Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTUDIO E IMPLANTACIÓN DE UN PROTOTIPO DE SISTEMA INFORMÁTICO PARA LA GESTIÓN Y MONITORIZACIÓN DE EQUIPOS INFORMÁTICOS

Alumno: José Ángel Pastrana Padilla

Tutor: Prof. D. José Ramón Balsas Almagro

Dpto: Departamento de Informática

Septiembre, 2022

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Trabajo Fin de Grado titulado: «**Estudio e implantación de un prototipo de sistema informático para la gestión y monitorización de equipos informáticos»**, que presenta José Ángel Pastrana Padilla, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, septiembre de 2022

El alumno: Los tutores:

José Ángel Pastrana Padilla José Ramón Balsas Almagro

ÍNDICE DE APARTADOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En este apartado introduciremos la temática general del proyecto describiendo la motivación del mismo. Una vez definido el contexto, estableceremos una serie de objetivos sobre el trabajo que realizaremos. En último lugar, indicaremos la metodología empleada y resumiremos la estructura de los siguientes apartados.

1.1. Motivación

Los equipos informáticos son un bien material presente en nuestro día a día para ayudarnos a la realización de una amplia variedad de tareas. Es por ello que también son un recurso habitual dentro de cualquier organización. Independientemente del número de equipos en posesión, es de sumo interés para una organización conocer el uso que tienen éstos con el fin de velar por el correcto mantenimiento de los mismos.

Las labores de supervisión de los equipos informáticos dentro de una organización no son sostenibles ejecutándose manualmente por el responsable al cargo debido al tiempo necesario para ello. Afortunadamente, las organizaciones modernas cuentan con una extensa infraestructura de red y la mayoría de sus equipos informáticos se encuentran conectados a ella. Así pues, se abre la posibilidad de gestionar y monitorizar dichos equipos de manera remota desde un sistema que facilite estas labores.

En este proyecto proponemos el estudio y desarrollo de un prototipo de sistema informático para la gestión y monitorización de equipos informáticos accesible mediante un navegador web.

1.2. Objetivos

Los objetivos que enumeramos a continuación definen las metas generales del proyecto dentro del contexto de la motivación. A partir de ellos, realizaremos un trabajo de ingeniería de software para satisfacerlos y que reflejaremos en el resto de la memoria.

- Realizar un estudio de necesidades, soluciones y tecnologías que tengan relación con el sistema propuesto, en este caso sobre la gestión y monitorización de equipos informáticos.

- Diseñar un sistema informático especialmente orientado a la utilización de tecnologías web para su uso.

- Implementar un prototipo del sistema propuesto donde se pueda verificar su viabilidad y funcionalidad.

1.3. Metodología

Las metodologías de trabajo establecen un marco de procesos para la ejecución de un proyecto dentro de los objetivos fijados en el mismo. Al respecto, hemos considerado que metodologías con un enfoque ágil como *Scrum* se adaptan mejor a las necesidades actuales del proyecto [1], ya que los objetivos planteados inicialmente son bastante abiertos y no queremos someterlos a un enfoque rígido, sino flexible.

Scrum es una metodología ágil enfocada en la flexibilidad y la satisfacción del cliente mediante una entrega continuada y constante de resultados con valor. No obstante, tendremos que apuntar algunas modificaciones sobre la versión original para adaptarla a las circunstancias particulares de nuestro proyecto.

Originalmente, se trata de una metodología colaborativa que se compone por múltiples roles: el propietario del producto (en inglés, *Product Owner*) define el alcance del producto y gestiona los requisitos del cliente, el experto en *Scrum* (en inglés, *Scrum Master*) coordina las prácticas de la metodología y el equipo de trabajo realiza el desarrollo que genera un incremento de valor en el producto. Sin embargo, este proyecto se sostiene por un único miembro y, por tanto, todos los roles mencionados los asume la misma persona; es decir, quien suscribe.

En consecuencia, algunos de los eventos dentro de *Scrum* no pueden cumplirse tal y como están planteados dentro de la metodología como son las reuniones diarias para coordinar el trabajo del equipo. Estas reuniones persiguen

identificar anomalías en el avance de las tareas y tomar decisiones rápidas para cumplir siempre con los objetivos fijados. En nuestro caso -con un único miembro en el equipo-, esto sólo puede realizarse adaptándolo a un ejercicio diario de introspección.

Sin embargo, en lo que respecta a la planificación del trabajo en *Scrum*, podemos mantener la mayor parte del proceso intacto. Así, comenzaremos recogiendo las ideas del cliente –rol que también ejerceremos– para obtener las historias de usuario a las que le asignaremos una estimación de esfuerzo y una prioridad en realizarlas. La suma de todas las historias de usuario dará lugar a la pila del producto (en inglés, *Product Backlog*).

Definiremos ciclos de iteraciones (en inglés, *Sprint*) de duración y esfuerzo constante donde trabajaremos sobre las historias de usuario más prioritarias de la pila del producto. En la planificación de una iteración, dividiremos las historias de usuario en tareas obteniendo como resultado la pila de tareas de la iteración (en inglés, *Sprint Backlog*). Tras finalizar las tareas de una iteración, obtendremos un resultado incremental y funcional del producto.

Repetiremos el ciclo de iteraciones hasta finalizar las historias de usuario contenidas en la pila del producto.

1.4. Estructura del documento

A continuación, explicaremos brevemente el contenido de cada uno de los siguientes apartados.

En el segundo apartado –dedicado al análisis del proyecto– profundizaremos sobre el contexto del proyecto definiendo las características, necesidades, soluciones existentes y usuarios objetivo de un sistema de gestión y monitorización de equipos informáticos. Como síntesis de lo anterior, formularemos una propuesta de solución sobre la que realizaremos una planificación de tareas y un estudio de viabilidad.

En el tercer apartado –dedicado a los aspectos de diseño– crearemos especificaciones sobre los componentes del sistema para satisfacer las funcionalidades contempladas en el apartado de análisis y que plasmaremos mediante diferentes tipos de diagramas e interfaces.

En el cuarto apartado –dedicado a la implementación– entraremos a discutir sobre la arquitectura y los detalles más relevantes de implementación del proyecto, así como también abordaremos la implicación de los diferentes *frameworks* seleccionados.

En el quinto apartado –dedicado a las pruebas– ejecutaremos el plan de pruebas definido en el apartado de diseño para validar el correcto funcionamiento del sistema y analizaremos los resultados obtenidos.

En el sexto apartado –dedicado a las conclusiones– recogeremos una reflexión sobre el trabajo realizado valorando los objetivos realizados, las tecnologías empleadas y la metodología seleccionada. También discutiremos sobre posibles mejoras en el proyecto en un hipotético trabajo futuro.

En último lugar, incluiremos cuatro anexos con la descripción de los contenidos suministrados, un manual de instalación del sistema, el acceso a una instalación de pruebas y un manual de usuario.

2. ANÁLISIS

En este apartado profundizaremos sobre el contexto del proyecto definiendo las características, necesidades, soluciones existentes y usuarios objetivo de un sistema de gestión y monitorización de equipos informáticos. Como síntesis de lo anterior, formularemos una propuesta de solución sobre la que realizaremos una planificación de tareas y un estudio de viabilidad.

2.1. Análisis preliminar

Cualquier organización posee una considerable cantidad de equipamiento informático en su inventario. La mayoría de este equipamiento no se encuentra almacenado ni apagado, sino que está en constante uso por los miembros de la organización. En el sector de la tecnología de la información, los sistemas de gestión y monitorización cumplen con el propósito de ayudar a una organización a salvaguardar su nivel de operatividad informática. Éstos proporcionan la gestión y seguimiento de la infraestructura de la organización mediante la recolección, almacenamiento, procesado y visualización de los eventos producidos por los diferentes componentes implicados a través de la red. Sin un sistema de estas características, la gestión y monitorización sería individualizada por cada equipo informático existente, repercutiendo en una mayor inversión de tiempo y capital humano para el mantenimiento de los mismos.

Ahora bien, cabe preguntarnos qué clase de equipamiento informático podemos incluir en estos sistemas. En la mayoría de los casos, si el equipo tiene acceso a la red y suministra herramientas propias de gestión y monitorización local, significa que podemos trasladar éstas a un frente remoto. Evidentemente existen algunas restricciones: los equipos con software privativo quizás necesiten integraciones específicas o sólo funcionen en su propia plataforma de gestión y monitorización. Pero en general, ya sean ordenadores de sobremesa, *routers*, *switches*, impresoras, portátiles o móviles, todos ellos son perfectos candidatos. En el caso de que un equipo sea irremediablemente imposible de integrar, algunos sistemas los tratan como un ítem en un inventario para constatar su existencia.

Respecto a las características de un sistema de monitorización, encontraremos el concepto de métrica, definiéndose como la medida de un recurso concreto. Por ejemplo, los sistemas operativos exponen métricas sobre sus recursos físicos como son el uso de CPU en porcentaje, el espacio en memoria principal ocupado en MiB, etc. También el software cuenta con sus propias métricas para indicar el uso de una característica en una aplicación dada, por ejemplo, el número de pestañas abiertas en un navegador. Cuando un programa externo es el que recoge las métricas, éste recibe el nombre de agente.

Por otro lado, los sistemas de gestión nos permiten efectuar operaciones de configuración o supervisión sobre los equipos, ya sea comprobando su conectividad, accediendo a una consola remota, bloqueando su funcionamiento al usuario, instalando o actualizando aplicaciones, administrando licencias, etc.

Sin embargo, un motivo que diferencia a un sistema de gestión y monitorización sobre otro es el tipo de organización para el que va a ir destinado. Así, diferentes organizaciones pueden requerir soluciones diferentes en función de su naturaleza (empresarial o educativo), ámbito (público o individual), dimensiones (una sede o varias), capacitación tecnológica de los miembros y recursos humanos destinados al soporte técnico.

En función de su naturaleza, podemos encontrar un uso distinguido entre el entorno empresarial o el educativo. En el entorno empresarial, el trabajo está segmentado y diferenciado, por lo que estos sistemas son utilizados únicamente entre los miembros del departamento de sistemas. Mientras, en el sector educativo, es común que un profesor realice la gestión de la clase a través de la misma plataforma y, por tanto, el sistema debe adaptarse también a usuarios menos expertos en este contexto tan particular.

En función de su ámbito, un equipo expuesto al público contrae medidas de monitorización más exhaustivas para velar por su disponibilidad, ya que potencialmente un mal funcionamiento de éste puede afectar a un rango mayor de usuarios. Por ejemplo, un punto de acceso inalámbrico situado en una zona común que ha dejado de enviar información de monitorización nos hace suponer que algo va mal. De este modo, las métricas recogidas instantes antes nos ayudan a valorar

lo que pudo ocurrir. Sin embargo, podemos considerar correcto que un portátil designado a un uso individual pueda permanecer una semana sin recolectar información o que la precisión sea menor.

En función de sus dimensiones, la sintetización de los datos de monitorización recogidos nos permite emitir comparaciones sobre la vida útil o rendimiento de unos equipos sobre otros. Para una organización con varias sedes, se trata de una característica atractiva, mientras que para una organización pequeña, su número de equipos no es suficientemente significativo para compararlos y arrojar conclusiones.

En función de la capacitación tecnológica de sus miembros, necesitaremos herramientas de gestión cuando éstos no poseen un buen dominio sobre la utilización de los equipos. Por ejemplo, para asistir a un miembro con baja capacitación tecnológica, podemos recurrir a la conexión remota de su escritorio. En caso de que nos encontremos en una organización altamente capacitada, los mecanismos de gestión pueden ser otros como matar el proceso conflictivo remotamente.

En función del personal de soporte técnico, las herramientas de gestión y monitorización son, además, herramientas colaborativas. Por ello, cada miembro del soporte técnico cubre el mantenimiento de cierta área dentro de la organización (por ejemplo, una planta o un edificio). De este modo, las métricas de su área son presentadas con mayor foco sobre el personal asignado a la misma.

Éstos sólo han sido ejemplos de situaciones dentro de las diferentes visiones y necesidades que una organización busca en un sistema de gestión y monitorización. Para comprender mejor el conjunto de características de uno, los siguientes apartados conformarán un breve estudio sobre estos sistemas.

2.1.1. Usuario objetivo

Los sistemas de gestión y monitorización son herramientas especializadas y destinadas a un perfil profesional concreto del ámbito de la informática: los administradores de sistemas. En España existen varias titulaciones oficiales que

reúnen las competencias necesarias para el desempeño de este cargo en diferentes niveles:

- Ciclo formativo de grado medio en sistemas microinformáticos y redes
- Ciclo formativo de grado superior en administración de sistemas informáticos en red
- Grado en ingeniería informática

Para el rol de administrador de sistemas, desde los ciclos formativos se instruye al profesional a nivel técnico para la instalación, configuración y reparación de equipos informáticos, mientras que en los grados universitarios y posgrados se adquieren las competencias para la documentación de la infraestructura informática, el diseño y desarrollo de módulos de monitorización o la formulación y análisis de métricas. Según el portal de empleo *Indeed*, la diferencia de salario base en España entre un técnico de sistemas [2] y un ingeniero de sistemas [3] se aproxima a los 8000 € anuales, como podemos ver en la siguiente ilustración.

Ilustración 1. Salario de ingeniero de sistemas y técnico de sistemas

Como complemento a la formación reglada, algunas entidades emiten certificaciones propias que avalan la adquisición de competencias para especializarse en la administración de sistemas. Cabe mencionar que estas certificaciones normalmente se centran en sistemas concretos. Por ejemplo, *Microsoft Azure Administrator Associate* está destinado a la monitorización de una organización con productos de Microsoft Azure [4] o *Linux Foundation Certified System Administrator* que se centra en la administración de sistemas Linux [5].

Desde el año 2000 –y como curiosidad–, la popularidad de este rol ha propiciado la celebración del día del administrador de sistemas en el último viernes del mes de julio [6].

En la actualidad, algunos sistemas de gestión y monitorización tratan de alcanzar otros perfiles de usuario fuera del ámbito de la informática, siendo más amigables tanto en la presentación de su interfaz como en su manejo. Por ejemplo, para obtener informes o tener un uso básico del control de equipos.

2.1.2. Seguridad

Una definición popular de la palabra seguridad es la ausencia de peligro, daño o riesgo [7]. Dado que los sistemas de gestión y monitorización nos permiten conocer el estado de los equipos informáticos de la organización, podríamos pensar que éstos son un buen mecanismo para implantar medidas de seguridad, de modo que consigamos alcanzar las premisas de la definición anterior. Pero si pensamos que esto sería suficiente, estaríamos errando profundamente.

Cuando los sistemas de monitorización observan una situación anómala, es decir, cuando una o varias métricas alcanzan ciertas condiciones fijadas, notifican sobre ello a los administradores de sistemas mediante alertas. Pero éstos no disponen de mecanismos para mitigar la alerta y desconocen si fue generada por una amenaza de seguridad o debido a un mal funcionamiento en los equipos. En otras palabras, los sistemas de monitorización son meros observadores.

Sin embargo, esto no quiere decir que no existan otros sistemas que sí puedan identificar y responder a las amenazas de seguridad. Los sistemas IDS e IPS nos

permiten implantar medidas de seguridad apropiadas y se complementan con los sistemas de gestión y monitorización.

Los sistemas IDS, *Intrusion Detection System* o sistema de detección de intrusiones en español, nos permiten detectar accesos no autorizados cotejando la actividad sospechosa con una base de datos de ataques conocidos [8]. De manera análoga a un sistema de monitorización, tampoco mitigan una posible situación de riesgo y se centran en la labor de identificación de amenazas.

Los sistemas IPS, *Intrusion Prevention System* o sistema de prevención de intrusiones en español, actúan preventivamente cuando se va a producir un incidente de seguridad o cuando éste ya se está produciendo y trabajan en conjunto con los sistemas IDS.

Aunque este tema queda fuera de nuestra temática, una organización debe considerar implantar sistemas IDS/IPS en su infraestructura informática.

2.2. Soluciones existentes

A continuación, vamos a realizar un recorrido por varias soluciones existentes que nos permitan profundizar en la temática que nos ocupa y nos detendremos a explicar las características más llamativas de cada una de éstas.

Para la selección de las distintas soluciones, hemos tenido en cuenta que abarquen diferentes funcionalidades y propósitos entre sí. De este modo, encontraremos soluciones que se centran en la monitorización, dejando en segundo plano la gestión del equipamiento informático, y viceversa. Algunas soluciones, que pueden llegar a ser muy populares y conocidas incluso por el lector, quizás no estén incluidas aquí para evitar reincidir en exceso sobre los mismos conceptos.

2.2.1. Amazon CloudWatch

Amazon CloudWatch pertenece al catálogo de herramientas para administradores de sistemas de AWS (*Amazon Web Services*) [9]. Es un sistema de pago que opera en la nube y está especialmente enfocado en la monitorización de

servicios e instancias de Amazon [10]. La aplicación web divide su área de trabajo en 5 secciones [11].

La sección «Alarmas» permite la configuración de avisos de relevancia sobre el comportamiento de los servicios e instancias. Por ejemplo: un aviso para cuando cierta instancia supere el 80% de memoria principal en uso.

La sección «Registros» (en inglés, «*Log*») almacena y visualiza los mensajes de depuración generados por los servicios e instancias. Dicha información facilita la trazabilidad de situaciones anómalas o desconocidas.

La sección «Métricas» almacena y visualiza los recursos empleados por los servicios e instancias. Por ejemplo: uso de CPU, ocupación de disco, etc.

La sección «Eventos» almacena y visualiza cambios sobre el estado de los servicios e instancias. Por ejemplo: la instancia fue reiniciada manualmente con éxito a las 6.30 am.

La sección «Monitorización de aplicaciones» enlaza las secciones anteriores visualmente para tener un enfoque global de la configuración de monitorización existente sobre los servicios e instancias.

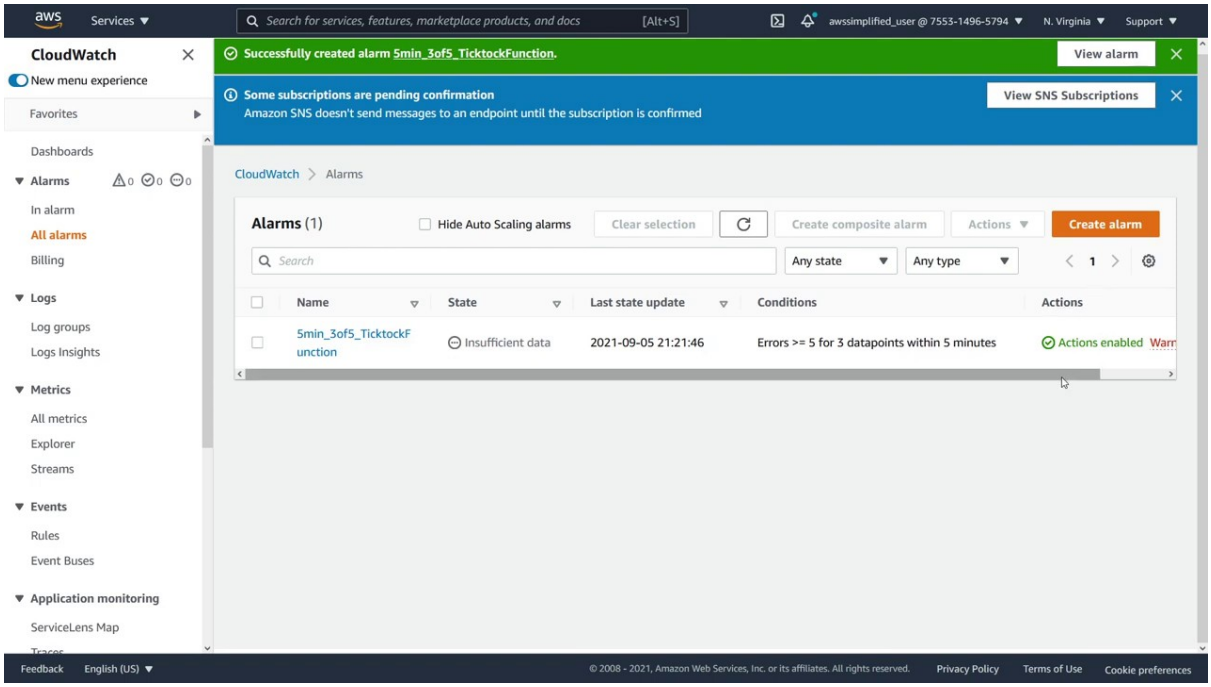

Ilustración 2. Panel de configuración de alertas de Amazon CloudWatch

Además, destacamos otras características como [12]:

- Exportación de métricas a sistemas de monitorización externos.
- Agrupación de alarmas cuando se producen múltiples sobre el mismo componente.
- Disparadores de acción a partir de umbrales en las métricas.
- Detección de anomalías a través de técnicas de inteligencia artificial.
- Política de almacenamiento de métricas de hasta 15 meses.
- Creación de métricas y gráficas personalizadas.

Este sistema, a diferencia de otros, no necesita que el administrador de sistemas elabore ni integre la recepción de métricas, ya que hablamos de una solución pensada por y para el ecosistema de Amazon. Actualmente, más de 70 productos de AWS se encuentran integrados.

2.2.2. Azure Monitor

Azure Monitor es un sistema de pago que opera en la nube para la gestión y monitorización de las aplicaciones y servicios de una suscripción de Microsoft Azure.

Sin embargo, no se limita a trabajar únicamente con los productos ya existentes de la suscripción, sino que permite incorporar la monitorización de elementos de una infraestructura local como las máquinas virtuales ejecutadas en el hipervisor Hyper-V o los contenedores de aplicaciones orquestadas mediante Kubernetes. Además de lo anterior, también posee un recolector de datos extensible para incorporar métricas personalizadas sobre dispositivos o entornos inicialmente no soportados [13].

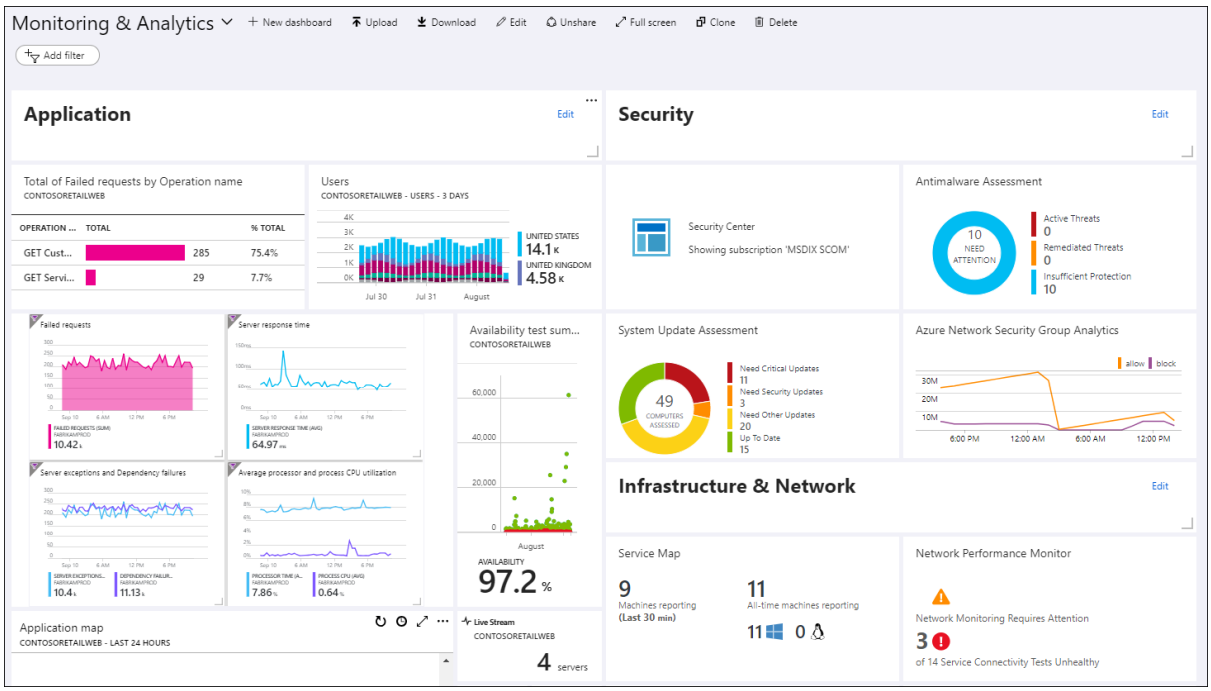

Ilustración 3. Panel principal de monitorización y análisis de Azure Monitor

El panel de monitorización de Azure Monitor es personalizable y permite escribir consultas propias sobre las métricas almacenadas utilizando el lenguaje de consulta Kusto, KQL [14], para generar nuevos gráficos y tablas.

Incorpora una herramienta para la redacción de informes técnicos llamada Workbooks que combina la escritura en lenguaje de marcas Markdown con la incrustación de registros, métricas, gráficos o alertas [15].

De cara a colaborar con otros sistemas de monitorización, los datos en Azure Monitor son consultables y exportables por pasarelas de integración.

2.2.3. Cisco Intersight Workload

Cisco Intersight Workload es un sistema de pago que opera en la nube de la empresa Cisco y que forma parte de la solución Cisco Intersight Platform para el control y monitorización de la infraestructura de una organización [16]. Centra sus esfuerzos en su versatilidad para recolectar métricas desde una amplia variedad de orígenes [17]:

- Orquestadores de aplicaciones: como Apache Tomcat o IBM Websphere.
- Orquestadores de contenedores: como Kubernetes o Red Hat Openshift.
- Protocolos de mensaje: como SNMP o WMI.
- Hipervisores: como Microsoft Hyper-V o VMWare vCenter.
- Proveedores de nube pública: como Amazon AWS, Microsoft Azure o Google Cloud.

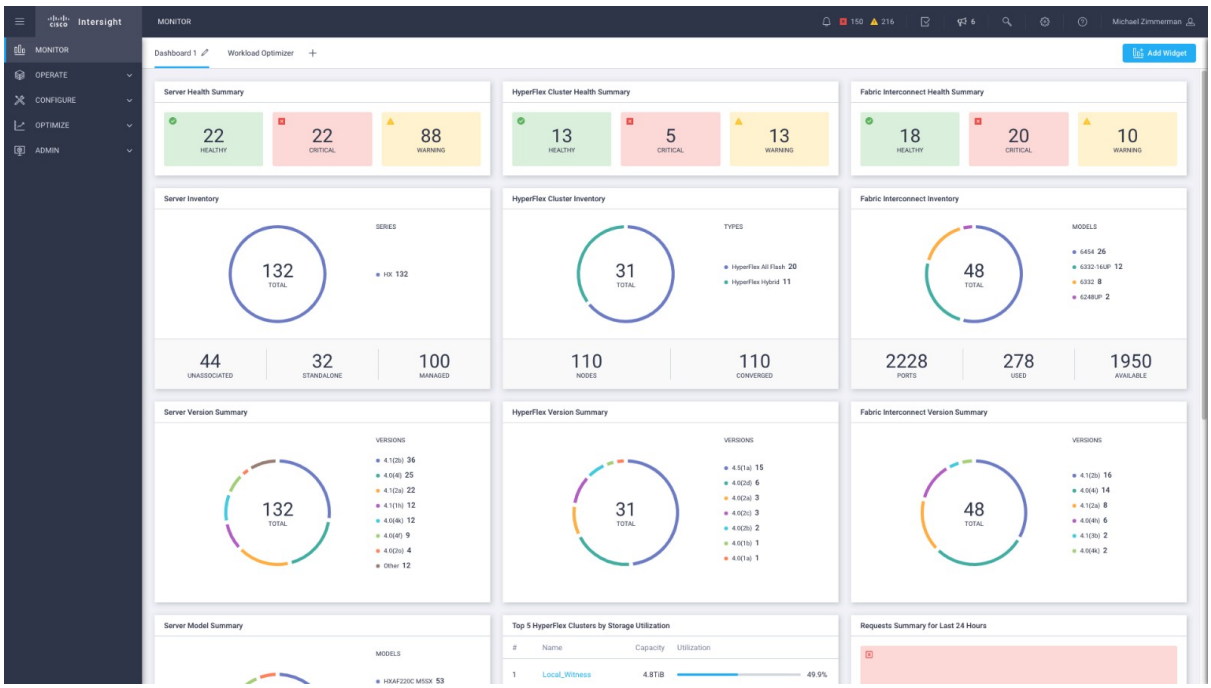

Ilustración 4. Panel de monitorización de Cisco Intersight Workload

Además, dicha versatilidad se extiende más allá de la monitorización individualizada de cada uno de los recursos permitiendo agruparlos cuando unos dependen de otros para gestionarlos coordinadamente. Por ejemplo, nos permite que un servidor, las máquinas virtuales que ejecuta y las aplicaciones dentro de éstas sean visualizadas como un grafo, ofreciéndonos un refinamiento sobre la consistencia de todas las métricas.

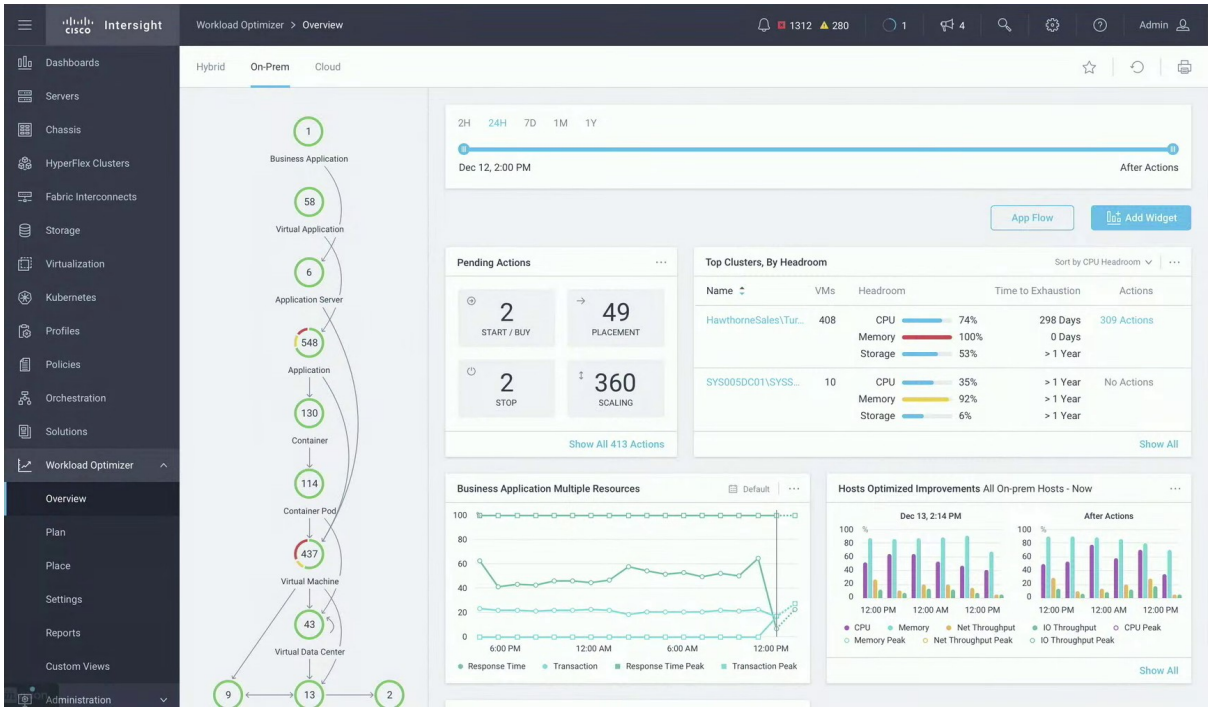

Ilustración 5. Grafo de métricas en Cisco Intersight Workload

2.2.4. GLPI

GLPI, cuyas siglas se traducen por “Gestión Libre de Parques Informáticos”, es un sistema de gestión de equipamiento informático de código abierto nacido en 2003 y gratuito. La empresa Teclib es la autora del proyecto y ofrece diferentes planes económicos para organizaciones que busquen un soporte adicional [18].

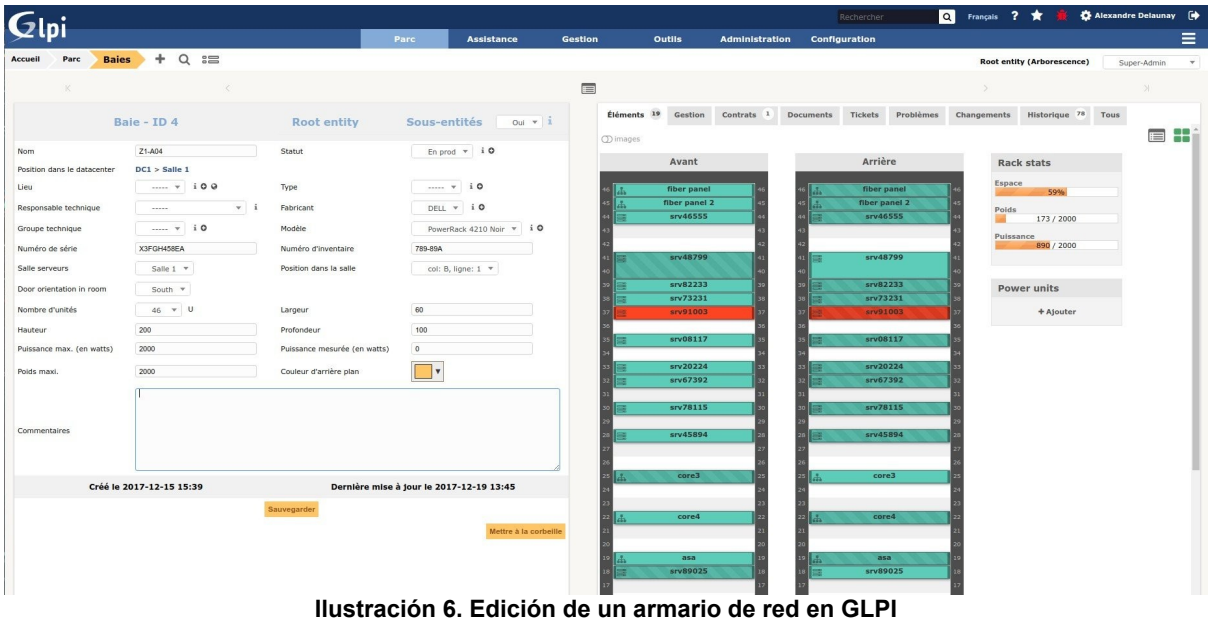

Ilustración 6. Edición de un armario de red en GLPI

Dispone de un exhaustivo control de inventario, donde quedan registrados ordenadores, monitores, *routers*, *switches*, impresoras, consumibles, teléfonos, licencias de software y otros elementos pasivos. Además, mantiene un historial con la documentación de compra, periodos de garantía, tiempo de vida útil, cambios de ubicación y reparaciones acometidas. También integra un gestor de incidencias con el que registrar anomalías, permitiendo definir cuál es su prioridad, categoría e impacto.

Sin embargo, este sistema no cuenta con un mecanismo de monitorización basado en métricas, sino que su monitorización está orientada a la generación de reportes sobre la infraestructura de la organización y a la implantación de avisos futuros tales como la caducidad de una licencia, el fin de la vida útil de un dispositivo o el fin del soporte del fabricante.

2.2.5. Nagios

Nagios se presenta a sí mismo como el estándar en la industria de la tecnología de la información para la monitorización de infraestructuras. El motor original, Nagios Core, es de código abierto y gratuito, mientras que Nagios XI es su evolución como producto comercial.

Tanto Nagios Core como Nagios XI permiten recolectar métricas sobre aplicaciones, servicios, sistemas operativos y protocolos de red. Sin embargo, Nagios XI añade asistentes para la configuración del motor, gráficas y tablas en tiempo real, paneles personalizados, mapas visuales para localizar dispositivos y reportes predictivos que señalan las tendencias para el futuro [19].

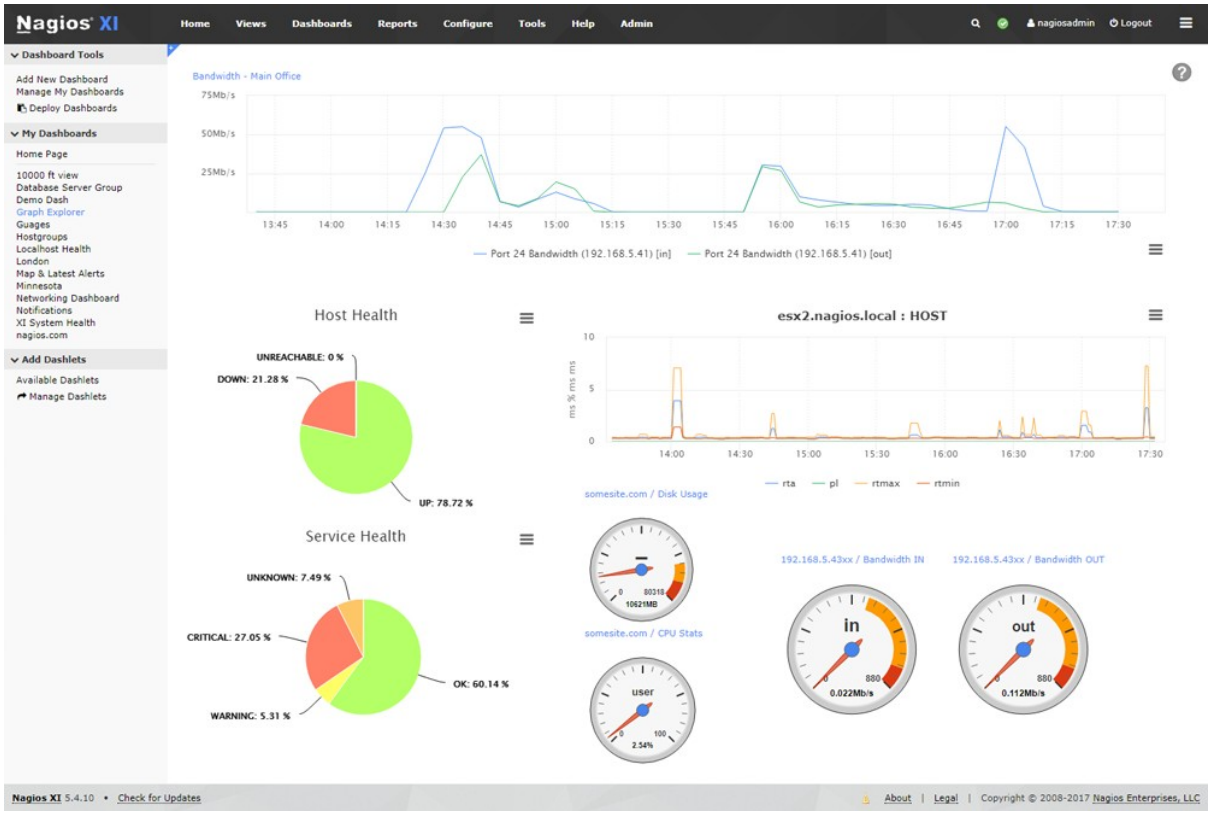

Ilustración 7. Nagios XI en el panel de salud de un equipo

Existen otras soluciones comerciales relacionadas con el producto principal: Nagios Log Server, Nagios Network Analyzer y Nagios Fusion [20]. Nagios Log Server y Nagios Network Analyzer complementan al motor principal, ya sea para el almacenamiento y análisis de registros como para la visualización e identificación del tráfico en la red, mientras que Nagios Fusion permite centralizar la gestión de múltiples instancias de Nagios Core o Nagios XI en organizaciones complejas con varias sedes [21].

Por último, cabe mencionar la extensibilidad de Nagios gracias a su sistema de *plugins* de terceros para obtener funcionalidades adicionales [22].

2.2.6. Zabbix

Zabbix es un sistema de monitorización de código abierto y gratuito, aceptado entre organizaciones de gran calibre en todo el mundo como Dell, ICANN, Orange o T-Systems y en todo tipo de sectores como el de las telecomunicaciones, el educativo, el gubernamental o el de la salud.

Sus características principales son, además de las características esperadas de un sistema de monitorización, la capacidad para ser un sistema distribuido mediante una red de *proxies* que canalizan las métricas, su alto rendimiento gracias a la aplicación de métodos de caché y su bajo coste de mantenimiento debido a su retrocompatibilidad entre versiones desplegadas.

También destaca su capacidad para manejar grandes volúmenes de dispositivos simultáneamente. Según su web principal, existen instalaciones de Zabbix con más de 100.000 dispositivos monitorizados al mismo tiempo u otras con más de 3.000.000 de comprobaciones de actividad por minuto [23].

Ilustración 8. Tablero personalizado de Zabbix

Además, cuenta con una amplia cantidad de plantillas para integrarse sobre diferentes tipos de aplicaciones y sistemas externos e incorpora mecanismos para el descubrimiento de equipamiento informático en la red de manera automática, lo que facilita su implantación por primera vez en organizaciones con recursos muy heterogéneos.

Aunque Zabbix sea gratuito, también podemos encontrar asesoramiento técnico, formación e implantación de servicios específicos para aquellas organizaciones que lo necesiten a través de su programa de distribuidores. En total,

247 empresas distribuyen oficialmente Zabbix a lo largo de todo el mundo. En España, encontramos a 9 de estas empresas distribuidoras [24].

2.2.7. Pandora FMS

Pandora Flexible Monitoring Solution es un sistema de monitorización de código abierto que se distingue por su extensibilidad. Posee un catálogo de más de 700 *plugins* [25] y se integra con sistemas externos de inventariado, resolución de incidencias y control remoto de equipamiento informático como Integra IMS o eHorus.

Este sistema se divide en dos versiones diferentes: la versión Community y la versión Enterprise. La versión Community está disponible gratuitamente para estudiantes y pequeñas organizaciones, mientras que la versión Enterprise está pensada para grandes organizaciones y es de pago.

Ambas versiones nos permiten visualizar las métricas almacenadas, obtener informes y recibir alertas. Destacamos la monitorización abstracta de la infraestructura de la organización, de entornos virtuales, de redes y de dispositivos IoT. Sin embargo, sólo en la versión Enterprise encontramos el resto de opciones avanzadas [26].

Pandora FMS Enterprise, a diferencia de la versión Community, es un sistema altamente escalable y ofrece una alta disponibilidad como servicio. Para lograrlo, sigue un sistema distribuido de n-servidores independientes y una metaconsola web para conectarnos a ellos [27]. Además, implementa el envío de métricas asíncronas y compactación de las mismas para reducir el número de conexiones diarias de los dispositivos con los servidores. Gracias a todo lo anterior y según su propio blog, Hughes India ha conseguido monitorizar más de 100.000 dispositivos [28].

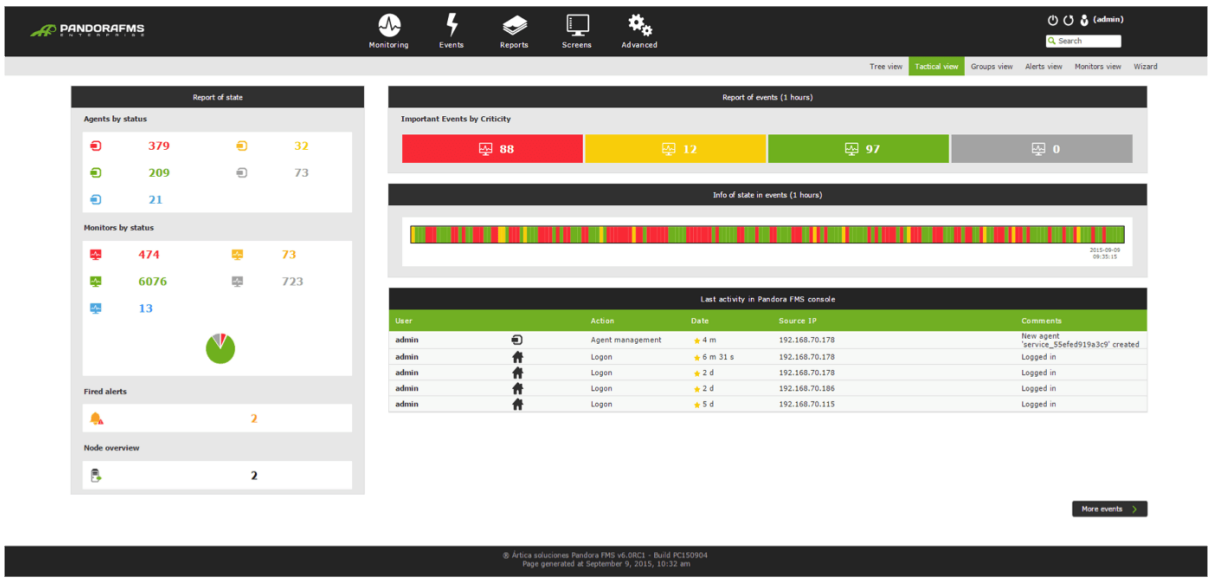

Ilustración 9. Vista principal de Pandora FMS

Actualmente, la versión Enterprise también se distribuye como un servicio ofrecido a través de la nube bajo pago por uso. Esto externaliza la configuración necesaria que un equipo de ingenieros de sistemas tendría que realizar para desplegar el servicio bajo la infraestructura de la propia organización, mencionándose en su página web como una de sus principales ventajas [29].

2.2.8. Veyon

Veyon, anteriormente conocido como iTALC [30], es un sistema de control remoto para equipos informáticos, de código abierto y gratuito, fundamentalmente orientado a la gestión de aulas en instituciones educativas [31].

Entre sus características, permite a un administrador monitorizar la actividad de múltiples equipos simultáneamente, así como interactuar mediante escritorio remoto con éstos. Además, permite realizar otras gestiones como bloquear la sesión de un usuario, reiniciar un equipo, distribuir documentos remotamente, establecer un canal de conversación por chat, impedir el acceso a internet o tomar una captura de pantalla.

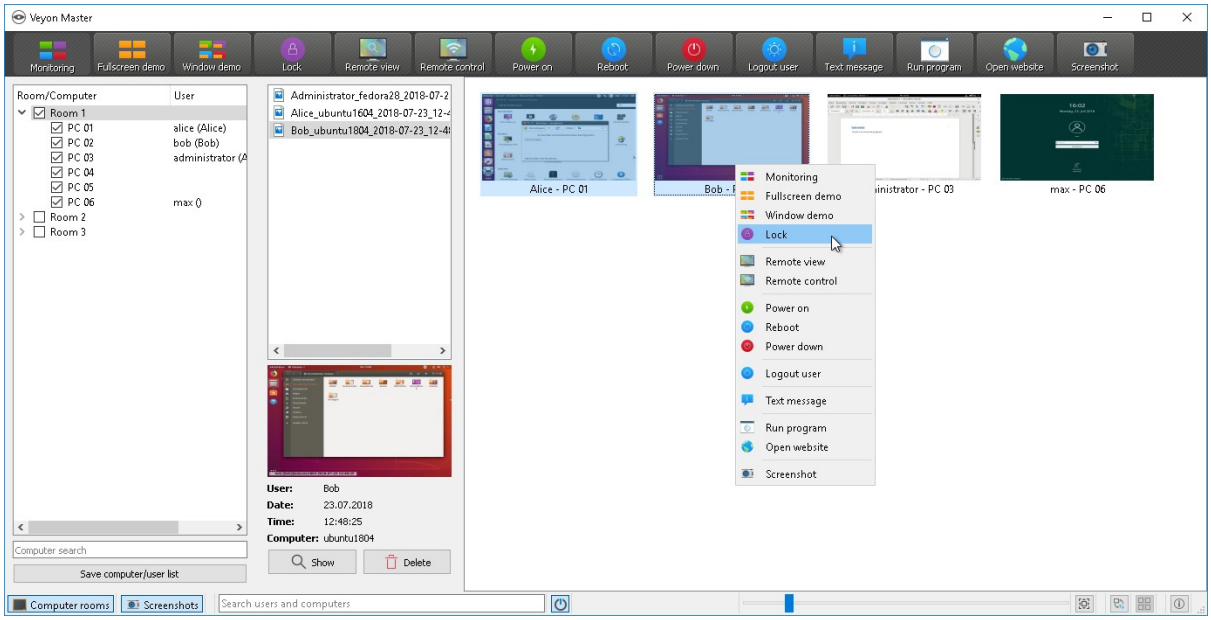

Ilustración 10. Panel de equipos encendidos en Veyon

Dispone de un repositorio de *addons* de pago para extender las funcionalidades de gestión, donde podemos encontrar *addons* para la grabación de pantalla o para el descubrimiento automático de nuevos equipos en la red local.

Sin embargo, este sistema no emplea tecnologías webs, sino que se apoya en una aplicación de escritorio para las tareas administrativas. Tampoco proporciona ningún elemento de monitorización basado en métricas, siendo un sistema totalmente orientado al control de equipos.

2.2.9. NetSupport Manager

NetSupport Manager es un sistema de pago centrado en el soporte y control remoto de equipos informáticos. Opera tanto con la infraestructura de la propia organización como siendo un servicio en la nube, pudiendo ser empleado para múltiples sedes separadas físicamente. Cuenta con una versión especializada para la educación llamada NetSupport School.

Soporta el control remoto tanto para equipos de escritorio como para dispositivos móviles e integra un pequeño módulo de monitorización donde quedan registradas las características tanto hardware como software de cada uno de ellos. Sin embargo, dicha monitorización no almacena un histórico de cambios.

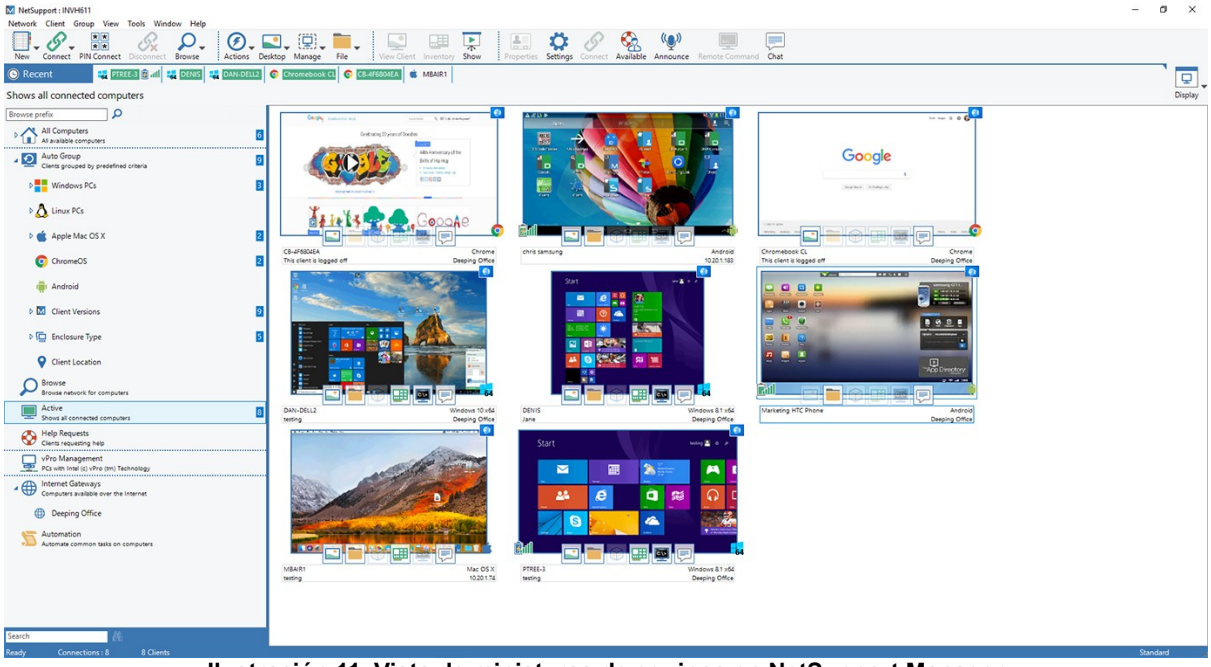

Ilustración 11. Vista de miniaturas de equipos en NetSupport Manager

Este sistema incluye un gestor de inventario básico con el que es posible definir habitaciones, reflejar los periféricos disponibles en cada puesto informático o facilitar la búsqueda anidada de un equipo dentro de la organización [32].

2.3. Comparativa de soluciones existentes

Después de introducir un conjunto de soluciones en el capítulo anterior, vamos a realizar una comparativa de éstas, no para confrontarlas y determinar cuál es la mejor o peor, sino para realizar una lectura sobre las virtudes de cada una y conocer su enfoque.

Amazon CloudWatch y Azure Monitor son sistemas de monitorización especializados en aplicaciones y servicios, no en equipamiento informático como tal. Están preconfigurados para recolectar métricas y registros de los servicios de suscripción de sus respectivas nubes, por lo que su puesta en marcha es inmediata. Si bien permiten recibir métricas desde sistemas externos [33], realmente el caso común es el contrario y sus métricas suelen exportarse hacia otros sistemas de monitorización. Esta situación no es anómala, los sistemas de monitorización especializados dan la posibilidad de enviar métricas a sistemas de monitorización más genéricos como Cisco Intersight Workload. Este último, además, sí se integra

con el equipamiento informático de la organización y permite obtener informes convergentes con los reportes de diferentes fuentes.

Por otro lado, Nagios, Zabbix y Pandora FMS son sistemas de monitorización que han sido pensados para la monitorización de equipos informáticos desde un inicio y son las soluciones que mejor se ajustan a la temática que estamos desarrollando. Sus versiones comunitarias son gratuitas y no están asociadas a ningún tipo de producto complejo que produzca dependencia con el proveedor. Por contra, añaden una considerable carga de trabajo al equipo de ingenieros de sistemas para la puesta en marcha y el mantenimiento de estos sistemas a lo largo del tiempo. Aquí entra en juego las habilidades comerciales de cada uno de ellos: Zabbix cuenta con distribuidores en todo el mundo para recibir servicios de consultoría, Nagios ofrece una versión comercial repleta de asistentes de configuración y Pandora FMS ofrece una solución empresarial en la nube similar a Amazon CloudWatch, Azure Monitor y Cisco Intersight Workload, donde uno accede a una plataforma de monitorización como servicio.

Sin embargo, en mayor o menor medida, todos ellos carecen de utilidades de gestión para equipos informáticos, centrándose únicamente en la observabilidad. Cuando hablamos de gestión de equipos, nos referimos a las capacidades que tienen sistemas como Veyon o NetSupport Manager para el control de los mismos. Lamentablemente, estos dos últimos no incorporan sistemas de monitorización lo suficientemente avanzados y sus utilidades de monitorización son poco prácticas. Pandora FMS resuelve este problema integrando una solución comercial de terceros para la gestión de equipos, eHorus, pero a fin de cuentas son dos productos diferentes funcionando conjuntamente.

GLPI nos ofrece un punto de vista diferente, pero complementario, para la definición anterior de gestión de equipos. En vez de pensar en tener el control de los equipos como una serie de acciones directas para, por ejemplo, reiniciarlos o bloquearles el acceso a internet, lo orienta en términos de documentación de la infraestructura de la organización, organización de inventario y tramitación de incidencias.

En resumen, cada uno de estos sistemas puede aportar un grano de arena a todos los demás. Huelga decir que, por sí solos, estos sistemas ya son bastante complejos y buscan satisfacer las necesidades de su base de usuarios, así que es natural que no profundicen en todos los aspectos que nosotros estamos evaluando.

2.4. Propuesta de solución

Nuestro objetivo es la realización de un prototipo de sistema informático para la gestión y monitorización de equipos informáticos accesible a través de la web. Durante el análisis preliminar, extrajimos algunas conclusiones y observamos que la mayoría de los sistemas de monitorización existentes flaquean en los aspectos relacionados con la gestión de los equipos monitorizados. Así pues, el principal nicho de mercado sobre el que nos moveremos será combinar los aspectos de gestión y monitorización sobre una única plataforma con ambos elementos integrados.

Orientaremos nuestra solución a una organización de pequeñas dimensiones; por ejemplo, para una biblioteca o una mediana empresa donde haya al menos un administrador de sistemas encargado del mantenimiento y supervisión de los equipos. Inicialmente, los equipos informáticos sobre los que nos centraremos serán ordenadores de sobremesa y portátiles con sistema operativo GNU/Linux, dejando de lado otros dispositivos igual de interesantes como son los dispositivos de electrónica de red (*switches*, *routers* y cortafuegos) o los dispositivos móviles.

La finalidad de la solución sigue siendo proporcionar una herramienta útil y atractiva para un administrador de sistemas en su día a día, convirtiéndolo en nuestro usuario objetivo. Es por ello que debemos considerar cuáles serían las características de la solución más relevantes para él. En este sentido, dada mi propia experiencia manejando sistemas similares en el ámbito laboral, puedo dar un punto de vista personal a la confección de las características de la solución.

Respecto a la gestión de equipos informáticos, las funcionalidades más interesantes que incorporaremos son el control remoto de los equipos mediante el envío de acciones como bloquear un equipo o reiniciar un equipo, el acceso remoto a la sesión gráfica o de consola y el inventariado de los equipos en las localizaciones

de la organización; por ejemplo, entre diferentes plantas o habitaciones considerando una única sede.

Por otro lado, la monitorización resulta un tema mucho más extenso y deberemos delimitar las características que desarrollaremos dentro de la solución. De hecho, las soluciones existentes vistas en el análisis preliminar invierten mucho esfuerzo en la monitorización para enriquecer la experiencia del administrador de sistemas y será difícil equipararse con uno al completo. En nuestro caso, nos centraremos en el almacenamiento y visualización de las métricas, pero no en el almacenamiento de registros ni en la configuración de alertas. Las consultas de monitorización las estableceremos ya prefabricadas de serie para agilizar la puesta en marcha, de modo que el administrador de sistemas no tenga que conocer cómo funciona por dentro el sistema ni tampoco involucrarse en aspectos avanzados del mismo.

En los siguientes capítulos realizaremos una búsqueda y selección de tecnologías y *frameworks* que nos ayuden a conseguir estos objetivos.

2.5. Búsqueda de tecnologías y *frameworks*

Desde el inicio, tenemos claro que el sistema a desarrollar trabajará con tecnologías web para la presentación de la información y su manejo por parte del usuario. Son varios los motivos que nos animan a tomar esta decisión, frente al desarrollo de una aplicación de escritorio. El primero de ellos es la portabilidad que proporcionan las tecnologías web para ser usadas desde cualquier dispositivo y sin restricciones de dependencias. Además, las actualizaciones y el mantenimiento quedan centralizados sobre la versión que decidamos otorgar al usuario. Aunque también tendremos algunas desventajas, ya que sacrificamos algo de rendimiento y los tiempos de respuesta de una aplicación web son ligeramente más altos en comparación a una aplicación de escritorio.

La mención de utilizar tecnologías web sólo cubre un segmento de las tecnologías implicadas en el sistema a desarrollar. En primer lugar, abordaremos los patrones de arquitectura de software sobre los que posteriormente definiremos una pila tecnológica.

El patrón de arquitectura modelo/vista/controlador, en adelante MVC, separa y modulariza una aplicación en tres componentes diferentes: los datos de la aplicación (modelo), la interfaz de usuario (vista) y la lógica de negocio (controlador) [34]. Durante su ciclo de vida, el usuario realiza una solicitud al controlador y éste recupera o modifica los datos del modelo para devolverlos al usuario mediante la composición de una vista. En la ilustración siguiente pueden observarse cada una de las interacciones realizadas sobre los componentes involucrados [35]. Gracias a este patrón, los datos de la aplicación quedan separados de la interfaz de usuario, facilitando, por ejemplo, la existencia de múltiples vistas basadas en los mismos datos.

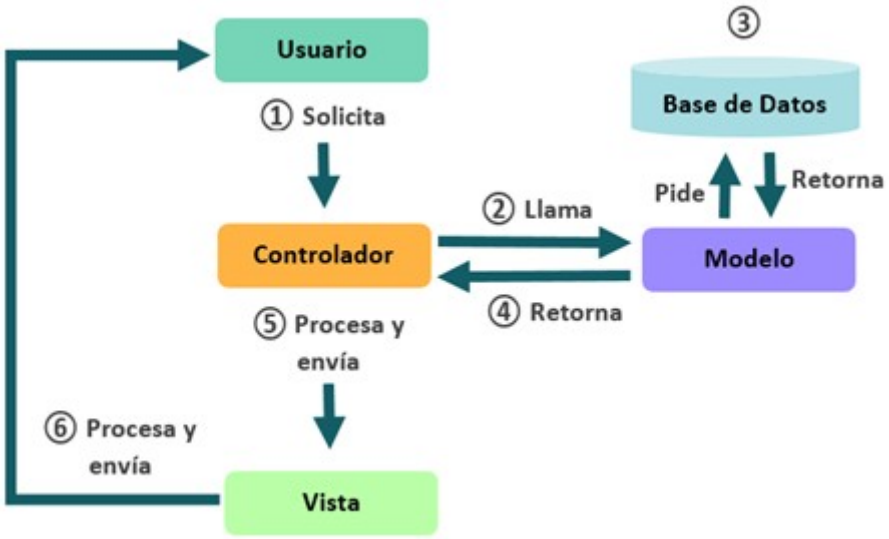

Ilustración 12. Ciclo de vida del modelo-vista-controlador

Debemos considerar que las aplicaciones web, a la hora de emplear el patrón MVC, siguen además el esquema de comunicación cliente-servidor. Ello origina cierta partición en el funcionamiento del controlador sobre la vista por parte del navegador, motivado por los enfoques de cliente ligero y cliente pesado [36]. Hablamos de cliente ligero cuando éste realiza el mínimo procesamiento y todo el esfuerzo para procesar modelo, vista y controlador recae sobre el servidor, mientras que hablamos de cliente pesado cuando éste procesa algún componente del patrón total o parcialmente. Disponemos de suficientes *frameworks* hoy en día que compatibilizan ambos enfoques.

El lado de la aplicación web que verá el cliente se conoce por el término *front-end*, mientras que el lado de la aplicación web que se ejecuta en un servidor se

conoce por el término *back-end*. Atendiendo al criterio anterior, para el desarrollo *front-end* encontramos *frameworks* para *front-end* orientados a ejecutar código en el navegador (cliente pesado) y *frameworks* para *front-end* orientados a trabajar como plantillas ejecutadas desde un servidor (cliente ligero).

Por otro lado, la naturaleza de este sistema involucra la interacción de los equipos informáticos para su control, recogida de métricas y almacenamiento de registros. Aunque la monitorización remota sigue también el esquema cliente-servidor, a nivel de arquitectura la elección entre quién es servidor y quién es cliente se define por una política *push* o una política *pull* [37]. Bajo una política *push*, los equipos inician la comunicación y envían las métricas generadas periódicamente a un recolector, mientras que bajo una política *pull*, el recolector inicia la comunicación y solicita métricas periódicamente a los equipos [38].

Una política *push* es especialmente útil para una configuración mínima y con un formato de mensaje preestablecido. Además, aporta mayor seguridad a los equipos al no obligarlos a recibir conexiones de red sobre ningún puerto de escucha. Sin embargo, añade latencia para disponer de su información a tiempo real y el servidor queda expuesto a sufrir ataques de denegación de servicio.

Una política *pull* ofrece como ventaja la aceptación de métricas desde dispositivos muy heterogéneos. Sin embargo, añade complejidad para dar soporte a diferentes formatos de mensaje y quizás no sea viable en infraestructuras con redes complejas.

Algunos sistemas de monitorización llegan a implementar ambas políticas, como es el caso del sistema de monitorización Zabbix estudiado en capítulos pasados. También podemos encontrar pequeñas utilidades llamadas *gateways* que actúan como punto intermedio para la conversión de una política a otra.

Otro aspecto que nos concierne es la elección de una base de datos apropiada para la pila de tecnologías web y para la pila de tecnologías de gestión y monitorización. Así, dependiendo de cada escenario, consideraremos una base de datos relacional, no relacional o de series temporales.

Una base de datos relacional almacena información entre un conjunto de tablas cuyas tuplas quedan enlazadas por las relaciones entre éstas, establecidas previamente siguiendo un modelo relacional. Los datos almacenados son altamente consistentes entre sí, ya que este tipo de base de datos satisfacen la integridad referencial de los mismos [39].

Una base de datos no relacional almacena información en colecciones cuyo tipo de dato es flexible a cambios. Dependiendo del propósito, encontramos cuatro variantes principales: orientada al almacenamiento de documentos, orientada al almacenamiento en columnas, orientada a la asociación de clave-valor y orientada a la construcción de grafos [40].

Una base de datos de series temporales optimiza el almacenamiento de información sujeta a una marca de tiempo y dispone de consultas especializadas para recuperar dicha información agregadamente en una escala temporal. Se sostiene tanto con implementaciones sobre base de datos relacionales como no relacionales [41].

De cara a las tecnologías web, los *frameworks* para *back-ends* suelen soportar tanto el acceso a bases de datos relacionales como no relacionales, mientras que las bases de datos de series temporales son el almacenamiento comúnmente utilizado por la mayoría de sistemas de monitorización.

A continuación, realizaremos un breve recorrido por varias tecnologías y *frameworks* que nos permitan establecer nuestra pila tecnológica. Como última sección dentro de este capítulo, también incluiremos tecnologías y *frameworks* complementarios que nos puedan ser de utilidad en algún momento en las siguientes etapas. En cualquier caso, la mayoría quedarán descartados en el siguiente capítulo, una vez que discutamos la viabilidad de su incorporación en el proyecto.

2.5.1. *Frameworks* para la implementación de *front-ends* de cliente ligero

Los siguientes *frameworks* para *front-end* de cliente ligero nos proporcionarán las facilidades para desarrollar una interfaz de usuario web mediante plantillas procesadas desde un servidor remoto.

- Thymeleaf

Thymeleaf es un motor de plantillas orientado a tecnologías web para el lenguaje de programación Java que permite generar documentos HTML, XML, ficheros de Javascript con extensión js, ficheros de hoja de estilos con extensión css y ficheros de texto plano. Su primera versión fue liberada el 17 de julio de 2011. Puede integrarse como un módulo dentro del *framework* Spring MVC. Cuenta con una licencia Apache 2.0 [42].

- Apache FreeMarker

Apache FreeMarker es un motor de plantillas multipropósito para el lenguaje de programación Java que permite generar documentos HTML, emails, ficheros de configuración e incluso ficheros de código fuente con plantillas basadas en el lenguaje FTL (FreeMarker Template Language) [43]. Su primera versión fue liberada el 18 de abril de 2002 [44]. Cuenta con licencia Apache 2.0.

- Mustache Template Engine

Mustache Template Engine es un motor de plantillas multipropósito y está soportado por varios lenguajes de programación como Java, Python, Ruby, PHP y Javascript. Sin embargo, no implementa lógica de control ni de iteración, realizando únicamente sustituciones de las etiquetas especificadas [45]. Su primera versión fue liberada el 5 de octubre de 2009. Cuenta con una licencia MIT.

- Jinja

Jinja es un motor de plantillas orientado a tecnologías web para el lenguaje de programación Python que permite generar documentos HTML, con soporte para

internacionalización y prevención de ataques de inyección de código. Puede integrarse como un módulo dentro de los *frameworks* Flask, Django y FastAPI. Su primera versión fue liberada el 17 de julio de 2008. Cuenta con una licencia BSD v3 [46].

2.5.2. *Frameworks* para la implementación de *front-ends* de cliente pesado

Los siguientes *frameworks* para *front-end* de cliente pesado nos proporcionarán las herramientas para el desarrollo de una interfaz de usuario web donde la lógica de la interfaz la maneja y procesa el navegador web del cliente.

- Angular

Angular es un *framework* para el desarrollo de aplicaciones web y móviles en el cliente basado en componentes. Es promovido por Google. Destaca por poseer librerías que le confieren inyección de dependencias, normas de diseño, enrutamiento y validadores de entrada. Utiliza el lenguaje de programación Typescript. Su primera versión fue liberada el 14 de septiembre de 2016. Cuenta con una licencia MIT [47].

- React

React es un *framework* para el desarrollo de aplicaciones web en el cliente basado en componentes. Es promovido por Facebook. Sobre móviles, dispone de un *framework* especializado llamado React Native. Utiliza el lenguaje de programación Javascript para la lógica de control y la extensión JSX para la escritura de componentes de un modo familiar a la maquetación con HTML. Admite la renderización de la vista desde un servidor de Node.js, lo que lo convierte tanto en cliente ligero como en cliente pesado. Su primera versión fue liberada el 29 de mayo de 2013. Cuenta con una licencia MIT [48].

- Vue

Vue es un *framework* ligero para el desarrollo de aplicaciones web en el cliente. Utiliza el lenguaje de programación Javascript e incorpora características concretas

que suelen ser suficientes para la mayoría de escenarios como son el manejo de eventos, animaciones y transiciones, propiedades computadas y enlace de datos sobre los componentes de la vista. Su primera versión fue liberada el 3 de febrero de 2014. Cuenta con una licencia MIT [49].

2.5.3. *Frameworks* para la implementación de *back-ends* modelo/vista/controlador

Los siguientes *frameworks* para *back-end* modelo/vista/controlador nos permitirán coordinar los diferentes componentes usados a lo largo del sistema y nos otorgarán la capacidad de desarrollar funcionalidades únicas.

- Spring Framework

Spring es un *framework* para el desarrollo de aplicaciones empresariales. Utiliza el lenguaje de programación Java, aunque también se encuentra disponible para otros lenguajes como Kotlin y Groovy. Se trata de un *framework* modular, cuya base ofrece inversión de dependencias, sistema de anotaciones siguiendo el patrón decorador y un ciclo de vida para los objetos del propio *framework*. Mediante otros módulos, podemos conseguir abstracción sobre la base de datos utilizada, validación del modelo de datos, autenticación y autorización de usuarios y recursos, soporte para la realización de pruebas unitarias o documentación del código desarrollado. Uno de los módulos más relevantes para nosotros es el módulo de Spring MVC para el desarrollo de aplicaciones web en el servidor.

Su primera versión fue liberada el 1 de octubre de 2002. Para facilitar la configuración de un proyecto de Spring, el 1 de abril de 2014 nació el proyecto Spring Boot. Cuenta con una licencia Apache 2.0 [50].

- ASP.NET

ASP.NET es un *framework* para el desarrollo de aplicaciones empresariales. Utiliza el lenguaje de programación C#, aunque también se encuentra disponible para otros lenguajes como F# o VB.NET. El módulo ASP.NET Core MVC incorpora enrutamiento, vinculación y validación de datos del modelo, inversión de

dependencias, filtros y un motor de vistas para cliente ligero llamado Razor [51]. Su primera versión fue liberada el 5 de enero de 2002. Antiguamente utilizaba una licencia Apache 2.0, pero desde el 20 de julio de 2021 quedó actualizada a una licencia MIT [52].

- FastAPI

FastAPI es un *framework* ligero para el desarrollo de aplicaciones web en el servidor. Utiliza el lenguaje de programación Python. Su modelo de consumo de peticiones no está basado en hilos de ejecución, sino en corrutinas para proporcionar un mayor rendimiento en cada hilo. Delega las necesidades del usuario a módulos externos, así que para una aplicación MVC completa se necesita a SQLAlchemy para el acceso al modelo de datos, Pydantic para la validación del modelo de datos, OpenAPI para la documentación de código y Jinja para la maquetación de la vista, entre otros. Su primera versión fue liberada el 5 de diciembre de 2018. Cuenta con una licencia MIT [53].

2.5.4. Base de datos relacionales

Las siguientes bases de datos relacionales nos otorgarán un almacén de información para el sistema siguiendo un modelo de datos relacional. Con ellas podremos almacenar información lógica del sistema como, por ejemplo, los equipos registrados y otras características de los mismos.

- MariaDB

MariaDB es un gestor de bases de datos relacionales. Proviene de una bifurcación de MySQL, después de que éste fuera adquirido por Sun Microsystems en el año 2009. Soporta los mecanismos de almacenamiento tradicionales de MySQL, MyISAM e InnoDB, además de la implementación de los suyos propios. Aria y XtraDB. Su primera versión fue liberada el 22 de enero de 2009. Cuenta con una licencia GNU GPL v2 [54].

- PostgreSQL

PostgreSQL es un gestor de bases de datos relacionales. Soporta el almacenamiento de vectores, estructuras clave-valor y tipos definidos por el usuario. También soporta la creación de vistas materializadas [55]. Su primera versión fue liberada el 5 de septiembre de 1995. Cuenta con una licencia propia llamada PostgreSQL License, similar a otras licencias permisivas como MIT o BSD.

2.5.5. Base de datos no relacionales

Las siguientes bases de datos relacionales nos otorgarán un almacén de información para el sistema siguiendo un modelo de datos no relacional. Con ellas podremos almacenar información lógica del sistema, como por ejemplo los equipos registrados y otras características de los mismos.

- MongoDB

MongoDB es un gestor de bases de datos no relacionales orientada al almacenamiento de documentos. Proporciona mecanismos para ser escalable como la replicación de los datos en servidores secundarios o la partición de grandes volúmenes de datos entre diferentes servidores. Su primera versión fue liberada el 11 de febrero de 2009. Antiguamente utilizaba una licencia GNU AGPL v3.0, pero desde el 16 de octubre de 2018 quedó actualizada a una licencia SSPL [56].

- Redis

Redis es un gestor de bases de datos no relacionales orientada al almacenamiento de asociaciones clave-valor. Mantiene la base de datos completa en memoria principal, por lo que suele usarse como sistema de caché volátil. También se emplea para el intercambio de mensajes y el almacenamiento del estado en aplicaciones distribuidas. Su primera versión fue liberada el 10 de mayo de 2009. Cuenta con una licencia BSD v3 [57].

2.5.6. Bases de datos de series temporales

Las siguientes bases de datos de series temporales nos permitirán implementar el almacenamiento de las métricas enviadas al sistema a lo largo del tiempo por diferentes equipos. Son especialmente útiles dado que son bases de datos especializadas en el acceso y agregación de datos con marcas temporales.

- InfluxDB

InfluxDB es un gestor de bases de datos para series temporales sustentado por una implementación no relacional. Sigue una política *push* para la recepción de medidas, series y puntos. Dispone de un lenguaje de consulta propio llamado Flux para el procesamiento de los datos agrupados en el tiempo. Su primera versión fue liberada el 24 de septiembre de 2013. Cuenta con una licencia MIT [58].

- Prometheus

Prometheus es un gestor de bases de datos para series temporales orientado a la monitorización y sustentado por una implementación no relacional. Sigue una política *pull* para consultar y almacenar métricas dentro de sí mismo o hacia otras bases de datos para series temporales remotas. Dispone de un lenguaje de consulta propio similar a SQL llamado PromQL. Su primera versión fue liberada el 26 de enero de 2015. Cuenta con una licencia Apache 2.0 [59].

- TimescaleDB

TimescaleDB es un gestor de bases de datos para series temporales sustentado por una implementación relacional bajo PostgreSQL. Sigue una política *push* para la recepción de medidas, series y puntos. Usa los comandos comunes de SQL para realizar consultas y permite correlacionar métricas a través de operaciones de unión. Su primera versión fue liberada el 1 de noviembre de 2018. Cuenta con una licencia Apache 2.0 [60].

2.5.7. Tecnologías y *frameworks* complementarios

Las siguientes tecnologías y *frameworks* cubren aspectos relevantes dentro del sistema a desarrollar para tareas específicas en los procesos de gestión y monitorización como pueden ser la recopilación y consulta de datos, comunicación entre componentes, integraciones con sistemas de terceros o el mismo despliegue del sistema.

- Grafana

Grafana es una aplicación que permite visualizar consultas de series temporales mediante tecnologías web. Además, sobre estas consultas permite establecer alertas y reportes. Se integra tanto con bases de datos como con plataformas de monitorización externas. Dispone de un extenso catálogo de paneles prediseñados por la comunidad. Su primera versión fue liberada el 21 de enero de 2014. Cuenta con una licencia GNU AGPL v3.0 [61].

- Chronograf

Chronograf es una aplicación que permite visualizar consultas de series temporales mediante tecnologías web. Se centra en el acceso a bases de datos InfluxDB y permite la administración remota de las instancias. Su primera versión fue liberada el 14 de noviembre de 2016. Cuenta con una licencia GNU AGPL v3.0 [62].

- Plotly

Plotly es una biblioteca para el renderizado de gráficas básicas, estadísticas, científicas y geométricas, sumando en total más de 80 tipos de gráficas diferentes. Se encuentra disponible tanto para aplicaciones matemáticas como para lenguajes de programación web. Su primera versión para Javascript fue liberada el 15 de noviembre de 2015. Cuenta con una licencia MIT [63].

- Pushgateway

Pushgateway es una aplicación intermedia que permite almacenar métricas recibidas mediante una política *push* y que posteriormente son consumidas por una

política *pull*. Esta aplicación es ideal en sistemas de monitorización que únicamente soportan políticas *pull*, pero éstas no pueden ser implementadas sobre todos los dispositivos. Su primera versión fue liberada el 5 de mayo de 2015. Cuenta con una licencia Apache 2.0 [64].

- SNMP

SNMP es un protocolo de comunicación para la monitorización de equipos informáticos en la red como *routers*, *switches*, servidores o impresoras. Los dispositivos ejecutan un agente que expone el puerto por defecto 161/UDP siguiendo una política de monitorización *pull*, donde un sistema de monitorización recoge e identifica los objetos recibidos mediante bases de identificadores de referencia [65].

- Osquery

Osquery es un *framework* que permite consultar información sobre un equipo informático mediante un lenguaje de consulta independiente de la plataforma de ejecución. Integra más de 250 verbos que nos permiten obtener el tiempo de encendido, carga de CPU o uso de memoria principal entre otros. Principalmente orientado a políticas *pull*, dispone de adaptadores a los lenguajes de programación Python y Go. Su primera versión fue liberada el 16 de octubre de 2014. Cuenta con una licencia múltiple entre Apache 2.0 y GNU GPL v2 [66].

- QEMU

QEMU es un emulador de aplicaciones y una herramienta de virtualización de hardware que, entre otras opciones, permite ejecutar un sistema operativo dentro de una máquina virtual. De este modo, se puede reaprovechar una misma máquina física para ejecutar varios sistemas operativos simultáneamente. Su primera versión fue liberada el 2 de diciembre de 2011. Cuenta con una licencia GNU GPL v2 [67].

- Bootstrap

Bootstrap es un *framework* para el desarrollo de interfaces de usuario en aplicaciones web. Distribuye el espacio en una cuadrícula divisible en filas y columnas. Incorpora un conjunto de hojas de estilo, animaciones, formularios, iconos

y componentes propios, siendo el diseño consistente visualmente entre todos ellos. Además, la interfaz se adapta a las dimensiones de pantalla del dispositivo donde se visualice, ya sea un dispositivo de escritorio o móvil. Su primera versión fue liberada el 19 de agosto de 2011. Cuenta con una licencia MIT [68].

- SDL

SDL es un *framework* para el desarrollo de aplicaciones gráficas en 2D principalmente utilizado dentro de la industria de los videojuegos. Está diseñado para el acceso a bajo nivel de los subsistemas de video, sonido y periféricos de un equipo. Además, soporta oficialmente los sistemas operativos Windows, Mac OS X, Linux y Android. Su primera versión fue liberada el 6 de diciembre de 1999. Cuenta con una licencia zlib [69].

- ActiveMQ

ActiveMQ es un sistema de mensajería basado en el protocolo AMQP. Proporciona un sistema de distribución e intercambio de mensajes y colas asíncronas. Pueden participar clientes desde aplicaciones escritas en los lenguajes de programación Javascript, Python o Java. Su primera versión fue liberada el 1 de febrero de 2007. Cuenta con una licencia Apache 2.0 [70].

- OpenLDAP

LDAP es un protocolo de aplicación que nos permite acceder a los servicios de directorio de una organización. En un servidor de directorio, los recursos, servicios y usuarios son objetos que forman una jerarquía. Los sistemas externos pueden consultar esta jerarquía para implementar ciertas funcionalidades como la identificación de usuarios. OpenLDAP implementa este protocolo y trabaja sobre los puertos 1389/TCP y 1636/TCP para comunicación cifrada. Cuenta con una licencia propia llamada OpenLDAP Public License, similar a otras licencias permisivas como MIT o BSD [71].

- Docker

Docker es una herramienta para facilitar el empaquetamiento y distribución de aplicaciones de consola junto con las librerías de las que éstas dependen. Permite la construcción de imágenes y la ejecución de contenedores siguiendo el estándar OCI. Además, aísla la aplicación sobre los recursos físicos del anfitrión al mismo tiempo que limita el aprovechamiento de los mismos. Su primera versión fue liberada el 20 de marzo de 2013. Cuenta con una licencia Apache 2.0 [72].

- Docker Compose

Docker Compose es una herramienta para la orquestación de contenedores de una aplicación a partir de la definición de un documento YAML. Dentro de dicho documento, gestiona los recursos de la aplicación como redes internas, variables de entorno o espacios de almacenamiento que, de otra manera, requirirían una configuración manual. Antes de la versión 2 era distribuido independientemente a Docker, pero en la actualidad viene incluido de serie. Su primera versión fue liberada el 16 de octubre de 2014. Cuenta con una licencia Apache 2.0 [73].

- Caddyserver

Caddyserver es un servidor web, balanceador de carga y *proxy* reverso que proporciona facilidades de configuración como la activación por defecto y renovación automática de certificados de navegación segura mediante el protocolo ACME. El proyecto se trata de una iniciativa de la empresa ZeroSSL, quien además ofrece estos mismos certificados de navegación gratuitamente. Su primera versión fue liberada el 28 de abril de 2015. Cuenta con una licencia Apache 2.0 [74].

- Nginx

Nginx es un servidor web, balanceador de carga y proxy reverso caracterizado por su rendimiento en el manejo de conexiones concurrentes y su ligereza en cuanto a demanda de CPU y memoria principal. Su primera versión fue liberada el 4 de octubre de 2004. Cuenta con una licencia BSD v2 [75].

2.6. Selección de tecnologías y *frameworks*

Después de haber abordado las tecnologías y *frameworks* que pueden sernos útiles para la implementación de nuestro propio sistema de gestión y monitorización, queda resolver cuáles vamos a utilizar finalmente y de qué modo.

Aún en este punto, todavía tenemos dudas sobre la elección de trabajar con un *framework* para cliente web ligero o pesado. Lo atractivo de un *framework* para cliente web ligero desde el punto de vista del programador es su liviana curva de aprendizaje. Sin embargo, un cliente web pesado nos permitiría construir aplicaciones web mucho más complejas a largo plazo. Aunque aún no hayamos definido cuántas funcionalidades tendrá nuestro sistema, sí podemos prever que será una cantidad considerable si tenemos en cuenta otros sistemas de monitorización vistos previamente. Así pues, descartaremos los *frameworks* para clientes web ligeros.

Dicho esto, la elección entre los *frameworks* para clientes web queda reducido a Angular, React y Vue. Elegiremos Angular, por encima de React y Vue, dada su amplia cantidad de funcionalidades incorporadas en el propio código base del *framework*. Lamentablemente, tanto React como Vue nos obligarían a agrandar nuestra pila tecnológica con otras librerías de terceros para suplir sus carencias y queremos evitar esa situación.

Sobre los *frameworks* para *back-end*, lo primero que debemos considerar es la interacción que tendrán éstos con la vista. Al elegir un *framework* como Angular, el código del cliente web carga independientemente del código del controlador. Al existir esta partición, la comunicación entre el controlador y la vista debe realizarse mediante alguna interfaz de comunicación, como una API REST, para la transmisión de los datos.

Una API REST es un servicio de comunicación que, apoyándose en el protocolo de comunicación web HTTP, envía y recibe información en un formato de intercambio de datos como JSON. Esto no supone ningún problema para los tres *back-ends* propuestos durante la búsqueda, ya que todos ellos soportan esta arquitectura de comunicación. Sus controladores pueden actuar como un servicio

REST que proporcionan mensajes en formato JSON para la vista de un cliente web pesado, en vez de generar por sí mismos la vista mediante el motor de plantillas de algún cliente web ligero.

El primero de los *back-ends* que descartamos es FastAPI, ya que, si bien es bastante adecuado por el rendimiento que ofrece, está ideado para servicios de menor tamaño o con funcionalidades mucho más concretas. Ahora, la decisión queda entre elegir ASP.NET o Spring. Esto último lo fundamentaremos más por las capacidades del equipo de desarrollo que por aspectos técnicos, ya que ambos son similares en cuanto a modularidad, rendimiento y complejidad. En este caso, el equipo de trabajo lo conformo yo, un estudiante de la rama de tecnologías de la información que conoce Spring por medios autodidactas, mientras que ASP.NET sólo lo conozco por recomendación sin haber indagado profundamente. Por lo tanto, la elección final para el *back-end* será Spring.

Respecto de la base de datos, hemos presentado tanto modelos relaciones como no relacionales, así que primero tenemos que despejar la duda de cuál vertiente usaremos en el modelo de datos de nuestro sistema. Para comenzar, lo que necesitamos representar son, a priori, un registro de dispositivos y otro de ubicaciones. Aunque dichos registros crezcan y adquieran un nivel de detalle mayor en el futuro, serán datos estables una vez sean introducidos por primera vez, cuyas modificaciones en el modelo responderán a cambios posteriormente introducidos en el sistema y no a cambios debidos a la naturaleza de los datos. Así pues, una base de datos relacional se ajusta mejor en finalidad al enunciado anterior. Por otro lado, para nuestro caso no necesitamos ninguna característica especial ni de MariaDB ni de PostgreSQL, así que ambas opciones son perfectamente válidas. Sin embargo, examinando sus licencias comprobamos que la de PostgreSQL resulta más permisiva, así que finalmente nos decantaremos por ésta.

La pila tecnológica para las tecnologías web ha quedado definida con Angular como *front-end* de cliente web, Spring como *back-end* de un servicio API REST y PostgreSQL como base de datos.

Ahora vamos a definir por primera vez cuál será nuestra política de monitorización. Como vimos en el capítulo anterior, podemos optar por una política

push, *pull* o ambas. Dado el carácter de este proyecto como prototipo, sólo elegiremos una de las dos. La política *pull* es la más interesante de las dos porque es la que nos permite integrarnos con una mayor cantidad de dispositivos diferentes. Sin embargo, como nos encontramos aún en las primeras etapas del análisis y definición del proyecto, aún no tenemos en mente ofrecer un grado de compatibilidad tan alto. Por lo tanto, una política *push* es más que adecuada para nuestras necesidades. En los equipos informáticos implementaremos un agente que extraiga métricas y las envíe a nuestro sistema de monitorización. Además, dicho agente también puede ser el encargado de recibir los comandos de gestión y control enviados desde el sistema por el usuario.

Las bases de datos de series temporales para una arquitectura *push* que podemos emplear son InfluxDB y TimescaleDB. Entre estas dos, TimescaleDB ofrece un lenguaje de consulta conocido, SQL, y además trabaja tecnológicamente sobre el mismo motor de base de datos que pretendemos utilizar para el *back-end*, PostgreSQL. Por otro lado, con InfluxDB tendríamos que aprender a utilizar su propio lenguaje de consultas Flux, que desconocemos.

En un inicio, TimescaleDB resulta más atractivo sobre el papel, ya que nos concedería la oportunidad de unir en una única base de datos todos los datos que va a manejar la aplicación, tanto del modelo de datos como de las métricas de monitorización de los equipos. Esto es posible gracias a que TimescaleDB se trata, en realidad, de una extensión instalable sobre cualquier instancia ordinaria de PostgreSQL. Sin embargo, dispone de una pobre interoperabilidad hacia la mayoría de lenguajes de programación y *frameworks*. Llegado al punto de integrarlo con nuestro *back-end*, Spring Framework, no disponemos ni oficialmente ni comunitariamente de un dialecto para interpretar TimescaleDB como un objeto abstracto; así que, o bien implementamos un dialecto nosotros mismos o utilizamos una conexión directa a base de datos escribiendo las sentencias SQL artesanalmente como sugiere su documentación oficial. Ambas fórmulas son inviables, ya que nos supondría una inversión considerable de tiempo.

Afortunadamente, InfluxDB sí cuenta con las integraciones necesarias dentro de Spring Framework para ofrecernos un objeto abstracto con el que interactuar de

manera amena y bajo el propio estilo del *framework*. Sin ningún otro impedimento, elegiremos InfluxDB.

El resto de tecnologías que involucraremos en el sistema atenderán a la demanda de funcionalidades particulares. Por ejemplo, Plotly nos permitirá renderizar las consultas de series temporales en gráficas, siendo descartados Grafana y Chronograf al encontrarnos impedimentos para embeber las gráficas con ellos dentro de nuestra aplicación web. También descartaremos Pushgateway y el protocolo SNMP al involucrar una arquitectura *pull*.

Por otro lado, encontramos que Spring Framework cuenta con su propio sistema de colas llamado SimpleBroker, por lo que no necesitaremos una solución tan avanzada como ActiveMQ. Sobre Bootstrap, seguramente también lo descartaremos en favor de Angular Material, incluido dentro de Angular. Por otro lado, sí usaremos OpenLDAP para la autenticación de usuarios, SDL para mostrar mensajes personalizados en una sesión gráfica, QEMU para la ejecución de máquinas virtuales que simulen ser equipos en el sistema, Osquery para la obtención de métricas, Docker para el empaquetamiento de aplicaciones, Docker Compose para la orquestación de servicios, Nginx como servidor web de los recursos estáticos para el *front-end* y Caddyserver como *proxy* reverso del sistema en producción.

Finalmente, y como hemos visto durante el capítulo, el sistema que pretendemos desarrollar no se trata únicamente de una plataforma web limitada a trabajar con tecnologías web, sino que necesitaremos incluir una amplia variedad de otro tipo de tecnologías y *frameworks* adicionales para proveer de sentido a dicha plataforma.

2.7. Historias de usuario

Para el desarrollo de este proyecto vamos a seguir una metodología ágil SCRUM. Un componente fundamental dentro de esta metodología son las historias de usuario, las cuales nos van a ayudar a expresar las diferentes funcionalidades del sistema con un valor para el usuario final [76]. Seguiremos la siguiente plantilla para su composición:

- **ID**: identificador numérico único que asignaremos a cada historia de usuario.

- **Título**: resumen sobre el contenido de la historia de usuario.

- **Descripción**: formulación de la historia de usuario mediante la frase "como <persona>, quiero <cierta característica> para <lograr un beneficio concreto>".

- **Estimación**: estimación del esfuerzo teórico para la implementación de cada historia de usuario utilizando como unidad de medida los puntos de historia.

- **Criterios de aceptación**: conjunto de pruebas que impondremos a cada historia de usuario para validarla.

Durante la elaboración de las historias de usuario, utilizaremos un lenguaje no técnico para que sean entendibles tanto por el equipo de desarrollo como por el cliente. En este caso, la naturaleza del proyecto es un trabajo fin de grado donde yo mismo desempeñaré los roles de equipo de desarrollo y usuario, pero considerando como usuario objetivo un administrador de sistemas.

Como se mencionó anteriormente, a cada historia de usuario le asignaremos puntos de historia que expresarán el esfuerzo estimado que nos supone su realización. Dichos puntos no son una estimación temporal, sino que serán una referencia que nos ayudará a entender el tamaño y complejidad de una historia de usuario frente a otra, así como de su riesgo e incertidumbre. Utilizaremos la serie de Fibonacci (1, 2, 3, 5, 8, 13, 21) como uno de los métodos más populares dentro de SCRUM para la asignación de los puntos de historia [77].

Posteriormente, cuando lleguemos al siguiente capítulo, abordaremos su priorización y descomposición en tareas para conformar la pila del producto.

A continuación, presentaremos las historias de usuario divididas en dos tablas diferentes, por puntos de historia y por criterios de aceptación, para facilitar su lectura.

- Puntos de historia

ID	Título	Descripción	Puntos de historia
1	Iniciar sesión	• Como administrador, quiero iniciar sesión para acceder a la aplicación web	3
2	Cerrar sesión	• Como administrador, quiero cerrar sesión para abandonar la aplicación web	1
3	Visualizar ubicaciones	• Como administrador, quiero visualizar las ubicaciones de la organización para consultarlas o modificarlas	5
4	Crear una ubicación	• Como administrador, quiero crear una ubicación para reflejar una habitación, planta o edificio de la organización	5
5	Modificar una ubicación	• Como administrador, quiero modificar una ubicación para cambiar su título o alguna de sus propiedades	3
6	Asignar una ubicación dentro de otra ubicación	• Como administrador, quiero asignar una ubicación dentro de otra ubicación para construir una jerarquía de ubicaciones que representen mi organización	5
7	Eliminar una ubicación	• Como administrador, quiero eliminar una ubicación si me he equivocado al crearla o ya no pertenece a la organización	2
8	Visualizar equipos informáticos	• Como administrador, quiero visualizar los equipos informáticos de la organización para consultarlos o modificarlos	5
9	Registrar un equipo informático	• Como administrador, quiero registrar un equipo informático para inventariarlo	5

10	Modificar un equipo informático	• Como administrador, quiero modificar un equipo informático para cambiar alguna de sus propiedades	3
11	Eliminar un equipo informático	• Como administrador, quiero eliminar un equipo informático para dejar de inventariarlo	2
12	Asignar un equipo informático a una ubicación	• Como administrador, quiero asignar un equipo informático a una ubicación para agruparlo en ella	2
13	Desasignar un equipo informático de una ubicación	• Como administrador, quiero desasignar un equipo informático de una ubicación para mantenerlo al margen	1
14	Visualizar tablero	• Como administrador, quiero visualizar un tablero con información resumida de monitorización de todo el sistema para detectar situaciones anómalas	8
15	Visualizar los equipos informáticos sin ubicación	• Como administrador, quiero visualizar los equipos informáticos sin una ubicación asignada para moverlos a una ubicación efectiva	2
16	Consultar el estado de uso en tiempo real de un equipo informático	• Como administrador, quiero consultar el estado de uso en tiempo real de un equipo informático para conocer su rendimiento actual	8
17	Consultar el histórico de uso de un equipo informático	• Como administrador, quiero consultar el estado de uso en el pasado de un equipo informático para conocer su rendimiento anterior	8
18	Bloquear un ordenador remotamente	• Como administrador, quiero bloquear la sesión de usuario de un ordenador remotamente para habilitar o deshabilitar su uso	8

19	Reiniciar un ordenador remotamente	• Como administrador, quiero reiniciar un ordenador remotamente para resolver incidencias	3
20	Acceder al escritorio remoto de un ordenador	• Como administrador, quiero acceder al escritorio remoto de un ordenador para interactuar junto con el usuario	8
21	Visualizar la ventana actual de varios ordenadores	• Como administrador, quiero visualizar una malla de miniaturas con la ventana actual de varios ordenadores para supervisar su uso correcto	13
22	Gestionar notas sobre un equipo informático	• Como administrador, quiero añadir notas compartidas para registrar en ellas algún tipo de aviso	1
23	Visualizar las alertas configuradas	• Como administrador, quiero visualizar las alertas establecidas sobre los equipos informáticos en una única vista para interactuar con ellas fácilmente	5
24	Añadir una alerta sobre un equipo informático	• Como administrador, quiero establecer una alerta sobre un equipo informático concreto cuando se presente una situación anómala para enterarme antes de que suponga un percance para algún miembro de la organización	8
25	Modificar una alerta sobre un equipo informático	• Como administrador, quiero modificar una alerta sobre un equipo informático cuando para ajustar sus parámetros a otros más adecuados	5
26	Eliminar una alerta sobre un equipo informático	• Como administrador, quiero eliminar una alerta sobre un equipo informático concreto cuando ya no siga siendo relevante para mí	2

27	Consultar los registros de un equipo informático	• Como administrador, quiero consultar los registros de arranque y uso de un equipo informático para analizar un mal funcionamiento del mismo	8
28	Impresión de código QR identificativo	• Como administrador, quiero imprimir una pegatina con un código QR que asocie los equipos informáticos con el sistema para disponer de sus características escaneando el código QR desde el móvil	5
29	Detección de equipos informáticos en la red	• Como administrador, quiero detectar los equipos informáticos en la red sin registrar para identificar los que aún se encuentren ausentes en el sistema	8

Tabla 1. Asignación de puntos de historia

- Criterios de aceptación

ID	Título	Criterio de aceptación
1	Iniciar sesión	• Con unas credenciales válidas, produce la identificación del usuario en el sistema y lo redirige a la vista principal • Con unas credenciales inválidas, el sistema notifica al usuario que sus credenciales son erróneas
2	Cerrar sesión	• Después de cerrar la sesión, debe producirse una redirección a la vista de inicio de sesión
3	Visualizar ubicaciones	• Se encuentran todas las ubicaciones almacenadas en la base de datos
4	Crear una ubicación	• La ubicación debe quedar almacenada en la base de datos • La ubicación debe ser recuperable después de su creación

5	Modificar una ubicación	• Los cambios deben quedar almacenados en la base de datos • La ubicación debe ser recuperable después de su modificación
6	Asignar una ubicación dentro de otra ubicación	• Las asignaciones realizadas por el usuario deben conjugar una estructura de grafo dirigido acíclico para ser aceptadas • Si la estructura no es un grafo dirigido acíclico, el sistema notifica al usuario que no es posible la asignación
7	Eliminar una ubicación	• Las ubicaciones descendientes, si las hubiese, se convierten en ubicaciones de inicio. • Los equipos informáticos directamente asignados a la ubicación, si los hubiese, quedarían desasignados • La ubicación debe quedar eliminada de la base de datos
8	Visualizar equipos informáticos	• Se encuentran todos los equipos informáticos almacenados en la base de datos
9	Registrar un equipo informático	• Si los datos introducidos sobre el equipo informático se encuentran repetidos, el sistema notifica al usuario del conflicto • El equipo informático debe ser recuperable después de su creación
10	Modificar un equipo informático	• Los cambios deben quedar almacenados en la base de datos • El equipo informático debe ser recuperable después de su modificación

11	Eliminar un equipo informático	• El equipo informático elimina las relaciones con otras entidades del sistema como, por ejemplo, ubicaciones • La información de monitorización del equipo informático permanece en el sistema, accesible desde datos agrupados • El equipo informático queda eliminado de la base de datos
12	Asignar un equipo informático a una ubicación	• Si el equipo ya se encuentra asignado a una ubicación, reasigna el equipo a la nueva • En la base de datos se establecen las relaciones entre equipo y ubicación
13	Desasignar un equipo informático de una ubicación	• En la base de datos se eliminan las relaciones entre equipo y ubicación
14	Visualizar tablero	• Los datos mostrados deben figurar acordes al intervalo temporal especificado
15	Visualizar los equipos informáticos sin ubicación	• El sistema debe mostrar únicamente los equipos informáticos que no se encuentren asignados a ninguna ubicación
16	Consultar el estado de uso en tiempo real de un equipo informático	• Los datos deben pertenecer al equipo informático consultado • Los datos visualizados deben ser inferiores a 5 minutos
17	Consultar el histórico de uso de un equipo informático	• Los datos deben pertenecer al equipo informático consultado • Los datos pueden tener valores por defecto, por ejemplo, para interpolar dos muestras • El intervalo de la muestra y el intervalo de fechas son configurables por el usuario • La ausencia de métricas debe ser distinguible de datos inválidos

18	Bloquear un ordenador remotamente	• Si el ordenador está encendido, debe bloquear la sesión del usuario inmediatamente • Si el ordenador está apagado, en el próximo encendido debe aparecer bloqueado • Desde el propio ordenador no será posible desbloquearlo
19	Reiniciar un ordenador remotamente	• Si el ordenador está encendido, éste se reinicia inmediatamente
20	Acceder al escritorio remoto de un ordenador	• Si el ordenador está encendido y es posible realizar la conexión remota, el sistema habilita una ventana que refleja la pantalla del ordenador remotamente • Si el ordenador está encendido, pero no es posible realizar la conexión remota, el sistema notifica un error al usuario indicando que no está disponible • Si el ordenador está apagado, el sistema notifica un error al usuario indicando que no es posible realizar la operación
21	Visualizar la ventana actual de varios ordenadores	• Debe ser distinguible la miniatura de un ordenador que no puede mostrar su ventana respecto de la de un ordenador que tiene su pantalla en negro • Las miniaturas mostradas deben corresponder a los ordenadores seleccionados
22	Gestionar notas sobre un equipo informático	• Todos los equipos informáticos permiten agregar notas • Los cambios quedan almacenados en la base de datos junto con la fecha de modificación
23	Visualizar las alertas configuradas	• Se encuentran todas las alertas configuradas en la base de datos

24	Añadir una alerta sobre un equipo informático	• Los rangos de mínimo o máximo deben ser consistentes y válidos • La alerta debe generar avisos basados en la configuración dada • La alerta debe quedar almacenada en la base de datos • La alerta debe ser recuperable después de su creación
25	Modificar una alerta sobre un equipo informático	• La alerta no debe generar avisos basados en la antigua configuración • Los cambios deben quedar almacenados en la base de datos • La alerta debe ser recuperable después de su modificación
26	Eliminar una alerta sobre un equipo informático	• La alerta no debe generar más avisos al usuario • La alerta queda eliminada de la base de datos
27	Consultar los registros de un equipo informático	• Los datos deben pertenecer al equipo informático consultado • El intervalo de fechas es configurable por el usuario y debe mostrar únicamente los registros del rango comprendido en éstas • La ausencia de registros debe ser distinguible de registros con datos vacíos
28	Impresión de código QR identificativo	• El código QR debe estar relacionado con el equipo informático seleccionado • Si el usuario ha iniciado sesión previamente, al leer el código QR debe llevarlo inmediatamente a la consulta de datos del dispositivo • Si el usuario no ha iniciado sesión, le redireccionará a la vista de inicio de sesión
29	Detección de equipos informáticos en la red	• Si un equipo informático ya se encuentra registrado, el sistema impedirá registrarlo de nuevo

Tabla 2. Declaración de los criterios de aceptación

2.8. Planificación inicial de tareas

En este capítulo realizaremos una planificación inicial del proyecto clasificando las historias de usuario por prioridad y dividiéndolas en tareas a las que les asignaremos un rol y una estimación temporal. El resultado de ello será la pila del producto.

Los roles involucrados para el desarrollo de este proyecto dentro de un equipo de informática son dos: analista y programador. Nuevamente en este caso, yo mismo desempeñaré ambos roles, pero durante la descomposición en tareas los reflejaremos como dos actores diferentes.

Para la priorización de las tareas recurriremos a la técnica de priorización MoSCoW. Esta técnica debe su nombre a la combinación de las palabras del inglés *Must Have* (debe tener), *Should Have* (debería tener), *Could Have* (podría tener) y *Would Not Have* (no tendrá esta vez) [78]. Cada una de ellas es una categoría donde clasificaremos nuestras historias de usuario. La combinación de estas categorías describe nuestro producto enfocándose en las necesidades reales en lugar de las características ideales [79]. Atendiendo a la combinación de estas categorías, conseguiremos diferentes versiones de nuestro producto:

- Las historias de usuario contenidas en la categoría *Must Have* dan lugar al producto mínimo viable.
- Las historias de usuario contenidas en la categoría *Must Have* y *Should Have* dan lugar al producto deseable.
- Las historias de usuario contenidas en la categoría *Must Have*, *Should Have* y *Could Have* dan lugar al producto ideal.

A continuación, presentaremos las tareas resultantes por la descomposición de las historias de usuario y las dividiremos por prioridad en diferentes tablas. Añadiremos a esta planificación una tarea adicional para reflejar el esfuerzo invertido durante el análisis preliminar. Para la estimación temporal, hemos establecido una relación de 1 punto de historia por media jornada de trabajo, 4 horas, y la hemos dividido entre los roles implicados.

- *Must Have* (debe tener)

ID	Título (Puntos de historia)	Tareas
-	Análisis preliminar (12)	• Analista / 8 horas: Análisis preliminar • Analista / 20 horas: Estudio de soluciones existentes • Analista / 20 horas: Estudio de *frameworks* y tecnologías
1	Iniciar sesión (3)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 4 horas: Implementación de la funcionalidad en el servidor • Programador / 4 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
2	Cerrar sesión (1)	• Analista / 1 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 1 horas: Implementación de la funcionalidad en el servidor • Programador / 1 horas: Implementación de la funcionalidad en el cliente • Programador / 1 horas: Ejecución del plan de pruebas
8	Visualizar equipos informáticos (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas

9	Registrar un equipo informático (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
10	Modificar un equipo informático (3)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 4 horas: Implementación de la funcionalidad en el servidor • Programador / 4 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
11	Eliminar un equipo informático (2)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 2 horas: Implementación de la funcionalidad en el servidor • Programador / 2 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
17	Consultar el histórico de uso de un equipo informático (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 8 horas: Implementación de la funcionalidad en el agente • Programador / 4 horas: Ejecución del plan de pruebas

18	Bloquear un ordenador remotamente (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 2 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 16 horas: Implementación de la funcionalidad en el agente • Programador / 4 horas: Ejecución del plan de pruebas
19	Reiniciar un ordenador remotamente (3)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 2 horas: Implementación de la funcionalidad en el servidor • Programador / 2 horas: Implementación de la funcionalidad en el cliente • Programador / 4 horas: Implementación de la funcionalidad en el agente • Programador / 2 horas: Ejecución del plan de pruebas

Tabla 3. Priorización de historias de usuario *Must Have*

- *Should Have* (debería tener)

ID	Título (Puntos de historia)	Tareas
3	Visualizar ubicaciones (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas

4	Crear una ubicación (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
5	Modificar una ubicación (3)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 4 horas: Implementación de la funcionalidad en el servidor • Programador / 4 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
7	Eliminar una ubicación (2)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 2 horas: Implementación de la funcionalidad en el servidor • Programador / 2 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
12	Asignar un equipo informático a una ubicación (2)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 2 horas: Implementación de la funcionalidad en el servidor • Programador / 2 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas

13	Desasignar un equipo informático de una ubicación (1)	• Analista / 1 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 1 horas: Implementación de la funcionalidad en el servidor • Programador / 1 horas: Implementación de la funcionalidad en el cliente • Programador / 1 horas: Ejecución del plan de pruebas
14	Visualizar tablero (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 16 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 4 horas: Ejecución del plan de pruebas
16	Consultar el estado de uso en tiempo real de un equipo informático (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 8 horas: Implementación de la funcionalidad en el agente • Programador / 4 horas: Ejecución del plan de pruebas

Tabla 4. Priorización de historias de usuario *Should Have*

- *Could Have (podría tener)*

ID	Título (Puntos de historia)	Tareas
6	Asignar una ubicación dentro de otra ubicación (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
15	Visualizar los equipos informáticos sin ubicación (2)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 2 horas: Implementación de la funcionalidad en el servidor • Programador / 2 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
20	Acceder al escritorio remoto de un ordenador (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 8 horas: Implementación de la funcionalidad en el agente • Programador / 4 horas: Ejecución del plan de pruebas

22	Gestionar notas sobre un equipo informático (1)	• Analista / 1 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 1 horas: Implementación de la funcionalidad en el servidor • Programador / 1 horas: Implementación de la funcionalidad en el cliente • Programador / 1 horas: Ejecución del plan de pruebas
28	Impresión de QR identificativo (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas

Tabla 5. Priorización de historias de usuario *Could Have*

- *Would Not Have* (no tendrá esta vez)

ID	Título (Puntos de historia)	Tareas
21	Visualizar la ventana actual de varios ordenadores (13)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 4 horas: Implementación de la funcionalidad en el servidor • Programador / 24 horas: Implementación de la funcionalidad en el cliente • Programador / 12 horas: Implementación de la funcionalidad en el agente • Programador / 8 horas: Ejecución del plan de pruebas

23	Visualizar las alertas configuradas (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
24	Añadir una alerta sobre un equipo informático (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 16 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 4 horas: Ejecución del plan de pruebas
25	Modificar una alerta sobre un equipo informático (5)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas
26	Eliminar una alerta sobre un equipo informático (2)	• Analista / 2 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 2 horas: Implementación de la funcionalidad en el servidor • Programador / 2 horas: Implementación de la funcionalidad en el cliente • Programador / 2 horas: Ejecución del plan de pruebas

27	Consultar los registros de un equipo informático (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 8 horas: Implementación de la funcionalidad en el agente • Programador / 4 horas: Ejecución del plan de pruebas
29	Detección de equipos informáticos en la red (8)	• Analista / 4 horas: Análisis, diseño y documentación de la funcionalidad • Programador / 8 horas: Implementación de la funcionalidad en el servidor • Programador / 8 horas: Implementación de la funcionalidad en el cliente • Programador / 8 horas: Implementación de la funcionalidad en el agente • Programador / 4 horas: Ejecución del plan de pruebas

Tabla 6. Priorización de historias de usuario *Would Not Have*

2.9. Planificación de las iteraciones

Un trabajo fin de grado se compone por 12 créditos ECTS, lo que equivale a 300 horas aproximadas de trabajo del estudiante. Este proyecto se ajustará a dicha cantidad de horas en su totalidad, iniciando desde el 21 de marzo de 2022 y con la finalidad de concluir el 26 de agosto de 2022. Esto supone una duración de 23 semanas entre ambas fechas.

En la metodología ágil SCRUM, establecemos lo que se conoce como iteraciones o *sprints* para definir ciclos temporales de una duración determinada en los que repartiremos las tareas de la planificación inicial. Cada iteración nos debe proporcionar una versión incremental del producto.

Fijaremos cada una de estas iteraciones en periodos de tres semanas y realizaremos un total de 7 iteraciones. Aunque podemos lograr casi una iteración extra entre las fechas marcadas anteriormente, consideraremos emplear dicho tiempo para periodos vacacionales durante la ejecución del proyecto.

En cada iteración estimamos la realización de 12 puntos de historia, lo que nos posibilita la realización de 84 puntos de historia totales sumando las iteraciones disponibles. Recopilando la clasificación de prioridad MoSCoW utilizada anteriormente tenemos:

- 50 puntos de historia en tareas *Must Have* (debe tener),
- 34 puntos de historia en tareas *Should Have* (debería tener)
- 21 puntos de historia en tareas *Could Have* (podría tener)
- 49 puntos de historia en tareas *Would Not Have* (no tendrá esta vez)

Según nuestras estimaciones, podremos cumplir las tareas dentro de las prioridades *Must Have* y *Should Have*, ya que éstas suman juntas 84 puntos de historia coincidiendo con la estimación de esfuerzo total del proyecto. A priori, lamentablemente, parece que las tareas *Could Have* y *Would Not Have* no podrán estar incluidas en la planificación de las iteraciones ya que exceden el esfuerzo necesario. Esto nos lleva a alcanzar el grado de producto deseable, pero no de producto ideal.

A continuación, presentaremos la planificación de las iteraciones del proyecto repartiendo las historias de usuario equitativamente entre ellas, atendiendo primero a las historias con una mayor prioridad. También calcularemos el término de velocidad considerando la realización de 12 puntos de historia por iteración.

Iteración	Historias	Puntos de historia	Periodo	Velocidad
1	(Análisis preliminar)	12	21/03/2022 - 10/04/2022	1

2	1, 8, 9	13	18/04/2022 - 8/05/2022	1,08
3	2, 10, 17	11	9/05/2022 - 29/05/2022	0,92
4	10, 18	11	30/05/2022 - 19/06/2022	0,92
5	3, 4, 19	13	20/06/2022 - 10/07/2022	1,08
6	5, 7, 16	13	11/07/2022 - 31/07/2022	1,08
7	12, 13, 14	11	01/08/2022 - 21/08/2022	0,92

Tabla 7. Planificación de iteraciones

2.10. Evolución de la planificación

Dentro de la metodología ágil SCRUM, podemos visualizar el progreso de nuestro proyecto a través de una gráfica de trabajo pendiente (también conocida por el término de gráfica *burn-down*) con el objetivo de evaluar el correcto avance de las iteraciones [80]. Dicha gráfica cuenta con dos dimensiones: una en el eje de abscisas para reflejar cada iteración y otra en el eje de ordenadas para medir los puntos de historia. Gracias a ella, podemos observar la velocidad con la que vamos completando las tareas conforme progresan las iteraciones [81].

Idealmente, los puntos de historia del eje de ordenadas deben consumirse progresivamente con la velocidad que establecimos al inicio hasta llegar a 0 puntos de historia en la última iteración. Durante la planificación de tareas anterior, incorporamos iteraciones con un mayor esfuerzo momentáneo para ajustar los puntos de historia entre iteraciones, por lo que el avance entre el consumo ideal y el planificado presentará ligeras variaciones.

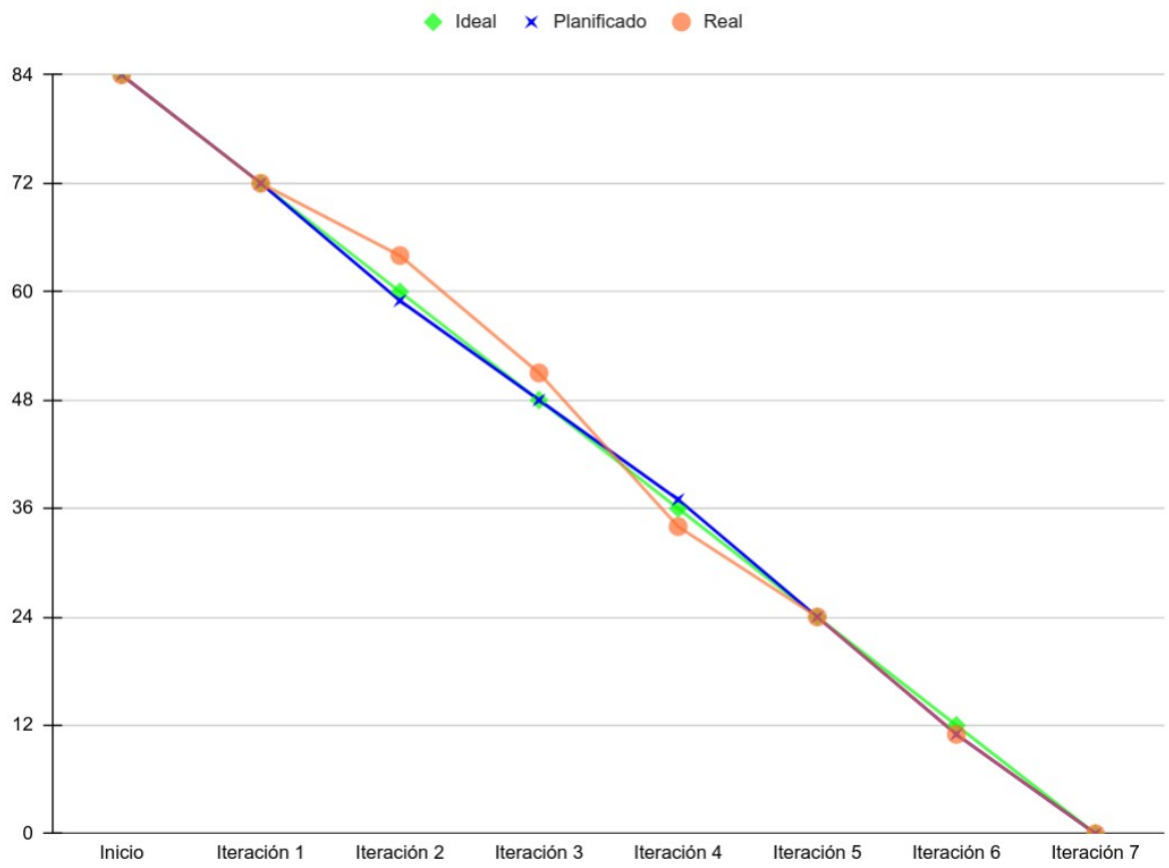

Ilustración 13. Evolución de la planificación

Nuestro propósito es medir el consumo real de los puntos de historia en cada iteración y enfrentarlos respecto del consumo ideal y del consumo planificado. Como observamos en la gráfica anterior, los puntos de historia entre el consumo ideal y el planificado son similares, lo que significa que esas pequeñas desviaciones que introdujimos en la planificación inicial de tareas fueron aceptables. Por otro lado, el consumo real sí presenta mayores desviaciones, donde el avance del proyecto se contrajo en las primeras iteraciones, pero sobre la segunda mitad se estabilizó finalizando sin retrasos y con un consumo de puntos de historia similar a los consumos ideal y planificado.

2.11.Estudio de viabilidad

Un proyecto sólo puede realizarse si disponemos de los recursos suficientes para ello, por lo que necesitaremos esclarecer una estimación del coste de desarrollo de este proyecto después de haber definido su alcance técnico y temporal en los capítulos anteriores [82]. A continuación, consideraremos los recursos

software, hardware y humanos que implican directamente al proyecto indicando cuáles son sus costes en la actualidad y cómo pueden llegar a repercutirnos.

2.11.1. Estimación de costes en recursos hardware y servicios externos

En cuanto a recursos hardware y servicios externos, emplearemos un único portátil para las tareas de análisis, diseño e implementación. Por otro lado, contrataremos un servidor virtual y un dominio para disponer de un entorno de pruebas donde desplegar nuestra aplicación, asemejándose a un entorno de ejecución real. Ambos conceptos se reflejan en las siguientes tablas:

ID	Concepto	Descripción
1	MSI GF75 Thin 9SC	Portátil de 17 pulgadas con 6c/12t, 16 GBs de RAM y NVMe de 500 GBs
2	Contabo Cloud VPS M	Servidor virtual de 6 vCores, 16 GBs de RAM y SSD de 400 GBs
3	Dominio .io	Adquisición de dominio gesmon.io

Tabla 8. Descripción de recursos hardware y servicios externos

ID	Concepto	Precio	Vida útil	Tiempo de uso	Coste repercutido
1	MSI GF75 Thin 9SC	1000 euros	5 años	6 meses	200 euros
2	Contabo Cloud VPS M	108 euros	1 año	6 meses	54 euros
3	Dominio .io	56 euros	1 año	6 meses	28 euros
Total					**282 euros**

Tabla 9. Coste de recursos hardware y servicios externos

Dado que el tiempo de uso estimado de los recursos son 6 meses desde el inicio del proyecto hasta su defensa, hemos establecido la parte proporcional del precio original tanto para el portátil como para el servidor virtual y el dominio, repercutiendo en un coste total de 282 euros.

2.11.2. Estimación de costes en recursos software

En cuanto a recursos software, añadiremos a la pila tecnológica resultante del análisis el sistema operativo que emplearemos, nuestro entorno de desarrollo preferido, un procesador de textos, un sistema de control de versiones y una aplicación para el modelado de diagramas.

ID	Concepto	Descripción
1	Spring Framework	*Framework* para *back-end*
2	Angular	*Framework* para *front-end*
3	PostgreSQL	Base de datos para aplicación web
4	InfluxDB	Base de datos para monitorización
5	OpenLDAP	Directorio activo LDAP
6	Plotly	Biblioteca para el renderizado de gráficas
7	Osquery	*Framework* de consulta de métricas
8	SDL	*Framework* de gráficos
9	QEMU	Virtualizador de máquinas
10	Docker	Empaquetamiento de software
11	Docker Compose	Orquestación de servicios
12	Nginx	Servidor web para recursos estáticos
13	Caddyserver	*Proxy* reverso
14	Debian	Sistema operativo basado en GNU/Linux
15	NetBeans	Entorno de desarrollo para aplicaciones
16	LibreOffice Writer	Procesador de textos
17	Git	Sistema de control de versiones
18	Visual Paradigm	Modelado de diagramas

Tabla 10. Descripción de recursos software

ID	Concepto	Licencia	Tiempo de uso	Coste repercutido
1	Spring Framework	Apache 2.0	6 meses	0 euros
2	Angular	MIT	6 meses	0 euros
3	PostgreSQL	PostgreSQL License	6 meses	0 euros
4	InfluxDB	MIT	6 meses	0 euros
5	OpenLDAP	OpenLDAP Public License	6 meses	0 euros
6	Plotly	MIT	6 meses	0 euros
7	Osquery	Apache 2.0	6 meses	0 euros
8	SDL	Zlib	6 meses	
9	QEMU	GNU GPL v2	6 meses	0 euros
10	Docker	Apache 2.0	6 meses	0 euros
11	Docker Compose	Apache 2.0	6 meses	0 euros
12	Nginx	BSD v2	6 meses	0 euros
13	Caddyserver	Apache 2.0	6 meses	0 euros
14	Debian	DFSG-Compatible	6 meses	0 euros
15	NetBeans	Apache 2.0	6 meses	0 euros
16	LibreOffice Writer	Mozilla 2.0	6 meses	0 euros
17	Git	GNU GPL v2	6 meses	0 euros
18	Visual Paradigm	Academic License	6 meses	0 euros
Total				**0 euros**

Tabla 11. Coste de recursos software

A excepción de Visual Paradigm –del que contamos de una licencia académica gracias a la Universidad de Jaén–, el resto de los recursos software que hemos seleccionado emplean licencias libres y gratuitas, tanto para su uso personal como comercial, por lo que no nos supondrán ningún coste.

2.11.3. Estimación de costes en recursos humanos

En cuanto a recursos humanos, manejamos dos roles en nuestro equipo de desarrollo: analista y programador. Según el portal de empleo Indeed, en España el salario base de un analista es de 165 euros/día [83], mientras que el de un programador es de 67 euros/día [84]. Prorrateadamente, esto son 20,625 euros/hora y 8,375 euros/hora respectivamente.

Ilustración 14. Salarios en regimen diario de un analista y de un desarrollador

Hemos considerado el cálculo de horas ya estipulado en la planificación inicial de tareas con prioridad *Must Have* y *Should Have*. La suma del salario base de ambos roles supondrían un coste total estimado de 3567,24 euros.

ID	Concepto	Salario en euros/Hora	Horas trabajadas (estimadas)	Coste repercutido
1	Analista	20,63	87	1794,37 euros
2	Programador	8,38	213	1783,87 euros
Total				**3567,24 euros**

Tabla 12. Coste de recursos humanos

2.11.4. Estimación final de costes

Resumiendo, las estimaciones de costes de los anteriores apartados, suman un total de 3849,24 euros sobre recursos directamente implicados con el proyecto.

ID	Recurso directo	Coste
1	Hardware/Servicios	282 euros
2	Software	0 euros
3	Humanos	3567,24 euros
Total		**3849,24 euros**

Tabla 13. Costes directos del proyecto

También podemos considerar la incorporación del coste de recursos indirectos que, aunque no están directamente implicados con el proyecto, pueden suponernos un incremento significativo sobre el coste final. Vamos a ejemplificar algunos de ellos en la siguiente tabla:

ID	Recurso indirecto	Precio estimado	Periodo	Tiempo de uso	Coste repercutido
1	Conexión a internet	50 euros	1 mes	6 meses	300 euros
2	Suministros de luz	50 euros	1 mes	6 meses	300 euros

3	Gastos en papelería (folios, bolígrafos, etc)	5 euros	1 mes	6 meses	30 euros
Total					**630 euros**

Tabla 14. Costes indirectos del proyecto

Así, si consideramos recursos directos e indirectos, concluimos un coste final estimado de 4479,24 euros.

ID	Concepto	Coste
1	Recursos directos	3849,24 euros
2	Recursos indirectos	630 euros
Total		**4479,24 euros**

Tabla 15. Suma completa de costes directos e indirectos

2.12. Modelo de dominio

El modelo de dominio es una representación conceptual del sistema que describe los aspectos relevantes del mismo mediante clases e identifica las relaciones entre todas las entidades implicadas junto con sus atributos y restricciones [85]. Dado que aún nos encontramos en la etapa de análisis, nos centraremos en las abstracciones de las clases con mayor relevancia y que nos sirvan de inspiración en el diseño de software para los siguientes apartados.

A continuación, representaremos el modelo de dominio mediante un diagrama de clases con notación UML.

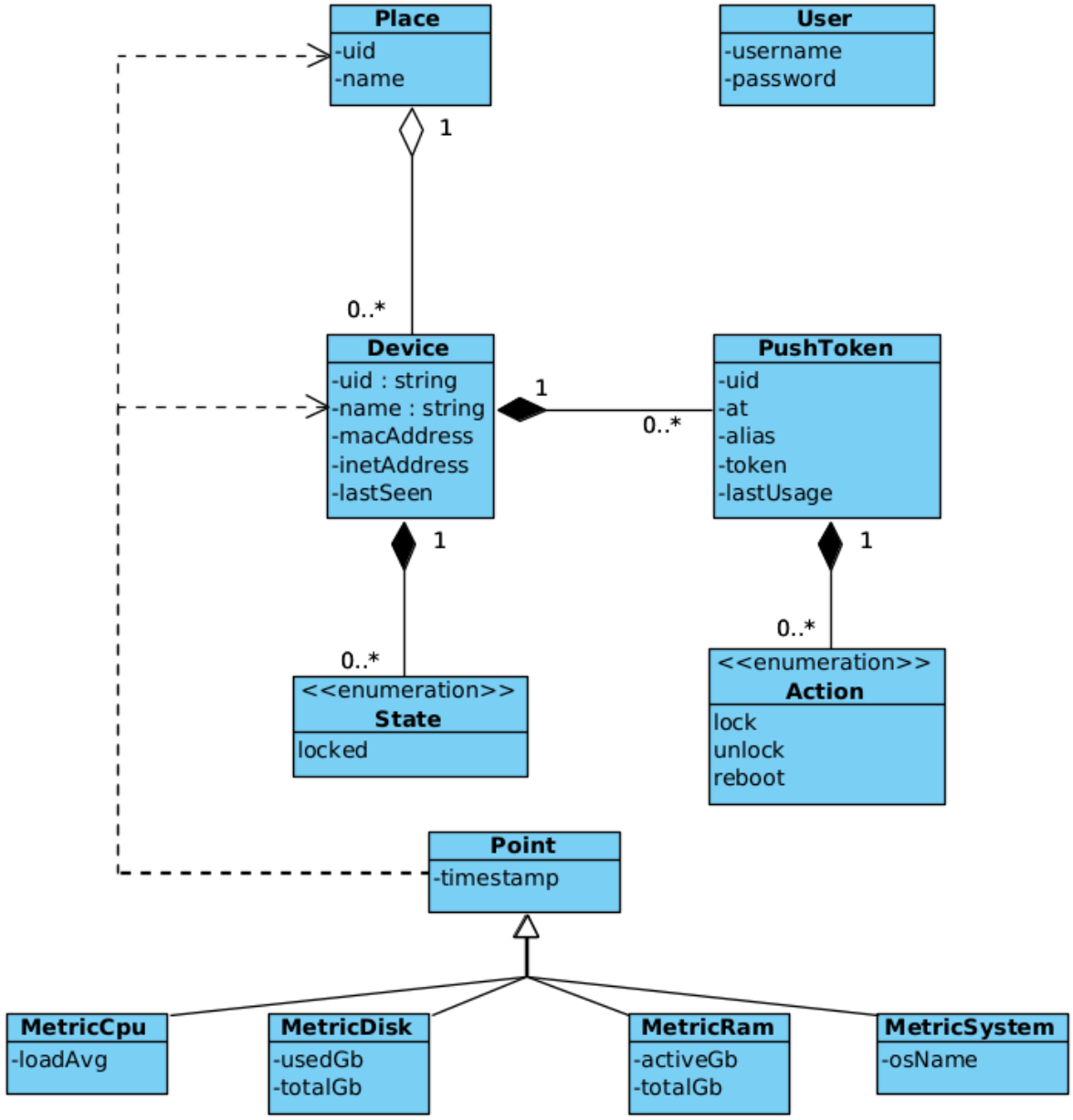

Ilustración 15. Modelo de dominio simplificado

Entre las clases del modelo de dominio encontramos la representación de las ubicaciones por la entidad *Place* y la representación de los equipos informáticos por la entidad *Device*. Estas dos entidades se encuentran vinculadas mediante una relación de agregación, ya que los equipos se podrán asignar a las ubicaciones. En cuanto a la cardinalidad de esta relación, como es evidente, un equipo sólo puede estar asignado a una única ubicación, ya que las ubicaciones reflejan espacios físicos, mientras que una ubicación puede albergar a múltiples equipos.

Por otro lado, los equipos disponen de una relación de composición con la entidad *PushToken*, que será la encargada de almacenar las credenciales utilizadas por los agentes instalados en los equipos y que ofrecen métricas bajo la arquitectura de monitorización *push*. Si en el futuro quisiéramos implementar una arquitectura *pull* sobre este mismo modelo de dominio, podríamos establecer una nueva entidad *PullToken* mediante otra relación de composición.

Una particularidad de los agentes de monitorización *push* es que son capaces de ejecutar comandos de acción, los cuales pueden afectar a los estados del equipo. Así, desde la entidad *PushToken* existe una relación de composición con los enumerados de tipo *Action*, mientras que en la entidad *Device* existe una relación de composición con los enumerados de tipo *State*.

De cara al almacenamiento de las métricas de monitorización, éstas tienen en común a la entidad *Point*. Representaremos cada una de las métricas como una entidad diferente. Así, cuando surja una nueva métrica, la incorporaremos en el sistema como una nueva entidad, tantas como deseemos profundizar en la observabilidad del equipo.

Por último, la entidad *User* integra a los usuarios del sistema. Sin embargo, tiene actualmente poca relevancia debido a que no hemos puesto el foco en ningún sistema de roles ni de permisos sobre ubicaciones.

3. DISEÑO

En este apartado crearemos especificaciones sobre los componentes del sistema para satisfacer las funcionalidades contempladas en el apartado de análisis y que plasmaremos mediante diferentes tipos de diagramas e interfaces.

3.1. Diagrama entidad-relación

Un diagrama entidad-relación es una representación gráfica del modelo de datos relacional dentro de un sistema que describe las características de las entidades participantes y las asociaciones entre éstas.

Nuestro proyecto recoge las entidades vistas del modelo de dominio en el capítulo anterior para formular un modelo de datos relacional. Este paso habitualmente es manual, pero gracias al uso de un ORM (*object-relational mapping*) podemos vincular los objetos de programación hacia entidades de una base de datos relacional. En este sentido, Spring Framework nos permite elegir a Hibernate como ORM del *back-end*. Entre las características de Hibernate encontramos la generación automática de tablas, relaciones y consultas entre el modelo de dominio y el modelo de datos relacional independientemente del motor de base de datos empleado, lo que nos ayudará además a interactuar con la capa de persistencia del sistema de manera abstracta.

En la siguiente ilustración mostramos el diagrama entidad-relación del servidor tras la generación de entidades mediante Hibernate como ORM.

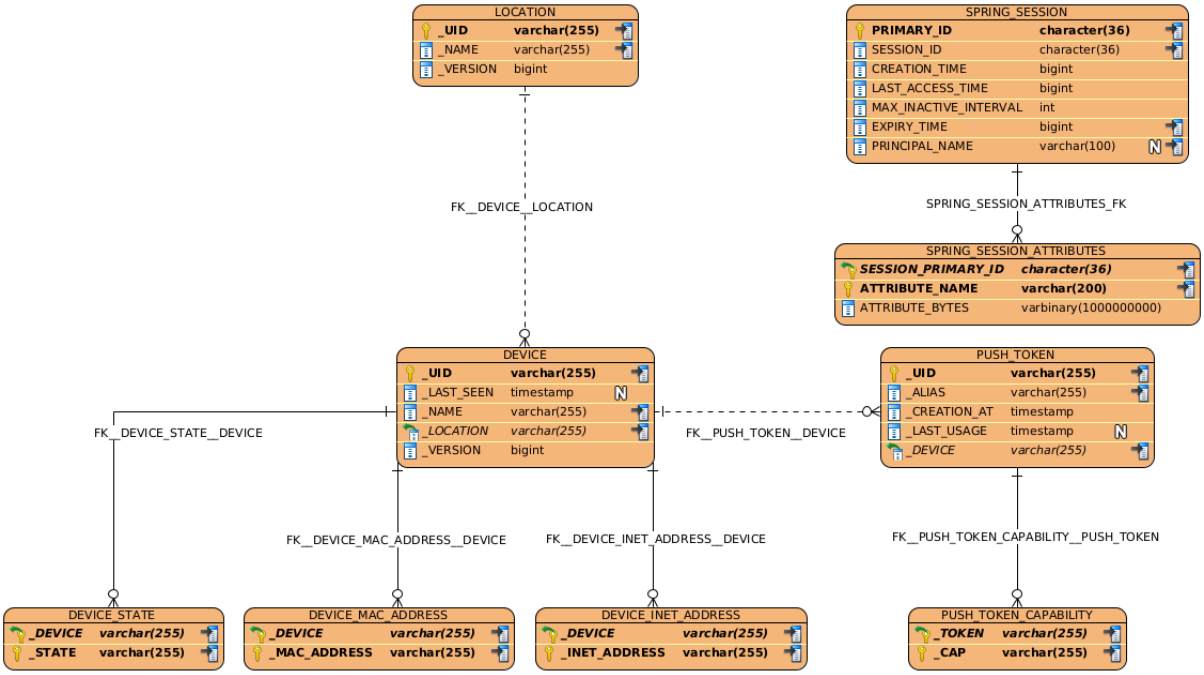

Ilustración 16. Diagrama Entidad-Relación

A este mecanismo de generación automática lo someteremos a los criterios de normalización de base de datos relacional enunciados por Edgar Frank Cood para detectar anomalías en el proceso, reducir la redundancia de los datos y mejorar la integridad de los mismos [86]. Dentro de los criterios de normalización, verificaremos que el diagrama anterior satisface la primera, segunda y tercera forma normal (1FN, 2FN y 3FN) [87].

Una tabla se encuentra en primera forma normal si todos sus atributos sólo recogen valores indivisibles y todos sus atributos que no sean clave primaria son funcionalmente dependientes de parte de la clave primaria. Este enunciado queda incumplido si, por ejemplo, dentro de un atributo encontramos múltiples valores o valores compuestos. En nuestro caso, el diseño anterior cumple con la 1FN.

Una tabla se encuentra en segunda forma normal si cumple con los criterios de la 1FN y todos sus atributos que no sean clave primaria dependen funcionalmente de forma completa de la clave primaria. Este enunciado incide en las claves primarias compuestas exigiendo que los atributos no clave deben de depender de todos los atributos que sean parte de la clave primaria. En nuestro caso, el diseño anterior cumple con la 2FN.

Una tabla se encuentra en tercera forma normal si cumple con los criterios de la 2FN y todos sus atributos que no sean clave primaria no dependen transitivamente de la clave primaria. Este enunciado involucra la eliminación de dependencias transitivas, donde un atributo no clave no puede depender de otro atributo no clave. En nuestro caso, el diseño anterior cumple con la 3FN.

A continuación, vamos a realizar una breve discusión sobre las cuatro entidades principales que pueden observarse en el diagrama para justificar su presencia y finalidad.

- Tabla *Location*

Representa una ubicación física dentro de la organización indicando su nombre. Establece una relación uno-a-muchos con la entidad *Device*, ya que el propósito de esta tabla es organizar a los equipos dentro de una ubicación.

- Tabla *Push_Token*

Representa un *token* de autenticación para un agente de control y monitorización. El agente es un programa informático instalado en los equipos para el envío de métricas de monitorización y recepción de comandos. Dicho agente utilizará el mencionado *token* para relacionarse a nivel lógico con el equipo correspondiente dentro del sistema. La tabla *Push_Token_Capability* registrará un listado de enumerados de las capacidades del agente después de la autenticación.

- Tabla *Device*

Representa un equipo informático dentro del sistema. Establece una relación uno-a-muchos con la entidad *Push_Token*; aunque esta relación podría ser uno-a-uno, es preferible una relación uno-a-muchos con el fin de rotar el *token* sin perder el acceso. Además, la tabla *Device_State* registra un listado de estados alterados del equipo a consecuencia de una configuración realizada desde el sistema. Por último, las tablas *Device_Mac_Address* y *Device_Inet_Address* identifican las direcciones físicas y de red del equipo.

- Tabla *Spring_Session*

Representa un *token* de sesión para los usuarios del sistema en la plataforma web. Como fue explicado durante el análisis, la aplicación no maneja información de usuarios, sino que usaremos un servicio LDAP como directorio de usuarios. Dicho *token* se genera durante el proceso de autenticación y se persiste temporalmente en la base de datos hasta su máximo tiempo de inactividad. Gracias a él, podemos conocer si un usuario pertenece o no al sistema. La definición del modelo de datos de esta tabla pertenece al módulo Spring Session JDBC dentro de Spring Framework.

3.2. Diagrama de series temporales

Un diagrama de series temporales es una representación gráfica de las estructuras utilizadas en el modelo de datos de una base de datos de series temporales. No existe un diagrama estándar para la representación del mismo, por lo que usaremos un diagrama de clases por conveniencia para ilustrar el almacenamiento de métricas en nuestro sistema.

Dependiendo del motor de base de datos empleado, los conceptos clave involucrados pueden ser diferentes. Por ejemplo, TimescaleDB se acerca más a una base de datos relacional, mientras que InfluxDB tiene conceptos propios. Durante el análisis, finalmente elegimos InfluxDB como motor para nuestra base de datos de series temporales, por lo que orientamos el contenido de este capítulo a dicho motor.

Los conceptos clave a considerar dentro de una base de datos InfluxDB son: cubetas (o *buckets*), marcas de tiempo (o *timestamps*), medidas (o *measurements*), campos (o *fields*), etiquetas (o *tags*) y puntos (o *points*). Dado que son conceptos específicos dentro del motor, definiremos a continuación cada uno de ellos [88].

Un punto es, comparándolo con una base de datos tradicional, una tupla compuesta por una marca de tiempo, una serie de etiquetas y una serie de campos.

Una cubeta es, comparándolo con una base de datos tradicional, el esquema, aunque funcionalmente se trata de un espacio de almacenamiento para los puntos

con restricciones de capacidad y con un periodo de retención máximo. Dentro de una cubeta, se favorecen las inserciones, pero no así las actualizaciones ni los borrados, que están desaconsejados.

Una medida es, comparándolo con una base de datos tradicional, una tabla para los puntos. Aunque en realidad, no sigue una estructura determinada y se limita a actuar como un contenedor para puntos con una estructura variable de etiquetas, campos y marcas de tiempo. Por lo tanto, es el programador quien le confiere estructura a las medidas, siendo mutables si el programador cambia la definición de los puntos almacenados.

Una etiqueta es un atributo indexado, mientras que un campo es un atributo no indexado. A la hora de realizar consultas, los campos son menos eficientes ya que el motor necesita inspeccionar todos los valores almacenados para encontrar coincidencias. Normalmente, las etiquetas se emplean para indicar los metadatos asociados a los puntos, con la finalidad de recuperar un subconjunto de puntos bajo un intervalo de tiempo.

Una marca de tiempo es un atributo indexado obligatorio asociado al punto en cuestión para indicar el instante temporal en el que se originó dicho punto. Soporta operaciones de búsqueda y agregación en el tiempo.

Conociendo ahora todos estos conceptos, el diagrama de clases modificado de la siguiente ilustración muestra las clases como medidas, los atributos privados como etiquetas y el resto de atributos como campos.

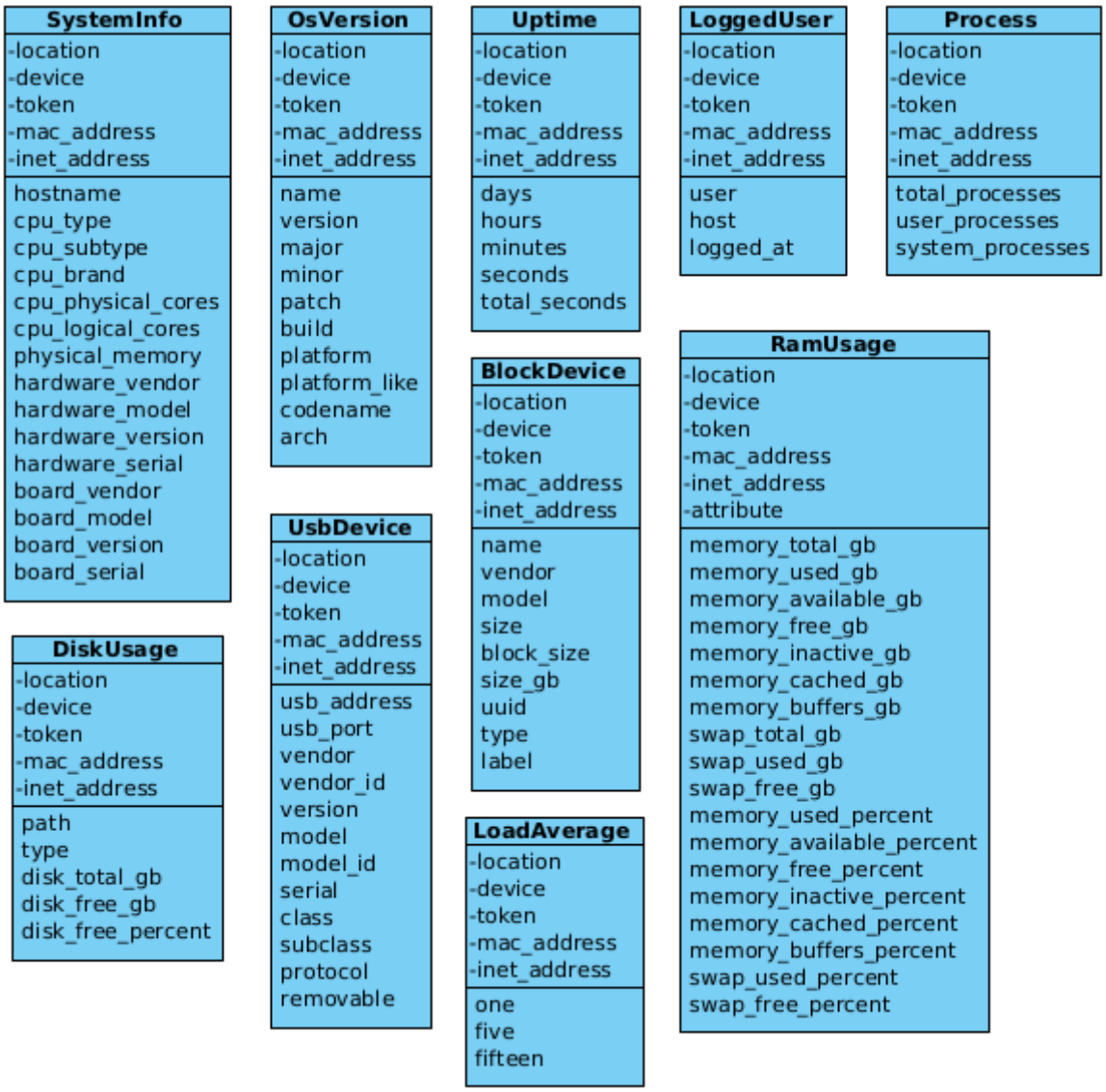

Ilustración 17. Diagrama de series temporales

Las medidas presentadas en el diagrama anterior responden a una selección de métricas obtenidas a través del *framework* Osquery. Hemos establecido las mismas etiquetas a todas las medidas, es decir: *location*, *device*, *token*, *mac_address* e *inet_address* con la finalidad de que podamos escribir consultas en base a esas etiquetas uniformemente desde el sistema.

3.3. Diagrama de directorio activo

Un diagrama de directorio activo es una representación gráfica de las estructuras organizativas pertenecientes a un modelo de datos jerárquico. No existe un diagrama estándar para la representación del mismo, por lo que usaremos un diagrama de árbol por conveniencia.

Los propósitos de un directorio activo son diversos y lo debemos entender como un recurso externo al que nos integraremos y con el que desarrollaremos una cierta funcionalidad. LDAP es el directorio activo que vamos a emplear en nuestro proyecto para implementar la autenticación de los usuarios. Dentro de LDAP, la jerarquía al completo se denomina *Directory Information Tree* (DIT) y se compone por objetos, que bien pueden contener a otros objetos o no. La ruta completa hacia un objeto dentro de la jerarquía se denomina *Distinguished Name* (DN), si es relativa *Relative Distinguished Name* (RDN). Los objetos son instancias de clases, que a su vez describen los atributos de los mismos [89]. Nosotros haremos uso de dos clases estandarizadas, *InetOrgPerson* y *GroupOfNames*.

Dentro del DIT nos interesan dos ramificaciones, una donde se ubican los usuarios de clase *InetOrgPerson* y otra donde se encuentran los grupos de clase *GroupOfNames*. De este modo, de los objetos *InetOrgPerson* podremos recoger la información del usuario como su nombre, apellidos, nombre de usuario y contraseña, mientras que al mismo tiempo verificaremos si un usuario dado pertenece a algún grupo de clase *GroupOfNames* para conceder su acceso finalmente al sistema o denegarlo.

En la siguiente ilustración mostramos el diagrama de directorio activo empleado para albergar instancias de objetos *InetOrgPerson* y *GroupOfNames* con las ramificaciones necesarias para implementar una autenticación de usuarios.

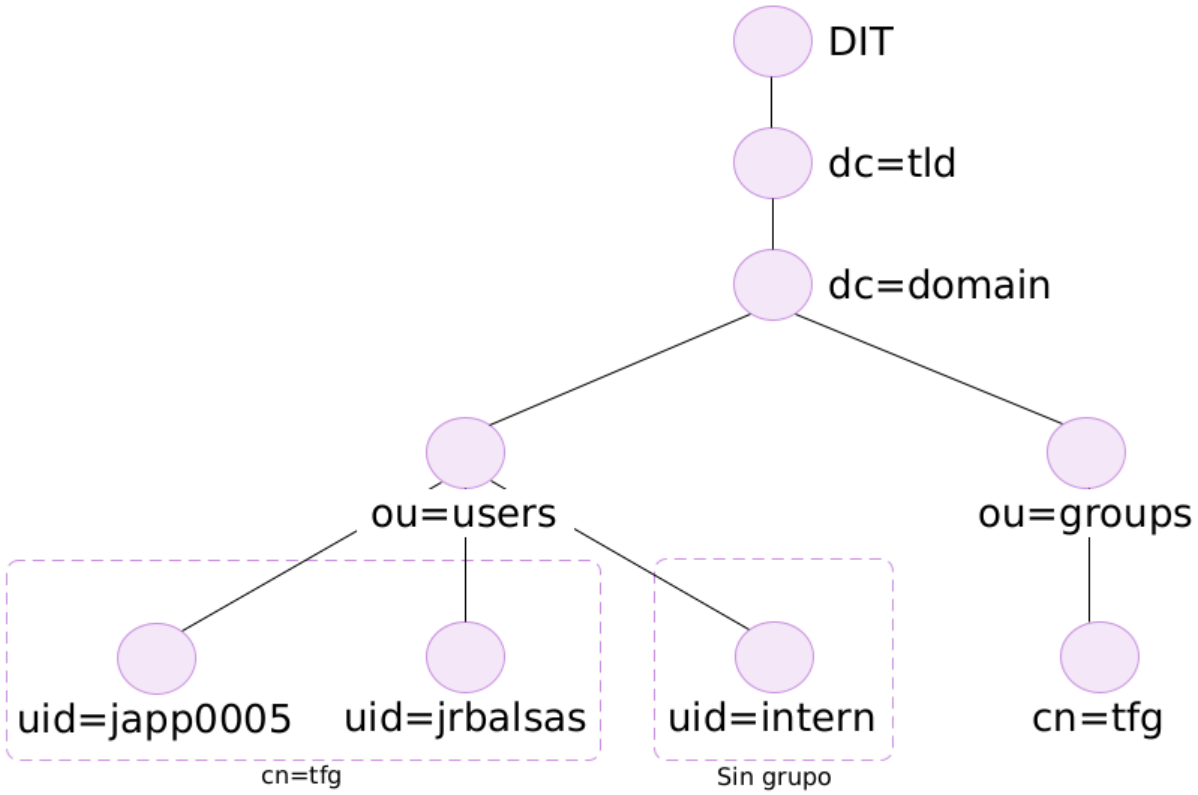

Ilustración 18. Diagrama de directorio activo

Como se puede intuir, los objetos con RDN: uid=japp0005, uid=jrbalsas y uid=intern son instancias de la clase *InetOrgPerson*, mientra que el objeto con RDN: cn=tfg es una instancia de la clase *GroupOfNames*. La pertenencia a un grupo viene dictada por el atributo *member* de la clase *GroupOfNames* y ha sido representada mediante una caja contextual. Así, los usuarios con RDN: uid=japp0005 y uid=jrbalsas tendrán acceso al sistema por pertenecer al grupo con RDN: cn=tfg, pero el usuario con RDN: uid=intern no podrá autenticarse pese a ser un usuario válido al no pertenecer a ningún grupo.

3.4. Diagrama de clases

Un diagrama de clases es una representación gráfica de la estructura de clases de un sistema describiendo sus atributos, métodos y relaciones entre sí.

Nuestro sistema se divide en tres aplicaciones bien diferenciadas: servidor, cliente web y agente de gestión y monitorización. A continuación, plantearemos el diagrama de clases de cada uno y comentaremos los aspectos de diseño más relevantes de los mismos.

3.4.1. Diagrama de clases del servidor

El servidor, el cual va a ser desarrollado mediante el *framework* Spring Framework, se encarga de ejecutar la lógica de negocio principal del sistema y de coordinar el modelo de datos general tanto para el cliente web como para el agente de gestión y monitorización cumpliendo con el patrón de arquitectura modelo/vista/controlador.

Dentro de las diferentes categorías de clases que podemos representar en este diagrama, distinguiremos las siguientes para el servidor:

- Clases controladoras
- Clases de servicio
- Clases de repositorio
- Clases del modelo de datos

En la siguiente ilustración mostramos el diagrama de clases del servidor resultante tras aunar las clases mencionadas anteriormente, así como las relaciones entre todas éstas.

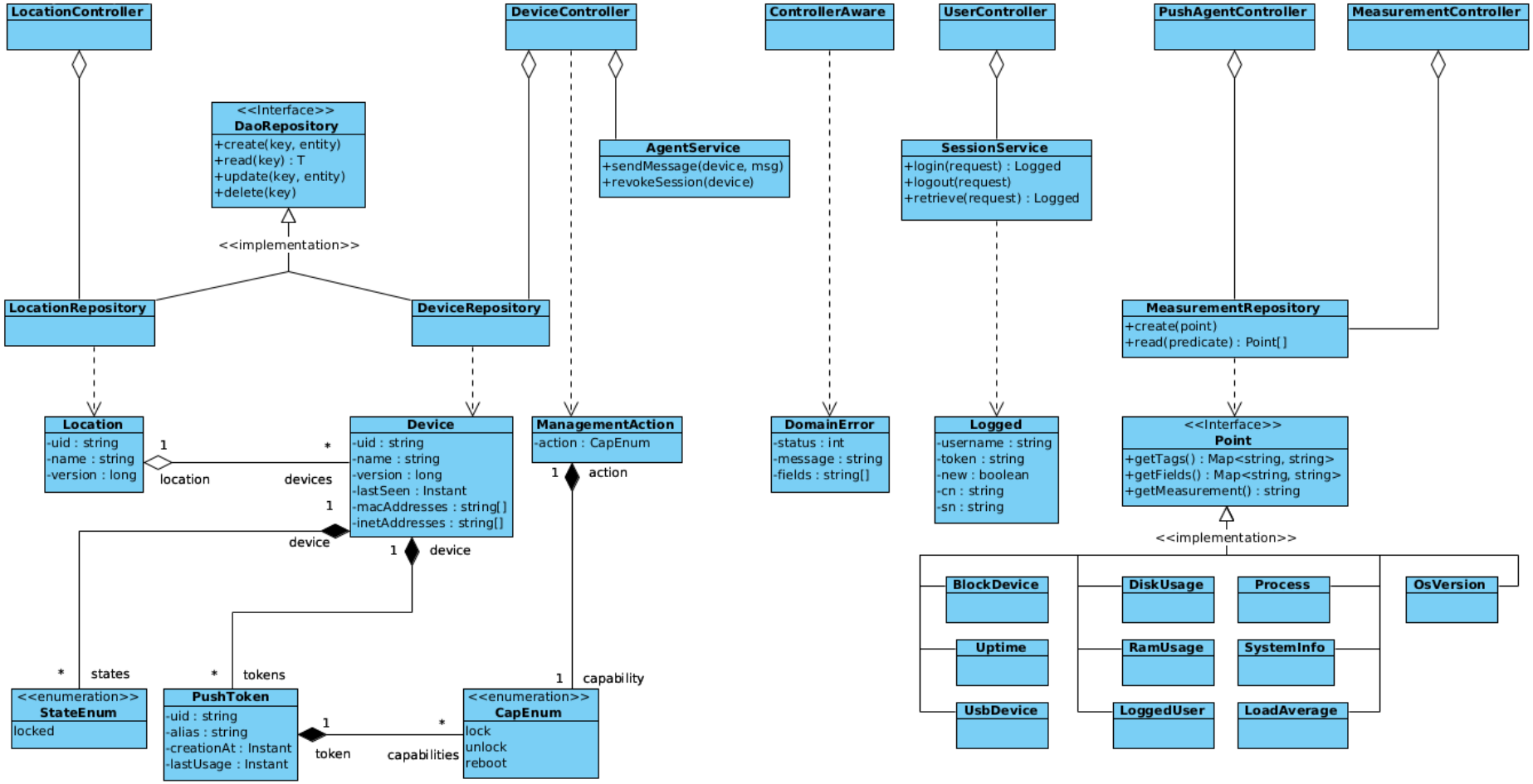

Ilustración 19. Diagrama de clases en el servidor

Las clases controladoras definen la interfaz de comunicación API REST del servidor, entre las cuales:

- *LocationController* define las interfaces para las ubicaciones.
- *DeviceController* define las interfaces para los equipos informáticos.
- *UserController* define las interfaces para la autenticación de usuarios.
- *PushAgentController* define las interfaces para los agentes de monitorización.
- *MeasurementController* define las interfaces de consulta de métricas de monitorización.
- *ControllerAware* define el tratamiento de las excepciones acontecidas en el resto de controladores.

Las clases de servicio implementan lógica de negocio propia dentro del servidor, entre las cuales:

- *AgentService* define métodos para la emisión de comandos a los agentes y la revocación de agentes en línea.
- *SessionService* define métodos para la autenticación de usuarios mediante un *token* de sesión.

Las clases de repositorio proporcionan abstracción hacia el acceso al modelo de datos, entre las cuales:

- *LocationRepository* define el acceso a las entidades de la clase *Location*.
- *DeviceRepository* define el acceso a las entidades de la clase *Device*.
- *MeasurementRepository* define el acceso a las entidades relativas a las métricas de monitorización.

Como puede observarse en el diagrama, las clases pertenecientes al modelo de datos se relacionan de manera similar a los diagramas de entidad-relación y de series temporales vistos anteriormente, así que nos centraremos en el resto de relaciones.

Las clases *LocationController* y *DeviceController* integran a *LocationRepository* y *DeviceRepository* respectivamente para satisfacer las operaciones de creación y modificación del modelo de datos desde la interfaz de comunicación API REST. Además, *DeviceController* emplea el servicio *AgentService* para implementar interfaces API REST adicionales con las que comunicarse con los equipos.

De manera análoga, *PushAgentController* integra a *MeasurementRepository* para la persistencia de las métricas.

Más adelante, también ahondaremos sobre el resto de clases relativas al propio *framework*, en el diagrama de paquetes del servidor del apartado de implementación.

3.4.2. Diagrama de clases del cliente web

El cliente web, el cual va a ser desarrollado mediante el *framework* Angular, supone la interfaz de usuario del sistema accesible desde un navegador web.

Dentro de las diferentes categorías de clases que podemos representar en este diagrama, distinguiremos las siguientes para el cliente web:

- Clases de componentes
- Clases de servicio
- Clases de guarda

En la siguiente ilustración mostramos el diagrama de clases del cliente web resultante tras aunar las clases mencionadas anteriormente, así como las relaciones entre todas éstas.

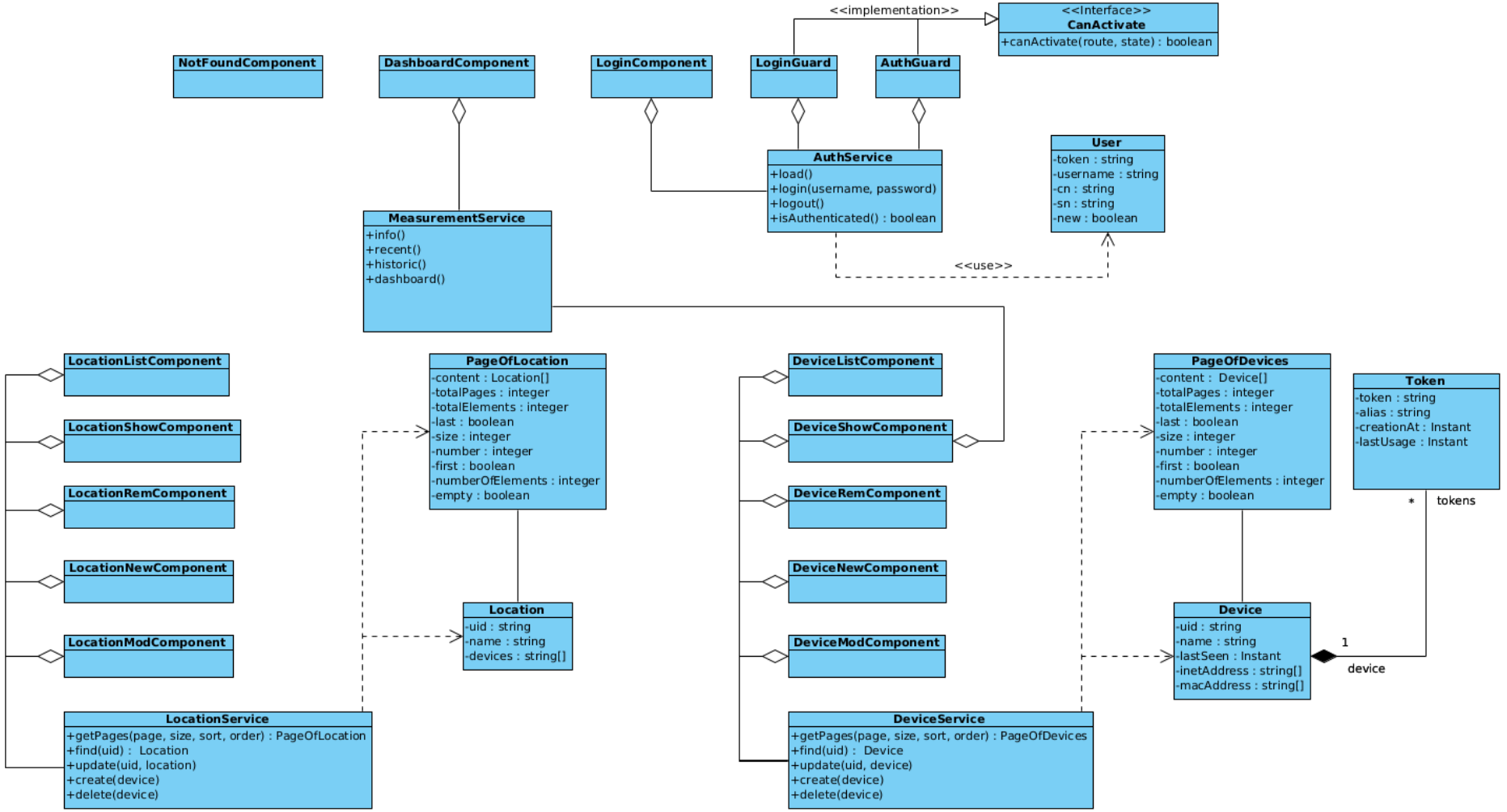

Ilustración 20. Diagrama de clases del cliente web

Las clases de componentes comunican al cliente web directamente con la vista a través del modelo de objetos del navegador, entre las cuales:

- *LoginComponent* define la visualización de la pantalla de inicio de sesión en la interfaz de usuario.
- *DashboardComponent* define la visualización de la sección «Tablero» en la interfaz de usuario.
- *LocationListComponent* y el resto de componentes prefijados con *Location* definen la visualización de la sección «Ubicaciones» en la interfaz de usuario.
- *DeviceListComponent* y el resto de componentes prefijados con *Device* definen la visualización de la sección «Equipos» en la interfaz de usuario.
- *NotFoundComponent* define la visualización de una página no encontrada en la interfaz de usuario.

Las clases de servicio intermedian para otorgar funcionalidades a otras clases, entre las cuales:

- *LocationService* define métodos para la gestión de ubicaciones mediante la interfaz de comunicación API REST del servidor.
- *DeviceService* define métodos de gestión de equipos mediante la interfaz de comunicación API REST del servidor.
- *AuthService* define métodos para la autenticación de usuarios mediante la interfaz de comunicación API REST del servidor.
- *MeasurementService* define métodos para la consulta de intervalos de métricas de monitorización mediante la interfaz de comunicación API REST.

Las clases de guarda interceptan las rutas del cliente web para aplicar una serie de reglas de acceso, entre las cuales:

- LoginGuard restringe el acceso al componente *LoginComponent* si el usuario se encuentra autenticado. En caso de activarse, realiza una redirección a *DashboardComponent*.

- AuthGuard restringe el acceso al resto de componentes si el usuario no se encuentra autenticado. En caso de activarse, realiza una redirección a *LoginComponent*.

Como puede observarse en el diagrama, en cada conjunto de clases componentes empleamos respectivamente a una clase de servicio propia. Es decir, todos los componentes de *Device* y *Location* se apoyan en *DeviceService* y *LocationService* respectivamente. Además, existe una pequeña réplica de los modelos de datos del servidor en los propios modelos del cliente web para ser capaz de recibir las respuestas de la interfaz de comunicación API REST.

Por otro lado, el servicio *AuthService* lo emplean tanto *LoginComponent*, cuando necesita validar las credenciales de un usuario durante el inicio de sesión del mismo, como por las clases guarda *LoginGuard* y *AuthGuard* durante el acceso a una ruta arbitraria en el cliente web.

Más adelante, también ahondaremos sobre el resto de clases relativas al propio *framework*, en el diagrama de paquetes del cliente web del apartado de implementación.

3.4.3. Diagrama de clases del agente de gestión y monitorización

El agente de gestión y monitorización, escrito en el lenguaje de programación Python, es un programa informático instalado en los equipos capaz de recibir comandos de acción desde el servidor y encargado de enviar métricas siguiendo una arquitectura de monitorización *push*.

No vamos a emplear ningún *framework* estructural durante el desarrollo del agente, por lo que tampoco tendremos una imposición sobre la disposición a seguir

de las clases. Así pues, propondremos un diseño de clases específico con una organización propia de los elementos.

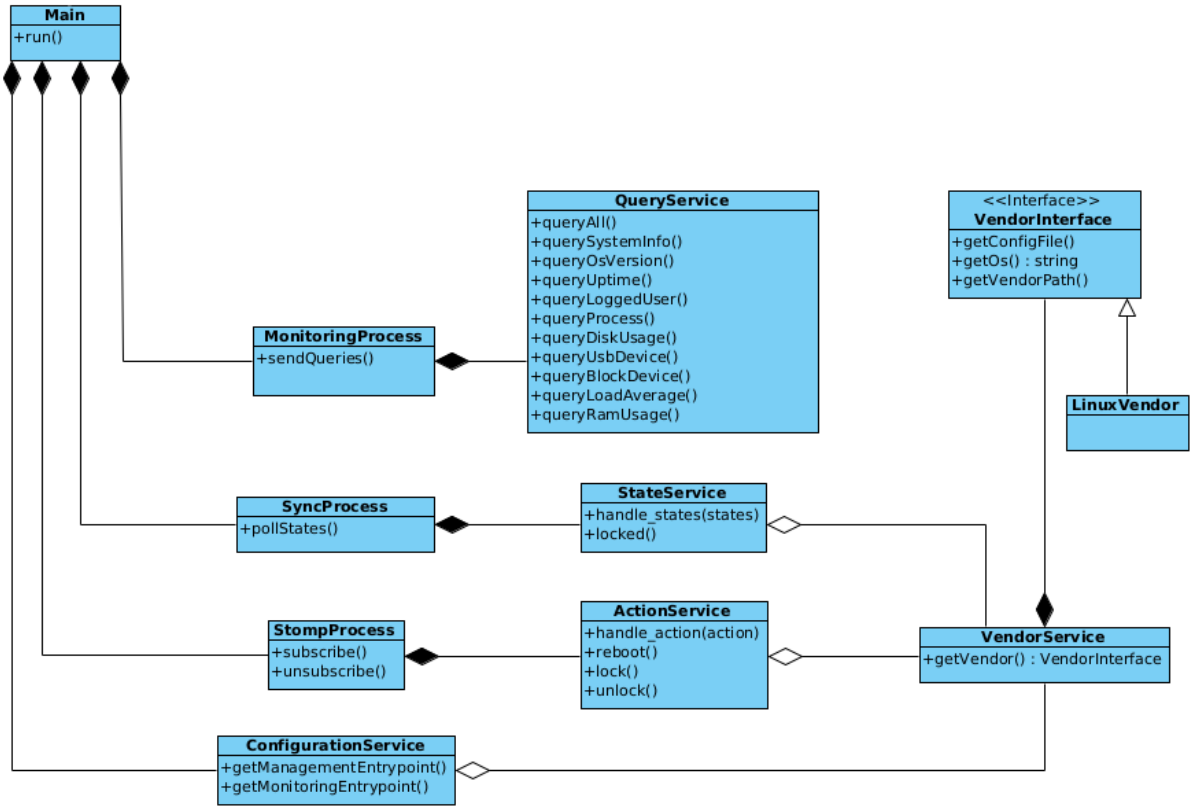

Ilustración 21. Diagrama de clases del agente de gestión y monitorización

La clase *Main* es el punto de partida de la aplicación. En ella preparamos tres subprocesos, además de un servicio de configuración.

Los subprocesos permanecerán en constante ejecución hasta el cierre del agente y con cada uno implementaremos una funcionalidad específica:

- *StompProcess* suscribe al agente al sistema de colas del servidor para la recepción de comandos de acción en tiempo real. Después de recibir un mensaje, el servicio *ActionService* ejecuta la acción recibida en el equipo.

- *MonitoringProcess* envía métricas de monitorización periódicamente, recogidas por el servicio *QueryService* al servidor mediante la interfaz de comunicación API REST.

- *SyncProcess* mantiene actualizada la configuración del agente si el modelo de datos del equipo ha sido modificado y establece un comprobador de salud básico para que el servidor sepa que el agente se encuentra activo.

El servicio *ConfigurationService* recupera la configuración del agente desde un fichero de configuración albergado en una ruta concreta del equipo. Desde ese fichero, obtendremos el *token* de autenticación necesario para asociar el agente a un equipo del sistema.

Por otro lado, el servicio *VendorService* obtiene una instancia de *VendorInterface* empleada desde otros servicios como *ConfigurationService*, *ActionService* o *StateService* para implementar, con cierto nivel de abstracción, llamadas a instrucciones dependientes del sistema operativo. Consideraremos únicamente el soporte para sistemas operativos basados en GNU/Linux implementando la interfaz *VendorInterface* como *LinuxVendor*.

3.5. Diagrama de secuencia

Un diagrama de secuencia es una representación gráfica de la interacción de unos objetos con otros a lo largo del tiempo para una tarea concreta.

Con este tipo de diagramas pretendemos facilitar al lector una mejor comprensión del funcionamiento interno de la aplicación y, además, complementar al diagrama de clases anterior proporcionando un sentido práctico al mismo.

A continuación, elegiremos las tareas más representativas del sistema para presentar su diagrama de secuencia y detallaremos el comportamiento seguido de unos objetos con otros. Además, dibujaremos cada objeto en el diagrama con un color concreto para señalar el área al que pertenecen: magenta para el cliente web, verde para el servidor, color amarillo para la base de datos, color rosa para el directorio activo y color naranja para el agente.

El resto de tareas o historias de usuario que no cubriremos tendrán un comportamiento similar a las elegidas.

3.5.1. Diagrama de secuencia: Control de acceso al sistema

La siguiente ilustración muestra el diagrama de secuencia correspondiente al acceso que realiza nuestro actor, el administrador de sistemas, desde un navegador web. En dicho diagrama queremos reflejar el comportamiento que tienen el cliente web y el servidor dependiendo de si el actor se encuentra autenticado o no; es decir, este es el punto de partida cada vez que nuestro actor abre una pestaña en blanco en su navegador y accede a la página web.

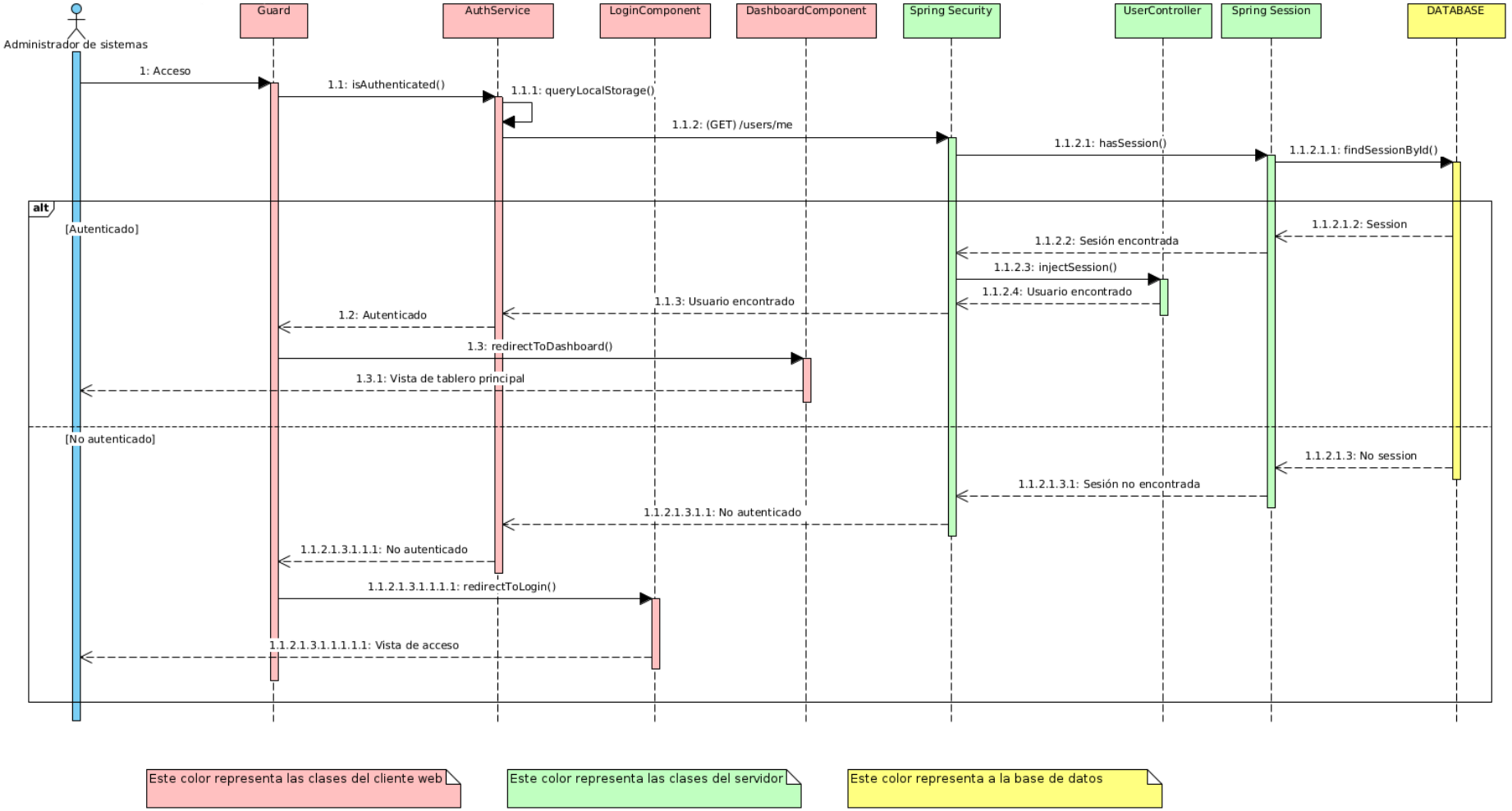

Ilustración 22. Diagrama de secuencia. Acceso al sistema

El primer objeto que nos encontramos es el objeto *Guard*. Los objetos *Guard* permiten regular el acceso a una vista u otra dependiendo del estado de autenticación del actor. Hemos omitido la dualidad entre *LoginGuard* y *AuthGuard*, porque la utilización de un objeto u otro sólo depende de la ruta por la que esté entrando el actor y añadiría complejidad al diagrama. En caso de estar autenticado, el objeto *Guard* redigirá al actor a la vista del tablero principal, *DashboardComponent*, mientras que si no está autenticado, su destino será la vista de inicio de sesión, *LoginComponent*.

El objeto *Guard* emplea el método *isAuthenticated* de *AuthService* para verificar el *token* de autenticación almacenado en la memoria local del navegador, realizando a su vez una petición REST al controlador *UserController* del servidor para obtener la identidad actualizada del usuario.

El controlador *UserController* está protegido por Spring Security, así que las peticiones REST pasarán por los filtros de seguridad del *framework* antes de ser entregadas al controlador. Spring Security pide ayuda a Spring Session para comprobar la vigencia de la sesión perteneciente al *token* proporcionado desde el cliente. Una respuesta positiva o negativa del método *findSessionById* determinará el resto del flujo. Cuando el objeto *Guard* reciba dicha respuesta, redirigirá al usuario a la vista en la que le corresponda estar.

3.5.2. Diagrama de secuencia: Inicio de sesión en el sistema

La siguiente ilustración muestra el diagrama de secuencia durante el inicio de sesión del actor en el sistema. El punto de partida de este diagrama es el diagrama de secuencia de acceso cuando el actor no se encuentra autenticado. Hemos sumado a este diagrama nuevos participantes en la interacción: los objetos LDAP y *SessionService*.

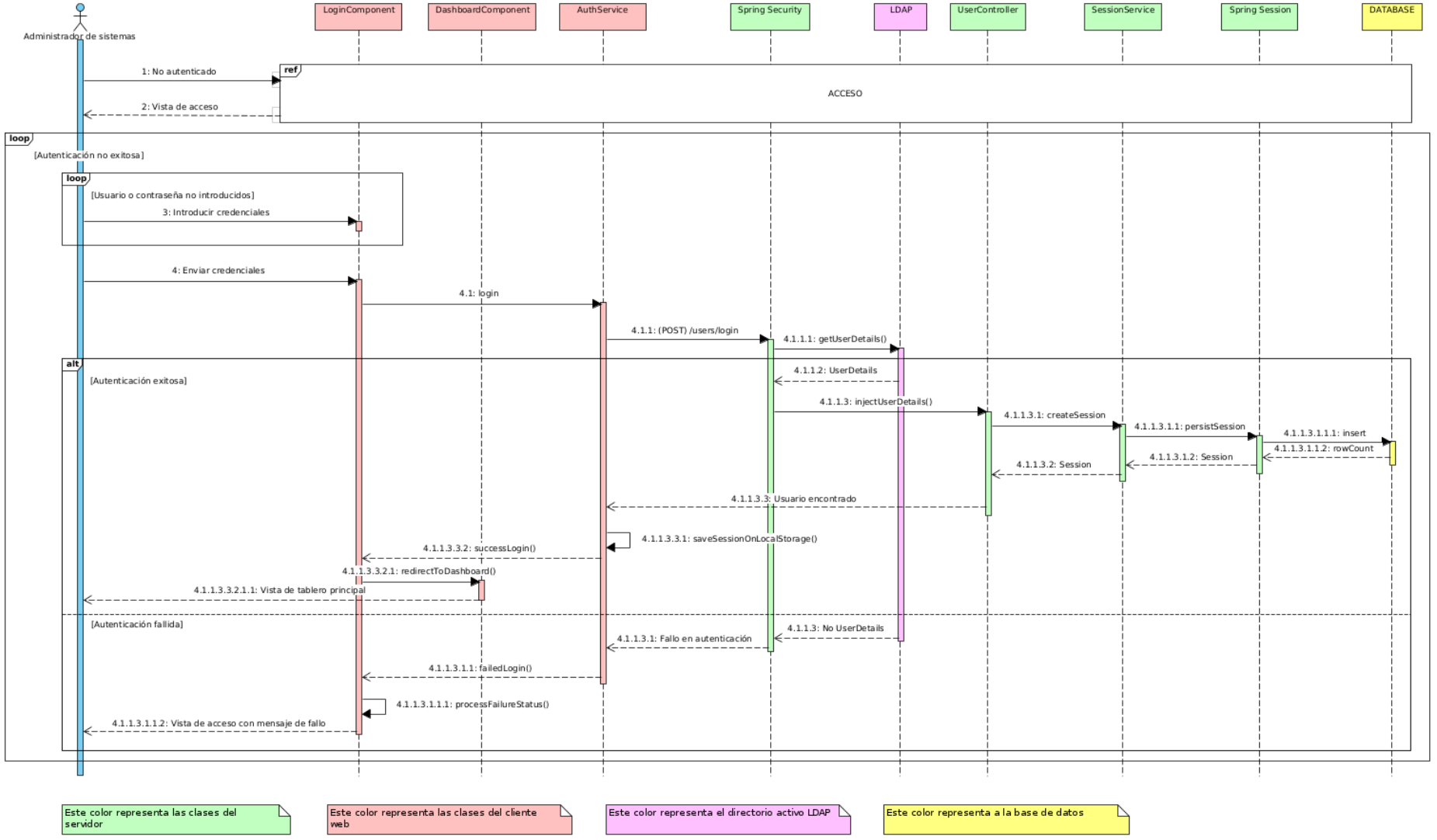

Ilustración 23. Diagrama de secuencia. Inicio de sesión al sistema

El primer paso que debe realizar el actor es la introducción de sus credenciales en la vista de acceso. Una vez introducidos y confirmados, el componente *LoginComponent* comunicará las credenciales de acceso al servicio *AuthService*. Dicho servicio preparará una solicitud de nueva sesión mediante una petición REST al servidor.

Ya en el servidor, Spring Security contrastará las credenciales proporcionadas leyendo la información del usuario en el directorio activo LDAP. En caso de que sean correctas, el servicio *SessionService* persistirá una nueva sesión en Spring Session generando un *token* de autenticación. Dicho *token*, junto con la información del usuario, serán devueltos como respuesta a la petición REST realizada por el servicio *AuthService* del cliente, que a su vez persistirá todo en el almacenamiento local del navegador. Por último, el servicio *AuthService* llevará al actor a la vista del tablero principal.

Si las credenciales proporcionadas hubieran sido incorrectas, el actor permanecería en la vista de acceso junto a un mensaje explicativo del fallo.

3.5.3. Diagrama de secuencia: Consulta de métricas de monitorización desde el cliente web

La siguiente ilustración muestra el diagrama de secuencia para la consulta de métricas de monitorización desde el cliente web. Nuestro actor accede a la vista del tablero principal a través del componente *DashboardComponent*. Dicho componente recurre al servicio *MeasurementService* con el fin de obtener ciertas métricas almacenadas en el sistema para, una vez obtenidas, presentarlas visualmente mediante gráficas y tablas al usuario.

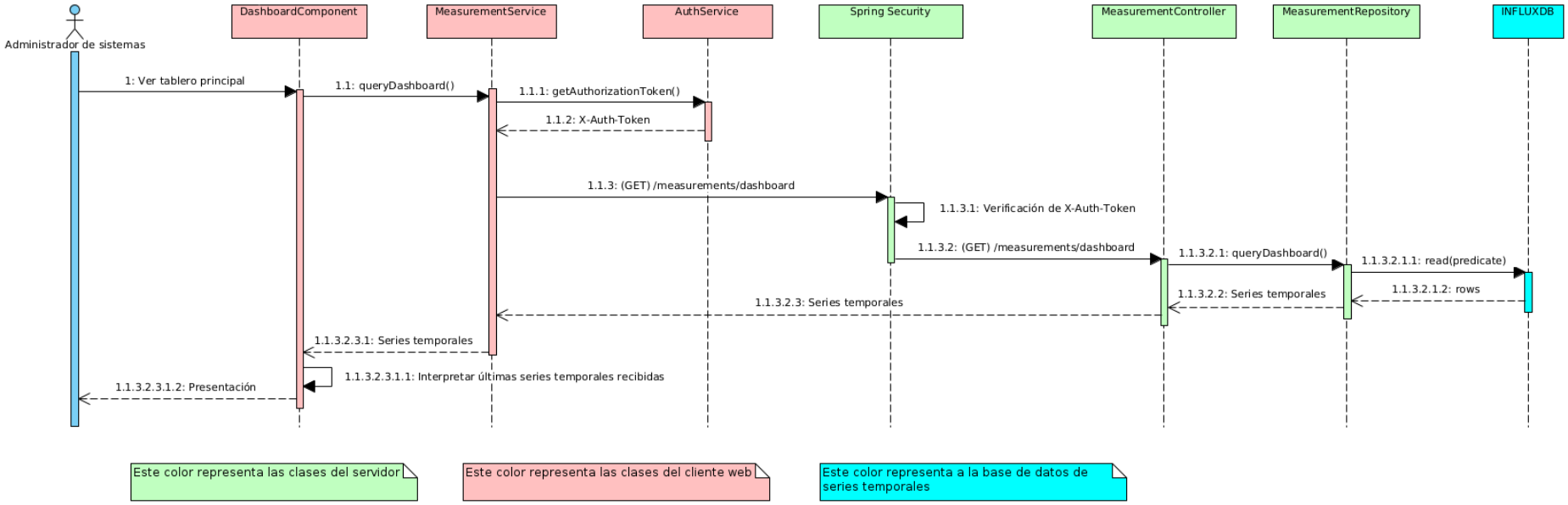

Ilustración 24. Diagrama de secuencia. Consulta de métricas de monitorización desde el cliente web

El método *queryDashboard* de *MeasurementService* obtiene el *token* de sesión del usuario gracias al servicio *AuthService* y realiza una petición específica a la interfaz de comunicación API REST del servidor.

Una vez llega la petición al servidor, Spring Security verifica la validez del *token* de sesión recibido antes de entregar la solicitud al controlador *MeasurementController*. A su vez, el controlador *MeasurementController* solicita al repositorio *MeasurementRepository* la recuperación de las métricas.

Finalmente, *MeasurementRepository* interactúa con la base de datos de series temporales InfluxDB para proceder a la recuperación de las métricas atendiendo a los criterios que acompañaban a la petición original.

3.5.4. Diagrama de secuencia: Recepción de acciones emitidas del servidor al agente

La siguiente ilustración muestra el diagrama de secuencia para la recepción de acciones emitidas del servidor al agente. Para comprenderlo adecuadamente, debemos prestar especial atención al objeto *SimpleBroker*, encargado de gestionar la interfaz de comunicación mediante colas.

Además, esta vez incluimos a dos actores: el primero, el equipo donde se encuentra instalado el agente; el segundo, el administrador de sistemas. El papel que desempeña este último es el de generar la acción pertinente desde la interfaz del cliente web, ya que el servidor no produce acciones por sí solo.

Así, dentro del diagrama podemos observar los dos eventos que desarrollan cada uno de los actores.

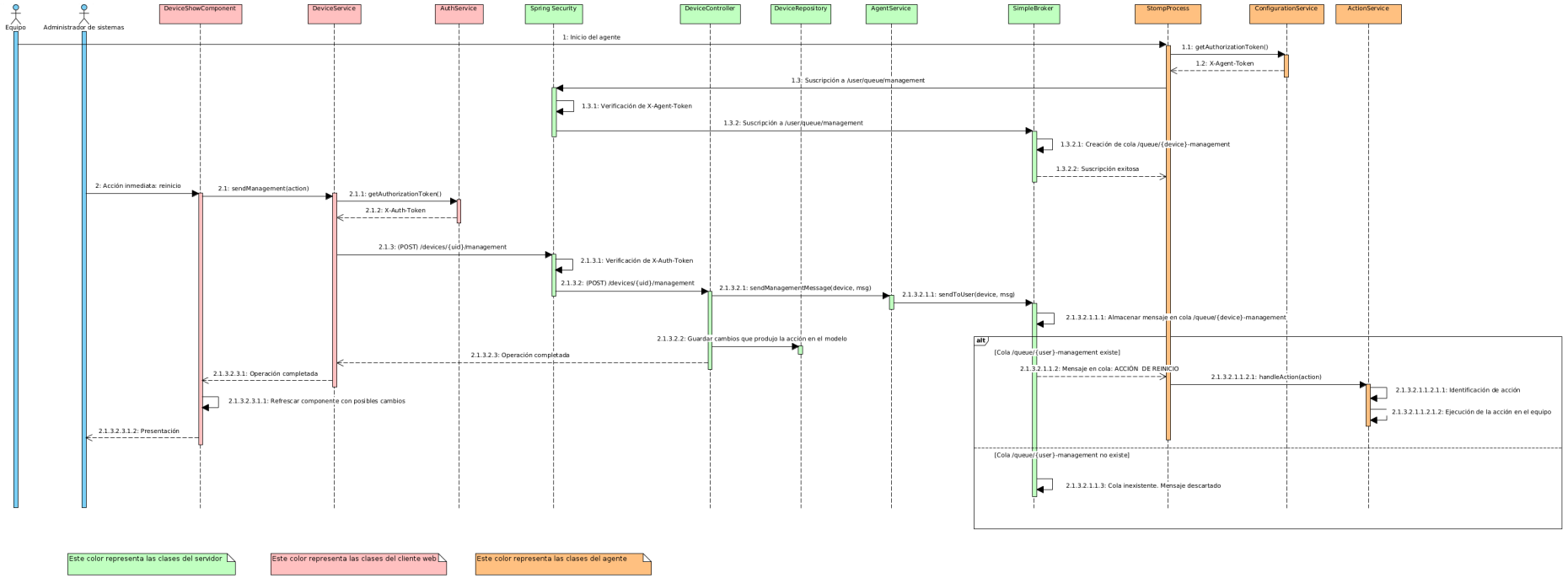

Ilustración 25. Diagrama de secuencia. Recepción de acciones emitidas del servidor al agente

En el primer evento, el proceso *StompProcess* del agente interactúa con el objeto *SimpleBroker* del servidor para suscribirse a la interfaz de colas. Como resultado de la suscripción, el servidor prepara una cola personalizada donde alojar los mensajes dirigidos al equipo asociado del agente. Dicha cola permanecerá activa mientras el agente permanezca suscrito a la misma.

En el segundo evento, el administrador de sistemas emite una acción desde el componente *DeviceShowComponent*, enviándola posteriormente al servicio *DeviceService* donde se transmite como una petición al servidor a través de la interfaz de comunicación API REST.

Ya en el servidor, y tras haber superado la verificación del *token* de sesión de Spring Security, el controlador *DeviceController* realiza dos tareas: comunica la acción al servicio *AgentService* y actualiza el modelo de datos relacional si la acción interactúa con la entidad.

Centrándonos sobre *AgentService*, éste comunica la acción en cuestión al objeto *SimpleBroker*, que lo enruta como un nuevo mensaje en la cola perteneciente al equipo destinatario. Si la cola existe –y sólo existe cuando hay un agente suscrito–, el mensaje se entrega al proceso StompProcess. En caso contrario, el mensaje se descarta sin llegar a ningún agente. Por lo tanto, las acciones contempladas tienen validez si el equipo destinatario dispone de comunicación con el servidor a través del agente en ese instante.

Finalmente, los mensajes entrantes en el proceso *StompProcess* son procesados por el agente a través del servicio *ActionService*, que identifica la acción y la ejecuta en el equipo.

3.5.5. Diagrama de secuencia: Envío de métricas de monitorización desde el agente al servidor

La siguiente ilustración muestra el diagrama de secuencia para el envío de métricas de monitorización desde el agente al servidor. A diferencia del resto de diagramas, no aparece ningún actor humano, siendo el equipo quien actúa autónomamente para la recogida y envío de métricas de monitorización.

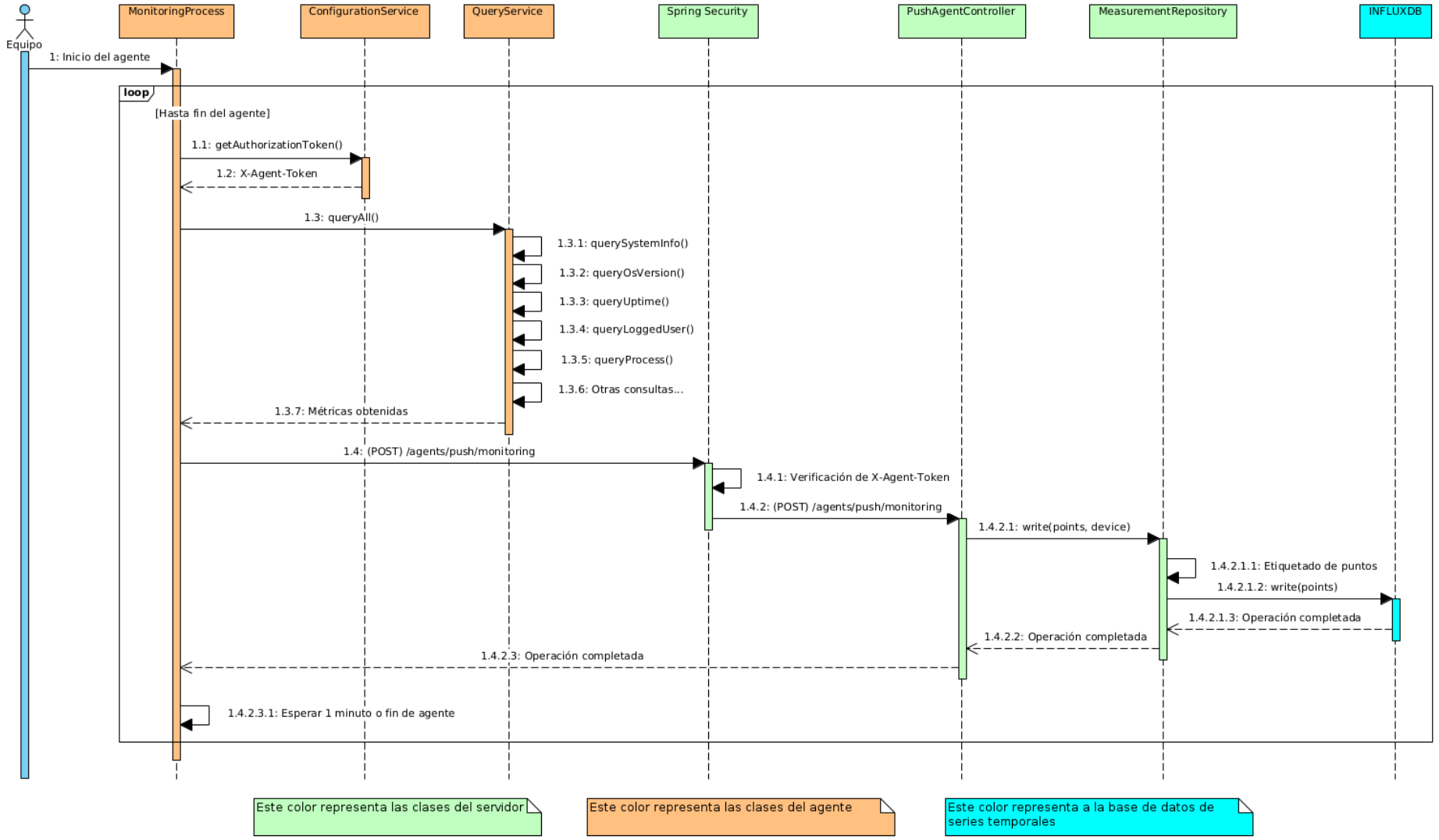

Ilustración 26. Diagrama de secuencia. Envío de métricas de monitorización desde el agente al servidor

El proceso *MonitoringProcess* trabaja en bucle durante la vida del agente. Así, cada minuto recoge métricas sobre diferentes aspectos software y hardware del equipo. En cada iteración, consulta el *token* de autenticación desde el servicio *ConfigurationService* y obtiene la mayor cantidad de métricas posibles ejecutando el método *queryAll* del servicio *QueryService* para enviarlas al servidor a través de la interfaz de comunicación API REST.

El servidor verifica el *token* de autenticación a través de Spring Security y traslada la petición al controlador *PushAgentController*, donde lleva las métricas al repositorio *MeasurementRepository* para su etiquetado y posterior almacenamiento en la base de datos de series temporales InfluxDB.

3.6. Diseño de la interfaz

El diseño de la interfaz describe las características de los elementos frontera que pueden interaccionar con los usuarios o con otros componentes software. Dichas características pueden ser tanto visuales como de comunicación.

En el caso de nuestro sistema, apreciamos tres interfaces diferentes:

- Interfaz de usuario a través del navegador web
- Interfaz de comunicación API REST
- Interfaz de comunicación mediante colas

A continuación, detallaremos el diseño de cada una de estas interfaces.

3.6.1. Interfaz de usuario a través del navegador web

Una interfaz de usuario a través del navegador web establece elementos visuales comprensibles e intuitivos que sirven como medio de interacción entre el usuario y una aplicación web.

A continuación, vamos a realizar el prototipado de la estructura visual (en inglés, *wireframe*) de nuestro cliente web. Los *wireframes* presentados serán

bocetos rápidos que nos acercarán a las funcionalidades que dispondrá la interfaz de usuario desde una etapa temprana del proyecto. Serán simples y sólo representarán información relevante, no llegando a cubrir todos los detalles de la interfaz final.

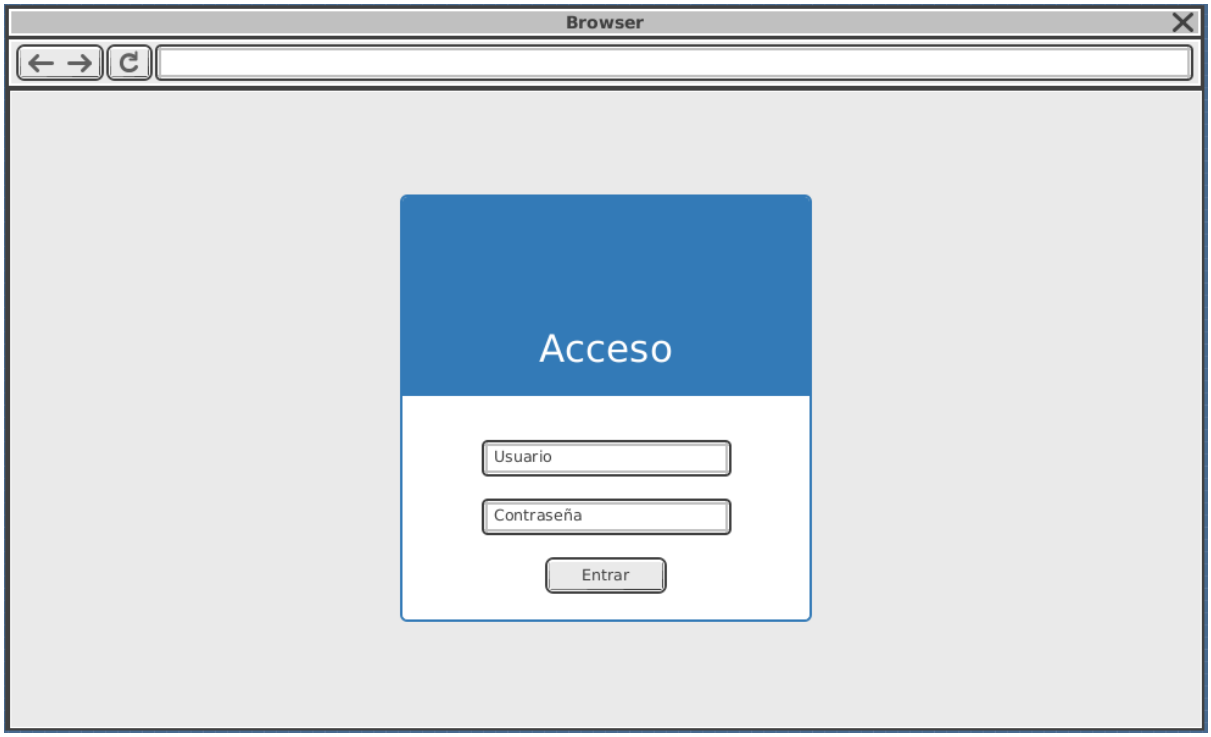

Ilustración 27. Diseño de interfaz de usuario. Inicio de sesión

En la ilustración anterior presentamos un formulario simple para que el usuario introduzca su usuario y contraseña. Si la autenticación es exitosa, el usuario pasará a la vista del tablero. En caso contrario, el usuario permanecerá en la misma vista con una alerta que ayude a guiar al usuario.

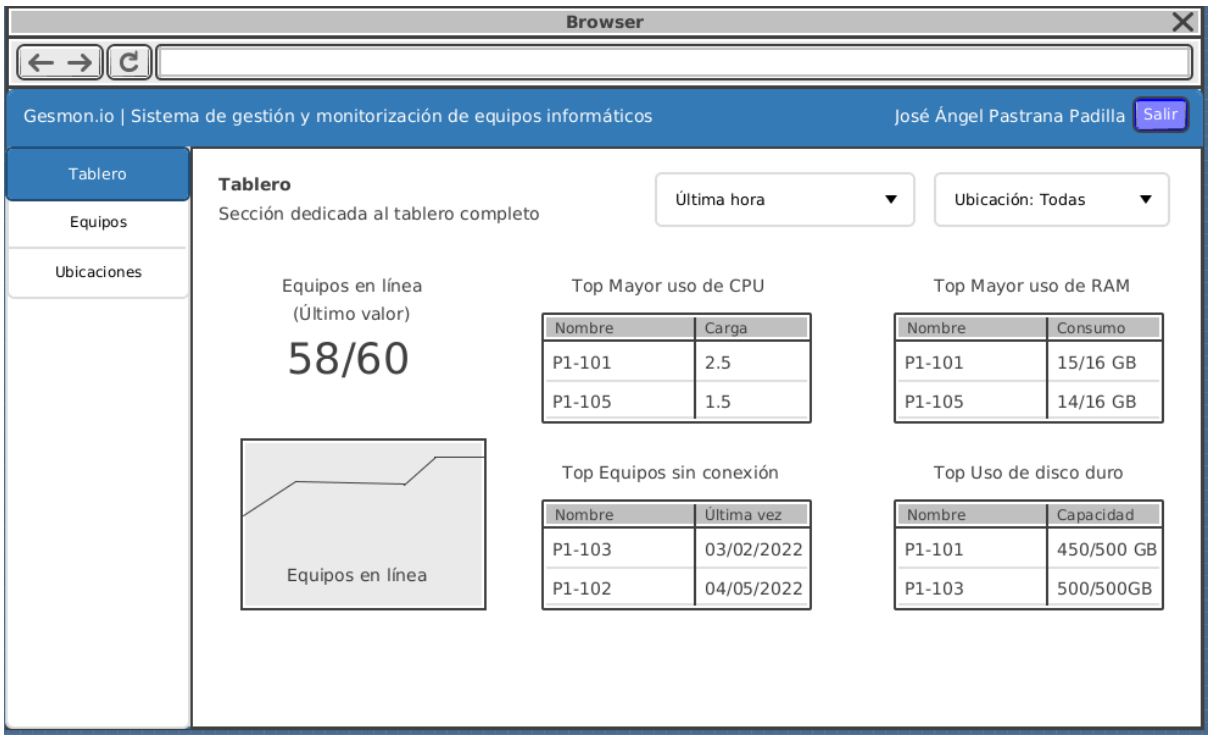

Ilustración 28. Diseño de interfaz de usuario. Vista de tablero

En la ilustración anterior presentamos un boceto de la vista principal donde se resumen las estadísticas de monitorización más relevantes entre los equipos que coinciden con el filtro de fecha y ubicación. En esta vista perseguimos detectar comportamientos anómalos de algún equipo.

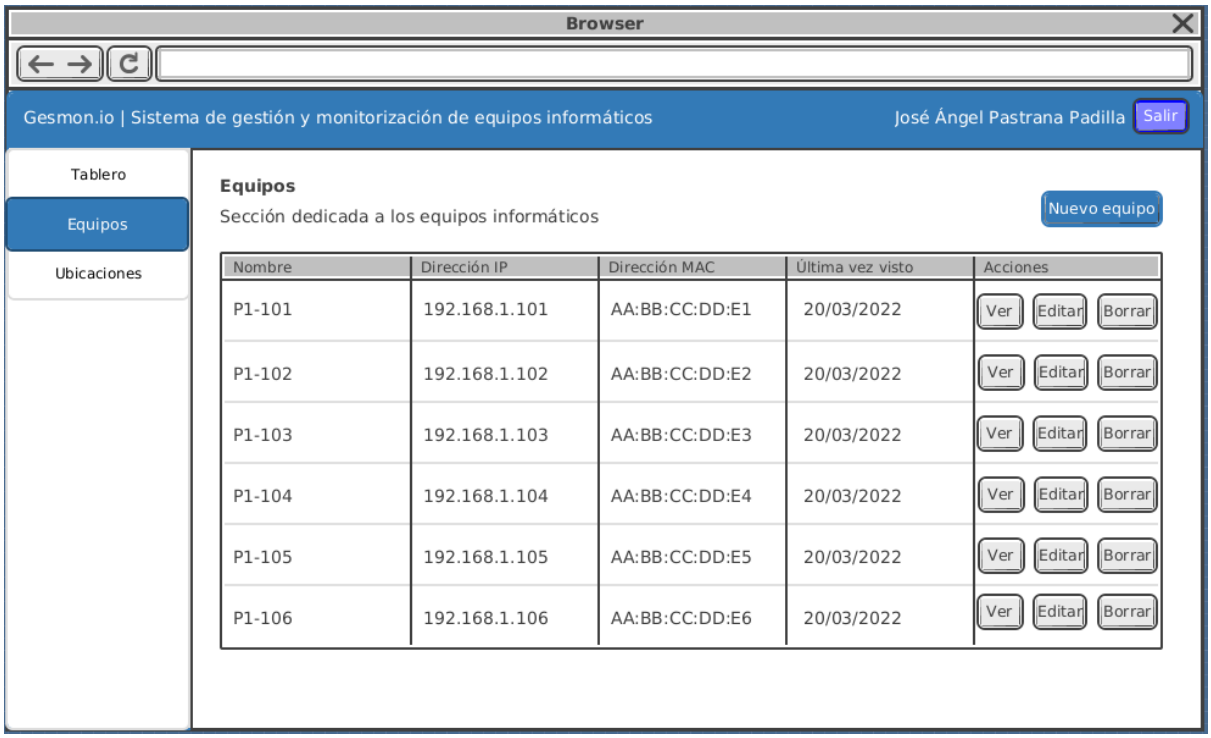

Ilustración 29. Diseño de interfaz de usuario. Vista de equipos

En la ilustración anterior presentamos un boceto de la vista de equipos donde listamos las entidades lógicas de equipos creadas por el usuario. Sobre cada equipo, podemos ampliar información, editar alguno de sus atributos o eliminarlo completamente.

Ilustración 30. Diseño de interfaz de usuario. Vista de información de un equipo

En la ilustración anterior presentamos un boceto de la información detallada de un equipo en el sistema bajo la pestaña “Información”. En dicha pestaña, podemos consultar métricas estáticas de monitorización enviadas por el agente y también emitir comandos de gestión al mismo.

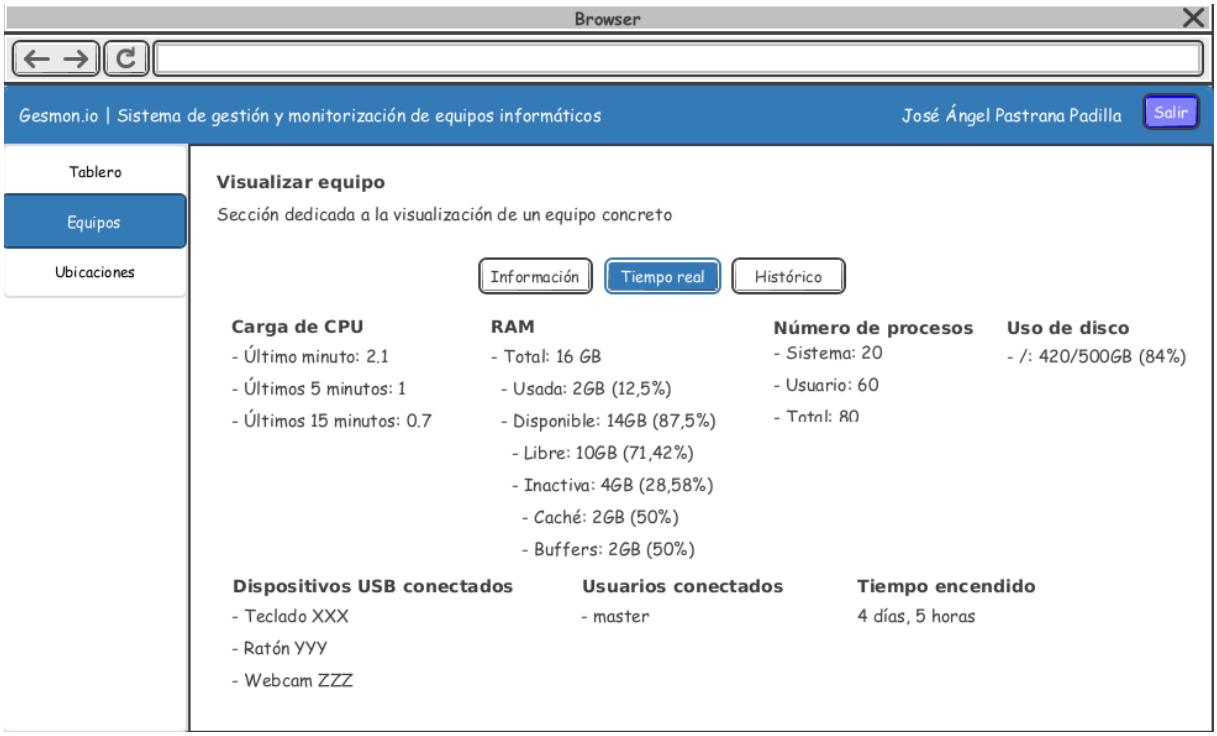

Ilustración 31. Diseño de interfaz de usuario. Vista de tiempo real de un equipo

En la ilustración anterior presentamos un boceto de la información detallada de un equipo en el sistema bajo la pestaña “Tiempo real”. En dicha pestaña, podemos consultar la última métrica de monitorización recibida si el equipo se encuentra encendido. En caso de que el equipo se encuentre apagado, esta vista aparecería sin datos.

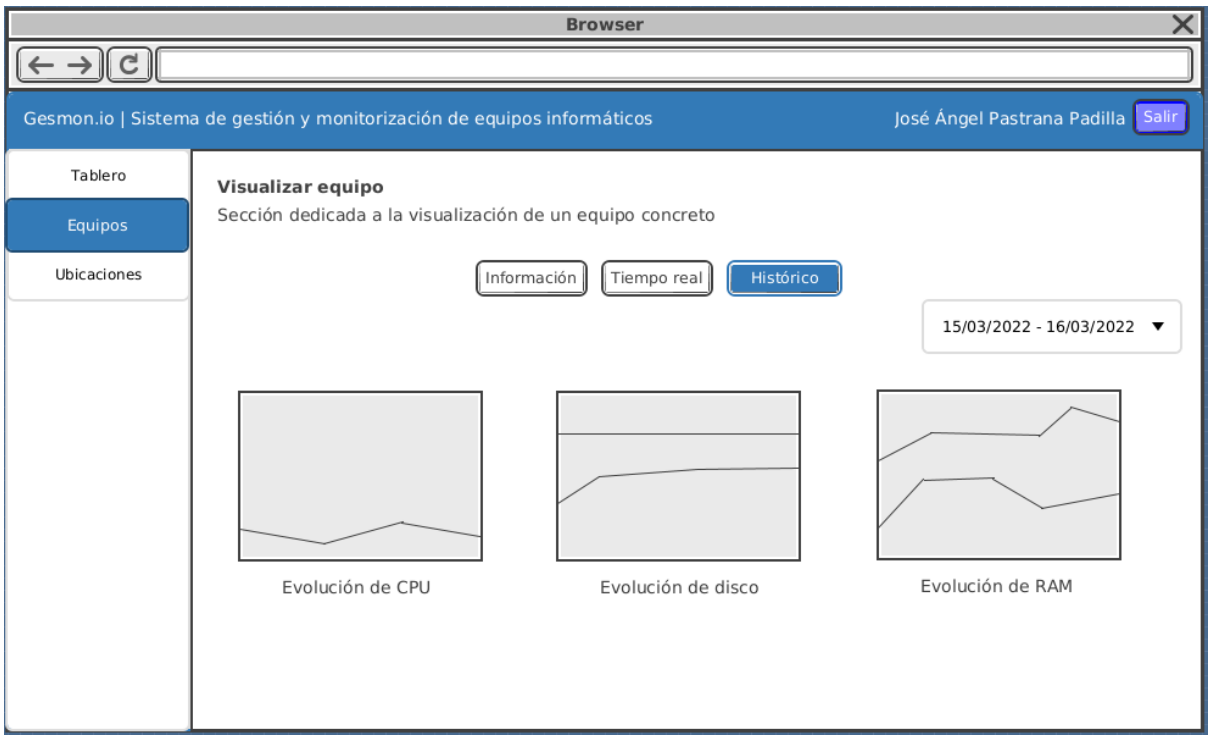

Ilustración 32. Diseño de interfaz de usuario. Vista de histórico de un equipo

En la ilustración anterior presentamos un boceto de la información detallada de un equipo en el sistema bajo la pestaña “Histórico”. En dicha pestaña, podemos consultar el histórico de algunas métricas de monitorización mediante gráficas que muestran la evolución de las métricas a lo largo del tiempo.

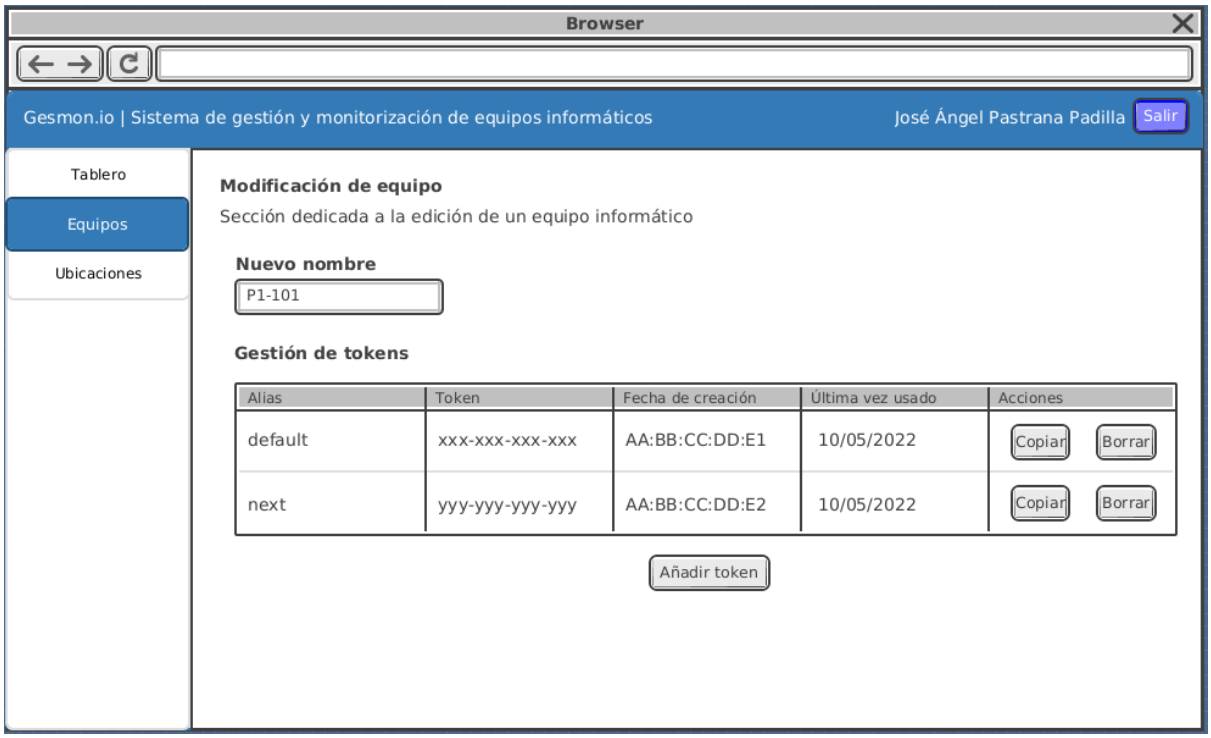

Ilustración 33. Diseño de interfaz de usuario. Vista de modificación de un equipo

En la ilustración anterior presentamos un boceto de la vista de modificación de un equipo donde podemos editar los atributos del mismo y realizar la gestión de los *tokens* de autenticación del agente. Sólo necesitamos un agente por equipo y cada agente tiene asociado un único *token*, por lo que la gestión de múltiples *tokens* únicamente cubre escenarios donde planeemos una migración hacia nuevos *tokens* revocando los anteriores al finalizar.

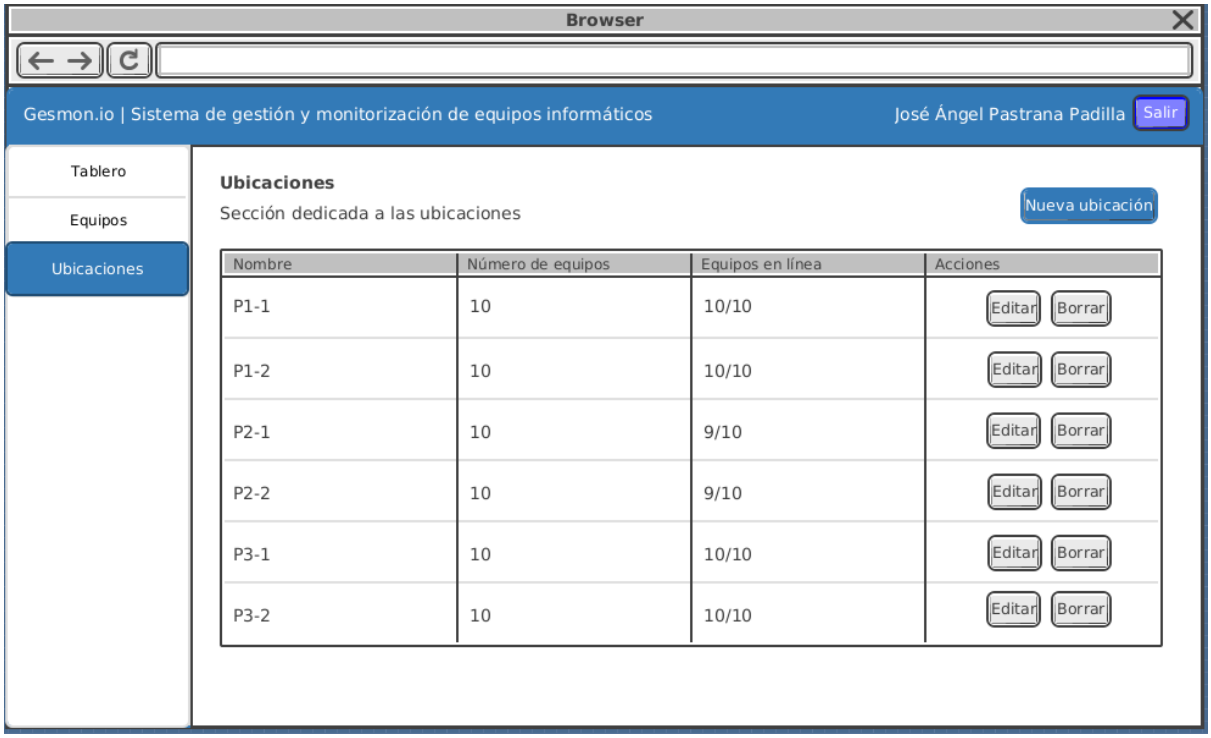

Ilustración 34. Diseño de interfaz de usuario. Vista de ubicaciones

En la ilustración anterior presentamos un boceto de la vista de equipos donde listamos las entidades lógicas de las ubicaciones creadas por el usuario. Sobre cada ubicación, podemos ampliar editar sus atributos o eliminarla completamente.

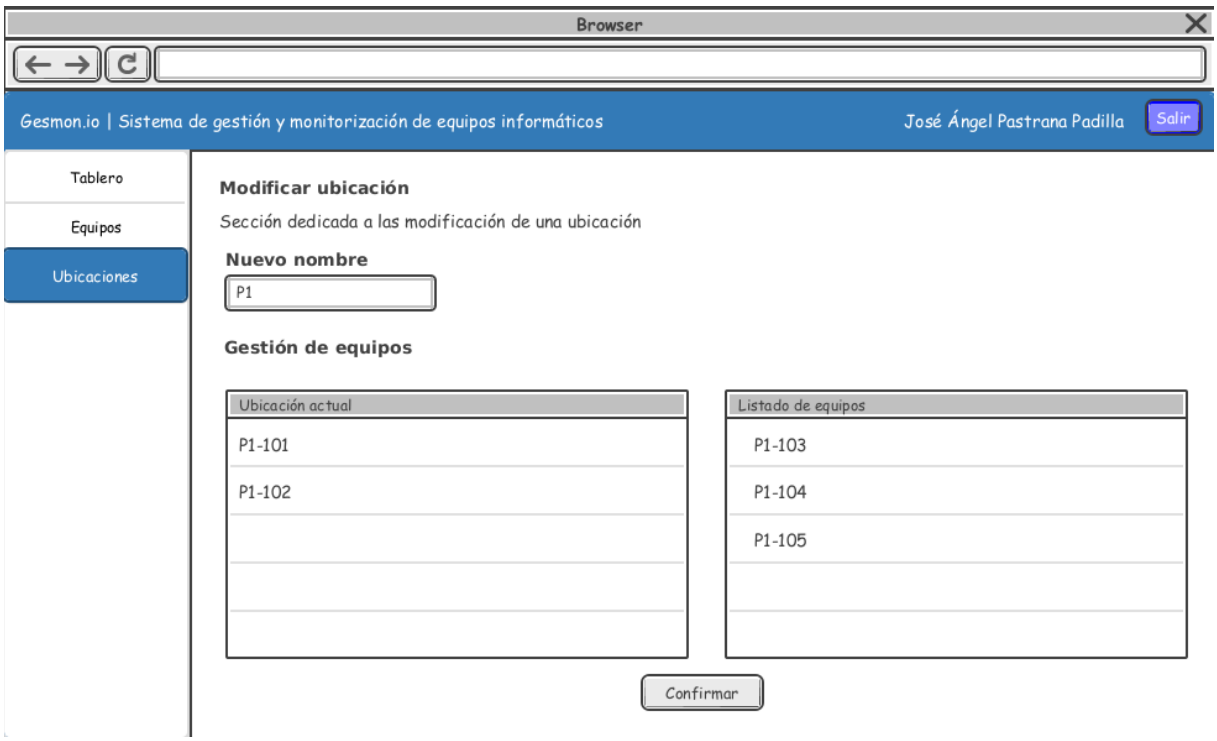

Ilustración 35. Diseño de interfaz de usuario. Modificación de ubicación

En la ilustración anterior presentamos un boceto de la vista de modificación de una ubicación donde podemos gestionar los equipos asociados.

3.6.2. Interfaz de comunicación API REST

Una interfaz de comunicación API REST permite el intercambio de mensajes entre un servidor y los clientes involucrados. Para ello, el servidor expone una serie de *endpoints* mediante sus clases controladoras a través del protocolo de comunicación HTTP.

Cada *endpoint* se compone de un método y una ruta. Los métodos describen el propósito del mensaje, donde encontramos *GET* para la consulta de un recurso, *POST* para la creación, *PATCH* para la actualización y *DELETE* para la destrucción.

Como respuesta tras ejecutar el *endpoint*, obtendremos un objeto acompañado de un código de respuesta. Los códigos de respuesta siguen una semántica para indicar cuando hubo éxito (entre 200 y 299), un error por parte de la solicitud construida por el cliente (entre 400 y 499) o un error por parte del servidor (entre 500 y 599).

Dentro de nuestro sistema, son dos los clientes que se benefician de esta interfaz de comunicación: el cliente web y el agente. Dichos clientes inician una nueva conexión con el servidor cada vez que solicitan un *endpoint*.

Todos los *endpoints* los protegeremos mediante algún tipo de autenticación. Al respecto, cabe destacar las diferentes cabeceras de autenticación: X-Auth-Token para la autenticación de las peticiones realizadas desde un cliente web a partir de una sesión previa establecida y X-Agent-Token para la autenticación de las peticiones realizadas desde el agente a partir de un *token* de autenticación asociado a la entidad del equipo.

A continuación, presentamos una tabla por cada controlador en el servidor desglosando los *endpoints* que expone cada uno.

- Controlador *UserController*

UserController es usado exclusivamente por el cliente web para la gestión de los *tokens* de sesión del usuario.

Descripción	*Endpoint*	Parámetros de entrada	Respuesta	Error
Autenticación de usuario	(POST) /users/login	• Cabecera de autenticación en formato *basic*	• (200) Objeto *Logged*	• (401/403) Objeto *Unauthorized* /*Forbidden*
Sesión activa	(GET) /users/me	• Cabecera X-Auth-Token	• (200) Objeto Logged	• (401/403) Objeto *Unauthorized* /*Forbidden*
Desautenticación de usuario	(DELETE) /users/logout	• Cabecera X-Auth-Token	• (204) Sin contenido	• (401/403) Objeto *Unauthorized* /*Forbidden*

Tabla 16. Diseño de interfaz API REST. Controlador *UserController*

- Controlador *DeviceController*

DeviceController es usado exclusivamente por el cliente web para la gestión de los equipos informáticos.

Descripción	*Endpoint*	Parámetros de entrada	Respuesta	Error
Listado de equipos	(GET) /devices	• Cabecera X-Auth-Token • page: Número de página • size: Tamaño de página • sort: Clave de ordenación	• (200) Objeto *Page<Device>*	• (401/403) Objeto *Unauthorized /Forbidden*
Obtención de equipo	(GET) /devices/{uid}	• Cabecera X-Auth-Token • uid: Identificador	• (200) Objeto *Device*	• (401/403) Objeto *Unauthorized /Forbidden* • (404) No encontrado
Creación de equipo	(POST) /devices	• Cabecera X-Auth-Token • Cuerpo con objeto parcial *Device*	• (201) Objeto *Device*	• (401/403) Objeto *Unauthorized /Forbidden* • (400) Objeto *DomainError*
Actualización de equipo	(PATCH) /devices/{uid}	• Cabecera X-Auth-Token • uid: Identificador • Cuerpo con objeto parcial *Device*	• (200) Objeto *Device*	• (401/403) Objeto *Unauthorized /Forbidden* • (400) Objeto *DomainError* • (404) No encontrado

Eliminación de equipo	(DELETE) /devices/{uid}	• Cabecera X-Auth-Token • uid: Identificador	• (204) Sin contenido	• (401/403) Objeto *Unauthorized /Forbidden* • (404) No encontrado
Envío de acción al equipo	(POST) /devices/{uid}/ management	• Cabecera X-Auth-Token • uid: Identificador • Cuerpo con objeto *Management Action*	• (200) Objeto *Device*	• (401/403) Objeto *Unauthorized /Forbidden* • (404) No encontrado

Tabla 17. Diseño de interfaz API REST. Controlador *DeviceController*

- Controlador *LocationController*

LocationController es usado exclusivamente por el cliente web para la gestión de ubicaciones.

Descripción	*Endpoint*	Parámetros de entrada	Respuesta	Error
Listado de ubicaciones	(GET) /locations	• Cabecera X-Auth-Token • page: Número de página • size: Tamaño de página • sort: Clave de ordenación	• (200) Objeto *Page<Location>*	• (401/403) Objeto *Unauthorized /Forbidden*
Obtención de ubicación	(GET) /location/{uid}	• Cabecera X-Auth-Token • uid: Identificador	• (200) Objeto *Location*	• (401/403) Objeto *Unauthorized /Forbidden* • (404) No encontrado

Creación de ubicación	(POST) /location	• Cabecera X-Auth-Token • Cuerpo con objeto parcial *Location*	• (201) Objeto *Location*	• (401/403) Objeto *Unauthorized /Forbidden* • (400) Objeto *DomainError*
Actualización de ubicación	(PATCH) /location/{uid}	• Cabecera X-Auth-Token • uid: Identificador • Cuerpo con objeto parcial *Location*	• (200) Objeto *Location*	• (401/403) Objeto *Unauthorized /Forbidden* • (400) Objeto *DomainError* • (404) No encontrado
Eliminación de ubicación	(DELETE) /location/{uid}	• Cabecera X-Auth-Token • uid: Identificador	• (204) Sin contenido	• (401/403) Objeto *Unauthorized /Forbidden* • (404) No encontrado

Tabla 18. Diseño de interfaz API REST. Controlador *LocationController*

- Controlador PushAgentController

PushAgentController es usado exclusivamente por el agente para la gestión y monitorización del equipo.

Descripción	*Endpoint*	Parámetros de entrada	Respuesta	Error
Recepción de métricas	(POST) /agents/push/ metrics	• Cabecera X-Agent-Token • Cuerpo con objetos *Point*	• (204) Sin contenido	• (401/403) Objeto *Unauthorized /Forbidden*

Sincronización del agente	(POST) /agents/push/sync	• Cabecera X-Agent-Token • Cuerpo con objetos *CapEnum*	• (200) Objetos *State*	• (401/403) Objeto *Unauthorized /Forbidden*
Suscripción a cola de comandos	(POST) /agents/push/stomp	• Cabecera X-Agent-Token	• (101) Cambio de protocolo a *Websocket*	• (401/403) Objeto *Unauthorized /Forbidden*

Tabla 19. Diseño de interfaz API REST. Controlador *PushAgentController*

- Controlador *MeasurementController*

MeasurementController es usado exclusivamente por el cliente web para la consulta de las métricas de monitorización.

Descripción	*Endpoint*	Parámetros de entrada	Respuesta	Error
Información general del equipo	(GET) /measurements/{uid}/info	• Cabecera X-Agent-Token • uid: Identificador	• (200) Objeto *FluxTable*	• (401/403) Objeto *Unauthorized /Forbidden*
Información reciente del equipo	(GET) /measurements/{uid}/recent	• Cabecera X-Agent-Token • uid: Identificador	• (200) Objeto *FluxTable*	• (401/403) Objeto *Unauthorized /Forbidden*

Histórico del equipo	(GET) /measurements/ {uid}/historic	• Cabecera X-Agent-Token • uid: Identificador • start: Opcional. Fecha de inicio • end: Opcional. Fecha de fin. • range: Opcional. Rango desde el momento actual • precision: Opcional. Número de puntos máximo	• (200) Objeto *FluxTable*	• (401/403) Objeto *Unauthorized* /*Forbidden*
Histórico para el tablero	(GET) /measurements/ dashboard	• Cabecera X-Agent-Token • location: Opcional. Filtro por ubicación • start: Opcional. Fecha de inicio • end: Opcional. Fecha de fin. • range: Opcional. Rango desde el momento actual • precision: Opcional. Número de puntos máximo	• (200) Objeto *FluxTable*	• (401/403) Objeto *Unauthorized* /*Forbidden*

Tabla 20. Diseño de interfaz API REST. Controlador *MeasurementController*

3.6.3. Interfaz de comunicación mediante colas

Una interfaz de comunicación mediante colas permite el intercambio de mensajes entre entidades productoras y consumidoras mediante un corredor de mensajes (en inglés, *broker*). Dependiendo del escenario, el enrutamiento de los mensajes puede realizarse punto a punto o mediante la suscripción de las entidades consumidoras.

En nuestro caso, emplearemos un enrutamiento mediante suscripción donde los agentes de gestión y monitorización actuarán como entidades consumidoras, mientras que el servidor adoptará conjuntamente los roles de *broker* y de productor de mensajes.

El servidor, en su rol de *broker,* instanciará las colas a las que se suscribirán los agentes. Por otro lado, como productor de mensajes, generará los mensajes que serán enviados a las colas.

Básicamente, consideraremos tres posibles aplicaciones del sistema de colas dentro de nuestro sistema: acciones por lotes, acciones individuales y acciones programadas. Definiremos las acciones por lotes como una acción que deseamos emitir simultáneamente a un conjunto de equipos, mientras que una acción individual sólo abarcaría a un equipo concreto. Por otro lado, las acciones programadas nos permitirían emitir acciones por lotes o individuales para una fecha concreta en el futuro. Tanto las acciones por lotes como las acciones programadas son un asunto que no hemos manejado ni siquiera en la propuesta de solución y del que tampoco vamos a tratar aquí, pero no dejan de ser aplicaciones tremendamente atractivas que conviene mencionar. Así pues, nos enfocaremos únicamente en la emisión de acciones individuales.

Como tal, para la implementación de las acciones individuales no necesitaremos emplear todas las características de un sistema de colas: persistencia de mensajes, tiempo de vida, priorización de entrega, etc; sino que nos limitaremos a un uso sencillo donde el servidor emitirá un mensaje dentro de una cola para ser recibido por el agente suscrito destinatario. En el caso de que no encontremos dicho agente destinatario, el mensaje se desechará, ya que pretendemos que sean mensajes con entrega inmediata y carece de sentido entregarlos más tarde.

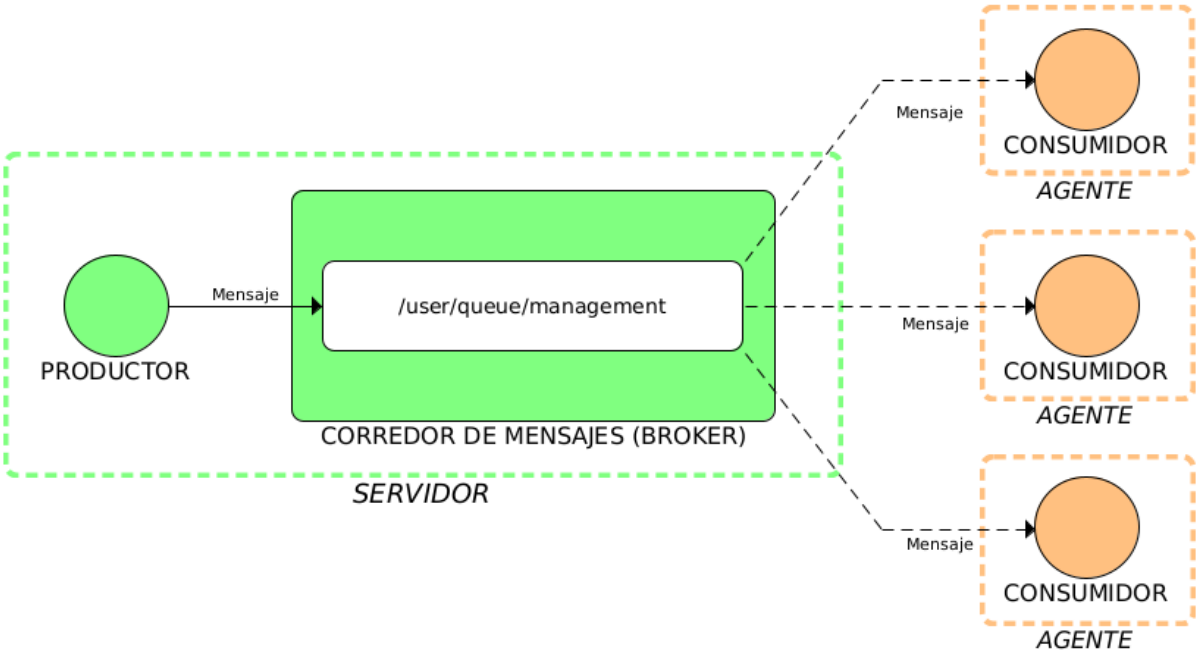

Ilustración 36. Diseño de interfaz de comunicación mediante colas

Cada agente se suscribirá a la cola */user/queue/management* empleando previamente su *token* de autenticación para la recepción de acciones individuales. En dicha cola, el agente consumirá mensajes de clase *ManagementAction* relativos al equipo en cuestión asociado con el *token* de autenticación configurado. Como puede observarse en la ilustración anterior, los mensajes viajarán exclusivamente desde el servidor (productor) al agente (consumidor).

3.7. Plan de pruebas

El plan de pruebas describe el proceso seguido para la validación del correcto funcionamiento de los componentes del sistema mediante pruebas unitarias, así como del sistema en su conjunto mediante pruebas de integración.

Los criterios de aceptación anteriormente definidos en las historias de usuario van a constituir la base principal de nuestro plan de pruebas. Efectuaremos una verificación manual de los mismos por un operador humano dada la complejidad de preparar pruebas de integración mediante alguna herramienta informática entre los componentes diseñados.

Por otro lado, incluiremos pruebas unitarias de la interfaz de comunicación API REST del servidor mediante la biblioteca de pruebas JUnit, fácilmente integrable en

Spring Framework. Para este caso, las pruebas unitarias contemplarían las operaciones de creación, lectura, modificación y actualización de las entidades del modelo relacional a través de los controladores principales y serán ejecutadas automáticamente tras cada compilación del proyecto obteniendo un resultado inmediato.

4. IMPLEMENTACIÓN

En este apartado trataremos los aspectos de implementación más importantes de los diferentes componentes del sistema. Entraremos a discutir sobre la arquitectura y los detalles de implementación del proyecto, así como también abordaremos la implicación de los diferentes *frameworks* seleccionados.

4.1. Arquitectura

La arquitectura de nuestro sistema comprende varios componentes de software: servidor, cliente web y agente. Dichos componentes los introdujimos por primera vez en el diagrama de clases, pero no llegamos a mencionar las interacciones producidas de unos componentes sobre otros. Con la finalidad de dejar claras dichas interacciones y las características internas en cada caso, detallaremos el funcionamiento de los mismos mediante diagramas arquitectónicos y diagramas de paquetes.

Un diagrama arquitectónico es una representación gráfica de las estructuras de software que componen un sistema. En ellos podemos observar el comportamiento y la relación de los elementos constituyentes, tanto si son desarrollados por nosotros mediante algún lenguaje de programación como si hemos empleado algún *framework* o motor externo.

Un diagrama de paquetes es una representación gráfica de las dependencias entre los paquetes de software dentro de un sistema.

Realizaremos un único diagrama arquitectónico que nos ofrezca una visión general de la arquitectura completa del sistema, mientras que los diagramas de paquetes nos permitirán complementar dicha visión hacia cada componente individualmente.

4.1.1. Diagrama arquitectónico

En la siguiente ilustración presentamos el diagrama arquitectónico del proyecto donde reflejamos los componentes del sistema, las relaciones que manifiestan los

unos con los otros y la pila tecnológica principal en cada uno de ellos representada por los logos de los *frameworks* y/o lenguajes de programación involucrados.

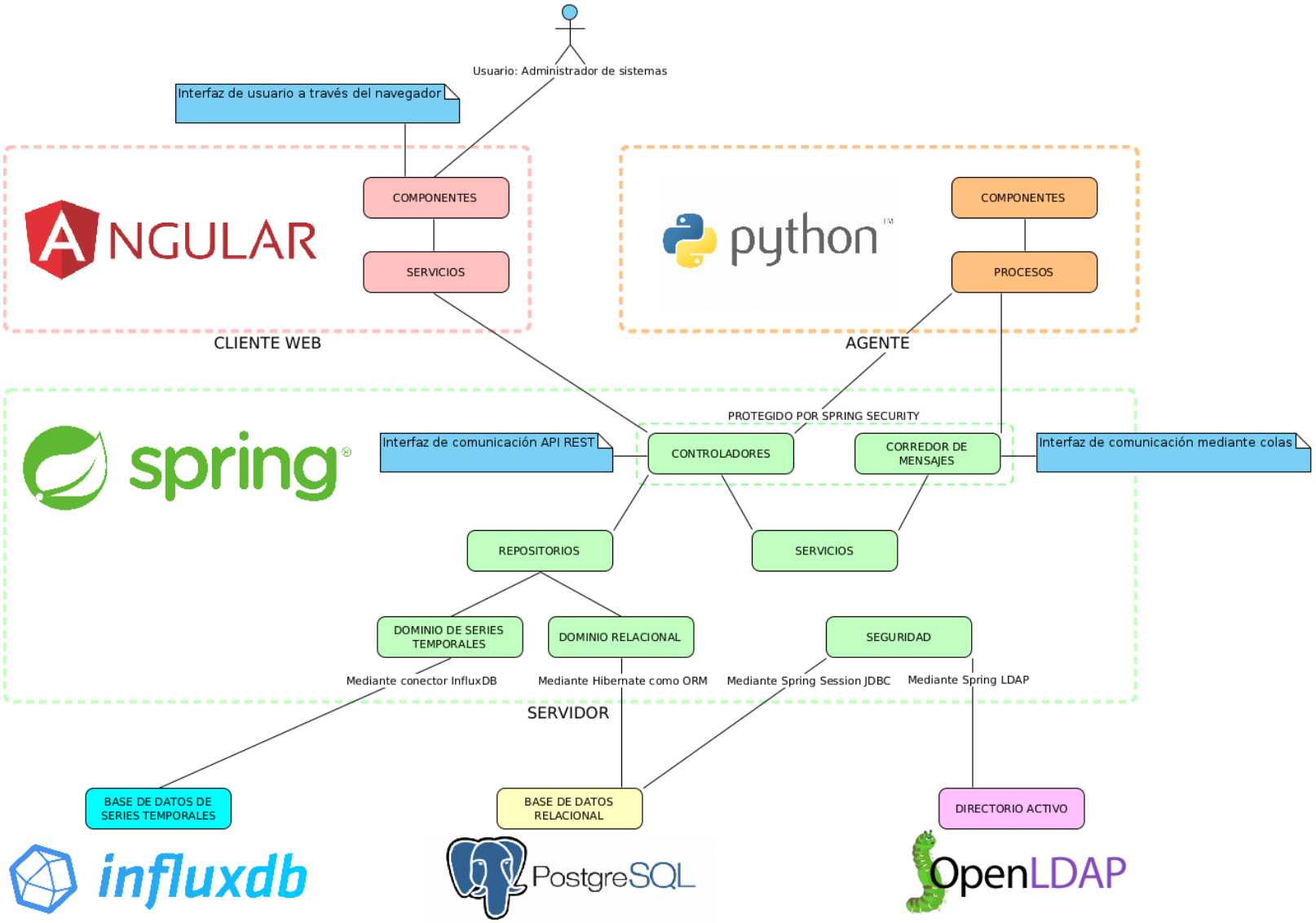

Ilustración 37. Diagrama arquitectónico del sistema

El usuario interactúa con el sistema mediante la interfaz de usuario dada en el paquete de componentes del cliente web. Luego, estas interacciones pueden llegar a transformarse en solicitudes a la interfaz de comunicación API REST desde el paquete de servicios en el cliente web al paquete de controladores en el servidor.

El agente se relaciona con el servidor desde el paquete de procesos involucrando dos interacciones: una hacia el paquete de controladores para el envío de métricas a la interfaz de comunicación API REST y otra hacia el corredor de mensajes para suscribirse a la interfaz de comunicación mediante colas. Este último no es un paquete, aunque lo hemos representado como tal por legibilidad.

El servidor emplea el paquete de repositorios para acceder a los modelos de datos tanto relacionales como de series temporales. Por otro lado, el paquete de seguridad establece la configuración para proteger los controladores y el corredor de mensajes mediante *tokens* de sesión y *tokens* de autenticación respectivamente.

Finalmente, los datos del sistema residen en una base de datos relacional para las entidades del modelo del dominio relacional, en una base de datos de series temporales para las métricas de monitorización y en un directorio activo como base de usuarios.

4.1.2. Diagrama de paquetes en el servidor

El servidor satisface las funciones de interfaz de comunicación API REST, interfaz de comunicación mediante colas, integración con los modelos de datos y autenticación de usuarios y agentes. Para todo ello, hemos usado Spring Framework como base para construir este componente.

A continuación, extenderemos el diagrama arquitectónico definiendo los paquetes anteriormente mostrados en el servidor y agregaremos otros paquetes que no fueron cubiertos en el mismo a través de las siguientes ilustraciones.

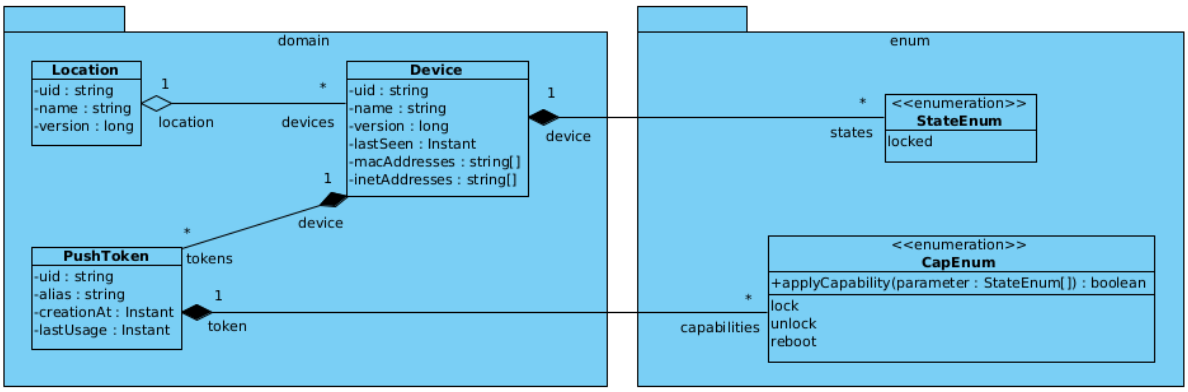

Ilustración 38. Diagrama de paquetes en el servidor. Dominio relacional

En la ilustración anterior mostramos el diagrama de paquetes en el dominio relacional llevado a término mediante objetos simples de programación (en inglés, por el acrónimo *POJO*, *Plain Old Java Object*). Las clases contenidas en estos paquetes guardan una especial relación con el diagrama entidad-relación, ya que se mantienen enlazados al modelo relacional gracias al ORM, Hibernate.

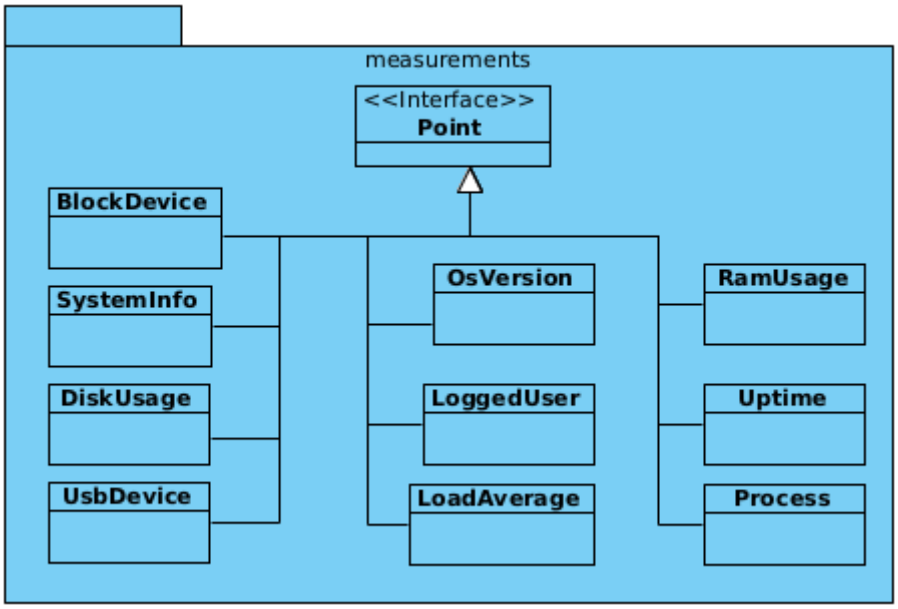

Ilustración 39. Diagrama de paquetes en el servidor. Dominio de series temporales

En la ilustración anterior mostramos el diagrama de paquetes en el dominio de series temporales. Al igual que ocurre con las clases del dominio relacional, estas clases también son *POJOs*, pero además implementan la interfaz *Point* para consultar las etiquetas y campos durante la serialización y deserialización en la base de datos de series temporales. Por otro lado, dado que los atributos de estas clases

son los mismos que los campos vistos en el diagrama de series temporales, los hemos omitido por legibilidad.

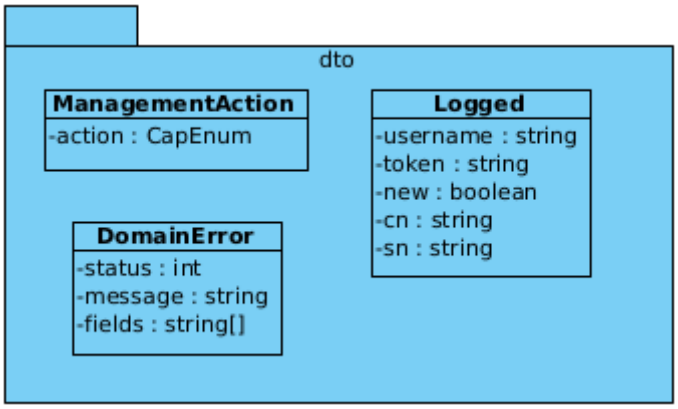

Ilustración 40. Diagrama de paquetes en el servidor. *DTO*

En la ilustración anterior mostramos el diagrama de paquetes con las clases *DTO*. Como su nombre en inglés sugiere, *Data Transfer Object*, dichas clases ofrecen interoperabilidad entre los diferentes paquetes del servidor mediante objetos fácilmente serializables. Así pues, estas clases son empleadas por los controladores del servidor para emitir respuestas o recibir datos.

Cuando pretendamos enviar una acción al agente desde la interfaz de comunicación API REST, emplearemos la clase *ManagementAction* con la acción detallada por el atributo *action*.

Cuando un usuario se autentique en el sistema correctamente, utilizaremos la clase *Logged* con los datos recuperados del directorio activo y un *token* de sesión. El atributo *token* permitirá a un usuario legítimo la interacción con el servidor hasta un tiempo máximo de inactividad definido en el *framework*.

Cuando ocurra un error durante el manejo del modelo de datos en la interfaz de comunicación API REST, utilizaremos la clase *DomainError* para proporcionar un mensaje descriptivo del fallo.

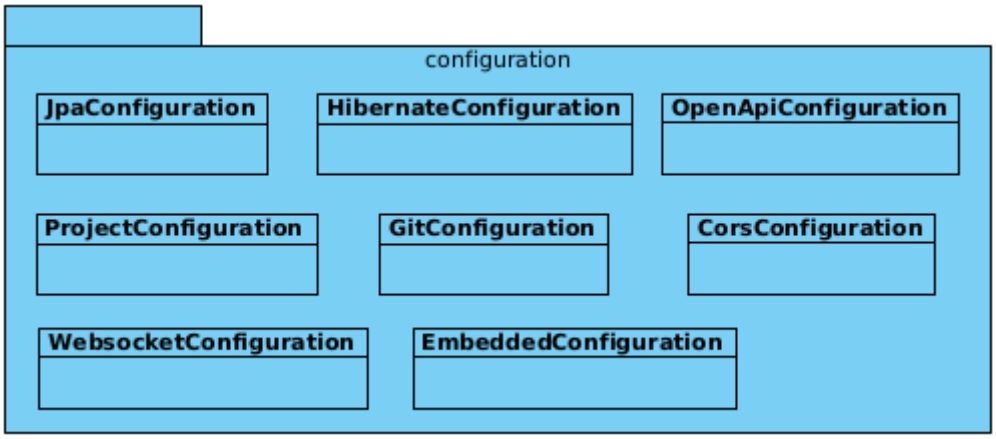

Ilustración 41. Diagrama de paquetes en el servidor. Configuración

En la ilustración anterior mostramos el diagrama de paquetes necesario para la configuración de los aspectos vitales del *framework* en el servidor. Hemos omitido los atributos y métodos de estas clases, ya que su lectura no reviste de importancia para su discusión:

JpaConfiguration apunta cuál es el paquete donde se almacenan los repositorios que implementan el patrón DAO (del acrónimo en inglés, *Data Access Object*).

HibernateConfiguration realiza ajustes adicionales en el ORM Hibernate.

OpenApiConfiguration prepara una página web interactiva que documenta y permite verificar las rutas expuestas en la interfaz de comunicación API REST.

GitConfiguration lee los metadatos existentes desde el control de versiones para ofrecerlos dentro del proyecto.

ProjectConfiguration inyecta metadatos sobre la compilación del proyecto en el compilado resultante.

CorsConfiguration establece la política de origen cruzado de peticiones en la interfaz de comunicación API REST para todos los controladores.

WebsocketConfiguration establece los puntos de entrada para los agentes en la interfaz de comunicación mediante colas.

EmbeddedConfiguration prepara los servicios embebidos necesarios para la ejecución del proyecto en el perfil de desarrollo utilizando Docker.

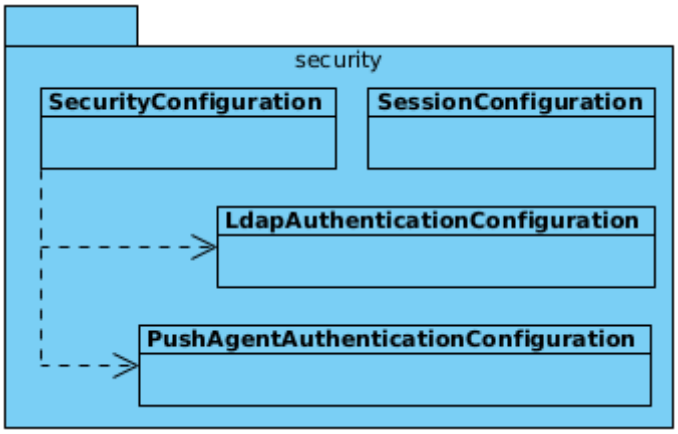

Ilustración 42. Diagrama de paquetes en el servidor. Seguridad

En la ilustración anterior mostramos el diagrama de paquetes necesario para la configuración de los aspectos de seguridad en las interfaces de comunicación del servidor. Hemos omitido los atributos y métodos de estas clases, ya que su lectura no reviste de importancia para su discusión.

SecurityConfiguration establece las políticas de seguridad mediante la biblioteca Spring Security del propio *framework*. Además, dicha clase depende de las clases *LdapAuthenticationConfiguration* y *PushAgentAuthenticationConfiguration* para completar la configuración de seguridad.

LdapAuthenticationConfiguration establece los criterios de lectura del directorio activo LDAP para identificar a los usuarios en el sistema.

PushAgentAuthenticationConfiguration establece los criterios de lectura de los *tokens* de autenticación almacenados en la entidad *PushToken* para identificar a los agentes en el sistema.

SessionConfiguration habilita el almacenamiento en la base de datos relacional de los *tokens* de sesión producidos durante la fase de autenticación de los usuarios mediante la biblioteca Spring Session JDBC del propio *framework*.

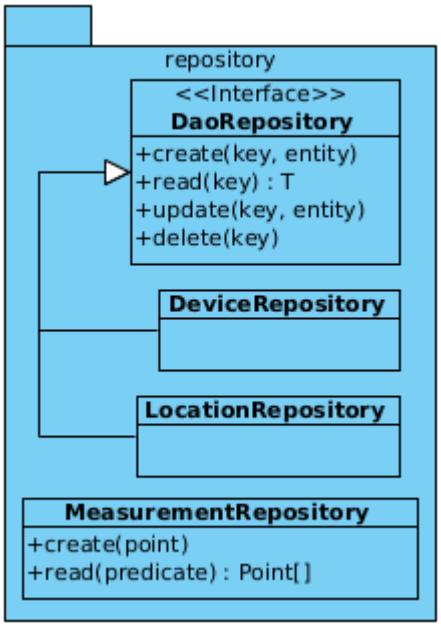

Ilustración 43. Diagrama de paquetes en el servidor. Repositorios

En la ilustración anterior mostramos el diagrama de paquetes de los repositorios para los diferentes modelos de datos.

Las entidades relacionales implementan el patrón DAO (en inglés, *Data Access Object*); en otras palabras, implementan una interfaz genérica para el acceso de los datos almacenados. En dicha interfaz definimos un conjunto de operaciones para la creación, lectura, modificación y eliminación de datos, conocidas también por el acrónimo *CRUD* (en inglés, *Create, Read, Update, Delete*). En el caso de Spring Framework, la interfaz DAO viene suministrada desde la biblioteca Spring JPA Data del propio *framework* con las operaciones CRUD comunes ya implementadas a través de la clase *JpaRepository*.

En el caso de las series temporales, la clase *MeasurementRepository* implementa la creación y lectura de puntos.

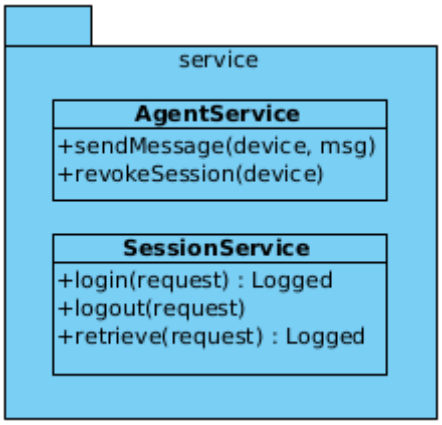

Ilustración 44. Diagrama de paquetes en el servidor. Servicios

En la ilustración anterior mostramos el diagrama de paquetes de los servicios que proporcionan alguna funcionalidad en la lógica de negocio del sistema. Las clases contenidas son empleadas por los controladores o por otros servicios.

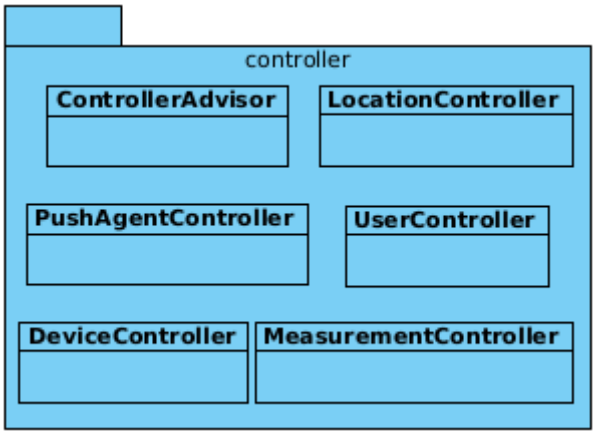

Ilustración 45. Diagrama de paquetes en el servidor. Controlador

En la ilustración anterior mostramos el diagrama de paquetes de los controladores en el servidor. Cada uno de estos controladores se componen por diferentes métodos (omitidos por legibilidad) que, sumados todos, exponen las rutas que conforman la interfaz de comunicación API REST.

A continuación, como última ilustración, mostraremos el diagrama de paquetes completo del servidor, donde a diferencia de los anteriores, en éste podrán observarse las interacciones de unos paquetes con otros.

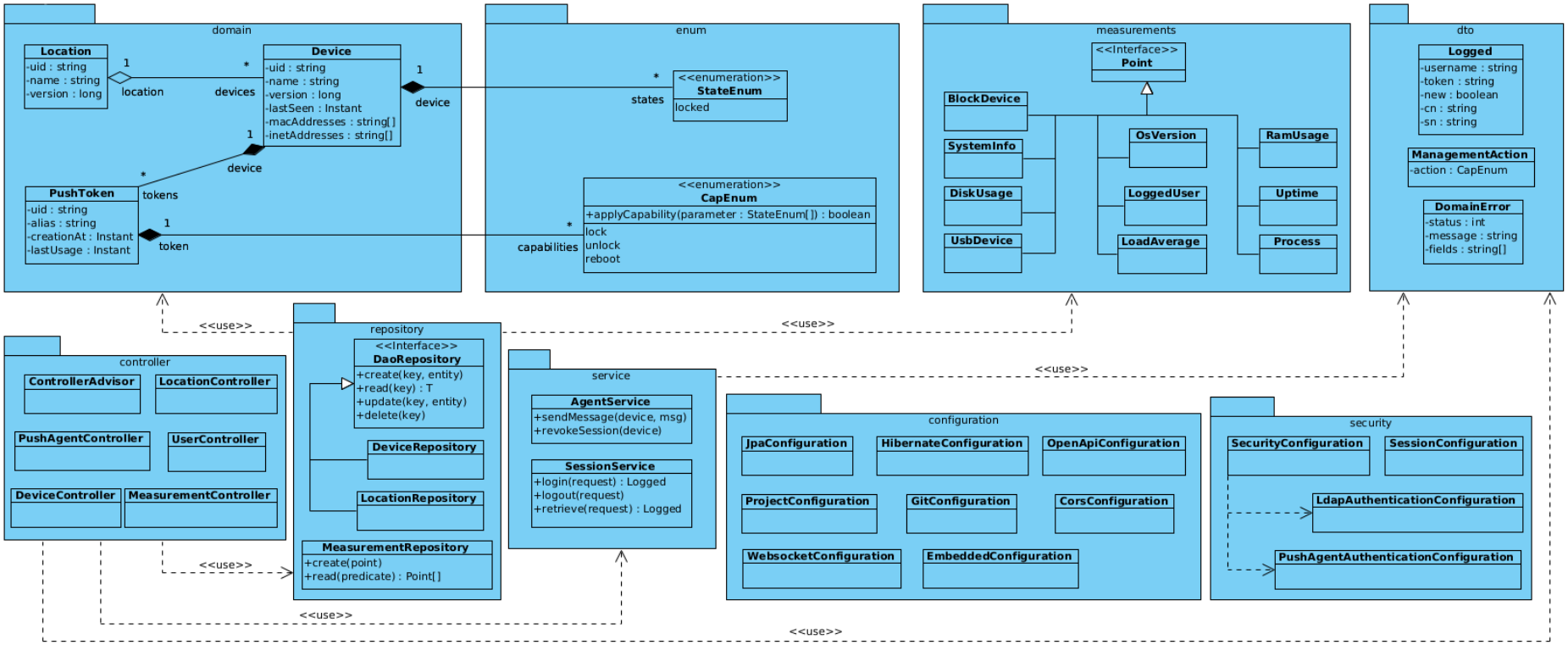

Ilustración 46. Diagrama de paquetes completo en el servidor

Entre las interacciones que podemos observar en la ilustración anterior se encuentran, por ejemplo, el paquete *repository* haciendo uso del paquete *domain* (base de datos relacional) y del paquete *measurements* (base de datos de series temporales). O también, desde el paquete *controller* hacia los paquetes *service*, *repository* y *dto*.

4.1.3. Diagrama de paquetes en el cliente web

El cliente web satisface la función de interfaz de usuario del sistema. Hemos usado el *framework* Angular como base para construir este componente.

Dentro del *framework*, nos regimos principalmente por dos tipos de clases: componentes y servicios. Los componentes se relacionan directamente con el modelo de objetos del navegador, mientras que los servicios intermedian para otorgar funcionalidades a otras clases como la solicitud de recursos a un servidor remoto.

Para ejemplificar mejor el diagrama arquitectónico y facilitar su comprensión, habíamos ilustrado las clases de componentes y de servicios mediante paquetes; pero en realidad, no se encuentran distribuidas así. Tal y como veremos en este apartado, hemos construido diferentes paquetes en torno a las funcionalidades del cliente web donde éstos son los que contienen dichas clases.

A continuación, cubriremos el diagrama de paquetes del cliente web a través de las siguientes ilustraciones.

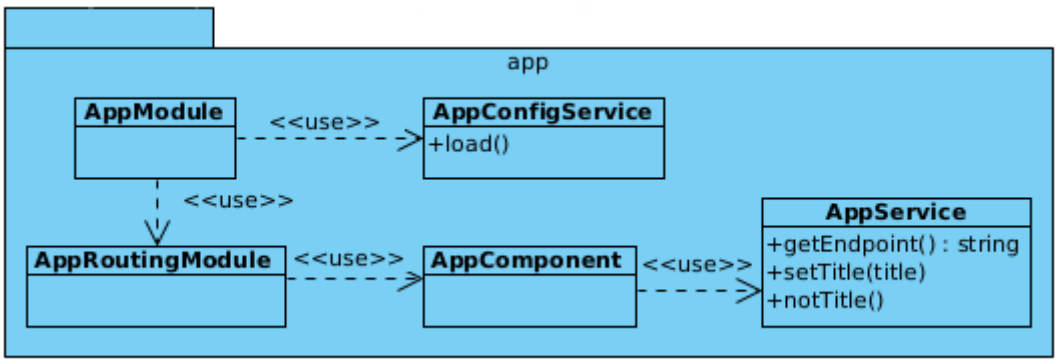

Ilustración 47. Diagrama de paquetes en el cliente web. Aplicación

En la ilustración anterior mostramos el diagrama de paquetes que sirve como punto de partida del cliente web. La clase que inicia toda la aplicación es *AppModule*, que a su vez solicita la lectura de los parámetros de configuración a través de la clase *AppConfigService*. El enrutamiento en el navegador lo establecemos desde la clase AppRoutingModule, que además cargará bajo demanda el resto de clases componentes distribuidas en otros paquetes, siendo por defecto la clase *AppComponent*. Por otro lado, AppService es un servicio general con métodos que pueden ser empleados desde cualquier clase componente. Este paquete, sus clases y sus nombres forman parte de las convenciones seguidas por la comunidad de desarrolladores web a la hora de construir aplicaciones con Angular.

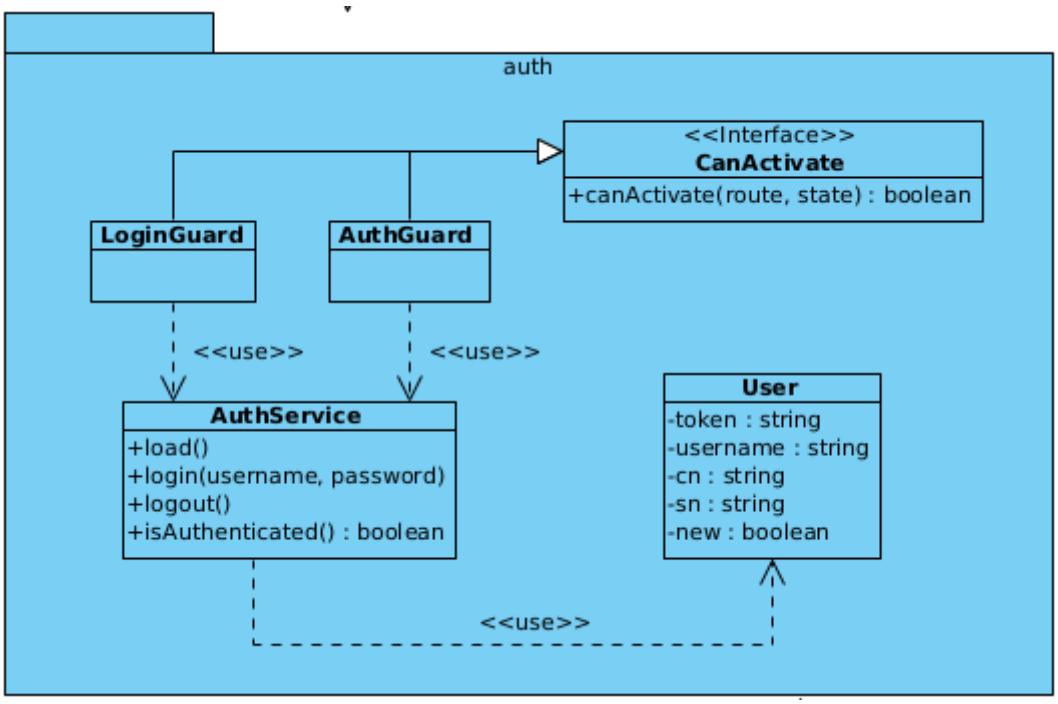

Ilustración 48. Diagrama de paquetes en el cliente web. Autenticación

En la ilustración anterior mostramos el diagrama de paquetes que permite establecer un sistema de autenticación en el cliente web. Las clases *LoginGuard* y *AuthGuard* restringen el acceso de ciertas rutas en el navegador según el nivel de autenticación en el que se encuentre el usuario. Ambas emplean a la clase *AuthService* para conocer el estado de autenticación actual.

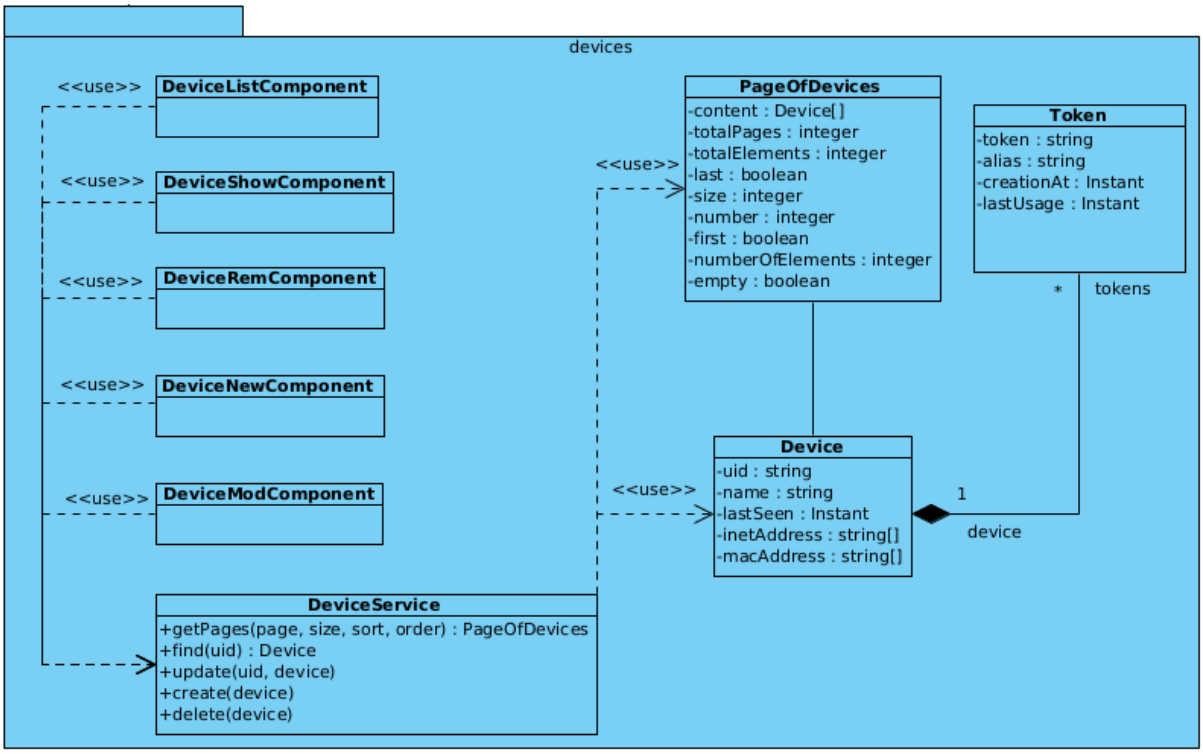

Ilustración 49. Diagrama de paquetes en el cliente web. Equipos

En la ilustración anterior mostramos el diagrama de paquetes empleado para la visualización de los equipos informáticos. Tenemos diferentes componentes en torno a las acciones que nos planteemos realizar: ver una lista de equipos, entrar en detalle sobre un equipo en concreto, modificar los atributos de un equipo, crear un equipo nuevo o eliminar algún equipo del sistema. Los componentes se sustentan a través de la clase *DeviceService* para realizar las acciones pertinentes de búsqueda, creación, modificación o eliminación en el servidor.

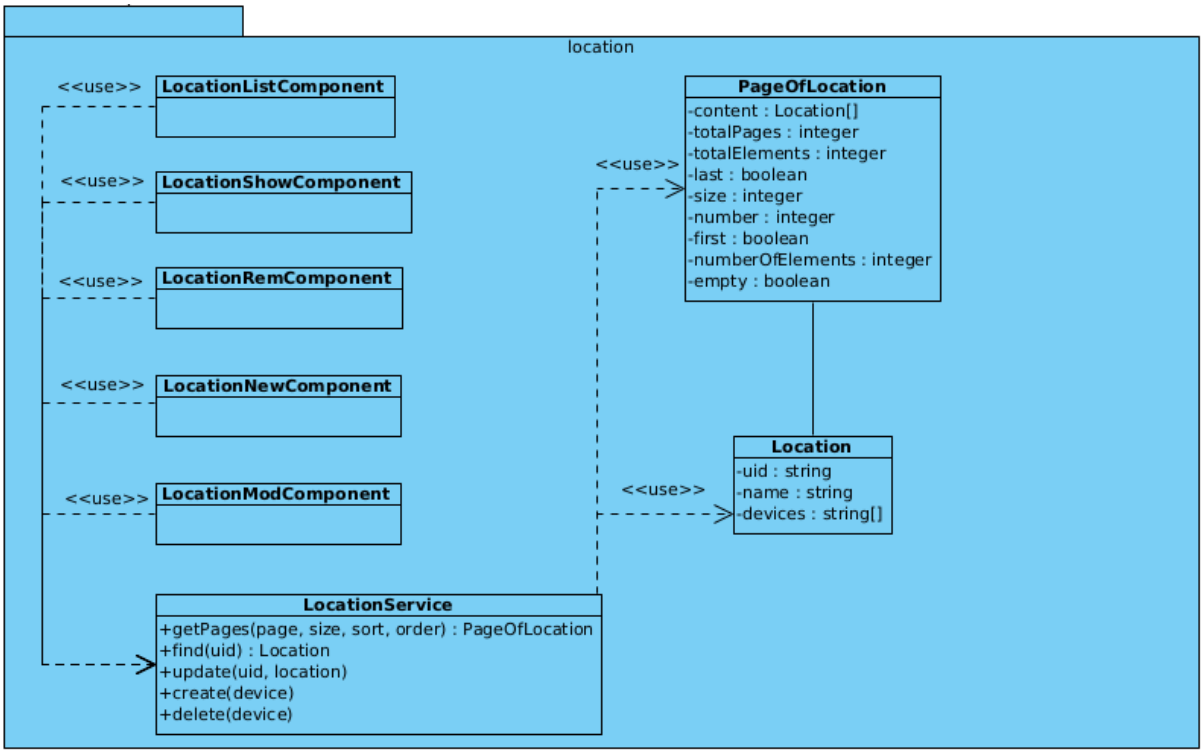

Ilustración 50. Diagrama de paquetes en el cliente web. Ubicaciones

En la ilustración anterior mostramos el diagrama de paquetes empleado para la presentación de las ubicaciones físicas. Este diagrama presenta algunas analogías con respecto al diagrama de paquetes para visualizar equipos. Evidentemente, el contenido de las clases componentes es diferente para reflejar sus propias particularidades. Cabe destacar que el modelo de datos *Location* conecta con el modelo de datos *Device* a través de su identificador.

A continuación, como última ilustración, mostraremos el diagrama de paquetes completo del cliente web, donde a diferencia de los anteriores, en éste podrán observarse las interacciones de unos paquetes con otros.

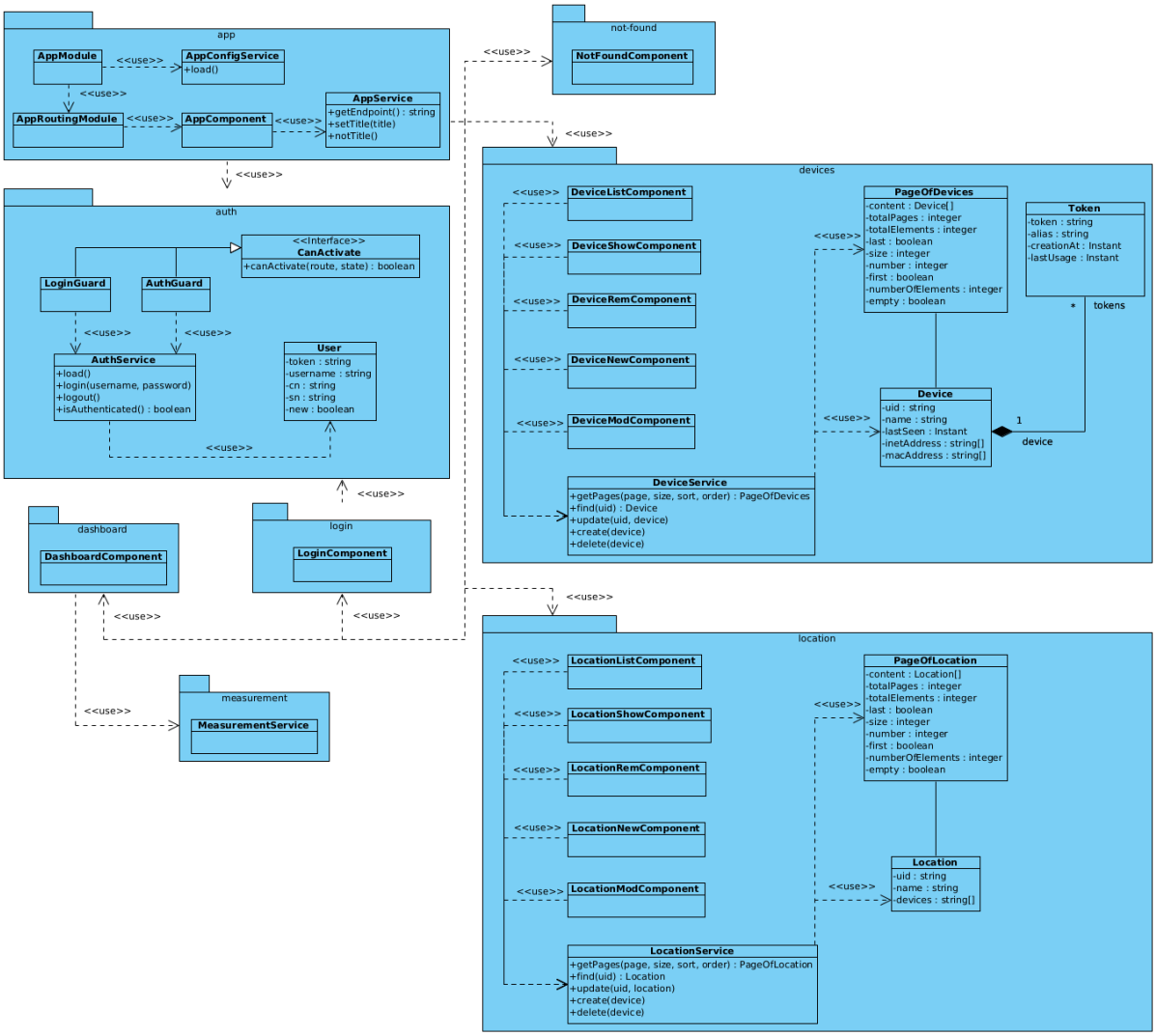

Ilustración 51. Diagrama de paquetes completo en el cliente web

Entre las interacciones que podemos observar en la ilustración anterior se encuentran, por ejemplo, el componente *LoginComponent* haciendo uso del servicio AuthService perteneciente al paquete de autenticación.

4.1.4. Diagrama de paquetes en el agente de gestión y monitorización

El agente satisface el envío de métricas de monitorización y la recepción de comandos de gestión para un equipo informático haciendo uso de las interfaces de comunicación API REST y de colas del servidor respectivamente. Hemos usado el lenguaje de programación Python junto con bibliotecas externas para su construcción, sin involucrar a ningún *framework* en la estructura de la aplicación.

A continuación, extenderemos el diagrama arquitectónico definiendo los paquetes anteriormente mostrados en el agente y agregaremos otros paquetes que no fueron cubiertos en el mismo a través de las siguientes ilustraciones.

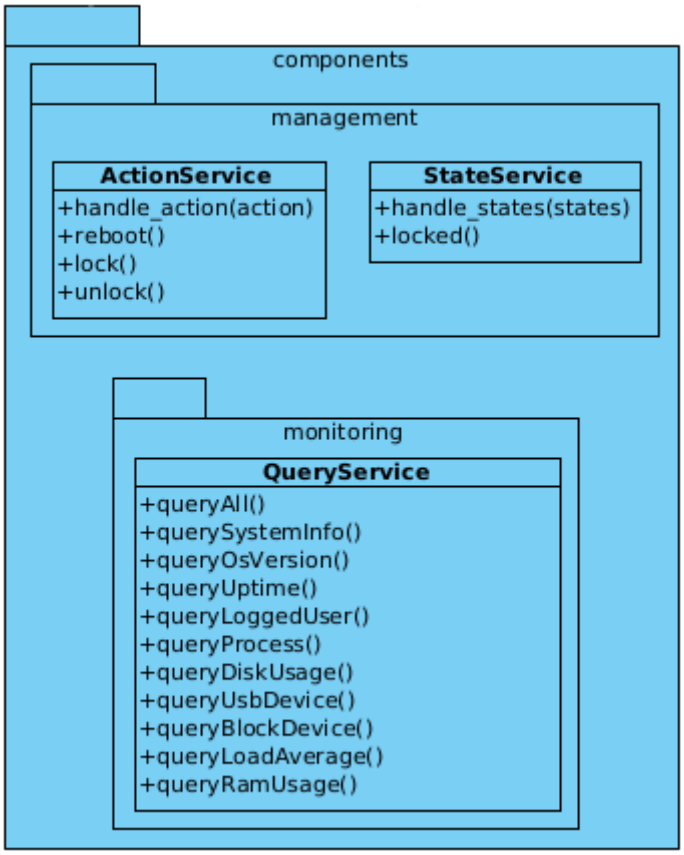

Ilustración 52. Diagrama de paquetes en el agente. Componentes

En la ilustración anterior mostramos el diagrama de paquetes que permite implementar las diferentes tareas de gestión y monitorización que cubre el agente. Dentro de este paquete, dividiremos cada componente en subpaquetes.

El subpaquete de gestión, *management*, se apoyará en otro paquete de la propia aplicación, *vendor*, para poder lanzar comandos de acción sobre el equipo.

El subpaquete de monitorización, *monitoring*, empleará la biblioteca Osquery para poder lanzar consultas de monitorización sobre el equipo.

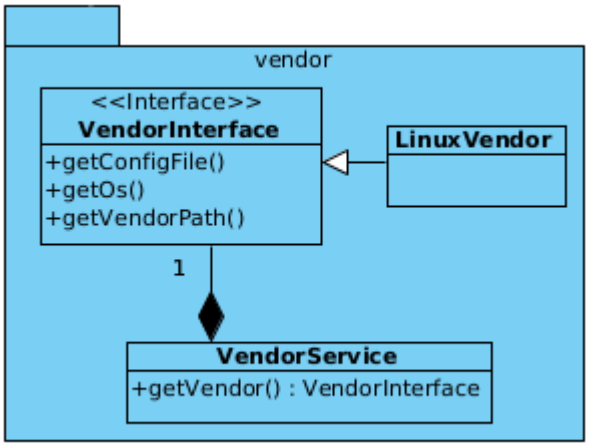

Ilustración 53. Diagrama de paquetes en el agente. *Vendor*

En la ilustración anterior mostramos el diagrama de paquetes para la implementación de una capa ligera de abstracción sobre el sistema operativo en el que se ejecuta el agente.

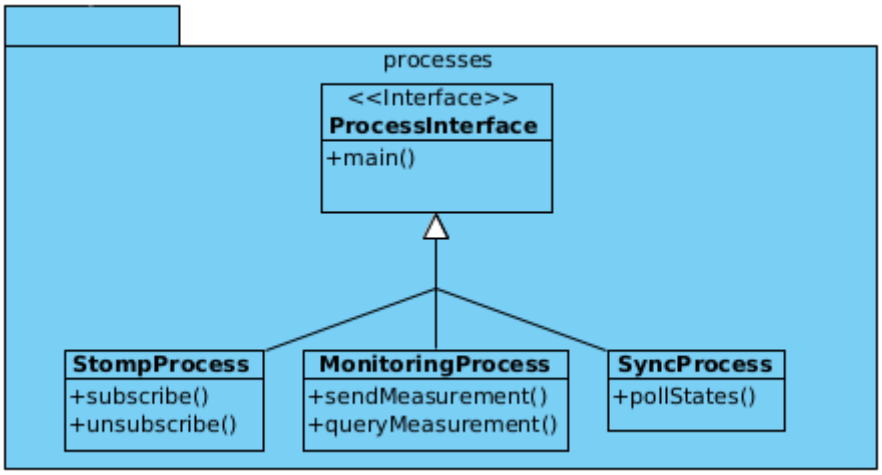

Ilustración 54. Diagrama de paquetes en el agente. Procesos

En la ilustración anterior mostramos el diagrama de paquetes con los subprocesos ejecutados por el agente. Todos los subprocesos en el diagrama interaccionan con alguna de las interfaces del servidor, que a su vez se apoyan sobre el paquete de componentes.

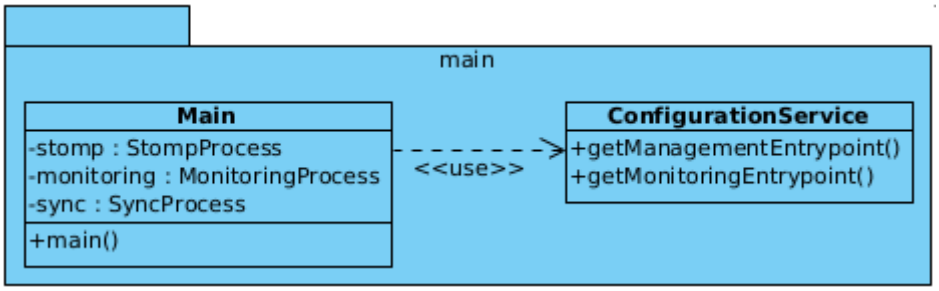

Ilustración 55. Diagrama de paquetes en el agente. Principal

En la ilustración anterior mostramos el diagrama de paquetes que sirve como punto de partida del agente. Aquí leemos la configuración del agente y gestionamos el ciclo de vida de los subprocesos.

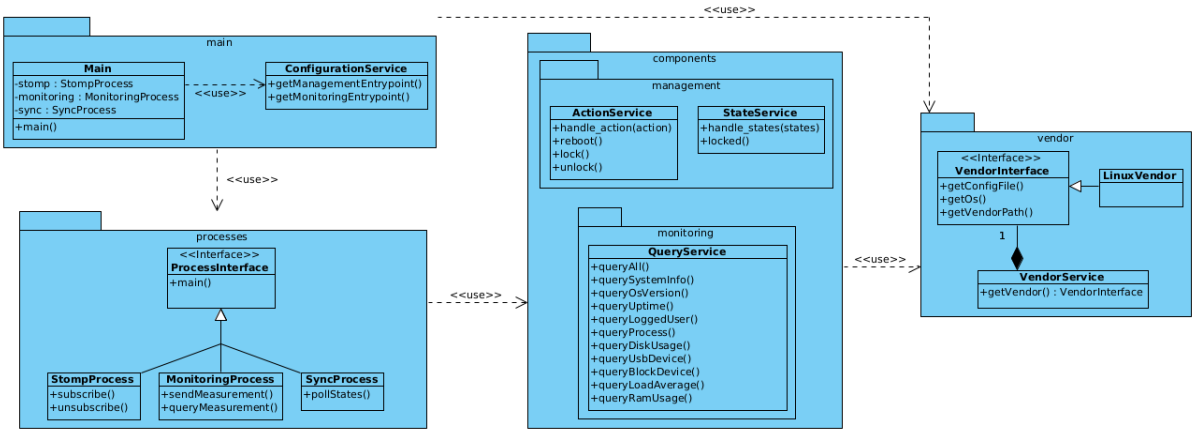

Ilustración 56. Diagrama de paquetes completo en el agente

Como última ilustración, mostramos el diagrama de paquetes completo del agente, donde observamos principalmente el uso que tiene el paquete de componentes por parte del paquete de procesos, tal y como ya veníamos reflejando desde el diagrama arquitectónico.

4.2. Detalles sobre la implementación

En este apartado analizaremos los detalles más relevantes sobre la implementación de los componentes del sistema desde un punto de vista técnico. Además, reflejaremos cómo hemos trabajado dentro del proyecto con los recursos que disponíamos, tanto software (*frameworks*, bibliotecas externas, etc) como hardware (portátiles, servidores, etc).

Antes de dar paso a los detalles individuales de cada componente, mencionaremos brevemente algunos de los aspectos comunes implicados.

Respecto de los recursos hardware, para lograr trabajar de manera cómoda y simultáneamente con los tres componentes (servidor, cliente web y agente), hemos empleado el modelo de portátil que planteamos durante el estudio de costes. Aunque en líneas generales ha ofrecido un rendimiento satisfactorio, un aspecto reseñable a destacar ha sido la demanda de memoria principal para virtualizar agentes; en caso de mejorar algún punto, sería éste.

Por otro lado, hemos ido intercalando el desarrollo del sistema entre un despliegue en local y otro en un servidor virtual contratado. De esta manera, las versiones resultantes de las iteraciones las hemos ido subiendo a dicho servidor como nuevas versiones para su revisión por parte del profesor.

Respecto de los recursos software, consideramos necesario el uso de Git como sistema de control de versiones del código. Aunque haya sido un proyecto trabajado en solitario, nos ha facilitado el correcto almacenamiento y posterior supervisión del código fuente. Por otro lado, como entorno de desarrollo principal hemos usado Apache NetBeans 13.

A continuación, nos extenderemos sobre los detalles de implementación referentes a cada componente: servidor, cliente web y agente.

4.2.1. Detalles de implementación sobre el servidor

En el servidor utilizamos Spring Framework como *framework* base. Lo podemos emplear sobre varios lenguajes de programación como Java, Groovy o Kotlin, así como también somos libres de instanciar el proyecto mediante diferentes gestores de dependencias como Maven o Gradle. Por familiaridad, hemos preferido realizar el desarrollo con Java bajo un proyecto en Maven.

Sin embargo, la configuración de Spring Framework por uno mismo es una labor tediosa y con ciertas dificultades para la correcta importación de su catálogo de bibliotecas sumado a la configuración de las mismas. De manera oficial, contamos

con la herramienta Spring Boot para la preparación del esqueleto del *framework* mediante la importación de unas pocas bibliotecas que actúan como raíz del resto.

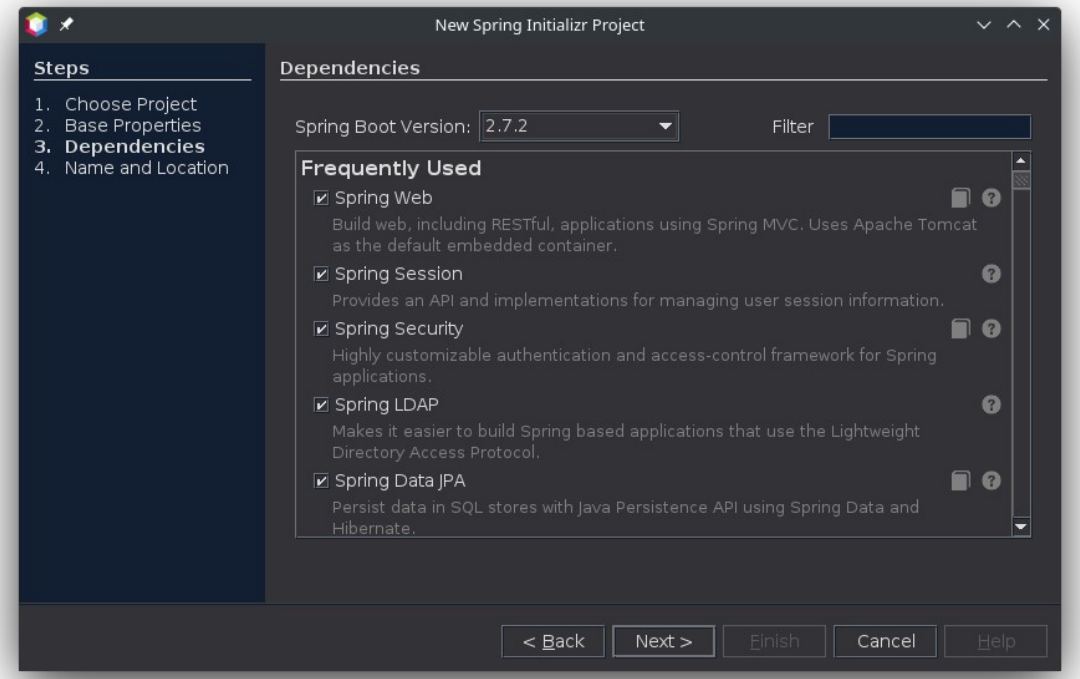

Ilustración 57. Spring Initializr en el entorno de desarrollo Apache NetBeans

Desde el entorno de desarrollo con NetBeans, podemos utilizar el complemento llamado Spring Initializr para la selección inicial de las bibliotecas de Spring Boot como reflejamos en la ilustración anterior. Algunos nombres ya los hemos mencionado en apartados pasados como Spring Session, Spring Security, Spring LDAP o Spring Data JPA.

En el momento de la redacción de este texto, nos encontramos en la versión 2.7.2 de Spring Boot. Sin embargo, nosotros empezamos a trabajar en el desarrollo del servidor usando la versión preliminar 2.7.0-M1. Al tratarse de un proyecto en Maven, podemos administrar las dependencias y contemplar futuras actualizaciones de versión desde el fichero pom.xml.

4.2.1.1. Perfiles de ejecución

Hemos agrupado los parámetros de ejecución del servidor mediante perfiles de ejecución. Dichos perfiles se encuentran preconfigurados para lanzarse sobre diferentes escenarios dependiendo del entorno en el que nos encontremos: producción, desarrollo o pruebas. Los ficheros de configuración donde podemos leer los parámetros de cada perfil son:

- Para producción: src/main/resources/application-production.yml
- Para desarrollo: src/main/resources/application-development.yml
- Para pruebas: src/test/resources/application-testing.yml

La ejecución del proyecto compilado inicia el perfil de producción. Este perfil exige conectividad con los servicios externos como bases de datos o el directorio activo.

La ejecución del proyecto desde NetBeans inicia el perfil de desarrollo. Este perfil arroja mayor cantidad de mensajes en el registro y genera documentación automáticamente para la interfaz de comunicación API REST. Además, dispone de servicios embebidos como bases de datos y de directorio activo, algunos de ellos siendo ejecutados a través de Docker, siendo un requisito durante el desarrollo.

La ejecución de las pruebas del proyecto inicia el perfil de pruebas. Este perfil es similar al de desarrollo, pero orientado a la ejecución de las pruebas unitarias mediante la biblioteca JUnit.

```yaml

# Profile: production

server:
    # Políticas de servidor
    port: 8080
    servlet:
        context-path: /v1
    error:
        include-stacktrace: never
    forward-headers-strategy: framework
    tomcat:
        remote-ip-header: X-Forwarded-For
        protocol-header: X-Forwarded-Proto
spring:
    # Metadatos de la aplicación
    application:
        name: @project.name@
        description: @project.description@
    build:
        version: @project.version@
        timestamp: @maven.build.timestamp@
logging:
    # Registro de mensajes (root para todos, web para Spring)
    level:
        root: INFO
springdoc:
    # Documentación
    show-actuator: false
    api-docs:
        enabled: false
```

Ilustración 58. Extracto del perfil de ejecución de producción en el servidor

En la ilustración anterior mostramos un fragmento de código del perfil de ejecución para producción. Algunos parámetros que pueden observarse son el puerto por defecto o el nivel de verbosidad del registro.

4.2.1.2. Interfaz de comunicación API REST

Hemos utilizado la biblioteca Spring Web para la implementación de una interfaz de comunicación API REST. Desde el punto de vista del *framework*, esto supone emplear una serie de anotaciones sobre las clases controladoras y sus métodos para especificar las características de cada *endpoint*. A continuación, explicaremos brevemente algunas de las anotaciones involucradas en la especificación de la interfaz.

```java
@Tag(name = "02. Devices", description = "Controlador de dispositivos")
@RestController
@RequestMapping("/devices")
@RequiredArgsConstructor
public class DeviceController {

    private final DeviceRepository deviceRepository;
    private final MapperComponent mapperComponent;
    private final AgentService agentService;
```

Ilustración 59. Extracto de la declaración de una clase controladora en el servidor

En la ilustración anterior ejemplificamos la declaración de una clase controladora basándonos en la clase *DeviceController*. Con la anotación *@RestController* indicamos el papel de la clase como controlador, mientras que con *@RequestMapping* detallamos la ruta base que seguirán sus métodos. Además, desde la anotación *@Tag* proporcionamos un título de sección para la documentación generada mediante la biblioteca Spring Doc. Por último, declaramos las dependencias del controlador como atributos de instancia para que sean inyectados por el *framework*.

```java
    @Operation(
        summary = "Obtiene un dispositivo dado su identificador",
        description = "")
    @SecurityRequirements({
        @SecurityRequirement(name = Realm.USERS_BASIC),
        @SecurityRequirement(name = Realm.USERS_X_AUTH_TOKEN),
    })
    @Parameters({
        @Parameter(name = "uid", description = "Identificador"),
    })
    @ApiResponses({
        @ApiResponse(
            responseCode = "200",
            description = "OK. Elemento encontrado"),
        @ApiResponse(
            responseCode = "404",
            description = "NOT_FOUND. Elemento no encontrado",
            content = @Content)})
    @GetMapping(value = "/{uid}",
                produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Device> findOne(
        @PathVariable
        String uid
    ) {
        return deviceRepository.findById(uid).map(
            found -> ResponseEntity.ok(found)
        ).orElse(
            ResponseEntity.notFound().build()
        );
    }
```

Ilustración 60. Extracto de un método en una clase controladora en el servidor

En la ilustración anterior ejemplificamos la implementación de un *endpoint* basándonos en el método *findOne* de la clase *DeviceController.* Con las anotaciones *@GetMapping* y *@PathVariable* fijamos las características del *endpoint*; así, mientras que con *@GetMapping* registramos el método de operación, la ruta definitiva y el formato del mensaje, con *@PathVariable* señalamos el parámetro de entrada ubicado dentro de la ruta –en este caso, *uid*–.

Sin embargo, la mayoría de las anotaciones mostradas no caracterizan al *endpoint*, sino que sólo proporcionan documentación para la biblioteca Spring Doc. Entre las anotaciones de documentación encontramos a *@Operation* para documentar el título del *endpoint*; *@SecurityRequirements*, los criterios de acceso; *@Parameters*, los parámetros de entrada, y *@ApiResponses*, las posibles respuestas resultantes.

Respecto de la implementación del método, podemos observar la comunicación entre el controlador y el repositorio con el fin de obtener un objeto perteneciente al modelo relacional dado su identificador. La respuesta queda envuelta en un objeto de clase *ResponseEntity* antes de ser enviada al cliente, variando el código de respuesta dependiendo del éxito al encontrarlo.

El resto de métodos siguen una especificación similar a la expuesta. Así, mediante otras anotaciones podemos registrar operaciones diferentes (como *@PostMapping*, *@PatchMapping* y *@DeleteMapping)* o leer parámetros de entrada desde otras fuentes (como *@QueryParam* y *@RequestBody)*.

Como resultado final, además de la propia interfaz de comunicación API REST, podemos encontrar la documentación generada por Spring Doc desde el perfil de desarrollo en la interfaz web ubicada en */docs.html*.

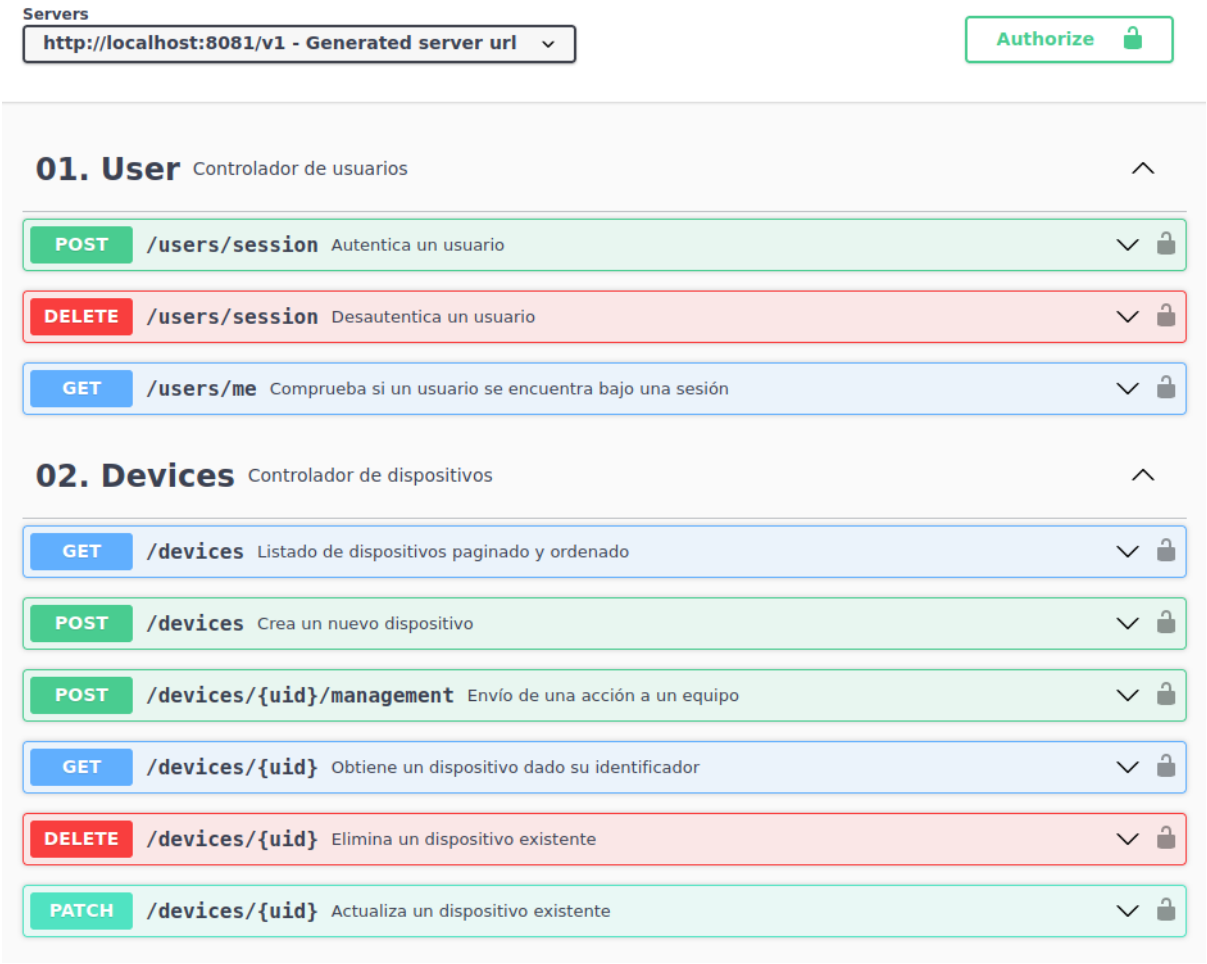

Ilustración 61. Documentación de Spring Doc para la interfaz de comunicación API REST

En la ilustración anterior mostramos la documentación generada por Spring Doc. Si expandimos alguna entrada, incluso podremos ejecutar el *endpoint*; aunque previamente, tendremos que autenticarnos mediante el botón ubicado en la esquina superior derecha llamado «*Authenticate*».

4.2.1.3. Interfaz de comunicación mediante colas

Un corredor de mensajes puede soportar múltiples protocolos de mensajería para las entidades productoras y consumidoras. De hecho, los más sofisticados son capaces de manejar múltiples protocolos de mensajería. Si nos fijamos en ActiveMQ, encontramos una amplia variedad de protocolos soportados como AMQP, MQTT, STOMP, WSIF e incluso REST [90]. En nuestro caso, hemos optado por el protocolo de mensajería STOMP (acrónimo del inglés, *Simple Text Oriented Messaging Protocol*) al ofrecernos una cómoda interoperabilidad entre entidades. Esta decisión no afecta al contenido de los mensajes en sí mismos, sino a las características

posteriormente disponibles. Por ejemplo, la especificación del protocolo STOMP contempla la suscripción de las entidades consumidoras a una cola [91].

Para integrarnos con algún corredor de mensajes compatible con mensajería STOMP (en inglés, *STOMP Broker*), hemos utilizado la biblioteca Spring Messaging. Dadas las necesidades actuales del proyecto, *SimpleBroker* –incluido en la propia biblioteca– es suficiente como corredor de mensajes en memoria [92].

Por otro lado, mediante la biblioteca Spring Websocket conectaremos las entidades consumidoras –los agentes– hacia un servidor de *websocket*. Websocket es un protocolo de comunicación bidireccional entre servidor y cliente que mantiene una conexión abierta para intercambiar mensajes hasta que alguno de los pares decida finalizar la sesión. Como tal, lo aplicaremos en la dirección servidor a agente para la entrega asíncrona de mensajes en la cola. Este enfoque es conocido como *STOMP over websocket* [93].

```
15  @Configuration
16  @EnableWebSocketMessageBroker
17  @RequiredArgsConstructor
18  public class WebSocketConfiguration implements WebSocketMessageBrokerConfigurer {
19
20      @Lazy
21      private final TaskScheduler taskScheduler;
22
23      @Override
24      public void registerStompEndpoints(StompEndpointRegistry registry) {
25          registry.addEndpoint("/agents/stomp").setAllowedOriginPatterns("*");
26      }
27
28      @Override
29      public void configureMessageBroker(MessageBrokerRegistry registry) {
30          registry.setPreservePublishOrder(true);
31          registry.enableSimpleBroker("/queue")
32              .setTaskScheduler(taskScheduler)
33              .setHeartbeatValue(this.heartbeatValue());
34      }
```

Ilustración 62. Extracto de la clase WebSocketConfiguration en el servidor

Spring Framework incluye mecanismos para la combinación de las bibliotecas Spring Messaging y Spring Websocket extendiendo la clase *WebSocketMessageBrokerConfigurer*. En la ilustración anterior observamos la configuración simultánea del servidor de *websocket* y del corredor de mensajes *SimpleBroker*.

Desde el método *registerStompEndpoints* indicamos el *endpoint* con el que una entidad consumidora podrá iniciar una conexión de *websocket*, en este caso, */agents/stomp*. Los mensajes intercambiados durante la sesión de *websocket* son los definidos dentro de la especificación del protocolo de mensajería STOMP. Dicho *endpoint* lo protegeremos con Spring Security, por lo que el agente deberá proporcionar su *token* de autenticación para acceder al sistema de colas.

Desde el método *configureMessageBroker* completamos los parámetros de conexión del corredor de mensajes. En este caso, al tratarse de un corredor en memoria, basta con habilitar algún contenedor de colas como */queue*. Dentro de este contenedor, podremos instanciar colas de usuario específicas por equipo cada vez que un agente se suscriba al corredor de mensajes.

La creación de las colas de usuario corre a cargo del *framework* mediante el componente *UserDestinationMessageHandler* [94]. Así, las entidades consumidoras no interactúan directamente con */queue*, sino con el alias */user/queue*. Cuando el servidor detecta el prefijo */user*, traduce dicho alias como la instancia de una cola de usuario. Por tanto, la traducción del alias */user/queue* se materializa en la cola de usuario */queue/user-uid*, donde *uid* es el identificador de usuario.

La interfaz de comunicación mediante colas que propusimos en el apartado de diseño establece una única cola en */user/queue/management* para la emisión de comandos de acción desde el servidor al agente. Considerando lo anterior, este alias se traduce en la instancia de una cola de usuario en */queue/management-user-uid*, donde el identificador de usuario responde al identificador de la entidad *Device* correspondiente al *token* de autenticación.

Respecto de cómo el servidor produce los mensajes que llegan a los agentes desde la interfaz de colas, nos tendremos que remitir a las clases *DeviceController* y *AgentService*.

```java
@PostMapping(value = "/{uid}/management",
             consumes = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Device> management(
    @Valid
    @RequestBody
    ManagementAction message,
    @PathVariable
    String uid
) {
    return deviceRepository.findById(uid)
    .map(found -> {
        agentService.sendManagementMessage(uid, message);
        return ResponseEntity.ok(found);
    }).orElse(ResponseEntity.notFound().build());
}
```

Ilustración 63. Extracto de la clase DeviceController en el servidor

En la clase controladora *DeviceController* encontramos un método que implementa la transmisión de un mensaje de acción *ManagementAction* desde la interfaz de comunicación API REST a la cola de mensajes. En concreto, el mensaje llega al servicio *AgentService* invocando al método *sendManagementMessage*.

```java
@Service
@RequiredArgsConstructor
@CommonsLog
public class AgentService {

    private final SimpMessagingTemplate simpMessagingTemplate;
    private final SimpUserRegistry simpUserRegistry;
    private final MessageChannel clientOutboundChannel;

    public void sendManagementMessage(String device, ManagementAction message) {
        simpMessagingTemplate.convertAndSendToUser(
            device,
            "/queue/management",
            message);
    }
```

Ilustración 64. Extracto de la clase AgentService en el servidor

Desde el servicio *AgentService* utilizamos el componente *SimpMessagingTemplate* del *framework* para enviar el mensaje anterior a través de la cola */queue/management* con el método *convertAndSendToUser*, que convenientemente realiza la traducción a la cola de usuario especificada en la variable *device*. En este punto, el mensaje ya se encuentra disponible en el corredor de mensajes que se encargará de su enrutamiento y entrega al agente suscrito correspondiente.

4.2.1.4. Autenticación para usuarios

Nuestro sistema no maneja usuarios por sí mismo, sino que delegamos su administración a un directorio activo LDAP externo. En este sentido, sólo debemos preocuparnos por la integración de la lectura de objetos de clase *InetOrgPerson* para validar las credenciales del usuario. Después de autenticarnos, generaremos un *token* de sesión que seguirá identificando al usuario posteriormente en el sistema. Para implementar todo lo anterior, hemos usado las bibliotecas Spring Security, Spring Session JDBC y Spring LDAP.

Configuraremos Spring Security proporcionando los objetos *AuthenticationManager* y *SecurityFilterChain* al *framework*. Con el objeto *AuthenticationManager* identificaremos a un usuario dentro del directorio activo dadas sus credenciales –aquí combinamos Spring Security con Spring LDAP–, mientras que con el objeto *SecurityFilterChain* indicaremos los criterios de seguridad sometidos sobre las peticiones recibidas –aquí combinamos Spring Security con Spring Session JDBC–.

```
15  @Configuration
16  public class LdapAuthenticationConfiguration {
17
18      @Bean
19      AuthenticationManager ldapAuthenticationManager(
20          final BaseLdapPathContextSource contextSource,
21          final LdapAuthoritiesPopulator authorities,
22          final LdapPasswordEncoder ldapPasswordEncoder,
23          @Value("${spring.ldap.user-base}")
24          final String userBase
25      ) {
26          LdapPasswordComparisonAuthenticationManagerFactory factory =
27              new LdapPasswordComparisonAuthenticationManagerFactory(
28                      contextSource, ldapPasswordEncoder);
29          factory.setUserSearchBase(userBase);
30          factory.setUserSearchFilter("(uid={0})");
31          factory.setPasswordAttribute("userPassword");
32          factory.setLdapAuthoritiesPopulator(authorities);
33          factory.setUserDetailsContextMapper(new InetOrgPersonContextMapper());
34          return factory.createAuthenticationManager();
35      }
```

Ilustración 65. Extracto de la clase LdapAuthenticationConfiguration en el servidor

En la ilustración anterior observamos la preparación del objeto *AuthenticationManager* dentro de la clase *LdapAuthenticationConfiguration*. Uno de los argumentos más relevantes es *userBase*, pues establece la rama de búsqueda de los objetos de clase *InetOrgPerson* dentro del directorio activo. Para identificar a

un usuario, tendremos en cuenta los campos *uid* como nombre de usuario y *userPassword* como contraseña.

```
111     @Bean
112     @Order(1)
113     public SecurityFilterChain userChain(
114         final HttpSecurity http,
115         @Qualifier("ldapAuthenticationManager")
116         final AuthenticationManager authenticationManager
117     ) throws Exception {
118         return http
119         .requestMatchers()
120             .anyRequest()
121         .and()
122         .authorizeHttpRequests((authorize) -> authorize
123             .anyRequest().authenticated()
124         )
125         .sessionManagement()
126             .sessionCreationPolicy(SessionCreationPolicy.NEVER)
127         .and()
128         .authenticationManager(authenticationManager)
129         .httpBasic()
130             .realmName("TFG USERS")
131         .and()
132         .build();
133     }
```

Ilustración 66. Extracto del objeto *SecurityFilterChain* para usuarios en el servidor

En la ilustración anterior observamos la preparación del objeto *SecurityFilterChain* para usuarios dentro de la clase *SecurityConfiguration*. Este filtro de seguridad queda aplicado sobre todos los *endpoints* de la interfaz de comunicación API REST e incorpora al objeto *AuthenticationManager* configurado previamente. Respecto de la gestión de sesiones, desactivamos la política por defecto de Spring Security, ya que queremos controlarlas nosotros programáticamente desde las clases *UserController* y *SessionService*.

```java
@Configuration
public class SessionConfiguration {

    @Bean
    public SessionRepositoryCustomizer<JdbcIndexedSessionRepository>
        sessionRepositoryCustomizer()
    {
        return (JdbcIndexedSessionRepository sessionRepository) -> {
            sessionRepository.setDefaultMaxInactiveInterval(
                (int) Duration.of(7, ChronoUnit.DAYS).getSeconds()
            );
        };
    }

    @Bean
    public HttpSessionIdResolver httpSessionIdResolver() {
        return HeaderHttpSessionIdResolver.xAuthToken();
    }

}
```

Ilustración 67. Configuración de Spring Data JDBC

En la ilustración anterior mostramos la configuración de Spring Session JDBC a través de la clase *SessionConfiguration*. En ella, establecemos el periodo máximo de inactividad de un *token* de sesión en 7 días antes de ser eliminado de la base de datos y fijamos la cabecera *X-Auth-Token* como parámetro de entrada para la autenticación.

```java
    @PostMapping(value = "/session",
                 produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Logged> login(
            HttpServletRequest httpServletRequest,
            @AuthenticationPrincipal
            InetOrgPerson inetOrgPerson
    ) {
        return ResponseEntity.ok(
            sessionService.login(httpServletRequest, inetOrgPerson));
    }
```

Ilustración 68. *Endpoint* de inicio de sesión en el servidor

En la ilustración anterior mostramos el *endpoint* responsable de generar un *token* de sesión. Previamente, el *framework* –mediante el objeto *SecurityFilterChain*– ha validado las credenciales del usuario recibidas bajo autenticación HTTP *Basic*. Si la autenticación resultó ser exitosa, dispondremos de una instancia *InetOrgPerson* como parámetro de entrada con la información del usuario. Finalmente, invocamos al método *login* del servicio *SessionService* para obtener el *token* de sesión.

```
17      public Logged login(
18          HttpServletRequest httpServletRequest,
19          InetOrgPerson inetOrgPerson
20      ) {
21          HttpSession httpSession = httpServletRequest.getSession(true);
22          return Logged.builder()
23              .username(inetOrgPerson.getUid())
24              .cn(String.join(" ", inetOrgPerson.getCn()))
25              .sn(inetOrgPerson.getSn())
26              .token(httpSession.getId())
27              .isNew(httpSession.isNew())
28              .build();
29      }
```

Ilustración 69. Método login de SessionService en el servidor

En la ilustración anterior mostramos la implementación del método *login* de la clase *SessionService* para generar el *token* de sesión donde, además, construiremos un objeto *Logged* con la información recuperada de la instancia *InetOrgPerson* como el nombre común y los apellidos.

4.2.1.5. Autenticación para agentes

Los equipos informáticos –mediante la entidad *Device*– gestionan los *tokens* de autenticación –mediante la entidad *PushToken*– que emplean los agentes para identificarse en el sistema. Así, todas las interacciones del agente son vinculantes con algún equipo a través del *token*. Para implementar la autenticación de los agentes, hemos empleado las bibliotecas Spring Security y Spring Data JPA.

De manera análoga a la vista durante la autenticación de usuarios, configuraremos Spring Security proporcionando los objetos *AuthenticationManager* y *SecurityFilterChain* al *framework*. Sin embargo, en esta ocasión *AuthenticationManager* necesita que le proporcionemos un objeto *AuthenticationProvider* para completar los pasos de la autenticación. Ya que vamos a recuperar objetos del modelo de datos, lo conveniente es optar por la clase *DaoAuthenticationProvider* destinada a este propósito e incluida en el propio *framework* [95]. Esta última nos exige la implementación de una clase *UserDetailsService* y una clase *UserDetails*.

```java
@Getter
public class DevicePrincipal extends User implements Principal {

    public DevicePrincipal(Device device, String token) {
        super(device.getUid(), token, Collections.emptyList());
    }

    @Override
    public String getName() {
        return this.getUsername();
    }

}
```

Ilustración 70. Clase *UserDetails* para los *tokens* de autenticación del agente

En la ilustración anterior mostramos la clase *DevicePrincipal* que hereda de *UserDetails* desde la clase *User*. En ella, recogemos los parámetros que deseamos almacenar durante la autenticación en una futura instancia. Dicha instancia estará disponible para las interfaces de comunicación API REST y de colas.

```java
@Service
@RequiredArgsConstructor
public class PushAgentDetailsService implements UserDetailsService  {

    private final DeviceRepository deviceRepository;

    @Override
    public UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException
    {
        return deviceRepository.findByToken(username).map(device ->
            new DevicePrincipal(device, username)
        ).orElseThrow(() ->
            new UsernameNotFoundException("Not found " + username));
    }

}
```

Ilustración 71. Clase *PushAgentDetailsService* en el servidor

En la ilustración anterior mostramos la clase *PushAgentDetailsService* que implementa la interfaz *UserDetailsService*. Con el método *loadUserByUsername* realizamos la búsqueda de una entidad *Device* que contenga el *token* de autenticación en cuestión mediante el repositorio *DeviceRepository*. En caso de éxito, construimos una instancia *DevicePrincipal*.

```java
@Configuration
public class PushAgentAuthenticationConfiguration {

    @Bean
    public AuthenticationProvider pushAgentAuthenticationProvider(
        final PushAgentDetailsService pushAgentDetailsService
    ) {
        DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();
        daoAuthenticationProvider.setUserDetailsService(pushAgentDetailsService);
        daoAuthenticationProvider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());
        return daoAuthenticationProvider;
    }

    @Bean
    public AuthenticationManager pushAgentAuthenticationManager(
        @Qualifier("pushAgentAuthenticationProvider")
        final AuthenticationProvider pushAgentAuthenticationProvider
    ) {
        return new ProviderManager(pushAgentAuthenticationProvider);
    }

}
```

Ilustración 72. Clase *PushAgentAuthenticationConfiguration* en el servidor

En la ilustración anterior mostramos la clase *PushAgentAuthenticationConfiguration* donde configuramos los objetos *AuthenticationProvider* y *AuthenticationManager* necesarios por el *framework* para continuar con la configuración de seguridad. En primer lugar, preparamos un objeto *DaoAuthenticationProvider* al que le proporcionamos el servicio anterior *PushAgentDetailsService.* Por otro lado, registramos el *AuthenticationProvider* recién construido dentro del *AuthenticationManager*.

```java
    @Bean
    @Order(0)
    public SecurityFilterChain deviceChain(
        final HttpSecurity http,
        @Qualifier("pushAgentAuthenticationManager")
        final AuthenticationManager authenticationManager
    ) throws Exception {
        return http
        .requestMatchers()
            .antMatchers("/agents/push/**",
                         "/agents/stomp/**")
        .and()
        .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        .authorizeHttpRequests((authorize) -> authorize
            .anyRequest().authenticated()
        )
        .authenticationManager(authenticationManager)
        .apply(xAuthAgentConfigurer())
        .and()
        .httpBasic()
            .realmName("TFG PUSH AGENT")
        .and()
        .build();
    }
```

Ilustración 73. Extracto del objeto *SecurityFilterChain* para agentes en el servidor

En la ilustración anterior observamos la preparación del objeto *SecurityFilterChain* para agentes dentro de la clase *SecurityConfiguration*. Este filtro de seguridad queda aplicado para los *endpoints* de la clase *PushAgentController* y los *endpoints* que inician la conexión de *websocket* para la interfaz de colas. Además, establecemos la cabecera *X-Auth-Token* como parámetro de entrada para la autenticación con el *token* mediante el método *xAuthAgentConfigurer* –no cubriremos su implementación–.

4.2.2. Detalles de implementación sobre el cliente web

En el cliente web utilizamos Angular como *framework* principal. Éste emplea el lenguaje de programación Typescript e involucra tecnologías adicionales como HTML para la maquetación de los componentes de la vista y CSS para la construcción de estilos.

Para trabajar con Angular, necesitamos contar previamente con el motor de ejecución Node.js junto con algún gestor de dependencias como npm o yarn. Respecto del entorno de desarrollo, desde NetBeans no contamos con ningún asistente gráfico para construir el esquema inicial del proyecto, así que recurriremos a la herramienta de consola angular-cli.

En cualquier caso, angular-cli es una herramienta fundamental que conviene conocer. Además de instanciar un nuevo proyecto, nos facilita la gestión de bibliotecas permitiéndonos añadir nuevas o actualizar las existentes. También nos proporciona comandos útiles durante el desarrollo, como la generación de plantillas para nuevos componentes y servicios o la ejecución del proyecto en modo desarrollador.

Finalmente, el cliente web se encuentra instanciado sobre Angular 13.3, Node.js 12.22 y npm 7.5. Como bibliotecas secundarias, cabe mencionar la presencia de Angular Routing para la navegación entre vistas y de Angular Material con su repertorio de temas, componentes, iconos y fuentes de letra disponibles que nos ayudan a lograr una experiencia de usuario consistente.

4.2.2.1. Navegación entre vistas

Una de las características más destacadas de Angular es su modelo de navegación bajo el concepto *Single-Page-Application* introducido en la biblioteca Angular Routing [96]. Este concepto considera la navegación en la aplicación web desde una única página, sin volver a realizar una nueva petición al servidor para cambiar de página. Según la vista en la que nos encontremos, el contenido de la página se actualiza mostrando u ocultando elementos convenientemente.

Angular Routing interpreta las rutas del navegador como vistas, y no como páginas, transparentemente para nosotros. A grandes rasgos, tan sólo tenemos que especificar los componentes asociados a cada vista desde el fichero *app-routing.module.ts* junto con la directiva *router-outlet* en los componentes que las renderizan.

```
20  const routes: Routes = [
21    { path: 'login', component: LoginComponent, canActivate: [LoginGuard] },
22    { path: '', component: HomeComponent, canActivate: [AuthGuard], children: [
23      { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
24      { path: 'dashboard', component: DashboardComponent },
25      { path: 'devices', children: [
26        { path: '', component: DeviceListComponent },
27        { path: 'new', component: DeviceNewComponent },
28        { path: ':uid/remove', component: DeviceRemComponent },
29        { path: ':uid/edit', component: DeviceModComponent },
30      ]},
31      { path: 'places', children: [
32        { path: '', component: PlaceListComponent },
33        { path: 'new', component: PlaceNewComponent },
34        { path: ':uid/remove', component: PlaceRemComponent },
35        { path: ':uid/edit', component: PlaceModComponent },
36      ]},
37    ]},
38    { path: '**', component: NotFoundComponent }
39  ];
```

Ilustración 74. Extracto de la clase *AppRoutingModule* en el cliente web

En la ilustración anterior mostramos la configuración de las rutas de navegación del cliente web. Las más relevantes son *LoginComponent* y *HomeComponent*. En *LoginComponent* cargamos la vista de acceso que se encuentra protegida por el objeto de guarda *LoginGuard*, mientras que *HomeComponent* lo protege *AuthGuard*. Por otro lado, *HomeComponent* no es en sí una vista completa, sino que se apoya en componentes hijos para construir una vista final a través de la directiva *router-outlet*.

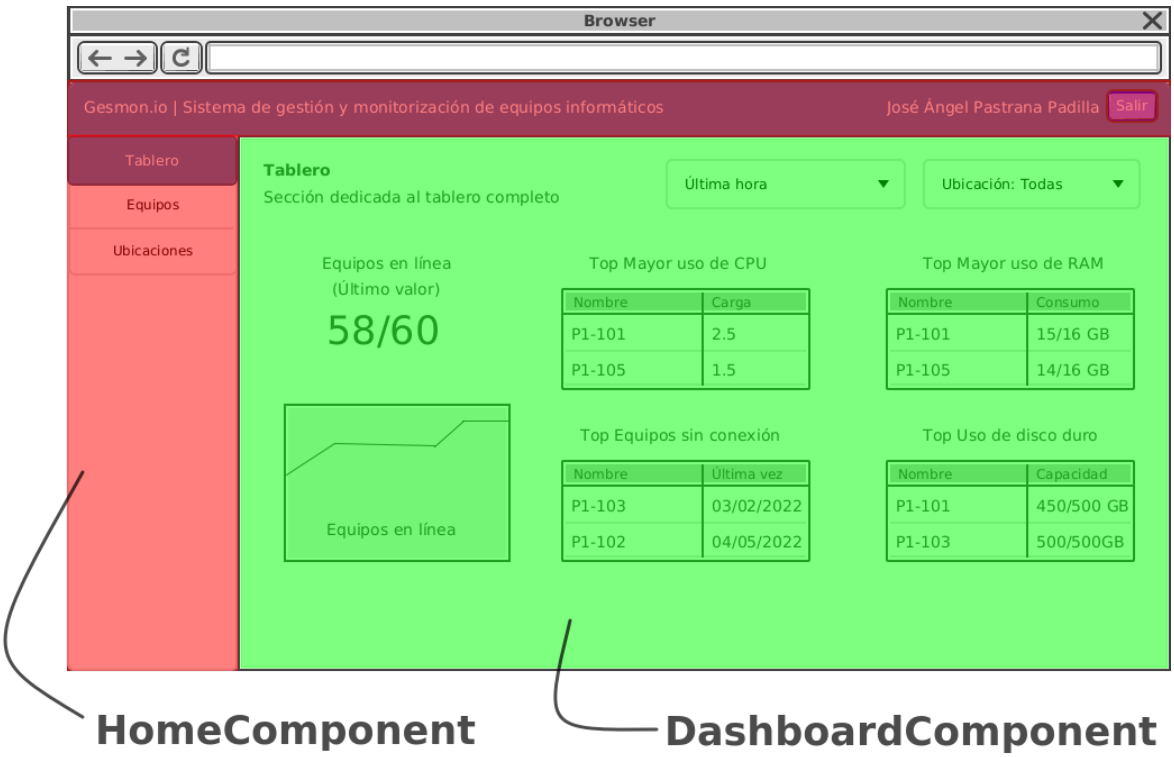

Ilustración 75. Interfaz de usuario del cliente web resaltando los componentes

Así, por ejemplo, al acceder a */dashboard* sumamos los componentes *HomeComponent* –sombreado en color rojo– y *DashboardComponent* –sombreado en color verde–. En los cambios de vista entre tablero, equipos y ubicaciones, *HomeComponent* sigue renderizándose y únicamente cambia el área principal –sombreada en color verde– por algún otro componente hijo. Este mecanismo de renderización puede anidarse múltiples veces, aunque no es nuestro caso.

```
<div class="main-container">
  <mat-toolbar ...30 lines />
  <mat-sidenav-container class="sidenav-container">
    <mat-sidenav ...28 lines />
    <mat-sidenav-content>
      <router-outlet></router-outlet>
    </mat-sidenav-content>
  </mat-sidenav-container>
</div>
```

Ilustración 76. Extracto de home.component.html en el cliente web

En la ilustración anterior podemos observar la maquetación en HTML del componente *HomeComponent*. Como explicamos al principio, los componentes hijos se renderizan en sustitución de la directiva *router-outlet*.

4.2.2.2. Autenticación y acceso

En el cliente web, hemos conjugado un servicio propio de comunicación con el servidor para autenticar los usuarios del sistema junto con los objetos de guarda del *framework* para limitar el acceso a las vistas.

Desde el servicio *AuthService*, implementamos métodos de autenticación para obtener el *token* de sesión de un usuario dadas sus credenciales, así como el almacenamiento de las respuestas recibidas del servidor en la memoria local del navegador para un posterior uso.

Desde los objetos de guarda, velamos que no se produzca el acceso a una vista sin el nivel de autenticación correspondiente. Tenemos dos objetos de guarda, *LoginGuard* para permitir la entrada a la vista de inicio de sesión sólo a los usuarios no autenticados y *AuthGuard* para permitir la entrada a las demás vistas sólo a los usuarios autenticados. Ambos objetos se apoyan en el servicio *AuthService* para conocer el estado de autenticación de la aplicación.

```
36   login(username: string, password: string): Observable<User> {
37     return new Observable(subscriber => {
38       const httpOptions = {
39         headers: new HttpHeaders({
40           'Authorization': `Basic ${btoa(username + ':' + password)}`
41         })
42       };
43       this.httpClient.post<User>(
44         this.appService.getEndpoint('/users/session'),
45         null,
46         httpOptions).subscribe(
47           success => {
48             this.add(success).subscribe(
49               success => subscriber.next(success),
50               error => subscriber.error(error),
51               () => subscriber.complete()
52             );
53           },
54           error => subscriber.error(error)
55         );
56     });
57   }
58
59   add(user: User): Observable<User> {
60     return new Observable(subscriber => {
61       try {
62         localStorage.setItem('user', JSON.stringify(user));
63         this._authenticated$.next(user);
64         subscriber.next(user);
65         subscriber.complete();
66       } catch (error) {
67         subscriber.error(error);
68       }
69     });
70   }
71
72   get headers(): HttpHeaders {
73     return new HttpHeaders({
74       'X-Auth-Token': this._authenticated$.getValue()!.token
75     });
76   }
```

Ilustración 77. Extracto de auth.service.ts en el cliente web

En la ilustración anterior observamos los métodos *login*, *add* y *headers* del servicio *AuthService*. En el método *login* enviamos las credenciales del usuario al servidor para obtener el *token* de sesión. En caso de éxito, dicho *token* se almacenará en la memoria local del navegador a través del método *add*. Posteriormente, otros servicios del cliente web pueden recuperar el *token* de sesión actual desde el método *headers* para realizar peticiones al servidor.

```
10  export class AuthGuard implements CanActivate {
11
12    constructor(private authService: AuthService,
13                private router: Router) {}
14
15    canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot) {
18
19      return this.authService.isAuthenticated()
20      .pipe(tap((value: boolean) => this.redirectIfNeeded(value)));
21    }
22
23    redirectIfNeeded(authenticated: boolean) {
24      if (!authenticated) {
25        this.router.navigate(['/login']);
26      }
27    }
28
29  }
```

Ilustración 78. Extracto de auth.guard.ts en el cliente web

En la ilustración anterior mostramos la implementación del objeto de guarda *AuthGuard*, donde inyectamos la dependencia al servicio *AuthService* que empleamos posteriormente desde el método *canActivate*, proveniente de la interfaz *CanActivate* del *framework*. Dentro de dicho método, comprobamos el estado de autenticación del cliente web. En el caso de *AuthGuard*, si no nos encontramos autenticados, redirigimos al usuario a la vista de inicio de sesión.

Los objetos de guarda se encuentran aplicados sobre las rutas de Angular Routing que vimos en el capítulo de navegación entre vistas.

4.2.2.3. Renderización de gráficas de monitorización

Hemos empleado la biblioteca Plotly.js para la renderización de las gráficas de monitorización. Mediante su versión específica para Angular, angular-plotly.js, podemos manejar la maquetación de las gráficas mediante la etiqueta *<plotly-plot>* y vincular los datos a visualizar con los atributos *data*, *config* y *layout* del mismo.

Los componentes del proyecto que emplean Plotly.js son *DeviceShowComponent* y *DashboardComponent*. En primer lugar, dichos componentes realizan una consulta al servicio *MeasurementService* para recuperar las métricas de monitorización en tablas formateadas en un documento *JSON* desde

la interfaz de comunicación API REST –a través del controlador *MeasurementController*–. Después de obtenerlas, los componentes sólo necesitan sustituir apropiadamente el contenedor donde se alojan los atributos *data*, *config* y *layout* para actualizar los datos previamente existentes.

```
<div class="tab-historic">
  <div class="plot">
    <plotly-plot [data]="historicTab?.uptime?.data"
                 [layout]="historicTab?.uptime?.layout"
                 [config]="historicTab?.uptime?.config">
    </plotly-plot>
  </div>
  <div class="plot">
    <plotly-plot [data]="historicTab?.cpu?.data"
                 [layout]="historicTab?.cpu?.layout"
                 [config]="historicTab?.cpu?.config">
    </plotly-plot>
  </div>
  <div class="plot">
    <plotly-plot [data]="historicTab?.ram?.data"
                 [layout]="historicTab?.ram?.layout"
                 [config]="historicTab?.ram?.config">
    </plotly-plot>
  </div>
  <div class="plot">
    <plotly-plot [data]="historicTab?.disk?.data"
                 [layout]="historicTab?.disk?.layout"
                 [config]="historicTab?.disk?.config">
    </plotly-plot>
  </div>
  <div class="plot">
    <plotly-plot [data]="historicTab?.session?.data"
                 [layout]="historicTab?.session?.layout"
                 [config]="historicTab?.session?.config">
    </plotly-plot>
  </div>
</div>
```

Ilustración 79. Extracto de device-show.component.html en el cliente web

En la ilustración anterior observamos la maquetación en HTML del componente *DeviceShowCompontent*. La variable de instancia *historicTab* está siendo contenedora de las métricas de monitorización para las gráficas de actividad del agente, carga de CPU, utilización de memoria RAM, capacidad de disco duro y número de sesiones abiertas del usuario.

El atributo *data* determina el modelo de gráfica a representar (en barras, en líneas, en área, circulares, etc) e inyecta los datos de las métricas en crudo. Aunque su formato depende de la gráfica representada, en la mayoría de ocasiones se trata

de un vector con objetos n-dimensionales. En nuestro caso, el eje de abscisas refleja la fecha y el eje de ordenadas el valor a representar.

El atributo *layout* establece la disposición de los elementos en la propia gráfica como la posición del título o de la leyenda, el rango mínimo o máximo, los prefijos o sufijos, etc.

El atributo *config* aplica configuraciones sobre las capacidades de visualización de la gráfica como zoom, barras de control, logos, etc.

```
264       cpu: {
265         data: [{
266           x: historic?.cpu?.x?.map((x: any) => {
267             return new Date(x);
268           }),
269           y: historic?.cpu?.y,
270           fill: 'tonexty',
271           name: 'Hilos usados',
272           mode: 'lines+markers',
273           connectgaps: false,
274         }],
275         layout: {
276           title: 'Evolución de CPU',
277           showlegend: true,
278           font: {size: 16},
279           yaxis: {
280             fixedrange: true,
281             ticksuffix: 'th',
282           },
283           autosize: true,
284           width: 0,
285           height: 450,
286           legend: {
287             orientation: 'h',
288             xanchor: 'center',
289             x: 0.5,
290             y: 1.15,
291           },
292         },
293         config: {
294           displayModeBar: false,
295           displaylogo: false,
296           responsive: true,
297           scrollZoom: true,
298         }
299       },
```

Ilustración 80. Extracto de device-show.component.ts en el cliente web

En la ilustración anterior observamos la instanciación de los atributos *data*, *layout* y *config* en *historicTab.cpu* para la renderización de la gráfica de carga de CPU.

4.2.3. Detalles de implementación sobre el agente

En el agente utilizamos el lenguaje de programación Python para implementar los comportamientos esperados de gestión y monitorización sobre los equipos. Sin embargo, aunque Python sea un lenguaje capaz de ejecutarse sobre múltiples plataformas, debemos advertir que nos hemos enfocado en el sistema operativo Debian –distribución basada en GNU/Linux– como plataforma principal.

Aunque no hemos incorporado ningún *framework* –ya que el propósito del agente resulta bastante específico–, sí hemos contado con diferentes bibliotecas externas para apoyarnos en el desarrollo. Algunas son bibliotecas de Python como *requests* –para realizar peticiones a la interfaz de comunicación API REST– o *stomp-ws* –para suscribirnos a la interfaz de comunicación mediante colas–. En cambio, otras son dependientes del sistema operativo como Osquery –para la recogida de métricas de monitorización– o SDL –para mostrar mensajes por pantalla en los equipos–.

En el flujo principal de la aplicación hemos preparado tres subprocesos que permanecerán en ejecución hasta el fin del programa. Cada subproceso responde a una tarea: *StompProcess* se suscribe a la cola de mensajes para recibir mensajes del servidor, *MonitoringProcess* recoge las métricas de monitorización y *SyncProcess* es un comprobador de salud entre el agente y el servidor.

```
# Leemos la configuración de la aplicación.
config = Configuration(path = vendor.get_config_file())

# Definimos los procesos del agente. Dedicaremos:
# - Proceso de escucha al protocolo de mensajería STOMP.
# - Proceso de envío de métricas al servidor.
# - Proceso de salud con el servidor.
processes = [
  Process(
    daemon = True,
    target = StompProcess(config, pShutdown).run,
  ),
  Process(
    daemon = True,
    target = MonitoringProcess(config, pShutdown).run,
  ),
  Process(
    daemon = True,
    target = SyncProcess(config, pShutdown).run,
  ),
]

# Iniciamos todos los subprocesos.
for process in processes: process.start()

# Esperamos una interrupción.
tShutdown.wait()

# Señalizamos el fin de la aplicación a los subprocesos.
pShutdown.set()

# Esperamos a la finalización de todos los subprocesos.
for process in processes: process.join()
```

Ilustración 81. Extracto de main.py en el agente

En la ilustración anterior observamos las acciones del agente en el arranque, en donde preparamos los tres subprocesos mencionados anteriormente.

4.2.3.1. Recepción de acciones

Los mensajes de acción surgen de un usuario que desea manipular el equipo vinculado al agente con algún comando soportado (por ejemplo, un reinicio). El servidor puede entregar inmediatamente dicho mensaje al agente a través de la interfaz de comunicación mediante colas. Dentro del agente, el subproceso encargado de la suscripción a la cola es StompProcess.

```python
def main(self):
  while not self.shutdown.is_set():
    try:
      logging.warn('Creating new connection to the STOMP broker', exc_info = False)
      client = stomp_client(self.__config['management']['entrypoint'], header = {
        'X-Auth-Agent': self.__config['connection']['token']
      })
      self.on_shutdown = lambda: client.disconnect()
      client.connect()
      client.subscribe("/user/queue/management", callback = self.handle_frame)
      logging.warn('Successful connected to the STOMP broker', exc_info = False)
      while not self.shutdown.is_set():
        logging.debug('Heartbeat every 10 seconds', exc_info = False)
        client.ws.send('\n')
        self.shutdown.wait(ten_seconds := 10)
    except:
      logging.warn('Unable to connect to the STOMP broker', exc_info = True)
    finally:
      self.on_shutdown = lambda: None
      self.shutdown.wait(one_second := 1)
```

Ilustración 82. Extracto de la clase *StompProcess* en el agente

En la ilustración anterior observamos el bucle principal del subproceso *StompProcess*. Desde el servicio de configuración, conseguimos el *endpoint* al que debe suscribirse el agente y el *token* de autenticación. Posteriormente, instanciamos una entidad consumidora llamada *client* que se suscribe a los mensajes de la cola */user/queue/management*. En último lugar, para mantener la suscripción activa, enviamos señales de actividad periódicas al corredor de mensajes.

Respecto a los mensajes recibidos en la cola, éstos son atendidos mediante la función de retorno *handle_frame*, que lee el contenido del mensaje y lo entrega al servicio *ActionService*.

```python
def handle_action(self, body):
  {
    'lock': self.lock_action,
    'unlock': self.unlock_action,
    'reboot': self.reboot_action,
  }.get(body['action'], self.default_action)(body)

def reboot_action(self, body):
  logging.warning('Reboot management action applied')
  os.system(Path(self.__vendor.get_vendor_path()) / 'reboot' / 'now')
```

Ilustración 83. Extracto de la clase *ActionService* en el agente

En la ilustración anterior observamos los métodos *handle_action* y *reboot_action* del servicio *ActionService*. Desde *handle_action* clasificamos el mensaje de acción entrante e invocamos al método correspondiente. Considerando que la acción sea un reinicio, quedaría implementado desde el método *reboot_action*

en colaboración con el servicio *VendorService* para obtener la ruta externa del binario que implementa la acción.

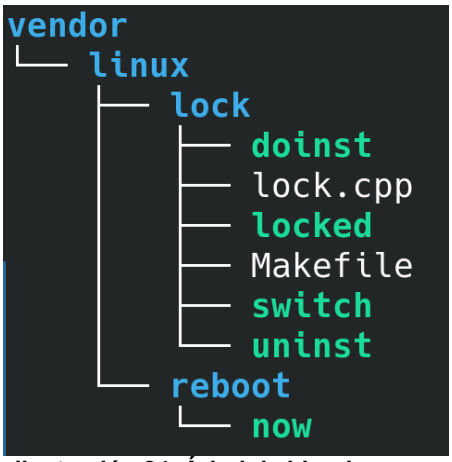

Ilustración 84. Árbol de binarios para la plataforma GNU/Linux en el agente

En la ilustración anterior recogemos el contenido de la carpeta *vendor* con los binarios de los comandos soportados por el agente. Actualmente, sólo encontraremos binarios para la plataforma GNU/Linux. Llegado el momento de añadir nuevas plataformas o nuevos comandos, ampliaríamos el contenido de esta carpeta donde el primer nivel representa a la plataforma y el segundo nivel al comando.

4.2.3.2. Recogida de métricas

La recogida de metricas de monitorizacion es una tarea que realiza el agente cada minuto autónomamente sobre el equipo en el que se encuentra instalado mediante el subproceso *MonitoringProcess.*

```
def main(self):
  while not self.shutdown.is_set():
    try:
      logging.warn('Sending metrics to the server', exc_info = False)
      requests.post(self.__config['monitoring']['entrypoint'] + '/monitoring', headers = {
        'X-Auth-Agent': self.__config['connection']['token']
      }, json = self.__queries.query_all(), timeout = (ten_seconds := 10))
    except:
      logging.warn('Unable to send metrics to the server', exc_info = True)
    self.shutdown.wait(one_minute := 60)
```

Ilustración 85. Extracto de la clase MonitoringProcess en el agente

En la ilustración anterior observamos el bucle principal del subproceso *MonitoringProcess*. Desde el servicio de configuración, conseguimos el *endpoint*

para el registro de las métricas en el servidor a través de la interfaz de comunicación API REST y el *token* de autenticación. Posteriormente, invocamos al método *query*_all de *QueryService* para recopilar y empaquetar las métricas que le serán enviadas al servidor. Después de enviarlas, el agente repetirá el proceso después de un minuto, salvo que llegue una interrupción que detenga al agente.

Dentro del servicio *QueryService* utilizamos el *framework* Osquery para inspeccionar las métricas del equipo. Dicho *framework* nos ofrece una vista de las métricas como una base de datos en memoria inspeccionable utilizando sentencias SQL.

```
  def __init__(self):
    self.instance = osquery.SpawnInstance()
    self.instance.open()

  def query_all(self):
    return [
       *self.query_mac_address(),
       *self.query_inet_address(),
       *self.query_system_info(),
       *self.query_os_version(),
       *self.query_uptime(),
       *self.query_logged_user(),
       *self.query_process(),
       *self.query_disk_usage(),
       *self.query_usb_device(),
       *self.query_block_device(),
       *self.query_load_average(),
       *self.query_ram_usage(),
    ]

  def query_logged_user(self):
    query = """
SELECT
  user,
  host,
  time as logged_at
FROM logged_in_users
WHERE type = 'user' and tty != '';
"""
    return self.wrap_up('logged_user', self.instance.client.query(query).response)
```

Ilustración 86. Extracto de la clase AgentService en el agente

En la ilustración anterior presentamos la clase QueryService acompañada por su constructor, el método *query_all* y el método *query_logged_user*. En el constructor, abrimos una conexión con la base de datos de Osquery que posteriormente emplearemos para realizar consultas. Un ejemplo práctico de ello lo tenemos bajo el método *query_logged_user*, donde ejecutamos una sentencia SQL

para recuperar las sesiones de usuario activas en el equipo. Finalmente, desde el método *query_all* reunimos en un vector todas las consultas implementadas en el servicio.

5. PRUEBAS

En este apartado ejecutaremos el plan de pruebas definido en el apartado de diseño para validar el correcto funcionamiento del sistema y analizaremos los resultados obtenidos.

5.1. Resultados sobre los criterios de aceptación

Nos basaremos en el desempeño de un operador humano para la ejecución y validación de los criterios de aceptación definidos en las historias de usuario.

Para ejecutar los criterios de aceptación, se ha preparado un entorno de preproducción con todos los servicios implicados desplegados mediante tecnología de contenedores en una estación de trabajo independiente. Además, los proyectos del cliente web y del servidor han sido compilados y ejecutados en modo producción para la ocasión.

Respecto a la incorporación de agentes en el sistema en preproducción, hemos empleado varias máquinas virtuales con una instalación limpia de Debian que simulan ser equipos reales junto con el portátil empleado durante el desarrollo.

Así, hemos ejecutado un total de 35 criterios de aceptación repartidos en 17 historias de usuario. A modo de ejemplo, comentaremos tres problemas en el sistema detectados gracias a ellos.

En el paso de desarrollo a producción, las credenciales erróneas provocaban la aparición de un *pop-up* nativo del navegador que preguntaba nuevamente por las credenciales, en vez de manejarse apropiadamente el fallo desde el cliente web.

Durante la modificación de un equipo informático, los cambios sobre algunos *tokens* de autenticación provocaban la eliminación en base de datos de los que no estaban siendo actualizados.

Desde la consulta del histórico de uso de un equipo informático, en ocasiones el primer o el último valor del intervalo temporal figuraba incompleto para precisiones altas.

En la finalización del trabajo, todas las pruebas han resultado exitosas y no han sido detectadas anomalías que comprometan los criterios de aceptación. En la siguiente tabla, presentamos los criterios de aceptación concernientes a las iteraciones realizadas en el proyecto.

ID	Título	Criterio de aceptación
1	Iniciar sesión	• Con unas credenciales válidas, produce la identificación del usuario en el sistema y lo redirige a la vista principal • Con unas credenciales inválidas, el sistema notifica al usuario que sus credenciales son erróneas
2	Cerrar sesión	• Después de cerrar la sesión, debe producirse una redirección a la vista de inicio de sesión
3	Visualizar ubicaciones	• Se encuentran todas las ubicaciones almacenadas en la base de datos
4	Crear una ubicación	• La ubicación debe quedar almacenada en la base de datos • La ubicación debe ser recuperable después de su creación
5	Modificar una ubicación	• Los cambios deben quedar almacenados en la base de datos • La ubicación debe ser recuperable después de su modificación
7	Eliminar una ubicación	• Las ubicaciones descendientes, si las hubiese, se convierten en ubicaciones de inicio. • Los equipos informáticos directamente asignados a la ubicación, si los hubiese, quedarían desasignados • La ubicación debe quedar eliminada de la base de datos
8	Visualizar equipos informáticos	• Se encuentran todos los equipos informáticos almacenados en la base de datos

9	Registrar un equipo informático	• Si los datos introducidos sobre el equipo informático se encuentran repetidos, el sistema notifica al usuario del conflicto • El equipo informático debe ser recuperable después de su creación
10	Modificar un equipo informático	• Los cambios deben quedar almacenados en la base de datos • El equipo informático debe ser recuperable después de su modificación
11	Eliminar un equipo informático	• El equipo informático elimina las relaciones con otras entidades del sistema como, por ejemplo, ubicaciones • La información de monitorización del equipo informático permanece en el sistema, accesible desde datos agrupados • El equipo informático queda eliminado de la base de datos
12	Asignar un equipo informático a una ubicación	• Si el equipo ya se encuentra asignado a una ubicación, reasigna el equipo a la nueva • En la base de datos se establecen las relaciones entre equipo y ubicación
13	Desasignar un equipo informático de una ubicación	• En la base de datos se eliminan las relaciones entre equipo y ubicación
14	Visualizar tablero	• Los datos mostrados deben figurar acordes al intervalo temporal especificado
16	Consultar el estado de uso en tiempo real de un equipo informático	• Los datos deben pertenecer al equipo informático consultado • Los datos visualizados deben ser inferiores a 5 minutos

17	Consultar el histórico de uso de un equipo informático	• Los datos deben pertenecer al equipo informático consultado • Los datos pueden tener valores por defecto, por ejemplo, para interpolar dos muestras • El intervalo de la muestra y el intervalo de fechas son configurables por el usuario • La ausencia de métricas debe ser distinguible de datos inválidos
18	Bloquear un ordenador remotamente	• Si el ordenador está encendido, debe bloquear la sesión del usuario inmediatamente • Si el ordenador está apagado, en el próximo encendido debe aparecer bloqueado • Desde el propio ordenador no será posible desbloquearlo
19	Reiniciar un ordenador remotamente	• Si el ordenador está encendido, éste se reinicia inmediatamente

Tabla 21. Criterios de aceptación sometidos al plan de pruebas

5.2. Resultados unitarias sobre la interfaz de comunicación API REST

Las pruebas unitarias de la interfaz de comunicación API REST se encuentran en el paquete de pruebas del código fuente del servidor. Dicho paquete contiene tres clases llamadas *UserControllerTests*, *LocationControllerTests* y *DeviceControllerTests* con la implementación de diferentes métodos que validan una serie de escenarios y sus respuestas según los parámetros de entrada empleados.

Cada vez que compilamos el proyecto, las pruebas se ejecutan disponiendo de una base de datos relacional y un directorio activo en memoria con datos de ejemplo. Además, cada prueba realizada es independiente entre sí.

Gracias a estas pruebas unitarias, hemos podido detectar problemas tempranamente durante el desarrollo sobre los cambios introducidos en los controladores y entidades. Sin embargo, los errores más frecuentes han sucedido

con *UserControllerTest* durante configuraciones posteriores en la seguridad del *framework*.

```
95      @Test
        public void cannotLoginWithBadCredentials() {
97          ResponseEntity<Logged> issueLogin = restTemplate.exchange(
98              LOGIN_ENDPOINT,
99              HttpMethod.POST,
100             this.withBasicAuth(FIRST_USER, FIRST_PASSWORD_BAD),
101             Logged.class);
102
103         assertThat(issueLogin.getStatusCode()).isEqualTo(HttpStatus.UNAUTHORIZED);
104     }
```

Ilustración 87. Extracto de prueba unitaria en *UserControllerTests*

Se han preparado 19 pruebas unitarias similares a las de la ilustración anterior con un resultado exitoso en todas ellas.

A continuación, presentamos las pruebas unitarias que ejecuta cada clase en las siguientes tablas:

- Clase *UserControllerTests*

Descripción	*Endpoint*	Parámetros de entrada	Resultado esperado
Un usuario no autenticado no puede acceder al sistema	(GET) /users/me	• Sin cabecera de X-Auth-Token	• 401 *Unauthorized*
Un usuario con credenciales correctas puede acceder al sistema	(POST) /users/sessions	• Autenticación *basic* con credenciales correctas	• 200 *Ok*
Un usuario introduciendo una contraseña errónea no puede acceder al sistema	(POST) /users/sessions	• Autenticación *basic* con usuario correcto y contraseña incorrecta	• 401 *Unauthorized*
Un usuario con credenciales correctas pero sin grupo asignado no puede acceder al sistema	(POST) /users/sessions	• Autenticación *basic* con credenciales correctas	• 403 *Forbidden*

Un usuario autenticado puede cerrar sesión	(DELETE) /users/sessions	• Cabecera X-Auth-Token válida	• 204 *No Content*

Tabla 22. Pruebas unitarias de *UserControllerTests*

- Clase *LocationControllerTests*

Descripción	*Endpoint*	Parámetros de entrada	Resultado esperado
Una ubicación puede ser localizada por su identificador	(GET) /locations/{uid}	• uid: identificador de ubicación válido	• 200 *Ok*
Una ubicación no puede ser localizada mediante un identificador inválido	(GET) /locations/{uid}	• uid: identificador de ubicación inválido	• 404 *Not Found*
Una ubicación puede ser creada si no existe previamente ninguna bajo el mismo nombre	(POST) /locations	• Cuerpo de la petición con nombre único	• 201 *Created*
Una ubicación puede cambiar de nombre	(PATCH) /locations/{uid}	• Cuerpo de la petición con nombre nuevo	• 200 *Ok*
Una ubicación existente puede ser eliminada	(DELETE) /locations/{uid}	• uid: identificador de ubicación válido	• 204 *No content*
Una ubicación inexistente no puede ser eliminada	(DELETE) /location/{uid}	• uid: identificador de ubicación no válido	• 404 *Not found*

Tabla 23. Pruebas unitarias de *LocationControllerTests*

- Clase *DeviceControllerTests*

Descripción	*Endpoint*	Parámetros de entrada	Resultado esperado
Un equipo puede ser localizado por su identificador	(GET) /devices/{uid}	• uid: identificador de equipo válido	• 200 *Ok*

Un equipo no puede ser localizado mediante un identificador inválido	(GET) /devices/{uid}	• uid: identificador de equipo inválido	• 404 *Not Found*
Un equipo puede ser creado si no existe previamente ninguno bajo el mismo nombre	(POST) /devices	• Cuerpo de la petición con nombre único	• 201 *Created*
Un equipo puede cambiar de nombre	(PATCH) /devices/{uid}	• Cuerpo de la petición con nombre nuevo	• 200 *Ok*
Un equipo no se puede actualizar si contiene atributos inválidos	(PATCH) /devices/{uid}	• Cuerpo de la petición con direcciones IP/MAC mal formadas	• 400 *Bad Request*
Un equipo puede poseer varios *tokens* de autenticación	(PATCH) /devices/{uid}	• Cuerpo de la petición con múltiples *tokens* de autenticación	• 200 *Ok*
Un equipo existente puede ser eliminado	(DELETE) /devices/{uid}	• uid: identificador de equipo válido	• 204 *No content*
Un equipo inexistente no puede ser eliminado	(DELETE) /devices/{uid}	• uid: identificador de ubicación no válido	• 404 *Not found*

Tabla 24. Pruebas unitarias de *DeviceControllerTests*

6. CONCLUSIONES

Con la presente memoria hemos recogido el trabajo realizado para la consecución de los objetivos inicialmente marcados aplicando metodologías de trabajo dentro de la ingeniería de software. Ahora, nos encontramos en el punto final del proyecto, donde reflexionaremos sobre los hitos conseguidos y lo que hemos aprendido en el camino.

En primer lugar, empezaremos repasando los objetivos cumplidos.

Hemos realizado un estudio de necesidades, soluciones y tecnologías. En el análisis preliminar perfilamos diferentes escenarios donde aplicar un sistema de gestión y monitorización de equipos informáticos. Luego estudiamos algunas de las soluciones existentes y pudimos observar múltiples enfoques al problema; de ahí, logramos tener una visión más completa de lo que queríamos realizar y lo materializamos en la propuesta de solución. De este modo, logramos enfocar nuestro proyecto a la realización de un sistema para una organización pequeña, como una biblioteca o una empresa, con herramientas sencillas de manejar por el usuario responsable del mantenimiento. Finalmente, planteamos y seleccionamos un conjunto de *frameworks* y herramientas que, tentativamente, nos ayudarían al desarrollo de un sistema propio.

Hemos diseñado un sistema informático especialmente orientado a la utilización de tecnologías web. Desde un inicio, tuvimos presente elegir un patrón de arquitectura compatible con tecnologías web como es el modelo/vista/controlador. Además, la comunicación entre servidor y cliente web viene dada por una interfaz de comunicación API REST que emplea protocolos de comunicación web HTTP. La elección de *frameworks* como Spring o Angular nos ha facilitado en gran medida la implementación de los patrones e interfaces correctos.

Hemos implementado un prototipo donde se pueda verificar la viabilidad y funcionalidad del sistema. En la propuesta de solución enunciamos algunas ideas sobre las características del sistema que concretamos con las historias de usuario. Pese a no haber completado todas las historias al exceder el tiempo total disponible, la priorización de tareas funcionó adecuadamente para seleccionar las principales

historias que dieran lugar a un producto deseable. Dichas historias están sujetas a criterios de aceptación que también hemos verificado.

Además de los mencionados objetivos, algo notable en este proyecto ha sido el aprendizaje continuo de nuevos *frameworks* y herramientas para dar solución a los problemas planteados. El núcleo del servidor ha sido desarrollado gracias a Spring Framework, el cual es bastante conocido para la implementación de interfaces de comunicación API REST; sin embargo, lo hemos complementado con tecnologías menos habituales como bases de datos de series temporales, colas de mensajes y directorios activos. Por otro lado, el agente de monitorización ha sido una pieza de software específica para este proyecto desarrollada sin ningún *framework* estructural por detrás, atendiendo a criterios de diseño propios. Por último, el cliente web ha sido el componente que ha unido toda la arquitectura subyacente bajo una presentación de interfaz de usuario elegante y sencilla de manejar.

También debemos hacer hincapié en la metodología de trabajo. En particular, el empleo de una metodología ágil como SCRUM ha dado frutos positivos en este proyecto para la planificación de historias de usuario y de iteraciones. Tanto el esfuerzo estimado de las tareas como la priorización de las mismas nos permitió evaluar en todo momento el correcto progreso del proyecto.

En términos personales, la realización de este proyecto ha sido un proceso de medio año de duración con el que he recibido una importante lección: la constancia.

La constancia ha sido la asignatura que tenía pendiente de aprobar y la pieza que, al unirla, me ha permitido aprovechar todos los conocimientos adquiridos en mi etapa como estudiante. Sin constancia, no podría estar escribiendo estas últimas palabras.

6.1. Mejoras y trabajo futuro

Esta memoria proporciona una extensa documentación que permite entender el diseño del sistema desde su concepción. Una de las finalidades de la memoria es que, llegado el caso, incluso una tercera persona pueda tomar el relevo en el desarrollo.

El trabajo realizado hasta ahora supone una sólida base para la continuidad del propio proyecto. Gracias a las decisiones arquitectónicas tomadas previamente, podemos implementar nuevas funcionalidades en el sistema sin involucrar cambios radicales en la estructura del mismo.

Cabe mencionar que cada cliente puede manifestar ideas diferentes atendiendo a sus propias necesidades. Nosotros, adoptando todavía el rol de cliente del producto, podemos pensar en cinco mejoras que aportarían un valor adicional al sistema.

Como primera mejora, podríamos implementar las últimas historias de usuario que quedaron inconclusas en el apartado de análisis. En este caso, las modificaciones sobre el diseño actual serían mínimas y conocemos el esfuerzo estimado necesario en todas ellas. Algunas de las más interesantes son la incorporación de alertas, el almacenamiento de registros o la detección de equipos en red local.

Como segunda mejora, podríamos enriquecer el repertorio de comandos de acción disponibles en los agentes. Por ejemplo, añadiendo un nuevo comando que habilite o deshabilite el uso de periféricos externos como *pendrives* en los equipos.

Como tercera mejora, deberíamos aprovechar todo el potencial de la interfaz de comunicación mediante colas. Como mencionamos en el apartado del diseño, podríamos implementar comandos de acción por lotes y comandos de acción programados. Para estos casos, el actual corredor de mensajes es insuficiente y deberíamos migrar a otro que admita la persistencia de mensajes encolados.

Como cuarta mejora, deberíamos añadir soporte en el agente para funcionar sobre equipos con Windows, ya que la implementación actual sólo contempla sistemas operativos basados en GNU/Linux. Afortunadamente, imaginamos esta situación y tanto la elección del lenguaje de programación mediante Python como la propia estructura de la aplicación permitirían esta adaptación.

Como quinta mejora, podríamos internacionalizar el cliente web. Actualmente, todos los textos de la interfaz de usuario se encuentran en español. A priori, sólo

compete cambios en el cliente web salvo que queramos guardar la elección del idioma en el servidor.

En resumen, todas estas mejoras añaden un plus de sofisticación al sistema. Dado el caso de tener que implementarlas, seguiríamos los mismos procesos de trabajo vistos en la memoria.

ANEXO 1. DESCRIPCIÓN DE CONTENIDOS SUMINISTRADOS

A continuación, enumeramos los contenidos suministrados en la entrega.

En la carpeta «Documentación» encontraremos:

- tfg-japp0005.pdf (~10MB): memoria en formato PDF.
- tfg-japp0005.odt (~10MB): memoria en formato ODT, compatible con LibreOffice Writer.
- tfg-japp0005.vpp (~10MB): diagramas UML y *wireframes* en Visual Paradigm.

En la carpeta «Fuentes» encontraremos:

- tfg-japp0005-backend (~1MB): carpeta con el código fuente del servidor.
- tfg-japp0005-frontend (~2MB): carpeta con el código fuente del cliente web.
- tfg-japp0005-agent (1MB): carpeta con el código fuente del agente de gestión y monitorización.

En la carpeta «Binarios» encontraremos:

- tfg-japp0005-backend.docker.tar.xz (~150MB): servidor empaquetado como imagen para Docker.
- tfg-japp0005-frontend.docker.tar.xz (~20MB): cliente web empaquetado como imagen para Docker.
- tfg-japp0005-agent.deb (~5MB): agente de gestión y monitorización empaquetado para Debian 11 bullseye amd64.
- tfg-japp0005-compose (~1MB): carpeta con la orquestación de servicios para Docker Compose.

ANEXO 2. MANUAL DE INSTALACIÓN DEL SISTEMA

La instalación del sistema comprende la preparación del servidor y de los agentes en los equipos objetivo. Hemos separado los pasos de preparación de cada uno de ellos en subcapítulos independientes.

Anexo 2.1. Instalación del servidor

Para la instalación del servidor, necesitaremos reunir los siguientes requisitos.

- Conexión a internet
- *Docker* y *Docker Compose*
- Contenidos suministrados: *tfg-japp0005-backend.docker.tar.xz*
- Contenidos suministrados: *tfg-japp0005-frontend.docker.tar.xz*
- Contenidos suministrados: *tfg-japp0005-compose*

Desde una terminal de comandos, importamos las imágenes del servidor y el cliente web en el repositorio local de imágenes de Docker.

```
docker load -i tfg-japp0005-backend.docker.tar.xz

docker load -i tfg-japp0005-frontend.docker.tar.xz
```

Después de completar estos pasos, podemos iniciar el servicio mediante Docker Compose.

```
docker compose -f tfg-japp0005-compose/compose.yml up --detach
```

En este punto, el sistema se encuentra escuchando en el puerto 80 bajo dirección IP local. Al entrar desde el navegador a http://localhost podemos acceder al cliente web, mientras que bajo http://localhost/v1 encontramos la interfaz de comunicación API REST.

Los usuarios existentes en el sistema están manejados a través del servicio OpenLDAP y siguen el esquema inicial que encontraremos en el fichero *bootstrap/openldap/schema.ldif*. Un usuario específico ha sido preparado para la ocasión:

- Usuario: **tribunal**
- Contraseña: **lanubirt**

Por último, para comprobar el estado de los servicios orquestados, detenerlos o eliminarlos, invocamos nuevamente a Docker Compose.

```
docker compose -f tfg-japp0005-compose/compose.yml ps

docker compose -f tfg-japp0005-compose/compose.yml stop

docker compose -f tfg-japp0005-compose/compose.yml down --volumes
```

Anexo 2.1.1. Configuraciones adicionales

Por comodidad, los servicios PostgreSQL, InfluxDB y OpenLDAP vienen incluidos en la orquestación de servicios del fichero *compose.yml*. Si dispusiéramos de estos servicios externamente y quisiéramos conectarnos a ellos, podemos editar las variables de entorno del servidor en la ruta *environ/backend.env* conforme a la siguiente tabla:

Variable de entorno	Servicio	Función
SPRING_DATASOURCE_URL	PostgreSQL	URL de conexión
SPRING_DATASOURCE_USERNAME	PostgreSQL	Usuario
SPRING_DATASOURCE_PASSWORD	PostgreSQL	Contraseña
SPRING_LDAP_URLS_0	OpenLDAP	URL de conexión
SPRING_LDAP_USERNAME	OpenLDAP	Usuario
SPRING_LDAP_PASSWORD	OpenLDAP	Contraseña

SPRING_LDAP_USER_BASE	OpenLDAP	Espacio de búsqueda de usuarios
SPRING_LDAP_GROUP_BASE	OpenLDAP	Espacio de búsqueda de grupos
INFLUX_URL	InfluxDB	URL de conexión
INFLUX_USERNAME	InfluxDB	Usuario
INFLUX_PASSWORD	InfluxDB	Contraseña
INFLUX_TOKEN	InfluxDB	*Token* de administración
INFLUX_ORG	InfluxDB	Organización
INFLUX_BUCKET	InfluxDB	Cubeta

Tabla 25. Lista de variables de entorno del servidor

Las variables de entorno preconfiguradas en este fichero son las siguientes:

```
  GNU nano 5.4              environ/backend.env
LANG=C.UTF-8
TZ=Europe/Madrid
SPRING_DATASOURCE_URL=jdbc:postgresql://postgresql:5432/backend
SPRING_DATASOURCE_USERNAME=xxxxx
SPRING_DATASOURCE_PASSWORD=xxxxx
SPRING_LDAP_URLS_0=ldap://openldap:1389
SPRING_LDAP_USERNAME=cn=xxxxx,dc=domain,dc=tld
SPRING_LDAP_PASSWORD=xxxxx
SPRING_LDAP_USER_BASE=ou=users,dc=domain,dc=tld
SPRING_LDAP_GROUP_BASE=ou=groups,dc=domain,dc=tld
INFLUX_URL=http://influxdb:8086
INFLUX_USERNAME=xxxxx
INFLUX_PASSWORD=xxxxx
INFLUX_TOKEN=xxxxx
INFLUX_ORG=organization
INFLUX_BUCKET=bucket
```

Ilustración 88. Variables de entorno preconfiguradas en el servidor

Anexo 2.2. Instalación del agente

Para la instalación del agente, necesitaremos reunir los siguientes requisitos.

- Conexión a internet
- Utilidades curl y nano
- Sistema operativo Debian 11 bullseye, amd64, versión escritorio
- Contenidos suministrados: *tfg-japp0005-agent.deb*

En primer lugar, descargamos e instalamos la última versión de Osquery disponible para Debian desde su página web [97].

```
curl -OJL https://pkg.osquery.io/deb/osquery_5.4.0-1.linux_amd64.deb

apt update && apt install ./osquery_5.4.0-1.linux_amd64.deb
```

Posteriormente, instalamos el empaquetado en Debian del agente.

```
apt update && apt install ./tfg-japp0005-agent.deb
```

A continuación, configuramos el contenido del fichero */etc/gesmonio/agent.conf* especificando el *token* de autenticación y los parámetros de conexión con el servidor correspondientes.

```
 GNU nano 5.4                /etc/gesmonio/agent.conf
[connection]
token = "00010203-0405-0607-0809-0a0b0c0d0e0f"
server = "localhost"
port = 80
secure = false
```

Ilustración 89. Fichero de configuración del agente

Por último, relanzamos el agente con la nueva configuración.

```
systemctl restart gesmonio-agent.service
```

ANEXO 3. ACCESO A INSTALACIÓN DE PRUEBAS

Desde la dirección web https://gesmon.io podemos acceder a una instalación de pruebas donde se encuentra el sistema desplegado. Un usuario específico ha sido preparado para la ocasión.

- Usuario: **tribunal**
- Contraseña: **lanubirt**

Los parámetros de configuración del agente cambian en cuanto a *server*, *port* y *secure* como mostramos en la siguiente ilustración.

Ilustración 90. Configuración del agente para la instalación de pruebas

Para enriquecer la experiencia de todos los usuarios que accedan a la instalación de pruebas, rogamos que no se borren equipos ni ubicaciones previamente existentes. En cambio, uno mismo sí puede crear sus propios equipos y ubicaciones para eliminarlos posteriormente antes de cerrar sesión.

Esta instalación de pruebas estará operativa hasta finales de septiembre de 2022.

ANEXO 4. MANUAL DE USUARIO

Este manual de usuario describe las funciones del sistema implementado para la gestión y monitorización de equipos informáticos desde el cliente web.

Anexo 4.1. Inicio de sesión

El usuario puede acceder al sistema introduciendo sus credenciales. Dichas credenciales están externalizadas sobre el servicio LDAP de la organización.

Ilustración 91. Caja de inicio de sesión

Anexo 4.2. Sección: Tablero

En el tablero se resume la información más relevante que ha sido recolectada por los equipos monitorizados del sistema atendiendo a un periodo temporal y a una ubicación seleccionada. La selección del periodo temporal puede establecerse sobre un rango (últimos 10 minutos, 30 minutos, 1 hora, etc) o sobre un intervalo de fechas concretas.

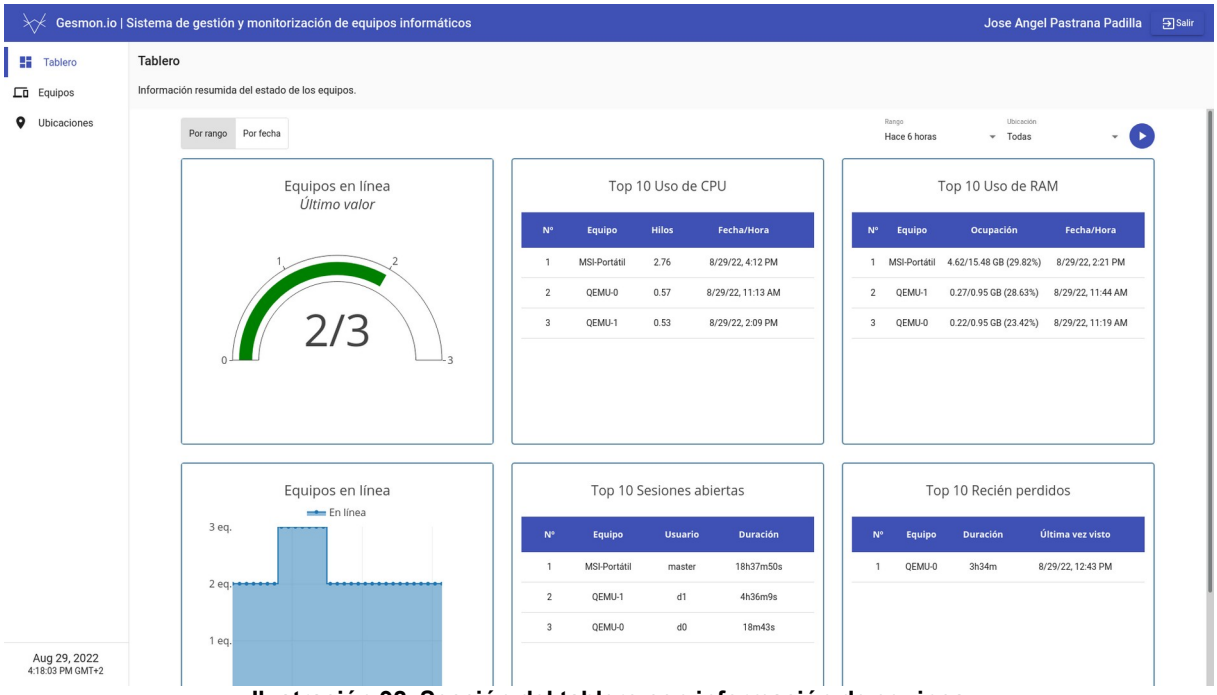

Ilustración 92. Sección del tablero con información de equipos

La vista del tablero se compone por seis cajas que pretenden mostrar la mayor cantidad de información útil para detectar anomalías en el funcionamiento de los equipos. A continuación, describimos cada una de ellas:

- Equipos en línea (Último valor): número de equipos en línea en el último momento del periodo y ubicación seleccionados. La finalidad de esta gráfica es la de leer de un único vistazo el número de equipos en línea y totales.

- Equipos en línea (gráfica): evolución de los equipos en línea a lo largo del periodo y ubicación seleccionados. La finalidad de esta gráfica es la de inspeccionar la evolución del número de equipos en línea.

- Top 10 Uso de CPU: entre todos los equipos del periodo y ubicación seleccionados, enumera los diez equipos cuyo procesador se encuentra con mayor carga de hilos. La finalidad de esta tabla es la de detectar situaciones de estrés en los equipos.

- Top 10 Uso de RAM: entre todos los equipos del periodo y ubicación seleccionados, enumera los diez equipos cuyo consumo de memoria física ha sido más alto, respecto de la capacidad máxima disponible. La

finalidad de esta tabla es la de detectar equipos bloqueados por insuficiencia de memoria.

- Top 10 Sesiones abiertas: entre todos los equipos del periodo y ubicación seleccionados, enumera los diez equipos con las sesiones abiertas más longevas. La finalidad de esta tabla es la detectar equipos cuyos usuarios no hayan cerrado sesión.

- Top 10 Recién perdidos: entre todos los equipos del periodo y ubicación seleccionados, enumera los últimos diez equipos que interactuaron con el sistema en algún momento, pero que llevan fuera de línea por más de 5 minutos. La finalidad de esta tabla es la detectar equipos que hayan dejado de mantener actividad con el sistema por cualquier motivo.

Anexo 4.3. Sección: Equipos

En la sección de equipos realizamos tareas de creación, consulta, modificación y eliminación de equipos en el sistema. Esta sección se compone de varias vistas sobre las que profundizaremos en los siguientes subcapítulos.

Anexo 4.3.1. Listado de equipos

La vista principal de la sección de equipos se compone por un listado paginado que muestra información destacada de cada uno de ellos.

Ilustración 93. Listado de equipos

En la esquina superior derecha encontramos el botón «Registrar nuevo equipo» con el que podemos dar de alta nuevos equipos.

En el lateral izquierdo de cada fila, el color verde simboliza el correcto estado en línea del equipo, mientras que el color rojo advierte que el equipo ha dejado de comunicarse con el servidor por más de un minuto. Por otro lado, desde la columna «Última vez visto» podemos conocer el último momento en el que hemos recibido comunicación con cada equipo.

Las columnas «Nombre» y «Última vez visto» son ordenables haciendo clic sobre sus respectivas cabeceras. En el caso de «Última vez visto», esto es útil para obtener un listado de equipos fuera de línea.

El resto de columnas nos aportan datos adicionales del equipo como el nombre que le hemos asignado, sus direcciones IP/MAC y las acciones de visualización, modificación y eliminación.

Anexo 4.3.2. Registrar nuevo equipo

Al hacer clic en «Registrar nuevo equipo», nos introducimos en una vista para crear un nuevo equipo en el sistema. El registro se encuentra guiado por una serie de pasos.

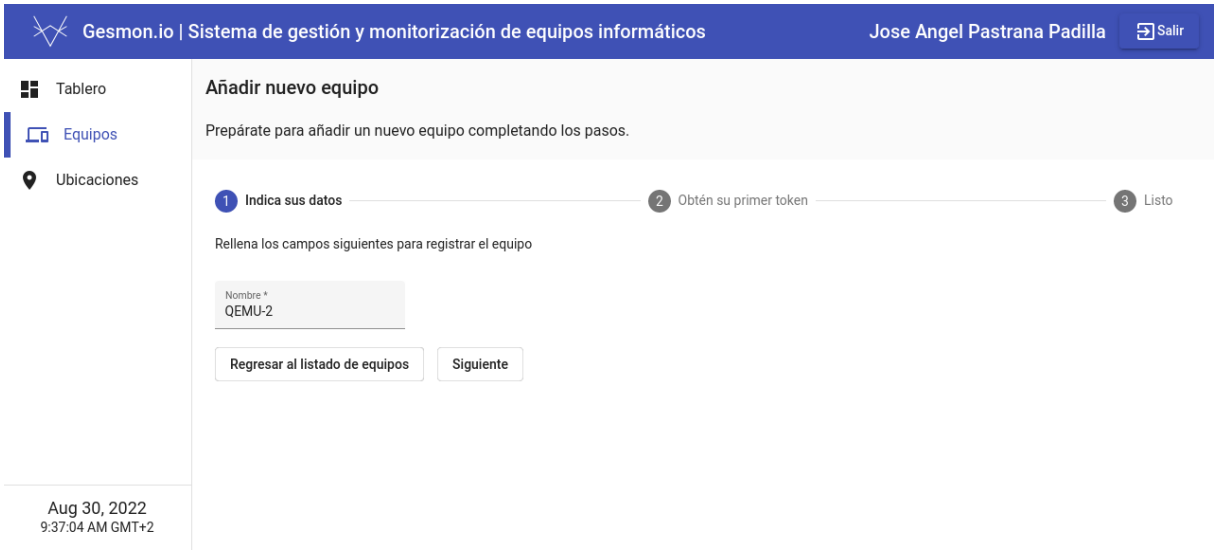

Ilustración 94. Registrar nuevo equipo. Primer paso

El primer paso, como puede observarse en la ilustración anterior, es rellenar los atributos lógicos del equipo como su nombre. El nombre es un campo único e identificativo.

Ilustración 95. Registrar nuevo equipo. Segundo paso

En el segundo paso obtenemos la confirmación de que el equipo ha sido registrado en el sistema. Además, un *token* de autenticación ha sido generado para vincular dicho equipo a un agente de gestión y monitorización.

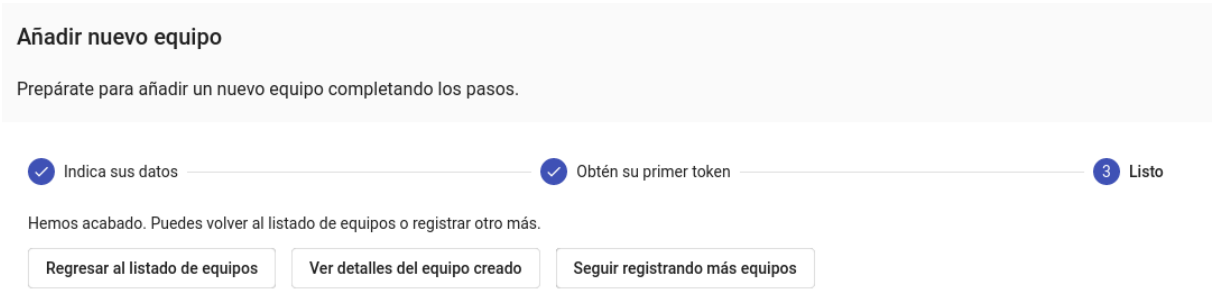

Ilustración 96. Registrar nuevo equipo. Tercer paso

En el último paso podemos regresar al listado de equipos, acceder a los detalles del equipo o seguir registrando más equipos.

Anexo 4.3.3. Detalles del equipo

Al hacer clic en la acción «Ver detalles del equipo» desde el listado de equipos, nos introducimos en una vista que profundiza sobre las características del mismo.

Esta vista se encuentra dividida en tres pestañas para separar la información general del equipo, información en tiempo real y el acceso al histórico de monitorización.

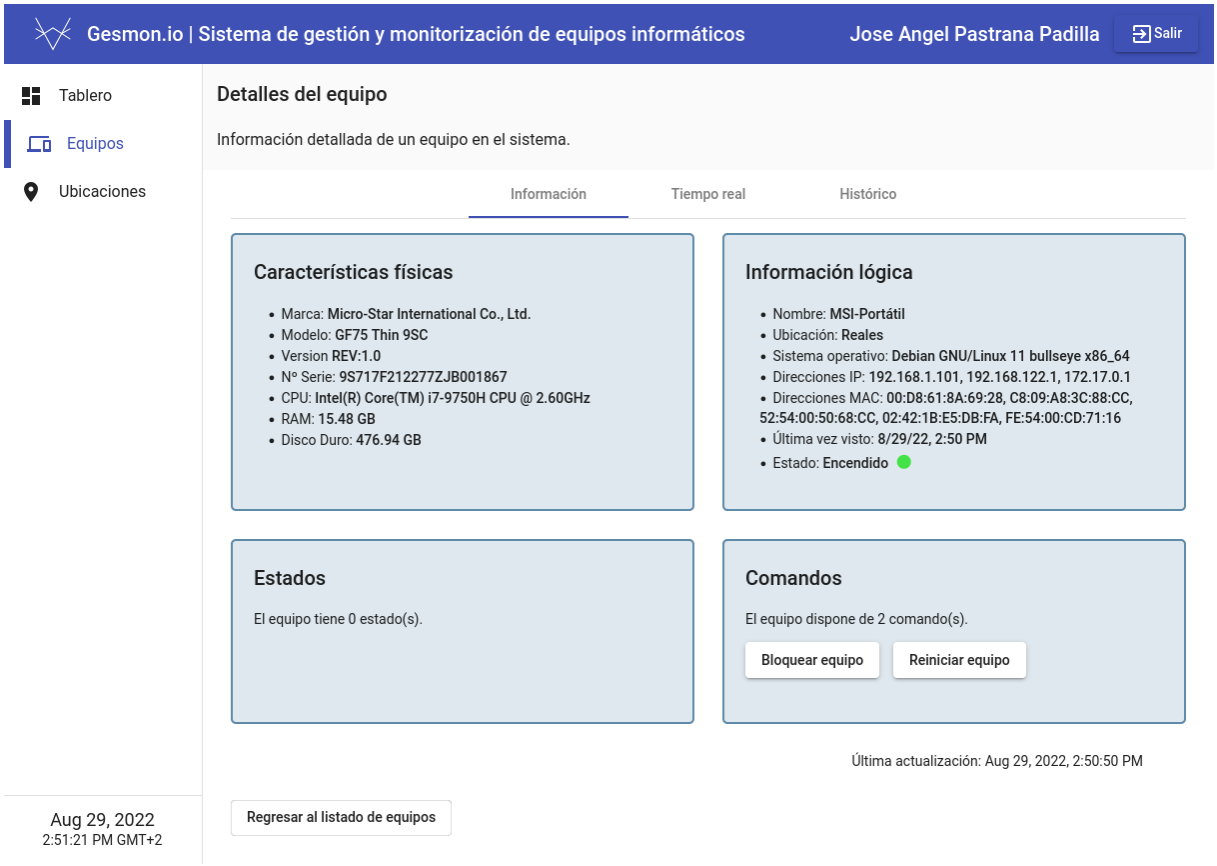

Ilustración 97. Vista de detalles del equipo. Pestaña Información

Desde la pestaña «Información» presentamos datos básicos del equipo mediante cuatro cajas con el siguiente contenido:

- Características físicas como marca, modelo, número de serie, procesador, memoria física total, entre otros.
- Información lógica como el nombre asignado en el sistema, la ubicación a la que pertenece, sistema operativo, direcciones IP/MAC, entre otros.
- Estados del equipo a consecuencia de algún comando previamente enviado, como el comando «Bloquear equipo».
- Comandos que pueden enviarse al equipo y que se encuentran soportados por el agente.

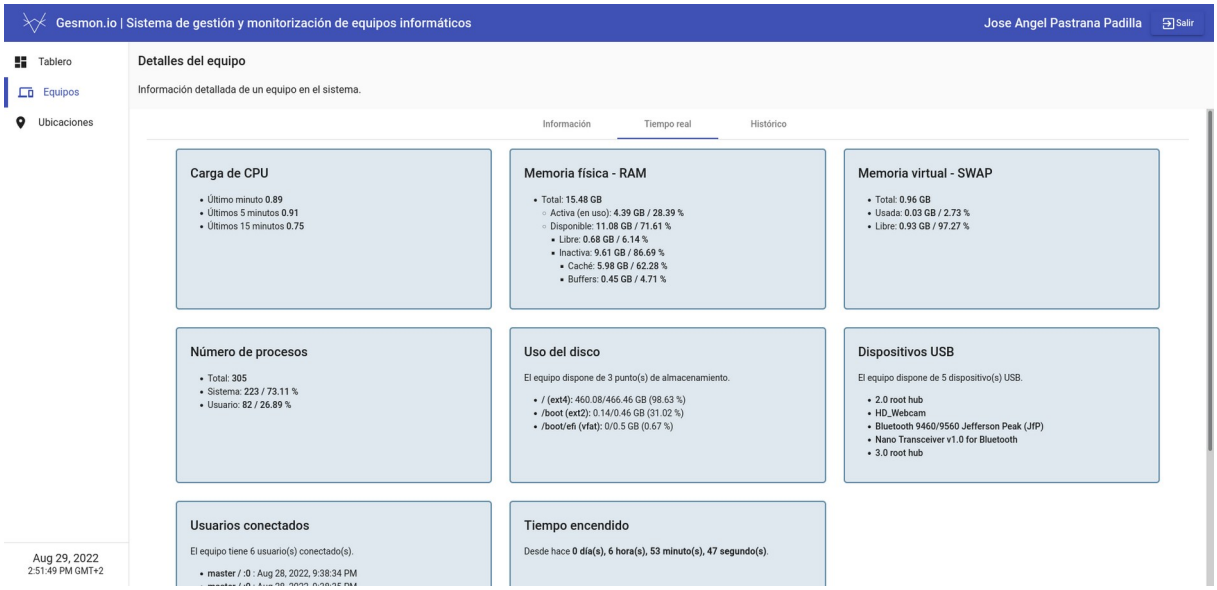

Ilustración 98. Vista de detalles del equipo. Pestaña Tiempo Real

Desde la pestaña «Tiempo real» presentamos la última métrica disponible del equipo mediante ocho cajas con el siguiente contenido:

- Carga de CPU con el número de hilos ocupados en promedio en el último minuto, últimos 5 minutos y últimos 15 minutos.
- Memoria física ocupada con el desglose en términos de memoria activa y disponible.
- Memoria virtual ocupada.
- Número de procesos del equipo desglosados entre procesos del sistema y procesos del usuario.
- Capacidad libre y máxima del disco duro.
- Nombre de los dispositivos USB actualmente conectados.
- Usuarios que han iniciado sesión en el equipo, incluidos los que han iniciado sesión remotamente o a través de la terminal de comandos.
- Tiempo encendido del equipo

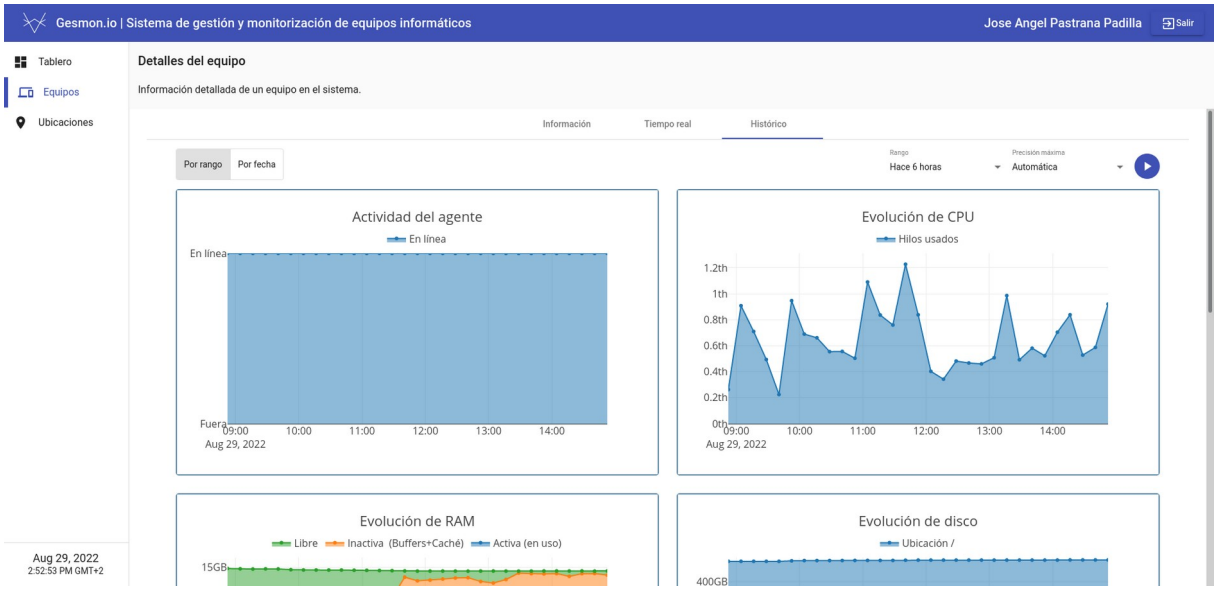

Ilustración 99. Vista de detalles del equipo. Pestaña Histórico

Desde la pestaña «Histórico» presentamos información visual del equipo en el pasado mediante cinco gráficas con el siguiente contenido:

- Estado en línea del agente a lo largo del tiempo.
- Evolución del número de hilos en ejecución a lo largo del tiempo.
- Evolución del consumo de memoria física a lo largo del tiempo.
- Evolución de la capacidad del disco duro
- Evolución del número de sesiones de usuario abiertas

Además de poder elegir el periodo temporal por rango o por intervalo de fechas a visualizar, también contamos con la posibilidad de elegir la precisión de las gráficas (número de puntos máximo para mostrar).

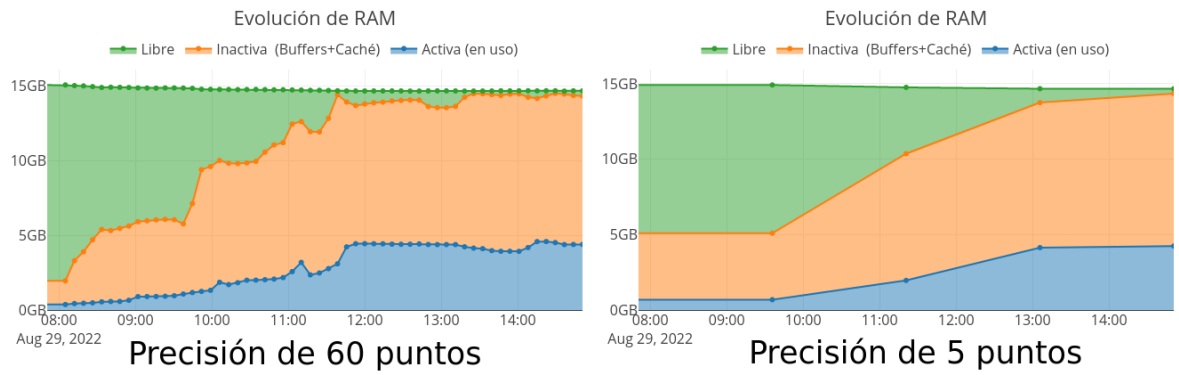

Ilustración 100. Ejemplo de precisión 60 puntos vs. 5 puntos

En la ilustración anterior observamos la configuración de precisión de 60 puntos contra 5 puntos. A menor precisión, las gráficas tienden a ser más suaves, pero ciertos detalles se pierden debido a que la distancia temporal entre un punto y el siguiente aumenta. Seleccionando precisión automática, el servidor computa la precisión óptima para el periodo temporal por rango o intervalo de fechas solicitado.

Anexo 4.3.4. Modificar equipo

Al hacer clic en la acción «Modificar este equipo» desde el listado de equipos, nos introducimos en una vista que nos permite modificar los atributos registrados del equipo en el sistema y los *tokens* de autenticación asociados.

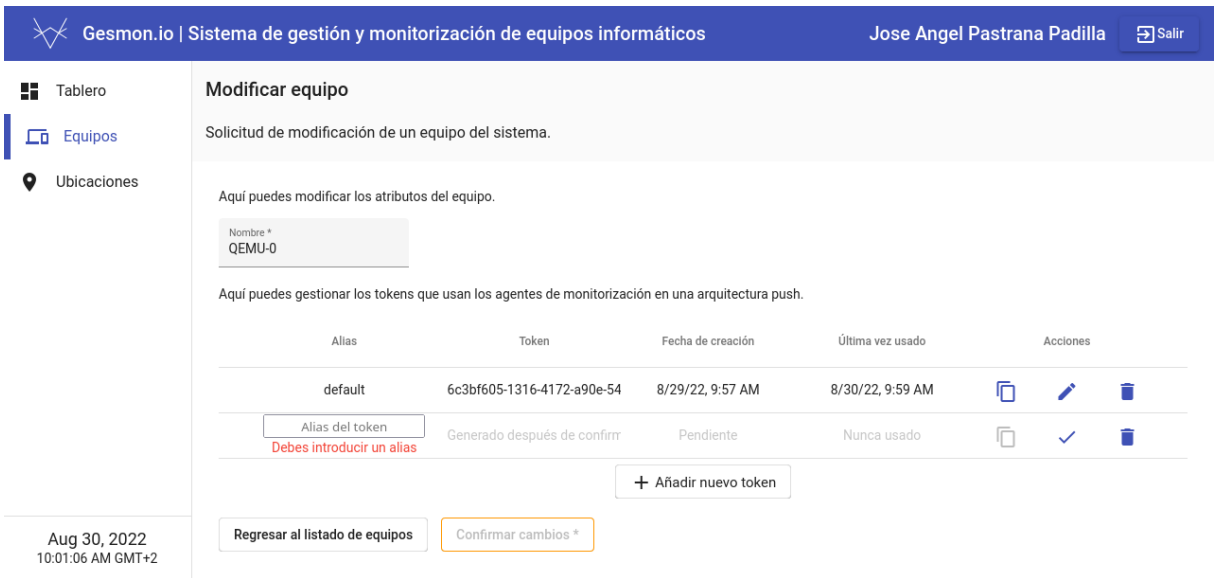

Ilustración 101. Vista de modificación del equipo

En general, no es necesario disponer de varios *tokens* de autenticación salvo que planifiquemos una sustitución del *token* actual manteniendo el antiguo y el nuevo operativos hasta completar el cambio.

Anexo 4.3.5. Eliminar equipo

Al hacer clic en la acción «Eliminar este equipo» desde el listado de equipos, nos introducimos en una vista que nos advierte sobre la acción destructiva que estamos emprendiendo.

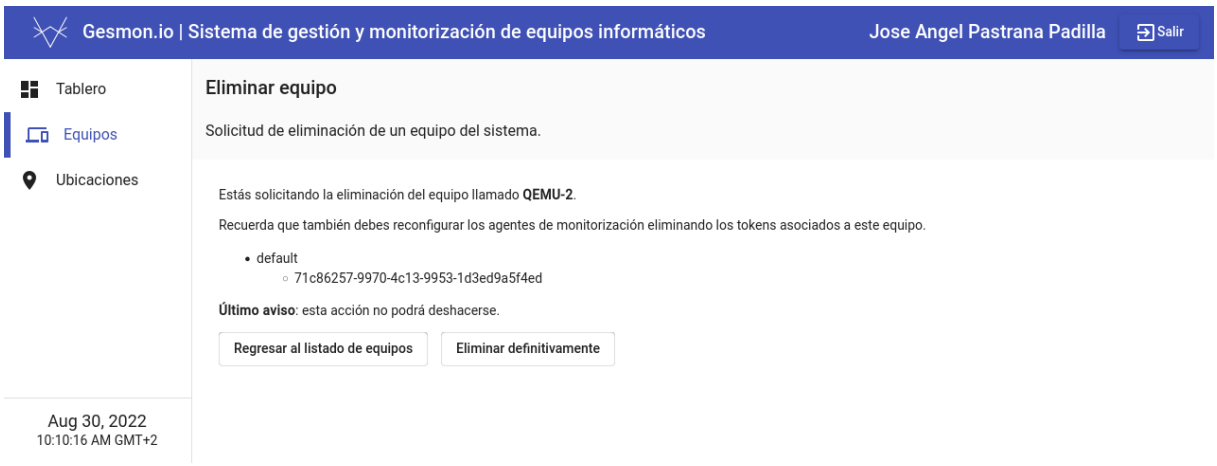

Ilustración 102. Vista de eliminación del equipo

Anexo 4.4. Sección: Ubicaciones

En la sección de ubicaciones realizamos tareas de creación, consulta, modificación y eliminación de ubicaciones en el sistema. Esta sección se compone de varias vistas sobre las que profundizaremos en los siguientes subcapítulos.

Anexo 4.4.1. Listado de ubicaciones

La vista principal de la sección de ubicaciones se compone por un listado paginado que muestra información destacada de cada uno de ellas.

Ilustración 103. Listado de ubicaciones

En la esquina superior derecha encontramos el botón «Registrar nueva ubicación» con el que podemos dar de alta nuevas ubicaciones.

En el lateral izquierdo de cada fila, el color verde simboliza que todos los equipos asociados con la ubicación se encuentran en línea, mientras que el color naranja advierte de algunos equipos fuera de línea. Por otro lado, el color rojo aparece cuando todos los equipos asociados se encuentran fuera de línea.

Las columnas «Número de equipos» y «Equipos en línea» nos indican un contador de los equipos asociados totales y en línea.

Las acciones disponibles en la columna «Acciones» nos permiten interactuar con la ubicación para visualizarla, modificarla o eliminarla.

Anexo 4.4.2. Registrar nueva ubicación

Al hacer clic en «Registrar nueva ubicación», nos introducimos en una vista para crear una nueva ubicación en el sistema. El registro se encuentra guiado por una serie de pasos.

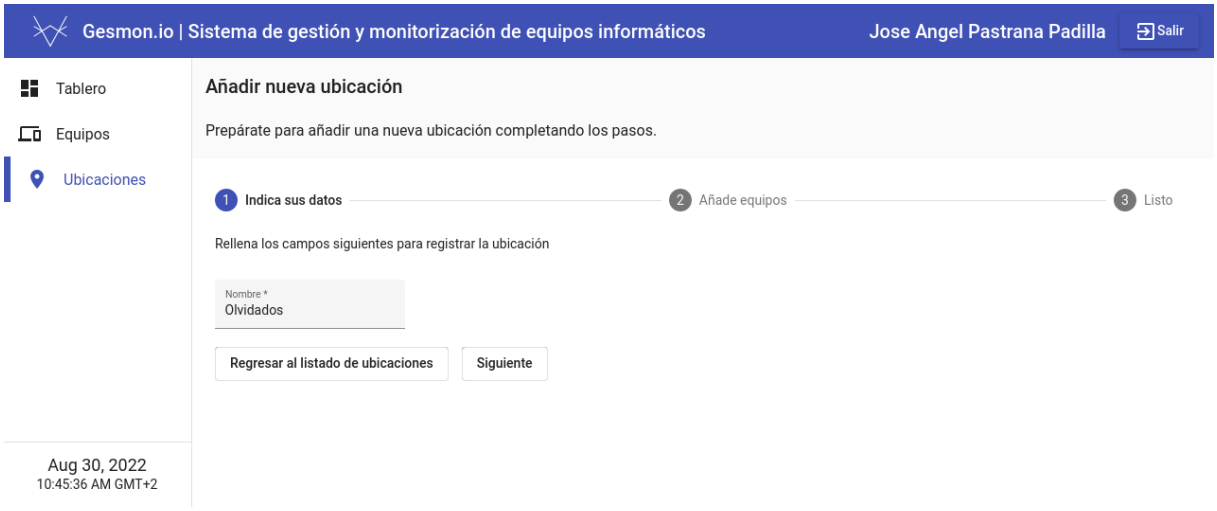

Ilustración 104. Registrar nueva ubicación. Primer paso

El primer paso, como puede observarse en la ilustración anterior, es rellenar los atributos lógicos de la ubicación como su nombre. El nombre es un campo único e identificativo.

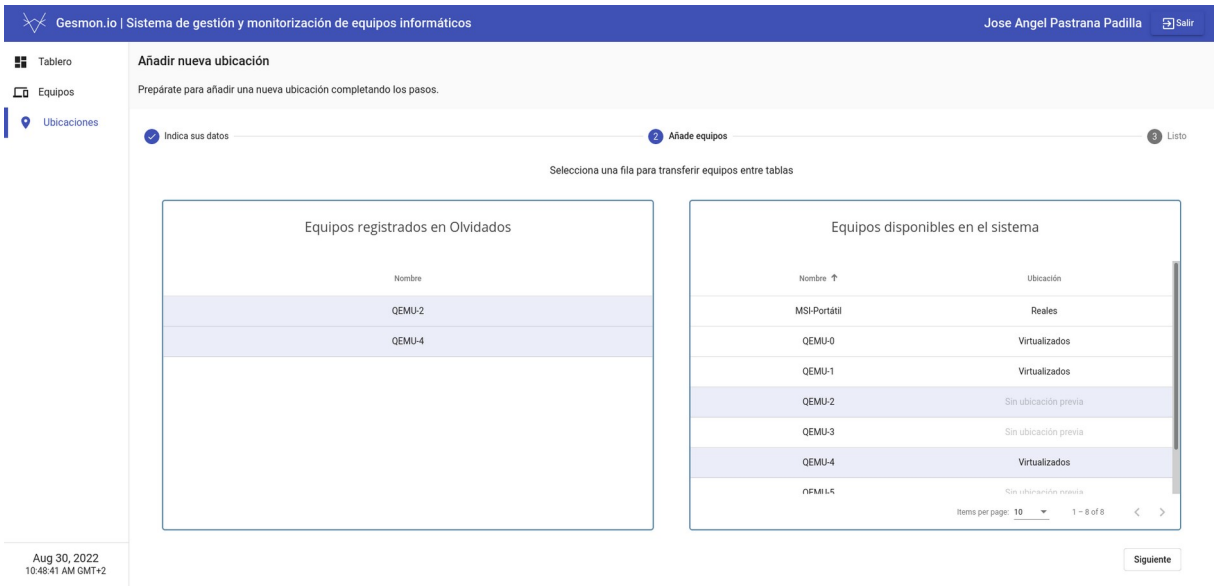

Ilustración 105. Registrar nueva ubicación. Segundo paso

En el segundo paso seleccionamos los equipos disponibles en el sistema para asociarlos con la nueva ubicación. En la caja de la izquierda disponemos del listado de equipos a asociar, mientras que en la caja de la derecha se encuentra un listado paginado de todos los equipos del sistema. Para seleccionar o deseleccionar un equipo en cualquiera de las dos cajas, basta con hacer clic sobre la fila.

Además, la caja de la derecha muestra en la columna «Ubicación» si el equipo que estamos seleccionando ya se encuentra registrado en otra ubicación. En caso de continuar, se desasociaría de la ubicación previa para asociarse con la actual.

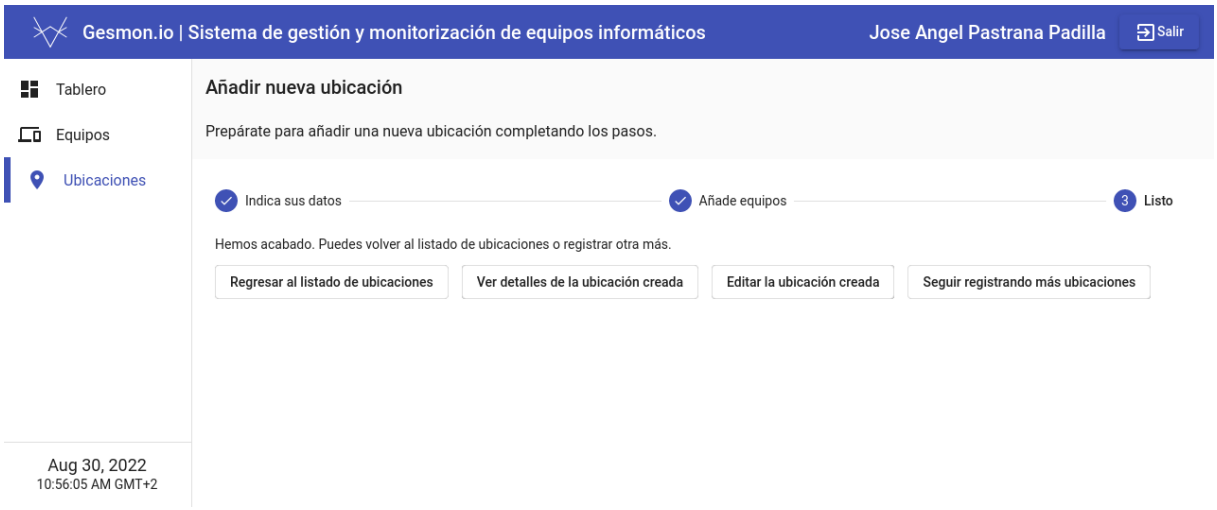

Ilustración 106. Registrar nueva ubicación. Tercer paso

En el último paso podemos regresar al listado de ubicaciones, acceder a los detalles de la ubicación o seguir registrando más ubicaciones.

Anexo 4.4.3. Detalles de la ubicación

Al hacer clic en la acción «Ver detalles de esta ubicación» desde el listado de ubicaciones, nos introducimos en una vista que profundiza sobre los detalles de la misma.

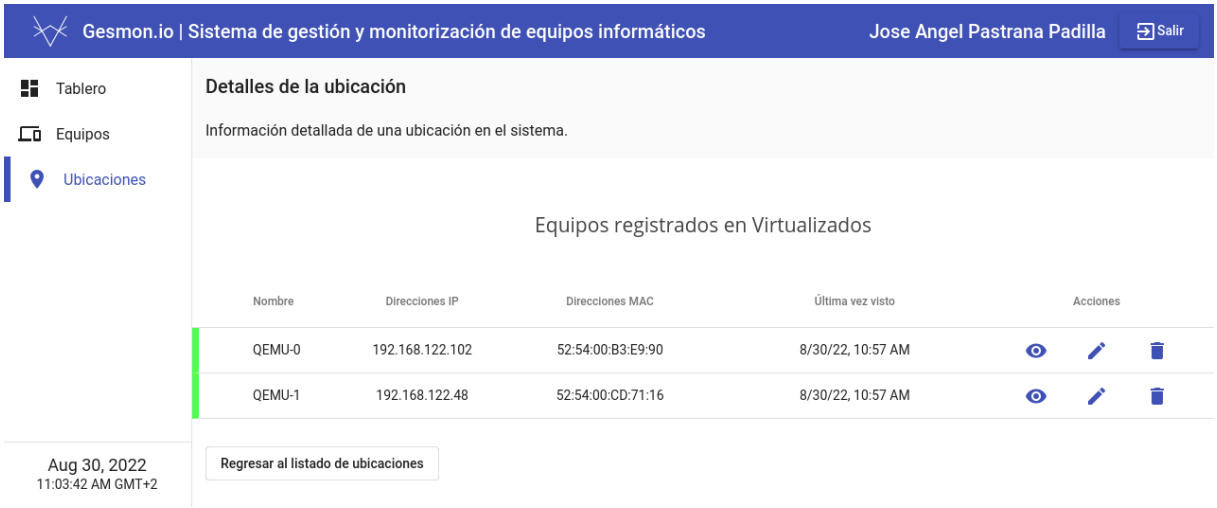

Ilustración 107. Vista de detalles de la ubicación

Esta vista guarda similitudes con respecto a la vista del listado de equipos. Aquí se reúnen los equipos asociados a la ubicación.

Anexo 4.3.4. Modificar ubicación

Al hacer clic en la acción «Modificar esta ubicación» desde el listado de ubicaciones, nos introducimos en una vista que nos permite modificar los atributos registrados de la ubicación en el sistema y los equipos asociados análogamente al visto durante la creación de una ubicación.

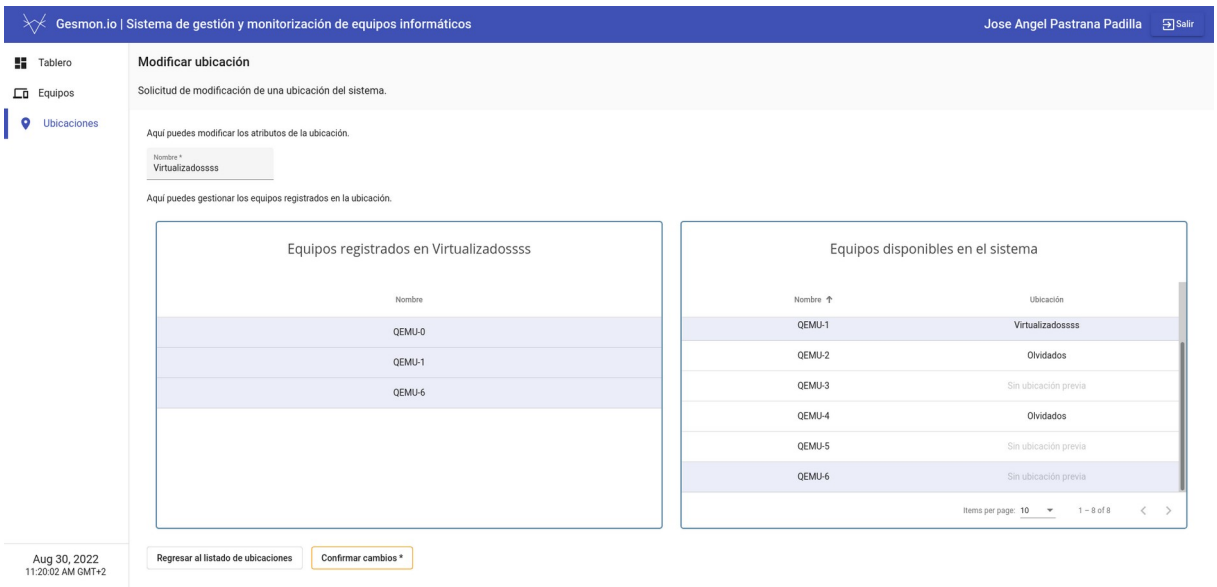

Ilustración 108. Vista de modificación de la ubicación

Anexo 4.4.5. Eliminar ubicación

Al hacer clic en la acción «Eliminar esta ubicación» desde el listado de equipos, nos introducimos en una vista que nos advierte sobre la acción destructiva que estamos emprendiendo.

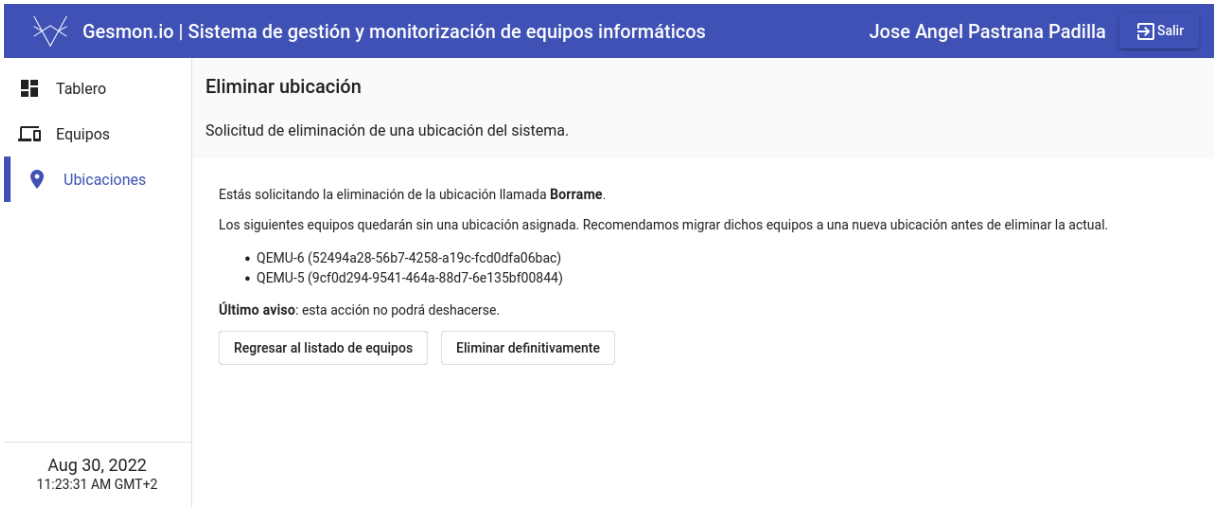

Ilustración 109. Vista de eliminación de ubicación

Después de la eliminación de la ubicación, los equipos asociados quedarán desasignados y la ubicación dejará de ser accesible desde el tablero.

BIBLIOGRAFÍA

[1] Claire Drumond (12-Agosto-2020), «*Scrum: qué es, cómo funciona y por qué es excelente*». [En línea]. Disponible en: https://www.atlassian.com/es/agile/scrum. [Acceso: 23-Marzo-2022]

[2] «*Salario de Técnico de sistemas en España*». [En línea]. Disponible en: https://es.indeed.com/career/técnico-de-sistemas/salaries. [Acceso: 25-marzo-2022]

[3] «*Salario de Ingeniero en sistemas en España*». [En línea]. Disponible en: https://es.indeed.com/career/ingeniero-en-sistemas/salaries. [Acceso: 25-marzo-2022]

[4] «*Microsoft Certified: Azure Administrator Associate*». [En línea]. Disponible en: https://docs.microsoft.com/es-es/certifications/azure-administrator/. [Acceso: 25-Agosto-2022]

[5] «*Linux Foundation Certified System Administrator (LFCS) Exam*». [En línea]. Disponible en: https://training.linuxfoundation.org/certification/linux-foundation-certified-sysadmin-lfcs/. [Acceso: 25-marzo-2022]

[6] «*When is System Administrator Appreciation Day*». [En línea]. Disponible en: https://sysadminday.com/?page_id=10. [Acceso: 25-Marzo-2022]

[7] Edgardo Rubén Frigo (09-Noviembre-2020), «*Qué es la Seguridad, la Seguridad Pública y la Seguridad Privada*». [En línea]. Disponible en: https://www.gestiondelriesgo.com/artic/discipl/4163.htm. [Acceso: 25-marzo-2022]

[8] Instituto Nacional de Ciberseguridad (03-septiembre-2020), «*¿Qué son y para qué sirven los SIEM, IDS e IPS?*». [En línea]. Disponible en: https://www.incibe.es/protege-tu-empresa/blog/son-y-sirven-los-siem-ids-e-ips. [Acceso: 25-marzo-2022]

[9] «*Herramientas de administración de AWS*». [En línea]. Disponible en: https://aws.amazon.com/es/products/management-tools/. [Acceso: 21-marzo-2022]

[10] «*Amazon CloudWatch: monitoreo de infraestructuras y aplicaciones*». [En línea]. Disponible en: https://aws.amazon.com/es/cloudwatch/. [Acceso: 26-marzo-2022]

[11] «*¿Qué es Amazon CloudWatch?*». [En línea]. Disponible en: https://docs.aws.amazon.com/es_es/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html. [Acceso: 26-marzo-2022]

[12] «*Características del producto Amazon CloudWatch*». [En línea]. Disponible en: https://aws.amazon.com/es/cloudwatch/features/. [Acceso: 26-marzo-2022]

[13] «*Introducción a Azure Monitor*». [En línea]. Disponible en: https://docs.microsoft.com/es-es/azure/azure-monitor/overview. [Acceso: 12-abril-2022]

[14] «*Introducción a las consultas de registro en Azure Monitor*». [En línea]. Disponible en: https://docs.microsoft.com/es-es/azure/azure-monitor/logs/get-started-queries. [Acceso: 12-abril-2022]

[15] «*Información general sobre los libros de Azure*». [En línea]. Disponible en: https://docs.microsoft.com/es-es/azure/azure-monitor/visualize/workbooks-overview. [Acceso: 12-abril-2022]

[16] «*Cisco Intersight Workload Optimizer for Hybrid Cloud*». [En línea]. Disponible en: https://www.cisco.com/site/us/en/products/computing/hybrid-cloud-operations/intersight-workload-optimizer/index.html. [Acceso: 29-marzo-2022]

[17] «*Cisco Intersight Platform Solution Overview* ». [En línea]. Disponible en: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/intersight/intersight-platform-so.html. [Acceso: 29-marzo-2022]

[18] «*Características de GLPI*». [En línea]. Disponible en: https://glpi-project.org/es/caracteristicas/. [Acceso: 29-marzo-2022]

[19] Mary Serie (02-julio-2021), «*Nagios Core vs. Nagios XI: 4 Key Differences*». [En línea]. Disponible en: https://www.nagios.com/news/2021/07/nagios-core-vs-nagios-xi/. [Acceso: 10-abril-2022]

[20] «*IT Monitoring Software from Nagios*». [En línea]. Disponible en: https://www.nagios.com/products/. [Acceso: 07-abril-2022]

[21] «*The Industry Standard in IT Monitoring*». [En línea]. Disponible en: https://assets.nagios.com/handouts/nagiosfusion/Nagios-Fusion-Product-Overview.pdf. [Acceso: 07-abril-2022]

[22] «*Nagios Exchange*». [En línea]. Disponible en: https://exchange.nagios.org/. [Acceso: 07-abril-2022]

[23] «*Enterprise IT monitoring with Zabbix*». [En línea]. Disponible en: https://www.zabbix.com/enterprise_monitoring. [Acceso: 10-abril-2022]

[24] «*Programa de Partners de Zabbix*». [En línea]. Disponible en: https://www.zabbix.com/partners?country=spain. [Acceso: 10-abril-2022]

[25] «*Monitoring Modules Library for Pandora FMS*». [En línea]. Disponible en: https://pandorafms.com/library/. [Acceso: 10-abril-2022]

[26] «*¿Por qué Pandora FMS Enterprise?*». [En línea]. Disponible en: https://pandorafms.com/es/por-que-pandora-fms-enterprise/. [Acceso: 10-abril-2022]

[27] «*Estudio de volumetría y capacidad*». [En línea]. Disponible en: https://pandorafms.com/manual/start?id=es/documentation/07_technical_annexes/03_capacity_planning. [Acceso: 10-abril-2022]

[28] Dimas Pardo López (27-julio-2021), «*Hughes India y Pandora FMS se unen como partners*». [En línea]. Disponible en: https://pandorafms.com/blog/es/hughes-india-pandora-fms/. [Acceso: 10-abril-2022]

[29] «*Monitorización como Servicio (MaaS) con Pandora FMS*». [En línea]. Disponible en: https://pandorafms.com/es/monitorizacion-como-servicio/. [Acceso: 10-abril-2022]

[30] Tobias Doerffel (23-abril-2017), «*Announcing Veyon, the successor of iTALC*». [En línea]. Disponible en: https://sourceforge.net/p/italc/mailman/message/35802371/. [Acceso: 10-abril-2022]

[31] «*Cross-platform computer control and classroom management*». [En línea]. Disponible en: https://veyon.io/. [Acceso: 10-abril-2022]

[32] «*Multi-Platform Remote Control software*». [En línea]. Disponible en: https://www.netsupportmanager.com/. [Acceso: 10-abril-2022]

[33] Matt Felton (05-julio-2019), «*Visualizing AWS Logging Data in Azure Monitor*». [En línea]. Disponible en: https://journeyofthegeek.com/2019/07/05/establishing-a-single-pane-of-glass-with-azure-monitor-part-1/. [Acceso: 12-abril-2022]

[34] Servicio de Informática. Universidad de Alicante (20-agosto-2012), «*Modelo vista controlador (MVC)*». [En línea]. Disponible en:

https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html. [Acceso: 12-abril-2022]

[35] Rodrigo Gómez (11-noviembre-2015), «*¿Qué es MVC?*». [En línea]. Disponible en: https://rodrigogr.com/blog/modelo-vista-controlador/. [Acceso: 12-abril-2022]

[36] Leff, Avraham; James T. Rayfield (Septiembre 2001), «*Web-Application Development Using the Model/View/Controller Design Pattern*». IEEE Enterprise Distributed Object Computing Conference. pp. 118-127

[37] Peter Phaal (27-agosto-2012), «*Push vs Pull*». [En línea]. Disponible en: https://blog.sflow.com/2012/08/push-vs-pull.html. [Acceso: 13-abril-2022]

[38] Steve Mushero (02-agosto-2019), «*Push vs. Pull Monitoring Configs*». [En línea]. Disponible en: https://steve-mushero.medium.com/push-vs-pull-configs-for-monitoring-c541eaf9e927. [Acceso: 14-abril-2022]

[39] «*What is a Relational Database (RDBMS)?*». [En línea]. Disponible en: https://www.oracle.com/database/what-is-a-relational-database/. [Acceso: 14-abril-2022]

[40] Tamara Pattinson (13-agosto-2020), «*Relational vs. non-relational databases*». [En línea]. Disponible en: https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases. [Acceso: 14-abril-2022]

[41] Matt Asay (26-junio-2019), «*Why time series databases are exploding in popularity*». [En línea]. Disponible en: https://www.techrepublic.com/article/why-time-series-databases-are-exploding-in-popularity/. [Acceso: 14-abril-2022]

[42] «*FAQ Thymeleaf*». [En línea]. Disponible en: https://www.thymeleaf.org/faq.html. [Acceso: 14-abril-2022]

[43] «*What is Apache FreeMarker?*». [En línea]. Disponible en: https://freemarker.apache.org/. [Acceso: 14-abril-2022]

[44] «*Apache FreeMarker Manual*». [En línea]. Disponible en: https://freemarker.apache.org/docs/versions_2_0.html. [Acceso: 14-abril-2022]

[45] «*Mustache - Logic-less templates*». [En línea]. Disponible en: https://mustache.github.io/mustache.5.html. [Acceso: 14-abril-2022]

[46] «*Jinja Documentation*». [En línea]. Disponible en: https://jinja.palletsprojects.com/en/3.1.x/. [Acceso: 14-abril-2022]

[47] «*Angular Developer Guides*». [En línea]. Disponible en: https://angular.io/guide/developer-guide-overview. [Acceso: 14-abril-2022]

[48] «*React - Getting Started*». [En línea]. Disponible en: https://reactjs.org/docs/getting-started.html. [Acceso: 14-abril-2022]

[49] «*VueJS - Overview*». [En línea]. Disponible en: https://www.tutorialspoint.com/vuejs/vuejs_overview.htm. [Acceso: 14-abril-2022]

[50] «*Spring Framework Overview*». [En línea]. Disponible en: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html. [Acceso: 14-abril-2022]

[51] «*Información general de ASP.NET Core MVC*». [En línea]. Disponible en: https://docs.microsoft.com/es-es/aspnet/core/mvc/overview?view=aspnetcore-6.0. [Acceso: 14-abril-2022]

[52] «*ASP.NET Core - Switch to MIT License*». [En línea]. Disponible en: https://github.com/dotnet/aspnetcore/issues/18873. [Acceso: 14-abril-2022]

[53] «*FastAPI - Features*». [En línea]. Disponible en: https://fastapi.tiangolo.com/features/. [Acceso: 14-abril-2022]

[54] «*Choosing the Right Storage Engine*». [En línea]. Disponible en: https://mariadb.com/kb/en/choosing-the-right-storage-engine/. [Acceso: 14-abril-2022]

[55] «*PostgreSQL vs. MySQL*». [En línea]. Disponible en: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-vs-mysql/. [Acceso: 14-abril-2022]

[56] «*Top 5 Features Of MongoDB*». [En línea]. Disponible en: https://www.mongodb.com/what-is-mongodb/features. [Acceso: 14-abril-2022]

[57] «*Introduction to Redis*». [En línea]. Disponible en: https://redis.io/docs/about/. [Acceso: 15-abril-2022]

[58] «*Compare InfluxDB to SQL databases*». [En línea]. Disponible en: https://docs.influxdata.com/influxdb/v1.8/concepts/crosswalk/. [Acceso: 15-abril-2022]

[59] «*Prometheus - Storage*». [En línea]. Disponible en: https://prometheus.io/docs/prometheus/latest/storage/. [Acceso: 15-abril-2022]

[60] «*TimescaleDB Overview*». [En línea]. Disponible en: https://docs.timescale.com/timescaledb/latest/overview/. [Acceso: 15-abril-2022]

[61] «*Grafana Features*». [En línea]. Disponible en: https://grafana.com/grafana/. [Acceso: 15-abril-2022]

[62] «*What is Chronograf?*». [En línea]. Disponible en: https://www.influxdata.com/time-series-platform/chronograf/. [Acceso: 15-abril-2022]

[63] «*Plotly Open Source Graphing Libraries*». [En línea]. Disponible en: https://plotly.com/graphing-libraries/. [Acceso: 15-abril-2022]

[64] «*When to use Pushgateway*». [En línea]. Disponible en: https://prometheus.io/docs/practices/pushing/. [Acceso: 15-abril-2022]

[65] Martina Wittmann (23-marzo-2017), «*SNMP Explained: What You Must Know About Monitoring via MIB and OIDs*». [En línea]. Disponible en: https://blog.paessler.com/snmp-monitoring-via-oids-mibs. [Acceso: 12-abril-2022]

[66] «*Osquery - Schema*». [En línea]. Disponible en: https://osquery.io/schema/5.2.2/. [Acceso: 15-abril-2022]

[67] «*About QEMU*». [En línea]. Disponible en: https://www.qemu.org/docs/master/about/index.html. [Acceso: 23-mayo-2022]

[68] «*Bootstrap - Grid system*». [En línea]. Disponible en: https://getbootstrap.com/docs/5.1/layout/grid/. [Acceso: 13-abril-2022]

[69] «*SDL Wiki*». [En línea]. Disponible en: https://wiki.libsdl.org/FAQs. [Acceso: 13-abril-2022]

[70] «*Using ActiveMQ*». [En línea]. Disponible en: https://activemq.apache.org/using-activemq-5. [Acceso: 13-abril-2022]

[71] «*OpenLDAP, Public License*». [En línea]. Disponible en: https://www.openldap.org/software/release/license.html. [Acceso: 13-abril-2022]

[72] «*Docker overview*». [En línea]. Disponible en: https://docs.docker.com/get-started/overview/. [Acceso: 13-abril-2022]

[73] «*Overview of Docker Compose*». [En línea]. Disponible en: https://docs.docker.com/compose/#compose-v2-and-the-new-docker-compose-command. [Acceso: 13-abril-2022]

[74] «*Caddy Documentation - Getting Started*». [En línea]. Disponible en: https://caddyserver.com/docs/getting-started. [Acceso: 13-abril-2022]

[75] «*Nginx - Basic HTTP server features*». [En línea]. Disponible en: https://nginx.org/en/#basic_http_features. [Acceso: 13-abril-2022]

[76] Max Rehkopf (25-septiembre-2019), «*Historias de usuario con ejemplos y plantilla*». [En línea]. Disponible en: https://www.atlassian.com/es/agile/project-management/user-stories. [Acceso: 16-abril-2022]

[77] Mike Cohn (10-septiembre-2019), «*Why the Fibonacci Sequence Works Well for Estimating*». [En línea]. Disponible en: https://www.mountaingoatsoftware.com/blog/why-the-fibonacci-sequence-works-well-for-estimating. [Acceso: 16-abril-2022]

[78] Chiyana Simões (14-julio-2020), «*MoSCoW. ¿Qué es y cómo priorizar en el desarrollo de tu aplicación?*». [En línea]. Disponible en: https://www.itdo.com/blog/moscow-que-es-y-como-priorizar-en-el-desarrollo-de-tu-aplicacion/. [Acceso: 16-abril-2022]

[79] José Miguel Espinar (12-mayo-2016), «*¿Cómo Priorizar requisitos? La Técnica de priorización MOSCOW*». [En línea]. Disponible en: https://proagilist.es/blog/agilidad-y-gestion-agil/priorizar-requisitos-tecnica-priorizacion-moscow/. [Acceso: 17-abril-2022]

[80] Juan Luis Vila Grau (01-junio-2017), «*Guía de uso paso a paso de los diagrama de quemado*». [En línea]. Disponible en: https://proagilist.es/blog/agilidad-y-gestion-agil/guia-uso-paso-paso-los-diagrama-quemado/. [Acceso: 24-abril-2022]

[81] David Baquero (23-noviembre-2020), «*Burndown y Burnup para el Scrum Master*». [En línea]. Disponible en: https://enciendelaluz.es/burnup-y-burndown-para-el-scrum-master/. [Acceso: 24-abril-2022]

[82] «*¿Qué es la estimación de costes en gestión de proyectos?*». [En línea]. Disponible en: https://www.wrike.com/es/project-management-guide/faq/que-es-la-estimacion-de-costes-en-gestion-de-proyectos/. [Acceso: 24-abril-2022]

[83] «*Salario de Analista programador en España*». [En línea]. Disponible en: https://es.indeed.com/career/analista-programador/salaries. [Acceso: 24-abril-2022]

[84] «*Salario de Desarrollador de software en España*». [En línea]. Disponible en: https://es.indeed.com/career/desarrollador-de-software/salaries. [Acceso: 24-abril-2022]

[85] Francisco José García Peñalvo; Alicia García Holgado (21-febrero-2018), «*Modelo de dominio - Ingeniería de software I - Universidad de Salamanca*». [En línea]. Disponible en: https://repositorio.grial.eu/bitstream/grial/1153/1/8.%20Modelo%20de%20dominio.pdf. [Acceso: 25-abril-2022]

[86] Edgar Frank Codd (Mayo 1971), «*Further Normalization of the Data Base Relational Model*». Courant Comp. Sc. Symp. 6: Data Base Systems, Prentice-Hall, N.J., pp. 65–98

[87] «*Introduction of Database Normalization*». [En línea]. Disponible en: https://www.geeksforgeeks.org/introduction-of-database-normalization/. [Acceso: 09-julio-2022]

[88] «*InfluxDB Key Concepts*». [En línea]. Disponible en: https://docs.influxdata.com/influxdb/v2.3/reference/key-concepts/. [Acceso: 09-julio-2022]

[89] «*Novell Doc Reference: Structure of an LDAP Directory Tree*». [En línea]. Disponible en: https://www.novell.com/documentation/opensuse110/opensuse110_reference/data/sec_ldap_tree.html. [Acceso: 10-julio-2022]

[90] «*ActiveMQ Protocols*». [En línea]. Disponible en: https://activemq.apache.org/protocols. [Acceso: 23-abril-2022]

[91] «*STOMP Protocol Specification, Version 1.2*». [En línea]. Disponible en: . [Acceso: 23-abril-2022]

[92] «*Simple Broker - Web on Servlet Stack*». [En línea]. Disponible en: https://docs.spring.io/spring-framework/docs/5.3.x/reference/html/web.html#websocket-stomp-handle-simple-broker. [Acceso: 23-abril-2022]

[93] «*Enable STOMP - Web on Servlet Stack*». [En línea]. Disponible en: https://docs.spring.io/spring-framework/docs/5.3.x/reference/html/web.html#websocket-stomp-enable. [Acceso: 23-abril-2022]

[94] «*User destinations - Web on Servlet Stack*». [En línea]. Disponible en: https://docs.spring.io/spring-framework/docs/5.3.x/reference/html/web.html#websocket-stomp-user-destination. [Acceso: 23-abril-2022]

[95] «*DaoAuthenticationProvider - Spring Security*». [En línea]. Disponible en: https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/dao-authentication-provider.html. [Acceso: 14-mayo-2022]

[96] «*Angular Routing*». [En línea]. Disponible en: https://angular.io/guide/routing-overview. [Acceso: 07-mayo-2022]

[97] «*Osquery - Downloads*». [En línea]. Disponible en: https://osquery.io/downloads/official/. [Acceso: 25-agosto-2022]