UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Sistema de visualización pública de estado de los sistemas integrado con Zabbix

Autor:
Sergio García Spínola

Tutores:
Tomás Fernández Pena
Fernando Guillén Camba
Jorge Suárez de Lis

Grado en Ingeniería Informática

Septiembre 2015

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática

D. Tomás Fernández Pena, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, **D. Fernando Guillén Camba** y **D. Jorge Suárez de Lis**.
INFORMAN:

Que la presente memoria, titulada *Sistema de visualización pública de estado de los sistemas integrado con Zabbix*, presentada por **D. Sergio García Spínola** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación de Grado en Ingeniería Informática, se realizó bajo nuestra dirección en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 4 de septiembre de 2015:

El tutor,	el cotutor	el cotutor	el alumno,
Tomás Fernández Pena	Fernando Guillén Camba	Jorge Suárez de Lis	Sergio García Spínola

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este trabajo de fin de grado busca responder a una necesidad del personal de administración de sistemas de un centro de investigación: poder establecer una comunicación con el público de manera eficaz y sencilla en aquellas ocasiones en que sea necesario informar del estado de los sistemas y de paradas programadas y no programadas.

En la actualidad, están utilizando el sistema de monitorización Zabbix. Este tipo de herramientas se centran en las funciones del administrador de cara al sistema, dejando un vacío en lo que se refiere a la interacción con los usuarios. La solución planteada para resolver este problema y desarrollar este trabajo de fin de grado sigue el modelo empleado por Google en su "pizarra de estado" para servicios, que encontramos en la web *Google Apps Status Dashboard*[1].

La herramienta Zabbix es empleada con el objetivo de recabar información en tiempo real sobre el estado de un equipo, una máquina virtual, una base de datos o incluso algunos componentes hardware, a los que internamente denomina *hosts*. A través de Zabbix los administradores tienen la posibilidad de obtener información de manera selectiva, es decir, pueden especificar qué parámetros se quiere observar en cada *host*. Por ejemplo, se puede conocer la cantidad de memoria libre en disco, la cantidad de procesos ejecutándose o si el *host* se ha reiniciado. Las opciones que ofrece esta herramienta son numerosas y variadas. Su funcionalidad no se limita a recoger información, sino que proporciona métodos de análisis y evaluación. Nuestro interés se centra en los elementos denominados *triggers*, ya que gran parte de la funcionalidad de la aplicación se desarrollará sobre ellos.

Los *triggers* en el entorno Zabbix se deben entender como la definición de una condición que, cuando se cumple, cambia su estado alertando a los administradores. Para formar un *trigger* se define a qué *host* afecta y en qué forma se evalúa un determinado parámetro de los que se observan para ese *host*. Por ejemplo, se podría definir un *trigger* que nos advierta cuando quede menos de un diez por

ciento de memoria libre en un determinado *host*. Así, con cada uno de los valores que Zabbix recibe periódicamente sobre la cantidad de memoria libre, calculará si el valor actual es inferior a un diez por ciento del total de memoria. Cuando esto se cumpla, modificará el estado del *trigger* activándolo.

Estos cambios de estado serán una pieza muy importante dentro de la lógica de la aplicación, puesto que con cada cambio que implique la activación de un *trigger* se proporcionará al administrador la posibilidad de publicar información para el conocimiento de los usuarios.

1.1. Objetivos

A continuación se enumeran los objetivos generales de la aplicación.

- Permitir a los administradores de sistemas comunicar la aparición de incidencias puntuales, ya sean generadas de manera automática a partir de los *triggers* de Zabbix, introducidas a mano o programadas con antelación.
- Comunicar a los usuarios el estado de los sistemas mediante una tabla temporal en la que se listen individualmente y que posibilite la ampliación de la información.

Como objetivos secundarios, se definen:

- Presentar una interfaz funcional e intuitiva con independencia del dispositivo desde el que se acceda.
- Basar el desarrollo de la aplicación en herramientas y software libre.

1.2. Estructura del presente documento

En esta memoria se documentan las distintas fases y procesos llevados a cabo para la realización de este TFG, explicando cada uno de ellos y las decisiones tomadas.

- En el **capítulo dos** se explica la metodología empleada en este proyecto.
- En el **capítulo tres** se detalla la recolección de requisitos realizada.
- En el **capítulo cuatro** se explican los procesos de gestión empleados en este proyecto en cuanto a planificación temporal, gestión de riesgos, análisis de costes y control de versiones.
- En el **capítulo cinco** se listan las tecnologías y herramientas seleccionadas y su motivación.

- En el **capítulo seis** se recoge la organización de los requisitos para su desarrollo.
- En el **capítulo siete** explican las decisiones de diseño tomadas y detalles de aspectos importantes de la implementación realizada.
- En el **capítulo ocho** se recogen las pruebas realizadas y sus resultados.
- En el **capítulo nueve** se recogen las conclusiones del trabajo realizado y se citan posibles ampliaciones o mejoras para la aplicación.
- Por último, se disponen dos **anexos** que comprenden un manual de despliegue de la aplicación y un manual de usuario.

Capítulo 2

Metodología del proyecto

Las metodologías ágiles representan una alternativa a la gestión de proyectos tradicional empleada en desarrollos software. En este tipo de metodologías se prioriza la funcionalidad sobre la documentación, al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas, la colaboración con el cliente y la respuesta ante los cambios. Pueden consultarse los principios del desarrollo ágil en su manifiesto[1].

2.1. Scrum

Se trata de un proyecto en que los requisitos no están claramente definidos en un inicio y que se verán sujetos a variaciones a lo largo del desarrollo. Un factor limitante es el tiempo, la necesidad de obtener un producto funcional en un período breve. Las características del proyecto hacen que el desarrollo se adapte mejor a metodologías ágiles que a otras clásicas como puede ser el ciclo de vida en cascada. Teniendo en cuenta estos factores, se elige la metodología Scrum, en la que el alcance se define en función del tiempo disponible, los procesos se adaptan a los cambios constantes y no existen largas etapas iniciales.

A continuación explicaremos brevemente el funcionamiento de Scrum apoyándonos en *La Guía de Scrum* [2], de la que tomaremos las definiciones y términos clave. En primer lugar, debemos conocer tres elementos que se emplean en los eventos de Scrum y que representan trabajo o recursos para proporcionar transparencia y oportunidades de adaptación. Los elementos empleados en Scrum se diseñan para maximizar la transparencia de información de forma que tengan el mismo significado para cada componente del equipo. Los principales elementos son:

[1] http://www.agilemanifesto.org/principles.html

1. Lista de Producto o *Product Backlog*: es una lista ordenada de todo lo que podría ser necesario en el producto, y es la única fuente de requisitos para cualquier cambio a realizarse en el producto. Un Product Backlog nunca está completo. El desarrollo más temprano del mismo solo refleja los requisitos conocidos y mejor entendidos al principio. El Product Backlog evoluciona a medida que el producto y el entorno en el que se usará también lo hacen. El Product Backlog es dinámico; cambia constantemente para identificar lo que el producto necesita para ser adecuado, competitivo y útil. Mientras el producto exista, su Product Backlog también existe. El Product Backlog enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a ser hechos sobre el producto para entregas futuras.

2. Lista de Pendientes del Sprint o *Sprint Backlog*: es el conjunto de elementos del Product Backlog seleccionados para el Sprint. Es una predicción hecha por el Equipo de Desarrollo acerca de qué funcionalidad formará parte del próximo Incremento y del trabajo necesario para entregar esa funcionalidad en un Incremento "Terminado". Hace visible todo el trabajo que el Equipo de Desarrollo identifica como necesario para alcanzar el Objetivo del Sprint.

3. Incremento o *Increment*: Es la suma de todos los elementos del Product Backlog completados durante un Sprint y el valor de los incrementos de todos los Sprints anteriores.

En Scrum se utilizan eventos predefinidos para establecer una regularidad y minimizar la necesidad de reuniones no definidas en el propio Scrum. Todo evento tiene un marco temporal con una duración máxima. Un desarrollo en Scrum se compone de *Sprints*, que actúan como contenedores de todos los restantes eventos.

1. *Sprint*: es un bloque de tiempo (time-box) de un mes o menos durante el cual se crea un incremento de producto "Terminado", utilizable y potencialmente desplegable. Cada nuevo Sprint comienza inmediatamente después de la finalización del Sprint previo. Durante el Sprint: no se realizan cambios que puedan afectar al Objetivo del Sprint , los objetivos de calidad no disminuyen y el alcance puede ser clarificado y renegociado entre el Dueño de Producto y el Equipo de Desarrollo a medida que se va aprendiendo más.

2. El trabajo a realizar durante el Sprint se planifica en el *Sprint Planning*, que responde a lo siguiente:

    a) ¿Qué puede entregarse en el Incremento resultante del Sprint que comienza?

    b) ¿Cómo se conseguirá hacer el trabajo necesario para entregar el Incremento?

3. Scrum Diario o *Daily Scrum*: Es un evento diario de quince minutos para el equipo de desarrollo, durante el cual sus miembros explican:

   a) ¿Qué hice ayer para ayudar al equipo de desarrollo a alcanzar el objetivo del Sprint?

   b) ¿Qué haré hoy para ayudar al equipo de desarrollo a alcanzar el objetivo del Sprint?

   c) ¿Veo algún impedimento para mi o el equipo de desarrollo para alcanzar el objetivo del Sprint?

4. Al final de cada Sprint se mantiene una "Revisión del Sprint" o *Sprint Review* para inspeccionar el incremento y adaptar el Product Backlog si es necesario. Se trata de una reunión informal, y la presentación del Incremento tiene como objetivo facilitar la retroalimentación de información y fomentar la colaboración. Incluye los siguientes elementos:

   a) En la reunión están presentes el equipo y los *stakeholders*[2] clave.

   b) El Dueño de Producto explica qué elementos del Product Backlog se han concluido y cuales no.

   c) El equipo de desarrollo discute qué fue bien durante el Sprint, qué problemas se encontró y cómo los solucionó.

   d) El equipo de desarrollo enseña el trabajo realizado y responde a las preguntas sobre el incremento.

   El resultado de la revisión del Sprint es un Product Backlog revisado que define los elementos candidatos del mismo para el próximo Sprint.

5. La retrospectiva de Sprint o *Sprint Retrospective* es una oportunidad para el equipo de examinarse y crear un plan para llevar a cabo mejoras que sean abordadas durante el siguiente Sprint. Su propósito es el de:

   a) Inspeccionar cómo fue el último Sprint en cuanto a personas, relaciones, procesos y herramientas;

   b) Identificar y ordenar los elementos más importantes que salieron bien y las posibles mejoras; y,

   c) Crear un plan para implementar las mejoras a la forma en la que el Equipo Scrum desempeña su trabajo.

Scrum emplea tres roles principales:

[2]Stakeholder es un término en inglés utilizado para referirse a quienes son afectados o pueden ser afectados por las actividades de una empresa.

1. El Dueño de Producto o *Product Owner* es la única persona responsable de gestionar el Product Backlog. Entre sus responsabilidades están las de asegurar que el equipo de desarrollo comprenda con claridad los items del Product Backlog, ordenar los mismos para optimizar la consecución de objetivos y la calidad de los resultados. Es una única persona, pero puede ser representativo de la voluntad de un comité. Todo cambio en el Product Backlog debe realizarse a través del Dueño de Producto.

2. El equipo de desarrollo o *The Development Team* se compone de los profesionales que generan los distintos incrementos del producto a lo largo de los Sprints. Disponen de libertad para la organización y gestión propia de su trabajo con el fin de optimizar la eficiencia y efectividad global del equipo.

3. El *Scrum Master* es el responsable de que Scrum sea entendido y llevado a cabo correctamente, asegurándose de que se sigue la teoría del Scrum, sus prácticas y sus reglas. Algunas de sus tareas son encontrar técnicas efectivas para la gestión del Product Backlog, ayudar al equipo a comprender la necesidad de elaborar historias de usuario claras y concisas o asegurarse de que el Dueño de Producto sabe como optimizar el Product Backlog.

El cuarto principio del manifiesto para el desarrollo ágil de software dice "*Responding to change over following a plan*", es decir, priorizar el responder a los cambios ante el seguimiento de un plan. Siguiendo este principio, los *ScrumButs*[16] nos permiten definir nuevas reglas para poder adaptar Scrum a las características de nuestro desarrollo.

Los tutores del proyecto actuarán con el rol de Scrum Masters, mientras que mi papel comprenderá los roles de Dueño de Producto - actuando en representación de un comité (los tutores del proyecto) - y de equipo de desarrollo. Habiendo explicado los aspectos fundamentales de esta metodología, nos encontramos con la necesidad de hacer las siguientes variaciones o *ScrumButs*:

1. Usamos Scrum, pero el equipo de desarrollo se compone de una única persona, por lo que el *Sprint Planning* se limitará a establecer qué se entregará en el incremento.

2. Usamos Scrum, pero el equipo de desarrollo se compone de una única persona, por lo que no se realizarán los Scrum Diarios.

3. Usamos Scrum, pero la retrospectiva de Sprint resulta en una pérdida de tiempo en un proyecto de corta duración y con un equipo de desarrollo de una única persona, por lo que no se realizará.

Capítulo 3

Especificación de requisitos

En este capítulo llevaremos a cabo la recolección de requisitos, entendido como el proceso de determinar, documentar y gestionar las necesidades de los *stakeholders* y los requisitos para alcanzar los objetivos del proyecto. El beneficio clave de este proceso es que proporciona las bases para definir y gestionar el alcance del proyecto.[9]

La recolección de requisitos con el uso de *Scrum* varía con respecto al método tradicional. En este caso se emplean las denominadas "*historias de usuario*". Se trata de representaciones de requisitos de una o dos frases escritas utilizando lenguaje comprensible por el usuario. En muchos casos estas *historias de usuario* son en un principio demasiado grandes y son descompuestas hasta que constituyan una unidad de trabajo mínima. Una vez definidas, el equipo puede realizar anotaciones con el fin de aclarar aspectos de su funcionalidad, diseño o implementación.

Como se explicó en el capítulo anterior, el Product Backlog reúne todo lo que podría ser necesario en el producto, mientras que el Sprint Backlog es el conjunto de elementos del Product Backlog seleccionados para el Sprint.

3.1. Product Backlog

A continuación, en los cuadros 3.1 a 3.30, se lista el conjunto de historias de usuario que componen el Product Backlog.

Visualizar la línea temporal de incidencias de servicios
- Dado un usuario. - Cuando acceda a la aplicación. - Deberá mostrarse una tabla temporal, con servicios por filas y fechas por columnas, en cuyas celdas se representen las incidencias de la semana actual.
- Se proporcionará al usuario la posibilidad de navegar en la tabla para visualizar semanas anteriores, con un máximo de dos meses atrás, así como a la próxima semana. En caso de haber una incidencia en la siguiente semana que se conozca de antemano, se señalizará con un icono al final de la fila del servicio en cuestión. En aquellas incidencias cuya duración supere el día de comienzo se extenderá el icono empleado hasta el día en que finalice. - Cada uno de los nombres de servicios podrá funcionar de enlace a una página relacionada según haya configurado el administrador, y cada uno de los iconos llevará a la visualización de los mensajes de la incidencia en una nueva página. El icono tendrá el mismo color que el icono del último mensaje asociado a la misma.

Cuadro 3.1: HU00

Visualizar los mensajes asociados a una incidencia
- Dado un usuario. - Cuando pulse sobre una entrada en la tabla temporal. - Deberán mostrarse los mensajes asociados a esa incidencia.
- Los mensajes se mostrarán en una nueva página, dispuestos sobre una tabla en la que se especifique la hora (primero el más actual) y gravedad del mensaje (mediante el circulo coloreado o un icono similar). Se habilitará una sección en los mensajes en la que se puedan disponer enlaces de ayuda y/o información. El color del icono del último mensaje publicado marcará el color del icono de la incidencia en la tabla temporal.

Cuadro 3.2: HU01

Identificarse en la aplicación
- Como administrador no identificado - Quieroidentificarme en la aplicación - Para acceder al panel con funcionalidades de administrador
- Se dispondrá un enlace en el footer de la aplicación para el acceso del administrador que llevará a una página de LogIn

Cuadro 3.3: HU02

Configurar tiempo de almacenamiento de datos
Configurar tiempo de almacenamiento de datos- Como administrado - Quiero poder establecer durante cuánto tiempo se almacenan los datos recabados - Para así poder mantenerlos durante el tiempo que se precise y con un tamaño acorde a las necesidades.

Cuadro 3.4: HU03

Notificar una incidencia por emai
- Dado una disparador de cualquier tipo registrado en la aplicación - Cuando se produzca una incidencia relacionada - Deberá notificarse a los administradores vía email.
- El correo deberá especificar si es un disparador autónomo o supeditado, y proporcionar enlaces para su administración

Cuadro 3.5: HU04

Añadir un servicio
- Como administrador - Quiero poder añadir un nuevo servicio - Para visualizar en la tabla temporal sus incidencias.
- En el proceso de creación se proporcionará un nombre y, opcionalmente, un enlace a una página del servicio. Se podrá proporcionar también un enlace de ayuda para mostrar en la página de mensa jes de incidencias. Se comprobará que el nombre no exista ya en la aplicación.

Cuadro 3.6: HU05

Listar los servicios registrados en la aplicación
- Como un administrador - Quiero listar los servicios registrados - Para poder visualizar todos los presentes en la aplicación

Cuadro 3.7: HU06

Modificar un servicio
- Dado un administrador - Cuando seleccione un servicio de la lista de servicios registrados - Deberá poder modificar sus datos
- Se dispondrá un icono junto con cada servicio de la lista que lleve a un formulario de modificación de sus datos.

Cuadro 3.8: HU07

Eliminar un servicio
- Dado un administrador - Cuando seleccione un servicio de la lista de servicios registrados - Deberá poder eliminar el mismo
- Se dispondrá un icono junto con cada servicio de la lista que permita la eliminación de sus datos

Cuadro 3.9: HU08

Añadir un Servidor Zabbix
- Como administrador - Quiero poder añadir un Servidor Zabbix - Para establecer disparadores sobre los triggers presentes en él.
- Al añadir un servidor deberá comprobarse que el usuario y contraseña proporcionados permiten identificarse en él, que la dirección IP es alcanzable y que el nombre identificativo no está repetido en la aplicación.

Cuadro 3.10: HU09

Listar Servidores Zabbix en la aplicación
- Como un administrador - Quiero listar los Servidores Zabbix registrados - Para poder visualizar todos los presentes en la aplicación.

Cuadro 3.11: HU10

Modificar Servidor Zabbix
- Dado un administrador - Cuando seleccione un Servidor Zabbix de la lista de servidores registrados - Deberá poder modificar sus datos
- Se dispondrá un icono junto con cada Servidor Zabbix de la lista que lleve a un formulario de modificación de sus datos.

Cuadro 3.12: HU11

Eliminar Servidor Zabbix
- Dado un administrado r- Cuando seleccione un Servidor Zabbix de la lista de Servidores Zabbix registrados - Deberá poder eliminar el mismo.
- Se dispondrá un icono junto con cada Servidor Zabbix de la lista que permita la eliminación de sus datos

Cuadro 3.13: HU12

Añadir un disparador
- Como administrador - Quiero poder añadir un disparador a la aplicación - Para poder asociarlo a un servicio.
- Al añadir un disparador deberá indicarse un nombre identificativo y elegir entre los servicios presentes en la aplicación a cuál asociarlo.

Cuadro 3.14: HU13

Seleccionar el tipo de un disparador
- Dado un administrador - Cuando esté registrando un nuevo disparador en la aplicación - Deberá poder seleccionar el tipo de disparador que será.
- Habrá dos tipos de disparadores: 1. - *Autónomos*: ante la aparición de incidencias se publicarán automáticamente en la tabla temporal junto con un mensaje establecido, por lo que se deberá indicar un mensaje a mostrar por defecto. 2. - *Supeditados*: ante la aparición de incidencias, se esperará a la autorización del administrador para publicar información al respecto en la aplicación.

Cuadro 3.15: HU14

Establecer mensaje por defecto para un disparador
- Dado un administrador - Cuando esté registrando un nuevo disparador en la aplicación - Deberá poder establecer un mensaje por defecto para mostrarse ante la aparición de incidencias.
- En el caso de los disparadores autónomos será obligatorio

Cuadro 3.16: HU15

Vincular un trigger de Zabbix a un disparador
- Dado un administrador - Cuando esté registrando un nuevo disparador en la aplicación - Deberá poder vincular un trigger de Zabbix a la misma sobre el que observar sus cambios de estado
- En el formulario de registro del disparador se le presentará una lista de los Servidores Zabbix y otra de los Servicios registrados en la aplicación y, tras seleccionar uno de cada lista, se presentará una lista de los triggers activos en el servidor y que no han sido usados ya en la aplicación para otros disparadores del mismo servicio.

Cuadro 3.17: HU16

Configurar cierre automático de incidencia
- Dado un administrador - Cuando esté registrando un nuevo disparador en la aplicación - Deberá poder configurar que las incidencias se den por finalizadas automáticamente cuando el trigger recupere su estado de corrección en el servidor de Zabbix
- Si se selecciona que el disparador tenga este comportamiento deberá establecerse un mensaje por defecto que se publique cuando se cierre la incidencia, por lo que en el formulario de registro deberá habilitarse un campo para introducirlo.

Cuadro 3.18: HU17

Modificar disparador
- Dado un administrador - Cuando seleccione un disparador de la lista de disparadores registrados en la aplicación - Deberá poder modificar sus datos.
- Se dispondrá de un icono junto con cada disparador de la lista que permita acceder a un formulario en el que modificar sus datos. - El cambio de tipo supondrá: 1. De supeditado a autónomo: de no existir, deberá proporcionarse un mensaje por defecto, tanto de apertura como de cierre de la incidencia. 2. De autónomo a supeditado: la posibilidad de descartar los mensaje por defecto - El cambio de la configuración de cierre automático: 1. Si se desactiva, se descartará el mensaje por defecto para el cierre. 2. Si se activa, deberá establecerse el mensaje para el cierre. - El cambio de nombre identificativo o de trigger estaría restringido a que no exista ya en la basede datos o no esté en uso por otro disparador del mismo servicio, correspondientemente. - Se incluye aquí el caso de que no se haya introducido durante el registro un mensaje por defecto y se quiera incluir como modificación.

Cuadro 3.19: HU19

Listar los disparadores de la aplicación
- Como un administrador - Quiero listar los disparadores registrados - Para poder tener visualizar todas las presentes en la aplicación
- Se podrán filtrar por servicios

Cuadro 3.20: HU18

Eliminar disparador
- Dado un administrador - Cuando seleccione un disparador de la lista de disparadores registrados en la aplicación - Deberá poder eliminar sus datos
- Se dispondrá un icono junto con cada disparador de la lista que permita su eliminación.

Cuadro 3.21: HU20

Publicar una incidencia manual
- Como administrado - Quiero poder introducir de manera manual una incidencia en la aplicación sin que se vincule a ningún disparador - Para poder advertir a los usuarios de incidentes que se conozcan de antemano.
- Se cubrirá un formulario en el que se seleccione servicio afectado, se introduzca el mensaje a publicar y se establezcan las fechas y horas de inicio y fin de la incidencia - Durante el período de la incidencia se hará caso omiso del resto de disparadores establecidos para el servicio

Cuadro 3.22: HU21

Listar incidencias
- Como administrador - Quiero poder listar las incidencias - Para poder visualizar todas las registradas en la aplicación
- Se pondrán de primeras las incidencias que requieran aprobación del administrador para su publicación.

Cuadro 3.23: HU22

Agregar mensaje a incidencia
- Dado un administrador - Cuando seleccione una incidencia de la lista de incidencias de la aplicación - Deberá poder agregar un nuevo mensaje a la serie de mensajes asociados a la incidencia
- Se dispondrá de un icono junto con cada incidencia de la lista que permita la introducción de un nuevo mensaje

Cuadro 3.24: HU23

Modificar mensaje de incidencia
- Dado un administrador - Cuando seleccione una incidencia de la lista de incidencias de la aplicación - Deberá poder modificar los mensajes asociados a la misma.
- Se dispondrá un icono junto con cada incidencia de la lista que permita el acceso a la modificación de la misma. - Se mostrará un formulario de modificación con los distintos mensajes presentes. - Si se modifica el primer mensaje publicado, cuando se trate de un mensaje por defecto, en lugar de sobrescribir éste, a efectos de la aplicación será como sustituir el mensaje por uno nuevo, de manera que el por defecto no se vea afectado para futuras incidencias.

Cuadro 3.25: HU24

Eliminar mensaje de incidencia
- Dado un administrador - Cuando seleccione una incidencia de la lista de incidencias de la aplicación - Deberá poder eliminar mensajes asociados a la misma.
- Se dispondrá un icono junto con cada incidencia de la lista que permita el acceso a la modificación de la misma (será el mismo que en la HU "Modificar mensaje de incidencia"). - Nunca se podrá eliminar la totalidad de los mensajes, siempre deberá quedar al menos un mensaje asociado a la incidencia

Cuadro 3.26: HU25

Eliminar incidencia
- Dado un administrador - Cuando seleccione una incidencia de la lista de incidencias de la aplicación - Deberá poder eliminar los datos de la misma
- Se dispondrá un icono junto con cada incidencia de la lista que permita su eliminación.

Cuadro 3.27: HU26

Finalizar incidencia
- Dado un administrador - Cuando seleccione una incidencia de la lista de incidencias de la aplicación que no esté finalizada - Deberá poder establecerla como tal.
- Se dispondrá un icono junto con cada incidencia de la lista que permita marcarla como finalizada y así no se siga extendiendo la incidencia en la tabla temporal. - Si no existe un mensaje de cierre por defecto para la incidencia, se deberá introducir uno.

Cuadro 3.28: HU27

Autorizar la publicación de una incidencia
- Dado un administrador - Cuando seleccione una incidencia de disparador no autónomo de la lista de incidencias de la aplicación y esta no haya sido publicada todavía - Deberá poder autorizar su publicación para que aparezca en la tabla temporal.
- Se dispondrá un icono junto con cada incidencia de la lista que permita al administrador autorizarla publicación

Cuadro 3.29: HU28

Descartar una incidencia
- Dado un administrador - Cuando seleccione una incidencia del listado general que aún no haya sido publicada - Podrá indicar que la incidencia no se llegará a publicar, de modo que no aparezca en la cabecera de la lista.
- Se dispondrá un icono junto con cada incidencia de la lista que permita al administrador descartarla publicación

Cuadro 3.30: HU29

Capítulo 4

Planificación y gestión

En este capítulo se explicarán los procesos empleados en las primeras etapas del proyecto para su gestión en términos temporales, económicos, de control de versiones y de análisis de riesgos.

4.1. Planificación Temporal

En este apartado se mostrará y explicará la planificación elaborada para el proyecto en desarrollo. Con el fin de identificar las tareas de las que se compone, emplearemos la herramienta conocida como *Estructura de Descomposición de Trabajo* o EDT. Esta consiste en la descomposición jerárquica del trabajo requerido para completar el proyecto, como podemos ver en la figura 4.1, extraídos del análisis de requisitos del capítulo 3.

Se muestra en la figura 4.2 una vista completa de la planificación del proyecto, entrando más tarde en detalle sobre cada una de las fases que lo componen. Como se ha explicado anteriormente, el proyecto emplea la metodología Scrum. Teniendo en cuenta el Product Backlog inicial, se han planificado cuatro Sprints de desarrollo de una duración de 15 días, a comenzar tras la etapa inicial de recolección de requisitos, y una etapa final en la que se llevará a cabo el despliegue del proyecto en los servidores del cliente.

Sprint 1

Como se puede ver en la figura 4.3, en el primer Sprint se desarrolla la funcionalidad necesaria para identificar a usuarios en la aplicación y para los servicios. Estos últimos representan la entidad más sencilla en cuanto a la lógica que la rodea, lo que facilitará el trabajo en esta primera etapa en que el desarrollador se familiariza con nuevas tecnologías y herramientas. También se desarrolla la

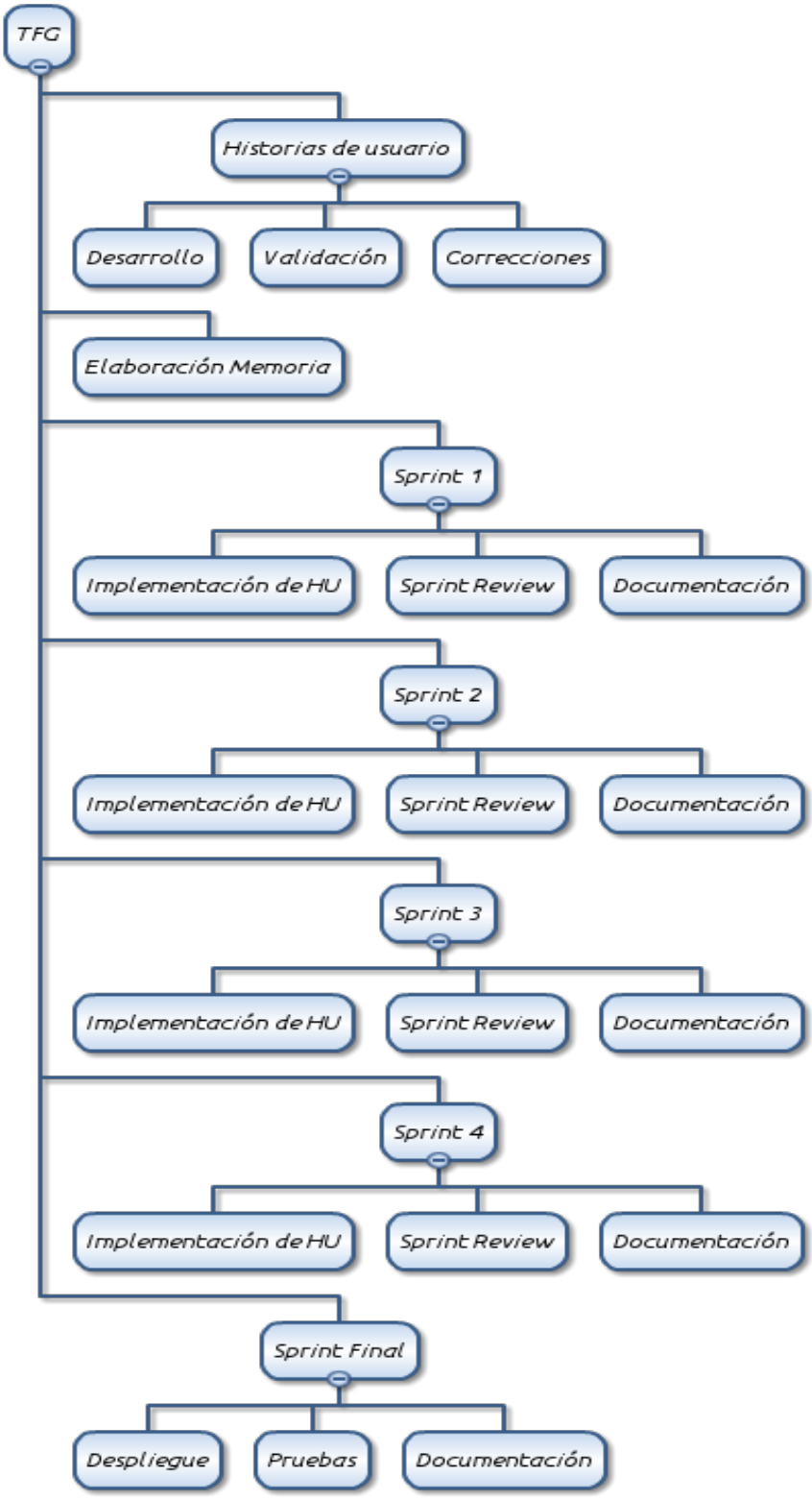


Figura 4.1: EDT del proyecto

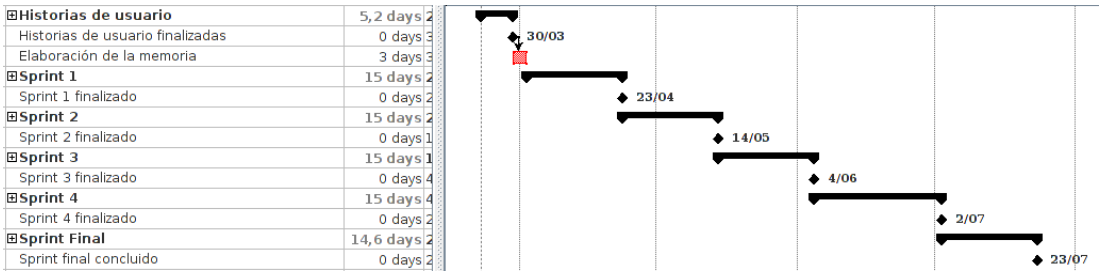


Figura 4.2: Planificación temporal

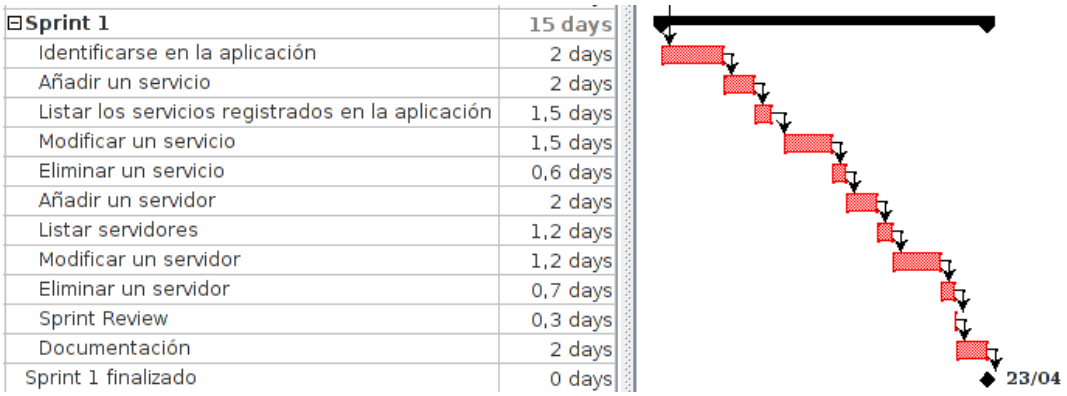


Figura 4.3: Cronograma del Sprint 1.

funcionalidad relativa a los servidores, lo que incrementa en cierto grado la complejidad del trabajo, ya que en estas tareas se entra en contacto con la API de Zabbix y el uso de llamadas AJAX.

Sprint 2

El segundo Sprint se centra en los disparadores y se comienza a desarrollar funcionalidad relativa a las incidencias. Será en este momento en el que se precise la codificación de demonios que actualicen el estado de la base de datos de la aplicación con datos obtenidos de los servidores de Zabbix. Se trata de una parte muy importante de la aplicación, ya que los cambios de estado de los disparadores serán los que marquen la aparición de incidencias.

Sprint 3

Este penúltimo Sprint se centra exclusivamente en las incidencias, introduciendo también las incidencias manuales o programadas por el administrador. Se desarrolla funcionalidad para alterar libremente el estado de las incidencias, ya sean generadas por los disparadores o introducidas por el administrador, y también funcionalidad relativa a los mensajes publicados con cada una de ellas y el

Sprint 2	15 days
Añadir un disparador	1 day
Vincular un trigger a un disparador	1,5 days
Seleccionar el tipo de un disparador	1,6 days
Listar disparadores	1 day
Establecer mensaje por defecto de apertura	0,7 days
Configurar cierre automático de incidencias	2 days
Modificar un disparador	1,5 days
Eliminar disparador	0,6 days
Listar incidencias	1 day
Publicar incidencia	0,6 days
Agregar mensaje a incidencia	0,6 days
Eliminar incidencia	0,6 days
Sprint Review	0,3 days
Documentación	2 days
Sprint 2 finalizado	0 days

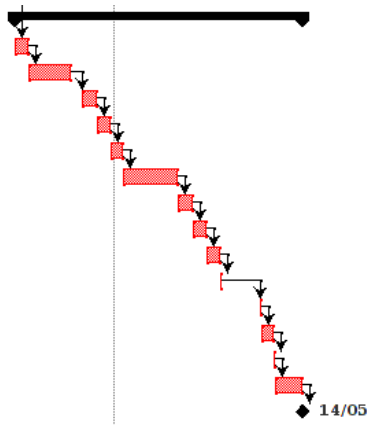


Figura 4.4: Cronograma del Sprint 2.

Sprint 3	15 days
Establecer mensaje por defecto de cierre	0,5 days
Modificar mensaje de incidencia	0,5 days
Eliminar mensaje de incidencia	0,5 days
Descartar una incidencia	1,25 days
Finalizar una incidencia	1,25 days
Listar incidencias manuales	1 day
Añadir incidencia manual	1,6 days
Establecer mensaje por defecto para apertura	0,5 days
Establecer mensaje por defecto para el cierre	0,5 days
Publicar incidencia manual	1,5 days
Añadir mensaje a incidencia manual	1 day
Editar mensaje de incidencia manual	0,5 days
Eliminar mensaje de incidencia manual	0,5 days
Eliminar incidencia manual	0,5 days
Notificar una incidencia por email	1 day
Sprint Review	0,4 days
Documentación	2 days
Sprint 3 finalizado	0 days

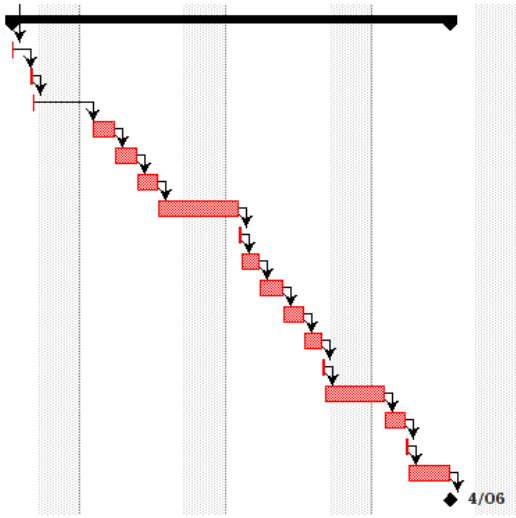


Figura 4.5: Cronograma del Sprint 3.

envío de notificaciones por correo de las incidencias que generan los disparadores.

Sprint 4

En último lugar queda desarrollar la parte pública de la aplicación, aquella de la que podrá hacer uso cualquier usuario. Se implementa la citada tabla temporal, así como la página de ampliación de información de las incidencias individuales.

Sprint final

Pese a compartir la denominación de Sprint, esta última etapa no incluye tareas de implementación. En este período se lleva a cabo el despliegue de la aplicación en los servidores del cliente, realizando las pruebas necesarias para

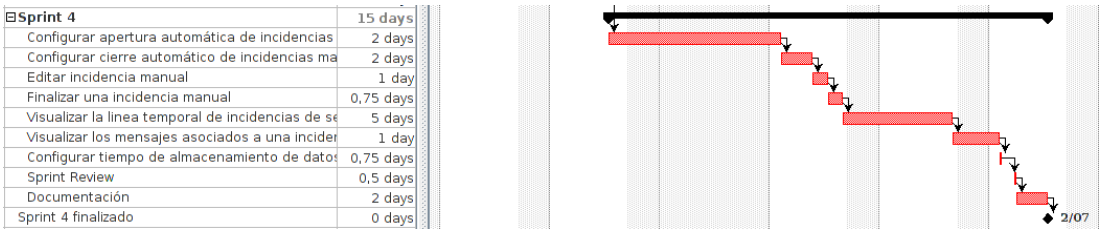


Figura 4.6: Cronograma del Sprint 4.

Figura 4.7: Cronograma del Sprint final.

cercionarse del correcto funcionamiento. Se incluye también en esta etapa la finalización de la memoria, elaborando aquellas partes que no se desarrollaron durante los Sprints anteriores.

4.2. Gestión de Riesgos

Para la elaboración de esta sección se ha tomado como referencia el capítulo homónimo del *PMBOK*[9], sirviendo como fuente de definiciones y base de procesos a seguir.

La gestión de riesgos del proyecto incluye los procesos de planificación de la misma, identificación, análisis, plan de respuesta y control de riesgos en el proyecto. Sus objetivos son incrementar la posibilidad y el impacto de eventos positivos, así como disminuir la posibilidad y el impacto de eventos negativos en el proyecto.

Un riesgo del proyecto es un evento o condición incierta que, de ocurrir, tiene un efecto positivo o negativo para uno o más objetivos del proyecto como el alcance, la planificación, el coste o la calidad. Un riesgo puede tener una o múltiples causas y, de ocurrir, puede tener más de un impacto. Una causa puede ser un requisito o suposición dados o potenciales, una restricción, o una condición que crea la posibilidad de resultados positivos o negativos.

Los riesgos de un proyecto tienen su origen en la incertidumbre presente en todos ellos. Los riesgos conocidos son aquellos que han sido identificados y analizados, posibilitando la planificación de respuestas a éstos.

Definimos a continuación la escala de probabilidad e impacto mediante la tabla 4.8 extraída del *PMBOK*.

Defined Conditions for Impact Scales of a Risk on Major Project Objectives (Examples are shown for negative impacts only)					
Project Objective	Relative or numerical scales are shown				
	Very low /0.05	Low /0.10	Moderate /0.20	High /0.40	Very high /0.80
Cost	Insignificant cost increase	< 10% cost increase	10 – 20% cost increase	20 – 40% cost increase	> 40% cost increase
Time	Insignificant time increase	< 5% time increase	5 – 10% time increase	10 – 20% time increase	> 20% time increase
Scope	Scope decrease barely noticeable	Minor areas of scope affected	Major areas of scope affected	Scope reduction unacceptable to sponsor	Project end item is effectively useless
Quality	Quality degradation barely noticeable	Only very demanding applications are affected	Quality reduction requires sponsor approval	Quality reduction unacceptable to sponsor	Project end item is effectively useless

Figura 4.8: Definition of Impact Scales for Four Project Objectives

En la figura 4.1 tomamos de la asignatura de *Gestión de proyectos informáticos* una escala del nivel de exposición relacionando la probabilidad y el impacto de los riesgos.

		Probabilidad		
		Alta	Media	Baja
Impacto	Alto	Alta	Alta	Media
	Medio	Alta	Media	Baja
	Bajo	Media	Baja	Baja

Cuadro 4.1: Escala de exposición

Con el fin de identificar los riesgos presentes en el proyecto, emplearemos las siguientes técnicas definidas en el *PMBOK* :

1. Revisión de la documentación: planes, suposiciones, archivos del proyecto previos, acuerdos y cualquier otra información se somete a una revisión estructurada.

2. *Brainstorming*. Se generan ideas sobre los riesgos del proyecto bajo el liderazgo de un facilitador, bien en una sesión tradicional de *brainstorming* o estructurado con técnicas de entrevistas en masa.

3. Análisis con lista de verificación: se desarrollan basándose en información histórica y conocimiento acumulado de proyectos anteriores similares y otras fuentes de información.

Siguiendo de nuevo la técnica empleada en la asignatura de *Gestión de proyectos informáticos*, se proporcionará para cada riesgo encontrado:

1. Código identificativo.
2. Probabilidad.
3. Impacto.
4. Nivel de exposición.
5. Breve descripción del riesgo.
6. Acción preventiva para disminuir la probabilidad de que ocurra el riesgo.
7. Acción correctiva en caso de ocurrir el riesgo.

4.2.1. Riesgos identificados

Se identifican los siguientes riesgos:

Código	R00		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	Los requisitos no se han definido correctamente y su redefinición aumenta el ámbito del proyecto.		
Acción Preventiva	Realizar un análisis de requisitos en profundidad asegurándose que todas las partes tienen el mismo entendimiento acerca de cada requisito		
Acción Correctiva	Redimensionar el ámbito del proyecto a un alcance realista dentro del plazo disponible.		

Cuadro 4.2: R00

Código	R01		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	Historias de usuario mal definidas.		
Acción Preventiva	Trabajar en comunicación constante con el cliente a la hora de definirlas y buscar la mayor transparencia posible.		
Acción Correctiva	Redifinir las historias de usuario conflictivas, alterar la planificación cuando fuese necesario y aumentar la carga de trabajo para evitar la reducción del alcance.		

Cuadro 4.3: R01

Código	R02		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	La curva de aprendizaje para el framework de desarrollo es más larga de lo esperado		
Acción Preventiva	Realizar una planificación pesimista para las primeras tareas en que se emplee.		
Acción Correctiva	Aumentar la carga de trabajo para mantener la planificación.		

Cuadro 4.4: R02

Código	R03		
Probabilidad	Baja	Impacto	Alto
Exposición	Media		
Descripción	La curva de aprendizaje para el lenguaje PHP es más larga de lo esperado		
Acción Preventiva	Realizar una planificación pesimista para las primeras tareas en que se emplee.		
Acción Correctiva	Aumentar la carga de trabajo para mantener la planificación.		

Cuadro 4.5: R03

Código	R04		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	La elección del framework de PHP resulta errónea.		
Acción Preventiva	Realizar un estudio comparativo previo.		
Acción Correctiva	Estudiar el cambio de framework o, si es posible, continuar el proyecto sin su empleo.		

Cuadro 4.6: R04

Código	R05		
Probabilidad	Baja	Impacto	Alto
Exposición	Media		
Descripción	La planificación no incluye tareas necesarias		
Acción Preventiva	Estudiar la metodología del proyecto analizando qué procesos se van a emplear, de qué forma y en qué momentos.		
Acción Correctiva	Rehacer la planificación incluyendo estas tareas y, si fuera necesario, aumentar la carga de trabajo para corregir la ausencia de las mismas en lo que va de proyecto.		

Cuadro 4.7: R05

Código	R06		
Probabilidad	Baja	Impacto	Alto
Exposición	Media		
Descripción	No se puede construir un producto de tal envergadura en el tiempo asignado.		
Acción Preventiva	No procede.		
Acción Correctiva	Reducir el alcance del proyecto.		

Cuadro 4.8: R06

Código	R07		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	Un retraso en una tarea produce retrasos en cascada en las tareas dependientes.		
Acción Preventiva	Evitar un camino crítico o dejar espacio a alternativas en la planificación.		
Acción Correctiva	Implicar más a los recursos (en cuanto a horas dedicadas) para no atrasar el proyecto. Si no fuese posible, prolongar este o disminuir el alcance.		

Cuadro 4.9: R07

Código	R08		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	Las áreas desconocidas del producto llevan más tiempo del esperado en el diseño y en la implementación		
Acción Preventiva	Incluir en la planificación de las tareas realicionadas con dichas áreas un período de familiarización.		
Acción Correctiva	Aumentar la carga de trabajo.		

Cuadro 4.10: R08

Código	R09		
Probabilidad	Baja	Impacto	Medio
Exposición	Baja		
Descripción	El proyecto languidece demasiado en etapas iniciales.		
Acción Preventiva	Establecer plazos límite para estas etapas.		
Acción Correctiva	Aumentar el plazo del proyecto o disminuir su alcance.		

Cuadro 4.11: R09

Código	R10		
Probabilidad	Baja	Impacto	Medio
Exposición	Baja		
Descripción	Las herramientas de desarrollo no funcionan como se esperaba; el desarrollador necesita tiempo para resolverlo o adaptarse a las nuevas herramientas.		
Acción Preventiva	Buscar herramientas que se adapten a las características del proyecto que sean familiares al desarrollador.		
Acción Correctiva	Llevar a cabo la formación en el uso de las herramientas necesaria para el proyecto.		

Cuadro 4.12: R10

Código	R11		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	El cliente insiste en nuevos requisitos		
Acción Preventiva	Realizar un análisis en profundidad tratando de definir todo aquel requisito existente o potencial, para que el conjunto inicial de requisitos sea lo más completo posible.		
Acción Correctiva	Estudiar su repercusión en la planificación y el alcance del proyecto, planteando los resultados al cliente para que éste valore su inclusión en el proyecto.		

Cuadro 4.13: R11

Código	R12		
Probabilidad	Baja	Impacto	Alto
Exposición	Media		
Descripción	Los ciclos de revisión/decisión del cliente para los planes, prototipos y especificaciones son más lentos de lo esperado.		
Acción Preventiva	Establecer plazos para dichos procesos.		
Acción Correctiva	De ser posible, avanzar hacia tareas independientes a estos procesos. En caso contrario aumentar la carga de trabajo o negociar una ampliación temporal o reducción del alcance.		

Cuadro 4.14: R12

Código	R13		
Probabilidad	Baja	Impacto	Medio
Exposición	Baja		
Descripción	El tiempo de comunicación del cliente (por ejemplo, tiempo para responder a las preguntas para aclarar los requisitos) es más lento del esperado.		
Acción Preventiva	Establecer canales de comunicación estables entre las partes.		
Acción Correctiva	Replanificar la distribución de tareas para desarrollar aquellas a las que la consulta no afecte mientras no se obtenga respuesta.		

Cuadro 4.15: R13

Código	R14		
Probabilidad	Baja	Impacto	Alto
Exposición	Media		
Descripción	El desarrollo de funciones software erróneas requiere volver a diseñarlas y a implementarlas.		
Acción Preventiva	Asegurar que todo miembro del equipo tiene la misma interpretación de cada requisito e item del backlog para reducir al mínimo los errores de diseño.		
Acción Correctiva	Analizar la fuente de error y rediseñar la función software, aumentando la carga de trabajo cuando fuese necesario para alterar al mínimo la planificación temporal.		

Cuadro 4.16: R14

Código	R15		
Probabilidad	Media	Impacto	Medio
Exposición	Media		
Descripción	El desarrollo de una interfaz de usuario inadecuada requiere volver a diseñarla y a implementarla		
Acción Preventiva	Realizar el diseño de la interfaz de usuario tratando de cumplir los principios de usabilidad[12] y emplear bocetos. Realizar revisiones con cada Sprint.		
Acción Correctiva	Analizar los puntos conflictivos con ayuda de usuarios para mejorar su diseño, teniendo en cuenta los principios de usabilidad.		

Cuadro 4.17: R15

Código	R16		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	La falta de la especialización necesaria aumenta los defectos y la necesidad de repetir el trabajo		
Acción Preventiva	Incluir en las tareas de implementación con esta necesidad un período para la consulta de documentación y códigos de ejemplo.		
Acción Correctiva	Buscar asistencia de personal formado para tratar de identificar y eliminar las causas de los defectos.		

Cuadro 4.18: R16

Código	R17		
Probabilidad	Alta	Impacto	Alto
Exposición	Alta		
Descripción	El personal necesita un tiempo extra para aprender un lenguaje de programación nuevo.		
Acción Preventiva	Incluir en las tareas relacionadas con el lenguaje un período inicial de formación o adaptación.		
Acción Correctiva	Tratar de reducir la duración de las tareas restantes en que se emplee el lenguaje una vez pasado el período formativo.		

Cuadro 4.19: R17

Código	R18		
Probabilidad	Alta	Impacto	Bajo
Exposición	Media		
Descripción	Las personas clave sólo están disponibles una parte del tiempo.		
Acción Preventiva	Adecuar la planificación a los períodos de disponibilidad		
Acción Correctiva	Avanzar en tareas que no requieran la presencia de estas personas		

Cuadro 4.20: R18

Código	R19		
Probabilidad	Baja	Impacto	Alto
Exposición	Media		
Descripción	Un diseño demasiado complejo exige tener en cuenta complicaciones innecesarias e improductivas en la implementación.		
Acción Preventiva	Estudiar distintas alternativas para los diseños tratando de encontrar la más eficiente y eficaz.		
Acción Correctiva	Rediseñar las partes afectadas restringiéndose a lo estrictamente necesario para los objetivos del proyecto.		

Cuadro 4.21: R19

Código	R20		
Probabilidad	Media	Impacto	Alto
Exposición	Alta		
Descripción	La utilización de metodologías desconocidas deriva en un periodo extra de formación y tener que volver atrás para corregir los errores iniciales cometidos en la metodología		
Acción Preventiva	Hacer un estudio previo al inicio del proyecto para definir claramente los métodos y procesos a seguir.		
Acción Correctiva	Estudiar y paliar las consecuencias, aumentando la carga de trabajo cuando sea necesario.		

Cuadro 4.22: R20

4.3. Análisis de Costes

La gestión de costes del proyecto incluye los procesos de planificación, estimación, elaboración de presupuestos, financiación, recolección de fondos, administración y control de costes, de forma que el proyecto pueda ser completado dentro del presupuesto aprobado.[9]

El plan de gestión de costes es el proceso que establece las políticas, procedimientos y documentación para la planificación, administración, el gasto y el control de costes del proyecto. Su principal beneficio es que proporciona una guía sobre cómo los costes del proyecto serán controlados a lo largo del mismo.

Cabe destacar que, al tratarse de un Trabajo de Fin de Grado, los costes expuestos a continuación son teóricos. Para llevar a cabo la estimación de costes, se emplearán las siguientes técnicas:

1. *Bottom-up Estimating o Estimación ascendente*. El coste de paquetes individuales de trabajo o actividades es estimado al mayor nivel de detalle posible, para luego sumarse o agruparse en niveles mayores.

2. Software de gestión de proyectos, tales como aplicaciones, hojas de cálculo, simulaciones y herramientas estadísticas se usan para asistir en la estimación de costes. Estas herramientas pueden simplificar el uso de algunas técnicas de estimación de costes. Se emplearán las herramientas proporcionadas en la asignatura de *Gestión de proyectos informáticos*.

4.3.1. Relativos al personal

Se expondrán en esta sección los costes relativos a los miembros del equipo del proyecto: los tres tutores y el desarrollador.

1. Cotutores del proyecto, parte del personal del CiTIUS. Actuaron como *Scrum Masters*, asumiendo además las tareas de revisión de documentación, solución de dudas, etc.

2. Tutor del proyecto, parte del personal de la USC. Actuó como *Scrum Masters*, asumiendo además las tareas de revisión de documentación, solución de dudas, etc.

3. Desarrollador. Encargado de elaborar el proyecto: realizar el análisis, diseño, implementación y pruebas del mismo. Será también el encargado de elaborar toda la documentación.

Como referencia para calcular el coste de cada uno de los miembros tomaremos los salarios brutos de la guía elaborada por *Vitae Consultores* para el período

2014-2015[13]. En ella se estipula que el sueldo medio de un programador junior en PHP es de 17.000€, mientras que el de un director de proyecto sénior ronda los 35.000€. Con esta información, se calculan los siguientes costes:

1. Tutores de proyecto: le dedican 9 horas, a 18,46€ la hora, suman un total de 166,14€ por tutor.

2. Desarrollador: le dedica 412 horas, a 8,97€ la hora, sumando un total de 3.695,64€.

Por lo tanto, los gastos relativos al personal ascienden a un total de 3.861,78€.

4.3.2. Relativos al material

En esta sección se agrupan los gastos provenientes del software empleado y otros relacionados con la impresión de la presente documentación, que se estiman en 40€. Tenemos así los siguientes gastos:

PHP 5.5.9	0€
MySQL 5.5.44	0€
MySQL Workbench	0€
Zabbix 2.0.6	0€
Servidor Apache	0€
Brackets 1.3	0€
GitLab	0€
ShareLaTex Community Edition	0€
Memoria del proyecto	40€

Cuadro 4.23: Costes de material

Obtenemos un coste total de 40€ en relación a materiales.

El coste total del proyecto, resultante de la suma de las partes, asciende a 3.901,78€.

4.4. Control de versiones

Para realizar el control de versiones y el almacenamiento del código desarrollado se empleó un gestor de repositorios Git, denominado GitLab, del cual el CiTIUS[1] aloja una versión Open Source. Los tutores de este proyecto habilitaron una cuenta para que pudiese hacer uso de esta herramienta. Para el proyecto se

[1] *Centro Singular de Investigación en Tecnologías de la Información*, lugar de trabajo de los tutores de este proyecto.

creó un repositorio vacío, al que se fue añadiendo el trabajo realizado progresivamente a través de los comandos de consola propios de Git, como se puede ver en la figura 4.9.

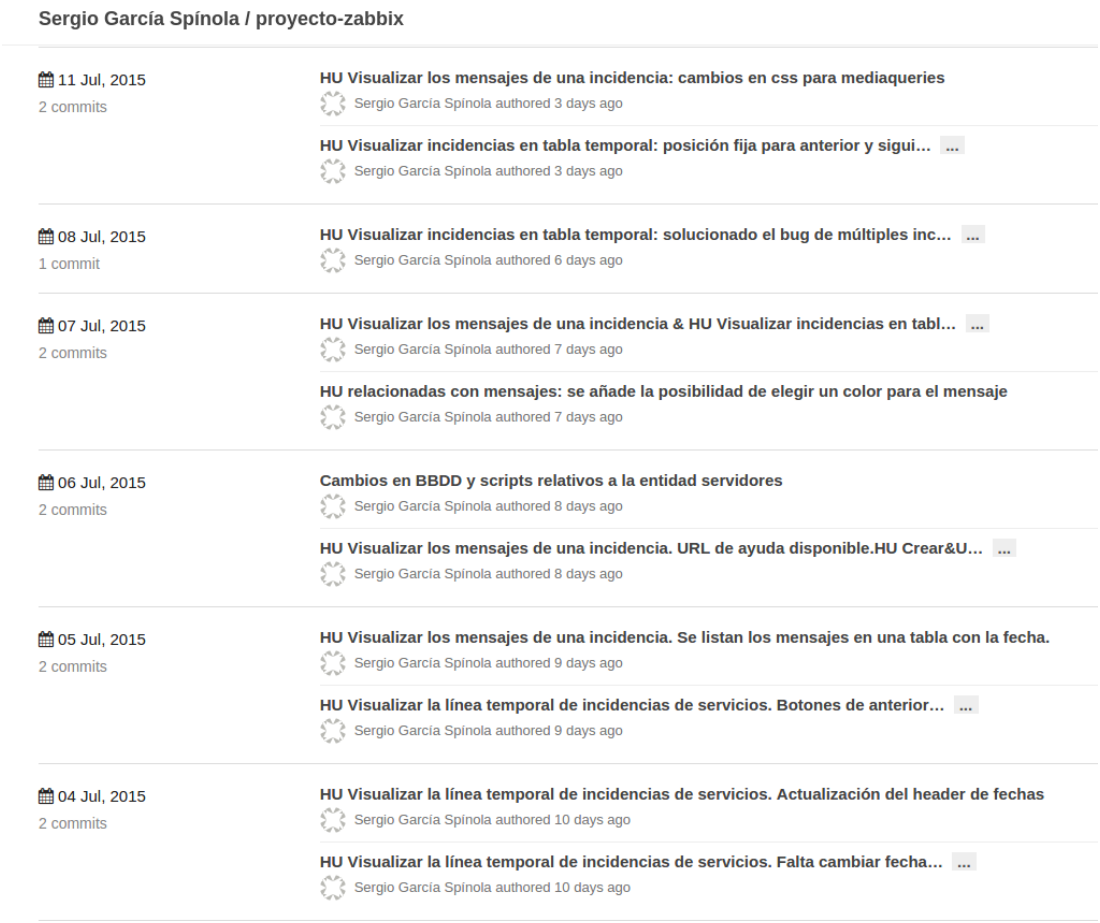


Figura 4.9: Repositorio en GitLab.

Capítulo 5

Tecnologías y herramientas

En este capítulo discutiremos el proceso de elección de las tecnologías y herramientas empleadas para el desarrollo, distinguiendo entre las elegidas para el servidor y las elegidas para el cliente.

5.1. Servidor

Una de las decisiones a tomar para el desarrollo del proyecto era elegir la tecnología que se emplearía para la implementación del servidor. La primera valoración que debía hacerse era discernir qué posibilidades nos ofrecía la API de Zabbix, ya que la interoperabilidad con Zabbix forma parte de uno de los objetivos principales del proyecto, e identificar qué lenguajes nos permitían establecer la comunicación necesaria. Ésta emplea el protocolo *"JSON-RPC 2.0"*, que define el formato de las comunicaciones utilizando JSON. La popularidad de este lenguaje de intercambio de datos hace que no sea en caso alguno una limitación a la hora de elegir el lenguaje de programación para el servidor.

Zabbix cuenta con un repositorio de productos software elaborados por terceros y que son accesibles gratuitamente a través de su web. Uno de estos productos es *PhpZabbixApi*, una librería PHP que envuelve todos los métodos de la API de Zabbix en una clase PHP siguiendo el estilo de la programación orientada a objetos, formando peticiones POST que siguen el formato JSON de la API de Zabbix.

La existencia de esta vía para facilitar el acceso a la API de Zabbix y las similitudes de PHP con el lenguaje de programación orientada a objetos Java, que hace que la curva de aprendizaje sea asumible teniendo en cuenta el tiempo disponible para el desarrollo, son motivos para la elección de este lenguaje. Se trata además de un lenguaje en el que no se ha recibido formación, de extenso uso en desarrollos web[19] y soportado por la mayoría de servicios de alojamiento, por lo que resulta de interés para el desarrollador.

Habiendo decidido emplear un lenguaje de programación desconocido para el servidor, era necesario encontrar un *framework* que facilitase ciertos aspectos del desarrollo del proyecto, en especial el uso del patrón MVC[3] para ayudar a mantener una estructura coherente en el proyecto. Existe una gran variedad de alternativas en cuanto a frameworks de desarrollo web para PHP: Laravel, CodeIgniter, CakePHP, Symfony, Zend Framework, Phalcon, Yii, Aura, Kohana y FuelPHP entre otros. Muchos de estos frameworks se caracterizan por su gran número de funcionalidades, módulos y complementos. En el caso de este proyecto, las exigencias sobre el framework eran mínimas, por lo que los factores para su elección se basaban en la ligereza del framework, el nivel de documentación existente y la curva de aprendizaje necesaria para su uso.

La información y artículos que podemos encontrar realizando comparativas provienen de la experiencia y las pruebas realizadas por programadores individuales publicados en sus respectivos blogs. *Yii* es mencionado en todos ellos como un framework en el que destaca su velocidad de respuesta, ligereza y sencillez de uso, y características como el uso del patrón *Active Record*[4] o facilidades para el uso de AJAX. El hecho de que se encuentre en cualquier lista de frameworks más populares en posiciones altas nos asegura encontrar documentación y una amplia comunidad de programadores publicando sus problemas y soluciones en sus desarrollos con *Yii*. Por lo tanto, se escoge para el desarrollo de este proyecto.

Yii ofrece la posibilidad de utilizar como gestor de base de datos MySQL, PostgreSQL u Oracle. En el espíritu de basar el proyecto en tecnologías de software libre, Oracle queda descartada. En este desarrollo buscamos un sistema gestor de bases de datos que sea de fácil utilización y rápido en las operaciones de lectura, ya que no habrá consultas complejas ni un gran número de inserciones o modificaciones. Tanto PostgreSQL como MySQL son sistemas que cumplen estos requisitos. MySQL es la opción más común para desarrollos web, ya que es muy fácil de usar, la existencia de herramientas visuales como *MySQL Workbench* facilitan el diseño de las bases de datos y se trata de una herramienta familiar para el desarrollador. Por otro lado, PostgreSQL es un sistema mucho más complejo, cuyo uso resulta más apropiado en grandes entornos donde la integridad de los datos y la fiabilidad son indiscutibles, con diseños complejos y desarrollo de procedimientos propios. Un factor decisivo podría ser la licencia de cada sistema, pero en esta ocasión ninguna representa una restricción: con MySQL no es necesaria una licencia comercial a menos que se utilice con tal fin, y PostgreSQL tiene licencia MIT[20], que ofrece total libertad. Valorando las características de ambas, nos decantamos por MySQL.

5.2. Cliente

Una vez conocidas las tecnologías a emplear en el lado del servidor, debemos elegir aquellas necesarias para construir una web dinámica en el lado del cliente. Emplearemos JavaScript, lenguaje de programación interpretado, implementación del estándar ECMAScript, que nos permite la manipulación del DOM (una interfaz de programación de aplicaciones para acceder, añadir y cambiar dinámicamente contenido estructurado en documentos). Para facilitar el uso de JavaScript, emplearemos una de sus bibliotecas más populares, jQuery, que llega a usarse en el 60 % del millón de sitios con más tráfico de internet[1]: "biblioteca de JavaScript, [...], que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web jQuery y AJAX. [...] ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio"[5].

Con objeto de conseguir una aplicación de la que el usuario pueda hacer uso sin que el dispositivo desde el que acceda sea un impedimento para la presentación, se empleará *Bootstrap*. Se trata de un framework de HTML, CSS y JavaScript que permite la adaptación de la página al tamaño de pantalla de los dispositivos. Esto se consigue a través del diseño "sensible" o *responsive*, que busca la adaptación de las páginas al dispositivo en que se está viendo. *Bootstrap* emplea un sistema de rejilla de 12 columnas que se redimensiona según el tamaño de pantalla del dispositivo en que se visualice la página. Ofrece además una serie de plantillas, componentes y estilos de HTML y CSS, y Plug-ins de JavaScript que simplifican el trabajo a realizar.[15]

5.3. Herramientas

Se enumeran a continuación una serie de herramientas que se han empleado tanto para el desarrollo del proyecto como para su gestión:

- ShareLaTeX: editor de textos de latex en línea que se ha utilizado para la elaboración del presente documento.
- Google Draw IO: herramienta en línea de dibujo de diagramas que sirvió de instrumento para la elaboración de bocetos de la interfaz gráfica y el diagrama de arquitectura del proyecto.
- Trello: herramienta en línea que se ha empleado para la gestión del Product Backlog y los Sprint Backlogs a través de la creación de listas en una

[1]Estadísticas de uso de JQuery en *http://trends.builtwith.com/javascript/jQuery*

pizarra simulada. Permite añadir comentarios a tareas, asignarles etiquetas y desglosarlas en subtareas, de manera que se puede establecer el nivel de completitud de cada tarea.

- MySQL Workbench: herramienta visual que, entre muchas otras funcionalidades, permite el diseño de bases de datos y generación de SQL.
- Brackets IO: editor de textos que se ha empleado para el desarrollo del código fuente del proyecto. Aporta complementos específicos para los lenguajes de programación seleccionados que facilitan las tareas del desarrollador.

Capítulo 6

Desarrollo

A partir del Product Backlog definido en el capítulo 3, y habiendo realizado la selección de tecnologías pasamos a describir el desarrollo de la aplicación en base a los Sprints definidos en la planificación temporal. Se decidió establecer quince días como duración del Sprint inicial. Este primer Sprint serviría para poder estimar con mayor conocimiento la velocidad de trabajo que se podría definir para el resto del desarrollo. Como se menciona en el capítulo anterior, para gestionar el proceso de Scrum y poder conocer en todo momento el estado de las historias de usuario se ha empleado la herramienta *Trello*. Con cada Sprint Review, se validarán las historias de usuario que se hayan completado y se establecerá el Sprint Backlog de la siguiente iteración.

6.1. Sprints del proyecto

6.1.1. Sprint 1

Para este primer incremento se establecieron las historias de usuario de la figura 6.1.

Durante el Sprint Review correspondiente se aprobaron las historias de usuario HU02, HU05, HU06, HU07, HU08, HU09, HU10, HU11. Sin embargo, para la HU12, se encontraron errores, por lo que se añadirá al Backlog del próximo Sprint.

6.1.2. Sprint 2

Con la revisión del primer Sprint se decidió que se mantendría la duración de quince días para cada iteración, pero también se aumentaría la carga de trabajo. Se definió el Sprint Backlog reflejado en la figura 6.2.

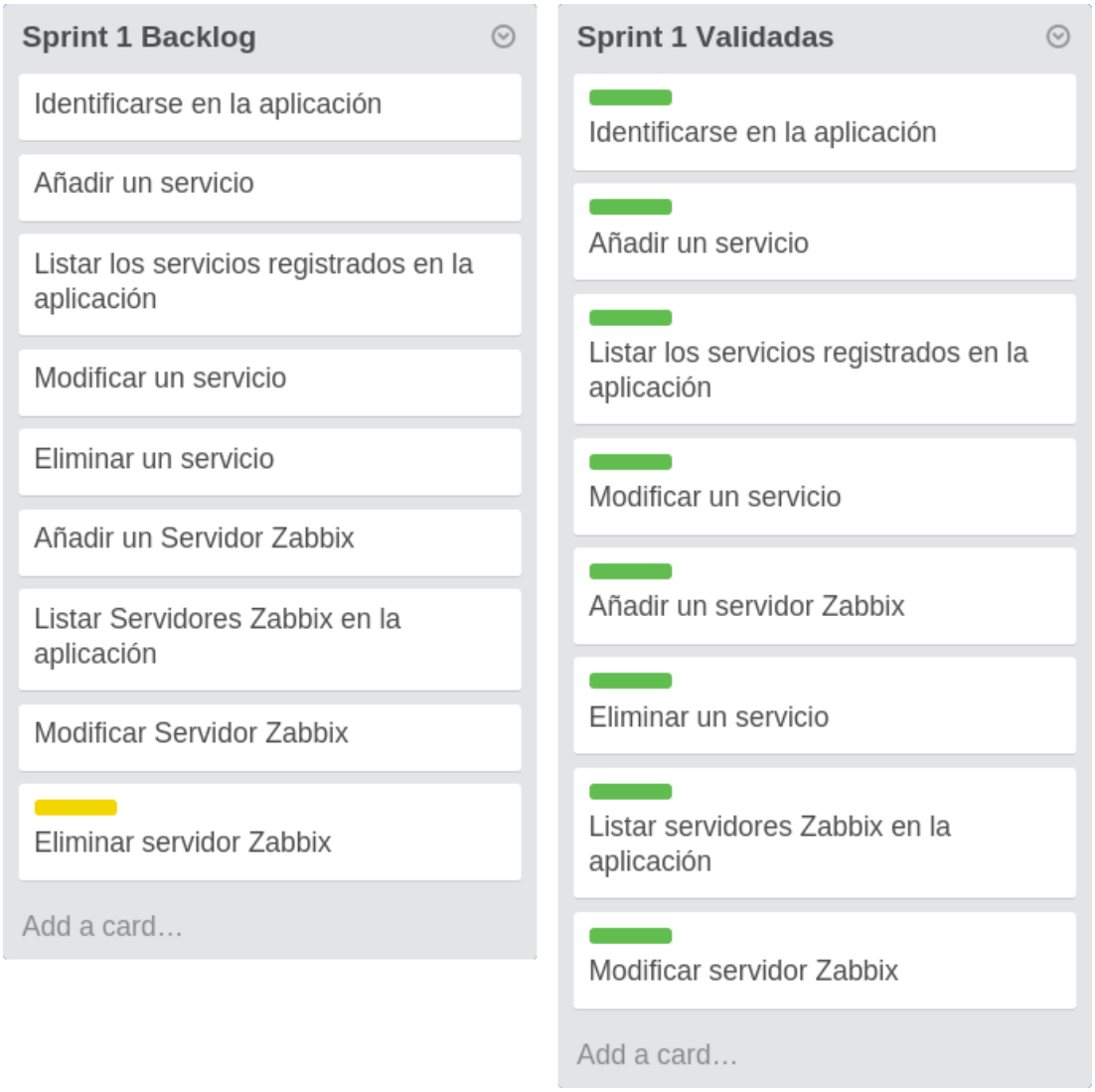


Figura 6.1: Trello: Sprint 1.

En esta ocasión se validaron las historias de usuario HU12, HU13, HU16, HU18, HU20, HU22, HU28, HU23. Las historias de usuario HU14, HU15, HU17, HU19 y HU26 no alcanzaron el la definición de concluidas, por lo que se incorporan al Sprint Backlog de la siguiente iteración. A excepción de la HU26, que no pasó las pruebas, el resto de historias de usuario no fueron validadas por razones de diseño. Por otro lado, el Sprint Review también sirvió para localizar una historia de usuario que no se había incluido en un principio, que se incorpora tanto al Product Backlog como al Sprint Backlog de esta iteración (para reflejar su aparición) y la próxima. Se define en el cuadro 6.1.

Establecer mensaje por defecto para el cierre de incidencias
- Dado un administrador - Cuando esté registrando un nuevo disparador en la aplicación - Deberá poder establecer un mensaje por defecto para mostrarse en el cierre de incidencias.
- Si se establece el cierre automático de incidencias para el disparador, será obligatorio establecer el mensaje.

Cuadro 6.1: HU30

6.1.3. Sprint 3

La figura 6.3 identifica las historias de usuario seleccionadas para esta iteración.

Durante el desarrollo de este Sprint de identificaron nuevas historias de usuario - figuras 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9 y 6.10 -, que se añadieron al Sprint y Product Backlogs. Además, se encontró la necesidad de redefinir la historia de usuario HU21 “Publicar una incidencia manual”, ya que el conocimiento ganado durante los Sprints previos probó su incorrección.

En el Sprint Review quedaron validadas las historias de usuario HU14, HU15, HU17, HU30, HU19, HU24, HU29, HU27, HU31, HU21, HU32, HU33 y HU04. Al mismo tiempo, se encontraron errores y añadieron al Sprint Backlog de la siguiente iteración las historias de usuario HU25, HU26, HU32, HU33, HU34, HU37 y HU38.

Figura 6.2: Trello: Sprint 2.

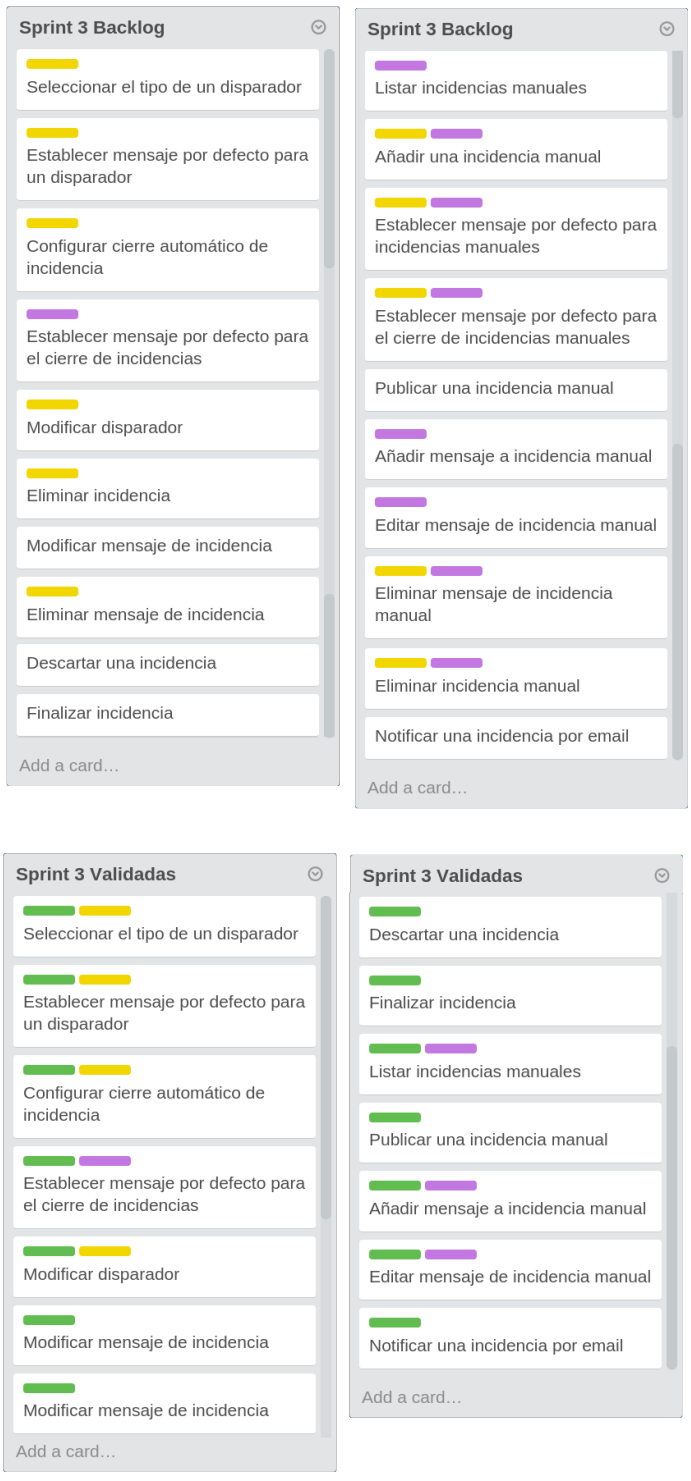


Figura 6.3: Trello: Sprint 3.

Publicar una incidencia manual
- Dado un administrador - Cuando seleccione una incidencia manual con fecha de inicio previa a la actual y esta no haya sido publicada todavía - Deberá poder autorizar su publicación para que aparezca en la tabla temporal.
- Se dispondrá un icono junto con cada incidencia de la lista que permita al administrador autorizarla publicación

Cuadro 6.2: HU21(modificada)

Listar incidencias manuales
- Como administrador - Quiero poder listar las incidencias manuales - Para poder visualizar todas las registradas en la aplicación
- Se pondrán de primeras las incidencias que requieran aprobación del administrador para su publicación.

Cuadro 6.3: HU31

Añadir una incidencia manual
- Dado un administrador - Quiero poder añadir una incidencia manual - Para poder advertir de eventos conocidos de antemano
- Se completará un formulario en el que se indiquen los disparadores afectados por la incidencia y las fechas de inicio y fin. Durante la vigencia de la incidencia, los disparadores afectados quedarán bloqueados, por lo que no crearán incidencias.

Cuadro 6.4: HU32

Establecer mensaje por defecto para incidencias manuales
- Dado un administrador - Cuando esté registrando una nueva incidencia manual en la aplicación - Deberá poder establecer un mensaje por defecto para mostrarse en la publicación de la incidencia.

Cuadro 6.5: HU33

Establecer mensaje por defecto para el cierre de incidencias manuales
- Dado un administrador - Cuando esté registrando una nueva incidencia manual en la aplicación - Deberá poder establecer un mensaje por defecto para mostrarse en el cierre de la incidencia.

Cuadro 6.6: HU34

Añadir mensaje a incidencia manual
- Dado un administrador - Cuando seleccione una incidencia manual de la lista - Deberá poder añadir un mensaje a la lista de mensajes publicados de la incidencia
- Se presentará un formulario para la introducción del contenido del mensaje y la elección del color asociado para mostrarse en la parte pública.

Cuadro 6.7: HU35

Editar mensaje de incidencia manual
- Dado un administrador - Cuando seleccione una incidencia de la lista - Deberá poder editar los mensajes publicados de la incidencia.
- Se presentará un formulario para la modificación del contenido del mensaje y del color asociado para mostrarse en la parte pública.

Cuadro 6.8: HU36

Eliminar mensaje de incidencia manual
- Dado un administrador - Cuando seleccione una incidencia de la lista - Deberá poder eliminar los mensajes publicados de la incidencia.
- Se presentará un botón junto con cada mensaje que permita su eliminación. No se podrá eliminar el primer mensaje publicado de cada incidencia.

Cuadro 6.9: HU37

Eliminar incidencia manual
- Dado un administrador - Cuando seleccione una incidencia de la lista - Deberá poder eliminar la incidencia.
- Se presentará un botón junto con incidencia que permita su eliminación.

Cuadro 6.10: HU38

6.1.4. Sprint 4

Durante el Sprint Review del anterior Sprint se evidenció la necesidad de añadir nuevas historias de usuario referentes a las incidencias manuales, que se exponen en las figuras 6.11, 6.12, 6.13 y 6.14. En la figura 6.4 podemos observar el Sprint Backlog al completo.

Configurar apertura automática de incidencias manuales
- Dado un administrador - Cuando esté registrando una nueva incidencia en la aplicación - Deberá poder establecer si la misma se publicará automáticamente.
- En caso afirmativo, será imperativo establecer un mensaje por defecto para la apertura

Cuadro 6.11: HU39

Configurar cierre automático de incidencias manuales
- Dado un administrador - Cuando esté registrando una nueva incidencia en la aplicación - Deberá poder establecer si la misma se cerrará automáticamente.
- En caso afirmativo, será imperativo establecer un mensaje por defecto para el cierre

Cuadro 6.12: HU40

Editar incidencia manual
- Dado un administrador - Cuando seleccione una incidencia manual de la lista que no haya sido publicada - Deberá poder modificar sus datos
- Se dispondrá de un botón de editar junto a cada incidencia para poder acceder al formulario de modificación.

Cuadro 6.13: HU41

Finalizar incidencia manual
- Dado un administrador - Cuando seleccione una incidencia manual de la lista - Deberá poder finalizarla.
- Se presentará un formulario en el que introducir un mensaje de cierre o modificar el establecido por defecto.

Cuadro 6.14: HU42

En el Sprint Review de esta iteración quedaron aprobadas las historias de usuario HU25, HU32, HU33, HU37, HU36, HU39, HU40, HU41, HU42, HU01 y HU03. Las historias de usuario HU00, HU27 y HU38 contenían errores y no alcanzaron el nivel de completitud necesario para su aprobación.

6.2. Sprint final

Este Sprint, que en un principio únicamente contemplaba el despliegue de la aplicación y las pruebas correspondientes, tuvo que ser ampliado para llevar a cabo las correcciones necesarias para validar las historias de usuario no completadas en el Sprint anterior. Además, las incidencias programadas sufrieron un cambio en su lógica, de forma que su creación llevaba implícita su publicación, y la selección de disparadores a los que afectan pasa a ser opcional, teniendo en su lugar que asociar la incidencia a al menos un servicio.

6.3. Burn Down Chart

El gráfico denominado "Burn Down Chart" se emplea para reflejar el trabajo restante en un marco temporal. En la figura 6.5 se muestra información sobre las historias de usuario que se fueron realizando en cada Sprint. En primer lugar, en forma de columnas, se muestra con cuántas historias de usuario contó cada Sprint Backlog en su inicio y cuántas se validaron en el Sprint Review. Por otro lado, en forma de líneas, tenemos la relación entre el número de historias de usuario planeadas para los Sprint Backlogs, el número que finalmente tuvieron y la cantidad de historias de usuario que se validaron.

Como se puede apreciar, tanto el tercer como el cuarto Sprint tuvieron un incremento considerable de trabajo con respecto a los planes iniciales. Tal incremento es debido a las correcciones necesarias para validar las historias de usuario rechazadas en el Sprint Review anterior y la aparición de nuevas historias de usuario a lo largo del desarrollo.

Figura 6.4: Trello: Sprint 4.

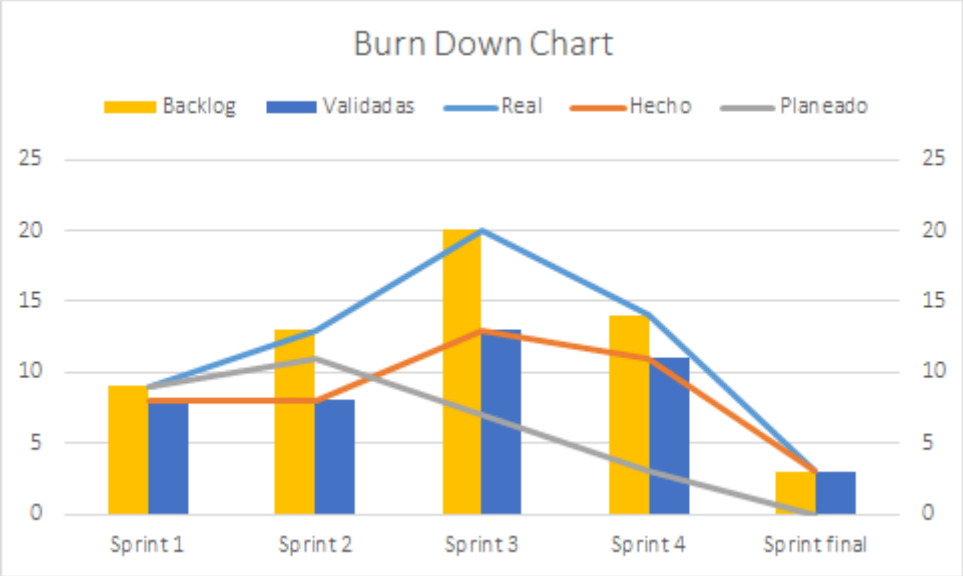


Figura 6.5: Burn Down Chart

Capítulo 7

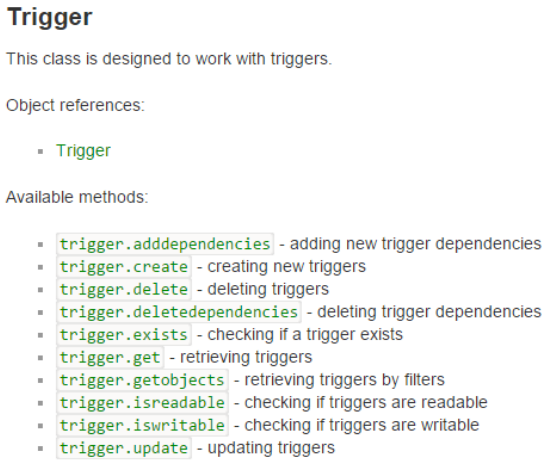
Diseño e implementación

En este capítulo expondremos los detalles sobre el desarrollo en cuanto a diseño e implementación. Analizaremos la API de Zabbix y sus posibilidades, el diseño del modelo de datos, la arquitectura de la aplicación y entraremos en detalle acerca de las distintas secciones que componen la aplicación, explicando la lógica interna de las acciones clave de cada una.

7.1. Análisis de la API de Zabbix

El potencial de esta API era un factor clave a la hora de definir el alcance de este proyecto. Sus funcionalidades eran desconocidas en un principio, por lo que fue necesario hacer un estudio sobre las posibilidades que ésta ofrecía. La web de Zabbix alberga una documentación[6] muy completa sobre su API. Se trata de una API no sólo de lectura, sino que también permite introducir y manipular la información presente en el servidor de Zabbix. Se divide en paquetes de funciones para cada elemento específico del sistema Zabbix, incluyendo lo que más nos interesa en este proyecto: los *triggers*. En la figura 7.1 podemos ver la serie de métodos disponibles para tratar con los triggers.

Nuestro interés se centra en la obtención de información sobre el estado de los *triggers*, por lo que se empleará la función *trigger.get*[7]. De la multitud de parámetros disponibles, se utilizarán *monitored* y *active*, con los que se establece que tan sólo se devuelvan los *triggers* activados que pertenecen a *hosts* monitorizados. Una primera prueba demostró que, en lugar de los citados parámetros, es necesario indicar los identificadores de los hosts monitorizados, puesto que de otra forma también se obtenían *triggers* pertenencientes a plantillas asociadas con los *hosts*, con independencia de estar éstos activados o no. Para obtener tan sólo aquellos que se encuentran activados, se usará el parámetro *filter*, que permite indicar que se filtre la respuesta a aquellos *triggers*[8] con *status = 0*, es decir, activados. De forma similar a los *triggers*, los *hosts* también cuentan con un método *get*, que se empleará para obtener los identificadores necesarios en la consulta de



Trigger

This class is designed to work with triggers.

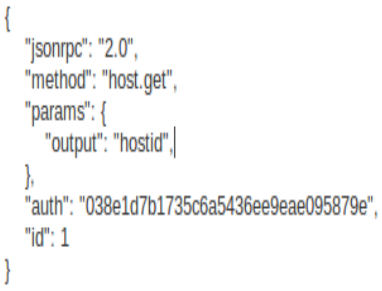
Object references:

- Trigger

Available methods:

- `trigger.adddependencies` - adding new trigger dependencies
- `trigger.create` - creating new triggers
- `trigger.delete` - deleting triggers
- `trigger.deletedependencies` - deleting trigger dependencies
- `trigger.exists` - checking if a trigger exists
- `trigger.get` - retrieving triggers
- `trigger.getobjects` - retrieving triggers by filters
- `trigger.isreadable` - checking if triggers are readable
- `trigger.iswritable` - checking if triggers are writable
- `trigger.update` - updating triggers

Figura 7.1: Zabbix API: métodos de Triggers



```
{
    "jsonrpc": "2.0",
    "method": "host.get",
    "params": {
        "output": "hostid",
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

(a) hosts.get

```
{
    "jsonrpc": "2.0",
    "method": "trigger.get",
    "params": {
        "hostids": {
            14062,
            14063
        },
        "output": { "triggerid", "value" },
        "filter": { "status" : 0 }
    },
    "auth": "038e1d7b1735c6a5436ee9eae095879e",
    "id": 1
}
```

(b) triggers.get

Figura 7.2: Peticiones a la API de Zabbix

los *triggers*. En la figura 7.2 puede verse un ejemplo de cómo se formarían estas llamadas.

Como se ha comentado en apartados anteriores, para la comunicación con la API de Zabbix se empleará la librería *PhpZabbixApi*, que encapsula las llamadas a la API en métodos de la clase *ZabbixApiAbstract*. En lugar de elaborar nosotros mismos la llamada, pasamos como argumento de la función en cuestión los parámetros, tal y como se observa en la figura 7.3. La librería también aporta una clase que extiende a *ZabbixApiAbstract, ZabbixApi*. En ella podremos crear nuestros propios métodos para explotar la API.

```
$triggers = $api->triggerGet([
    'hostids' => $hosts,
    'output' => ['triggerid','description'],
    'filter' =>['status' => 0],
]);
```

Figura 7.3: Método de *ZabbixApi*

7.2. Modelo de datos

En esta sección se explicará el modelo diseñado para la base de datos de la aplicación, reflejado en la figura 7.4, comentando el diseño de cada entidad y sus relaciones con el resto.

1. Usuarios: compuesto de tan sólo dos campos, guardará el correo electrónico del usuario (empleado para identificarse) y el valor *hash* de la contraseña elegida por el usuario.

2. Servicios: guardará la información necesaria para este ente de la aplicación, es decir, el nombre establecido así como enlaces que se emplearán en la parte pública de la aplicación.

3. Servidores: en este caso, además de guardar un nombre, necesitamos conservar la url en que se aloja el servidor, así como las credenciales para identificarse en el mismo y acceder a sus datos.

4. Disparadores: son una adaptación de los *triggers* de Zabbix. Cada instancia estará relacionada con un servicio y un servidor, y guardará el número identificativo del *trigger* de Zabbix que represente, así como su nombre. A mayores, el usuario lo configurará con un nombre, y establecerá si sus incidencias se abren y/o cierran automáticamente. Con el fin de conocer su estado, se mantienen dos *flags* que nos indicarán si está "activo", es decir, si actualmente hay alguna incidencia en curso relacionada con el disparador; y un segundo valor indicará si se encuentra "bloqueado", es decir, si existe alguna incidencia programada en curso relacionada con el disparador.

A raíz de los disparadores y de los cambios de estado de los *triggers* con que estos se asocian nacen las incidencias. A excepción de la tabla de usuarios, todas las entidades explicadas hasta el momento contienen un número entero identificativo que se genera automáticamente. Sin embargo, para las incidencias, al estar representada cada tipo en una entidad propia, se empleará un *trigger* de MySQL[14] para calcular este número identificativo.

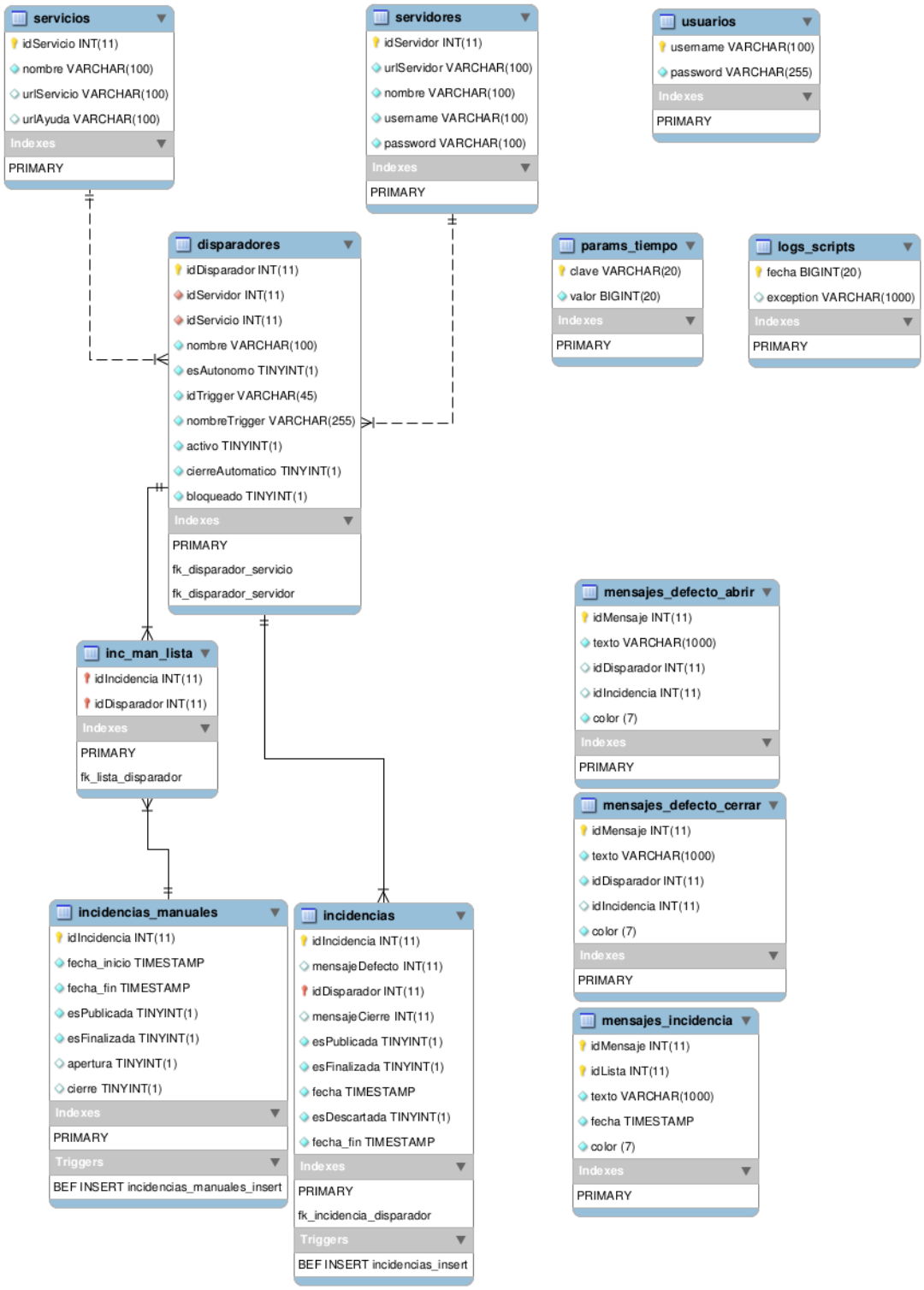


Figura 7.4: Modelo de datos

5. Incidencias: guardará las fechas de inicio y fin de cada instancia, la referencia de los mensajes por defecto para la apertura y/o cierre de incidencias si estos se han configurado para el disparador y una serie de *flags* de estado. Éstos se emplearán para conocer si se ha publicado, finalizado o descartado la incidencia. Tendrá también la referencia del disparador al que pertenece.

6. Incidencias Manuales: al que igual que las incidencias regulares, mantendrá información sobre la fecha de inicio, de fin, y los *flags* de estado(a excepción de *esDescartada*). A mayores guardará la configuración de automatización de su finalización. Para este tipo de incidencias se mantendrá una lista (la tabla "*inc_man_lista*") en la que se relacionen incidencias manuales y disparadores, y otra lista (la tabla "*inc_man_servicios*") en la que se relacionen incidencias manuales y servicios, ya que cada incidencia manual puede afectar a múltiples disparadores y al menos a un servicio.

7. Mensajes por defecto: estas dos entidades ("*mensajes_defecto_abrir*", "*mensajes_defecto_cerrar*") guardan el texto introducido por el usuario como mensaje por defecto para la apertura o el cierre de las incidencias de un disparador o de una incidencia manual. Se mantiene además el color elegido para acompañar al mensaje y la referencia del disparador o la incidencia manual a que pertenece el mensaje.

8. Mensajes de Incidencia: esta entidad representa cada uno de los mensajes publicados en la aplicación. Guardará el texto publicado, la fecha de publicación, el color asociado y la referencia de la incidencia para la que se publicó.

Cabe destacar los cambios sufridos por la entidad Incidencias Manuales con la inclusión de las historias de usuario descritas en los cuadros 6.11, 6.12, 6.13 y 6.14 en el cuarto Sprint. En un principio la entidad tan sólo contaba con los campos de *fecha_inicio*, *fecha_fin*, *esPublica* y *esFinalizada*, ya que no se contemplaban las funcionalidades recogidas en las nuevas historias de usuario. Con ellas, se hizo necesario la modificación de la entidad para incluir los *flags* de configuración *apertura* y *cierre*, que indican si se publicará y/o finalizará la incidencia automáticamente. Los cambios sufridos en la lógica de esta entidad en el Sprint final hacen que los campos *apertura* y *esPublicada* carezcan de uso, pero se mantienen en el modelo de datos por si volviesen a ser necesarios.

7.3. Diagramas de arquitectura y despliegue de la aplicación

Como se ha mencionado anteriormente, uno de los requisitos para el framework escogido era la implementación del patrón de diseño *Modelo-Vista-Controlador*. Este busca separar los datos, la interfaz de usuario y la lógica de negocio en tres componentes distintos, de forma que su desarrollo se desacople y se facilite su posterior mantenimiento.

1. Modelo: representación de la información del sistema a través de la cual se gestiona todo acceso a la misma. Proporciona a la vista la información que debe ser mostrada en cada momento. Las peticiones y operaciones sobre los datos llegan al modelo a través del controlador.

2. Controlador: es el encargado de gestionar todas las peticiones del usuario desde la vista, interactuando con el modelo cuando fuese preciso. Es el intermediario entre la vista y el modelo.

3. Vista: representa la interfaz de usuario, presenta la información en un formato adecuado para la interacción.

En la figura 7.5 podemos observar la arquitectura de la aplicación. En nuestro caso, el componente de vista está compuesto a su vez de los módulos de *admin*, *site* y *layouts*. En *Yii*, las vistas se conforman con la combinación de una plantilla o *layout*, y un contenido (las vistas recogidas en *admin* y sus submódulos y en *site*). Cada vista envía sus peticiones al controlador correspondiente, quien en comunicación con las clases del modelo realiza las actualizaciones o modificaciones oportunas en la información de la base de datos.

Los diagramas de despliegue de *UML*[18], como el de la figura 7.6, modelan la disposición física de los componentes del proyecto. Reflejan nodos, entendido como cualquier cosa que pueda albergar software (ya sean componentes hardware o entornos de ejecución); artefactos, que son las manifestaciones físicas del software: normalmente, archivos; y las vías de comunicación entre los nodos.

En la figura 7.6 se diferencian los tres nodos principales del proyecto:

- El nodo representativo del dispositivo desde el que los usuarios harán uso de la interfaz gráfica a través de un navegador web, quien se comunicará con el nodo servidor a través del protocolo HTTPS;

- El nodo del servidor de aplicaciones que alojará el proyecto. Se empleará el servidor web Apache, que dará acceso a la aplicación desarrollada con el framework Yii. Diferenciamos la arquitectura de la aplicación del artefacto denominado “launch.php”, quien será el encargado de actualizar la

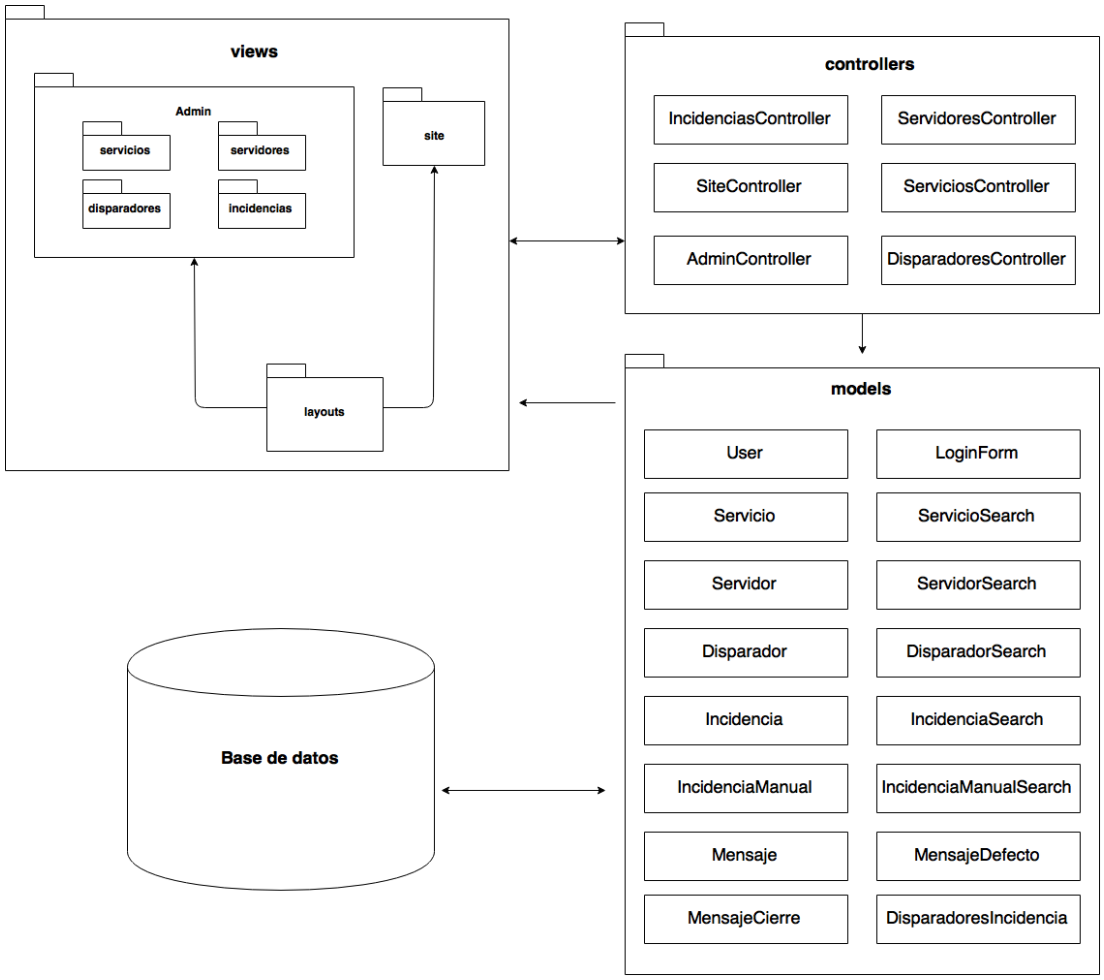


Figura 7.5: Diagrama de arquitectura

información alojada en el servidor de MySQL. El artefacto "cronscript.sh" se ejecutará a través de una tarea cron[1], asegurándose de que el script "launch.php" esté en ejecución en todo momento.

- El nodo representativo de los servidores Zabbix, con quienes se establecerán comunicaciones empleando el protocolo HTTP.

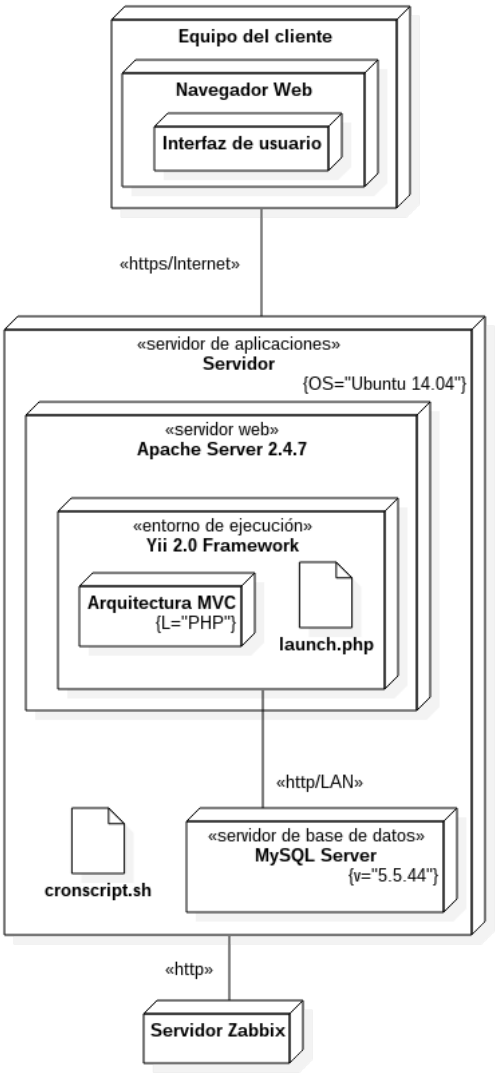


Figura 7.6: Diagrama de despliegue

[1]En los sistemas operativos Linux, las *cron tasks* son procesos que se ejecutan periódicamente según las reglas definidas en el archivo *crontab* gracias al demonio *cron*.

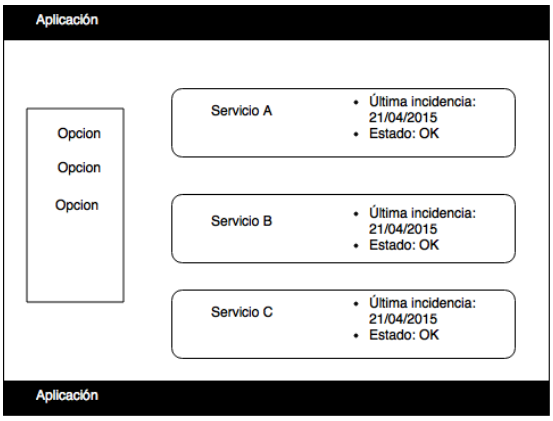


Figura 7.7: Boceto de la sección de servicios.

7.4. Secciones

Habiendo escogido las tecnologías a emplear y tomado decisiones sobre el diseño del modelo de datos y la arquitectura de la aplicación, en este apartado entraremos en detalle acerca del diseño y la implementación de las distintas secciones en que se divide la aplicación.

7.4.1. Servicios

Una de las facilidades que nos proporciona *Yii* es el uso del patrón *Active Record*. Gracias a ello no tenemos que implementar las operaciones contra la base de datos, sino que se nos provee de una serie de clases con métodos para tal fin. Así, nos encontramos con que la definición de las clases varía considerablemente con respecto al lenguaje orientado a objetos conocido por el desarrollador, ya que en lugar de establecer una correspondencia entre columnas de una tabla y atributos de una clase, se empleará una declaración de reglas para definir la clase.

En la figura 7.8 podemos ver el ejemplo de la clase "*Servicio*". En primer lugar deberemos extender la clase del framework *Active Record* y a continuación declaramos sus atributos mediante reglas. En la misma figura podemos observar la declaración de "*labels*", esto es, las etiquetas que llevarán los campos de los formularios en que empleemos esta clase. Del mismo modo que las etiquetas son empleadas en los formularios, *Yii* automatiza la validación de los mismos sometiendo los datos de entrada a las reglas definidas por el usuario para cada campo mediante el método *validate* que hereda de la clase *Active Record*.

Para cumplir la HU06 se precisa listar todo servicio creado en la aplicación. En la figura 7.7 podemos observar el diseño preliminar de la sección de servicios.

```php
public function rules()
{
    return [
        [['nombre'],'required', 'message' => 'Debe indicar un nombre para el servicio'],
        [['nombre'], 'unique'],
        [['nombre', 'urlServicio', 'urlAyuda'],'string', 'max' => 100],
        [['urlServicio','urlAyuda'], 'url', 'defaultScheme' => 'http']
    ];
}

/**
 * @inheritdoc
 */
public function attributeLabels()
{
    return [
        'nombre' => 'Nombre',
        'urlServicio' => 'Url Servicio',
        'urlAyuda' => 'Url Ayuda',
    ];
}
```

Figura 7.8: Declaración de reglas y etiquetas.

Planteamos el diseño de la interfaz de usuario con una serie de bocetos que servirán de guía para la implementación. Se valoró la opción de someter el diseño a un proceso de evaluación en el que se realizaran en primer lugar bocetos con sus respectivos *storyboards*[2], pero la restricción de tiempo de este proyecto y el hecho de que sus usuarios finales son personal de administración con conocimientos informáticos hizo que, tras el análisis y la aprobación de un serie de bocetos de distintas secciones de la aplicación, se continuara realizando el diseño a la par que la implementación, tratando de respetar los bocetos en la medida de lo posible y teniendo en cuenta los principios de usabilidad de Nielsen[12]. El diseño de la interfaz será discutido en la próxima sección de este capítulo.

En el desarrollo se ha tomado como referencia el libro "*Web Application Development with Yii 2 and PHP* "[11] y la guía de *Yii* [10]. Tras consultar estas fuentes, se modificó el diseño para incluir el *widget* [3] de *Yii* denominado *GridView*, el cual forma una tabla a partir de las instancias presentes en la base de datos de una determinada clase, ofreciendo funcionalidades de búsqueda y paginación. Podemos observar los resultados a nivel gráfico en la figura 7.10. Para el

[2]Una *storyboard* se compone de una serie de ilustraciones que pretenden servir de guión para la realización de alguna tarea.

[3] Un *widget* es una pequeña aplicación cuyo objetivo es dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

```php
public function search($params)
{
    $query = Servicio::find();

    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);

    $this->load($params);

    $query->andFilterWhere(['idServicio' => $this->idServicio]);
    $query->andFilterWhere(['like', 'nombre', $this->nombre]);
    $query->andFilterWhere(['like', 'urlAyuda', $this->urlAyuda]);
    $query->andFilterWhere(['like', 'urlServicio', $this->urlServicio]);

    return $dataProvider;
}
```

Figura 7.9: Método *search* de servicio.

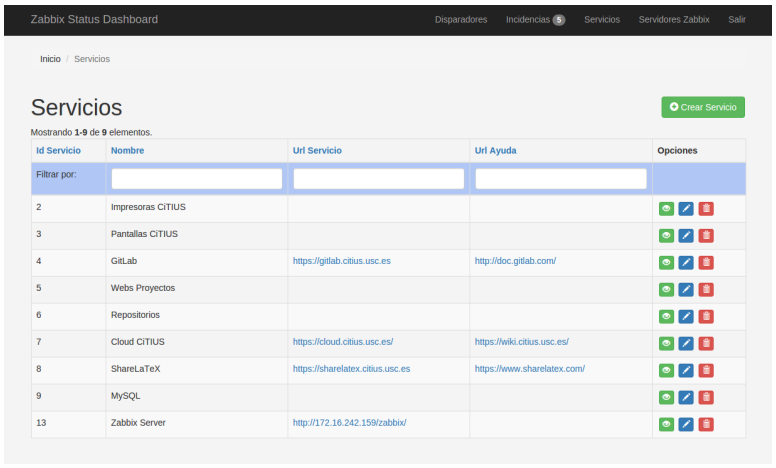


Figura 7.10: Listado de Servicios.

uso del *GridView* deberemos implementar una nueva clase que extienda a aquella sobre la que queremos crear la tabla - *Servicio* en este caso - o directamente implementar el método *search* en la clase. En este método definiremos cómo se realiza la búsqueda. Una vez más se emplean métodos propios de las clases del framework y se facilita la declaración de los parámetros de búsqueda. Podemos ver un ejemplo en la figura 7.9.

7.4.2. Servidores Zabbix

Siguiendo las lineas de diseño empleadas en la sección de servicios y tomando como plantillas sus distintas vistas, la diferencia más importante que encontramos en el desarrollo de la sección de servidores es que por primera vez entramos en contacto con la API de Zabbix. Como en el caso de servicios, deberemos implementar dos clases: *Servidor* y *ServidorSearch*. En el momento de creación del servidor debemos comprobar que el usuario y la contraseña proporcionadas son correctas, tal y como se especifica en la HU09. Para realizar esta validación efec-

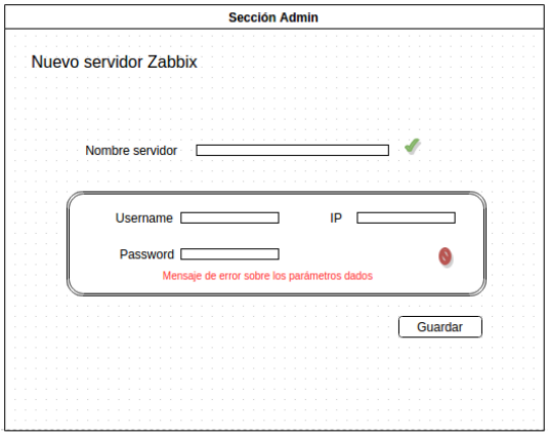


Figura 7.11: Boceto de la creación de un servidor.

tuaremos una llamada AJAX a una función del controlador de servidores, quien a su vez tratará de comunicarse con el servidor de Zabbix e identificarse con los datos del formulario.

En la figura 7.12 podemos ver un diagrama de secuencia de esta validación, donde en un primer momento se comprueba si existe algún servidor con el mismo nombre o la misma URL, para después tratar de conectarse al servidor de Zabbix con los datos proporcionados por el usuario. Si cualquiera de estos casos incurre en error, el controlador devuelve a la vista un mensaje para el usuario.

7.4.3. Disparadores

De nuevo, se sigue el modelo empleado para las secciones anteriores, implementando las clases *Disparador* y *DisparadorSearch*. En este caso, la mayor dificultad reside en la lógica interna de la creación de un nuevo servidor, que responde a la HU16, y la automatización de la consulta del estado de los *triggers* de Zabbix asociados a los disparadores para crear incidencias.

En la figura 7.14 podemos ver un diagrama de secuencia del proceso de creación. En un primer momento el usuario debe seleccionar un servicio y un servidor a los que asociar el disparador, momento en el que se cargan los *triggers* que se pueden elegir con la combinación de servicio-servidor actual. A continuación se introduce un nombre identificativo, contrastado mediante una llamada AJAX contra la base de datos de la aplicación para que sea único. Superados estos pasos, se realiza la validación del resto de campos.

Los disparadores creados pueden ser o bien *autónomos* o bien *supeditados*,

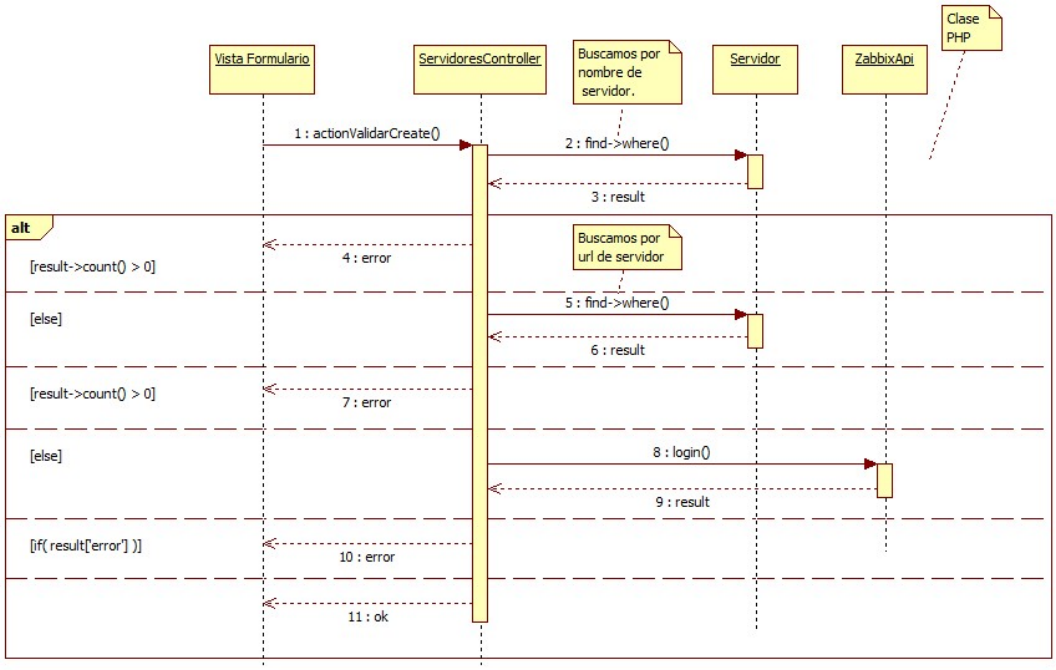


Figura 7.12: Diagrama de secuencia de la validación de servidor.

```
$objects = $api->triggerGet([
    'hostids' => $hosts,
    'triggerids' => $ids,
    'output' => ['triggerid','value'],
    'filter' =>['status' => 0],
]);
```

Figura 7.13: Consulta de estado de los *triggers*

pero en ambos casos necesitamos que se compruebe constantemente el estado de los *triggers* que se han vinculado para detectar cambios de estado. Esta comprobación, realizada a través de la API de Zabbix, se automatiza con la ejecución de *scripts* en PHP a través de un *daemon*. En la figura 7.13 podemos observar la llamada realizada a la API a través de la *PhpZabbixApi*, en la que se pasan como argumento los ids de los *triggers* vinculados a disparadores de la aplicación. Una vez consultado el estado, si este ha cambiado respecto al registrado en la aplicación, o bien se activa el disparador, publica la incidencia si es autónomo y se notifica por email al administrador, o bien, cuando se ha configurado el cierre automático, se desactiva el disparador publicando un último mensaje.

7.4.4. Incidencias

Los cambios de estado de los *triggers* de Zabbix generan las incidencias. Por lo tanto, a diferencia del resto de entidades de la aplicación, éstas son elementos

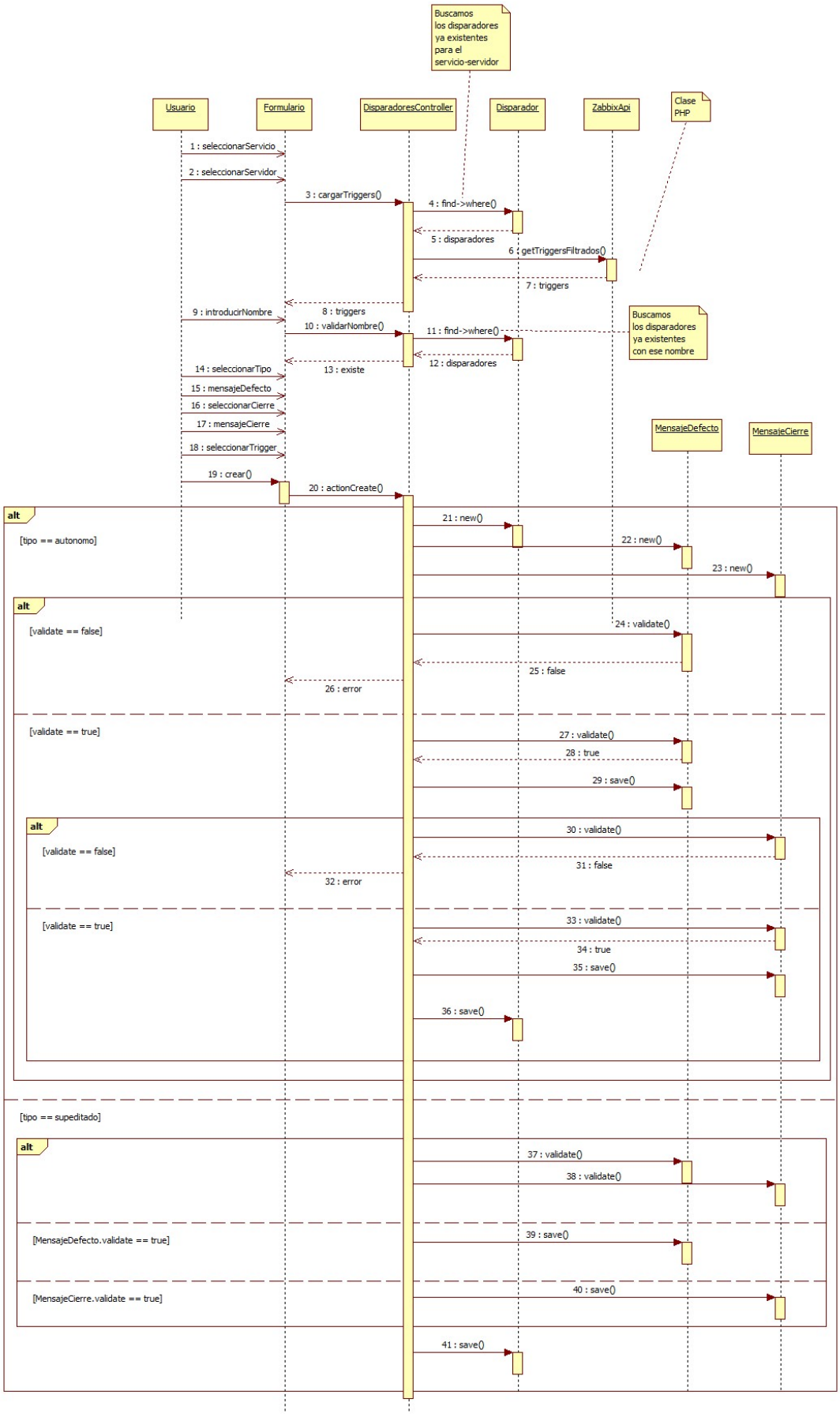


Figura 7.14: Diagrama de secuencia de la creación de disparador.

de creación automática, independientes del usuario. Sin embargo, soportan un mayor número de operaciones que el resto: publicación, descarte, visualización de detalles, finalización o eliminación de la incidencia y adición, edición o eliminación de mensajes. Las posibilidades mostradas para cada incidencia individual dependerán de su estado: no podremos, por ejemplo, descartar una incidencia que haya sido publicada.

Cuando un disparador haya sido configurado como autónomo, sus incidencias se publicarán automáticamente. En el caso de los disparadores supeditados, la incidencia se mostrará en las primeras posiciones de la tabla de incidencias. Sea cual sea el estado de la incidencia, siempre podremos eliminarla o consultar la información detallada. Cuando contemos con una incidencia generada por un disparador supeditado podremos publicarla o descartarla. Si optamos por la última, tan sólo quedarán las opciones de eliminarla o consultar los detalles. En cambio, si publicamos la incidencia, pasaremos a poder añadir un mensaje o finalizarla. Para modificar o eliminar un mensaje publicado tendremos que acceder a los detalles y, a la derecha de cada mensaje, pulsar el botón correspondiente.

Incidencias Programadas

La aplicación contempla la situación en que el administrador conoce de antemano una incidencia que va a producirse y por lo tanto crea una incidencia programada con antelación. Este tipo de incidencias pueden bloquear disparadores de forma que mientras estén vigentes no se consultará el estado de sus *triggers* y no se crearán incidencias regulares. En la creación de la incidencia, se seleccionan los disparadores que se desean bloquear y al menos un servicio al que afectará la incidencia, se indican las fechas de inicio y fin, se introduce el mensaje que la acompañará en su publicación, se establece si se finalizará automáticamente y se podrá definir un mensaje por defecto para el cierre.

Al igual que las incidencias regulares, cuentan con las operaciones de visualización de detalles, finalización o eliminación de la incidencia y adición, edición o eliminación de mensajes, pero también se podrá modificar, algo que, al ser las regulares creadas automáticamente, no se contemplaba.

Visualización Pública

El objetivo final de las incidencias es informar al usuario público sobre el estado de los servicios. Esta información se mostrará a través de una tabla en la que se enumeran los distintos servicios de la aplicación, permitiendo al usuario navegar semanalmente y consultar las distintas incidencias que se hayan dado, teniendo la posibilidad de consultar información acerca de cada una de ellas. En las figuras 7.15 y 7.16 podemos ver bocetos de esta presentación.

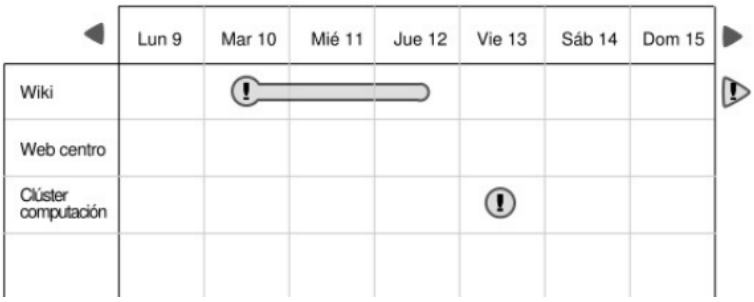


Figura 7.15: Boceto de la tabla temporal de incidencias.

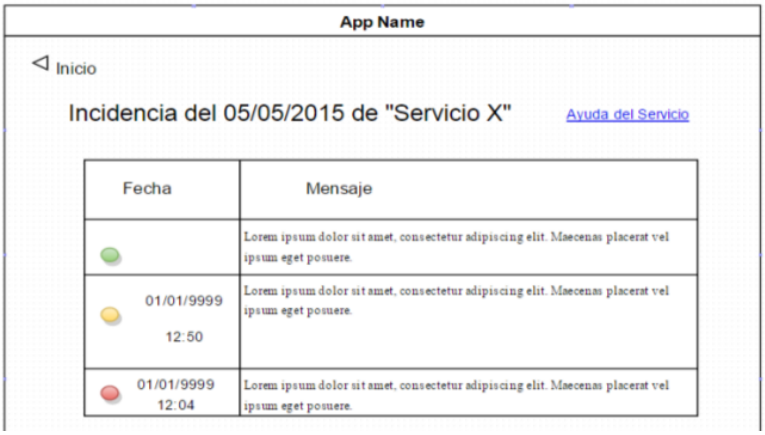


Figura 7.16: Boceto de la información de una incidencia de la tabla temporal.

Los datos necesarios para dibujar la tabla temporal son obtenidos del servidor a través de llamadas AJAX, tanto al cargar por primera vez la página como al navegar entre semanas en la tabla. Cada una de las incidencias representadas en la tabla sirve de enlace para acceder a los mensajes de la misma. Cada celda de la columna de servicios servirá de enlace a la página establecida en sus atributos como "URL del servicio".

7.5. Interfaz

Como se ha comentado anteriormente, el diseño de la interfaz de usuario pasó por una etapa inicial en la que se realizaron una serie de bocetos para, tras su aprobación, pasar a realizar el diseño a la par que la implementación, tratando de cumplir en todo momento los principios de diseño de Nielsen. El objetivo es que, pese a que los usuarios finales tendrán conocimientos informáticos, la interfaz resulte comprensible, clara y funcional para todo tipo de usuario. A continuación nombramos los principios que se han tenido en cuenta y qué medidas se tomaron

(a) Barra de navegación. (b) Breadcrumbs

Figura 7.17: Visibilidad del sistema.

para cumplirlos.

- **Visibilidad del estado del sistema**. En toda página de la aplicación se presenta al usuario una barra de navegación en la parte superior en la que se distingue en qué sección se encuentra. Además, se hace uso del widget "Breadcrumbs", que permite proporcionar al usuario información sobre en qué página concreta dentro de la sección se encuentra, así como enlaces a niveles anteriores. Podemos ver un ejemplo de estas dos medidas en la figura 7.17.

- **Utilizar el lenguaje de los usuarios**. En este caso, debemos diferenciar dos situaciones:

  - Administradores: los términos técnicos empleados en la parte de administración de la aplicación son términos con los que los administradores están familiarizados, ya sea por su uso del sistema Zabbix o por sus conocimientos informáticos. Los diálogos presentados son escasos, con lenguaje claro y directo.
  - Usuario público: en este caso no tienen contacto con los términos técnicos de la aplicación, ya que tan sólo consultarán la tabla temporal de incidencias y los mensajes relacionados. Será responsabilidad del administrador que publique mensajes el que éstos empleen un lenguaje comprensible por cualquier tipo de usuario.

- **Control y libertad para el usuario**. Ante cualquier operación que el administrador se disponga a hacer tendrá siempre disponible la opción de cancelar la misma. Las modificaciones realizadas nunca serán definitivas, y acciones "terminales" como son el descarte de una incidencia, su finalización o eliminación cuentan con diálogos de confirmación.

- **Consistencia y estándares**. Un aspecto importante es que el usuario sepa en todo momento qué significa cada palabra, icono o acción de la aplicación. En todas las secciones se emplean los mismos botones para acceder a las mismas operaciones, con las mismas etiquetas y los mismos colores. Además,

la disposición de los elementos en las distintas secciones es idéntica, es decir, en todas las secciones encontraremos, por ejemplo, el botón para editar una entidad en los mismos lugares, con el mismo icono y la misma etiqueta. Toda entidad tiene una única denominación, de forma que no encontraremos dos palabras distintas designando una misma entidad.

- **Minimizar la carga de la memoria del usuario**. Además del ya mencionado "Breadcrumbs" para ayudar al usuario a reconocer en todo momento qué parte exacta de la aplicación se encuentra, en operaciones en que el administrador puede necesitar información no perteneciente a esa vista, ésta se ha incluido. Por ejemplo, cuando operamos sobre las incidencias para añadir mensajes tendremos visible toda la información de la incidencia, incluida una lista de los mensajes anteriores.

- **Diálogos estéticos y diseño minimalista**. En toda pantalla se ha procurado mostrar únicamente la información pertinente para las acciones que en ella se desarrollan, evitando así que información irrelevante disminuya la visibilidad de la realmente importante.

- **Ayuda a los usuarios y documentación**. Se presentan diálogos de ayuda en aquellas secciones en que pueden surgir dudas al usuario, como puede ser qué implica cada campo de un formulario o las consecuencias de una acción. Además se proporciona un enlace en el pie de página que da acceso al manual de usuario.

7.6. Seguridad

El uso del framework Yii también ha ahorrado trabajo y facilitado el desarrollado en cuanto a medidas de seguridad. En esta sección se comentarán algunas de ellas.

Control de acceso

Un punto que necesita especial protección es la identificación de los usuarios. En este proyecto se identifica al usuario mediante una cuenta de correo y una contraseña. Se calcula un valor *hash* de la contraseña y se almacena en la base de datos. De esta forma, cuando el usuario vuelve a conectarse, se recalcula el valor hash de la contraseña proporcionada y se compara con el valor almacenado. Este proceso es uno de los que facilita el framework, proporcionando la encapsulación de estas funciones en métodos de una clase que emplea los métodos propios de

PHP para el cálculo de valores hash[4].

Por otra parte también se ha implementado una limitación de los intentos de acceso. Para cada usuario, se guardará desde qué dirección IP se está tratando de conectar. Si alcanza tres intentos sin conseguir identificarse correctamente, la cuenta será bloqueada durante media hora. Para alcanzar esta funcionalidad se creó una tabla específica en la base de datos y un script PHP que elimina los bloqueos una vez superada la media hora.

Por último, Yii proporciona la opción de declarar reglas de acceso para cada uno de los métodos de los controladores. De esta manera se permite restringir el acceso a funcionalidad específica en base a roles. En esta ocasión únicamente se contemplan dos roles: el visitante y el usuario identificado. Declarando estas reglas en cada controlador, para todos sus métodos, se asegura que tan sólo los usuarios con los roles especificados puedan acceder a ellos.

Otras medidas

Como se mencionó anteriormente, Yii también proporciona el patrón Active Record. Esto permite que toda consulta o modificación de la base de datos se realice a través de estas clases, evitando así la posibilidad de manipular las acciones con inyecciones de SQL.

Se han tomado también medidas para proteger el proyecto ante ataques *XSS* o *cross-site*. Estos ataques consisten en la introducción en el sistema de datos que, cuando se muestren posteriormente, puedan producir daños. Por ejemplo, si en lugar de un nombre de servicio se introduce código JavaScript, cada vez que se muestre el nombre del servicio se ejecutará ese código. Para evitar estas situaciones Yii proporciona métodos con los que codificar la información mostrada como texto plano.

Otro tipo de ataques contra los que ofrece protección son los *Cross Site Request Forgery*, esto es, peticiones que se entienden como realizadas por el usuario, pero que en realidad provienen de comandos no autorizados. Yii proporciona protección con tan sólo habilitar la validación CSRF, sin que sea preciso desarrollar una sóla línea de código. Únicamente se tendrá que proporcionar el valor almacenado en una cookie cuando se empleen herramientas externas al framework.

Las comunicaciones entre el cliente y el servidor se realizan a través de un canal seguro en el que se cifran los datos al emplear el protocolo HTTPS[17],

[4]En la actualidad PHP emplea el algoritmo *Blowfish*, contra el que todavía no se han encontrado técnicas de criptoanálisis efectivas

protegiendo de esta manera la información sensible que el usuario comparta con la aplicación y viceversa.

7.7. Mantenimiento

Se han empleado las facilidades que ofrece Yii para el mantenimiento de logs para el registro de errores en la aplicación. Para ello se han configurado tres destinos diferentes en el archivo de *web.php* de la configuración del framework:

- Archivos del sistema: se especifican empleando la clase *FileTarget* como uno de los destinos o *targets* para los logs. A cada uno de estos destinos pueden atribuírsele distintos niveles de log, categorías y la ruta específica en que guardar el archivo. Con el nivel se indica qué tipo de registros guardar (de error, de avisos, trazas establecidas por el programador o de información). La categoría puede determinarse en cada caso, de forma que se puede realizar una clasificación propia.

  Para poder realizar esta clasificación, creando una categoría por cada sección, se crea una clase que extiende el controlador base del framework. A continuación se establece en la configuración que los errores serán manejados por el método *actionError* de esta clase. Por último, se desarrolla el controlador de cada sección extendiendo la clase creada y sobrescribiendo el método de manejo de errores. De esta forma, ante la ocurrencia de errores en la funcionalidad de cada sección podremos categorizar su registro.

- Base de datos: Yii proporciona un script de SQL para la creación de una tabla en la base de datos de la aplicación. Indicando la clase *DbTarget* como uno de los destinos para los logs se guardarán en la base de datos de forma transparente al desarrollador.

- Envío de email: en esta ocasión, además de especificar la clase *EmailTarget* como uno de los destinos, se deberá configurar el componente *mailer* del framework.

La configuración necesaria para estos tres destinos se detalla en la sección 6 del manual de despliegue.

Capítulo 8

Pruebas

En este capítulo se recogen las pruebas realizadas para confirmar que se ha alcanzado el estado de "Terminado" necesario para la validación de cada historia de usuario de los diferentes Sprints. Se realizan por tanto al término de la implementación de cada historia de usuario, repitiendo aquellas pruebas con resultado adverso al finalizar su desarrollo en el Sprint siguiente. En el momento del Sprint final se han realizado de nuevo todas las pruebas aquí recogidas, con el objetivo de asegurar el correcto funcionamiento una vez desplegada la aplicación.

8.1. Sprint 1

1. ---

   - **Descripción de la prueba:** en la página principal, pulsar "Admin Panel", introducir una cuenta de usuario y su contraseña en el formulario presentado y pulsar "Entrar".
   - **Precondición:** existe una cuenta de administrador en la base de datos con esos valores.
   - **Resultado esperado:** Los datos del usuario son validados y se le redirige a la sección de incidencias.
   - **Estado:** Aceptado

2. ---

   - **Descripción de la prueba:** en la página principal, pulsar "Admin Panel", introducir una cuenta de usuario y su contraseña en el formulario presentado y pulsar "Entrar".
   - **Precondición: no** existe una cuenta de administrador en la base de datos con esos valores.

- **Resultado esperado:** Los datos del usuario son incorrectos y se le muestra un mensaje.
- **Estado:** Aceptado

3. ---

- **Descripción de la prueba:** en la sección de servicios, pulsar "Crear Servicio" y rellenar el formulario presentado. Pulsar "Crear".
- **Precondición**: no existe ya un servicio con el nombre aportado en el formulario.
- **Resultado esperado:** se crea el servicio, siendo visible en la lista de la sección de servicios.
- **Estado:** Aceptado.

4. ---

- **Descripción de la prueba:** en la sección de servicios, pulsar "Crear Servicio" y rellenar el formulario presentado. Pulsar "Crear".
- **Precondición:** existe ya un servicio con el nombre aportado en el formulario.
- **Resultado esperado:** se advierte al usuario de la existencia de un servicio con el mismo nombre.
- **Estado:** Aceptado.

5. ---

- **Descripción de la prueba:** acceder a la sección de servicios.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se muestra una tabla en que se listan los servicios existentes en la aplicación.
- **Estado:** Aceptado.

6. ---

- **Descripción de la prueba:** en la sección de servicios, pulsar sobre el botón "Editar" de un servicio de la tabla. Se presenta un formulario en que modificar los datos del servicio. Se pulsa "Guardar".
- **Precondición:** usuario identificado.
- **Resultado esperado:** se pueden observar los cambios efectuados en la sección de servicios.
- **Estado:** Aceptado.

7. ---

   - **Descripción de la prueba:** en la sección de servicios, pulsar sobre el botón "Eliminar" de un servicio de la tabla. Confirmar la eliminación en el diálogo presentado.
   - **Precondición:** usuario identificado.
   - **Resultado esperado:** el servicio se ha eliminado de la base de datos y no está presente en la tabla de la sección de servicios.
   - **Estado:** Aceptado.

8. ---

   - **Descripción de la prueba:** en la sección de servidores, pulsar el botón "Crear servidor". Completar el formulario presentado y pulsar "Crear".
   - **Precondición**: no existe un servidor con el mismo nombre o la misma URL en la aplicación.
   - **Resultado esperado:** se establece conexión correctamente y en consecuencia se crea el servidor, devolviendo al usuario a la sección de servidores.
   - **Estado:** Aceptado.

9. ---

   - **Descripción de la prueba:** en la sección de servidores, pulsar el botón "Editar". Se presenta un formulario en el que modificar los datos del servidor. Pulsar el botón "Guardar".
   - **Precondición:** usuario identificado.
   - **Resultado esperado:** se pueden apreciar los cambios guardados en la tabla de la sección de servidores.
   - **Estado:** Aceptado.

10. ---

    - **Descripción de la prueba:** acceder a la sección de servidores.
    - **Precondición:** usuario identificado.
    - **Resultado esperado:** se muestra una tabla en que se listan los servidores existentes en la aplicación.
    - **Estado:** Aceptado.

11. ---

- **Descripción de la prueba:** acceder a la sección de servidores, pulsar el botón "Eliminar" de uno de los servidores listados en la tabla. Confirmar la eliminación en el diálogo presentado.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se elimina correctamente el servidor de la base de datos y ya no está presente en la tabla de servidores.
- **Estado:** Incorrecto.

8.2. Sprint 2

1. ---

- **Descripción de la prueba:** acceder a la sección de servidores, pulsar el botón "Eliminar" de uno de los servidores listados en la tabla. Confirmar la eliminación en el diálogo presentado.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se elimina correctamente el servidor de la base de datos y ya no está presente en la tabla de servidores.
- **Estado:** Aceptado.

2. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *autónomo* como tipo de disparador e indicando un mensaje por defecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador junto con el mensaje, quedando asociado al mismo en la base de datos.
- **Estado:** Aceptado.

3. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *supeditado* como tipo de disparador e indicando un mensaje por defecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador junto con el mensaje, quedando asociado al mismo en la base de datos.

- **Estado:** Aceptado.

4. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *supeditado* como tipo de disparador, sin introducir un mensaje por defecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador en la base de datos.
- **Estado:** Aceptado.

5. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *autónomo* como tipo de disparador, sin introducir un mensaje por defecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se advierte al usuario de la necesidad de introducir un mensaje por defecto.
- **Estado:** Aceptado.

6. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado activando el cierre automático de incidencias e introduciendo un mensaje por defecto para tal efecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador junto con el mensaje, quedando asociado al mismo en la base de datos.
- **Estado:** Aceptado.

7. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado activando el cierre automático de incidencias sin introducir un mensaje por defecto para tal efecto.
- **Precondición:** usuario identificado.

- **Resultado esperado:** se advierte al usuario de la necesidad de introducir un mensaje por defecto.
- **Estado:** Aceptado.

8. ---

- **Descripción de la prueba:** acceder a la sección de disparadores.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se muestra una tabla en que se listan los disparadores existentes en la aplicación.
- **Estado:** Aceptado.

9. ---

- **Descripción de la prueba: acceder a la sección de disparadores. Sobre uno de los dispuestos en la tabla en que se listan, seleccionar uno pulsando el botón "Editar" de su fila. Modificar los datos presentados en un formulario. Pulsar el botón "Guardar".**
- **Precondición:** usuario identificado. Al menos un disparador creado.
- **Resultado esperado:** se actualizan correctamente los datos modificados en el disparador.
- **Estado:** Incorrecto.

10. ---

- **Descripción de la prueba:** acceder a la sección de disparadores. Sobre uno de los dispuestos en la tabla en que se lista, seleccionar uno pulsando el botón "Eliminar" de su fila. Aceptar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos un disparador creado.
- **Resultado esperado:** Se elimina correctamente el disparador de la base de datos de la aplicación.
- **Estado:** Aceptado.

11. ---

- **Descripción de la prueba:** acceder a la sección de incidencias.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se muestra una tabla en que se listan las incidencias existentes en la aplicación.
- **Estado:** Aceptado.

12.

- **Descripción de la prueba:** acceder a la sección de incidencias. Sobre una de las incidencias listadas en la tabla presentada, pulsar el botón "Publicar". Introducir un mensaje para asociar a la publicación y pulsar el botón "Publicar".
- **Precondición:** usuario identificado. Existe al menos una incidencia que aún no ha sido publicada.
- **Resultado esperado:** se actualiza la incidencia y pasa a ser visible en la parte pública de la aplicación.
- **Estado:** Aceptado.

13.

- **Descripción de la prueba:** acceder a la sección de incidencias. Sobre una de las incidencias listadas en la tabla presentada, pulsar el botón "Añadir Mensaje". Introducir un mensaje para asociar a la publicación y pulsar el botón "Añadir".
- **Precondición:** usuario identificado. Existe al menos una incidencia publicada.
- **Resultado esperado:**
- **Estado:**

14.

- **Descripción de la prueba:** acceder a la sección de incidencias. Sobre una de las dispuestas en la tabla en que se listan, seleccionar una pulsando el botón "Eliminar" de su fila. Aceptar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia creada.
- **Resultado esperado:** Se elimina correctamente la incidencia de la base de datos de la aplicación.
- **Estado:** Incorrecto.

8.3. Sprint 3

1.

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *autónomo* como tipo de disparador e indicando un mensaje por defecto.

- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador junto con el mensaje, quedando asociado al mismo en la base de datos.
- **Estado:** Aceptado.

2. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *supeditado* como tipo de disparador e indicando un mensaje por defecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador junto con el mensaje, quedando asociado al mismo en la base de datos.
- **Estado:** Aceptado.

3. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *supeditado* como tipo de disparador, sin introducir un mensaje por defecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador en la base de datos.
- **Estado:** Aceptado.

4. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado seleccionando *autónomo* como tipo de disparador, sin introducir un mensaje por defecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se advierte al usuario de la necesidad de introducir un mensaje por defecto.
- **Estado:** Aceptado.

5. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado activando el cierre automático de incidencias e introduciendo un mensaje por defecto para tal efecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador junto con el mensaje, quedando asociado al mismo en la base de datos.
- **Estado:** Aceptado.

6. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado activando el cierre automático de incidencias sin introducir un mensaje por defecto para tal efecto.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se advierte al usuario de la necesidad de introducir un mensaje por defecto.
- **Estado:** Aceptado.

7. ---

- **Descripción de la prueba:** acceder a la sección de disparadores, pulsar el botón "Crear Disparador" y rellenar el formulario presentado sin activar el cierre automático de incidencias, introduciendo un mensaje por defecto para el cierre.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se crea correctamente el disparador junto con el mensaje, quedando asociado al mismo en la base de datos.
- **Estado:** Aceptado.

8. ---

- **Descripción de la prueba: acceder a la sección de disparadores. Sobre uno de los dispuestos en la tabla en que se listan, seleccionar uno pulsando el botón "Editar" de su fila. Modificar los datos presentados en un formulario. Pulsar el botón "Guardar".**
- **Precondición:** usuario identificado. Al menos un disparador creado.
- **Resultado esperado:** se actualizan correctamente los datos modificados en el disparador.

- **Estado:** Aceptado.

9. ---

- **Descripción de la prueba:** acceder a la sección de incidencias. Sobre una de las dispuestas en la tabla en que se listan, seleccionar una pulsando el botón "Eliminar" de su fila. Aceptar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia creada.
- **Resultado esperado:** Se elimina correctamente la incidencia de la base de datos de la aplicación.
- **Estado:** Incorrecto.

10. ---

- **Descripción de la prueba:** acceder a la sección de incidencias, pulsando sobre el botón "Detalle" de una que se haya publicado y cuente con mensajes añadidos. En la página de detalle, pulsar el botón "Editar" dispuesto a la derecha de uno de los mensajes añadidos. Modificar el mensaje publicado que se presenta en un formulario. Pulsar el botón "Guardar".
- **Precondición:** usuario identificado. Al menos una incidencia creada, publicada y con mensajes añadidos.
- **Resultado esperado:** se observa el cambio en el mensaje modificado al acceder a la página de detalle de la incidencia.
- **Estado:** Aceptado.

11. ---

- **Descripción de la prueba:** acceder a la sección de incidencias, pulsando sobre el botón "Detalle" de una que se haya publicado y cuente con mensajes añadidos. En la página de detalle, pulsar el botón "Eliminar" dispuesto a la derecha de uno de los mensajes añadidos y confirmar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia creada, publicada y con mensajes añadidos.
- **Resultado esperado:** el mensaje no aparece en la página de detalle de la incidencia ni en la vista pública de los mensajes de la incidencia.
- **Estado:** Incorrecto.

12. ---

- **Descripción de la prueba:** acceder a la sección de incidencias y pulsar sobre el botón "Descartar" de una incidencia de las dispuestas en la tabla que no haya sido ya publicada. Confirmar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia creada y no publicada.
- **Resultado esperado:** se actualiza el estado de la incidencia y las únicas operaciones permitidas son las de consultar detalle y eliminar.
- **Estado:** Aceptado.

13. ---

- **Descripción de la prueba:** acceder a la sección de incidencias y pulsar sobre el botón "Finalizar". Introducir un mensaje en el formulario que se muestra. Pulsar el botón "Finalizar" y confirmar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia creada y publicada.
- **Resultado esperado:** se actualiza el estado de la incidencia y las únicas operaciones permitidas son las de consultar detalle y eliminar.
- **Estado:** Aceptado.

14. ---

- **Descripción de la prueba:** acceder a la sección de incidencias.
- **Precondición:** usuario identificado.
- **Resultado esperado:** se muestra una tabla en que se listan las incidencias manuales/programadas existentes en la aplicación.
- **Estado:** Aceptado.

15. ---

- **Descripción de la prueba:** acceder a la sección de incidencias y pulsar el botón "Crear incidencia". Completar el formulario presentado seleccionando un disparador, indicando fechas de inicio y fin y mensajes para la apertura y el cierre. Pulsamos el botón "Crear".
- **Precondición:** usuario identificado. Al menos un disparador creado.
- **Resultado esperado:** se crea la incidencia y es visible en la tabla en que se listan las incidencias manuales/programadas.
- **Estado:** Aceptado.

16. ---

- **Descripción de la prueba:** acceder a la sección de incidencias y, sobre una de las incidencias listadas en la tabla de incidencias programadas, pulsar el botón "Publicar". Introducir un mensaje en el formulario que se presenta. Pulsar el botón "Publicar".
- **Precondición:** usuario identificado. Al menos una incidencia programada creada, cuya fecha de inicio es inferior a la actual y no ha sido publicada.
- **Resultado esperado:** se actualiza el estado de la incidencia, visible en el detalle y en la tabla de incidencias.
- **Estado:** Aceptado.

17. ---

- **Descripción de la prueba:** acceder a la sección de incidencias y, sobre una de las incidencias listadas en la tabla de incidencias programadas, pulsar el botón "Añadir Mensaje". Introducir un mensaje para asociar a la publicación y pulsar el botón "Añadir".
- **Precondición:** usuario identificado. Al menos una incidencia programada creada y publicada.
- **Resultado esperado:** se añade el mensaje a la lista de mensajes publicados de la incidencia.
- **Estado:** Aceptado.

18. ---

- **Descripción de la prueba:** acceder a la sección de incidencias, pulsando sobre el botón "Detalle" de una incidencia programada que se haya publicado y cuente con mensajes añadidos. En la página de detalle, pulsar el botón "Editar" dispuesto a la derecha de uno de los mensajes añadidos. Modificar el mensaje publicado que se presenta en un formulario. Pulsar el botón "Guardar".
- **Precondición:** usuario identificado. Al menos una incidencia programada creada, publicada y con mensajes añadidos.
- **Resultado esperado:** se observa el cambio en el mensaje modificado al acceder a la página de detalle de la incidencia.
- **Estado:** Aceptado.

19. ---

- **Descripción de la prueba:** acceder a la sección de incidencias, pulsando sobre el botón "Detalle" de una incidencia programada que se

haya publicado y cuente con mensajes añadidos. En la página de detalle, pulsar el botón "Eliminar" dispuesto a la derecha de uno de los mensajes añadidos y confirmar el diálogo presentado.

- **Precondición:** usuario identificado. Al menos una incidencia programada creada, publicada y con mensajes añadidos.
- **Resultado esperado:** el mensaje no aparece en la página de detalle de la incidencia ni en la vista pública de los mensajes de la incidencia.
- **Estado:** Incorrecto.

8.4. Sprint 4

1. ---

   - **Descripción de la prueba:** acceder a la sección de incidencias. Sobre una de las dispuestas en la tabla en que se listan, seleccionar una pulsando el botón "Eliminar" de su fila. Aceptar el diálogo presentado.
   - **Precondición:** usuario identificado. Al menos una incidencia creada.
   - **Resultado esperado:** Se elimina correctamente la incidencia de la base de datos de la aplicación.
   - **Estado:** Incorrecto.

2. ---

   - **Descripción de la prueba:** acceder a la sección de incidencias, pulsando sobre el botón "Detalle" de una que se haya publicado y cuente con mensajes añadidos. En la página de detalle, pulsar el botón "Eliminar" dispuesto a la derecha de uno de los mensajes añadidos y confirmar el diálogo presentado.
   - **Precondición:** usuario identificado. Al menos una incidencia creada, publicada y con mensajes añadidos.
   - **Resultado esperado:** el mensaje no aparece en la página de detalle de la incidencia ni en la vista pública de los mensajes de la incidencia.
   - **Estado:** Aceptado.

3. ---

   - **Descripción de la prueba:** acceder a la sección de incidencias y pulsar el botón "Crear incidencia". Completar el formulario presentado seleccionando los disparadores afectados, indicando fechas de inicio y fin y mensajes para la apertura y el cierre. Pulsamos el botón "Crear".
   - **Precondición:** usuario identificado. Al menos un disparador creado.

- **Resultado esperado:** se crea la incidencia y es visible en la tabla en que se listan las incidencias manuales/programadas.
- **Estado:** Aceptado.

4. ---

- **Descripción de la prueba:** acceder a la sección de incidencias, pulsando sobre el botón "Detalle" de una incidencia programada que se haya publicado y cuente con mensajes añadidos. En la página de detalle, pulsar el botón "Eliminar" dispuesto a la derecha de uno de los mensajes añadidos y confirmar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia programada creada, publicada y con mensajes añadidos.
- **Resultado esperado:** el mensaje no aparece en la página de detalle de la incidencia ni en la vista pública de los mensajes de la incidencia.
- **Estado:** Aceptado.

5. ---

- **Descripción de la prueba:** acceder a la sección de incidencias. Sobre una de las incidencias programadas dispuestas en la tabla en que se listan, seleccionar una pulsando el botón "Eliminar" de su fila. Aceptar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia creada.
- **Resultado esperado:** Se elimina correctamente la incidencia de la base de datos de la aplicación.
- **Estado:** Incorrecto.

6. ---

- **Descripción de la prueba:** acceder a la sección de incidencias y pulsar sobre el botón "Editar" de una de las incidencias programadas listadas. Modificar los datos del formulario presentado y pulsar el botón "Guardar".
- **Precondición:** usuario identificado. Al menos una incidencia programada creada y no publicada.
- **Resultado esperado:** se actualizan correctamente los datos de la incidencia.
- **Estado:** Aceptado.

7. ---

- **Descripción de la prueba:** acceder a la sección de incidencias y pulsar sobre el botón "Finalizar" de una incidencia programada publicada. Introducir un mensaje en el formulario que se muestra. Pulsar el botón "Finalizar" y confirmar el diálogo presentado.
- **Precondición:** usuario identificado. Al menos una incidencia programada creada y publicada.
- **Resultado esperado:** se actualiza el estado de la incidencia y las únicas operaciones permitidas son las de consultar detalle y eliminar.
- **Estado:** Aceptado.

8. ---

- **Descripción de la prueba:** acceder a la página principal de la aplicación.
- **Precondición:** al menos un servicio creado.
- **Resultado esperado:** se visualiza un tabla temporal en la que se muestran los servicios registrados en la aplicación.
- **Estado:** Aceptado.

9. ---

- **Descripción de la prueba:** acceder a la página principal de la aplicación.
- **Precondición:** al menos un servicio creado con una incidencia que empieza y termina en el período mostrado en la tabla.
- **Resultado esperado:** se visualiza un tabla temporal en la que se muestran los servicios registrados en la aplicación y sus incidencias correctamente.
- **Estado:** Aceptado.

10. ---

- **Descripción de la prueba:** acceder a la página principal de la aplicación.
- **Precondición:** al menos un servicio creado con una incidencia que termina en el período mostrado en la tabla y empieza en una fecha anterior.
- **Resultado esperado:** se visualiza un tabla temporal en la que se muestran los servicios registrados en la aplicación y sus incidencias correctamente.
- **Estado:** Incorrecto.

11. ---

- **Descripción de la prueba:** acceder a la página principal de la aplicación.
- **Precondición:** al menos un servicio creado con una incidencia que empieza en el período mostrado en la tabla y termina en una fecha posterior.
- **Resultado esperado:** se visualiza un tabla temporal en la que se muestran los servicios registrados en la aplicación y sus incidencias correctamente.
- **Estado:** Incorrecto.

12. ---

- **Descripción de la prueba:** acceder a la página principal de la aplicación.
- **Precondición:** al menos un servicio creado con una incidencia que empieza en una fecha anterior al período de la tabla y termina en una fecha posterior.
- **Resultado esperado:** se visualiza un tabla temporal en la que se muestran los servicios registrados en la aplicación y sus incidencias correctamente.
- **Estado:** Incorrecto.

13. ---

- **Descripción de la prueba:** accder a la página principal de la aplicación y pulsar sobre el icono de una de las incidencias presentes en la tabla temporal mostrada.
- **Precondición:** al menos un servicio presente con incidencias.
- **Resultado esperado:** se muestra una tabla con los mensajes asociados a la incidencia seleccionada ordenados del más reciente al más antiguo.
- **Estado:** Aceptado.

Capítulo 9

Conclusiones

Este trabajo de fin de grado comenzó con unos objetivos muy poco definidos, teniendo que hacer un pequeño estudio inicial para determinar de qué opciones se disponía. Este estudio dio la posibilidad de definir el alcance del proyecto y realizar una recolección de requisitos lo más completa posible. Se quería imitar el modelo de Google en su pizarra de estado de servicios, *Apps Status Dashboard*, y tenía que realizarse en comunicación con el sistema de monitorización Zabbix. El objetivo global era aportar al administrador la capacidad de informar a los usuarios de sus servicios de los distintos problemas que podían surgir en ellos. Esta etapa inicial del proyecto evidenció que los requisitos serían cambiantes y que la elección de una metodología ágil era lo más acertado.

Como se refleja en apartados de esta memoria, se produjeron numerosas variaciones en los requisitos durante el desarrollo, dando lugar a la aparición de múltiples historias de usuario nuevas a lo largo de los Sprints, así como redefiniendo las existentes según cambiaban los requisitos definidos por los clientes. Gracias al tipo de metodología escogida este continuo cambio no supuso un esfuerzo inabarcable, y finalmente se pudo cumplir con todos los requisitos definidos y alcanzar los objetivos del proyecto.

Entre las metas alcanzadas con éxito se incluyen el desarrollo de un canal de comunicación entre los servidores de Zabbix y la aplicación, permitiendo al administrador seleccionar qué información recuperar y cómo gestionarla; la automatización del proceso de observación y actualización de dicha información; y dotar al administrador de funcionalidades para su publicación, haciéndola así accesible a cualquier usuario, en un formato que permite su contextualización temporal.

En concreto, el trabajo abordado por el proyecto, y reflejado en esta memoria, se resume en los siguientes puntos:

- Se han desarrollado métodos de comunicación con la API de Zabbix para

recuperar la información relativa al estado de sus triggers.

- Se ha desarrollado funcionalidad que permite al administrador:
  - Comunicarse con distintos servidores de Zabbix.
  - Seleccionar sobre qué triggers de Zabbix observar los cambios de estado.
  - Configurar la respuesta ante dichos cambios de estado.
  - Ser advertido de que se ha producido un cambio de estado.
  - Crear publicaciones en la aplicación para cada uno de estos cambios de estado.
  - Crear publicaciones para advertir con antelación de posibles incidencias.
  - Organizar estas publicaciones en entidades que identifiquen a los servicios ofrecidos por sus sistemas.
  - Modificar y ampliar las publicaciones realizadas.
  - Consultar las publicaciones, organizándolas temporalmente en una tabla de acceso público.
- Se ha desarrollado la interfaz web de la aplicación de manera que el dispositivo de acceso no limite la funcionalidad ofrecida ni la presentación de información.

La realización de este trabajo de fin de grado ha servido para ampliar las competencias adquiridas durante el grado, tanto a nivel tecnológico, por haber empleado tecnologías en un principio desconocidas o en las que se tenía escasa formación, como a nivel de gestión de proyectos, habiendo ganado experiencia en un tipo de metodología de desarrollo de software cada vez más popular.

Mejoras y ampliaciones

Habiendo cumplido con los requisitos establecidos y desarrollado toda la funcionalidad requerida, existen ampliaciones de la misma que podrían ser de interés para futuras versiones de la aplicación. La estructura del proyecto, empleando el patrón MVC, hace que el acoplamiento de nuevas funciones no requiera grandes esfuerzos. Las posibles extensiones que se enumeran a continuación fueron descartadas desde las etapas iniciales por ser de carácter secundario respecto a los objetivos del proyecto o bien por no haberse desarrollado su definición suficientemente.

- Gráficos de historial: si bien es cierto que la elaboración de gráficas es una de las funcionalidades que aporta el sistema Zabbix, debemos tener en cuenta que en este proyecto la aplicación se puede relacionar con más de un servidor Zabbix. Puede resultar de gran interés para los administradores poder consultar el historial de incidencias de un disparador, un servidor o un servicio en formato de gráfica. Esta funcionalidad también podría ofrecerse a los usuarios de la sección pública limitándolo a gráficos de historial de incidencias en los servicios.

- Suscripción de alertas: una posible mejora en la funcionalidad ofrecida al usuario público es la opción de recibir correos alertándolo de las incidencias ocurridas en los servicios de su interés, de forma que pudiese conocer el estado del servicio de antemano y no consultando la página tras un comportamiento anómalo del servicio.

- Habilitación y deshabilitación de disparadores: puede darse la situación en que un administrador necesite inhabilitar temporalmente un disparador para que este no genere incidencias. En este desarrollo hay dos opciones de conseguir un efecto similar: creando una incidencia programada que afecte a dicho disparador, con lo que quedaría bloqueado; o configurando el disparador como supeditado e ignorando la incidencia creada con el cambio de estado del trigger de Zabbix, es decir, ni publicar ni descartar la incidencia. Sería de interés poder establecer un período durante el cual se ignoren los cambios de estado del trigger de Zabbix vinculado al disparador.

- Creación de disparadores compuestos: el sistema Zabbix ofrece una potente herramienta para la creación de triggers definidos por el usuario. Sin embargo, ofreciendo la aplicación la posibilidad de comunicarse con varios servidores Zabbix distintos, una posible ampliación de la funcionalidad consistiría en la creación de disparadores compuestos. Estos se podrían definir con reglas del tipo: el disparador compuesto A se activará si el disparador B y el disparador C se encuentran activos pero el disparador D mantiene su estado normal.

Apéndice A

Glosario del Product Backlog

Término	Significado
Servidor Zabbix	Actor externo al sistema que proveerá de información a la aplicación a través de una API de programación
Disparador	Elemento que, a partir de los cambios observados en el estado de un determinado trigger en un Servidor Zabbix, genereautomáticamente una incidencia para el servicio asociado
Incidencia	Cada uno de los problemas o paradas programadas que se produzcan a partir de un disparador o por acción del administrador y que se mostrarán en la fila correspondiente de la tabla temporal
Servicio	Cada uno de los elementos que recogerá la aplicación a los que se les asociarán disparadores y que constituirán una fila de la tabla temporal, mostrando en ella sus respectivas incidencias

Cuadro A.1: Glosario de términos del Product Backlog

Apéndice B

Manual de despliegue

En esta sección explicaremos el proceso de despliegue de la aplicación junto con la instalación de los paquetes necesarios. En este despliegue asumimos que se realiza sobre el sistema operativo Ubuntu 14.04.3 LTS.

1. **Instalación de PHP 5.5**. Como superusuarios, ejecutamos en terminal el siguiente comando:

   ```
   apt-get install php5
   ```

2. **Instalación de Apache**. En un principio los paquetes de Apache se incluyen en la instalación de PHP5. Para consultarlo podemos ejecutar:

   ```
   dpkg -l |  grep apache
   ```

   Si no fuese así el caso, ejecutamos:

   ```
   apt-get install apache2
   ```

   En este momento, si accedemos a la IP del host en que estemos realizando el despliegue a través del navegador, deberíamos ver la página de bienvenida de Apache.

3. **Instalación de MySQL**. Procedemos a instalar el sistema gestor de bases de datos MySQL en su versión 5.5. Para ello ejecutamos el comando:

   ```
   apt-get install mysql-server-5.5
   ```

   Al instalar el servidor automáticamente se incluye el paquete del cliente. Si no fuese así, ejecutamos:

   ```
   apt-get install mysql-client-5.5
   ```

Por último, instalamos el paquete de MySQL para PHP5:

```
apt-get install php5-mysql
```

4. **Despliegue aplicación**. La carpeta en que se alojan las aplicaciones web por defecto con Apache es */var/www/html*. Copiamos a esta, o la carpeta que hayamos elegido el comprimido con la aplicación y lo descomprimimos.

   ```
   mv /<<ruta>>/tfg.tar.gz $APACHE_HOME/
   ```

   Situándonos en *$APACHE_HOME*:

   ```
   tar xzf tfg.tar.gz
   ```

   Una vez descomprimido, si accedemos a *«host»/tfg/src/requirements.php* en el navegador podremos comprobar si necesitamos instalar algún paquete adicional. En nuestro caso deberemos cambiar un parámetro de la configuración de PHP en Apache. En el archivo */etc/php5/apache2/php.ini* establecemos con el valor *Off* el parámetro *expose_php*.

5. **Creación de la base de datos**. Durante la instalación de MySQL hemos establecido una contraseña para el usuario *root*. Nos conectamos a MySQL como superusuarios y creamos la base datos y el usuario para la aplicación:

   ```
   mysql -uroot -p
   create database <<database>>;
   create user '<<usuario>>'@'localhost' identified by '<<contrasena>>';
   grant all privileges on <<database>>.* to '<<usuario>>'@'localhost';
   ```

   En este momento cerramos la conexión ejecutando *exit* y pasamos a conectarnos como el usuario creado:

   ```
   mysql -u<<usuario>> -p <<database>>
   source /$APACHE_HOME/tfg/database_structure.sql
   ```

   donde *$APACHE_HOME* es el directorio en que se encuentra la aplicación.

   Con la estructura de la base de datos creada, introducimos el usuario administrador. La aplicación emplea la función de PHP *password_verify*, por lo que para el cálculo del *hash* de una contraseña debe emplearse la función *password_hash*, cuyo resultado se almacena en la tabla de usuarios.

   ```
   insert into usuarios('<<email_usuario>>', '<<hash>>');
   ```

6. **Configuración de la aplicación**.

- **Base de datos**. Deberemos establecer los parámetros de acceso a la base de datos en los archivos de configuración de la aplicación. En *$APACHE_HOME/src/config/db.php* modificamos las siguientes líneas:

```
return [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=<<host>>;dbname=<<database>>',
    'username' => '<<mysql_user>>',
    'password' => '<<mysql_pass>>',
    'charset' => 'utf8',
];
```

  Además, en las primeras lineas del archivo *$APACHE_HOME/daemons/launch.php*, introducimos también los parámetros de la base de datos:

```
$HOME_URL = 'http://<<host>>/tfg/src/web/index.php';

// DB params
$db_server = '<<host>>';
$db_user = '<<mysql_user>>';
$db_pass = '<<mysql_pass>>';
$db_name = '<<database>>';
```

- **Correos de incidencias**. En el mismo archivo que el punto anterior configuramos las cuentas de email de envío y recepción de avisos de incidencias.

```
// Mail Transport
// direccion desde la que se envian los correos.
$mail_username = '<<account>>';
$mail_password = '<<password>>';

$mail_admins = array(
    '<<receiver email>>' => '<<receiver name>>',
    '<<receiver email>>' => '<<receiver name>>'
);// Lista de correos a los que enviar incidencias.
```

- **Limpieza de datos**. Deberemos establecer el período de mantemiento de los datos relativos a las incidencias, con objetivo de que la base de datos no llegue a ocupar un volumen excesivo. Para ello, debermos introducir el tiempo de antigüedad en segundos en la tabla *params_tiempo*:

```
mysql -u<<usuario>> -p <<database>>
update params_tiempo set valor = <<seconds>> where clave='
    CLEAN_DB_PERIOD'
```

- **Scripts**. Las tareas de apertura y cierre de incidencias de ambos tipos se automatizan, junto con la limpieza de la base de datos, con el uso de demonios[1]. Para iniciarlos, deberemos ejecutar en consola:

[1]Un *daemon* es un proceso que se ejecuta en segundo plano, sin interacción del usuario.

```
php $APACHE_HOME/daemons/launch.php
```

Para automatizar la ejecución de los scripts introducimos las siguientes lineas en el archivo */etc/crontab*:

```
15 *     * * *   root    $HOME_APACHE/tfg/daemons/cronscript.sh
45 *     * * *   root    $HOME_APACHE/tfg/daemons/cronscript.sh
```

De esta forma en cada minuto 15 y 45 de cada hora se ejecutará un script que comprueba que los restantes scripts de la aplicación se estén ejecutando, lanzándolos en caso contrario.

- **Mantenimiento de Logs**. Se proporcionan tres métodos de registro de errores en la aplicación, todos ellos configurables en el archivo *web.php* de la carpeta *$APACHE_HOME/tfg/src/config*:
  - Archivos del sistema: se puede especificar el destino de los logs de cada una de las secciones de la aplicación, así como los niveles de log que se desea mantener y de qué categorías. Por ejemplo, para la sección incidencias:

```
'targets' => [
    [
        'class' => 'yii\log\FileTarget',    // Tipo de destino
        'categories' => ['incidencias'],    // Categorias que
            registrar
        'logFile' => '@app/runtime/logs/incidencias.log',   //
            Ruta del archivo
        'exportInterval' => 1   // Cuantos registros almacenar
            antes de escribir en el archivo
    ],
```

- Base de datos. La creación de la tabla en que se guardarán se incluye en el script empleado para la creación de la misma en puntos anteriores. La definición de este destino es idéntica a la mostrada en el punto anterior, con la diferencia de que esta vez se emplea la clase *DbTarget*.
- Email. Para poder emplear esta característica, además de definir el destino siguiendo el modelo de los puntos anteriores, debe configurarse el componente *mailer* presente en el mismo archivo tal como sigue:

```
// Dentro del array de components:
'mailer' => [
    'class' => 'yii\swiftmailer\Mailer',
    'useFileTransport' => false,
    'transport' => [
        'class' => 'Swift_SmtpTransport',
        'host' => 'smtp.gmail.com',     // Cada servidor de correo
            tendra su configuracion propia, en este ejemplo Gmail
        'username' => 'direccion_para_enviar_correos',
        'password' => 'nuestra_contrasenha',
        'port' => '587',
        'encryption' => 'tls'
]



// Dentro del array de targets:
```

```
[
    'class' => 'yii\log\EmailTarget',
    'levels' => ['error'],  // En esta ocasion solo se envian los
        errores
    'message' => [
        'from' => 'log@dashboard.com',  // direccion a mostrar como
            origen del mensaje
        'to' => ['sergiogarciaspinola@gmail.com'],  // lista de
            correos destino
        'subject' => 'Error Log',   // Asunto del mensaje
    ]
]
```

Llegados a este punto la aplicación ya se encuentra lista para su explotación.

Apéndice C

Manual de usuario

En este anexo explicaremos cómo hacer uso de la aplicación, mostrando los pasos a seguir para realizar cada una de las operaciones disponibles en sus secciones. En el cuadro C.1 podemos ver una relación entre los iconos empleados a lo largo de la aplicación y su significado. Estos iconos son indicativos de las acciones disponibles en cada momento y son comunes a todas las secciones.

C.1. Acceso

Para acceder a la parte de administración de la aplicación deberemos identificarnos. Para ello pulsamos sobre el botón "Panel de administración" que vemos situado en la parte inferior derecha de la página principal. Se mostrará un formulario como el de la figura C.1 al usuario en el que introducir el correo electrónico y la contraseña. Si los datos son correctos, se redirigirá al usuario a la página principal de la parte de administración de la aplicación.

C.2. Servicios, Servidores y Disparadores

Las tres secciones de la aplicación que dan nombre a este apartado, pese a operar sobre entidades distintas, contemplan las mismas acciones. En todos los casos el acceso a las operaciones se encuentra en los mismos lugares de la página, con idénticos iconos y nombres. Por esta razón se explican a continuación, de manera conjunta, los pasos para realizar cada una de las operaciones en las tres secciones.

En la parte superior de la aplicación habrá en todo momento una barra de navegación con enlaces a, de izquierda a derecha, la sección pública de la aplicación, la sección de disparadores, la sección de incidencias, la sección de servicios, la sección de servidores y por último un enlace para cerrar la sesión.

Icono	Acción
	Ver detalles
	Crear
	Editar
	Eliminar
	Publicar
	Descartar
	Añadir
	Finalizar
	Guardar

Cuadro C.1: Iconos de la aplicación.

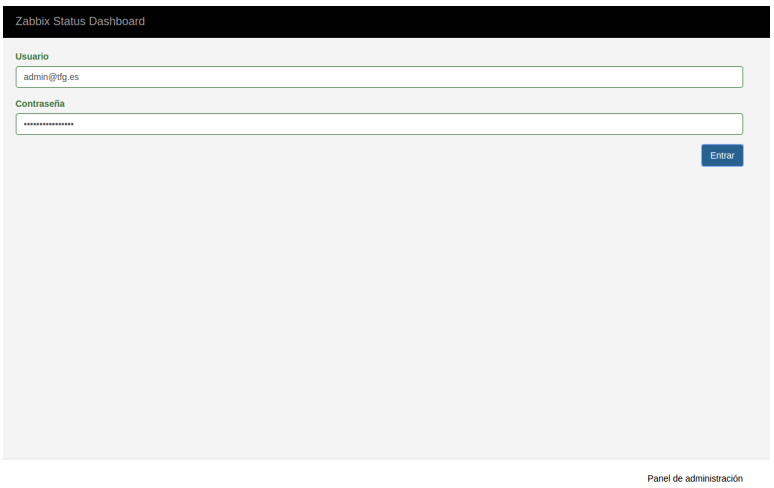


Figura C.1: Formulario de acceso

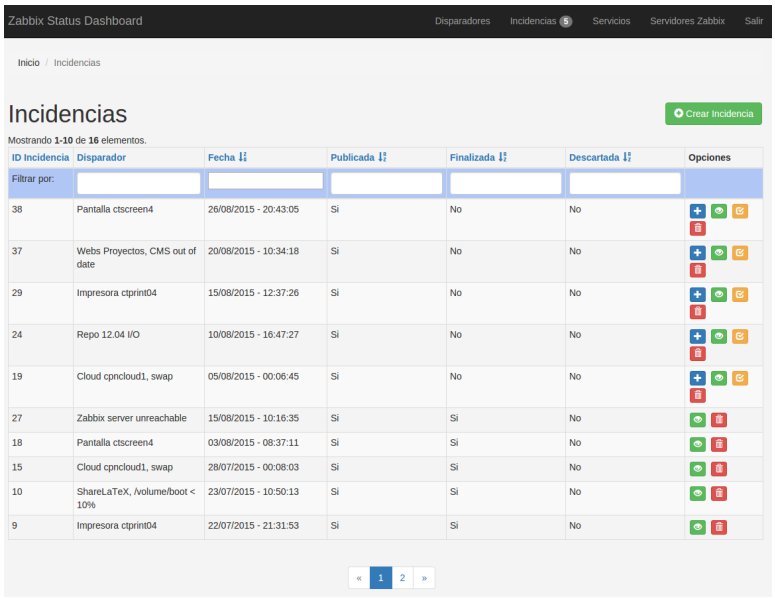


Figura C.2: Página principal de la sección de incidencias.

En cada sección se presenta una tabla en la que se listan los disparadores, servicios o servidores presentes en la aplicación. Esta tabla presentará información básica sobre cada entidad, junto con una última fila en la que se disponen las operaciones habilitadas con los iconos del cuadro C.1. La primera fila sirve como formulario de búsqueda, y podremos ordenar la tabla pulsando sobre la cabecera de cada columna.

C.2.1. Crear

En la parte superior derecha de la página principal de cada sección habrá un botón con la etiqueta “Crear «entidad»”. Se presentará un formulario para introducir:

- Servicios.
  - Un nombre identificativo.
  - Una dirección de enlace a la web del servicio (opcional).
  - Una dirección a una página de ayuda y/o información del servicio (opcional).
- Servidores.
  - Un nombre identificativo.
  - La URL de acceso a la API de Zabbix en el servidor.
  - Un nombre de usuario para acceder al servidor Zabbix.

- La contraseña del usuario.

- Disparadores.
  - Servicio al que asociar el disparador.
  - Servidor que aloja el trigger al que asociar el disparador.
  - El tipo de disparador.
  - Un mensaje por defecto para la apertura de incidencias del disparador (opcional si es de tipo supeditado).
  - Si se cierran automáticamente las incidencias.
  - Un mensaje por defecto para el cierre de incidencias del disparador (opcional si no se selecciona el cierre automático).

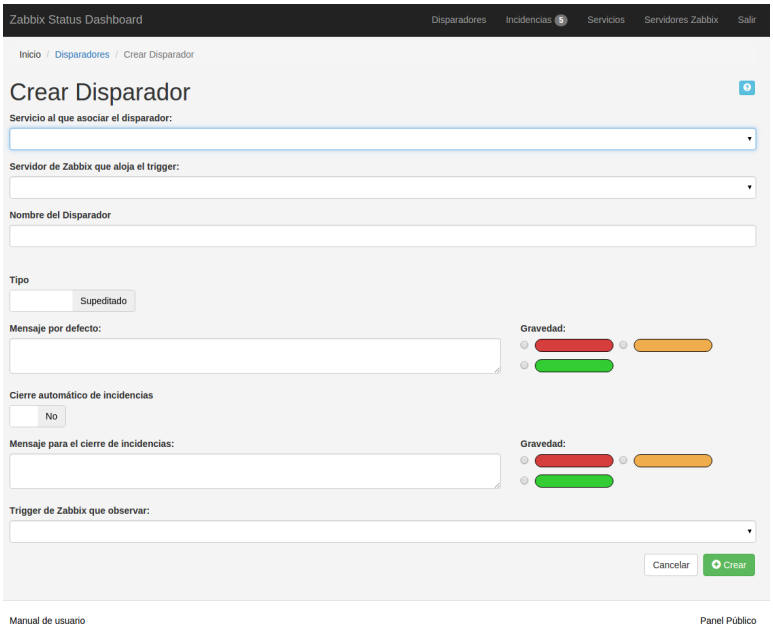


Figura C.3: Creación de un disparador

C.2.2. Consultar y Editar

En la última columna de la tabla presentada en la página principal de cada sección tendremos el icono de la operación "Ver detalles". Al pulsar este icono se redirigirá a una página en la que se muestra toda la información disponible sobre cada entidad. También en la última columna de la tabla tendremos el icono para la operación "Editar". En este caso se muestra un formulario como el de creación, pero completado con los datos actuales de la entidad. A esta operación

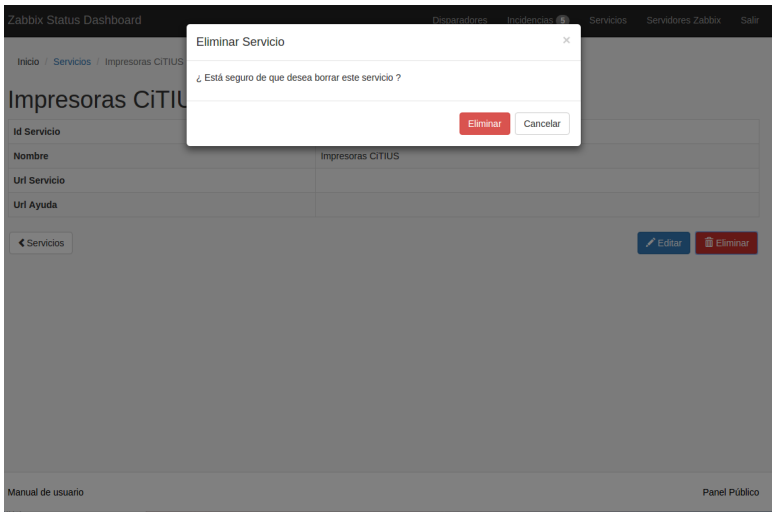


Figura C.4: Eliminación de un servicio

podemos acceder también desde la página de detalles. Una vez realizados los cambios deseados, pulsamos el botón con la etiqueta “Guardar”.

C.2.3. Eliminar

Del mismo modo que para editarla, para eliminar una entidad existen dos opciones: se tendrá disponible el botón de “Eliminar” tanto en cada una de las filas de la tabla de la página principal de la sección, como si entramos a ver los detalles de un entidad en particular. De cualquier modo, se presentará un diálogo de confirmación al usuario como el de la figura C.4 antes de eliminar la entidad. Debe tenerse en cuenta que, al eliminar un servidor o un servicio estará también eliminando todo disparador asociado e incidencia creada por tal disparador.

C.3. Incidencias

Esta sección se presenta en un apartado nuevo por su mayor funcionalidad. En este caso, al acceder a la sección de incidencias no se nos presentará tan sólo una tabla, sino una para las incidencias generadas por disparadores y otra para las incidencias que introduzcamos manualmente. Al igual que en el resto de secciones, la primera fila de cada tabla se podrá emplear como formulario de búsqueda, y sus cabeceras para ordenar los datos dispuestos.

Se explican a continuación cada una de las operaciones presentes en la última columna (podemos encontrar los iconos que las representan en el cuadro C.1), también accesibles desde la pantalla de detalles de cada incidencia, así como las operaciones disponibles sobre los mensajes.

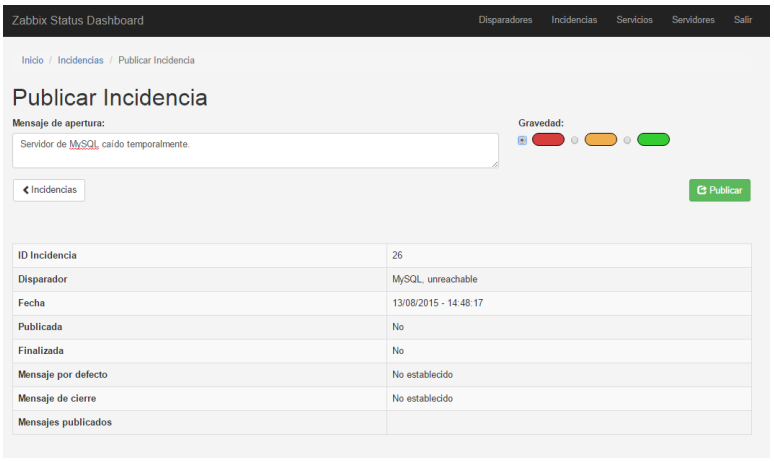


Figura C.5: Publicar incidencias.

C.3.1. Publicar

Cuando un disparador sea del tipo supeditado, sus incidencias no se publicarán automáticamente al generarse, sino que necesitarán la autorización del administrador. Pulsando sobre el botón con etiqueta "Publicar" se presentará un formulario en el que introducir dos datos: el mensaje para acompañar a la incidencia (si se estableció un mensaje por defecto en el disparador, aparecerá en este campo) y el color que tendrá. Para finalizar la operación pulsamos el botón "'Publicar". Puede verse el proceso en la figura C.5.

C.3.2. Descartar

Cuando no se quiera publicar una incidencia, pero tampoco eliminarla del sistema, tendremos la opción de descartarla. De esta manera la incidencia permanecerá en la base de datos de la aplicación, pero no se podrá ejercer mayor operación sobre ella que la consulta y la eliminación. Podremos descartar una incidencia siempre y cuando no haya sido publicada. Esta operación es accesible desde la columna de "Opciones" de la página principal de incidencias y es exclusiva de las incidencias generadas por disparadores. Debe tenerse en cuenta que, al descartar una incidencia, el disparador que la ha generado volverá a la normalidad, por lo que si el trigger de Zabbix todavía se encuentra activo, se generará una nueva incidencia.

C.3.3. Mensajes

Los mensajes acompañan en todo momento a las incidencias. Cuando se publica una incidencia, es obligatorio que se haga con un mensaje asociado, y la situación es idéntica cuando se finaliza una incidencia. Mientras no se finalice una

incidencia, también se podrán crear mensajes nuevos para actualizar la información sobre la incidencia. A continuación se explican las operaciones disponibles para los mensajes.

Añadir

Cuando se haya publicado una incidencia se habilitará la opción de añadir un mensaje. Desde la página principal de la sección de incidencias, se accede pulsando el botón "Añadir mensaje". Se presentará un formulario similar al de la publicación de una incidencia: un campo para introducir el texto y otro para la elección del color del mensaje.

Editar y Eliminar

Para realizar esta operación se deberá acceder a los detalles de la incidencia. En esta página se listan los mensajes de la incidencia y, a la derecha de cada uno, se tendrán disponibles el botón de "Editar" y el de "Eliminar". Con el primero se accederá a un formulario como el de la creación del mensaje, completado con los datos actuales para poder modificarlos. Con el segundo se presentará un diálogo de confirmación que se deberá aceptar para eliminar el mensaje. Podremos editar cualquier mensaje, pero no podremos eliminar el primer mensaje de cada incidencia.

C.3.4. Finalizar

Desde el momento en que se publica una incidencia se habilitará la operación de finalización de la misma. Pulsando el botón "Finalizar" en la columna de opciones de la tabla de incidencias se accede a un formulario similar al de publicación, ya que también se necesita un mensaje: habrá un campo para el texto del mensaje y otro para la elección del color. El formulario estará completado si se ha establecido mensaje por defecto para el cierre de incidencias del disparador correspondiente. Cuando se complete el formulario, al pulsar el botón "Finalizar" se presentará un diálogo de confirmación que deberá aceptarse para terminar la operación.

C.3.5. Crear

Esta operación está limitada a las incidencias introducidas por el administrador, ya que las incidencias regulares son creadas a partir de los disparadores, sin interacción del administrador. Pulsando el botón de "Crear incidencia" de la página principal de la sección se accede a un formulario en el que se podrá indicar:

- La serie de disparadores a los que afectará la incidencia.

- El servicio o servicios a los que se asociará la incidencia.
- La fecha de inicio de la incidencia.
- La hora de inicio de la incidencia.
- Un mensaje para la publicación.
- La fecha de finalización de la incidencia.
- La hora de finalización de la incidencia.
- Si la incidencia se finalizará automáticamente.
- un mensaje para la finalización (opcional si no se finaliza automáticamente).

C.3.6. Consultar y Editar

Al igual que en el resto de secciones, se tendrá en la columna de opciones de ambas tablas la posibilidad de consultar toda la información disponible sobre una incidencia a través del botón "Ver detalles". Sin embargo, la operación de editar estará restringida a aquellas incidencias introducidas por el administrador. Se dispondrá también del botón "Editar" en la columna de opciones cuando el tipo de la incidencia lo permita. Como en cualquier operación de edición de la aplicación, se presentará un formulario como el de la creación pero cumplimentado con los datos actuales de la incidencia para su modificación.

C.3.7. Eliminar

En la columna de opciones de cada fila de ambas tablas se tendrá disponible en todo momento la opción de "Eliminar" incidencia. Al pulsar el botón correspondiente se presentará un diálogo de confirmación que deberemos aceptar para llevar a cabo la operación.

C.4. Tabla temporal e incidencias

La sección pública de la aplicación se compone de tres partes: el acceso o login, que se explica en la sección C.1; la presentación de una tabla temporal en que se muestran todos los servicios de la aplicación y sus incidencias; y la página de detalles de una incidencia, en la que se listan todos los mensajes publicados de la misma.

Tabla temporal

En esta tabla se muestra una fila por cada servicio presente en la aplicación. La primera celda, en la que se muestra el nombre del servicio, sirve también como enlace a la página del mismo si se proporcionó una dirección al crearlo. Las columnas centrales representan un conjunto de cinco días, en el que el día de hoy ocupa la columnacentral, es decir, engloba el período que va desde las 00:00h de hace dos días hasta las 23:59h de dentro de dos días. Por cada incidencia del servicio, sea del tipo que sea, se mostrará una barra horizontal, cuyas dimensiones serán equivalentes a la duración de la incidencia, y que se situará en la fila del servicio atendiendo a las fechas y horas de inicio y fin. El color de esta barra será aquel que se haya asociado al último mensaje de la incidencia. A su vez, la barra servirá de enlace a la página de información de la incidencia, en la que se mostrarán todos los mensajes existentes para la misma. En el pie de la tabla dispondremos de tres enlaces para desplazarnos en el tiempo, pudiendo retroceder o avanzar en libremente.

Mensajes de la incidencia

Al acceder a los detalles de una incidencia pulsando en la barra correspondiente de la tabla temporal se accederá a un página en la que se muestran, dispuestos en una tabla, los mensajes publicados para la incidencia junto con información sobre la hora y fecha de publicación de cada uno. Dispondremos también de un enlace a la sección de ayuda del servicio - si se proporcionó tal información al crearlo - en la parte inferior derecha de la página.

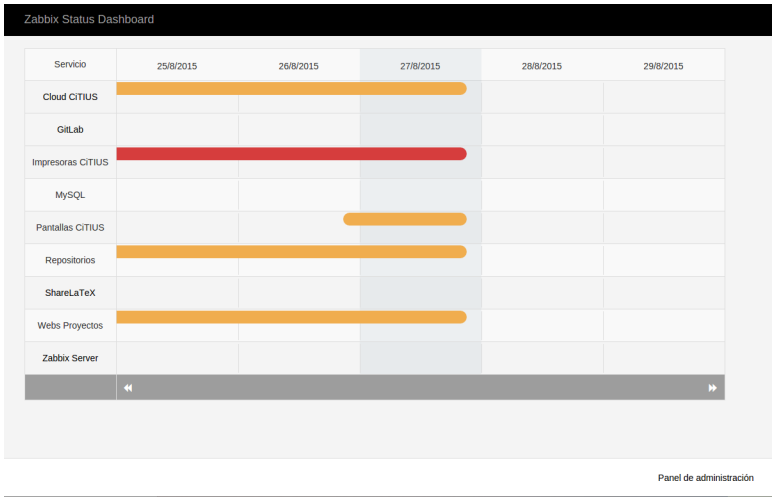


Figura C.6: Tabla temporal

Bibliografía

[1] Apps Status Dashboard. Accesible en *http://www.google.com/appsstatus#hl=es&v=status*.

[2] Ken Schwaber y Jeff Sutherland, "The Scrum Guide", Scrum Alliance, 2011. Disponible en *http://www.scrumguides.org/download.html*.

[3] Patrón Modelo-Vista-Controlador. Artículo de la wikipedia (*https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador*). Consultado el 10 de julio de 2015.

[4] Patrón Active Record. Artículo de la wikipedia (*https://es.wikipedia.org/wiki/Active_record*). Consultado el 20 de julio de 2015.

[5] jQuery. Artículo de la wikipedia (*https://es.wikipedia.org/wiki/JQuery*). Consultado el 20 de julio de 2015.

[6] Zabbix API. Sección de la documentación de Zabbix (*https://www.zabbix.com/documentation/2.2/manual/api*).

[7] Trigger.get . Método de la API de Zabbix (*https://www.zabbix.com/documentation/2.2/manual/api/reference/trigger/get*).

[8] Clase Trigger. Objeto de la API de Zabbix (*https://www.zabbix.com/documentation/2.2/manual/api/reference/trigger/object#trigger*).

[9] Project Management Institute, *A Guide to the Project Management Body of Knowledge*, pp. 193-215, 337-383, 5th edition, Project Management Institute, Inc. 14 Campus Boulevard, Newtown Square, Pennsylvania. 2013.

[10] Guía de Yii 2.0. Disponible en *http://www.yiiframework.com/doc-2.0/guide-README.html*.

[11] Mark Zafronov, Jeffrey Winesett, *Web Application Development with Yii 2 and PHP*, 3rd Edition, Packt Publishing Ltd., Livery Place, 35 Livery Street, Birmingham B3 2PB, UK.

[12] Jakob Nielsen. "Heuristic evaluation." Usability inspection methods 17, no. 1 (1994): 25-62.

[13] Guia salarial del sector TI en Galicia 2014-2015. Disponible en *http://www.vitaedigital.com/#Guia-Salarial-Sector-TI-Galicia-2014-2015*.

[14] Trigger en MySQL. Documentación de MySQL (*https://dev.mysql.com/doc/refman/5.0/en/create-trigger.html*)

[15] Bootstrap, disponible en *http://getbootstrap.com*.

[16] Definición de ScrumBut. Disponible en *https://www.scrum.org/ScrumBut*.

[17] Protocolo HTTPS. Artículo de la wikipedia (*https://en.wikipedia.org/wiki/HTTPS*). Consultado el 25 de julio.

[18] Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, pp.78-80, 3th edition, Addison-Wesley Professional, 2004.

[19] Estadísticas de uso de PHP en desarrollos web. Artículo disponible en *http://trends.builtwith.com/framework/PHP*

[20] Licencia MIT. Disponible en *http://opensource.org/licenses/MIT*.