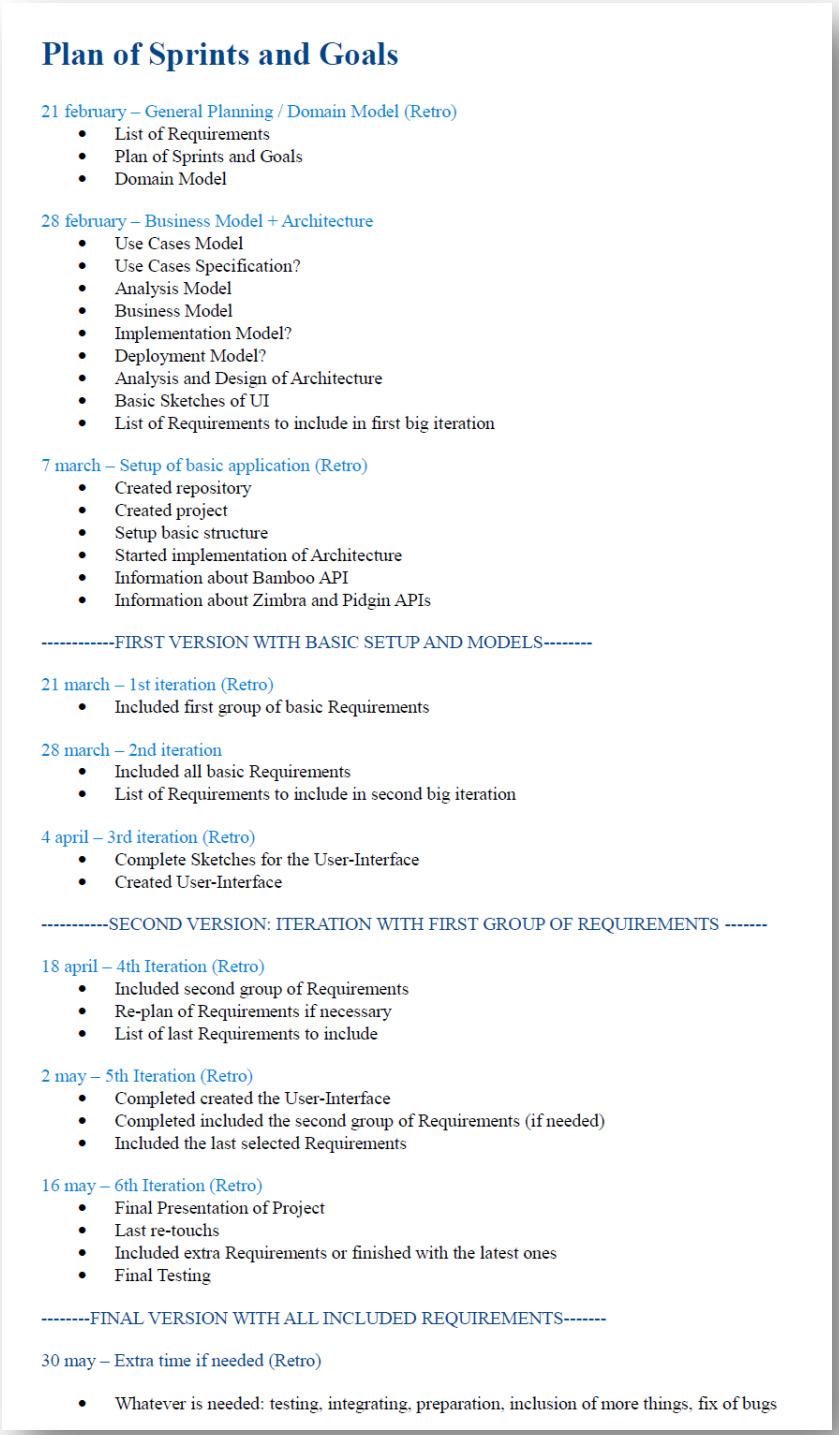


Ilustración 22. Diagrama de secuencia - Dominio del sistema antes de utilizar la aplicación

15.3. Anexo 3: Plan de Sprints

Este ha sido el más grande elemento de gestión del proyecto utilizado. En este anexo se encuentra el Plan de Sprints original utilizado durante todo el proceso de desarrollo del proyecto. En él se encuentran distintas "deadlines" con objetivos a conseguir fijados.



Plan of Sprints and Goals

21 february – General Planning / Domain Model (Retro)
- List of Requirements
- Plan of Sprints and Goals
- Domain Model

28 february – Business Model + Architecture
- Use Cases Model
- Use Cases Specification?
- Analysis Model
- Business Model
- Implementation Model?
- Deployment Model?
- Analysis and Design of Architecture
- Basic Sketches of UI
- List of Requirements to include in first big iteration

7 march – Setup of basic application (Retro)
- Created repository
- Created project
- Setup basic structure
- Started implementation of Architecture
- Information about Bamboo API
- Information about Zimbra and Pidgin APIs

----------FIRST VERSION WITH BASIC SETUP AND MODELS--------

21 march – 1st iteration (Retro)
- Included first group of basic Requirements

28 march – 2nd iteration
- Included all basic Requirements
- List of Requirements to include in second big iteration

4 april – 3rd iteration (Retro)
- Complete Sketches for the User-Interface
- Created User-Interface

----------SECOND VERSION: ITERATION WITH FIRST GROUP OF REQUIREMENTS -------

18 april – 4th Iteration (Retro)
- Included second group of Requirements
- Re-plan of Requirements if necessary
- List of last Requirements to include

2 may – 5th Iteration (Retro)
- Completed created the User-Interface
- Completed included the second group of Requirements (if needed)
- Included the last selected Requirements

16 may – 6th Iteration (Retro)
- Final Presentation of Project
- Last re-touchs
- Included extra Requirements or finished with the latest ones
- Final Testing

--------FINAL VERSION WITH ALL INCLUDED REQUIREMENTS-------

30 may – Extra time if needed (Retro)

- Whatever is needed: testing, integrating, preparation, inclusion of more things, fix of bugs

Ilustración 23. Plan de Sprints

15.4. Anexo 4: Sketches básicos

En este anexo se encuentran los Sketches básicos para la Interfaz de Usuario, utilizados al comienzo del proyecto para mejorar la comunicación con los clientes y garantizar una visión única sobre la usabilidad del proyecto y su posible futura apariencia.

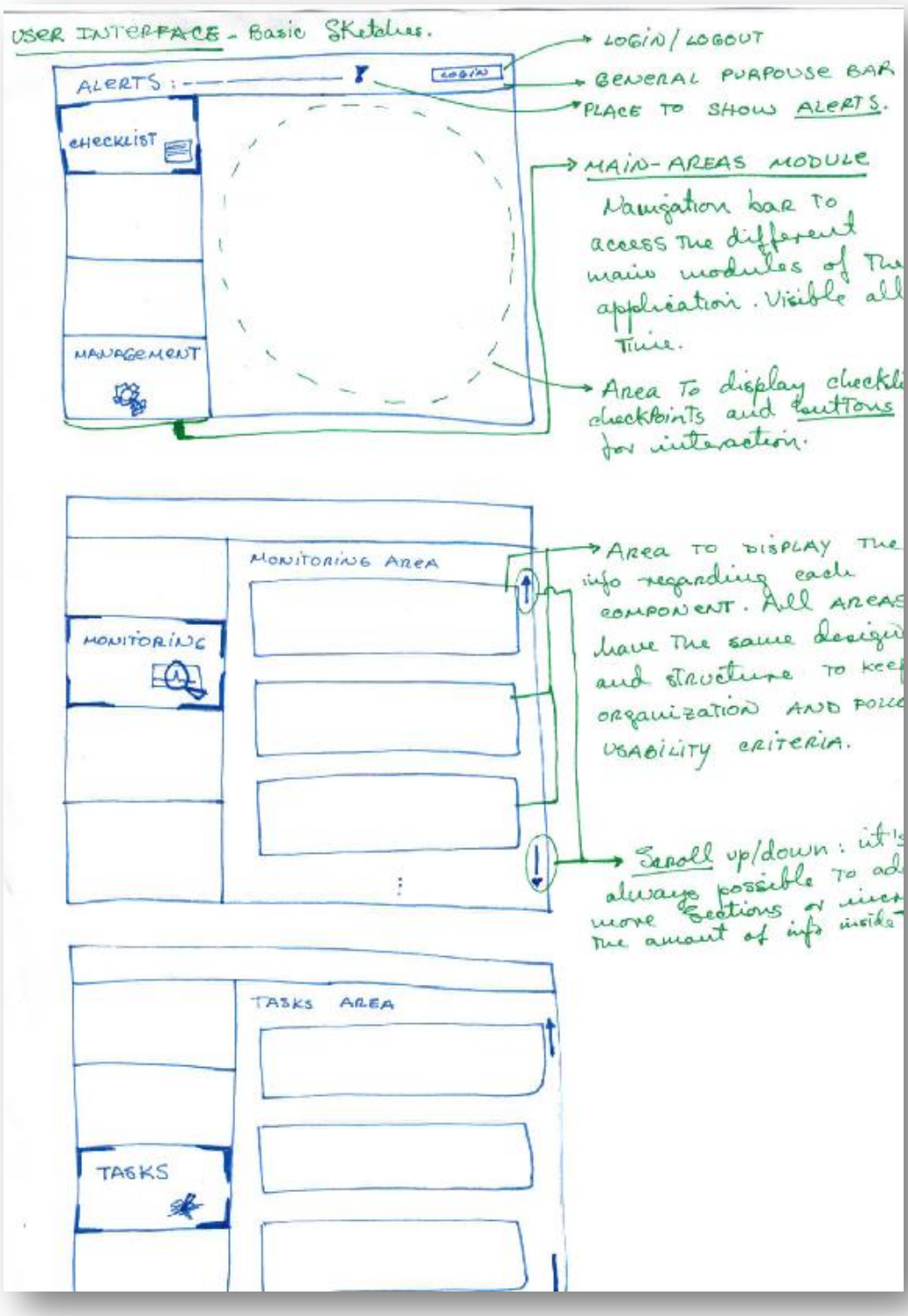


Ilustración 24. Sketches básicos para la Interfaz de Usuario

15.5. Anexo 5: Evolución de la pizarra de trabajo

Este constituye uno de los anexos más curiosos, interesantes e importantes de esta memoria. En él se muestra la evolución y progresos de la pizarra de trabajo utilizada para este proyecto y con ello se muestra la propia evolución y desarrollo del proyecto.

Durante el desarrollo de este TFG se ha tomado fotografías de la pizarra de trabajo en momentos claves para mostrar su evolución. A continuación se muestran dichas imágenes de forma cronológica.

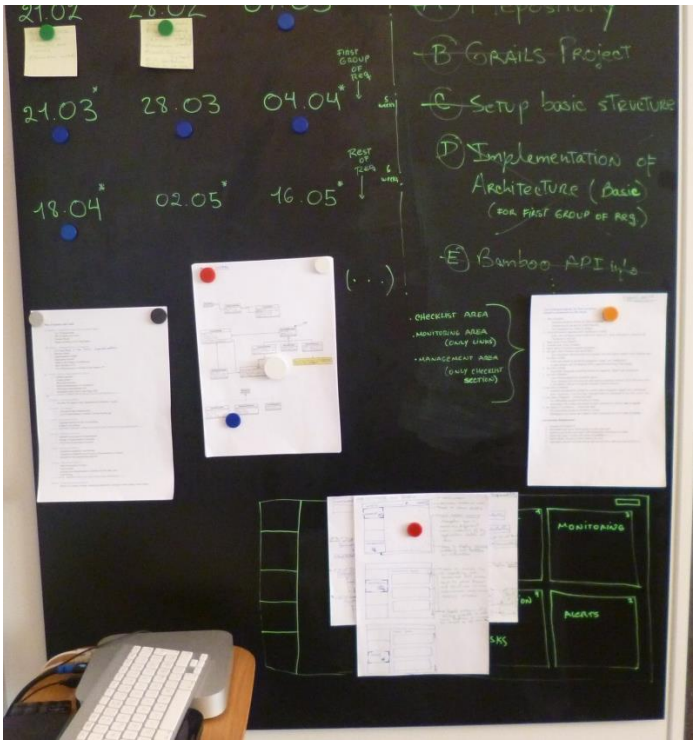


Ilustración 25. Pizarra de trabajo - primera versión

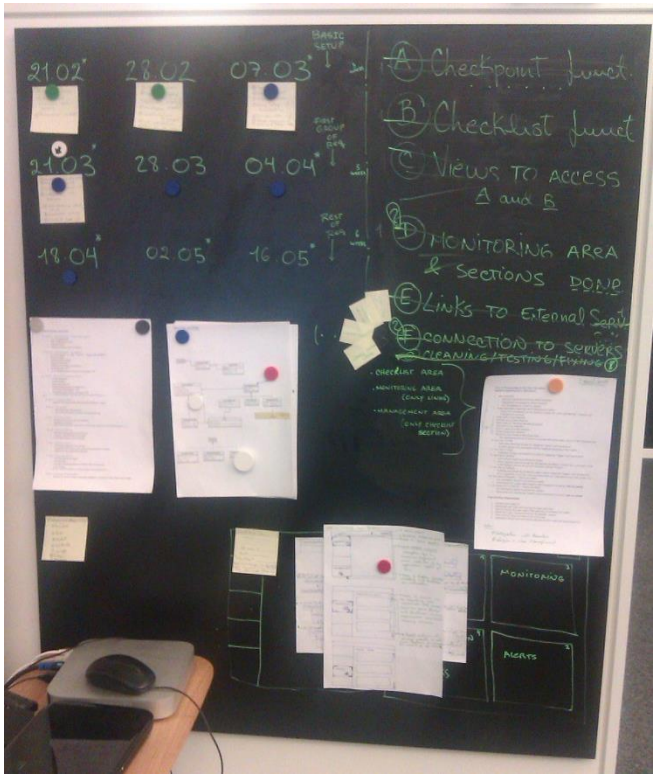


Ilustración 26. Pizarra de trabajo - segunda versión

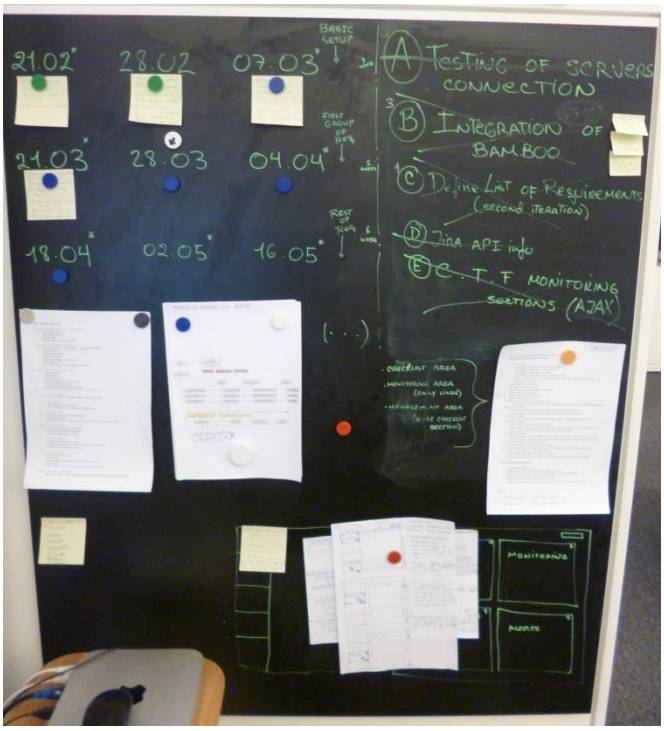


Ilustración 27. Pizarra de trabajo - tercera versión

Ilustración 28. Pizarra de trabajo - cuarta versión

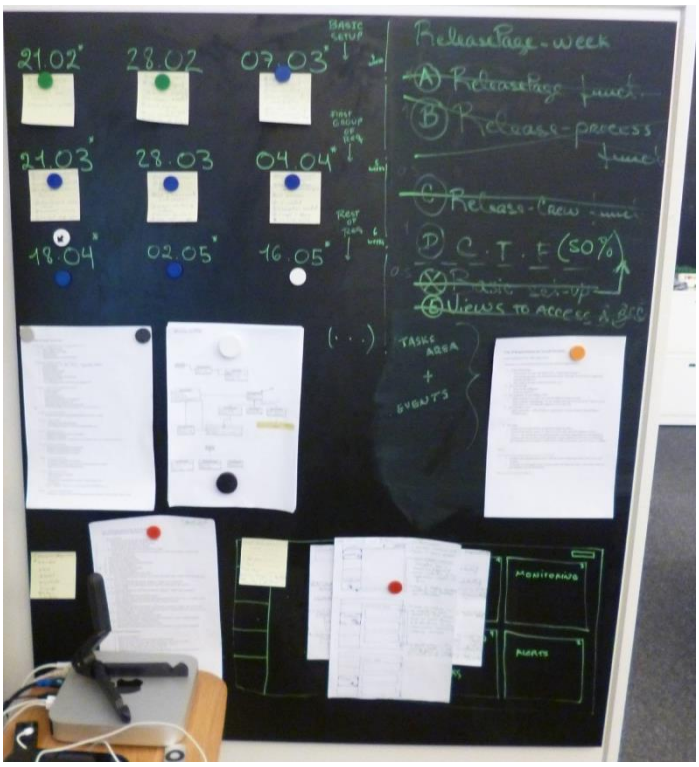


Ilustración 29. Pizarra de trabajo - quinta versión

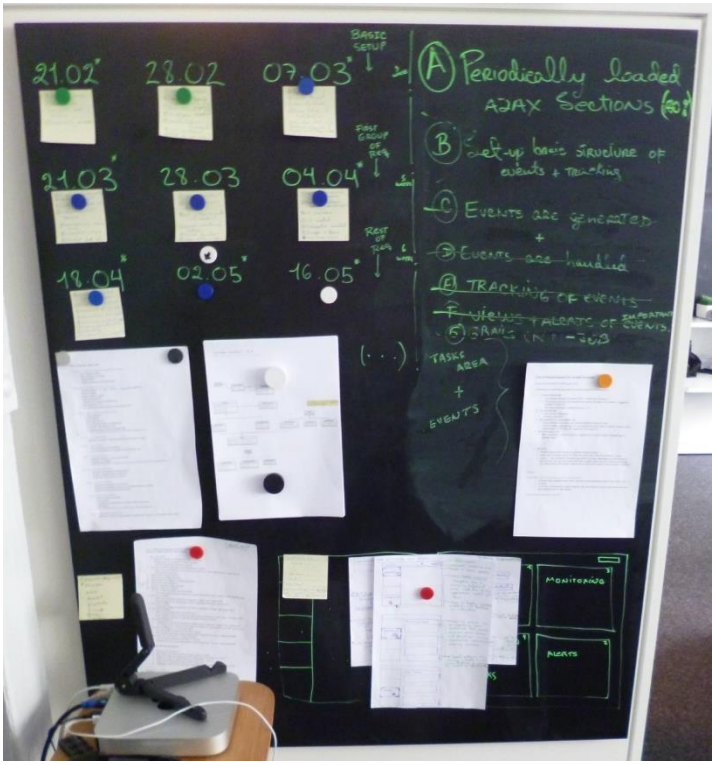


Ilustración 30. Pizarra de trabajo - sexta versión

Ilustración 31. Pizarra de trabajo - séptima versión

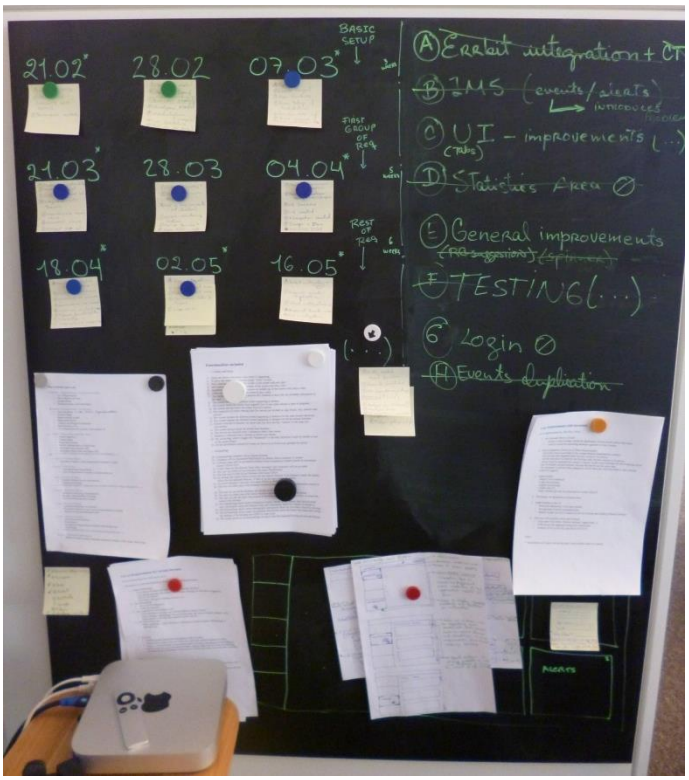


Ilustración 32. Pizarra de trabajo - octava versión (final)

15.6. Anexo 6: Manual de usuario

La aplicación consta de varias áreas principales accesibles en todo momento en la barra de navegación lateral de la aplicación, que agrupan contenido y funcionalidades. A continuación se explica de forma breve y a modo de tutorial introductorio qué esperar de cada área como usuario final y cómo sacar el mejor provecho de la información y funcionalidades ofrecidas.

Se aclara que en la barra superior de color azul solamente se posiciona el nombre y logo de la aplicación, contenedores de un enlace a la vista principal del sistema (home). Además, se posiciona a forma de "mock" la palabra "Login" como señalización del sitio donde se ofrecerá en el futuro dicha funcionalidad, cuando sea implementada.

15.6.1. Checklist

Este área es realmente simple y muestra la Checklist a seguir en su totalidad. Se encuentra dividida por días, marcando los principales objetivos para cada día y los puntos (Checkpoints) a completar durante el proceso de Release. Cada punto puede ser marcado y desmarcado en cualquier momento. De forma visual se conoce el progreso de la Release al mostrar una marca de aceptación para cada punto en estado "realizado".

Notas:

- Al comenzar una Release todos los puntos deben encontrarse en estado "pendiente".
- Existe una única Checklist en toda la aplicación, por lo que dos Releases que sigan la Checklist no pueden ser llevadas a cabo al mismo tiempo.
- Se proporciona una barra de navegación donde acceder a distintas utilidades.
- En la barra de navegación se sitúa un link a un documento externo que contiene las mejores prácticas y guías válidas para el proceso general de Release.
- En la barra de navegación se puede acceder a una funcionalidad que permite limpiar la Release ("Clean"), es decir, desmarcar todos los puntos de la Checklist. Esto es especialmente útil para dejar la Checklist en su estado inicial y lista para ser usada en otra Release.

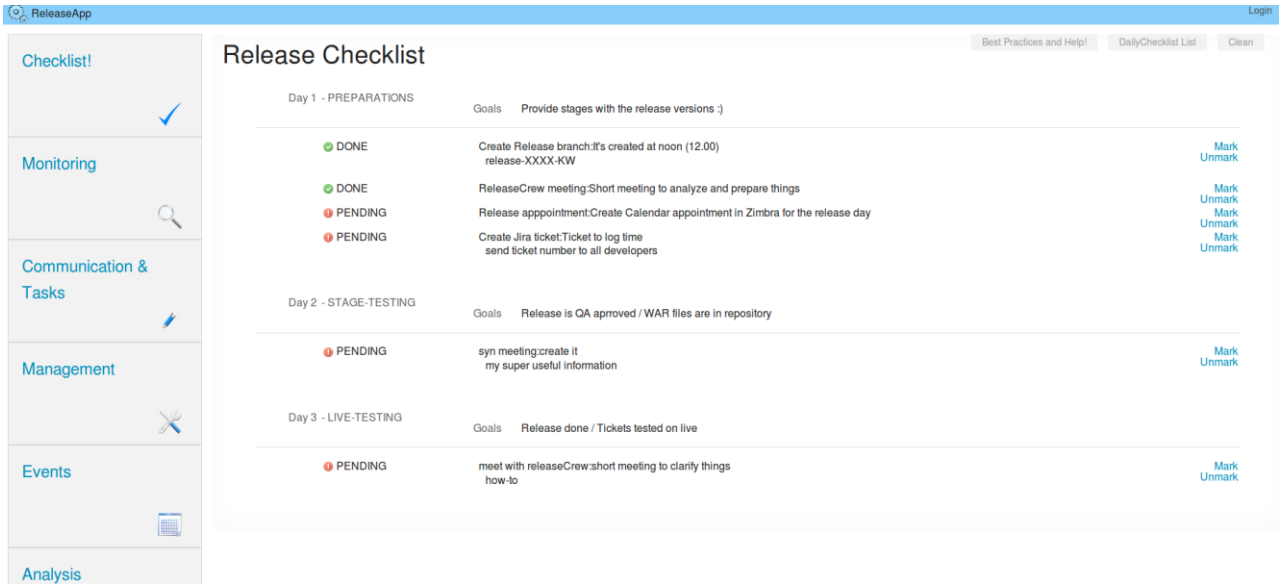


Ilustración 33. Manual de usuario – Área de Checklist

15.6.2. Monitoring

Este área ofrece las distintas vistas necesarias para monitorizar el estado del sistema. Cada servicio de monitoreo integrado tiene su propia sección para mostrar su información. Existe una barra de navegación con enlaces a cada una de estas secciones. A continuación se resume el contenido de cada sección, omitiendo aquellas que existen pero cuyos servicios asociados aún no han sido integrados en la aplicación (Icinga y TestRails) y para las que sólo se muestran los links de interés a los servicios.

Notas:

- Cada sección es renderizada periódicamente con información actualizada.
- Se envía un email de notificación cada vez que se produce un evento: cambio en el estado de alguna rama de Bamboo o cambio en la versión de la aplicación de alguno de los servidores.

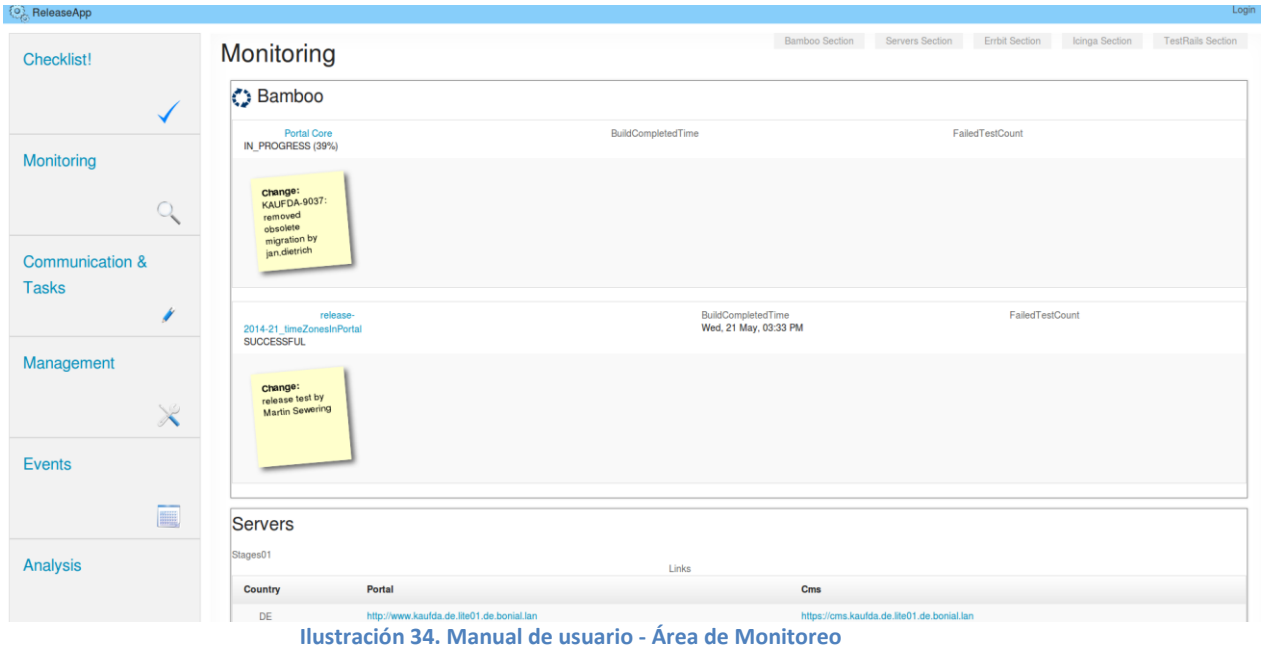


Ilustración 34. Manual de usuario - Área de Monitoreo

15.6.2.1. Sección de Bamboo

Se muestra información relativa a la rama principal de la aplicación (de la cual se deben crear todas las ramas de Release) y a todas las ramas de Release (contenedoras de una versión que se va a desplegar) existentes y activas en Bamboo.

Notas:

- La información de cada rama de bamboo se muestra de forma agrupada: cada rama en una fila distinta.
- La infomación mostrada abarca estado de la rama, nombre de la rama con un link a Bamboo, fecha/hora de finalización de ejecución de los tests, cantidad de tests que fallan y una representación a modo “post-it” de todos los cambios incluídos en la nueva construcción de la rama.
- Los post-its representantes de los cambios muestran un mensaje (mensaje de “commit”) que contiene el número de ticket asociado por convención, además de mostrar el nombre del desarrollador que lo ha realizado.
- Los post-its muestran una versión reducida del mensaje. Sin embargo, es posible leer el mensaje completo haciendo roll-over.
- Cada cambio realizado se muestra en un post-it independiente.
- Se muestra una alerta visual cuando alguna de las ramas está fallando, o sea, cuando alguno de los tests unitarios o de integración no ha tenido éxito.

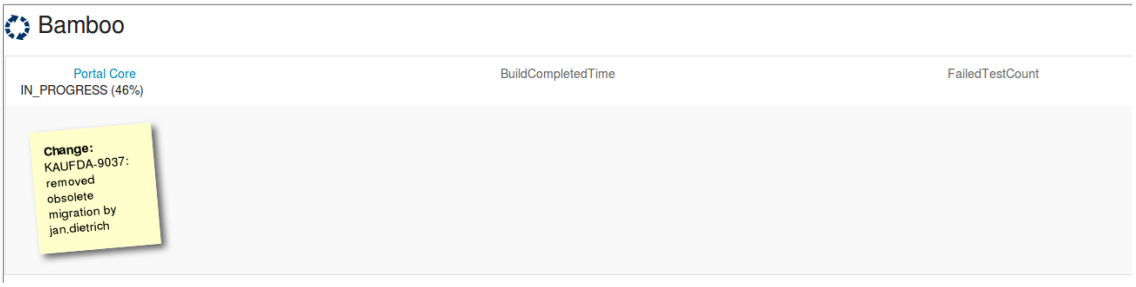


Ilustración 35. Manual de usuario - Sección de Bamboo

15.6.2.2. Sección de Servidores

Se muestra información relativa a los Stages usados para testear la nueva versión a desplegar y seguidamente información relativa a los Servidores de Producción. En ambos casos se sigue el mismo patrón de representación de la información, con ligeras variaciones que se comentan a continuación.

Servidores de Stage

- Se muestran todos los link directos al portal web y cms para cada país, agrupados bajo la etiqueta “Links”.
- Se muestra información relativa a cada servidor agrupada bajo la etiqueta “Status” que contiene: estado (representado por una marca visual), versión de la aplicación que se encuentra desplegada y nombre de la rama sobre la que se construyó dicha versión.
- Se muestra además un link directo al Job de Bamboo asociado.
- Se muestra una alerta visual cada vez que hay un cambio de versión en cualquiera de los servidores.

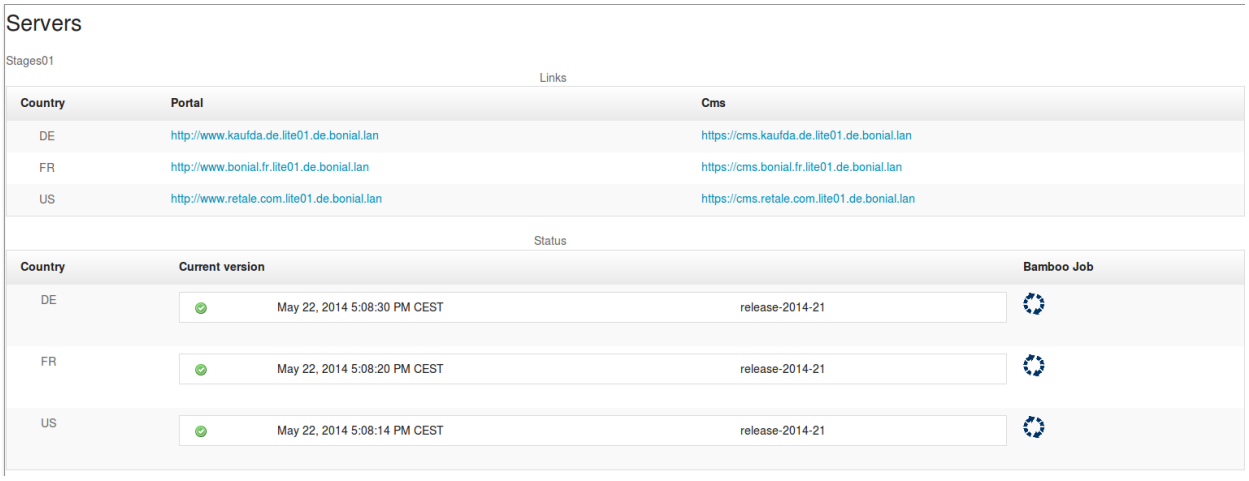


Ilustración 36. Manual de usuario - Sección de Servidores (Stages)

Servidores de Producción

- Se muestran todos los links directos a las versiones en producción del portal web y cms para cada país, agrupados bajo la etiqueta “Links”.
- Se muestra información relativa a los servidores de portal y cms agrupada bajo la etiqueta “Status”: estado (representado por una marca visual), versión de la aplicación que se encuentra desplegada y nombre de la rama sobre la que se constryó dicha versión.
- Si alguno de los servidores no está operativo se muestra una marca roja de alerta en señal de un estado erróneo y no se muestra información alguna sobre la versión desplegada.
- Además se muestra información relativa al estado de los distintos Tomcats de la aplicación que sirve al portal web. Esta información no se encuentra visible por defecto pero es sencillamente accesible haciendo click en el link “Extra info” disponible junto al estado de cada servidor.

- Al visualizar la información completa sobre la versión en producción del portal web y la versión de los tomcats para dicha aplicación, se puede apreciar qué tomcat es el proveedor de la aplicación visible en producción en caso de tener versiones distintas en los tomcats y detectar posibles inconsistencias en el sistema, ya que todos los tomcats deberían tener la misma versión.
- Se muestra una alerta visual cuando, para alguno de los países, los tomcats tienen desplegada una versión distinta. Esta es una alerta de consistencia.
- Se muestra una alerta visual cada vez que hay un cambio de versión en cualquiera de los servidores.

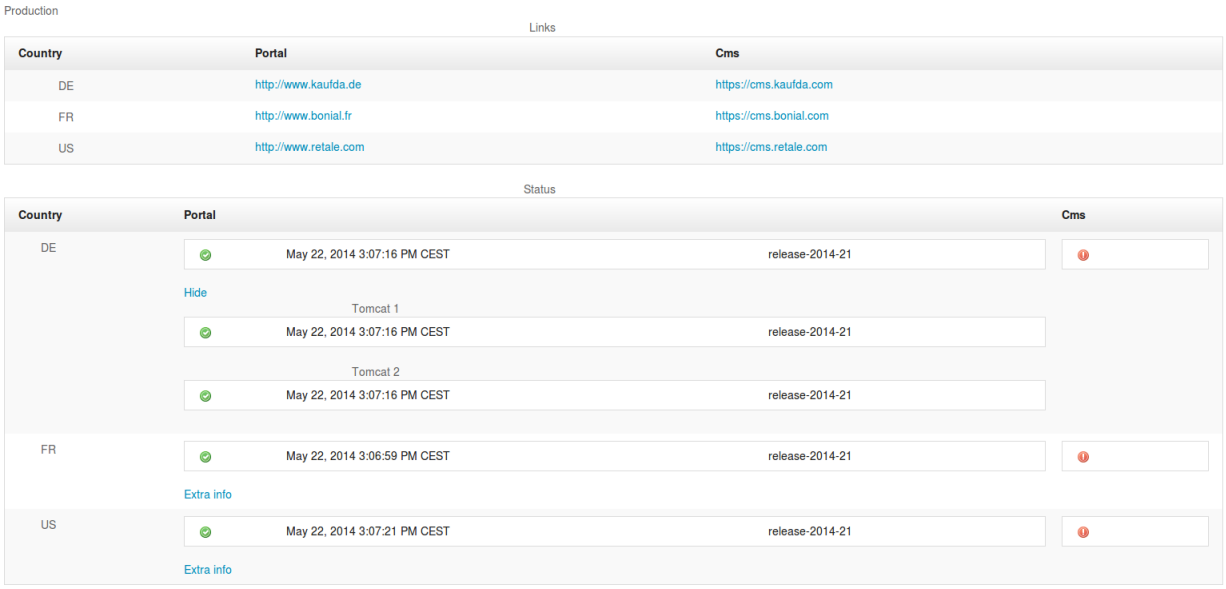


Ilustración 37. Manual de usuario - Sección de Servidores (Producción)

15.6.2.3. Sección de Errbit

Se muestra información relativa a los errores de Errbit almacenados solamente para los servidores de producción. La información se divide acorde a la realidad: información para servidor boreus (almacena errores para las aplicaciones que sirven en Alemania y Francia), información para servidor rackspace (almacena errores para la aplicación que sirve en Estados Unidos).

Notas:

- Se muestran los links directos a los servidores Errbit, incluyendo el link al servidor que contiene todos los errores para los distintos servidores de stage.
- Se muestran todos los errores que han ocurrido por última vez después de que la última versión fuera desplegada en el servidor asociado. Es decir, se muestran aquellos errores que están sucediendo después de desplegar la nueva versión.
- Se marcan de forma visual aquellos errores que han ocurrido por primera vez después de desplegar la nueva versión en el servidor asociado. Estos errores son potencialmente ocasionados por la nueva versión deplegada y por lo tanto son los más peligrosos y a los que más atención se debe prestar.

- Para cada error se muestra la marca de novedad, el nombre de la aplicación donde ha ocurrido, un mensaje representativo del error (en azul) con un link al error en Errbit, información acerca de la localización del código culpable del error, fechas de primera aparición y última aparición del error y finalmente el número de ocurrencias del error.

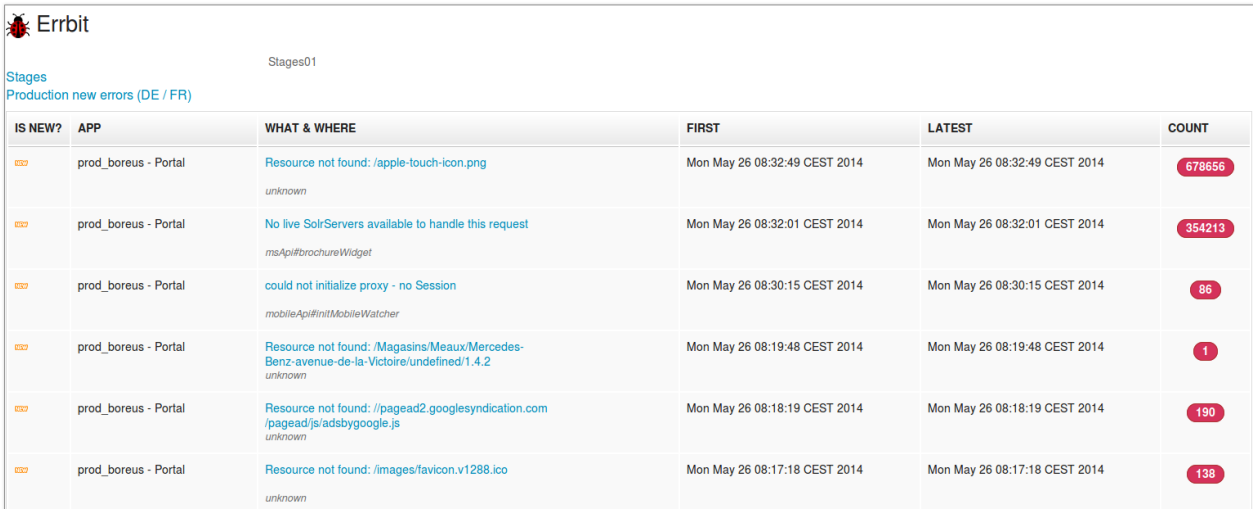


Ilustración 38. Manual de usuario - Sección de Errbit

15.6.3. Communication and Tasks

Este área ofrece la posibilidad de realizar tareas de forma automatizada y que influyen directamente en procesos de comunicación. Hasta el momento está compuesta por 2 secciones de trabajo: tareas relacionadas con la Release (o ReleasePage) y tareas relacionadas con el servicio externo "Jira". Nuevamente se incluye una barra de navegación con enlaces a cada una de estas secciones.

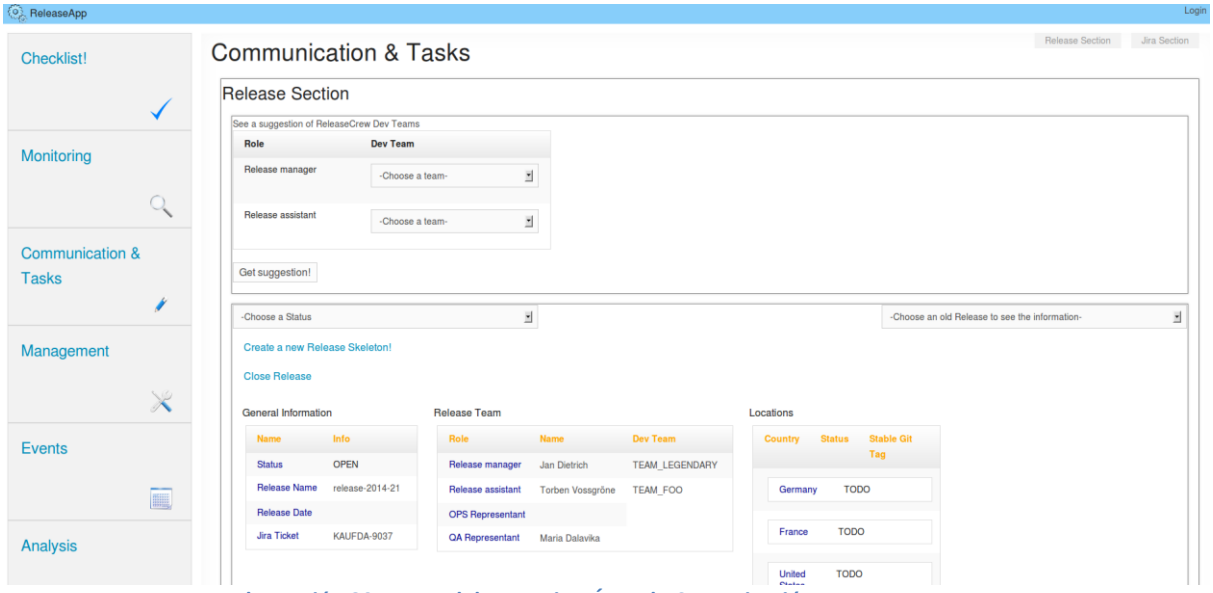


Ilustración 39. Manual de usuario - Área de Comunicación y Tareas

15.6.3.1. Sección de Release

Contiene todas las tareas directamente relacionadas con el concepto de Release / ReleasePage que se describen a continuación.

Obtención de sugerencia de equipos de desarrollo para la próxima Release-Crew. Simplemente es necesario introducir la información necesaria acerca de la última Release-Crew.

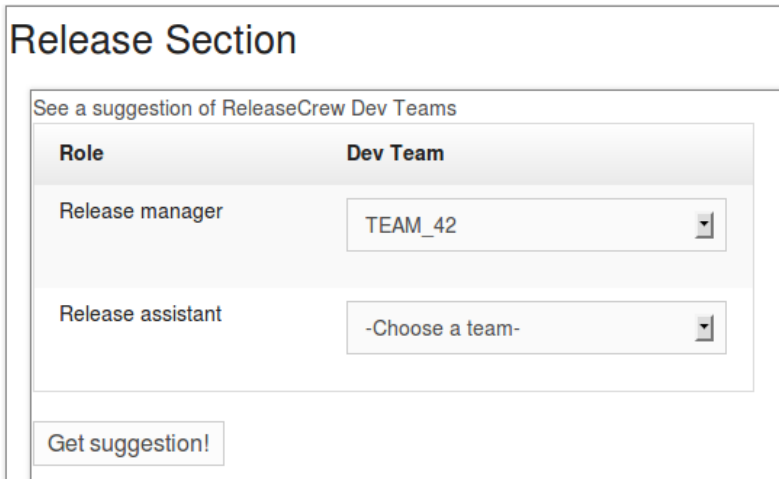


Ilustración 40. Manual de usuario - Sugerencia de próxima ReleaseCrew

Selección de ReleasePage para su posterior visualización.

- Se debe seleccionar primero un estado, para visualizar todas las Releases almacenadas por la aplicación que se agrupan en ese estado.
- Se debe seleccionar posteriormente una Release, atendiendo a su nombre.

Ilustración 41. Manual de usuario - Selección para visualizar ReleasePage

Visualización del prototipo de ReleasePage para cualquier Release.

- Se muestra la información general y básica de la Release, incluyendo información sobre la ReleaseCrew.
- Se muestra información sobre el ticket asociado a la Release en caso de existir.
- Se muestra información sobre los eventos ocurridos en el período en que la Release estuvo abierta y activa. La información sobre los eventos se encuentra dividida en distintos grupos, atendiendo a la fuente de los mismos.

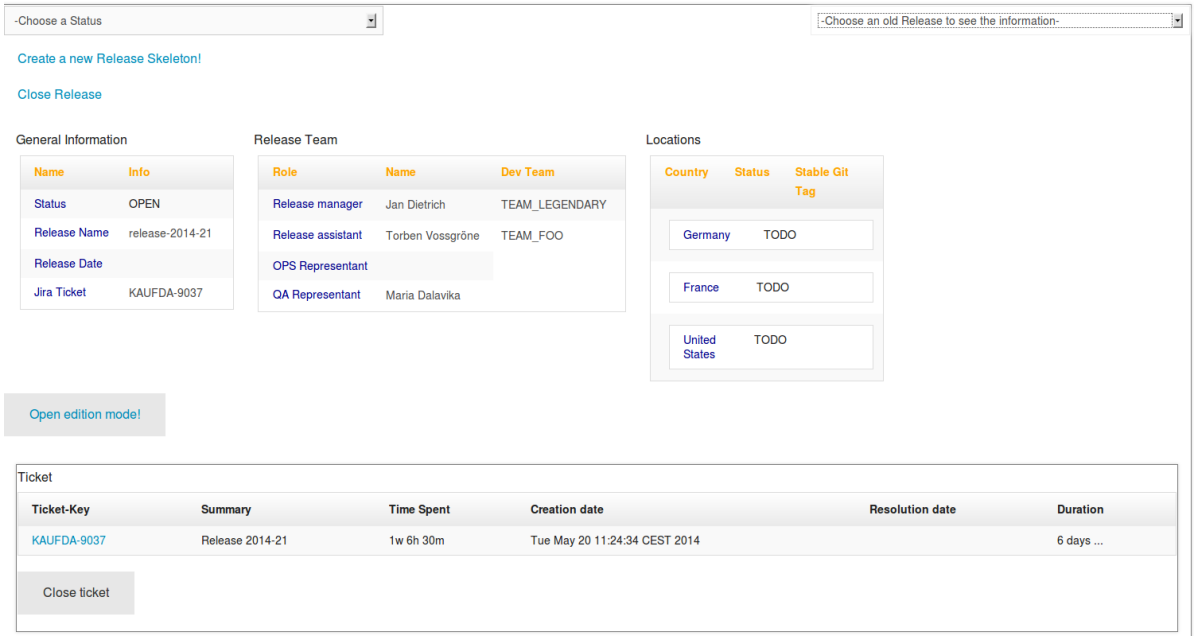


Ilustración 42. Manual de usuario - Manejo de Release

Creación del prototipo de ReleasePage para una nueva Release, disponible en cualquier momento.

- Por defecto algunos campos de la información básica serán automáticamente completados a modo de sugerencia.
- Los equipos de desarrollo de la ReleaseCrew serán sugeridos a partir de la información almacenada acerca de la ReleaseCrew de la última ReleasePage creada.

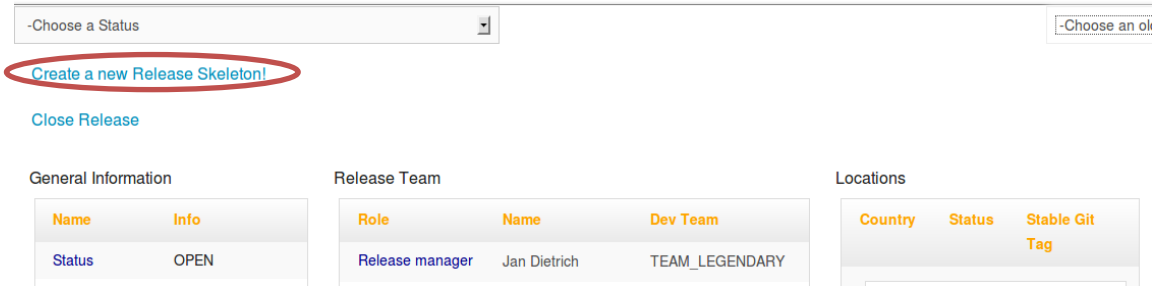


Ilustración 43. Manual de usuario - Creación de nueva ReleasePage

Edición del prototipo de ReleasePage, disponible en cualquier momento independientemente del estado de la Release.

- Se puede editar la mayoría de elementos de la información básica.
- Se debe salvar la información modificada explícitamente en el botón “Save” asignado al área que se haya modificado.
- Se puede eliminar o crear la asociación de una Release a un ticket previamente creado.
- Al crear una asociación con un ticket previamente creado, se comprobará que dicho ticket sea válido para evitar futuros errores al importar la información relacionada.
- Desde la vista de edición se puede volver a la vista general anterior.

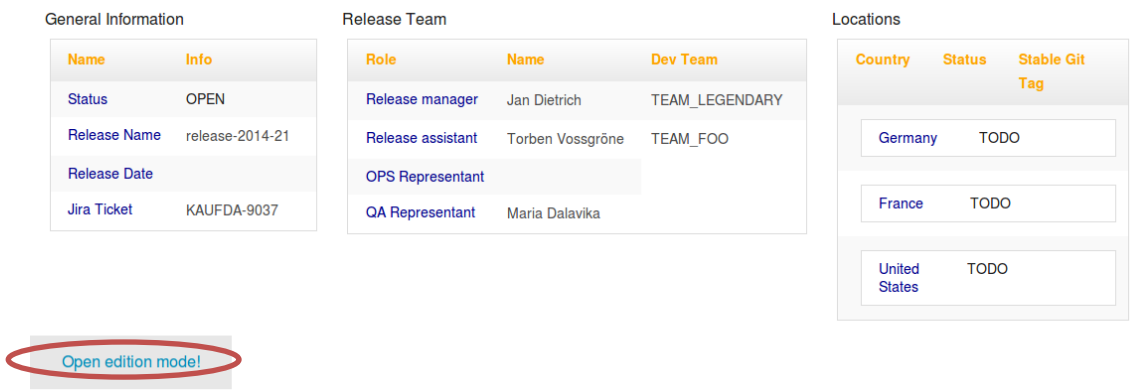


Ilustración 44. Manual de usuario - Información básica de ReleasePage

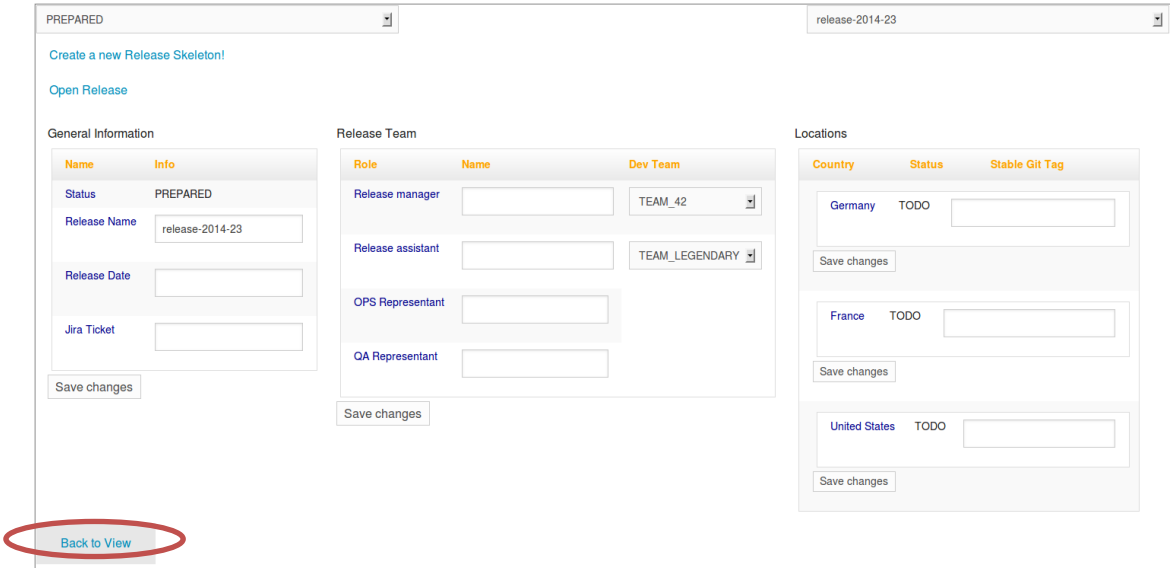


Ilustración 45. Manual de usuario - Edición de información básica de ReleasePage

Creación de un ticket para una Release específica.

- Esta acción asociará el ticket creado con la Release seleccionada.
- Sólo disponible cuando la Release no tienen ningún ticket asociado.
- Se envía un email automáticamente al grupo de personas interesadas con información acerca del nuevo ticket creado para su posterior utilización.

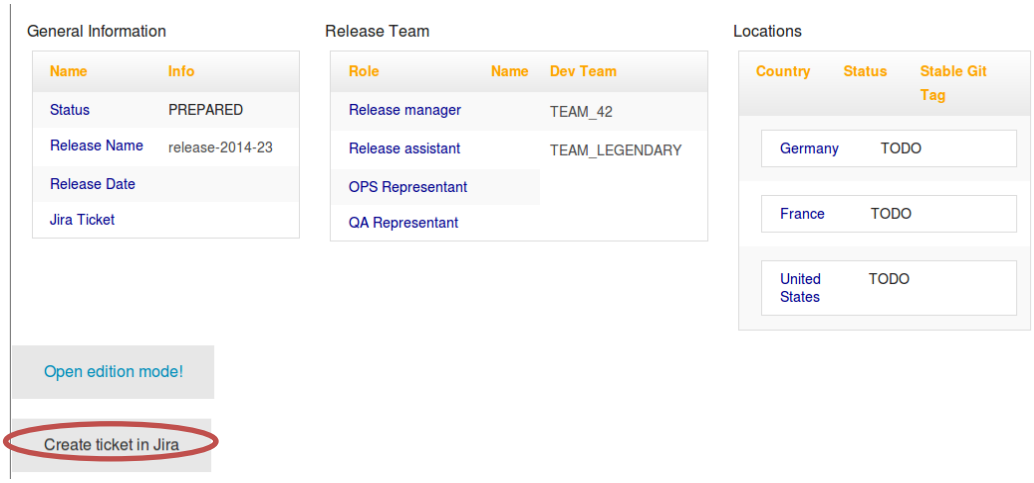


Ilustración 46. Manual de usuario - Creación de ticket

Cerrar el ticket asociado a una Release: sólo disponible si la Release tiene un ticket asociado.

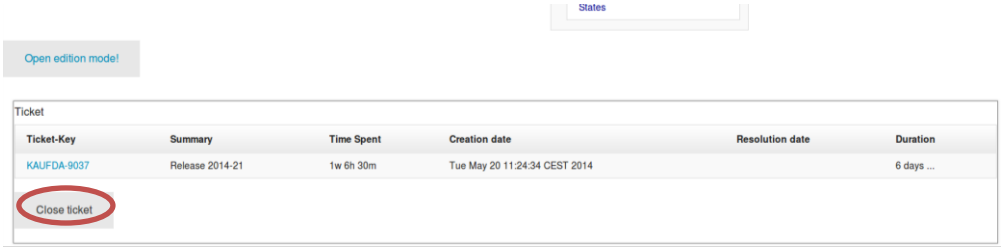


Ilustración 47. Manual de usuario - Cierre de ticket

Abrir una Release. Para abrir una Release esta debe estar siendo visualizada y debe encontrarse en estado “preparado”. La apertura de la Release la convertirá en una Release activa.

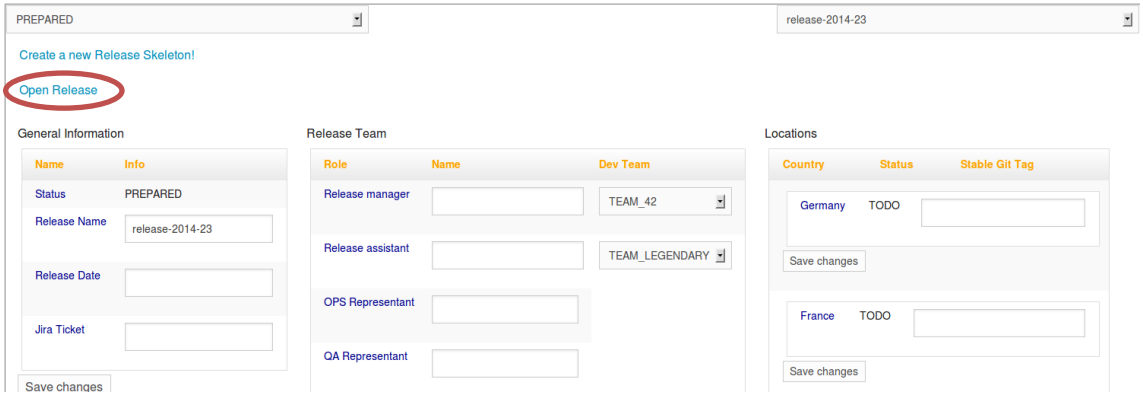


Ilustración 48. Manual de usuario - Apertura de Release

Cerrar una Release.

- Se generará un análisis de la Release.
- Se enviará un email a las personas interesadas con los resultados de dicho análisis.

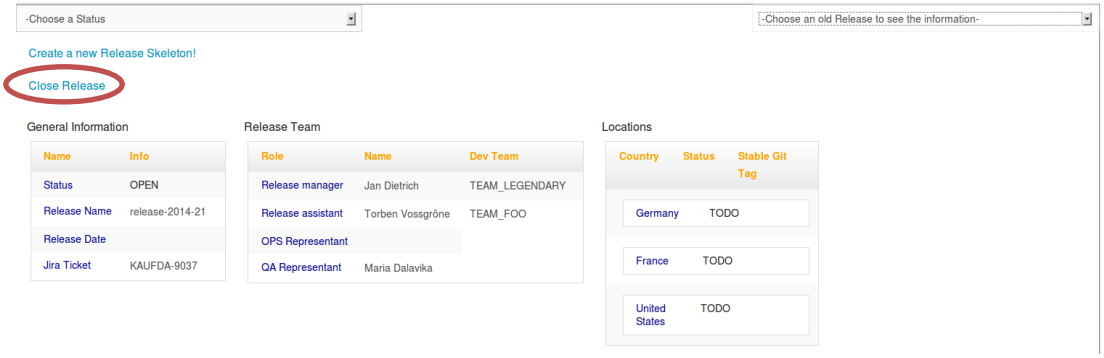


Ilustración 49. Manual de usuario - Cierre de Release

15.6.3.2. Sección Jira

Contiene tareas exclusivamente relacionadas con Jira. En esta sección se pueden ver todas las "fixVersions" existentes en Jira para la organización de la aplicación para la que se hace Release y además, mediante la selección de una fixVersion específica, se puede observar la lista de tickets marcados con esa fixVersion, o sea, la lista de tickets incluidos en la Release a la que hace referencia la fixVersion seleccionada.

La lista de tickets se muestra en distintos colores atendiendo a la naturaleza del ticket. El color rojo se utiliza para aquellos tickets creados para la resolución de Impediments (errores que bloquean el sistema y que necesitan una rápida solución). El color amarillo se utiliza pra marcar aquellos tickets creados para la resolución de bugs del sistema. Finalmente, el color verde se utiliza para marcar el resto de tickets que serán, en su mayoría, nuevas funcionalidades del sistema.

Para cada ticket se ofrece un link al ticket en Jira, el título del ticket y el nombre del desarrollador asignado que ha implementado la funcionalidad o arreglo requerido.

Ilustración 50. Manual de usuario - Selección de fixVersion

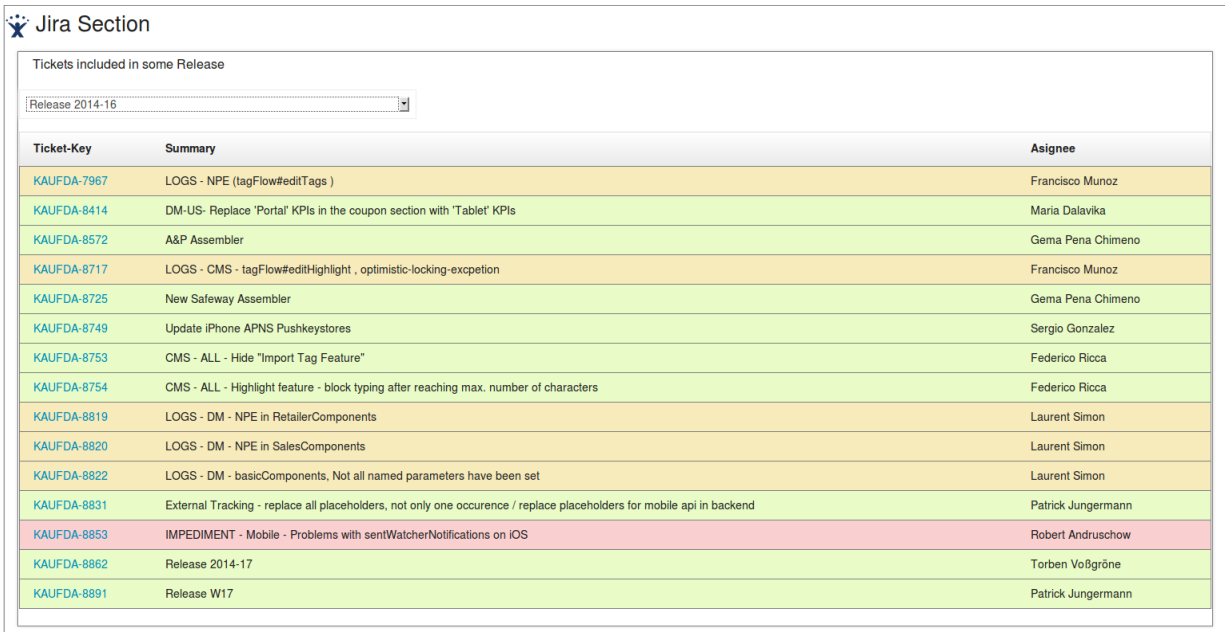


Ilustración 51. Manual de usuario - Visualización de tickets incluidos en una Release

La otra funcionalidad que ofrece esta sección está enfocada al análisis de los tickets creados para las distintas Releases. Se muestra una tabla que contiene información referente a los tickets asociados a cada ReleasePage creada en la aplicación.

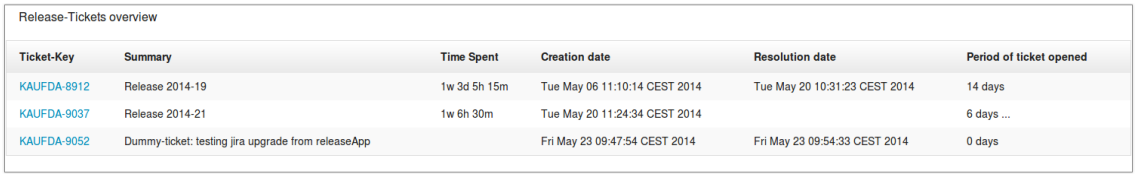


Ilustración 52. Manual de usuario - Visualización del resumen de tickets de Releases

15.6.4. Management

Este área permite acceder a las funcionalidades de gestión de la aplicación. Hasta el momento, el único componente para el que es necesario realizar alguna gestión es la Checklist, por lo tanto sólo se ofrecen funcionalidades relacionadas con la creación, edición y borrado de los componentes de la Checklist.

Se muestran todas las Checklists diarias creadas, con toda la información relativa y los distintos botones de edición. Desde esta vista se puede acceder a vistas de edición o eliminar cada Checklist diaria.

Además, de forma global se ofrecen las funcionalidades de creación de una Checklist diaria o un Checkpoint.

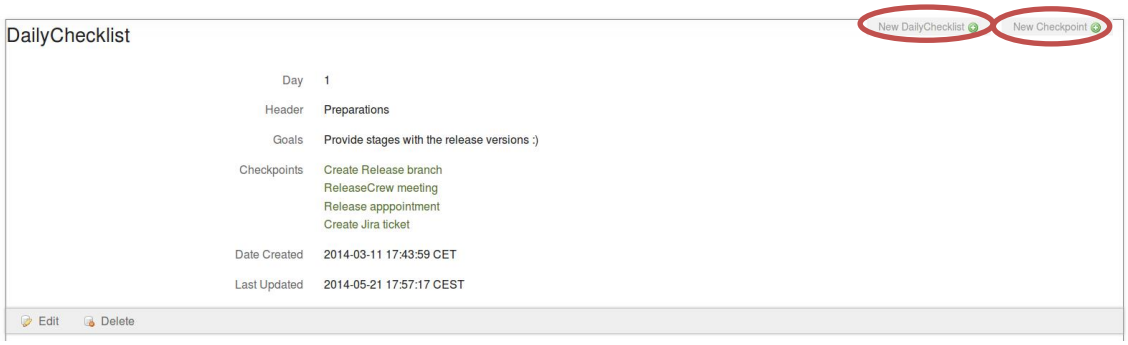


Ilustración 53. Manual de usuario - Gestión de Checklist diaria

Desde la vista de edición de una Checklist diaria se puede acceder además al resto de funcionalidades de edición de los Checkpoints relacionados.

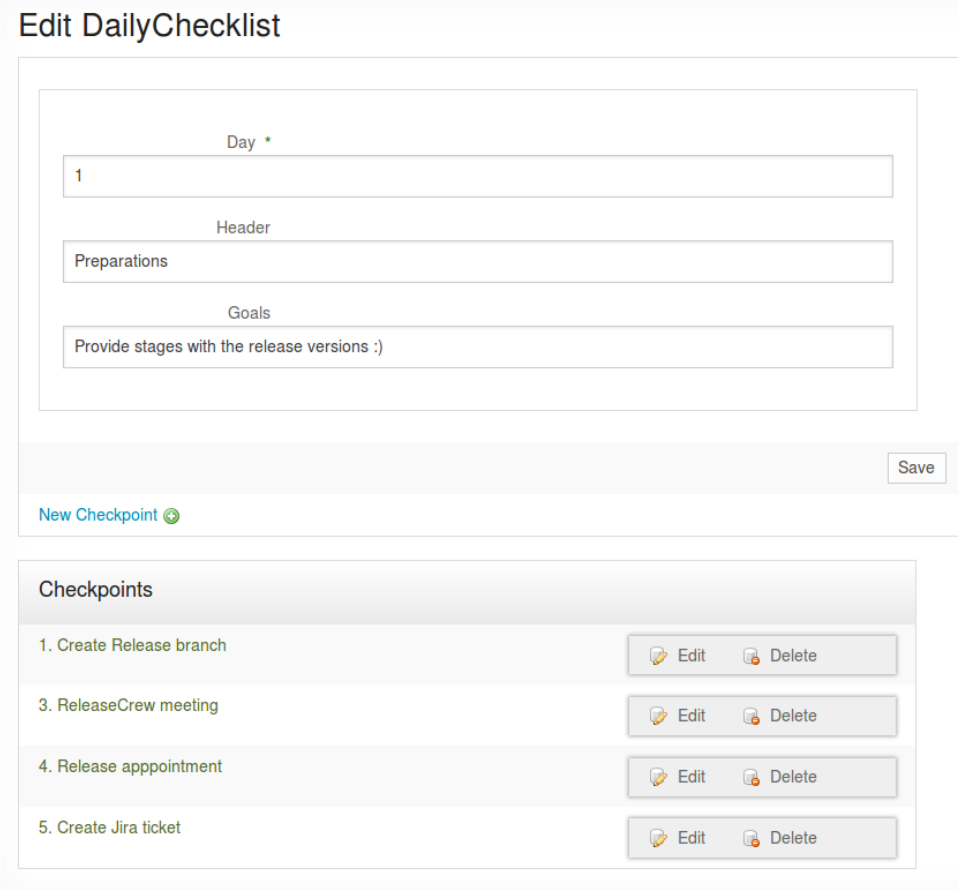


Ilustración 54. Manual de usuario - Edición de Checklist diaria

15.6.5. Events

En este área se muestran por defecto todos los eventos almacenados en base de datos, divididos en distintos grupos según el tipo de evento. Los eventos se encuentran ordenados de forma descendente para que siempre los primeros eventos visibles sean los que han ocurrido recientemente.

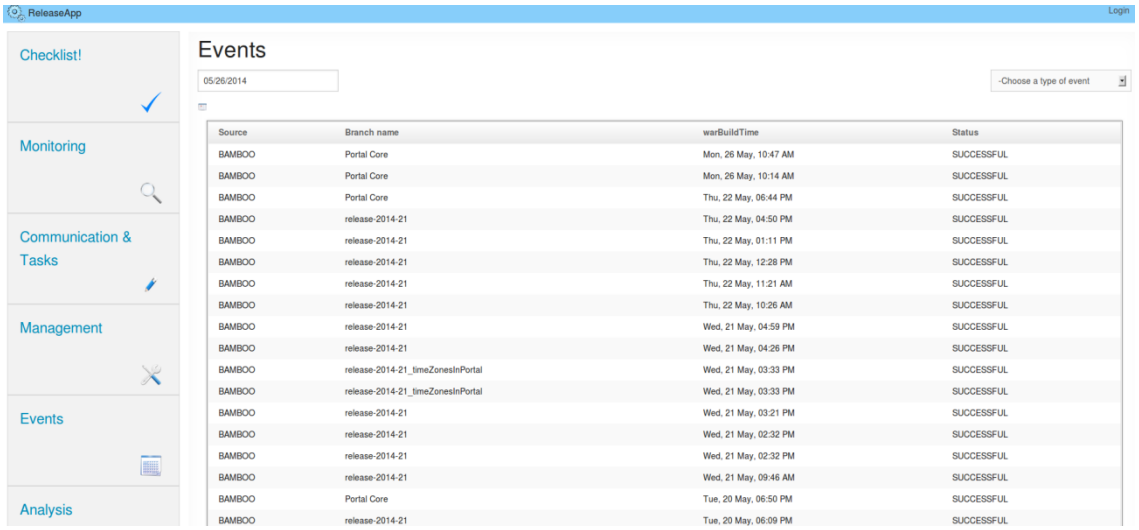


Ilustración 55. Manual de usuario - Área de Eventos

Se proporcionan 2 filtros, **independientes** entre sí y **excluyentes**, que permiten un mejor manejo de los eventos:

- Se pueden filtrar los eventos atendiendo a su fuente (o tipo de evento).

Ilustración 56. Manual de usuario - Filtrado de eventos por fuente

- Se pueden filtrar los eventos según la fecha en que ocurrieron, mediante la selección de alguna fecha en el calendario ofrecido.

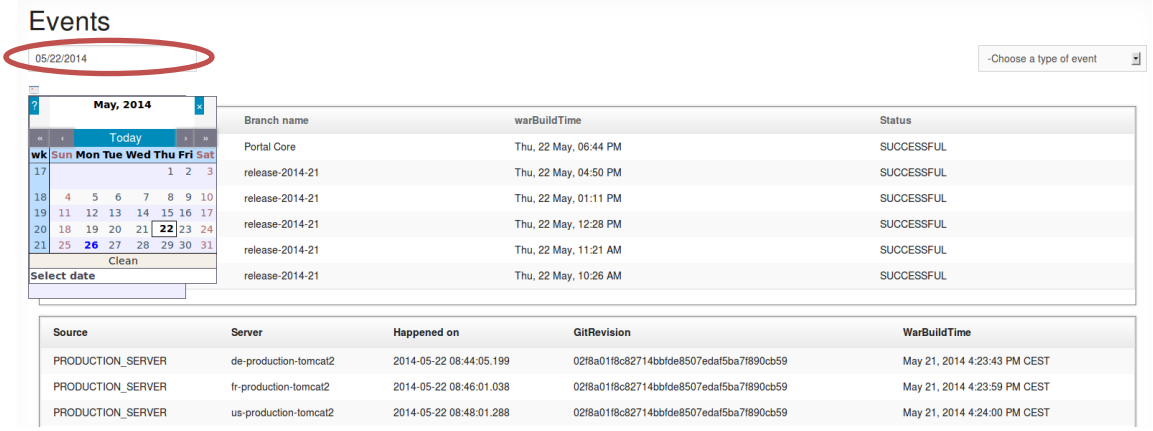


Ilustración 57. Manual de usuario - Filtrado de eventos por fecha

15.6.6. Analysis

En este área se muestran los análisis generados para cada Release en el momento en que se cierra la Release desde la aplicación. Dichos análisis son los mismos que se enviaron en su momento vía email a los usuarios interesados y que ahora se encuentran almacenados en base de datos. La representación de los análisis permite realiazar un análisis y comparación entre las distintas Releases. La información mostrada en cada análisis se puede apreciar en la siguiente imagen.

Además, desde este área se puede generar un análisis sobre cualquiera de las Releases que se encuentran abiertas y activas en un momento dado, con el fin de saber el estado actual de la Release y tener una previa idea acerca de su progreso y evolución. Esta acción no generará el envío de ningún email y este análisis no se hará persistente en base de datos ya que no corresponde a un análisis final de la Release. Para generar un análisis instantáneo de una Release abierta basta con seleccionar el nombre de la Release en el selector marcado en la siguiente imagen. En este selector sólo se podrán elegir aquellas Releases que se encuentren abiertas.

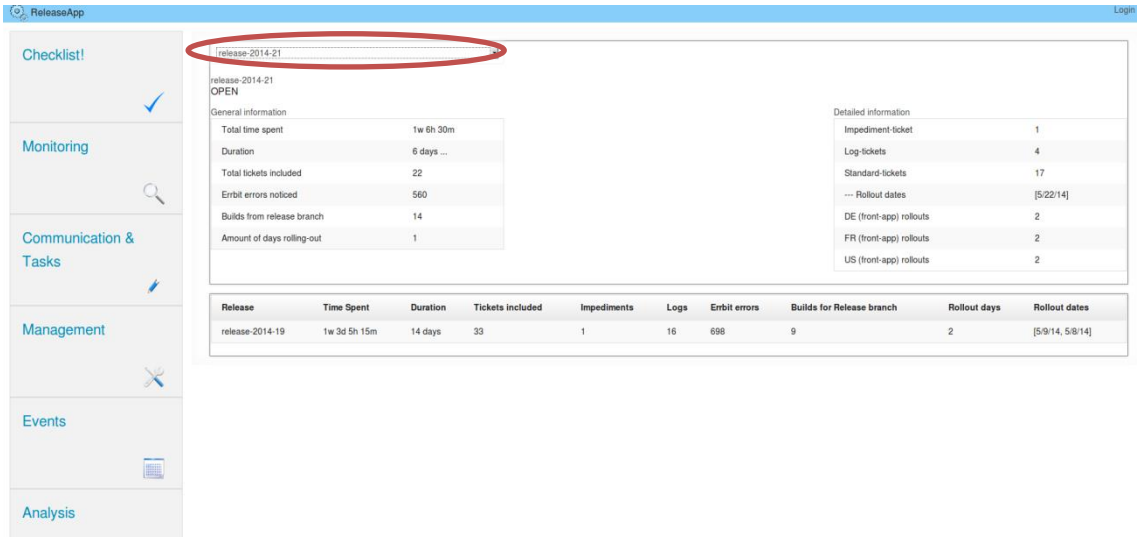


Ilustración 58. Manual de usuario - Área de análisis