Rexoubapp: plataforma de monitorización de servidores

Eduardo Salguero López
Grado en Ingeniería Informática

Albert Grau Perisé

13/06/2017

A Marta, a miña muller e compañeira.
Ás miñas pequerrechas.
A todos os que me ofreceron o seu valioso tempo para acadar a meta.
Grazas e grazas.

FICHA DEL TRABAJO FINAL

Título del trabajo:	Rexoubapp: Plataforma de monitorización de servidores
Nombre del autor:	Eduardo Salguero López
Nombre del consultor:	Albert Grau Perisé
Fecha de entrega (mm/aaaa):	06/2017
Área del Trabajo Final:	Java EE
Titulación:	*Grado en Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

"Rexoubapp" es una plataforma multiusuario, sencilla de usar, que posibilita monitorizar servidores físicos o virtuales basados en Debian GNU/Linux. La monitorización consiste en la obtención y la supervisión de un conjunto de las métricas del sistema y la comprobación del estado de los servicios indicados.

"Rexoubapp" permite indicar umbrales críticos de uso en las métricas supervisadas o cambios de estado en los servicios que deben ser registrados. Además posibilita añadir una serie de contactos que serán notificados, mediante correo electrónicos o mensajes de Slack, cuando dichos eventos ocurren.

La plataforma está formada por dos elementos:

- Un *backend* implementado sobre Spring Framework con Spring Boot que consta de:
  - Una API REST, que permite interactuar a través de HTTP con la plataforma.
  - Un *worker* encargado de la recolección y procesamiento de los datos.
  - Un bus de mensaje asíncrono utilizado para el registro de eventos de monitorización y notificación de los mismos.
- Un d*ashboard* simple implementado en JavaScript (React+Redux) que consume ciertas rutas de la API y permite visualizar la información más relevante de una cuenta.

Abstract (in English, 250 words or less):

"Rexoubapp" is a multi-user, easy-to-use monitoring platform build to monitor physical and virtual servers based on Debian GNU/Linux. Monitoring is to collect and supervise a set of system metrics and check the status of some services.

"Rexoubapp" allows to indicate critical usage thresholds in metrics or changes in service status that must be logged. In addition to this, it allows to add contacts that will be notified, by email or Slack messages, when such events occur.

The platform is composed of two elements:

- A Spring Framework based backend with Spring Boot consisting of:
    - The REST API which allows you to use the platform using HTTP.
    - A worker responsible to harvest and process servers' data.
    - An asynchronous message bus used for recording monitoring events and to notify them.
- A simple JavaScript dashboard build with React and Redux. "Rexoubapp dashboard" consumes the API and show most relevant account's information.

Palabras clave (entre 4 y 8):

Monitorización, Servidor, Java, API, Spring Boot, ReactJS, Javascript

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

A día de hoy cualquier empresa o proyecto medianamente serio necesita contar con una infraestructura de monitorización que permita por un lado prever incidencias y por otro conocer el aprovechamiento de los recursos informáticos.

Durante mi carrera como desarrollador he vivido diversas situaciones. Desde empresas con grandes recursos que cuentan con un equipo de administradores de sistemas y en donde los desarrolladores no se preocupan de lo que ocurre en los servidores hasta pequeñas *Startups* con recursos limitados centradas en el desarrollo de su producto y que delegan estas responsabilidades en servicios, más o menos caros, de terceros. Pasando también por pequeños proyectos con recursos mínimos, en los que directamente se elude esta responsabilidad.

Si exploramos el mercado encontramos muchas opciones. Soluciones como *Nagios o Icinga*. Estas son fabulosas y flexibles soluciones de software libre que requieren, por un lado bastantes recursos - en muchos casos de una infraestructura mayor de la que se desea monitorizar- y por el otro de una instalación y configuración que no resulta trivial. También nos encontramos con soluciones comerciales que siguiendo un modelo de SaaS como *Monitis*, no requieren de grandes conocimientos pero no resultan económicas, además son herramientas privadas poco flexibles y fuera de control.

Rexoubapp pretende ofrecer una solución intermedia. Rexoubapp es una plataforma multiusuario, simple y fácil de usar, que permite monitorizar servidores físicos o virtuales basados en Debian GNU/Linux. La monitorización consiste en la obtención y la supervisión de un conjunto de las métricas del sistema y la comprobación del estado de los servicios indicados.

Rexoubapp permite indicar umbrales críticos de uso en las métricas supervisadas o cambios de estado en los servicios que deben ser registrados. Además, es posible añadir una serie de contactos que serán notificados mediante correo electrónico o mensajes (a un canal o usuario) de la herramienta de comunicación en equipo Slack, en el momento en que dichos eventos tienen lugar.

1.2 Objetivos del Trabajo

El trabajo busca cumplir los siguientes objetivos:

- Desde una perspectiva de utilidad, generar una solución de monitorización que será usada en proyectos propios.

- Desde una perspectiva de futuro, desarrollar un producto funcional que sirva de punto de partida para un proyecto abierto.

- Desde un punto de vista didáctico:
  - Fomentar el aprendizaje sobre la tecnología J2EE y el framework Spring con Spring Boot mediante su uso.
  - Fomentar el aprendizaje de la librería JavaScript React mediante su uso.
  - Aplicación de principios de "*Clean Architecture*" [1] y DDD [2] en el desarrollo de un producto de software.

1.3 Enfoque y método seguido

Inicialmente para el desarrollo del proyecto se había planteado un modelo de ciclo de vida en cascada. Este, a pesar de ser bastante rígido, parecía adaptarse perfectamente al cumplimiento de los hitos predeterminados por el calendario.

Más adelante, y condicionado tanto por decisiones tomadas en la fase de bocetado inicial como por la curva de aprendizaje que implicaban las tecnologías seleccionadas, se replantea el enfoque y se decide que para la fase de implementación lo mejor es optar por una estrategia de desarrollo incremental más ágil y que permita contar cuanto antes con un producto entregable que permita llegar a la fecha de entrega con un proyecto funcional, aunque no esté completamente acabado.

El plan es apoyarse en un marco de desarrollo que implemente el marco que proporciona *Scrum* pero teniendo en cuenta que el proyecto es unipersonal y que el desarrollador acumula todos los roles. Para ello se prepara un *Backlog* con todos los requisitos del sistema que se se priorizan y estiman. En paralelo se fija un tamaño de sprint de 15 días, se considera que este tamaño permite adaptarse mejor en caso de haber errado con las estimaciones y más teniendo en cuenta que la entrega final tiene una fecha fija.

En cada Sprint se selecciona el trabajo a realizar de entre los ítems del *Backlog* y se ejecuta. El Sprint incluye tareas de análisis y refinamiento del diseño inicial, tareas relacionadas con el aprendizaje e investigación de cuestiones técnicas relacionadas con los trabajos concretos a realizar, implementación y pruebas. Al finalizar ese período, si hay tareas inacabadas, se devuelven al *Backlog* general y se planifica la siguiente iteración.

1.4 Planificación del Trabajo

Se programa una planificación con hitos directamente relacionada con las fechas clave de las fases del TFG:

Fecha	Actividad	Descripción
09/03/2017	PEC1: Plan de trabajo	Elaboración del plan de trabajo, detalle de requisitos y funcionalidades y planificación del proyecto.
11/04/2017	PEC2: Análisis y diseño	Estudio de la arquitectura, análisis general del proyecto, funcionalidades (diagrama de clases, diagrama de interrelación...), diseño de la interface.
30/05/2017	PEC3: Implementación	Implementación y codificación del proyecto (versión Beta).
13/06/2017	Memoria y presentación	Fecha de finalización del proyecto con la entrega de la memoria, presentación final, vídeo y código depurado (producto final).

Las diferentes tareas se planifican del siguiente modo:

Nombre	Fecha de inicio	Fecha de fin
Trabajo Fin de Grado	22/02/17	13/06/17
Planificación	22/02/17	09/03/17
Brainstorming y preparación de la propuesta	22/02/17	1/03/17
Estudio del estado del arte	2/03/17	3/03/17
Funcionalidades	6/03/17	6/03/17
Boceto arquitectónico	6/03/17	7/03/17
Calendarización	8/03/17	19/03/17
Análisis y diseño	8/03/17	11/04/17
Especificación de requisitos funcionales	8/03/17	10/03/17
Modelado del sistema y análisis del dominio	13/03/17	27/03/17
Casos de uso	13/03/17	15/03/17
Estructura de la información	15/03/17	18/03/17

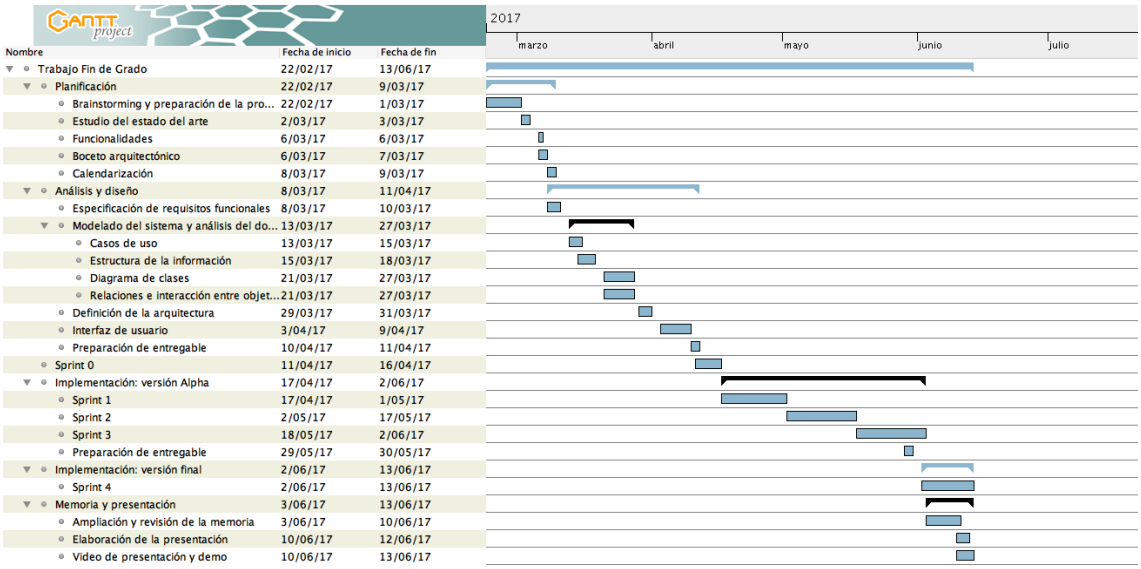
Diagrama de clases	21/03/17	27/03/17
Relaciones e interacción entre objetos	21/03/17	27/03/17
Definición de la arquitectura	29/03/17	31/03/17
Interfaz de usuario	3/04/17	09/04/17
Preparación de entregable	10/04/17	11/04/17
Sprint 0: Preparación del entorno de trabajo	11/04/17	16/04/17
Implementación: Sprints 1, 2 y 3	17/04/17	02/05/17
Preparación entrega PEC3	29/05/2017	30/05/17
Memoria y presentación	2/06/17	13/06/17
Ampliación y revisión de la memoria	2/06/17	09/06/17
Elaboración de la presentación	10/06/17	12/06/17
Video de presentación y demo	10/06/17	13/06/17

Como se ha comentado, la fase de implementación se ha realizado usando el *framework Scrum*. Se ha definido un duración de Sprint de 15 días, exceptuando el Sprint 0 con una duración de 5 días y un Sprint final que abarca los últimos 10 días de trabajo.

A continuación se enumeran y describen los sprints:

Sprint	Inicio	Fin	Descripción
Sprint 0	11/04	16/04	Setup del backlog y preparción del entorno de trabajo
Sprint 1	17/04	01/05	Spring Boot setup, autenticación, gestión de usuario y contactos.
Sprint 2	02/05	17/05	Gestión de servidor, monitorización, eventos de dominio y notificaciones.
Sprint 3	18/05	02/06	Dashboard, gestión de errores y nuevos requisitos derivados de la construcción del cliente. Preparación entregable PEC3.
Sprint 4	03/06	13/06	Corrección de bugs e incorporación de mejoras.

A continuación se presenta el diagrama de Gantt con las planificación del proyecto:



Nombre	Fecha de inicio	Fecha de fin
Trabajo Fin de Grado	22/02/17	13/06/17
Planificación	22/02/17	9/03/17
Brainstorming y preparación de la pro...	22/02/17	1/03/17
Estudio del estado del arte	2/03/17	3/03/17
Funcionalidades	6/03/17	6/03/17
Boceto arquitectónico	6/03/17	7/03/17
Calendarización	8/03/17	9/03/17
Análisis y diseño	8/03/17	11/04/17
Especificación de requisitos funcionales	8/03/17	10/03/17
Modelado del sistema y análisis del do...	13/03/17	27/03/17
Casos de uso	13/03/17	15/03/17
Estructura de la información	15/03/17	18/03/17
Diagrama de clases	21/03/17	27/03/17
Relaciones e interacción entre objet...	21/03/17	27/03/17
Definición de la arquitectura	29/03/17	31/03/17
Interfaz de usuario	3/04/17	9/04/17
Preparación de entregable	10/04/17	11/04/17
Sprint 0	11/04/17	16/04/17
Implementación: versión Alpha	17/04/17	2/06/17
Sprint 1	17/04/17	1/05/17
Sprint 2	2/05/17	17/05/17
Sprint 3	18/05/17	2/06/17
Preparación de entregable	29/05/17	30/05/17
Implementación: versión final	2/06/17	13/06/17
Sprint 4	2/06/17	13/06/17
Memoria y presentación	3/06/17	13/06/17
Ampliación y revisión de la memoria	3/06/17	10/06/17
Elaboración de la presentación	10/06/17	12/06/17
Video de presentación y demo	10/06/17	13/06/17

Ilustración 1: Diagrama de Gantt de la planificación del proyecto

1.5 Breve sumario de productos obtenidos

Una vez finalizado el proyecto se han obtenido:

- Entregables intermedios de las PECs.

- Una implementación estable y funcional de *Rexobapp* formado por:
  - Un repositorio Git con el *backend* escrito en Java denominado *rexoubapp*.
  - Un fichero JAR.
  - Un repositorio Git con el frontend denominado rexoubapp-dashboard.
  - Un fichero HTML, un fichero CSS, un fichero JavaScript y una serie de *assets* asociados.

- Especificación y documentación de la API REST:
  - Una documentación accesible mediante http expuesta por el *backend*.
  - Una especificación completa en formato JSON y YAML mediante *OpenAPI Specification* [3].

- Esta memoria.

- Una presentación en vídeo.

1.6 Breve descripción de los otros capítulos de la memoria

El capítulo 2 contiene una breve descripción de conceptos relacionados tanto con cuestiones técnicas o de implementación como con la propia plataforma que ayudará al lector a poder contextualizar los siguientes apartados.

Durante el capítulo 3 se describe el análisis funcional y los requisitos de la plataforma. Estos se completan con la descripción de la API que tiene lugar en el capítulo cuarto. El capítulo quinto contiene toda la información de diseño técnico de la solución.

Los capítulos 6 y 7 contienen las conclusiones y posibles líneas de trabajo. Le siguen el glosario de términos y las referencias bibliográficas, respectivamente, y para acabar un listado de anexos.

2. Definición de conceptos

2.1 Arquitectura hexagonal

La Arquitectura Hexagonal, dada a conocer por Alistair Cockburn [4] -y también conocida como arquitectura de puertos y adaptadores-, tiene como principal motivación separar nuestra aplicación en distintas capas con su propia responsabilidad. De esta manera consigue desacoplar capas de nuestra aplicación permitiendo que evolucionen de manera aislada. Además, tener el sistema separado por responsabilidades nos facilitará la reutilización.

Gracias a este desacoplamiento obtenemos también la ventaja de poder testear estas capas sin que intervengan otras externas (falseando el comportamiento con dobles de pruebas).

Esta arquitectura se suele representar con forma de hexágono, pero el número de lados no es lo que importa, sino lo que estos representan. Cada lado representa un puerto hacia dentro o fuera de la aplicación. Por ejemplo, un puerto puede ser el HTTP, y hacer peticiones a nuestra aplicación, otro puerto puede ser el SOAP y también hace peticiones a la aplicación. Otro puede ser un servidor de base de datos en donde persitir los datos de nuestro dominio.

La arquitectura hexagonal propone que nuestro dominio sea el núcleo de las capas y que este no se acople a nada externo. En lugar de hacer uso explícito y mediante el principio de inversión de dependencias nos acoplamos a contratos (interfaces o puertos) y no a implementaciones concretas.

A grandes rasgos y a alto nivel, lo que se propone es que nuestro núcleo sea visto como una API con un contratos bien especificados. Definiendo puertos o puntos de entrada e interfaces (adaptadores) para que otros módulos (UI, BBDD, Test) puedan implementarlas y comunicarse con la capa de negocio sin que esta deba saber el origen de la conexión.

Esto es lo llamado puertos y adaptadores, que podrían ser definidos de la siguiente manera:

- Puerto: definición de una interfaz pública.
- Adaptador: especialización de un puerto para un contexto concreto.

2.2 DDD

Un dominio de software es el modelo conceptual de todos los temas y cuestiones relacionados con el problema de software a resolver.

De forma muy simplificada se puede definir el diseño guiado por el dominio, *domain-driven design* [2] (DDD) en inglés, como un enfoque para el desarrollo de software que propone una profunda conexión entre la implementación y los conceptos del modelo y núcleo del negocio. DDD es un conjunto de prácticas, técnicas, herramientas y enfoques para dar respuesta a las necesidades

complejas en el desarrollo de software, orientadas al core del negocio, su evolución y sus objetivos.

Sin entrar en más detalles, se puede decir el objetivo principal de DDD es aislar el dominio del resto de la aplicación. Enlazando esto con la Arquitectura Hexagonal, y dado que esta es una arquitectura que fomenta que nuestro dominio sea el núcleo de todas las capas y que no se acople a nada externo, ambas ideas encajan como dos piezas de un puzzle. Podríamos decir, por lo tanto, que DDD se basa en la Arquitectura Hexagonal como pilar central en términos de arquitectura.

El motivo de haber definido y relacionado ambos conceptos es que DDD con Arquitectura Hexagonal es el enfoque escogido para implementar el core de la plataforma. Con ciertos matices podemos afirmar que el dominio está en el centro, define el lenguaje ubicuo y es independiente de framework.

2.3 Desacoplamiento y cola de mensajes

El acoplamiento hace referencia al nivel de dependencia entre entidades. o mejor dicho, en el modelo de comunicación que se establece entre ellas. Existen dos tipos de acoplamiento, espacial y temporal:

- Acoplamiento espacial: implica que el remitente debe (o necesita) conocer la identidad del receptor o receptores. Es decir, el remitente debe conocer a quien envía los mensajes y viceversa.
- Acoplamiento temporal: implica que el remitente y el receptor o receptores deben tener una línea temporal común, o dicho de otro modo, todos ellos deben existir al mismo tiempo para poder realizar la comunicación.

Podemos definir “Comunicación indirecta” como la comunicación entre entidades en un sistema distribuido a través de un intermediario sin un acoplamiento directo entre el remitente y el receptores o receptores.

Por definición, en *Rexoubapp* existe un acoplamiento entre el subsistema encargado de recolectar las métricas y el encargado de procesar los eventos. Uno se encarga realizar las mediciones y mediante mensajes le indica al otro que registre ciertas circunstancias y envíe notificaciones si procede. Este planteamiento representa, al menos, dos problemas:

- Uno de escalabilidad. Dado que no permite añadir más agentes de un tipo o del otro de forma independiente según las necesidades. Y además, limita su separación espacial.
- Otro de dependencia con el exterior. Dado que ambos componentes usan sistemas externos, cualquier circunstancia en esos agentes repercute directamente en ambos componentes.

Por ejemplo, ¿qué sucede si el servidor de correo está sobrecargado y se tarda 1 minuto en entregar un mail? ¿Y si se decide dejar de enviar correos? ¿Y si hay demasiados servidores que recolectar y queremos añadir nuevos recolectores?

Ambos problemas se solucionan contando con un mecanismo de indirección. En este caso mediante el uso de una cola de mensaje, implementada sobre RabbitMQ, se logra desacoplar completamente ambos componentes del sistema. De este modo se consigue que el recolector y el emisor de mensajes llevan vidas separadas y que la comunicación entre ellos sea asíncrona. Esto permitirá, de forma sencilla, añadir recolectores o mensajeros de forma independiente. Permite evitar que problemas al enviar correos o mensajes de Slack influyan en la recolección del siguiente servidor o incluso que recolectores y mensajeros vivan en instancias diferentes de la aplicación.

2.4 Worker

Un *worker* es un proceso en segundo plano que se encarga de ejecutar una tarea de larga ocupación sin necesidad de interrumpir la respuesta del usuario.

En *Rexoubapp* existe un tipo de *worker* recolector que de forma periódica se encarga de acceder a los servidores monitorizados para obtener recoger los datos de las métricas y estado de los servicios.

2.5 SSH

SSH o *Secure Socket Shell* es un protocolo de red que permite el acceso de modo seguro a una computadora remota.

Rexoubapp, o mejor dicho, su *worker* recolector, utiliza SSH con autenticación mediante criptografía de clave pública para acceder a los servidores y recoger los datos de forma sencilla mediante la ejecución de comandos Linux.

2.6 Base de datos relacional

Una base de datos relacional es un tipo de base de datos que se basa en la recopilación de elementos de datos con relaciones predefinidas entre ellos. Estos elementos de datos se organizan como un conjunto de tablas formadas por columnas y filas. Cada fila de la tabla representa una recopilación de valores relacionados de un objeto o entidad. Las tablas se utilizan para guardar información sobre los objetos que se van a representar en la base de datos. Cada columna de una tabla guarda un determinado tipo de datos y un campo almacena el valor real de un atributo.

Por definición, una base datos relacional posee un esquema de datos fijo que la define. Por esto y las capacidades en cuanto a restricciones que esto ofrece, es la solución seleccionada para representar las principales entidades de nuestro dominio.

2.7 Base de datos NoSQL

No todos los datos de la plataforma conviven bien con el mundo relacional. Es decir, no representan estructuras fijas ni se encuentran normalizados. Este es el caso de los informes de monitorización resultado de una recolección y que

son representados mediante documentos JSON. Para almacenar este tipo de datos se ha decidido optar por utilizar MongoDB como solución de persistencia.

2.8 Docker y Docker Compose

Por lo que hemos visto hasta ahora, Rexoubapp necesita una serie de aplicaciones para poder funcionar:

- Una base de datos MySQL
- Una base de datos de documentos MongoDB
- Una solución de colas de mensajes RabbitMQ
- Un servidor de aplicaciones Apache Tomcat para correr la aplicación (integrado con Spring Boot)

Docker es una plataforma de software que permite empaquetar software en unidades llamadas contenedores que incluyen todo lo necesario para que el software se ejecute.

En Rexoubapp se ha utilizado Docker para aprovisionar cada uno de los contenedores (servidor MySQL, servidor MongoDB y servidor RabbitMQ), lo cual facilita el despliegue y posibilita contar con el entorno adecuado de forma sencilla y portable.

Docker Compose es una utilidad y un fichero YAML que permite especificar una serie de contenedores, sus propiedades y relacionarlos entre sí. Dado que, como se ha indicado, la infraestructura incluye múltiples contenedores interconectados, se ha utilizado esta solución para orquestar y facilitar el despliegue multi-contenedor que la plataforma requiere.

2.9 Spring Framework y Spring boot

Spring es un framework de aplicaciones con soporte para inyección de dependencias, gestor de transacciones, aplicaciones web, acceso a datos, mensajería y *testing*, entre otros, que en cierto modo se ha convertido en un estándar “de facto”.

Spring Boot es un proyecto del ecosistema Spring creado para simplificar la creación de aplicaciones y servicios Spring facilitando la configuración y el despliegue que además provee un servidor de aplicaciones, Tomcat en nuestro caso, embebido.

Por todo esto, su robustez, extensa documentación y las ventajas que proporcionan al desarrollador, Spring y Spring Boot, son la solución Java seleccionada para la implementación del *backend* de la plataforma.

2.10 React y Redux

React es una librería JavaScript lanzada por Facebook encargada del renderizado de las vistas de una aplicación web. *Redux* es una librería que implementa el patrón *Flux*.

Flux nace también en el seno de Facebook como respuesta a los problemas que patrones con comunicación bidireccional como MVC o MVVM provocan en aplicaciones web con cierto grado de complejidad. *Flux* es una arquitectura que facilita la gestión y flujo de datos de una aplicación web que, de forma básica, propone que el camino de los datos tenga un único sentido. De este modo todo el flujo acaba llegando a un sitio que almacena el estado y que se encarga de actualizar las vistas, que en cierto modo están suscritas a los cambios que en este tienen lugar.

Como se ha indicado, *Rexoubapp* no solo es una aplicación escrita en Java. Es también una interfaz web simple que proporciona un cuadro de mando en el que visualizar los datos más importantes que la plataforma ofrece. Esta interfaz web es un cliente JavaScript puro que consume la API REST expuesta por el *backend*. *React* es la elección tecnológica para construir con *Redux* la aplicación web.

2.11 Rexoubapp

Rexoubapp es una plataforma multiusuario, simple y fácil de usar, que permite monitorizar servidores físicos o virtuales basados en Debian GNU/Linux. La monitorización consiste en la obtención y la supervisión de un conjunto de las métricas del sistema y la comprobación del estado de los servicios indicados.

Rexoubapp permite la posibilidad de indicar umbrales críticos de uso en las métricas supervisadas o cambios de estado en los servicios que deben ser registrados. Además posibilita añadir una serie de contactos que serán notificados, mediante correo electrónico o mensajes de Slack, cuando dichos eventos ocurren.

Conceptos:

- Usuario: es un agente que puede registrarse en la plataforma y realizar las operaciones que esta proporciona. Decimos agente y no persona dado que mediante la API REST, otro sistema, *script* o programa, puede ser usuario de *Rexoubapp*.
- Servidor: es la parte centrar del la plataforma. Un servidor es un sistema basado en Debian, accesible a través de Internet y en el que se pueden ejecutar comandos de forma remota, a través de SSH, para realizar las operaciones de monitorización. Los servidores son propiedad de un usuario concreto.
- Contacto: un contacto es una entidad asociada a un usuario que será notificado, mediante correo y/o el envío de mensajes mediante un

"Incoming WebHook" de Slack, de los eventos de los servidores del usuario relacionado.

- Evento: un evento es cualquier circunstancia excepcional que se haya detectado en relación con los servidores a monitorizar.
- Notificación: una notificación es el hecho de comunicar, si así ha sido configurado, un evento a los contactos del usuario propietario del servidor afectado.
- Monitor: un monitor es una entidad abstracta que define una regla de seguimiento de una métrica o un proceso. Un monitor por ser una definición puede compartirse entre servidores. Existen dos tipos de monitores:
  - *Harvesters*:

    Es un monitor encargado de la recolección y seguimiento de las méticas del servidor.

    Un *harvester* requiere de un valor de alerta y de peligro. Estos serán utilizados como referencia a la hora de comprobar si la métrica ha alcanzado valores críticos que deben ser registrados y notificados.

    Inicialmente el conjunto de métricas recolectables es:

    - Uso de disco.
    - Uso de memoria RAM.
    - Carga media del sistema
  - Observers:

    Es un monitor encargado de vigilar el estado de la ejecución de un proceso. Inicialmente se limita a servicios del sistema.

    Un *observer* supervisa y registra la inactividad o el cambio de estado del elemento observado. Ambas circunstancias pueden ser notificadas.
- Monitor de servidor: es una realización de un monitor en un servidor concreto. Dicho de otro modo, es la configuración de un monitor aplicado en un servidor.
- Monitorización: es la acción de observar, recolectar y procesar los datos obtenidos por los diversos monitores de servidores y registrar los eventos que hayan tenido lugar y enviar las notificaciones oportunas.
- Informe: es el documento en formato JSON con todos los datos obtenidos en una monitorización de un servidor.

3. Análisis funcional y requisitos

3.1 Requisitos de producto

Un requisito es una característica observable de un sistema que expresa una necesidad o restricción que un *stakeholder* tiene sobre él.

Los requisitos funcionales hacen referencia a la funcionalidad que debe proporcionar el sistema y los datos que tiene que conocer y guardar.

Después de una sesión de brainstorming se detectaron esta serie de requisitos funcionales candidatos, que acabaron convirtiéndose en requisitos del sistema. Estos se enumeran, en forma de historias de usuario, a continuación:

- Como usuario quiero poder registrarme en la plataforma.
- Como usuario registrado quiero poder darme de baja de la plataforma.
- Como usuario quiero poder identificarme en la plataforma.
- Como usuario identificado quiero poder añadir servidores indicando su IP y una etiqueta.
- Como usuario identificado quiero poder modificar la IP y la etiqueta de un servidor.
- Como usuario identificado al añadir un servidor quiero obtener la clave pública con la que el sistema accederá a mi servidor para poder instalarla.
- Como usuario identificado quiero poder eliminar servidores de la plataforma.
- Como usuario identificado quiero poder seleccionar las métricas recolectables de un servidor.
- Como usuario identificado quiero poder seleccionar los valores y los umbrales de cambio sobre los cuales recibir notificaciones.
- Como usuario identificado quiero poder indicar los servicios a monitorizar y conocer el cambio de su estado mediante una notificación.
- Como usuario registrado quiero que cuando un servidor no responda se me notifique.
- Como usuario registrado quiero poder recibir notificaciones mediante correo electrónico.
- Como usuario registrado quiero poder recibir notificaciones mediante mensajes de Slack.
- Como usuario identificado quiero poder obtener un registro de los eventos producidos en mis servidores (listado de notificaciones).
- Como usuario identificado quiero poder obtener un histórico de cada una de las métricas obtenidas de un servidor.
- Como usuario identificado quiero poder obtener los últimos valores de todas métricas monitorizadas de mis servidores o de uno en concreto.
- Como usuario identificado quiero poder obtener el *uptime* de mis servidores o de uno en concreto.

- Como usuario identificado quiero poder obtener un listado del último estado registrado de los servicios monitorizados de mis servidores o de uno en concreto.
- Como usuario identificado quiero disponer de una interfaz web que me permita ver de forma sencilla los datos básicos de mi cuenta y el detalle de cada uno de mis servidores.

También nos encontramos con una serie de requisitos no funcionales que indican calidades esperadas del sistema:

- De presentación:
  - La interfaz web debe ser simple de usar.
  - Los colores deben ser los adecuados.
- De usabilidad:
  - Los menús de interfaz web deben de tener como máximo dos niveles de profundidad.
- De seguridad:
  - Las contraseñas deben de almacenarse encriptadas.
  - Un usuario únicamente tendrá acceso a aquella información para la que esté autorizado
- Operacionales y de entorno:
  - La interfaz web debe adaptarse al tamaño del dispositivo por lo que debe poder visualizarse desde un teléfono móvil.
  - La API debe devolver datos en formato JSON.
- Culturales:
  - La interfaz web debe de estar exclusivamente en inglés.
  - Las fechas se mostrarán siempre con UTC como zona horaria.

Todo el conjunto de requisitos definen necesidades o restricciones sobre el producto. Es decir, que una vez desarrollado los satisfaga.

En ningún momento se definen requisitos de proceso más alla de las reglas definidas por el consultor para el TFG.

3.2 Casos de uso

Un caso de uso recoge el contrato entre el sistema y los *stakeholders* mediante la descripción del comportamiento observable del mismo. Dado que un actor es alguien que tiene comportamiento, y no solo intereses en el sistema, sabemos que de todos esos *stakeholders* o interesados, solo una parte de ellos son actores.

A raíz de las historias de usuario especificadas en el plan de trabajo, se han identificado los siguientes actores:

- Usuario no registrado: sera n aquellas entidades que pueden registrarse en el sistema para poder interactuar con él posteriormente.
- Usuario registrado: sera n aquellas entidades que se han registrado en el sistema creando una cuenta y se autentican en él para realizar acciones.
- Usuario identificado: son usuarios registrado que se han autenticado y pueden realizar operaciones en el sistema.

Aquí se presenta el diagrama de casos de uso:

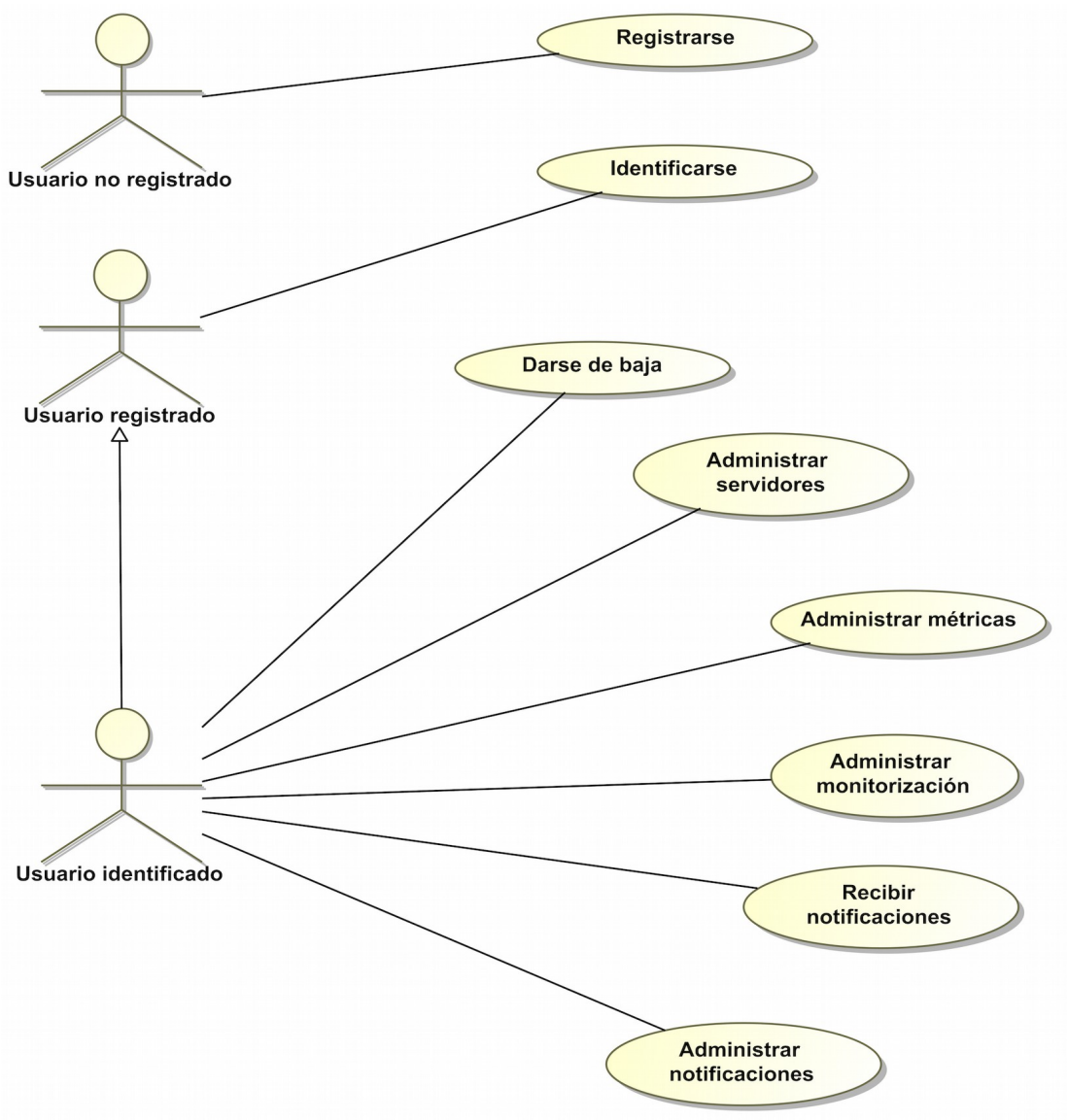


Ilustración 2: Diagrama UML de casos de uso

A continuación se detalla la especificación del caso de uso de registro en el sistema.

Código: CU-01

Caso de uso: Registrarse en el sistema

Actor principal: Usuario no registrado

Ámbito: Plataforma de monitorización

Nivel de objetivo: Usuario

Stakeholders e intereses:

Usuario no registrado: el usuario quiere poder darse de alta en la plataforma para poder usarla.

Precondición: La cuenta de correo que identifica al usuario no puede existir previamente.

Garantías en caso de éxito: Se crea la cuenta de usuario.

Escenario principal en caso de éxito:

1. El usuario indica el email y contraseña que desea registrar.

2. El sistema realiza las operaciones necesarias para validar los datos y proceder al registro de usuario.

3. El sistema responde con el código de http correspondiente (201)

Extensiones:

1.a) El usuario indica un correo ya registrado.

1.a.1) El sistema le indica que está registrado denegando la operación con el código http correspondiente (400) y un mensaje informativo.

1.b) El usuario proporciona datos inválidos.

1.b.1) El sistema le indica que no se puede realizar el registro denegando la operación con el código http correspondiente (400) y un mensaje informativo.

Dado que la interacción de cualquiera de los *stakeholders* con el sistema se realiza a través de una API REST, en lugar de continuar detallado los casos de uso, se ha optado por describir y documentar la API de forma completa.

4. La API

4.1 Descripción

Rexoubador es visto desde fuera como una simple API REST que permite interactuar a través de HTTP con la plataforma.

La API está compuesta por un total de 39 rutas. Para su definición se ha seleccionado la versión 2.0 de la OpenAPI Specification[3] –formalmente conocido como la especificación Swagger– , el cual es considerado el estándar de la industria para definición de APIs REST.

Dado que la especificación completa es bastante extensa se indicará a continuación un resumen con las rutas y detalles principales, dejando el documento completo en formato YAML en el *Apéndice I.*

4.2 Seguridad

Por definición y dado que se ha decidido construir una API sin información de estado se ha seleccionado para la implementación del sistema de autenticación el estándar JSON Web Token (JWT). JWT es un estándar abierto (RFC 7519)[5] que nos permite la transmisión de forma segura y confiable gracias a su firma digital.

La API expone una ruta pública necesaria para el registro de usuarios y otra que permite autenticación y obtención del *token* que deberá enviarse mediante *Auth Basic.* El resto de rutas requieren el envío de un token válido para ser ejecutadas.

4.3 Respuestas y errores

La API consume y produce objetos JSON y, en algunos casos, respuestas vacías con el código HTTP correspondiente.

En el caso de los errores todos las excepciones controladas generarán una respuesta con código de error y un formato como el que sigue:

```
{
  "code": "A numeric string with the related error code",
  "message": "A descriptive message with related info"
}
```

4.4 Rutas

Podemos clasificar las rutas de la API en varios grupos que se exponen a continuación:

Autenticación

Son el conjunto de rutas relacionadas, el registro y la obtención de token de aunteticación.

Son las únicas rutas que no requiren autenticación.

Url	Método	Descripción
/v1/auth/register		** Ruta sin autenticación*
	POST	Registra un nuevo usuario en la plataforma
/v1/auth/login		** Ruta sin autenticación*
	POST	Sirve para verificar la identidad de un usuario en el sistema y devuelve el token necesario para poder realizar las acciones que requieren privilegios.

Gestión de usuarios

Son el conjunto de rutas relacionadas con las acciones que se pueden realizar sobre las entidades usuario.

Url	Método	Descripción
/v1/users/self		
	GET	Obtiene la información del usuario autenticado
	PUT	Actualiza parte de la información del usuario autenticado
	DELETE	Elimina el usuario y lo da de baja de la plataforma

Obtención de eventos

Rutas relacionadas con las acciones que se pueden realizar sobre las entidades evento. En este caso se entiende como evento cualquier monitorización que haya provocado el envío de una notificación.

Url	Método	Descripción
/v1/events		
	GET	Obtiene los últimos eventos relacionados con el usuario autenticado. Permite indicar el número de días.

Gestión de contactos

Un contacto es el destinatario de una notificación, realizada mediante Slack y/o correo electrónico, de cualquier evento producido en el sistema.

Url	Método	Descripción
/v1/contacts		
	GET	Obtiene los contactos del usuario autenticado.
	POST	Añade un contacto del usuario autenticado
/v1/contacts/*{contactId}*		
	GET	Obtiene el contacto con el ID indicado como parámetro
	PUT	Actualiza el contacto con el ID indicado como parámetro
	DELETE	Elimina el contacto con el ID indicado como parámetro

Uptime

El *uptime* define el tiempo de actividad de un servidor en segundos.

Url	Método	Descripción
/v1/uptime		
	GET	Obtiene el uptime de todos los servidores del usuario autenticado.

Monitores

Gestión de observadores

Actualmente la API solo soporta el subtipo “SERVICE”. Estas son las rutas para la gestión de monitores “*observadores*”:

Url	Método	Descripción
/v1/observers		
	GET	Obtiene los observadores definidos por el usuario autenticado.
	POST	Añade una definición de observador asociada al usuario autenticado.
/v1/observers/{observerId}		
	GET	Obtiene el observador con el ID indicado como parámetro
	PUT	Actualiza el observador con el ID indicado como parámetro
	DELETE	Elimina el observador con el ID indicado como parámetro

Gestión de recolectores

Permiten la gestión de monitores de tipo “*recolectores*”. Los subtipos soportados son:

- Uso de disco: DISK_USAGE
- Uso de memoria RAM: MEMORY_USAGE
- Carga media del sistema: LOAD

A continuación las rutas:

Url	Método	Descripción
/v1/harvesters		
	GET	Obtiene los recolectores definidos por el usuario autenticado.
	POST	Añade una definición de recolector asociado al usuario autenticado.
/v1/harvesters/{harvestersId}		
	GET	Obtiene el recolector con el ID indicado como parámetro
	PUT	Actualiza el recolector con el ID indicado como parámetro

	DELETE	Elimina el recolector con el ID indicado como parámetro

Gestión de servidores

Como se ha indicado, el servidor es el elemento central de la plataforma. Un usuario puede gestionar un número indefinido de servidores sobre los que ejecutar los procesos de monitorización (observadores y recolectores) previamente definidos.

Url	Método	Descripción
/v1/servers		
	GET	Obtiene los servidores del usuario autenticado.
	POST	Añade un servidor asociado al usuario autenticado.
/v1/servers/{serverId}		
	GET	Obtiene los datos servidor con el ID indicado como parámetro
	PUT	Actualiza ciertos datos del servidor con el ID indicado como parámetro
	DELETE	Elimina el servidor y todas las monitorizaciones asociadas
Observadores del servidor		
/v1/servers/{serverId}/observers		
	GET	Obtiene los *observadores* configurados en el servidor con el ID indicado como parámetro
	POST	Añade la monitorización de un *observador* al servidor con el ID indicado como parámetro
/v1/servers/{serverId}/observers/{serverObserverId}		
	GET	Obtiene los datos del *observador* indicado.
	DELETE	Elimina la monitorización del *observador* indicado
Métricas		
/v1/servers/{serverId}/harvesters		
	GET	Obtiene los *recolectores* configurados en el servidor con el ID indicado como parámetro
	POST	Añade la monitorización de un *recolector* al servidor con el ID indicado como parámetro.
/v1/servers/{serverId}/harvesters/{serverHarvestersId}		
	GET	Obtiene los datos del recolector indicado.
	DELETE	Elimina el recolector
Uptime		
/v1/servers/{serverId}/uptime		
	GET	Obtiene el *uptime* del servidor con el ID indicado como parámetro.

Obtención de informes

La obtención de informes es otra pieza fundamental de cualquier herramienta de monitorización. Permite obtener un histórico de las monitorizaciones realizadas sobre los servidores.

Url	Método	Descripción
/v1/reports		
	GET	Obtiene los informes de todas las monitorizaciones configuradas sobre los servidores del usuario autenticado.
/v1/reports/{serverId}		
	GET	Obtiene los informes de todas las monitorizaciones configuradas sobre el servidor con el ID indicado.

Ping

La API define una ruta básica pensada para comprobar si la API está accesibe.

Url	Método	Descripción
/v1/ping		
	GET	Obtiene el “pong” de respuesta.

5. Diseño técnico

En el presente capítulo, relacionado con la fase de diseño, se presentan los principales artefactos producidos con el objetivo de modelar los requerimientos anteriores.

5.1 Diagrama de clases

A continuación se incluye el diagrama estático de clases que representa las entidades del dominio, sus atributos y las relaciones entre los objetos:

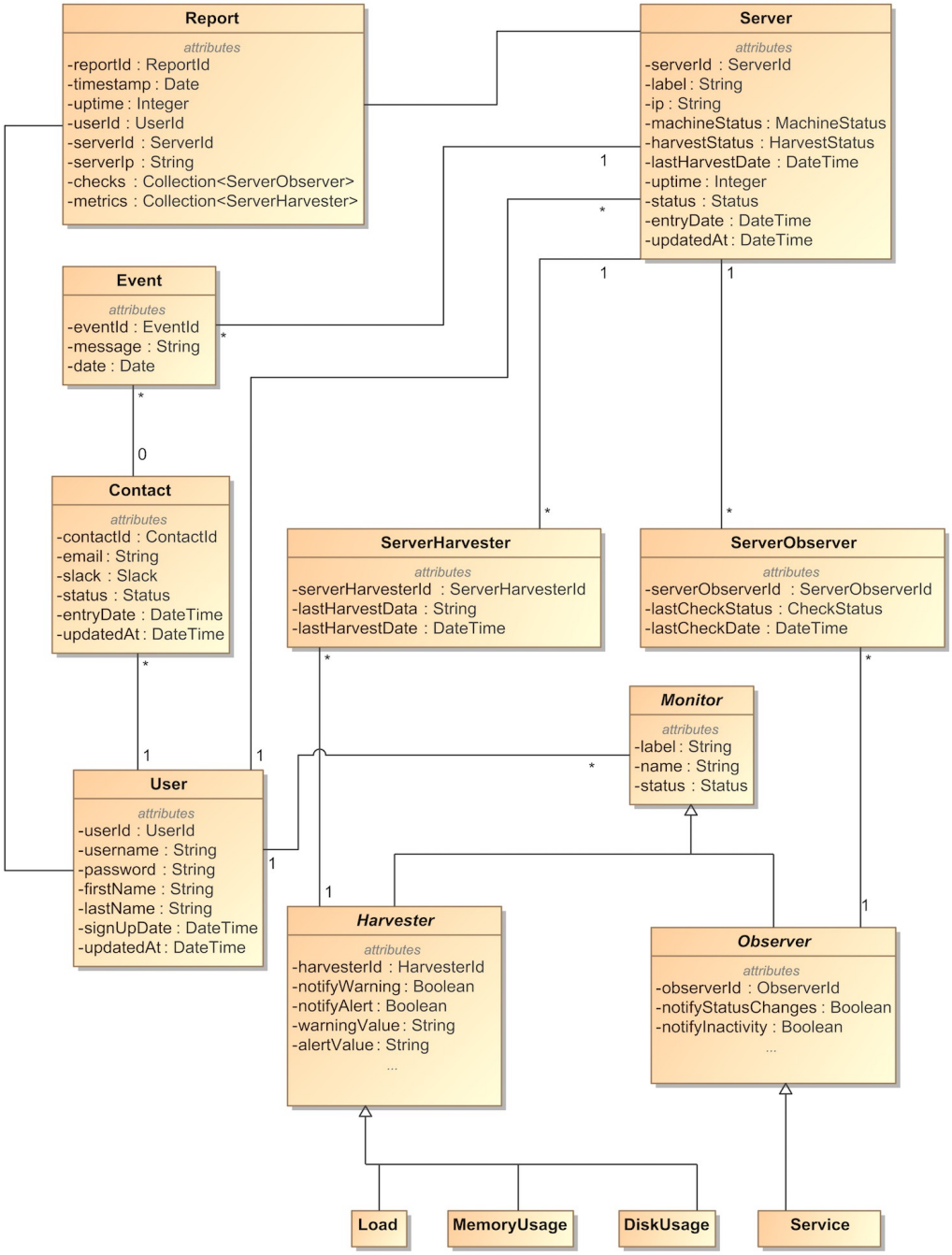


Ilustración 3: Diagrama de clases del Dominio

Durante la fase diseño se ha decidido definir una serie de enumeraciones con importancia dentro del dominio:

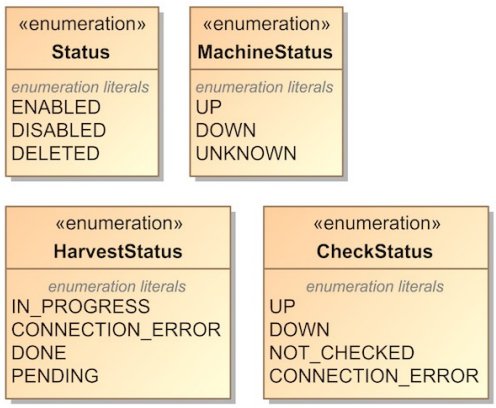


Ilustración 4: Enumerados relevantes en el Dominio

También se ha decidido definir una serie de tipos de datos propios importantes y que por lo tanto debían ser modelados.

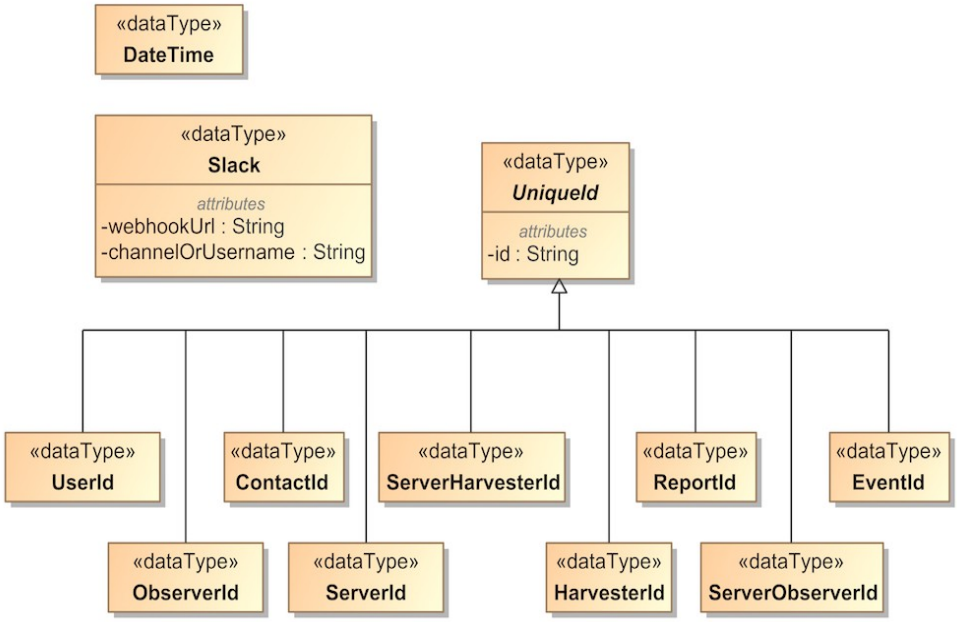


Ilustración 5: Tipos de datos propios

5.2 Diagrama relacional de BBDD

A continuación se indica el diagrama entidad-relación con el modelo de datos estático del sistema:

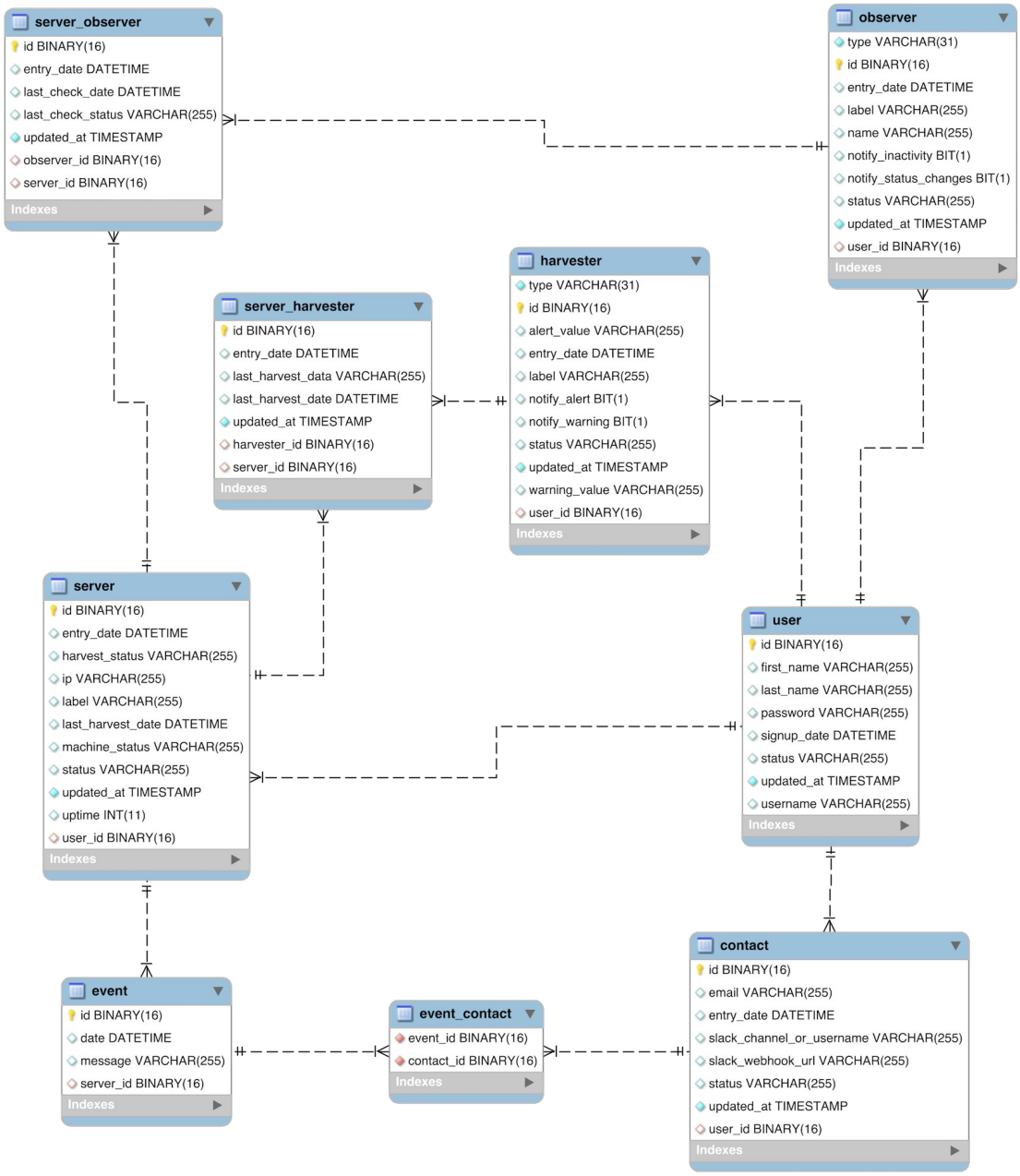


Ilustración 6: Diagrama ER de Base de Datos

De forma resumida la información almacenada en cada tabla es:

Tabla	Descripción
contact	Registra los contactos de un usuario.
event	Registra los eventos ocurridos en el sistema.
event_contact	Relaciona un evento con todos los contactos relacionados.
harvester	Registra una definición de un recolector.

observer	Almacena una definión de un observador
server	Almancena las entidades servidor
server_harvester	Representa la asociación de un recolector a un servidor con la información de la última recolección.
server_observer	Representa la asociación de un observador a un servidor con la información de la última observación.
user	Sirve para persistir entidades usuario.

Cabe indicar que, a mayores de estos datos estructurados, la plataforma debe almacenar información sin un esquema predefinido utilizando una solución NoSQL. Es el caso de los informes.

Un informe se refiere a los datos obtenidos en un proceso de monitorización para cada uno de los servidores en un momento concreto. Estas entidades serán almacenadas en una colección de documentos JSON representados con una estructura como la que sigue:


```
{
  "id": "e8eda4a4-366d-4f33-8fb3-f52273c1852b",
  "timestamp": 1497047360393,
  "server": {
    "ip": "188.226.189.35",
    "id": "c2b46bc3-a32c-cb86-e280-a04ec399e280"
  },
  "uptime": 3250733,
  "metrics": [
    {
      "data": {
        "mountPoints": [
          {
            "total": 20030,
            "percentageOfUse": 28,
            "used": 5248,
            "free": 13752,
            "type": "ext4",
            "filesystem": "/dev/vda1",
            "mountedOn": "/"
          }
        ]
      },
      "id": "4d42c3b1-c3b0-c5a1-c5a0-4575c2af3a08",
      "type": "DISK_USAGE"
    },
    {
      "data": {
        "last": 0.08,
        "last5mins": 0.08,
        "last15mins": 0.03
      },
      "id": "57c38d24-c3b2-e280-98c5-92420ee2809e",
      "type": "LOAD"
    },
    {
      "data": {
        "total": 500388,
```

```
        "used": 472024,
        "free": 28364
      },
      "id": "c384c392-c388-76c2-b43c-4cc3bbc2a035",
      "type": "MEMORY_USAGE"
    }
  ],
  "checks": {
    "services": [
      {
        "name": "apache2",
        "id": "c2acc2b3-c38f-1921-464d-c394e280a1c3",
        "status": "UP"
      }
    ]
  }
}
```

5.3 Proceso de monitorización

Definimos monitorización como el conjunto de acciones (realizadas en diferentes momentos) que permiten ejecutar los monitores asociados a cada un de los servidores *y* obtener las métricas, almacenar la información recolectada y enviar las notificaciones oportunas a los contactos configurados.

Para una mejor comprensión de la secuencia de acciones y procesos se añade el siguiente diagrama de secuencia iniciado con la adición de un servidor:

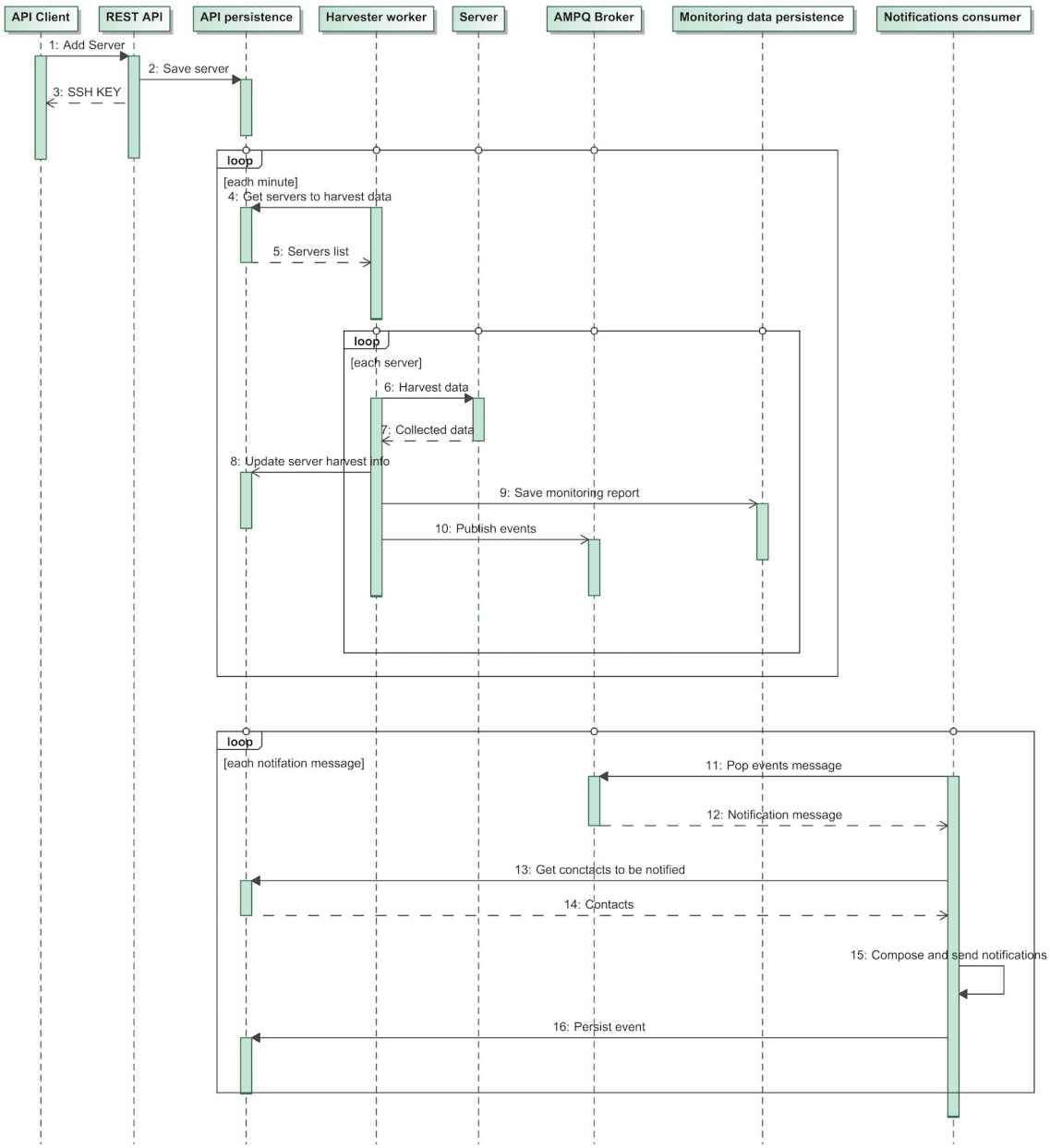


Ilustración 7: Proceso de monitorización

La pieza fundamental del proceso es un worker, que por definición corre en segundo plano, con una frecuencia de minuto y ejecuta los siguientes pasos:

1. Obtiene el listado de servidores pendientes de recolectar.
2. Una vez ha obtenido el listado de servidores y para cada uno de ellos:
    1. Inicia el proceso de recolección puro recogiendo datos a través de la ejecución remota de comandos usando el protocolo SSH.
    2. Trata los datos obtenidos para ser persistidos a través de operaciones del dominio.
    3. Realizan las comprobaciones relacionadas con las alertas de los monitores y emite los mensajes oportunos.
    4. Dichos mensajes, en forma de eventos, son publicados en una cola de mensajes que son procesados por los manejadores adecuados. Estos, de forma asíncrona, crean nuevos objetos de tipo eventos que son persistidos y envía las notificaciones oportunas a cada uno de los contactos asociados al usuario propietario del servidor monitorizado.

5.4 Arquitectura del sistema

A continuación se incluye un diagrama de alto nivel que representa la arquitectura del sistema y sus relaciones con el exterior:

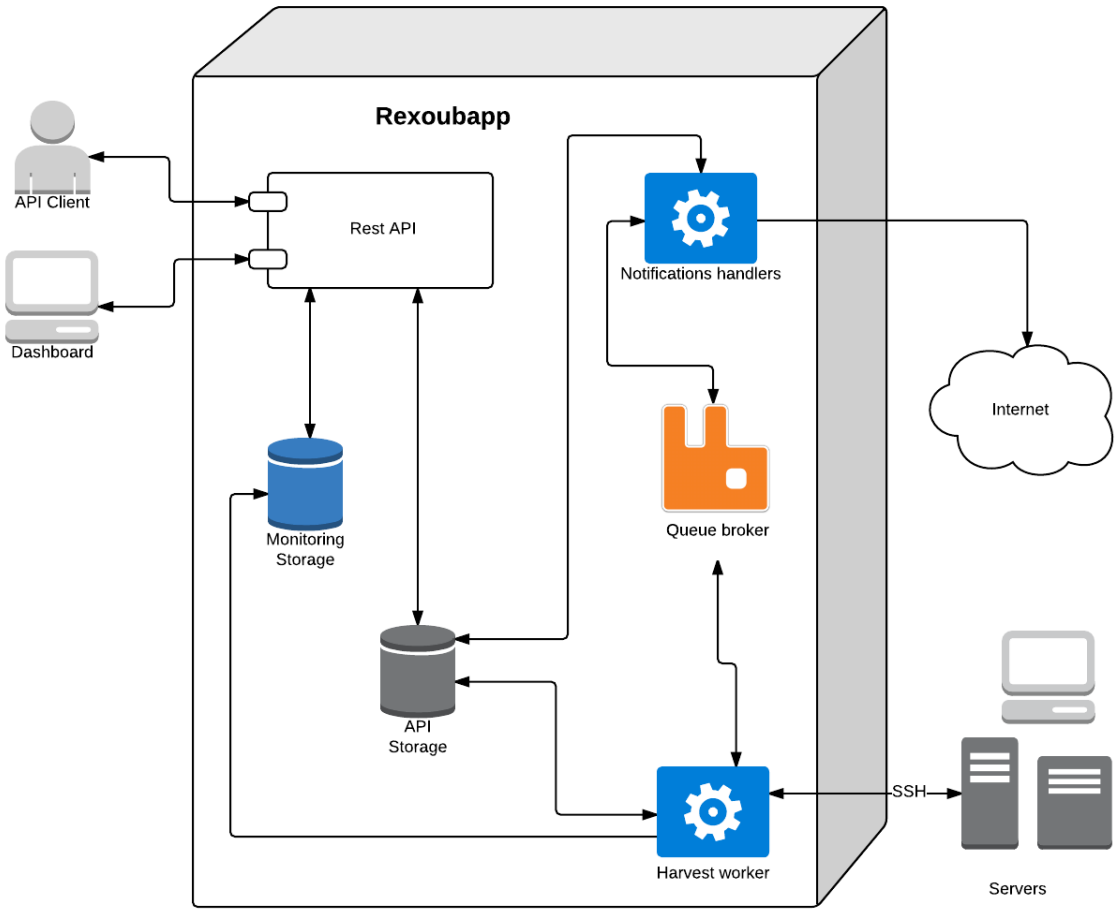


Ilustración 8: Arquitectura del sistema

Rexoubapp posee un estilo arquitectónico heterogéneo. Implementa un sistema cliente-servidor en donde la parte servidor se compone de diferentes componentes organizados en capas; además ciertos componentes se comunican mediante eventos realizando invocación implícita.

El cliente

Rexoubapp dashboard es un cliente puro JavaScript que consume la API Rest expuesta por la parte servidor.

La aplicación está escrita intentando seguir el estándar ES6 y utiliza Webpack para empaquetar los componentes lo cual permite, junto co con Babel, transpilar ES6 a ES5 y hacer la aplicación compatible con los principales navegadores del mercado.

Para la construcción de los diversos componentes se ha utlizado ReactJS generando un layout básico con un esquema formado por una cabecera, un cuerpo y un pie. La cabecera cambia según el estado y lo mismo ocurre con el

cuerpo, que además es el lugar en donde se renderizan los componentes dependientes de la ruta. Para esto último se ha utilizado React Router v4. Por otra parte y para la renderización de los gráficos se ha utilizado Chart.js

Como se ha indicado anteriormente la aplicación utiliza Redux, el cual utiliza el patrón Flux para encargarse, en cierta manera, de desacoplar el estado global de la aplicación web de la parte visual, es decir los componentes.

De forma esquematizada el flujo de los datos es el siguiente:

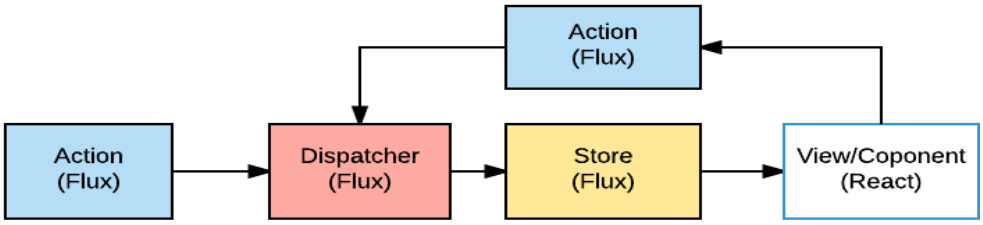


Ilustración 9: Esquema Redux

Las ***vistas*** o componentes son las partes de la interfaz. La ***store*** representa en cierto modo al modelo de la aplicación cuyos datos solo pueden ser modificados a través de ***dispatchers*****.** Estos no son más que mediadores entre las acciones y el store. Un ***action*** por su parte no es más que un objeto JavaScript que encapsula una intención de hacer algo y que puede llevar datos asociados.

Cuando el usuario realiza una acción en una vista mediante un evento, envía una acción al dispatcher que propaga los cambios al Store, el cual actualiza su estado y notifica a las vistas de ese cambio.

En el caso de *Rexoubapp dashboard* la comunicación con el *backend* se realiza mediante llamadas AJAX realizadas a través de un *middelware,* o intermediario, situado entre el *action* y el *dispatcher* y que por lo tanto queda completamente integrado en el esquema anterior.

En el Anexo I se pueden consultar las diferentes pantallas de la aplicación.

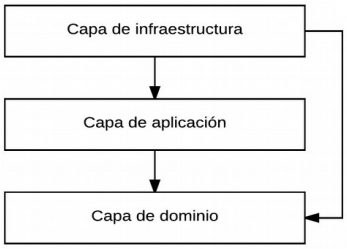
El servidor

Está formado por diferentes componentes organizados en capas siguiendo los principios e intenciones de la Arquitectura Hexagona,l con el dominio como centro y respetando el principio de inversión de dependencias.

Dicho principio sostiene que los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones. Las abstracciones no deben depender de detalles, los detalles deben depender de las abstracciones. Gracias a esto podemos hacer que el código, que es el núcleo de nuestra aplicación, no dependa de los detalles de implementación, como pueden ser el framework, la base de datos… Todos estos aspectos se especificarán mediante interfaces, y el núcleo no tendrá que conocer cuál es la implementación real para funcionar.

En la siguiente figura se puede ver la arquitectura en donde la flecha indica el sentido de la dependencia:

Ilustración 10: Arquitectura de capas e inversión de dependencias



- La capa de infraestructura:

  Es la capa superior que provee los servicios de bajo nivel que dependen de las interfaces definidas en los componentes de las capas subyacentes. Ente otros contiene las implementaciones concretas de los repositorios de acceso a datos, los controladores expuestos mediante la API REST, la implementación del recolector de métricas, etc.

- La capa de aplicación:

  Contiene la lógica de aplicación. Está orientada a los casos de uso de usuario y es quien recibe los parámetros del exterior. Es un cliente directo de la capa de dominio.

- La capa de dominio:

  Contiene la lógica de negocio definida mediante servicios y el modelado de las entidades del mismo. Aloja las interfaces necesarias de acceso a datos, define los procesos y las reglas de negocio de manera independiente a la tecnología y publica los eventos de dominio.

A continuación se presenta un diagrama para el caso concreto de *Rexoubapp:*

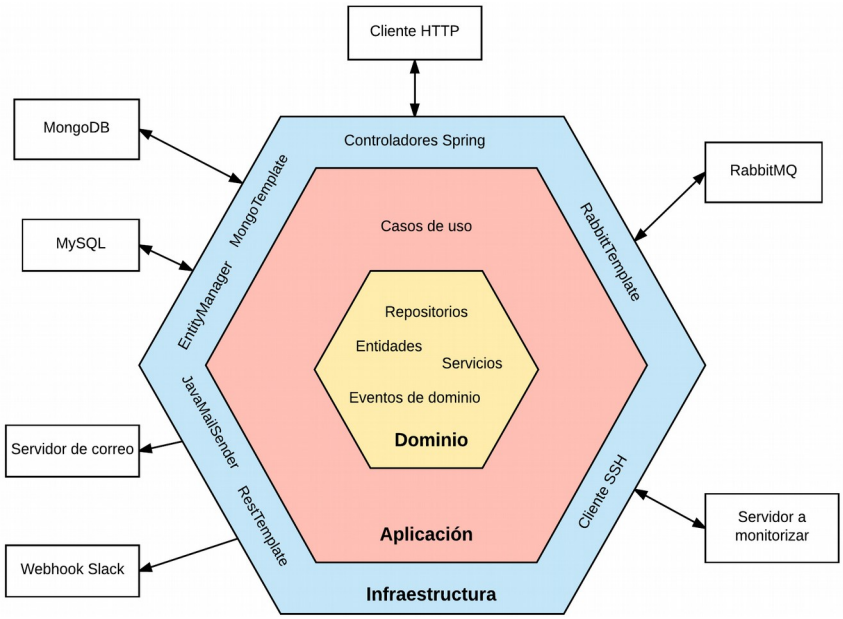


Ilustración 11: Arquitectura hexagonal en Rexoubapp

6. Conclusiones

Todo proyecto, entendido este como una secuencia de actividades que se desarrolla durante un tiempo para obtener un resultado único, tiende a requerir de una base, tanto histórica como actual, de conocimientos teóricos o prácticos, de esfuerzo, trabajo y motivación. Todo esto, además de generar un producto, proporciona una maravillosa oportunidad para incrementar las capacidades y habilidades de quien en él se involucre.

Desde antes de empezar tuve claro que mis objetivos personales en relación con el TFG iban más allá de la superación del mismo. He entendido el desarrollo del trabajo como la oportunidad de crear un producto útil, al menos para mí y mi entorno, cuya construcción fuese posible dentro de los límites temporales del TFG, así como de vehículo para explorar nuevas tecnologías desde un punto de vista práctico, utilizando Spring para construir algo desde cero explorando el *framework* e intentando adquirir la mayor cantidad de conocimientos posibles sin perder de vista el objetivo marcado por el proyecto en sí.

Durante todo el proceso he afianzado mi creencia de que un mismo problema puede solucionarse de múltiples maneras, que el lenguaje o el *framework* son el medio y no el fin o, dicho de otro modo, que estos son detalles de implementación. Lo importante es conocer lo que se quiere solucionar, conceptualizar tanto el problema como la solución, extraer las conceptos relevantes y las relaciones entre ellos. Después el proceso consiste simplemente en implementar y orquestar las diferentes partes.

Para la ejecución de este TFG y por la elección de *framework* la curva de aprendizaje que este conlleva ha requerido una gran inversión de tiempo. Aunque si bien ha sido algo satisfactorio he tenido que aprender muchas cosas sobre la marcha, cometer errores y tomar decisiones que simplificaran el dibujo mental inicial -aunque no las funcionalidad del producto- al verme obligado a restar horas de ejecución con cargo al aprendizaje.

Por otra parte he corroborado la idea de que “estimar es timar”, que planificar a 3 meses vista, sobre todo con tecnologías desconocidas de por medio, es imposible. Solo un planteamiento ágil que busque un desarrollo iterativo e incremental puede asegurar que pueda entregarse un software funcional (aunque no completo) desde fases iniciales y que esta es la medida real de progreso y no un calendario (irreal) de fechas. Que la sobreingeniería en el diseño puede ser un error cuando no se dispone de tiempo infinito (o al menos muy elevado), y grandes conocimientos de las herramientas, y que saber adaptase al cambio es la clave de la supervivencia de un proyecto.

Por todo esto, y después de una fase inicial con un planteamiento basado en un ciclo de vida en cascada, se ha decidido replantear la fase de ejecución con un planteamiento *“agile”* que ha permito contar con la flexibilidad necesaria para la consecución del proyecto a pesar de haber contado con múltiples contratiempos.

Sobre Spring y Spring Boot he de decir que han sobrepasado mis expectativas. El framework es completo y robusto y junto con un buen gestor de dependencias y un IDE como IntelliJ ha convertido mi personal incursión en el ecosistema Java en un bonito paseo. Considero su elección un acierto y debo concluir que su uso me ha mostrado otra cara del mundo Java más atractiva y moderna que, al menos, la idea que inicialmente tenía sobre ello.

En cuanto a React, y del mismo modo que la Arquitectura Hexagonal me ha permitido organizar el *backend*, han hecho que el desarrollo en *frontend* esté bastante bien componentizado, que sea entendible y que junto al patrón Flux (Redux) no se haya convertido en una maraña de código JavaScript difícil de mantener. Además ha sido un buen complemento a la API JSON que expone el *backend* Java.

Finalmente, y teniendo en cuenta que el resultado es mejorable y que, como suele ser habitual, una vez acabado lo ideal sería poder volver a empezar de cero con todo lo aprendido en la mochila, me considero satisfecho y contento con el resultado final.

7. Líneas de trabajo futuro

Uno de los objetivos del proyecto ha sido desarrollar un producto funcional que sirva de punto de partida para un proyecto abierto. Teniendo en cuenta esto, se pueden diferenciar dos lineas principales de trabajo futuro:

- Una orientada a la apertura del mismo que pasaría por:
  - Inclusión de test automatizados.
  - Mejoras en la documentación.
  - Apertura de los repositorios a la comunidad.
  - Creación de una página de presentación del proyecto.

- Otra que incluye mejoras a nivel funcional fáciles de implementar tales como:
  - Rotación de las métricas para poseer valores agregados y no solo valores por minuto que hacen que el volumen de datos a almacenar sea inmanejable a corto plazo.
  - Inclusión de recolectores de métricas de uso de red.
  - Inclusión de observadores de procesos y no solo servicios.
  - Inclusión de observadores que permitan comprobar el estado de servicios HTTP y tiempos de respuesta de los mismos.

A mayores sería interesante desacoplar cuanto antes las entidades del dominio con la capa de persistencia moviendo el *mapping* de las mismas de anotaciones a ficheros XML residentes, como debe ser, en la capa de infraestructura.

8. Glosario

API: una API (Application Programming Interface en inglés) es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas de la misma manera en que la interfaz de usuario facilita la interacción humano-software.

API HTTP: es una API accesible de forma remota usando el protocolo HTTP.

REST: *Representational State Transfer* o Transferencia de Estado Representacional. Es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

JSON: es un formato ligero de intercambio de datos simple de leer y escribir para humanos, y simple de interpretar y generar para las máquinas. Está basado en un subconjunto del Lenguaje de Programación JavaScript. JSON es un formato de texto que es completamente independiente del lenguaje.

JWT: es un standard que define un método compacto, autónomo y seguro para transmitir información entre 2 sistemas mediante un objeto JSON.

Frontend: es la parte del software que interactúa con él o los usuarios.

Backend: comprende los componentes que procesan la salida o entrada del *frontend*.

IDE: un entorno de desarrollo integrado (en inglés Integrated Development Environment), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

Servicio: es un tipo de proceso ejecutado en segundo plano.

Scrum: es un marco de trabajo por el cual las personas pueden acometer problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente.

Backlog: es el conjunto de todos los requisitos de proyecto, el cual contiene descripciones genéricas de funcionalidades deseables

Sprint: un sprint debe entenderse como un subconjunto de requerimientos, extraídos del product backlog, que deben ser ejecutados durante un periodo de tiempo determinado y definido a priori.

YAML: es un formato de serialización de datos legible por humanos.

Stakeholder: es cualquier agente (humano u otro sistema) que tiene algún interés en el sistema.

9. Bibliografía

[1] ALISTAIR COCKBURN. Hexagonal architecture. [en linea]. *http://alistair.cockburn.us/Hexagonal+architecture* [fecha de consulta: 20 de abril de 2017]

[2] UNCLE BOB. The Clean Architecture. [en linea]. *https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html* [fecha de consulta: 20 de abril de 2017]

[3] OPEN API SPECIFICATIONS. OpenAPI Specification. [en linea]. *https://www.openapis.org/* [fecha de consulta: 10 de marzo de 2017]

[4] INTERNET ENGINEERING TASK FORCE (IETF). JSON Web Token, RFC 7519. [en linea]. *https://tools.ietf.org/html/rfc7519* [fecha de consulta: 11 de marzo de 2017]

Enlaces de interés:

- Icinga - Open Source Monitoring [https://www.icinga.com]

- Nagios - The Industry Standard In IT Infrastructure Monitoring [https://www.nagios.org/]

- Monitis: Web Performance Monitoring Tools [http://www.monitis.com/es]

- Spring [https://spring.io/]

- React - A JavaScript library for building user interfaces [https://facebook.github.io/react/]

- Redux [http://redux.js.org/]

- Docker [https://www.docker.com/]

- RabbitMQ [https://www.rabbitmq.com/]

10. Anexos

Apéndice I: Pantallas de *Rexoubapp dashboard*

1. Bienvenida y login

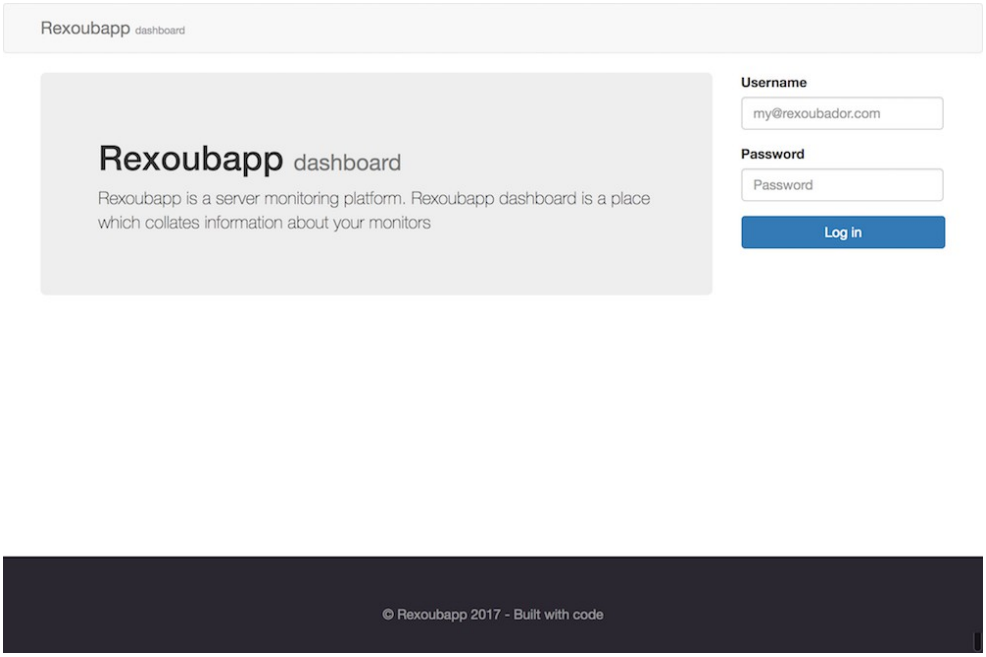


Ilustración 12: Rexoubapp dashboard: pantalla de bienvenida

2. Pantalla principal de usuario autenticado

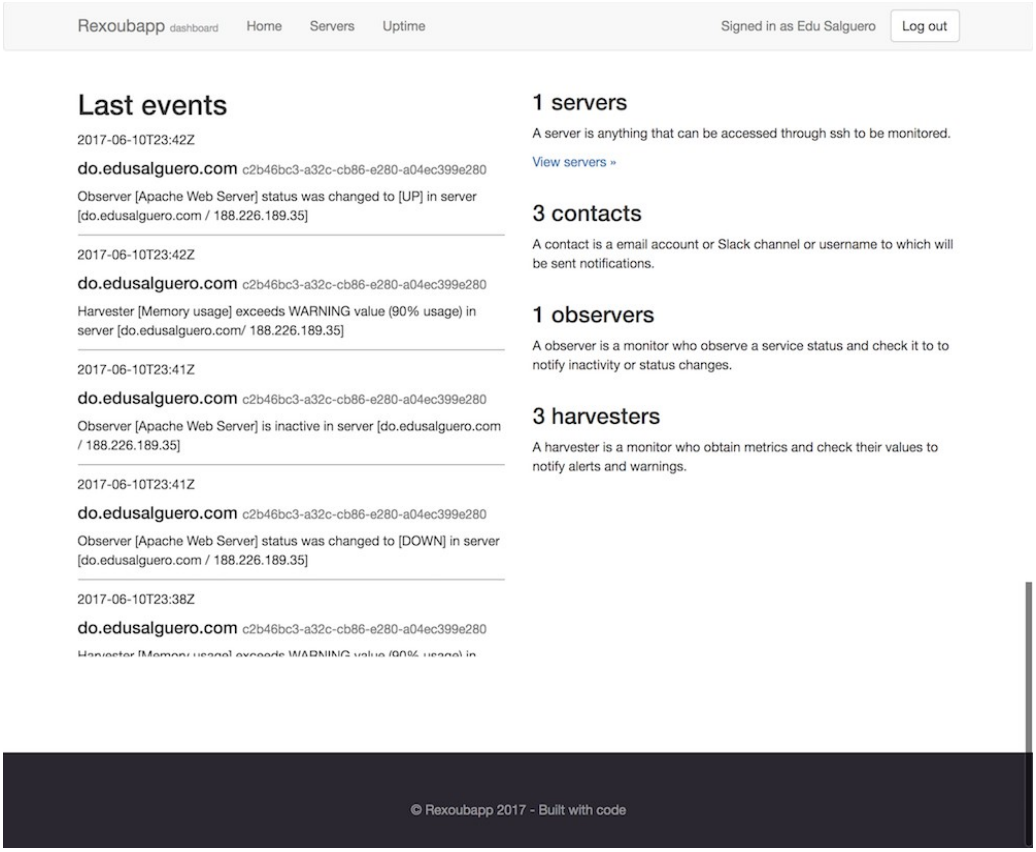


Ilustración 13: Rexoubapp dashboard: pantalla inicial de usuario autenticado

3. Listado de servidores

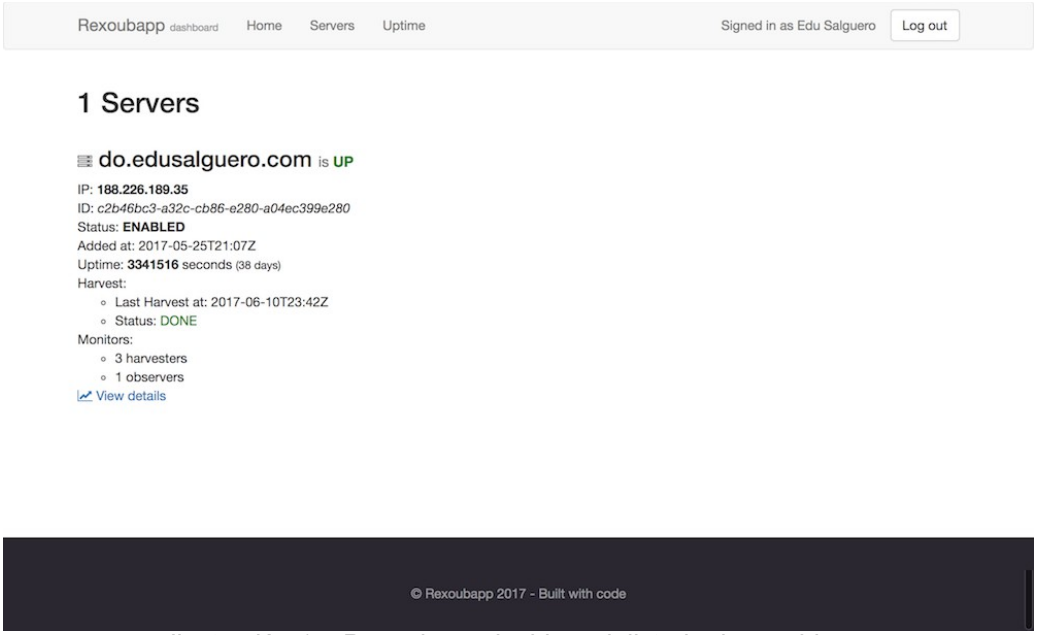


Ilustración 14: Rexoubapp dashboard: listado de servidores

4. Uptime de servidores

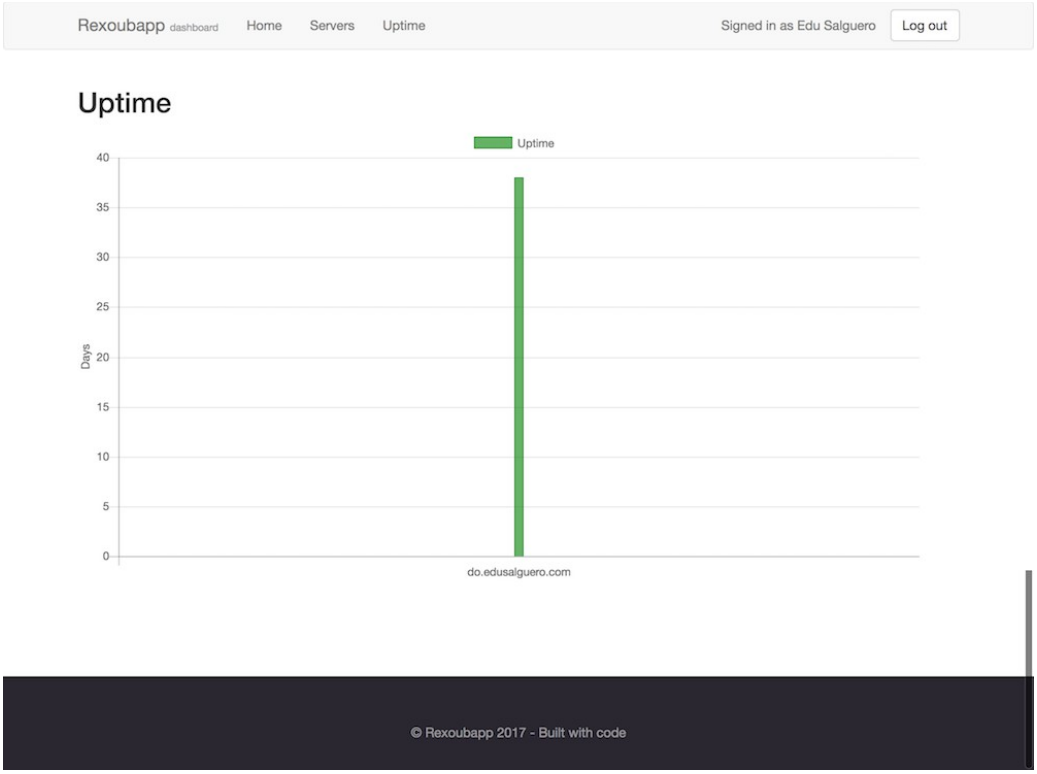


Ilustración 15: Rexoubapp dashboard: uptime de servidores

5. Detalle de servidor

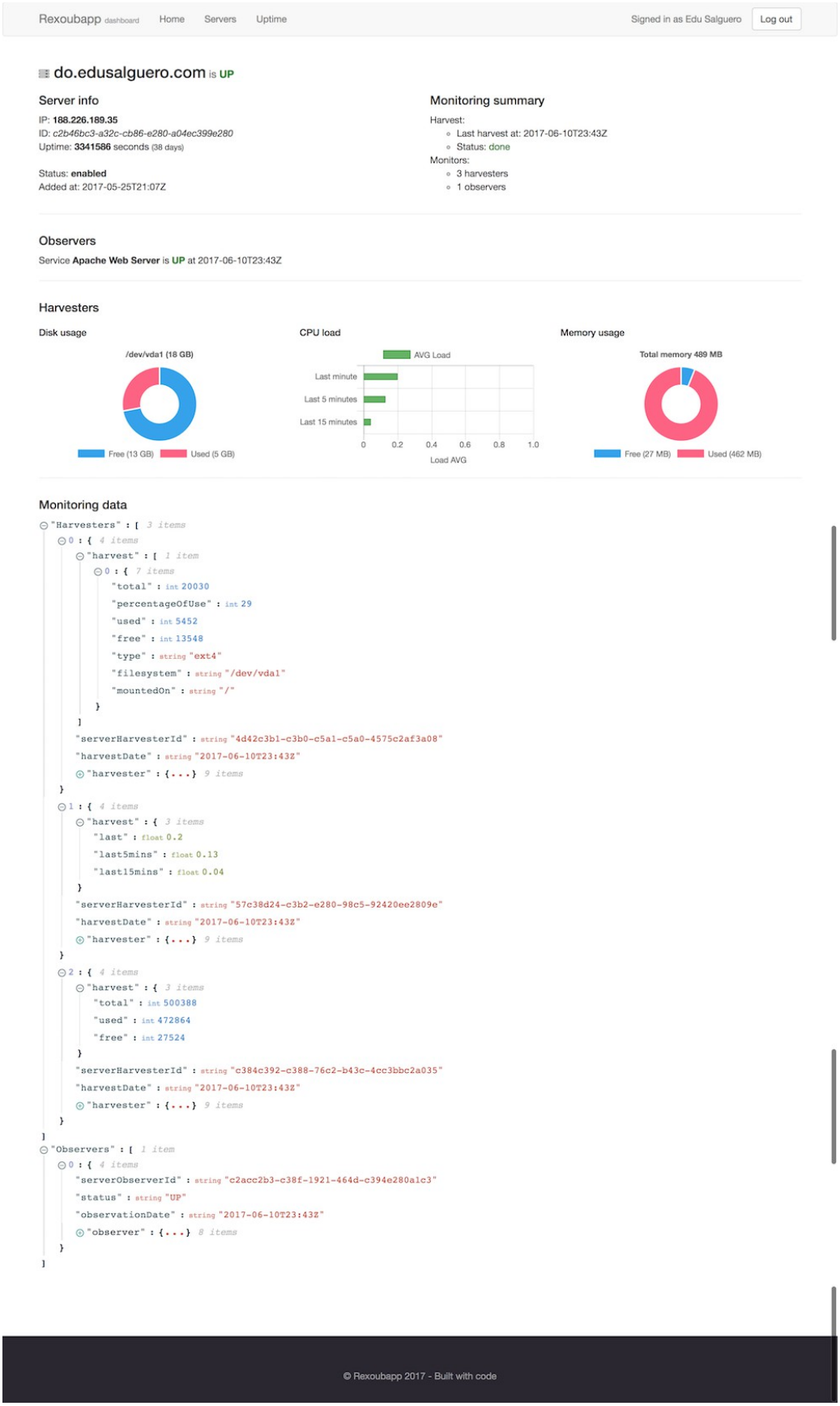
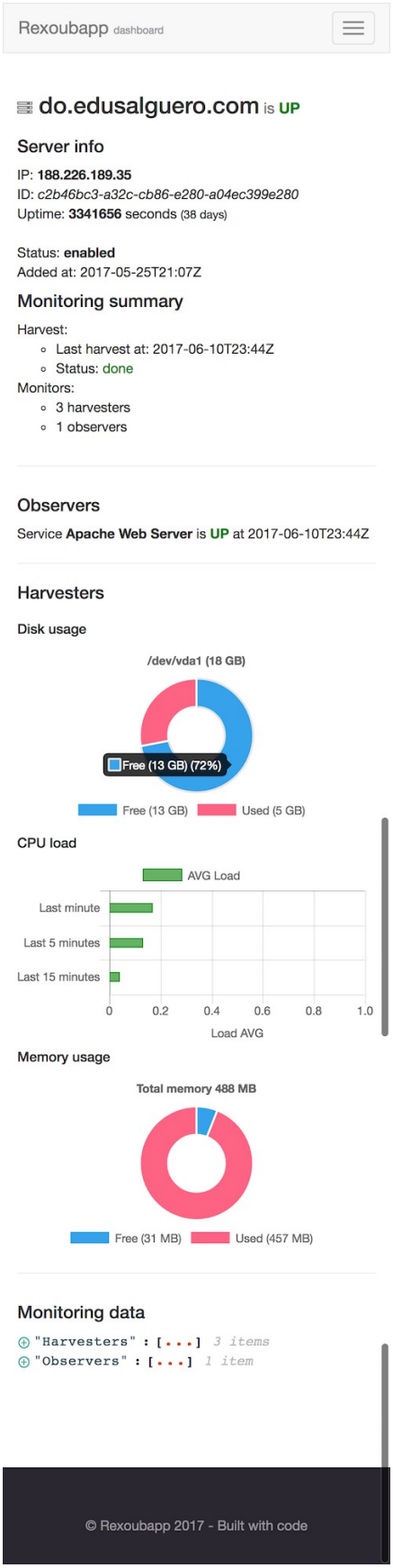


Ilustración 16: Rexoubapp dashboard: detalle de servidor

6. Detalle de servidor en versión móvil



Apéndice II: Especificación de la API mediante *OpenAPI Specification*

Para su mejor lectura se a utilizado la versión en YAML en lugar de JSON.

```yaml
Swagger: '2.0'
# Metadatos de la definición de la API

info:
  description: REST API for Rexoubapp monitoring platform
  version: '1.0'
  title: Rexoubapp REST API
  termsOfService: Terms of service
  contact:
    name: Edu Salguero
    url: 'http://edusalguero.com'
    email: edusalguero@gmail.com
  license:
    name: Apache License Version 2.0
    url: 'https://www.apache.org/licenses/LICENSE-2.0'
host: 'localhost:8080'
basePath: /
consumes:
  - application/json
produces:
  - application/json
tags:
  - name: auth-controller
    description: Login and register
  - name: server-controller
    description: Authenticated user' servers management operations
  - name: contact-controller
    description: Authenticated user contacts management operations
  - name: harvester-controller
    description: Authenticated user harvesters management operations
  - name: event-controller
    description: Authenticated user's events
  - name: ping-controller
    description: Ping operation
  - name: server-harvester-controller
    description: Server harvesters management operations
  - name: uptime-controller
    description: Servers uptime list
  - name: report-controller
    description: Authenticated user reports queryfier
  - name: server-observer-controller
    description: Server observers management operations
  - name: user-controller
    description: Authenticated user management operations
  - name: observer-controller
    description: Authenticated user observers management operations

# Rutas
paths:

# Rutas de autenticación y registro
  /v1/auth/login:
    post:
      tags:
```

```
      - auth-controller
    summary: login
    operationId: loginUsingPOST
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: username
        in: query
        description: username
        required: true
        type: string
      - name: password
        in: query
        description: password
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/LoginResponse'
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

/v1/auth/register:
  post:
    tags:
      - auth-controller
    summary: register
    operationId: registerUsingPOST
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: username
        in: query
        description: username
        required: true
        type: string
      - name: password
        in: query
        description: password
        required: true
        type: string
      - name: firstName
        in: query
        description: firstName
        required: true
        type: string
      - name: lastName
        in: query
```

```yaml
          description: lastName
          required: true
          type: string
      responses:
        '200':
          description: OK
          schema:
            $ref: '#/definitions/UserId'
        '201':
          description: Created
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found

# Rutas de gestión de contactos
  /v1/contacts:
    get:
      tags:
        - contact-controller
      summary: list
      operationId: listUsingGET
      consumes:
        - application/json
      produces:
        - application/json
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/ContactResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
    post:
      tags:
        - contact-controller
      summary: add
      operationId: addUsingPOST
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: email
          in: query
          description: email
          required: false
          type: string
        - name: slackWebhookUrl
          in: query
          description: slackWebhookUrl
          required: false
```

```
        type: string
      - name: slackChannelOrUsername
        in: query
        description: slackChannelOrUsername
        required: false
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ContactId'
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

/v1/contacts/{contactId}:
  get:
    tags:
      - contact-controller
    summary: view
    operationId: viewUsingGET
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: contactId
        in: path
        description: contactId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ContactResponse'
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found
  put:
    tags:
      - contact-controller
    summary: update
    operationId: updateUsingPUT
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: contactId
        in: path
        description: contactId
```

```
        required: true
        type: string
      - name: email
        in: query
        description: email
        required: false
        type: string
      - name: slackWebhookUrl
        in: query
        description: slackWebhookUrl
        required: false
        type: string
      - name: slackChannelOrUsername
        in: query
        description: slackChannelOrUsername
        required: false
        type: string
      - name: status
        in: query
        description: status
        required: false
        type: string
        default: ENABLED
        enum:
          - ENABLED
          - DISABLED
          - DELETED
    responses:
      '200':
        description: OK
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found
  delete:
    tags:
      - contact-controller
    summary: delete
    operationId: deleteUsingDELETE
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: contactId
        in: path
        description: contactId
        required: true
        type: string
    responses:
      '200':
        description: OK
      '204':
        description: No Content
      '401':
        description: Unauthorized
```

```yaml
        '403':
          description: Forbidden

  # Rutas de obtención de eventos
    /v1/events:
      get:
        tags:
          - event-controller
        summary: list
        operationId: listUsingGET_1
        consumes:
          - application/json
        produces:
          - application/json
        parameters:
          - name: days
            in: query
            description: days
            required: false
            type: integer
            default: 30
            format: int32
        responses:
          '200':
            description: OK
            schema:
              type: array
              items:
                $ref: '#/definitions/EventResponse'
          '401':
            description: Unauthorized
          '403':
            description: Forbidden
          '404':
            description: Not Found

  # Rutas de gestión de recolectores
    /v1/harvesters:
      get:
        tags:
          - harvester-controller
        summary: list
        operationId: listUsingGET_2
        consumes:
          - application/json
        produces:
          - application/json
        responses:
          '200':
            description: OK
            schema:
              type: array
              items:
                $ref: '#/definitions/HarvesterResponse'
          '401':
            description: Unauthorized
          '403':
            description: Forbidden
          '404':
            description: Not Found
```

```yaml
post:
  tags:
    - harvester-controller
  summary: add
  operationId: addUsingPOST_1
  consumes:
    - application/json
  produces:
    - application/json
  parameters:
    - name: label
      in: query
      description: label
      required: true
      type: string
    - name: notifyWarning
      in: query
      description: notifyWarning
      required: true
      type: boolean
    - name: notifyAlert
      in: query
      description: notifyAlert
      required: true
      type: boolean
    - name: warningValue
      in: query
      description: warningValue
      required: true
      type: string
    - name: alertValue
      in: query
      description: alertValue
      required: true
      type: string
    - name: type
      in: query
      description: type
      required: true
      type: string
      enum:
        - LOAD
        - MEMORY_USAGE
        - DISK_USAGE
    - name: status
      in: query
      description: status
      required: false
      type: string
      default: ENABLED
      enum:
        - ENABLED
        - DISABLED
        - DELETED
  responses:
    '200':
      description: OK
      schema:
        $ref: '#/definitions/HarvesterId'
    '201':
```

```
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

/v1/harvesters/{harvesterId}:
  get:
    tags:
      - harvester-controller
    summary: view
    operationId: viewUsingGET_1
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: harvesterId
        in: path
        description: harvesterId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/HarvesterResponse'
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found
  put:
    tags:
      - harvester-controller
    summary: update
    operationId: updateUsingPUT_1
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: harvesterId
        in: path
        description: harvesterId
        required: true
        type: string
      - name: label
        in: query
        description: label
        required: false
        type: string
      - name: notifyWarning
        in: query
        description: notifyWarning
        required: false
        type: boolean
```

```
    - name: notifyAlert
      in: query
      description: notifyAlert
      required: false
      type: boolean
    - name: warningValue
      in: query
      description: warningValue
      required: false
      type: string
    - name: alertValue
      in: query
      description: alertValue
      required: false
      type: string
    - name: status
      in: query
      description: status
      required: false
      type: string
      default: ENABLED
      enum:
        - ENABLED
        - DISABLED
        - DELETED
  responses:
    '200':
      description: OK
    '201':
      description: Created
    '401':
      description: Unauthorized
    '403':
      description: Forbidden
    '404':
      description: Not Found
delete:
  tags:
    - harvester-controller
  summary: delete
  operationId: deleteUsingDELETE_1
  consumes:
    - application/json
  produces:
    - application/json
  parameters:
    - name: harvesterId
      in: path
      description: harvesterId
      required: true
      type: string
  responses:
    '200':
      description: OK
    '204':
      description: No Content
    '401':
      description: Unauthorized
    '403':
      description: Forbidden
```

```yaml
# Rutas de gestión de recolectores
  /v1/observers:
    get:
      tags:
        - observer-controller
      summary: list
      operationId: listUsingGET_3
      consumes:
        - application/json
      produces:
        - application/json
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/ObserverResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
    post:
      tags:
        - observer-controller
      summary: add
      operationId: addUsingPOST_2
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: name
          in: query
          description: name
          required: true
          type: string
        - name: label
          in: query
          description: label
          required: true
          type: string
        - name: notifyStatusChanges
          in: query
          description: notifyStatusChanges
          required: true
          type: boolean
        - name: notifyInactivity
          in: query
          description: notifyInactivity
          required: true
          type: boolean
        - name: type
          in: query
          description: type
```

```
        required: false
        type: string
        default: SERVICE
        enum:
          - SERVICE
      - name: status
        in: query
        description: status
        required: false
        type: string
        default: ENABLED
        enum:
          - ENABLED
          - DISABLED
          - DELETED
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ObserverId'
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

/v1/observers/{observerId}:
  get:
    tags:
      - observer-controller
    summary: view
    operationId: viewUsingGET_2
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: observerId
        in: path
        description: observerId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ObserverResponse'
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found
  put:
    tags:
      - observer-controller
    summary: update
```

```yaml
    operationId: updateUsingPUT_2
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: observerId
        in: path
        description: observerId
        required: true
        type: string
      - name: label
        in: query
        description: label
        required: false
        type: string
      - name: notifyStatusChanges
        in: query
        description: notifyStatusChanges
        required: false
        type: boolean
      - name: notifyInactivity
        in: query
        description: notifyInactivity
        required: false
        type: boolean
      - name: status
        in: query
        description: status
        required: false
        type: string
        default: ENABLED
        enum:
          - ENABLED
          - DISABLED
          - DELETED
    responses:
      '200':
        description: OK
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found
  delete:
    tags:
      - observer-controller
    summary: delete
    operationId: deleteUsingDELETE_2
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: observerId
        in: path
        description: observerId
```

```
          required: true
          type: string
      responses:
        '200':
          description: OK
        '204':
          description: No Content
        '401':
          description: Unauthorized
        '403':
          description: Forbidden

# Ruta de ping
  /v1/ping:
    get:
      tags:
        - ping-controller
      summary: ping
      operationId: pingUsingGET
      consumes:
        - application/json
      produces:
        - application/json
      responses:
        '200':
          description: OK
          schema:
            type: string
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found

# Rutas de obtención de informes
  /v1/reports:
    get:
      tags:
        - report-controller
      summary: ofUser
      operationId: ofUserUsingGET
      consumes:
        - application/json
      produces:
        - application/json
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/ServerReportResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
```

```yaml
  /v1/reports/{serverId}:
    get:
      tags:
        - report-controller
      summary: ofServer
      operationId: ofServerUsingGET
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: serverId
          in: path
          description: serverId
          required: true
          type: string
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/ServerReportResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found

# Rutas de gestión de servidores
  /v1/servers:
    get:
      tags:
        - server-controller
      summary: list
      operationId: listUsingGET_4
      consumes:
        - application/json
      produces:
        - application/json
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/ServerSummaryResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
    post:
      tags:
        - server-controller
      summary: add
      operationId: addUsingPOST_3
      consumes:
```

```
      - application/json
    produces:
      - application/json
    parameters:
      - name: label
        in: query
        description: label
        required: true
        type: string
      - name: ip
        in: query
        description: ip
        required: true
        type: string
      - name: status
        in: query
        description: status
        required: false
        type: string
        default: ENABLED
        enum:
          - ENABLED
          - DISABLED
          - DELETED
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ServerCreateResponse'
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

/v1/servers/{serverId}:
  get:
    tags:
      - server-controller
    summary: view
    operationId: viewUsingGET_3
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: serverId
        in: path
        description: serverId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ServerFullResponse'
      '401':
```

```
      description: Unauthorized
    '403':
      description: Forbidden
    '404':
      description: Not Found
put:
  tags:
    - server-controller
  summary: update
  operationId: updateUsingPUT_3
  consumes:
    - application/json
  produces:
    - application/json
  parameters:
    - name: serverId
      in: path
      description: serverId
      required: true
      type: string
    - name: label
      in: query
      description: label
      required: true
      type: string
    - name: ip
      in: query
      description: ip
      required: true
      type: string
    - name: status
      in: query
      description: status
      required: false
      type: string
      default: ENABLED
      enum:
        - ENABLED
        - DISABLED
        - DELETED
  responses:
    '200':
      description: OK
    '201':
      description: Created
    '401':
      description: Unauthorized
    '403':
      description: Forbidden
    '404':
      description: Not Found
delete:
  tags:
    - server-controller
  summary: delete
  operationId: deleteUsingDELETE_3
  consumes:
    - application/json
  produces:
    - application/json
```

```yaml
      parameters:
        - name: serverId
          in: path
          description: serverId
          required: true
          type: string
      responses:
        '200':
          description: OK
        '204':
          description: No Content
        '401':
          description: Unauthorized
        '403':
          description: Forbidden

# Rutas de gestión de recolectores de servidores
  /v1/servers/{serverId}/harvesters:
    get:
      tags:
        - server-harvester-controller
      summary: list
      operationId: listUsingGET_5
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: serverId
          in: path
          description: serverId
          required: true
          type: string
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/ServerHarvesterResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
    post:
      tags:
        - server-harvester-controller
      summary: add
      operationId: addUsingPOST_4
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: serverId
          in: path
          description: serverId
          required: true
```

```
        type: string
      - name: harvesterId
        in: query
        description: harvesterId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ServerHarvesterId'
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

/v1/servers/{serverId}/harvesters/{serverHarvesterId}:
  get:
    tags:
      - server-harvester-controller
    summary: show
    operationId: showUsingGET
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: serverId
        in: path
        description: serverId
        required: true
        type: string
      - name: serverHarvesterId
        in: path
        description: serverHarvesterId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ServerHarvesterResponse'
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found
  delete:
    tags:
      - server-harvester-controller
    summary: delete
    operationId: deleteUsingDELETE_4
    consumes:
      - application/json
    produces:
```

```yaml
      - application/json
    parameters:
      - name: serverId
        in: path
        description: serverId
        required: true
        type: string
      - name: serverHarvesterId
        in: path
        description: serverHarvesterId
        required: true
        type: string
    responses:
      '200':
        description: OK
      '204':
        description: No Content
      '401':
        description: Unauthorized
      '403':
        description: Forbidden

# Rutas de gestión de observadores
  /v1/servers/{serverId}/observers:
    get:
      tags:
        - server-observer-controller
      summary: list
      operationId: listUsingGET_6
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: serverId
          in: path
          description: serverId
          required: true
          type: string
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/ServerObserverResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
    post:
      tags:
        - server-observer-controller
      summary: add
      operationId: addUsingPOST_5
      consumes:
        - application/json
      produces:
```

```
      - application/json
    parameters:
      - name: serverId
        in: path
        description: serverId
        required: true
        type: string
      - name: observerId
        in: query
        description: observerId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ServerObserverId'
      '201':
        description: Created
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

/v1/servers/{serverId}/observers/{serverObserverId}:
  get:
    tags:
      - server-observer-controller
    summary: show
    operationId: showUsingGET_1
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - name: serverId
        in: path
        description: serverId
        required: true
        type: string
      - name: serverObserverId
        in: path
        description: serverObserverId
        required: true
        type: string
    responses:
      '200':
        description: OK
        schema:
          $ref: '#/definitions/ServerObserverResponse'
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found
  delete:
    tags:
```

```yaml
        - server-observer-controller
      summary: delete
      operationId: deleteUsingDELETE_5
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: serverId
          in: path
          description: serverId
          required: true
          type: string
        - name: serverObserverId
          in: path
          description: serverObserverId
          required: true
          type: string
      responses:
        '200':
          description: OK
        '204':
          description: No Content
        '401':
          description: Unauthorized
        '403':
          description: Forbidden

  /v1/servers/{serverId}/uptime:
    get:
      tags:
        - server-controller
      summary: uptime
      operationId: uptimeUsingGET
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: serverId
          in: path
          description: serverId
          required: true
          type: string
      responses:
        '200':
          description: OK
          schema:
            $ref: '#/definitions/ServerUptimeResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found

# Rutas de obtención de UPTIME
  /v1/uptime:
    get:
      tags:
```

```yaml
      - uptime-controller
    summary: list
    operationId: listUsingGET_7
    consumes:
      - application/json
    produces:
      - application/json
    responses:
      '200':
        description: OK
        schema:
          type: array
          items:
            $ref: '#/definitions/ServerUptimeResponse'
      '401':
        description: Unauthorized
      '403':
        description: Forbidden
      '404':
        description: Not Found

# Rutas de gestión del usuario
  /v1/users/self:
    get:
      tags:
        - user-controller
      summary: show
      operationId: showUsingGET_2
      consumes:
        - application/json
      produces:
        - application/json
      responses:
        '200':
          description: OK
          schema:
            $ref: '#/definitions/UserInformationResponse'
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
    put:
      tags:
        - user-controller
      summary: update
      operationId: updateUsingPUT_4
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: password
          in: query
          description: password
          required: false
          type: string
        - name: firstName
          in: query
```

```
          description: firstName
          required: false
          type: string
        - name: lastName
          in: query
          description: lastName
          required: false
          type: string
      responses:
        '200':
          description: OK
        '201':
          description: Created
        '401':
          description: Unauthorized
        '403':
          description: Forbidden
        '404':
          description: Not Found
    delete:
      tags:
        - user-controller
      summary: delete
      operationId: deleteUsingDELETE_6
      consumes:
        - application/json
      produces:
        - application/json
      responses:
        '200':
          description: OK
        '204':
          description: No Content
        '401':
          description: Unauthorized
        '403':
          description: Forbidden


# Definiciones

definitions:
  ServerHarvesterId:
    type: object
    properties:
      id:
        type: string
  LoginResponse:
    type: object
    properties:
      token:
        type: string
  ServerUptimeResponse:
    type: object
    properties:
      harvestDate:
        type: string
      server:
        type: object
        additionalProperties:
```

```
        type: string
    uptime:
      type: integer
      format: int32
ContactResponse:
  type: object
  properties:
    email:
      type: string
    entryDate:
      type: string
    id:
      type: string
    slack:
      $ref: '#/definitions/Slack'
    status:
      type: string
      enum:
        - ENABLED
        - DISABLED
        - DELETED
ServerReportResponse:
  type: object
  properties:
    checks:
      type: object
      additionalProperties:
        type: array
        items:
          type: object
    id:
      type: string
    metrics:
      type: array
      items:
        type: object
    server:
      type: object
      additionalProperties:
        type: string
    timestamp:
      type: string
      format: date-time
    uptime:
      type: integer
      format: int32
ObserverResponse:
  type: object
  properties:
    entryDate:
      type: string
    label:
      type: string
    name:
      type: string
    notifyInactivity:
      type: boolean
    notifyStatusChanges:
      type: boolean
    observerId:
```

```
      type: string
    status:
      type: string
      enum:
        - ENABLED
        - DISABLED
        - DELETED
    type:
      type: string
      enum:
        - SERVICE
ServerHarvesterResponse:
  type: object
  properties:
    harvest:
      $ref: '#/definitions/Harvest'
    harvestDate:
      type: string
    harvester:
      $ref: '#/definitions/HarvesterResponse'
    serverHarvesterId:
      type: string
ContactId:
  type: object
  properties:
    id:
      type: string
UserInformationResponse:
  type: object
  properties:
    contactsCount:
      type: integer
      format: int32
    firstName:
      type: string
    harvestersCount:
      type: integer
      format: int32
    id:
      type: string
    lastName:
      type: string
    observersCount:
      type: integer
      format: int32
    serversCount:
      type: integer
      format: int32
    signUpDate:
      type: string
    username:
      type: string
Harvest:
  type: object
  properties:
    date:
      type: string
      format: date-time
ServerIdentificationResponse:
  type: object
```

```
  properties:
    id:
      type: string
    label:
      type: string
ServerCreateResponse:
  type: object
  properties:
    publicSSHKey:
      type: string
    serverId:
      type: string
Slack:
  type: object
  properties:
    slackChannelOrUsername:
      type: string
    slackWebhookUrl:
      type: string
EventResponse:
  type: object
  properties:
    eventDate:
      type: string
    message:
      type: string
    recipients:
      type: array
      items:
        $ref: '#/definitions/EventRecipient'
    server:
      $ref: '#/definitions/ServerIdentificationResponse'
ServerFullResponse:
  type: object
  properties:
    entryDate:
      type: string
    harvestStatus:
      type: string
      enum:
        - IN_PROGRESS
        - CONNECTION_ERROR
        - DONE
        - PENDING
    harvesters:
      type: array
      items:
        $ref: '#/definitions/ServerHarvesterResponse'
    harvestersCount:
      type: integer
      format: int32
    id:
      type: string
    ip:
      type: string
    label:
      type: string
    lastHarvestDate:
      type: string
    machineStatus:
```

```
      type: string
      enum:
        - UP
        - DOWN
        - UNKNOWN
    observers:
      type: array
      items:
        $ref: '#/definitions/ServerObserverResponse'
    observersCount:
      type: integer
      format: int32
    status:
      type: string
      enum:
        - ENABLED
        - DISABLED
        - DELETED
    uptime:
      type: integer
      format: int32
UserId:
  type: object
  properties:
    id:
      type: string
ServerObserverId:
  type: object
  properties:
    id:
      type: string
HarvesterResponse:
  type: object
  properties:
    alertValue:
      type: string
    entryDate:
      type: string
    harvesterId:
      type: string
    label:
      type: string
    notifyAlert:
      type: boolean
    notifyWarning:
      type: boolean
    status:
      type: string
      enum:
        - ENABLED
        - DISABLED
        - DELETED
    type:
      type: string
      enum:
        - LOAD
        - MEMORY_USAGE
        - DISK_USAGE
    warningValue:
      type: string
```

```yaml
ServerSummaryResponse:
  type: object
  properties:
    entryDate:
      type: string
    harvestStatus:
      type: string
      enum:
        - IN_PROGRESS
        - CONNECTION_ERROR
        - DONE
        - PENDING
    harvestersCount:
      type: integer
      format: int32
    id:
      type: string
    ip:
      type: string
    label:
      type: string
    lastHarvestDate:
      type: string
    machineStatus:
      type: string
      enum:
        - UP
        - DOWN
        - UNKNOWN
    observersCount:
      type: integer
      format: int32
    status:
      type: string
      enum:
        - ENABLED
        - DISABLED
        - DELETED
    uptime:
      type: integer
      format: int32
ServerObserverResponse:
  type: object
  properties:
    observationDate:
      type: string
    observer:
      $ref: '#/definitions/ObserverResponse'
    serverObserverId:
      type: string
    status:
      type: string
      enum:
        - UP
        - DOW
        - NOT_CHECKED
        - CONNECTION_ERROR
EventRecipient:
  type: object
  properties:
```

```
    email:
      type: string
    slack:
      type: string
HarvesterId:
  type: object
  properties:
    id:
      type: string
ObserverId:
  type: object
  properties:
    id:
      type: string
```