UNIVERSITAT DE BARCELONA

Trabajo de Final de Grado

GRADO DE INGENIERIA INFORMÁTICA

Facultad de Matemáticas e Informática

Universidad de Barcelona

---

squid-remote: SISTEMA DE ANÁLISIS FORENSE REMOTO SOBRE ENDPOINTS

Rodrigo Cabezas Quirós

Director: Raül Roca Cánovas

Realizado en: Departamento de Matemáticas e Informática

Barcelona, 20 de Junio de 2021

Abstract

In any organization, having control over the endpoints is crucial to avoid, detect and give an effective and immediate response to cyberattacks. However, it is not until a later phase (post-cyberincident) that a thorough forensic analysis can be done and the origin, costs and scope of the incident identified. This second phase is essential for the organization as it serves as a guideline to handle and mitigate similar attacks in the future, as well as a method of evidence collection that can be used against the perpetrators of the attack.

This project focuses in the post-incident phase and its goal is to design a remote forensic analysis system that allows the monitoring of diverse information from a remote machine such as the processes running, the network traffic or the devices connected to the machine, among others.

To carry out this goal, a webapp, a client and a server have been implemented to execute the functions of control, information recovery and data exchange and processing, respectively.

Keywords: Webapp; Flask; Distributed system; Vue.js; Python; CSS; HTML; Heroku, Javascript; Forensic analysis; Cybersecurity.

Resumen

En toda organización, el control sobre los equipos finales es uno de los puntos clave para evitar, detectar y dar una respuesta eficaz e inmediata a los ciberataques. No obstante, no es hasta una fase posterior (post-ciberincidente) cuando es posible realizar un análisis forense exhaustivo que permite identificar el origen, alcance y costes del incidente. Esta segunda fase es esencial para que la organización pueda defenderse ante futuros ataques de naturaleza similar, así como para recopilar evidencias que puedan usarse en las actuaciones legales contra los autores del ataque.

Este trabajo está enfocado en esta segunda fase y consiste en diseñar un sistema de análisis forense remoto que permita monitorizar información diversa en una máquina remota como los procesos que se están ejecutando, el tráfico de red o la conexión de dispositivos, entre otros.

Para llevar estas funciones a cabo se ha diseñado una aplicación web, un cliente y un servidor que realizan las funciones de control, recopilado de la información y de intercambio y procesado de datos, respectivamente.

Palabras clave: Aplicación web; Flask; Sistema distribuido; Vue.js; Python; CSS; HTML; Heroku; Javascript; Análisis forense; Ciberseguridad.

Resum

En qualsevol organització, el control sobre els equips finals és un dels punts clau per evitar, detectar i donar una resposta eficaç i immediata als ciberatacs. Però és en una fase posterior (post-ciberincident) quan és possible realitzar un anàlisi forense exhaustiu que permet identificar l'origen, abast i costos de l'incident. Aquesta segona fase és essencial per a que l'organització pugui defensar-se de futurs atacs de naturalesa similar, així com per a recopilar evidències que puguin emprar-se en les actuacions legals contra els autors de l'atac.

Aquest treball està enfocat en aquesta segona fase i consisteix en la implementació d'un sistema d'anàlisi forense remot que permeti la monitorització d'informació diversa d'una màquina remota com els processos que s'estan executant, entre d'altres.

Per portar aquestes funcions a terme s'ha dissenyat una aplicació web, un client i un servidor que realitzen les funcions de control, recopilat de la informació i d'intercanvi i processament de dades, respectivament.

Paraules clau: Aplicació web; Flask; Sistema distribuït; Vue.js; Python; CSS; HTML; Heroku; Javascript; Anàlisi forense; Ciberseguretat.

Agradecimientos

Quiero agradecer en primer lugar a mi familia y en especial a mis padres, por su apoyo y ánimos que me han brindado a lo largo de estos años de la ingeniería.

También, a mis amigos y compañeros de carrera por su paciencia y confianza en las largas sesiones de testeo.

Muchas gracias a todos.

Índice

1 Introducción

1.1 Contexto

La ciberseguridad es la práctica encargada de la protección de sistemas informáticos, redes y programas de ataques digitales. Ataques cuyo enfoque se centra en la difusión, alteración o destrucción de información sensible, y que resultan en métodos de extorsión o en la disrupción del correcto funcionamiento de una organización. Para minimizar la exposición a estos ataques, es clave disponer de unas medidas de defensa a medida de la organización.

En una organización, las personas, las directivas y tecnologías se complementan entre sí para crear un sistema de defensa efectiva contra los ciberataques. Una buena defensa puede estar formada por múltiples capas y está distribuida entre los ordenadores, redes, programas o datos que se intenta proteger. Antivirus, firewalls, UTM (*Unified Threat Management)* o sistemas de detección / intrusión son ejemplos de sistemas de defensa, útiles para la prevención de ciberataques.

La problemática pues, llega cuando una organización sufre un ataque. En este caso los sistemas ya no requieren medidas preventivas, sino de mitigación y recuperación. Las grandes organizaciones disponen de planes, directivas y personal específico para tratar con estos incidentes. Estas pueden desde el minuto cero evaluar y localizar el origen de la amenaza y mitigarla. Sin embargo, las organizaciones más pequeñas pueden no disponer de los recursos suficientes para mantener a personal experto en ciberataques de forma continua, caso en que esta tarea es externalizada a empresas especializadas.

En ambos casos hay un factor clave: el tiempo. Tras una amenaza, una organización querrá ver restablecida su operación lo antes posible, por lo que se recurre a sistemas de análisis remoto. Estos sistemas permiten de forma remota y rápida conectarse a una máquina comprometida e iniciar las labores de investigación y mitigación de la amenaza.

En este trabajo se desarrollará desde cero un sistema de análisis forense remoto, squid-remote. Siguiendo los factores clave mencionados anteriormente, el sistema será distribuido y dispondrá de unos procesos de despliegue y ejecución rápidos, seguros y fáciles. La idea entonces, es que cualquier persona pueda solicitar el control remoto por parte un técnico y que este pueda realizar las labores de monitorización, rastreo y mitigación desde cualquier lugar y dispositivo.

1.2 Motivación

El principal motivo que me ha llevado a escoger este trabajo es la gran cantidad de áreas que abarca. Por una parte, está el diseño de la aplicación web que actúa como controlador y por otra la implementación del cliente y de la lógica distribuida del servidor, además de su despliegue. Partiendo de una implementación sencilla, el trabajo ya requiere de elementos muy diversos como desarrollo frontend y backend, base de datos y otras tecnologías.

Además de la diversidad del trabajo y la libertad de la que disponía para desarrollar el sistema, la constante aparición de retos, con la incorporación de nuevas tecnologías y funciones en el sistema, ha sido otro de los puntos clave que ha hecho que ni la dificultad ni las horas de dedicación fueran obstáculo en el proceso de desarrollo.

1.3 Aproximación inicial

Antes de enumerar los objetivos, requisitos e historias de usuario, es necesario tener una visión general previa sobre cómo se estructurará el sistema y que terminología se usará a lo largo de la memoria.

La finalidad del sistema es que se pueda realizar una conexión remota a un equipo, la intervención remota. En esta intervención remota habrá dos roles involucrados: el usuario y el técnico. El equipo del usuario será el que será objeto de la conexión, y el técnico será quien ejecute las acciones y diagnósticos sobre este de forma remota.

En lo que respecta a la arquitectura del sistema, estará dividido y distribuido en tres partes: cliente (el programa que el usuario afectado ejecuta), servidor e interfaz. Las tres son independientes entre sí, por lo que pueden estar desplegadas en entornos separados y existir sin que haya problemas de dependencias.

La conexión entre el cliente y el servidor seguirá el paradigma cliente – servidor, pero con una conexión inversa, es decir, el cliente preguntará cada cierto tiempo al servidor que es lo que ha de hacer. La comunicación entre servidor e interfaz será asíncrona y seguirá el patrón de diseño Observer. Cuando se produzca una actualización en los datos, el servidor notificará a la interfaz para que los muestre.

El cliente residirá en la máquina comprometida del usuario y será el componente que recompile la información que luego el técnico recibirá. También ejecutará las instrucciones que el técnico pueda mandar remotamente.

La interfaz será el medio con que el técnico interactuará y donde visualizará los datos que el cliente reporte. Únicamente mostrará los datos recopilados y permitirá ejecutar instrucciones en el cliente. Su único propósito es actuar como fachada del sistema.

Por último, el servidor, que actuará como intermediario entre el cliente y la interfaz. Cualquier comunicación que tenga que ocurrir entre ambos, pasará siempre por el servidor. Será el encargado de procesar los datos.

1.4 Objetivos

El objetivo principal de este trabajo es diseñar, implementar, desplegar y distribuir un sistema de análisis forense remoto sobre equipos finales.

1.4.1. Objetivos específicos

Para la realización del trabajo se han de cumplir los siguientes objetivos específicos:

- Diseño del cliente: Se ha de diseñar un cliente que se ejecute en la máquina de la máquina comprometida. Los datos que el cliente debe ser capaz de recuperar son los siguientes:
  - Información de los procesos que se ejecutan en la máquina remota.
  - Datos sobre el tráfico de red (capturado) en la red de la máquina comprometida.
  - Información sobre los dispositivos conectados.
  - Información sobre el rendimiento de la máquina remota.
  - Información diversa recuperada por consola de comandos remota.

  El cliente entonces, realizará tareas de monitorización del equipo y de ejecución de comandos de forma remota, siempre reportando los resultados al servidor.
- Diseño de la interfaz: Para la visualización de los resultados y para el control del cliente, es necesario una interfaz web. Esta actuará como controlador y lanzador de las siguientes funciones:
  - Ejecución de comandos en la máquina remota.
  - Envío de mensajes a la máquina remota.
  - Ejecución de función ping a la máquina.
  - Gestiones sobre la ejecución del cliente.
  - Gestiones relativas a la base de datos.
  - Cambio de funciones en el cliente.
- Diseño del servidor: El servidor actuará de intermediario entre interfaz y cliente. Sus funciones incluirán el procesado y tratamiento de datos para su distribución a cliente y/o interfaz. También realizará tareas relacionadas con la administración del sistema.
  - Gestión de cuentas de usuario.
  - Gestión de descargas.
  - Visualización de registros de conexión.

- Gestión de conexiones.
- Gestión de máquinas y sus casos.
- Visualización de registros de actividad.

1.5 Estructura del documento

El contenido de este documento se encuentra organizado en los siguientes capítulos:

- Análisis: Definición de requisitos que cada componente del sistema debe cumplir y desglose del producto backlog.
- Planificación: Explicación de la metodología de trabajo aplicada y distribución de las diferentes tareas en los periodos de desarrollo (sprints).
- Tecnologías: Listado de las tecnologías usadas en el sistema y justificación de porqué han sido escogidas.
- Diseño: Explicación de la arquitectura del sistema y de otros aspectos críticos del sistema.
- Implementación: Exposición de los aspectos técnicos que implementan el sistema internamente.
- Despliegue: Explicación de como se ha planteado y automatizado el proceso de entrega e implantación del sistema.
- Pruebas y resultados: Exposición de las pruebas realizadas en el sistema y los errores encontrados.
- Costes: Descripción de los costes económicos que supondría el desarrollo comercial del sistema.
- Trabajo futuro: Listado de las posibles y futuras líneas de mejora.
- Conclusiones: Resumen del trabajo contextualizado a la problemática inicial del trabajo.
- Anexos: Documentación y guías de uso y mantenimiento del sistema.

2 Análisis

En esta sección se detallan todos los requisitos que debe satisfacer el producto y el producto backlog generado a partir de todas las User Stories.

2.1 Requisitos

Los requerimientos de las diferentes partes se han dividido en requisitos generales y requisitos relacionados con la seguridad.

2.1.1. Cliente

El cliente será el programa que el usuario ejecutará en la máquina remota y que se encargará de recuperar la información y llevar a cabo las ordenes remotas.

Requisitos generales:

- Compatibilidad con diferentes plataformas (Linux, Windows).
- Ejecución sencilla, clicar y listo, sin instalación.
- Operación en cualquier red con salida a internet.
- Deberá poder cambiarse el idioma (castellano, catalán e inglés).
- Ha de permitir la comunicación del técnico con el cliente.
- Recuperación de información de los procesos en ejecución.
- Captura del tráfico de red.
- Identificación de los dispositivos conectados.
- Monitorización del rendimiento del equipo.
- Identificación de tarjetas de red.
- Identificación de características de hardware y software del equipo.
- Ejecución de comandos por consola.

El hecho de que el cliente no requiera de instalación tiene por objeto evitar que el cliente, quien probablemente requiera de asistencia tras sufrir algún ciberataque, se vea obligado a instalar en su equipo un software desconocido. Gracias a su diseño, una vez termine la intervención remota el cliente puede ser eliminado sin dejar ningún rastro en el sistema.

El técnico que realice la intervención en remoto puede necesitar la cooperación del usuario en algún caso, como por ejemplo para que desconecte un dispositivo o ejecute algún programa. Por este motivo, el cliente deberá ser capaz de mostrar los mensajes que el técnico necesite hacer llegar al cliente.

La información que se recopile sobre los procesos debe ser la suficiente como para que sea posible distinguir un proceso legítimo de uno que no lo sea (malware). Por este motivo, información como los hilos en uso, fecha de creación, usuario que ha creado el proceso o uso de CPU y memoria deberían recuperarse.

Los dispositivos deben poder ser identificados ya sea por nombre, identificador interno, fabricante o todas las anteriores. También deberán poder ser recuperados en cualquier momento.

Las características que se recuperen, tanto a nivel de hardware como software, serán útiles para buscar con más profundidad vulnerabilidades específicas del equipo. Es imperativo que se recuperen las tarjetas de red de que disponga la máquina, puesto que se usarán en la captura del tráfico de red. Además, en el caso de las tarjetas de red, las direcciones de cada una también deberán ser recopiladas para permitir, por ejemplo, la identificación del origen de los paquetes de red.

Por último, el cliente tendrá que "simular" un intérprete de comandos y permitir la ejecución de comandos remotos enviados por el técnico. La salida de estos comandos deberá ser enviada al servidor.

Requisitos relacionados con la seguridad:

- Establecimiento de una conexión segura y privada con el servidor.
- Implementación de medidas que eviten la ejecución accidental o malintencionada del cliente.

La conexión que realiza al servidor debe ser segura y sencilla. Para ello, el intercambio de información se realiza sobre HTTPS (puerto 443) evitando así la dependencia de VPNs, túneles SSH o la apertura de puertos en la red del cliente.

Dado que el cliente es capaz de ejecutar comandos remotos, es imprescindible que el cliente disponga de alguna medida para validar la conexión y evitar ejecuciones accidentales. Con esta medida también se evitan las ejecuciones malintencionadas, en caso de que algún atacante obtenga acceso a la interfaz y quiera conectarse a alguna máquina.

2.1.2. Servidor

Requisitos generales:

- Procesado de información del cliente.
- Intercambio de información mediante cache remota.
- Gestión de información sobre las máquinas existentes en base de datos.

- Gestión de información sobre las cuentas de usuario almacenadas de base de datos.
- Gestión de eventos cliente – interfaz.

Dado que el cliente realiza una conexión inversa por HTTPS no es posible transmitir los datos directamente, por lo que hay que pasar por un almacenamiento intermediario. De este modo el cliente almacenará los datos en una base de datos a modo de cache y la interfaz podrá recuperarlos. Será función del servidor entonces, la recuperación y actualización de esta información.

Un problema parecido y relacionado con el tipo de conexión entre el cliente y el servidor es el muestreo de información. No es viable que la interfaz esté realizando consultas continuamente a base de datos para ver si ha habido algún cambio. Para que la interfaz sepa cuando ha habido un cambio en la información, el servidor deberá llevar el control de un sistema de notificaciones entre él y la interfaz. Así, la interfaz solo accederá a la información cuando el servidor le notifique que ha habido una actualización.

Requisitos relacionados con la seguridad:

- Registrado de accesos de usuarios.
- Registrado de operaciones sobre todas las máquinas remotas.
- Implementación de medidas contra *spoofing*.
- Gestión de permisos en las operaciones siguiendo el modelo usuario — rol.

En el campo de la ciberseguridad los registros son una fuente de información muy valiosa. Por este motivo, el servidor deberá mantener un registro de todos los accesos y de las acciones que se realicen sobre cualquier máquina registrada en el sistema.

La interacción se realizará a través de una API usando peticiones HTTP. Es un requisito obligatorio, que el servidor tenga medidas que impidan acciones malintencionadas que puedan comprometer el sistema. Una de estas acciones seria que un usuario hiciera una petición con los datos de otro, suplantando así su identidad.

2.1.3. Interfaz

Requisitos generales:

- Diseño sencillo, intuitivo y compatible con todos los dispositivos.
- Deberá poder traducirse (castellano, catalán e inglés).
- Visualización de procesos en ejecución en tiempo real.
- Configuración de la captura del tráfico de red (filtros, tarjetas de red, etc.).
- Visualización del tráfico de red capturado.

- Visualización del rendimiento del equipo en tiempo real.
- Ejecución de comandos en máquina remota y visualización de su salida en pantalla.
- Gestión de los códigos de descarga y conexión.
- Gestión de las cuentas de usuario.
- Visualización de los registros de operaciones sobre máquina y acceso de usuarios.

Toda la información que se recupere por parte del cliente, se ha de poder visualizar en la interfaz web. En el caso de la consola remota, se deberá diseñar una consola web de comportamiento similar a la encontrada en cualquier máquina. Esta enviará los comandos al equipo remoto y mostrará su salida acorde con el formato original.

También debe permitir todas las labores de gestión relacionadas con los equipos registrados y usuarios en base de datos.

Requisitos sobre la seguridad:

- Operativa de ciertas funciones limitada según el rol del usuario.

2.2 Product backlog

A partir de los requisitos listados en el apartado anterior, se han concretado las User Stories que hará falta implementar.

Las User Stories son representaciones informales de los requisitos de un proyecto de software. Se usan en las metodologías de desarrollo agiles como Scrum por su capacidad para dar una respuesta rápida a los requisitos cambiantes.

A continuación, se expresarán las historias de usuario con los siguientes apartados: identificador, rol, acción, finalidad y criterio de aceptación. El identificador sirve para distinguir de forma inequívoca una historia de usuario. El rol, la acción y la finalidad constituyen la historia de usuario que se interpreta de la siguiente forma: Como [ROL] quiero [ACCIÓN] para [FINALIDAD]. Por ejemplo, como usuario quiero acceder a mi cuenta para ver mis datos. El criterio de aceptación indica el resultado que debe darse para considerar la historia como completada.

Previa enumeración de las historias de usuario, es necesario definir los roles que habrá en el sistema. Serán tres: usuario, técnico y administrador. El rol usuario corresponderá al cliente que solicita la intervención remota. Tanto el técnico como el administrador serán los roles que realicen la intervención remota, el técnico, pero, tendrá menos permisos que el administrador.

Hay que recalcar que los roles anteriores no son los mismos que los usados internamente en el sistema. Se usan con un propósito meramente informativo en el contexto de las historias de usuario.

User Story	ROL [Como]	ACCIÓN [Quiero]	FINALIDAD [Para]	Criterio de aceptación
US1	Usuario	Pedir ayuda remota	Mitigar un ciberataque en mi sistema	
US2	Usuario	Ejecutar el cliente	Recibir ayuda remota	El cliente muestra mensaje de conexión satisfactoria con la hora local del servidor
US3	Usuario	Cambiar el idioma del programa	Entender las instrucciones	Las instrucciones por pantalla del programa se muestran en el idioma seleccionado
US4	Técnico / Administrador	Iniciar sesión en el sistema	Acceder al panel de control	Inicia sesión con sus credenciales y es redirigido al panel de control del sistema
US5	Técnico / Administrador	Cerrar la sesión en el sistema	Salir del sistema	Cierra la sesión y deja de ver el panel de control
US6	Técnico / Administrador	Enviar un enlace de descarga	Registrar una nueva máquina en el sistema	Se crea un enlace de descarga y es recibido por el destinatario
US7	Técnico / Administrador	Modificar la información de una máquina	Almacenar toda la información sobre la investigación	Se modifica la información de la máquina y los nuevos cambios persisten
US8	Técnico / Administrador	Hacer un test de conexión al equipo	Comprobar el estado de la conexión	Se muestra un mensaje de conexión correcta o errónea dependiendo del caso
US9	Técnico / Administrador	Recuperar toda la información del sistema	Buscar vulnerabilidades específicas del sistema	Se visualizan las características de hardware y software del equipo
US10	Técnico / Administrador	Conectarme a una máquina	Realizar una intervención remota	Se muestra el panel de control de la máquina

US11	Técnico / Administrador	Recuperar todos los procesos en ejecución	Realizar un diagnóstico del ciberataque	Que se pueda visualizar un listado con todos los procesos en ejecución el sistema y sus características
US12	Técnico / Administrador	Capturar el tráfico de red	Identificar e investigar tráficos no deseados en la red del equipo	Se visualizan los paquetes capturados en la interfaz del sistema
US13	Técnico / Administrador	Identificar los dispositivos conectados	Buscar dispositivos sospechosos en el sistema comprometido	Se muestran los dispositivos conectados junto con sus características
US14	Técnico / Administrador	Monitorizar el rendimiento de la máquina remota	Identificar consumos excesivos de recursos	Se visualiza datos sobre el rendimiento del equipo y gráficas de evolución
US15	Técnico / Administrador	Ejecutar comandos remotamente	Ejecutar acciones correctivas y especificas en la máquina remota	El comando introducido produce la salida esperada
US16	Técnico / Administrador	Enviar un mensaje al cliente	Explicarle alguna cosa	El mensaje enviado se muestra por pantalla en la máquina remota
US17	Técnico / Administrador	Desconectar una máquina	Finalizar la conexión con ella	La máquina se muestra como desconectada y en la máquina remota el programa se cierra
US18	Técnico / Administrador	Ver los códigos de descarga y conexión existentes	Ver cuándo se han usado, activarlos o desactivarlos	Se muestra un listado con todos los códigos de descarga registrados en el sistema
US19	Técnico / Administrador	Ver las páginas de ayuda	Tener una guía visible en pantalla	La guía del sistema se muestra con el contenido contextual
US20	Técnico / Administrador	Ver los registros de acceso y operación de las máquinas	Ver las veces que se ha conectado un usuario	Los registros de acceso y operación de las máquinas se muestran en pantalla

US21	Administrador	Administrar las cuentas de usuario	Añadir, desactivar o cambiar las contraseñas de las cuentas	Las cuentas existentes y las diferentes opciones se muestran en pantalla

3 Planificación

El desarrollo del trabajo se ha llevado a cabo siguiendo la metodología de trabajo ágil SCRUM. El uso de esta metodología conlleva aplicar una serie de buenas prácticas enfocadas a obtener el mejor resultado posible de un proyecto y que se basan en la manera de trabajar de los equipos altamente productivos.

SCRUM es usado frecuentemente en proyectos complejos donde los requisitos no son fijos o no están muy definidos y donde se necesita obtener resultados pronto. Por este mismo motivo, en Scrum se realizan entregas parciales del producto de forma regular que además ayudan a resolver situaciones en que no se está entregando al cliente lo que quiere.

3.1 SCRUM

Los proyectos que usan la metodología Scrum se ejecutan en ciclos temporales, llamados *sprints*, de fija y corta duración. Estos sprints suelen tener una duración de dos semanas, aunque su duración puede variar entre unos pocos días hasta tres o cuatro semanas, dependiendo del rol del equipo dentro del equipo. Al acabar un sprint el cliente debe recibir un resultado completo, que será un incremento del producto final entregado en la iteración anterior.

Los equipos de Scrum generalmente están formados por entre tres a nueve miembros, más el Scrum Master y el Product Owner. Cada equipo está formado entonces por tres roles cuyas funciones son las siguientes:

- Product Owner (PO): Su principal responsabilidad es la de optimizar y maximizar el valor del producto. También es función suya la de ejercer de interlocutor entre los stakeholders del proyecto, el equipo de desarrollo y el cliente. Además, de este último, también deberá actuar como portavoz para manifestar las peticiones, requerimientos y feedback del cliente.

  Resumiendo, en cada sprint el Product Owner debe hacer una inversión en desarrollo que tendrá que generar valor. Para que el valor del producto se incremente, es necesario que el PO acuerde y marque de forma clara el objetivo del sprint en ejecución.
- Scrum Master (SC): Se encarga de gestionar el proceso de Scrum. Debe cerciorarse de que el proceso se ejecuta de forma correcta, así como facilitar la aplicación del mismo al equipo. Debe ayudar a eliminar de forma constante y progresiva los impedimentos que van surgiendo en el equipo y que pueden afectar a su capacidad de incrementar valor.

- Equipo de desarrollo: Son los profesionales que se encargan de llevar a cabo el producto. Se caracterizan por ser multifuncionales y por organizarse y gestionarse de forma autónoma, siempre con el objetivo de entregar un incremento del producto a la finalización del sprint. Al ser equipos multifuncionales, son capaces de generar el incremento de valor sin dependencias externas.

Cada sprint viene precedido por una fase de planificación. En esta fase se seleccionan en primer lugar los requisitos que se van a llevar a cabo y que el cliente proporciona. Tras esta fase, y una vez se han aclarado las dudas que hayan podido surgir durante la selección de requisitos, el equipo elabora una lista de tareas (producto backlog) a llevar a cabo en la iteración. Se realiza una estimación del esfuerzo necesario para completar cada ítem de la lista de forma conjunta y que será clave para establecer la prioridad de cada uno. Finalmente, el equipo se autoasigna tareas y se autoorganiza para empezar a trabajar. La finalidad de esta autonomía de la que gozan los miembros, es que compartan su conocimiento y trabajen conjuntamente para crear un equipo más resiliente y resolver los objetivos de especial complejidad.

Cada día, durante la ejecución de un sprint, el equipo realiza una reunión (daily scrum meeting) de entre cinco y 15 minutos. Suele hacerse de pie, para que su duración no se prolongue innecesariamente. El objeto de esta reunión es revisar el trabajo que se está realizando para poder analizar las posibles dependencias entre tareas, el progreso de cara al objetivo del sprint y los problemas que pueden evitar que se llegue al mismo. Cada miembro debe responder a las siguientes preguntas:

- ¿Qué tareas he hecho desde el último daily scrum?
- ¿Qué tareas voy a hacer de cara al próximo daily scrum?
- ¿Con qué obstáculos me he encontrado o creo que me voy a encontrar de cara a las próximas reuniones?

La idea es que en estas reuniones se hagan las adaptaciones necesarias que permitan cumplir con los objetivos previstos y eliminar las trabas con que los miembros del equipo se hayan podido encontrar.

El último día del sprint se realiza una revisión del mismo. En una primera parte, se hace una demostración al cliente. En esta el equipo presenta al cliente los requisitos completados, quien en función de los resultados y de los cambios que se hayan producido en el backlog, realizará las modificaciones necesarias en la planificación del proyecto para incorporar sus requerimientos.

En una segunda parte, el equipo hace una retrospectiva del sprint. Analiza cual ha sido su manera de trabajar y cuales han sido los problemas que les han impedido progresar de forma adecuada. El Scrum Master deberá, en la medida de lo posible, escalar o eliminar los problemas identificados y que queden más allá del equipo.

Las mejoras y reflexiones que surjan de esta última reunión, servirán para definir el rumbo del siguiente sprint. La inclusión de mejoras en el proceso durante el siguiente sprint constituye una adaptación del proyecto sobre el propio equipo.

3.2 Temporización

Aunque en este caso no se aplique Scrum al completo, dado que este trabajo no se ha hecho en equipo, las ventajas de esta metodología se han visto igualmente reflejadas. Principalmente, la obtención resultados, los periodos de prueba cortos y la mejor tasa de cumplimiento.

Otro punto clave en la elección de esta metodología ha sido su flexibilidad, que permite modificar el hilo conductor de los hechos en cualquier instante logrando así la resolución de conflictos sobre la marcha y casi de forma inapreciable, además de evitar el perfeccionismo innecesario.

En este caso, el proyecto se ha divido en dos grandes bloques: uno dedicado al código y otro a la memoria. El considerarlos como proyectos separados, ha permitido usar duraciones distintas para los sprints de cada uno. La duración de los sprints ha sido de una y dos semanas, para el código y la memoria respectivamente.

Para poder distribuir las historias de usuario entre todos los sprints, primero se estimó de forma aproximada las horas que su implementación requeriría. Tras la estimación, las historias quedaron de la siguiente manera:

User Story	ACCIÓN [Quiero]	Estimación en horas
US1	Pedir ayuda remota	40
US2	Ejecutar el cliente	10
US3	Cambiar el idioma del programa	22
US4	Iniciar sesión en el sistema	10
US5	Cerrar la sesión en el sistema	1
US6	Enviar un enlace de descarga	17
US7	Modificar la información de una máquina	17
US8	Hacer un test de conexión al equipo	35
US9	Recuperar toda la información del sistema	26
US10	Conectarme a una máquina	19
US11	Recuperar todos los procesos en ejecución	20
US12	Capturar el tráfico de red	100
US13	Identificar los dispositivos conectados	20

US14	Monitorizar el rendimiento de la máquina remota	35
US15	Ejecutar comandos remotamente	120
US16	Enviar un mensaje al cliente	20
US17	Desconectar una máquina	1
US18	Ver los códigos de descarga y conexión existentes	15
US19	Ver las páginas de ayuda	22
US20	Ver los registros de acceso y operación de las máquinas	22
US21	Administrar las cuentas de usuario	18

Tras esto, las historias de usuario se distribuyeron a lo largo de los sprints. Tanto a mitad del desarrollo como al final, se dedicaron un par de iteraciones exclusivamente para el testeo exhaustivo y la resolución debugs por lo que no se completó ninguna historia de usuario. El diagrama de Gantt a continuación ilustra la distribución final prevista para los sprints:

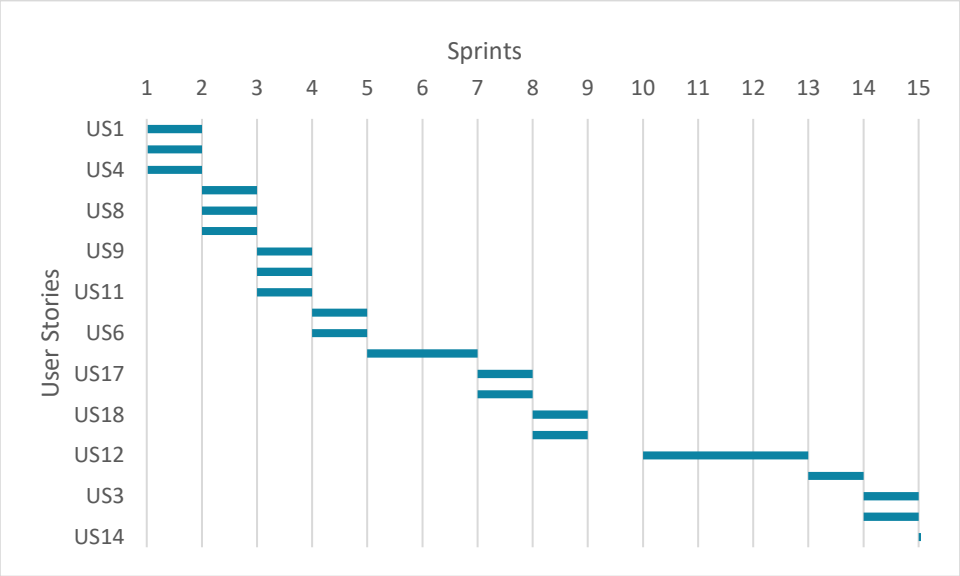


Ilustración 1: Diagrama de Gantt de las historias de usuario por sprint. (Fuente: Elaboración propia)

Finalmente, la siguiente tabla muestra las fechas de comienzo y finalización de los sprints, así como la duración y objetivo principal de los mismos.

Sprint	Objetivo	Inicio	Duración (semanas)	Finalización
1	Configuración de proyectos y entornos.	11/01/2021	1	17/01/2021
2	Comunicación frontend, backend y cliente.	18/01/2021	1	24/01/2021
3	Monitorización de procesos.	25/01/2021	1	31/01/2021

4	Descarga del cliente y autorización de 2 factores.	01/02/2021	1	07/02/2021
5	Shell remota en Windows (backend).	08/02/2021	1	14/02/2021
6	Diseño consola remota en interfaz.	15/02/2021	1	21/02/2021
7	Gestión de registros (logs) de todas las operaciones.	22/02/2021	1	28/02/2021
8	Gestión de cuentas, enlaces de descarga y registros en frontend.	01/03/2021	1	07/03/2021
9	Solución de bugs (sprints 5 a 7).	08/03/2021	1	14/03/2021
10	Diseño interfaz captura tráfico de red.	15/03/2021	1	21/03/2021
11	Captura de tráfico en cliente.	22/03/2021	1	28/03/2021
12	Procesado de tráfico de red y muestreo en interfaz.	29/03/2021	1	04/04/2021
13	Recuperación de dispositivos.	05/04/2021	1	11/04/2021
14	Traducciones en todo el sistema.	12/04/2021	1	18/04/2021
15	Monitorización de rendimiento.	19/04/2021	1	25/04/2021
16	Pruebas y solución de bugs.	26/04/2021	1	02/05/2021
17	Memoria y documentación.	03/05/2021	2	16/05/2021
18	Memoria y documentación.	17/05/2021	2	30/05/2021
19	Memoria y documentación.	31/05/2021	2	13/05/2021

4 Tecnologías

4.1 Python

Pese a no ser una tecnología de por sí y, por lo tanto, estar mencionado en la sección equivocada, resultaría absurdo enumerar las tecnologías integradas sin haber explicado antes el lenguaje en que se usan. Por este motivo se ha incluido en esta sección.

Python es un lenguaje interpretado de alto nivel y propósito general. Las instrucciones se ejecutan directamente, sin requerir que hayan sido previamente compiladas en lenguaje máquina. Además, proporciona un fuerte nivel de abstracción en lo que a los detalles del equipo se refiere. Todo ello sin restringir sus campos de aplicación.

Por su sintaxis simplificada y su énfasis en el lenguaje natural, su popularidad ha aumentado en los últimos años. Gracias a esto, se dispone de una gran variedad de librerías para casi cualquier propósito. Esto último, sumado a su portabilidad, hace que sea el lenguaje perfecto para el desarrollo del sistema objeto de este trabajo, ya que la portabilidad a múltiples plataformas viene de “serie”.

Como todo, Python también tiene sus desventajas. Tiene una velocidad de ejecución lenta (en comparación con otros lenguajes), usa una gran cantidad de memoria, el código compilado pesa más de lo que debería y es propenso a los errores de ejecución.

Este último punto se soluciona fácilmente con un testeo exhaustivo y completo del código. Sin embargo, el resto, no tienen solución y dependen de la optimización y buenas prácticas del programador para ser más o menos justificables.

Por la naturaleza de la implementación, que será explicada más adelante, ni la velocidad ni el uso de memoria o el tamaño final son elementos críticos en el sistema. Esta decisión, sumada a la portabilidad y al gran soporte de librerías, son los motivos que justifican la elección de Python como lenguaje.

4.2 Flask

El servidor, estructura esencial en este trabajo, ha sido implementado con Flask un framework que enfocado al desarrollo de aplicaciones web.

Flask es un micro-framework escrito en Python. Se caracteriza por sus pocas dependencias con librerías externas. Esto, por un lado, es perfecto ya que lo convierte en un framework web ligero y fácil de actualizar. Por otro lado, al no tener librerías integradas, resulta en más trabajo para el programador

ya sea por desarrollar las características manualmente o por tener que construir la lista de dependencias manualmente.

También, aunque no por defecto, Flask tiene soporte para el desarrollo de APIs, que para el sistema que se está desarrollado es un elemento clave. Tanto el cliente como la interfaz se comunicarán con el servidor mediante una API web, por lo que este soporte constituye uno de los motivos más importantes que justifican la elección de Flask como backend.

Otras de sus características son que tiene un despliegue muy sencillo, soporta las peticiones RESTful, dispone de una documentación amplia y una alta compatibilidad con muchas tecnologías.

En cuanto al largo plazo, las aplicaciones escritas de Flask se pueden desarrollar, mantener y escalar fácilmente pese a basarse en un framework minimalista.

4.3 Vue.js

Vue.js es un framework de frontend de código abierto y escrito en JavaScript para el diseño de interfaces usuario y aplicaciones web. A diferencia de otros frameworks monolíticos, este está pensado para ser usado de forma incremental.

Su librería central está únicamente centrada en la capa de visualización por lo que la integración con otras librerías o proyectos ya existentes se consigue sin apenas complicaciones.

Es muy ligero, por lo que no solo se consigue una alta velocidad de descarga e instalación, sino que además se puede conseguir un fuerte impacto positivo en relación al SEO y la experiencia de usuario. Combinado con su ligereza, Vue.js destaca, entre otros frameworks populares como Angular o React, por su rendimiento.

Internamente cada pieza de la aplicación web desarrollada se estructura en componentes, que representan elementos encapsulados de la interfaz. Esta división en componentes de la aplicación es parte de una aproximación llamada *Component Based Architecture* que también se aplica en Angular y React. La estructura en componentes permite reusar los componentes con mucha facilidad e integrar código HTML, CSS y JavaScript en un único fichero, facilitando así su lectura y mantenimiento.

Sus desarrolladores también mantienen una documentación muy concisa, por lo que su curva de aprendizaje se aplana. La comunidad alrededor de este framework es igualmente numerosa, por lo que existen infinidad de librerías, herramientas y soluciones como para diseñar cualquier cosa.

Al ser un framework relativamente nuevo, la cantidad de plugins y componentes de que Vue.js dispone no hace sombra a los que tanto Angular como React soportan, por ejemplo. Su rápida evolución

complica también el seguimiento de los desarrolladores, que frecuentemente están obligados a reaprender el framework para mantenerse al día.

Sin embargo, estos motivos no han sido pega alguna al escogerlo para el desarrollo de la interfaz. Su fácil portabilidad entre diferentes dispositivos y características internas como el renderizado condicional de elementos o los bucles, hacen que sea un framework ideal para el desarrollo de páginas web dinámicas y aplicaciones web adaptables y espectaculares.

El motivo por el que se ha usado este framework y no otro para la interfaz ha sido meramente personal. Tras haberlo usado previamente otras aplicaciones web y haber obtenido muy buenos resultados, lo he preferido frente a otros que pudieran ser más capaces.

4.4 Redis

Uno de los principales problemas que se han encontrado al desarrollar el sistema, ha sido el del almacenamiento de datos. Al tratarse de un sistema distribuido que usa una conexión inversa (explayado en profundidad más adelante) los datos no se pueden transferir directamente del cliente a la interfaz. Primero porque el volumen de datos puede ser elevado como para tener que realizar constantemente escrituras y lecturas en base de datos. Y segundo porque la gran parte de datos que envía el cliente tienen carácter temporal y su almacenaje en base de datos no tiene sentido.

Esta problemática se ha solucionado con Redis. Se trata de una base de datos en memoria. Su funcionamiento se basa en las tablas hash, en que se almacena un valor en base a otro que se usa como clave o identificador.

Redis tiene soporte para una amplia variedad de estructuras de datos abstractas como cadenas de caracteres, listas, sets, streams o índices espaciales. Aunque no está pensado para mantener los datos almacenados por mucho tiempo, las bases de datos de Redis pueden ser configuradas con persistencia.

Al almacenar datos en formato clave – valor, es sencillo guardar cualquier dato y de cualquier tamaño en Redis para luego recuperarlo con la misma dificultad. Por ejemplo, unos datos estructurados en JSON pueden guardarse tal cual en Redis convirtiéndolos en una String. Luego, en el momento en que se recuperan, pueden ser parseados como JSON de nuevo.

Esta flexibilidad evita el tener que diseñar una base de datos con una estructura compleja, que sea capaz de almacenar todos los tipos de datos que pueden obtenerse y sus variaciones. Flexibilidad que además evita la innumerable cantidad de operaciones de lectura / escritura que se realizarían en una

base de datos convencional. Todo esto, sin mencionar el coste de implementación y mantenimiento, en sistemas de mayor complejidad o que requieran de un escalado.

Entonces, en el contexto del sistema, Redis se usa como una memoria caché distribuida. El cliente envía los datos al servidor, quien los almacena en la "caché" a la espera de que sean solicitados. Cuando son solicitados, el servidor los recupera con la clave con que han sido almacenados.

Dado que el tiempo de uso de los datos es mínimo debido a la frecuente actualización de los mismos, los procesos por ejemplo se recopilan cada segundo, la persistencia de los datos no está configurada en el sistema.

4.5 Pusher

Uno de los problemas encontrados en la comunicación servidor – interfaz es el sincronismo. No es eficiente aplicar una conexión al frontend de una aplicación web, además si el consumo de memoria es excesivo el navegador puede hasta bloquearse.

La solución a este problema implementada en el sistema es Pusher. Se trata de una API que simplifica la transferencia de datos y funciones en tiempo real para aplicaciones web y móviles.

Implementa un comportamiento similar al que se consigue con el patrón de diseño *Observer*. Este patrón define una dependencia del tipo uno a muchos entre objetos. Cuando un objeto cambia, notifica a todos los que dependen de él. Para que esto sea posible, se mantiene una lista con todos los objetos dependientes.

Pusher funciona de forma similar, se sitúa internamente como una capa intermedia entre los clientes y servidores que lo usan. Cuando se produce un cambio en el servidor, es posible notificar en tiempo real a todos los clientes o a un grupo de ellos. Esto es posible gracias a que Pusher mantiene una conexión persistente con cada uno de los clientes en todo momento, salvo cuando son desconectados, claro.

En el contexto del trabajo, esta tecnología se usa para notificar a la interfaz cuando el cliente ha enviado nuevos datos al servidor.

4.6 Heroku

Los contenedores son tecnologías que aíslan las aplicaciones junto con todo su entorno de ejecución y dependencias, abstrayendo a la aplicación del entorno en que se ejecuta. Esta abstracción hace que

el despliegue de las mismas sea mucho más sencillo y que factores como las características del entorno no afecten a la aplicación.

Heroku es un contenedor de aplicaciones en la nube y que usa los servicios en la nube de Amazon Web Services. Ahorra la necesidad de disponer de servidores propios para tener el contenedor y al estar centrado en el desarrollador, que este tenga que dedicar tiempo a la administración del mismo.

También es fácil de escalar, usar y empezar, por lo que es ideal para los proyectos pequeños y con grades miras. Heroku también incluye un conjunto de herramientas que permiten integrar bases de datos o aplicaciones de terceros como Redis To Go, Sendgrid o Bucketeer.

4.7 Sendgrid

Para el envío de correos electrónicos en el sistema se ha usado Sendgrid. Se trata de una infraestructura cloud que permite a los desarrolladores enviar correos electrónicos de forma muy sencilla, escalable y en tiempo real.

En los despliegues de las primeras iteraciones del sistema, no se usaba está tecnología para enviar los correos electrónicos. Al aumentar la cantidad de mensajes enviados, Heroku empezó a mandar los mensajes con mucho retraso, a veces incluso ni los mandaba. Por este motivo se integró Sendgrid en el sistema.

4.8 Github

Se trata de una plataforma de desarrollo colaborativa dedicada al alojamiento de proyectos usando el control de versiones Git. Proporciona varias funciones útiles como el seguimiento de errores, gestión de flujos de trabajo e integración continua.

Durante el desarrollo de un proyecto, se realizan innumerables cambios en el código fuente de los mismos al mismo tiempo que se despliegan o se entregan a los clientes. El control de versiones de Git, lleva un registro de todas las modificaciones que se han hecho sobre el código fuente. Cada registro se genera mediante un commit. Cada commit tiene un identificador único que identifica un cambio o un conjunto de ellos. Al mantener un registro de todos los commits, se proporciona a los desarrolladores la posibilidad de volver a ver el código fuente anterior a un determinado commit o incluso restaurarlo.

Github, basado en Git, incorpora algunas funcionalidades para el trabajo colaborativo. Una de estas funciones son las ramas. Esencialmente son bifurcaciones del código fuente principal, con

modificaciones respecto al mismo. Con esta función es posible el desarrollo por parte de múltiples personas al mismo tiempo, ya que las ramas son independientes entre sí.

4.9 Travis CI

En los proyectos en que se aplican las metodologías agile, la rapidez no es exclusiva del proceso de desarrollo, sino que también se extiende a otros procesos como el despliegue o el de QA. Es por este motivo que no solo se anima a los equipos a automatizarlos, sino que además se recomienda.

Travis es un servicio en la nube de integración continua. Puede configurarse para implantar cualquier comportamiento deseado en la cadena de producción del software.

Por ejemplo, que tras hacer una modificación al código se ejecuten unos test automáticos y se despliegue en el entorno de producción automática. Las posibilidades de configuración son amplias y con el gran soporte de lenguajes que dispone, no existen prácticamente limitaciones.

5 Diseño

En esta sección se detallan temas sobre el diseño interno del sistema en lo referente a la arquitectura, componentes, modelos de base de datos y aspectos sobre la seguridad.

5.1 Arquitectura del sistema

El modelo aplicado es el de cliente – servidor. En este tipo de arquitectura particiona las tareas o cargas de trabajo entre los proveedores de un recurso o servicio, llamados servidores, y los que demandan el servicio, los clientes. Este último, generalmente, no comparte sus recursos con otros clientes.

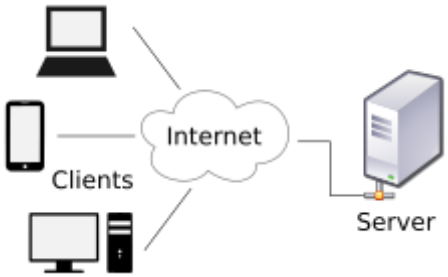


Ilustración 2: Diagrama genérico de una arquitectura cliente - servidor. (Fuente: Wikipedia)

Si un equipo es un cliente, un servidor o ambos, es determinado por la naturaleza de la aplicación que requiere las funciones de servicio.

La comunicación entre cliente y servidor se realiza mediante un patrón de mensajes con formato petición – respuesta. El cliente envía una petición al servidor, que devolverá una respuesta. Para que esto sea posible, debe existir un protocolo de comunicación conocido y usado por ambas partes, y que defina el lenguaje, sintaxis y orden de las comunicaciones.

Todos los protocolos de comunicación operan en la capa de aplicación. Es posible que un servidor implemente una interfaz de programación de aplicaciones, o API. Una API es una capa de abstracción en el acceso a un servicio. Al abstraer el acceso y restringir las comunicaciones a un cierto formato, se simplifica el intercambio de información entre plataformas distintas.

En el contexto del sistema, tanto el cliente como la interfaz toman el rol de clientes contra el servidor que implementa. El servidor responde a las peticiones de cliente e interfaz, devolviendo datos o procesándolos según sea necesario, siempre mediante una API.

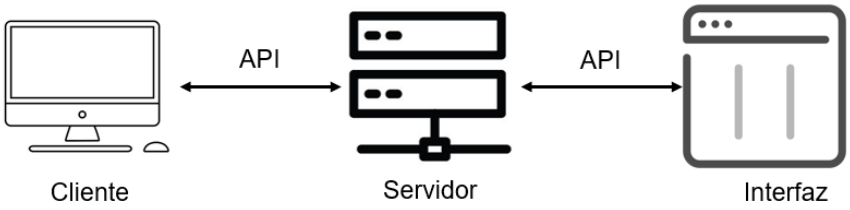


Ilustración 3: Diagrama de la arquitectura del sistema. (Fuente: Elaboración propia)

5.2 Componentes

5.2.1. Cliente

El cliente está diseñado como un elemento pasivo. No realiza ninguna acción por cuenta propia. Constantemente monitoriza el servidor mediante peticiones a las cuales el servidor responde con un código de operación que el cliente debe ejecutar.

No mantiene ningún estado. En todo momento monitoriza unos parámetros del servidor que indican la operación a realizar en cada momento. Por cada parámetro, el cliente crea un hilo de ejecución separado e independiente. De esta forma es posible ejecutar múltiples operaciones de forma simultánea sin problemas de solapamiento.

La función de monitorización de acciones es la misma para todos los hilos, lo único que difiere es el parámetro de operación que se controla. Su implementación es similar a la de un bloque *switch- case*, donde en base a un valor se ejecutan unas acciones u otras.

En concreto el sistema mantiene cinco códigos de operación internos, llamados:

- **opcode0**
- **opcode1**
- **opcode2**
- **opcode3**
- **opcode4**

De estos, opcode2, tiene un funcionamiento diferente al resto ya que tiene la capacidad de terminar la ejecución del resto. Los flujos de ejecución relativos a la monitorización de los parámetros de operación se explicarán en la sección de implementación.

La compatibilidad entre diferentes plataformas es prácticamente nativa en Python. No obstante, algunas librerías usadas en el sistema están enfocadas hacia ciertas plataformas en concreto, por lo que sus funciones son limitadas o no funcionan para otras.

El caso anterior se da en el sistema con las funciones de ejecución de comandos remota y en la recuperación de dispositivos. La estructura interna en ambos casos es similar ya que se requiere de librerías específicas para cada plataforma.

En el caso del listado de dispositivos la clase *DeviceLister* (DeviceLister.py) se encarga de procesar el código de operación interno y listar los dispositivos conectados al equipo. En el caso de un sistema Windows usará el método de la clase *WindowsDeviceLister*. Para Linux usará la clase *LinuxDeviceLister.* En ambos casos se han usado métodos de clase, puesto que los métodos no son específicos de una instancia en concreto. El diagrama de clases resultante es el siguiente:

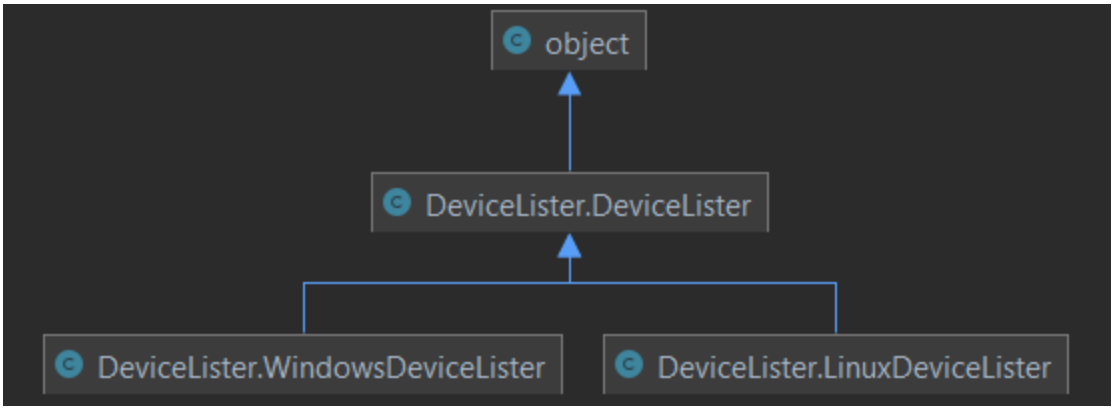


Ilustración 4: Diagrama de las clases encargadas de recuperar los dispositvos conectados. (Fuente: squid-remote)

La ejecución de comandos remotos, internamente, se divide en dos clases: una de gestión y otra de ejecución. La clase de gestión, *ShellManager* (Console.py), se encarga de crear, detener y reiniciar las diferentes instancias de Shell remota que se ejecutan en el cliente. Por otra parte, son las instancias que heredan de la clase abstracta *ShellInstance* (Console.py) las que realmente ejecutan los comandos remotos.

Mientras que la clase de gestión es válida para cualquier plataforma, las instancias que esta crea no lo son. Por este motivo existen las clases *LinuxShellInstance* y *WindowsShellInstance*, que implementan los métodos de la clase madre acorde con la plataforma a la que sirven. En este caso se ha optado por este diseño ya que las funciones que estas clases implementan si afectan a instancias específicas. El diagrama de clases de las clases de gestión y ejecución de comandos es el siguiente:

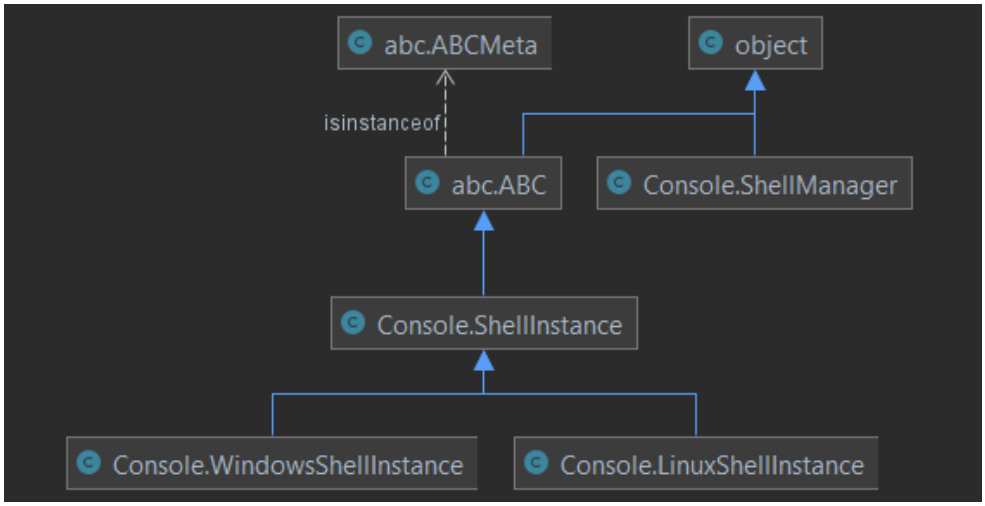


Ilustración 5: Diagrama de clases de los componentes de la shell remota. (Fuente: squid-remote)

5.2.2. Servidor

El backend del sistema se divide en tres módulos principales: recursos, modelos y utilidades. El módulo de recursos contiene todas las clases relacionadas con las operaciones sobre los objetos de base de datos, así como el procesado de datos y la notificación a servicios externos. Todas las clases de este módulo extienden de la clase *Resource* de Flask, que permite implementar una API modular.

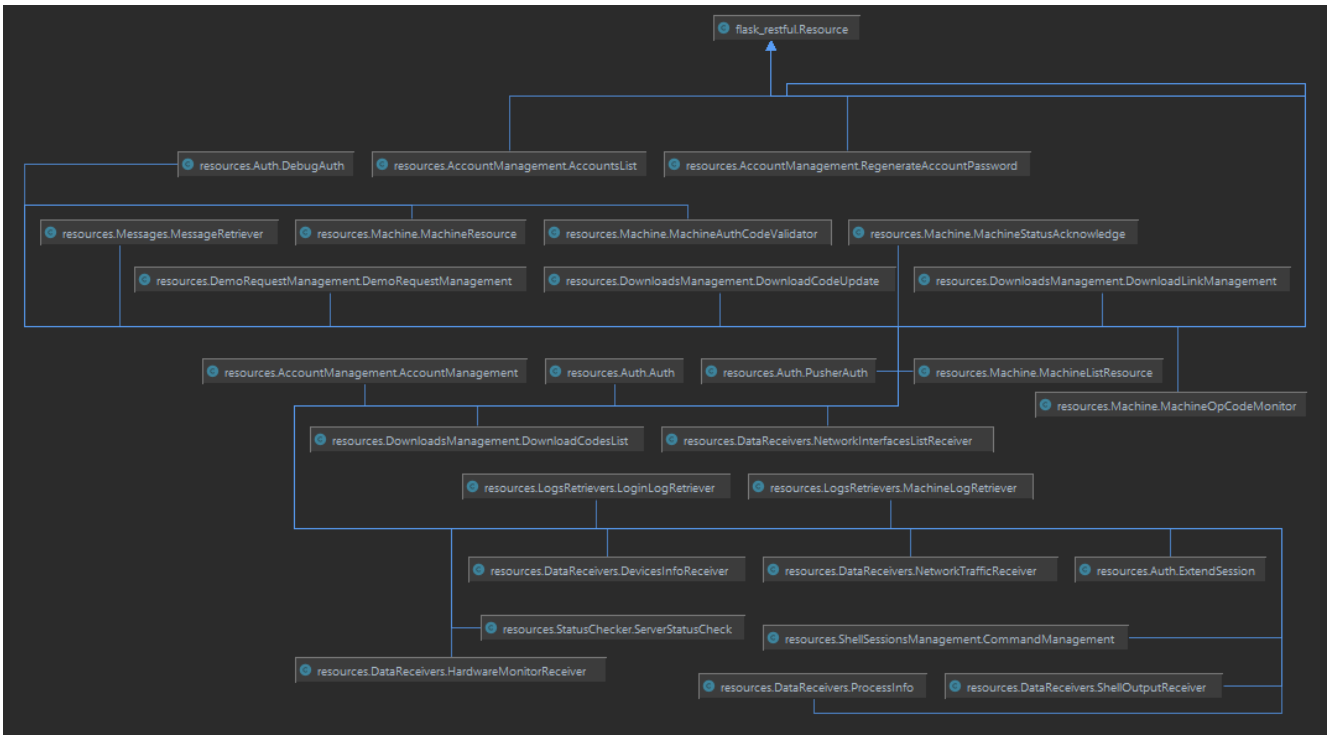


Ilustración 6: Diagrama de todos los recursos del sistema. (Fuente: squid-remote)

La estructura de las tablas de la base de datos, así como los métodos de cada clase e instancia se encuentran en el módulo de modelos. Todos extienden de la clase *Model* de SQLAlchemy, que se usa para la interacción con la base de datos.

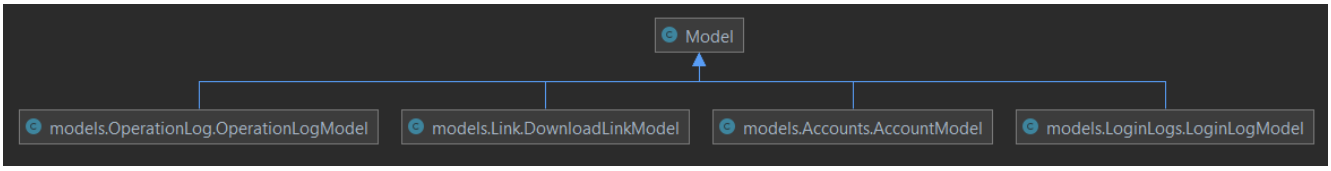


Ilustración 7: Diagrama de clases de los modelos de la base de datos. (Fuente: squid-remote)

Por último, en el módulo de utilidades, se encuentran la clase encargada del envío de correos electrónicos, así como las plantillas que esta usa.

Otra clase aparte, y esencial para la carga la configuración individual de cada entorno, es la clase *Config* (Config.py). De esta clase extienden otras tres que se usan para cambiar la base de datos usada en función del entorno de despliegue, son, *DevelopmentConfig* y *ProductionConfig*. El resto de los ficheros se encuentran en el directorio raíz y no tiene ninguna dependencia especial.

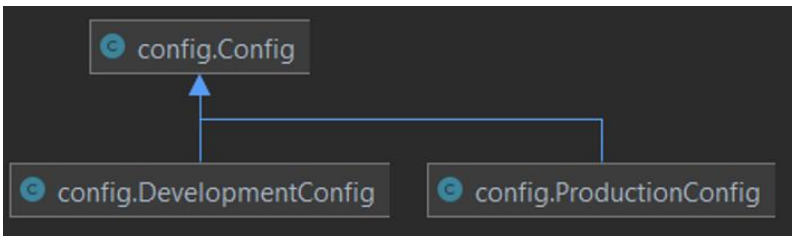


Ilustración 8: Diagrama de clases de las clases de configuración. (Fuente: squid-remote)

5.2.3. Interfaz

El siguiente prototipo de interfaz es el resultado del primer diseño del frontend, generado a partir de las historias de usuario extraídas al principio del documento. Junto con las historias de usuario, los requisitos referentes a la implementación también se han tenido en cuenta para la elaboración de este boceto. Para desarrollar el prototipo se ha usado la herramienta de prototipado Balsamiq.

Principalmente, lo que se quería plasmar en el boceto es una interfaz sencilla, clara y útil, pero que a la vez es estéticamente agradable. Para dar sensación de continuidad, el diseño de las diferentes secciones mantiene un estilo en común, manteniendo siempre en la parte superior una barra de navegación con el nombre del sistema.

Para hacer la experiencia de usuario más atractiva, la interfaz se ha desarrollado alrededor de un nombre y logotipo que a su vez dan nombre al equipo, squid-remote. Pese a no ser un aspecto crítico en el sistema, el hecho de que el sistema tenga una marca hace más sencillo darle un aire más comercial y profesional al diseño.

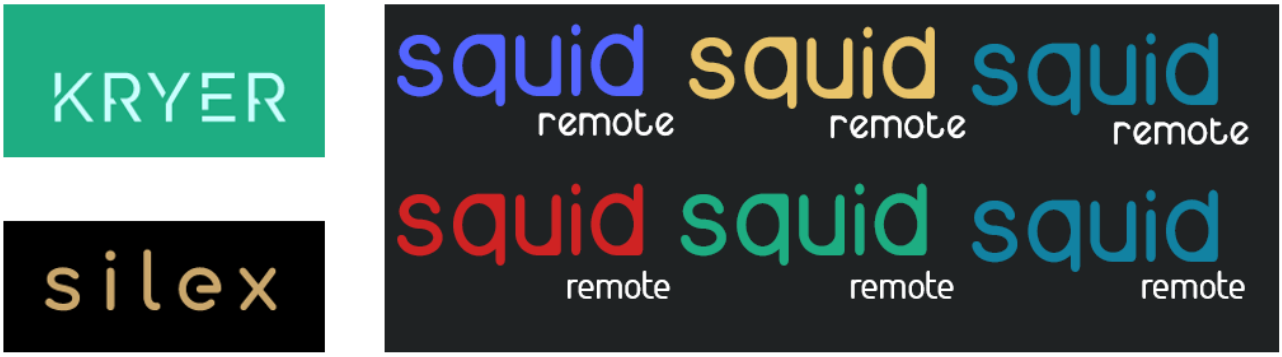


Ilustración 9: Ideas de logotipos en las fases tempranas del proyecto. (Fuente: Elaboración propia)

El logotipo del sistema, resultado de varias modificaciones y consultas y que es elemento transversal en el diseño del sistema, es el siguiente:

Ilustración 10: Logotipo final usado por el sistema. (Fuente: squid-remote)

La pantalla principal del sistema es la de acceso al mismo. El usuario, técnico o administrador, accedería mediante su nombre de usuario y contraseña. No existe un botón de recuperación de contraseña o de registro ya que, al ser un sistema cerrado, la gestión de las cuentas queda delegada en los administradores del sistema.

Ilustración 11: Prototipo de la pantalla de acceso al sistema. (Fuente: Elaboración propia)

Tras acceder al sistema, el usuario es redirigido a la página principal. Esta sirve como puente entre el resto de páginas. De entrada, se le muestran, en formato de tarjeta, aquellas máquinas que han estado registradas en el sistema junto con toda su información. Desde la barra de navegación es posible enviar un enlace de descarga a un equipo nuevo, acceder a la sección de administración y cerrar sesión.

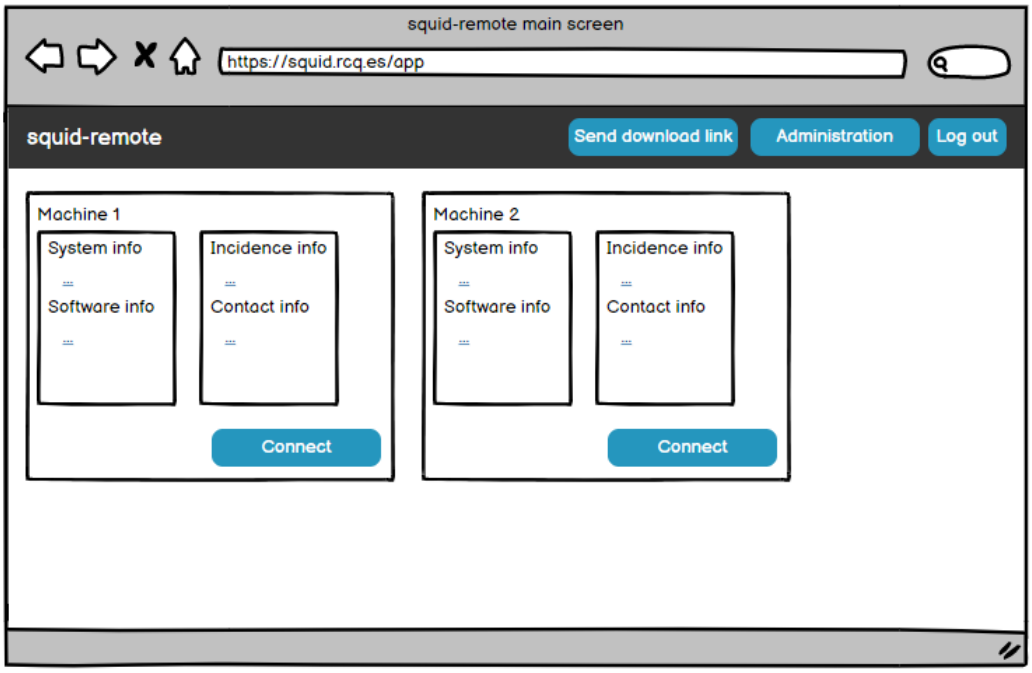


Ilustración 12: Prototipo de la pantalla principal del sistema. (Fuente: Elaboración propia)

La sección de administración permite visualizar los enlaces de descarga, los registros de acceso al sistema, los registros de las máquinas y las cuentas existentes en el sistema. En la imagen a continuación únicamente se muestran las cuentas existentes en el sistema puesto que el resto de funcionalidades de esta pantalla se rigen por el mismo formato que el de las cuentas de usuario.

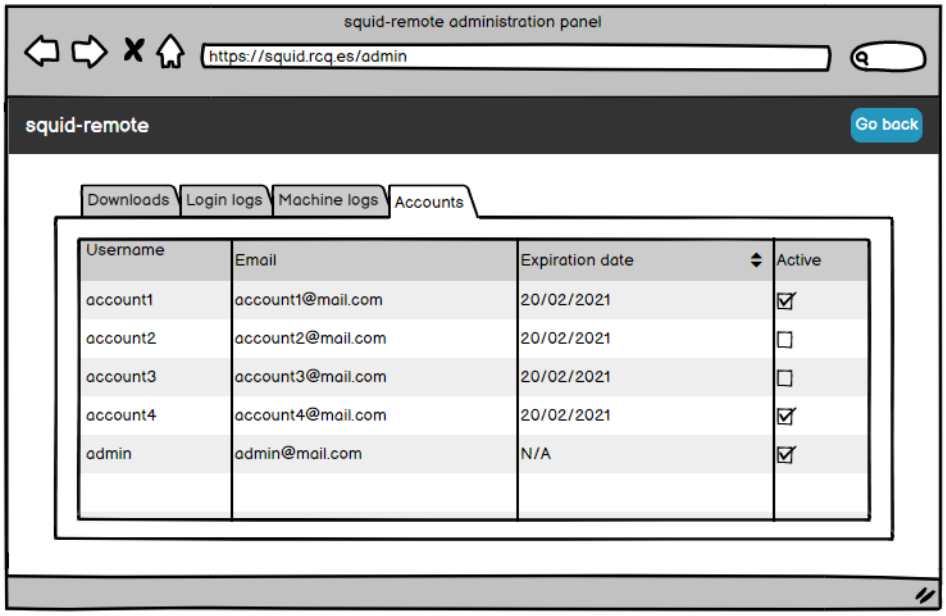


Ilustración 13: Prototipo de la sección de administración del sistema. (Fuente: Elaboración propia)

El resto de la interfaz está relacionado con el monitoreo del equipo remoto. El boceto siguiente representa la vista que un usuario tendría de los procesos del sistema. Cada uno de ellos se muestra con formato de fila en la tabla de procesos. De cada uno se muestra su nombre, identificador único, consumo de memoria RAM y CPU, así como la cantidad de hilos de ejecución que está usando. Hay que recalcar que este boceto se realizó en las etapas tempranas del proyecto, y que, a lo largo del desarrollo del mismo, la información mostrada en esta sección se ha ampliado.

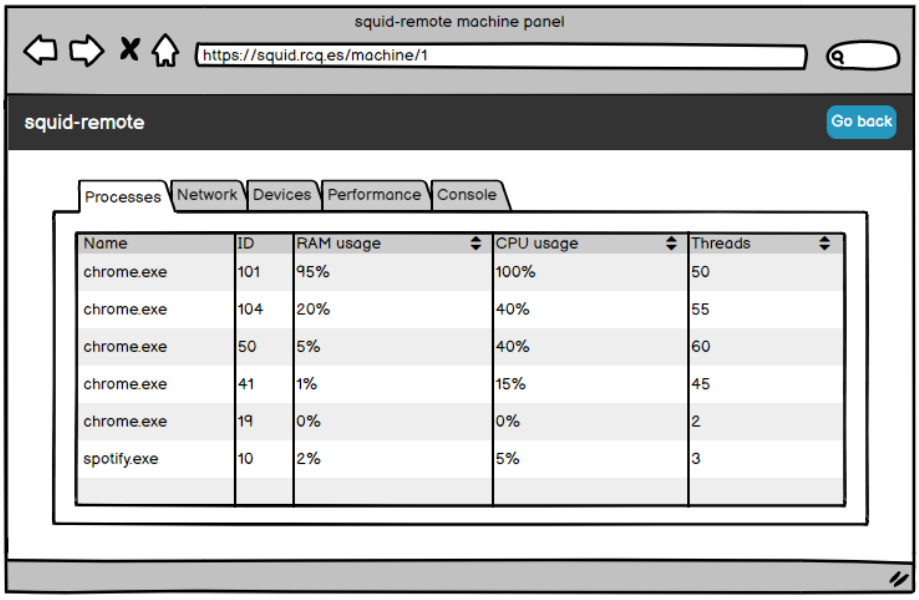


Ilustración 14: Prototipo de la vista de procesos en ejecución. (Fuente: Elaboración propia)

Otra función que debe implementar el sistema es la captura del tráfico de la red en que el equipo está conectado. De cada paquete capturado se mostrará información básica como el origen y el destino del mismo, así como otros datos relevantes sobre el mismo. Las capas del paquete y sus características serán mostradas en la segunda tabla. El inicio y la detención de la captura se ordenarán mediante el botón llamado "*Capture traffic*".

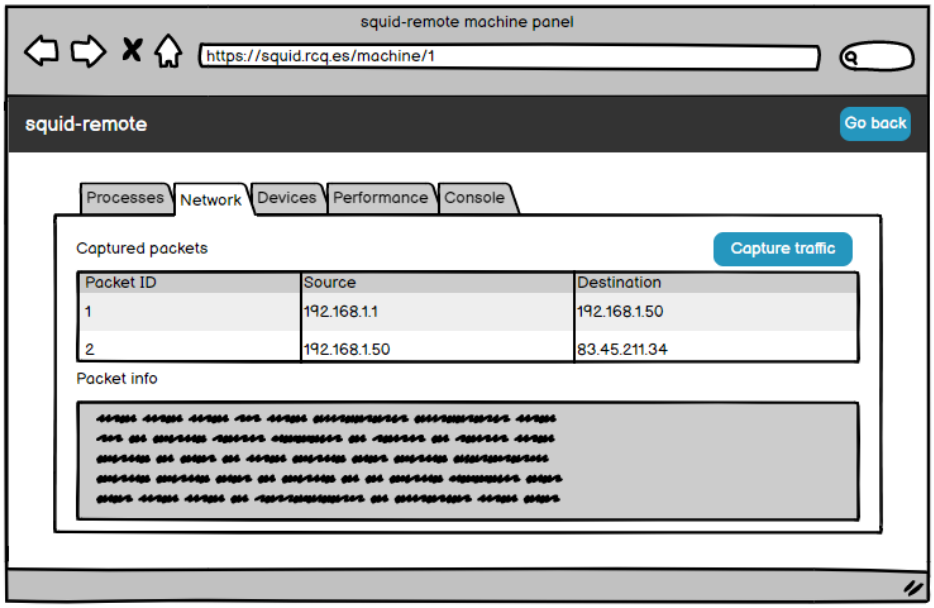


Ilustración 15: Prototipo de la sección de captura de tráfico. (Fuente: Elaboración propia)

El listado de dispositivos conectados tiene un diseño similar al de la página principal. Cada dispositivo se muestra en formato de tarjeta junto con sus características. La orden de enumerar los artefactos conectados se dará con el botón de "*Search devices*". Es importante que exista este botón y que no sea automática la recuperación de dispositivos, ya que durante la conexión un dispositivo puede desconectarse, modificando la lista visualizada en pantalla. De este modo, el usuario puede en cualquier momento refrescar el listado.

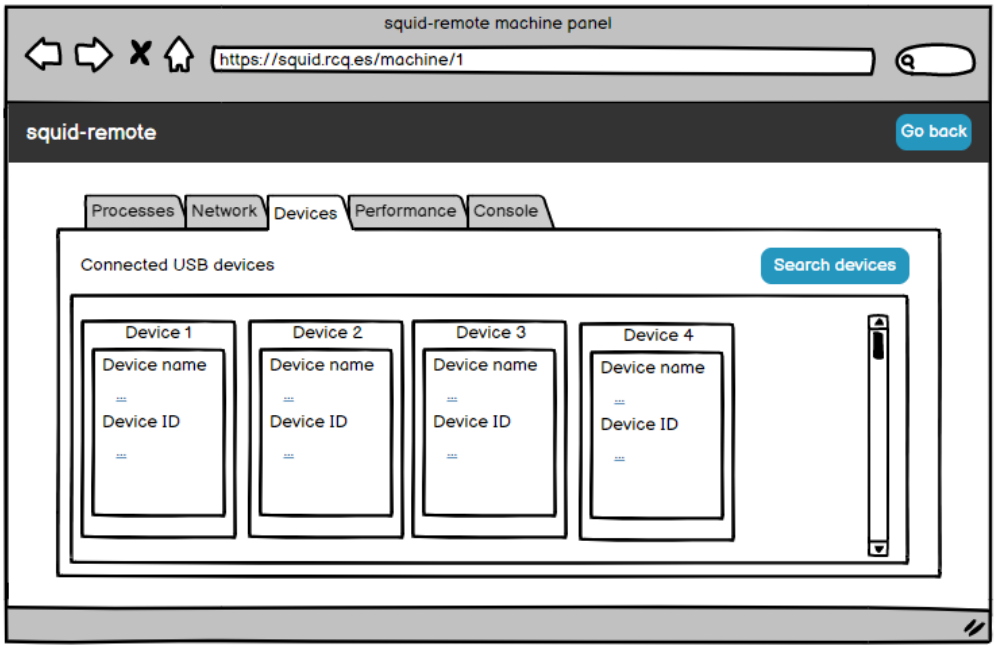


Ilustración 16: Prototipo de la sección de captura de dispositivos conectados. (Fuente: Elaboración propia)

La sección de monitorización del rendimiento tendrá varias gráficas e indicadores para mostrar el estado del equipo en tiempo real, así como la disponibilidad de sus recursos.

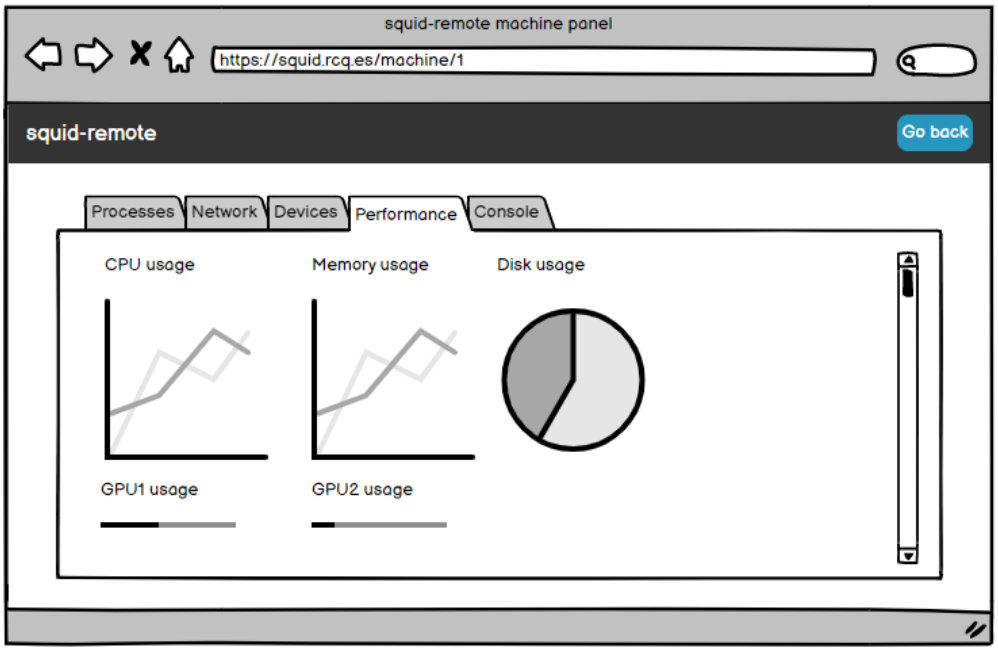


Ilustración 17: Prototipo de la sección de monitorización de rendimiento del equipo. (Fuente: Elaboración propia)

Por último, la consola remota. El objetivo es que sea lo más similar a una consola de un equipo real. Para conseguir esto su estilo se inspira en el diseño de la consola de Windows. Esta únicamente tiene dos colores, blanco para el texto y negro para el fondo, y aumenta en scroll por la parte inferior. Será posible mantener varias consolas remotas a la vez. El mecanismo para alternar entre una consola u otra es por ventanas, tal y como se muestra en la próxima figura.

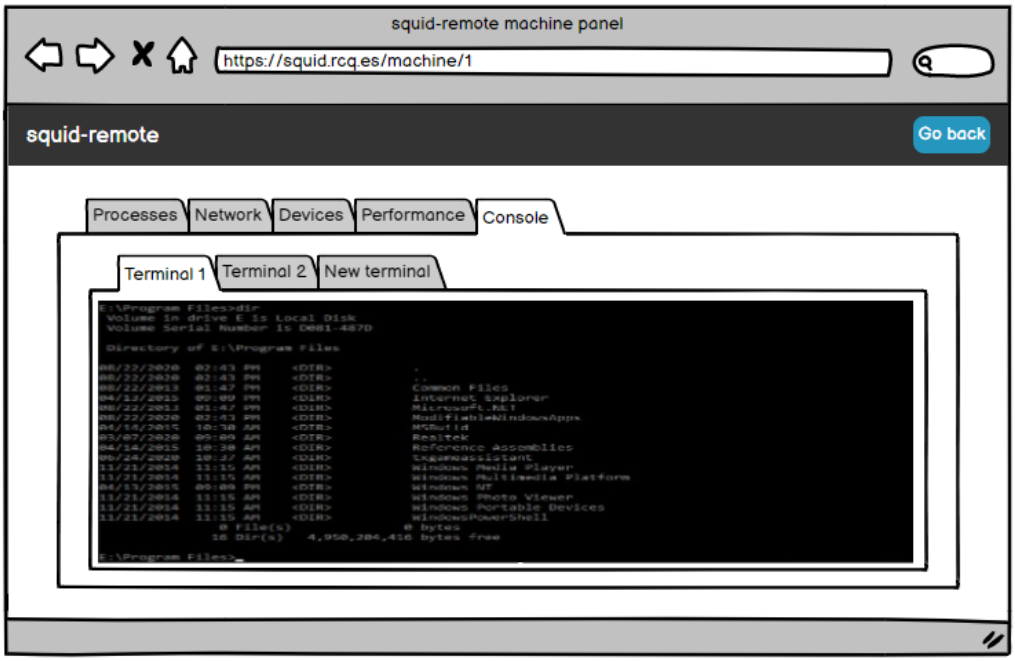


Ilustración 18: Prototipo de la sección de consola remota. (Fuente: Elaboración propia)

5.3 Diseño de base de datos

La base de datos tiene un diseño muy simple que se estructura en cinco tablas. Son las siguientes:

- Accounts: Modelo para las cuentas de usuario del sistema. Tiene las siguientes columnas:
  - **ID:** Identificador único de la cuenta.
  - **USERNAME:** Nombre de usuario de la cuenta.
  - **PASSWORD:** Contraseña para la cuenta. Se guarda encriptada.
  - **FIRST_NAME:** Nombre del usuario.
  - **EMAIL:** Dirección de correo del usuario.
  - **LAST_NAME:** Apellidos del usuario.
  - **TEL:** Número de teléfono del usuario.
  - **TYPE:** Entero que indica el tipo de cuenta (-1: máquina, 0: técnico, 1: administrador).

- **EXPIRATION_DATE:** Fecha de caducidad de la cuenta (duración por defecto de 1000 años, 2 años y 2 días para administradores, técnicos y máquinas, respectivamente y aplicada desde y en el mismo momento de su creación).
- **ACTIVE:** Booleano para identificar las cuentas activas (‘True’: activa, ‘False’: inactiva).

- Login_log: Almacena todos los accesos autenticados al sistema, independientemente del tipo de cuenta o servicio que lo realice. Columnas:
  - **ID:** Identificar único del acceso.
  - **ACCOUNT_ID:** Identificador de la cuenta que ha realizado el acceso.
  - **ACCOUNT_USERNAME:** Nombre de usuario de la cuenta que realiza el acceso.
  - **DATE:** Fecha del acceso.
  - **EXTENSION_LOGIN:** Booleano, indica si el acceso se debe a que la sesión del usuario se ha extendido (‘True’: sesión extendida, ‘False’: acceso convencional sin extensión).
  - **IP:** Dirección IP desde la cual se ha realizado el acceso.
  - **AGENT_TYPE:** Además de los propios usuarios, hay otros servicios que acceden al sistema y cuyo tipo queda indicado en este campo (0: usuario, 1: máquina, 2: Pusher).
  - **DEBUG_AUTH:** Los accesos durante el desarrollo, que evaden la triple autenticación quedan marcados con este campo (‘True’: acceso desarrollador, ‘False’: acceso convencional).
- Dlinks: Modelo para los enlaces de descargas que son enviados a los clientes. Tiene las siguientes columnas:
  - **ID:** Identificador único del código de descarga.
  - **PLATFORM:** Plataforma de la cual el código permitirá descargar su archivo ejecutable.
  - **DOWNLOAD_USED:** Booleano para indicar si el código ha estado usado (‘True’: descarga usada, ‘False’: descarga no usada’).
  - **CONNECTION_USED:** Booleano para indicar si el código de descarga ha estado usado para conectar un equipo remoto (‘True’: se ha usado, ‘False’: no se ha usado).
  - **LANGUAGE:** Indica el lenguaje en que se le enviarán los correos electrónicos, como el del enlace de descarga.
  - **EMAIL:** Dirección de correo electrónico del cliente.
  - **PATH:** Ruta del ejecutable dentro del servidor. Se usa para servir el archivo correcto.

- **ISSUING_ACCOUNT:** Identificador de la cuenta que ha generado el código de descarga.
- **CREATION_DATE:** Fecha en que se ha creado el código de descarga.
- **USED_DATE:** Fecha en que se ha usado el código para conectar un equipo remoto.
- **DOWNLOAD_DATE:** Fecha de descarga del ejecutable.

- Operation_logs: Tabla para registrar todas las operaciones que se realizan sobre todas las máquinas remotas. Columnas:
  - **ID:** Identificador único de la operación.
  - **ACCOUNT_ID:** Identificador de la cuenta que ha ejecutado la operación.
  - **ACCOUNT_USERNAME:** Nombre de usuario de la cuente que ha ejecutado la operación.
  - **MAC:** Dirección MAC de la máquina sobre la que se ha ejecutado la operación.
  - **OPERATION:** Tipo de operación (listado de posibles valores en el anexo).
  - **PREVIOUS_VALUE / NEW_VALUE:** En caso que la operación cambiará algún parámetro, el valor anterior y el nuevo quedan guardados en estos campos, respectivamente.
  - **RESOURCE_ID:** Identificador interno de la máquina sobre la que se ha realizado la operación.
  - **DATE:** Fecha de registro de la operación.
  - **IP:** Dirección IP desde la cual se ha ordenado la operación.
- Machines: La tabla más importante, guarda toda la información sobre las máquinas registradas en el sistema. Almacena los siguientes campos:
  - **ID:** Identificador único de máquina.
  - **MAC_ADDRESS:** Dirección MAC de la máquina.
  - **PLATAFORM:** Nombre de la distribución.
  - **PLATFORM_RELEASE:** Código de release de la plataforma.
  - **PLATFORM_VERSION:** Versión del sistema operativo.
  - **ARCHITECTURE:** Arquitectura del sistema.
  - **HOSTNAME:** Nombre del equipo.
  - **IP_ADDRESS:** Dirección IP de la máquina.
  - **PROCESSOR:** Modelo del procesador del equipo.
  - **RAM:** Cantidad de memoria RAM instalada en el sistema.
  - **NETWORK_CARDS:** Tarjetas de red instaladas en el sistema.
  - **ACCOUNT_ID:** Cuenta de usuario asociada a la máquina, usada para las peticiones autenticadas.
  - **STATUS:** Estado de la máquina (-1: desconectada, 0: conectada, 1: reiniciándose).

- **OPCODE0 / OPCODE1 / OPCODE2 / OPCODE3 / OPCODE4:** Códigos de operación internos para ejecutar operaciones sobre la máquina (listado con su significado completo en el anexo).
- **COMMAND1 / COMMAND2 / COMMAND3 / COMMAND4 / COMMAND5:** Comandos a ejecutar en la máquina remota. Tantos como consolas simultáneas puedan mantenerse.
- **AUTH_CODE:** Código de seis dígitos que efectúa la triple autenticación.
- **INDICENT:** Información sobre el ciberincidente ocurrido en el equipo.
- **INCIDENT_DATE:** Fecha en que ocurrió el incidente.
- **INCIDENT_INFO:** Información adicional sobre el ciberincidente ocurrido.
- **CONTACT:** Nombre persona o institución responsable del equipo.
- **CONTACT_TEL:** Teléfono de contacto.
- **CONTACT_EMAIL:** Correo electrónico de contacto.
- **ADDITIONAL_INFO:** Información adicional sobre el caso.
- **MESSAGE:** Mensaje enviado desde la interfaz al cliente.

El diagrama de clases de la base de datos y sus relaciones es el siguiente:

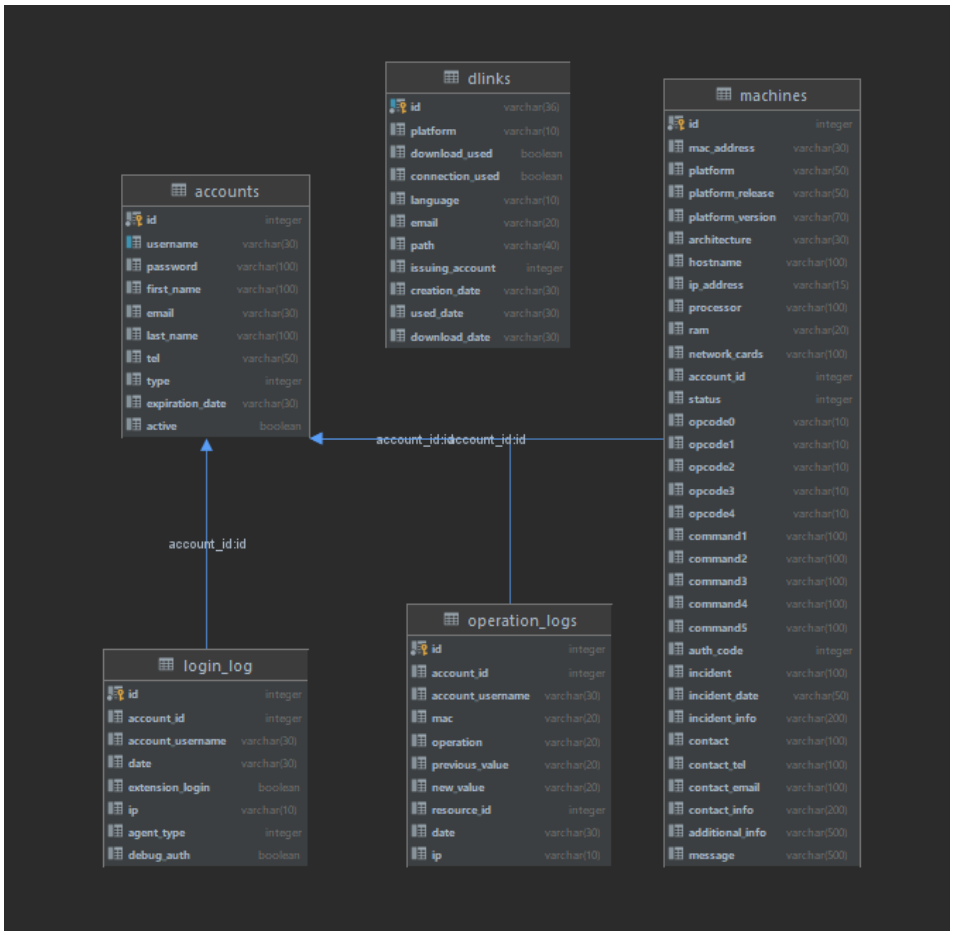


Ilustración 19: Diagrama de tablas de la base de datos. (Fuente: squid-remote)

5.4 Aspectos sobre la seguridad

El sistema implementa varias medidas de seguridad para proteger la intrusión de terceros sin autorización. La más amplia es quizás la autenticación de la API para toda petición. Menos un endpoint, el resto exigen una autenticación. En el momento en que el usuario accede al sistema, internamente, el usuario recibe un token que sirve para autenticar los requests en la API, como si por cada petición autenticara con su usuario y contraseña.

Como medida de protección ante ataques de spoofing, los endpoints a los que las máquinas envían los datos también están autenticados. De este modo, solo las máquinas tienen permitido enviar datos a estas direcciones, evitando así que un tercero envíe información errónea o maliciosa haciéndose pasar por una máquina. Como las cuentas de máquina las crea y asigna el servidor, es muy difícil que una tercera persona pueda generar una cuenta de este tipo. Además, la contraseña de estas cuentas cambia en cada conexión, así que, en el peor de los casos, si es interceptada solo será válida una vez.

Los tokens están asociados a un usuario y tiene una duración limitada. Para los accesos al sistema por parte de usuarios la duración es de una hora. Para las máquinas remotas de 24 horas, por si la intervención remota se alarga. Tras este tiempo dejan de ser válidos y el servidor no los reconoce como autenticación en los endpoints.

Conjuntamente con la caducidad de los tokens, la interfaz incorpora medidas contra las sesiones inactivas. A los 55 minutos, notifica al usuario de que su sesión expirará y le dará la posibilidad de extenderla. Si el usuario está ausente o no quiere extender su sesión, a los 60 minutos se le desconecta y se le redirige a la página de acceso.

Tanto la gestión de los tokens como el mantenimiento de la sesión en la interfaz son procesos completamente ajenos al usuario. Únicamente son medidas de seguridad para el servidor y para la interfaz.

Con el cliente, las medidas de seguridad están orientadas a evitar conexiones no intencionadas de la máquina al sistema. El proceso de conexión de un equipo al sistema se autentica mediante dos pasos. Primero el cliente solicitará el código de descarga usado y la dirección de correo en que se ha recibido. Si los datos introducidos coinciden con los registrados en el servidor, se envía por correo electrónico un código de verificación de seis dígitos. Solo si se introduce y coincide con el enviado, el servidor autorizará la conexión del equipo.

Los códigos de descarga, por lo general, tienen un único uso y su generación es completamente aleatoria. La posibilidad de que un tercero genere un código de descarga válido es remota, pero como además se requiere que el correo electrónico esté registrado, el riesgo de que alguien altere una conexión es mínimo. De cada código de descarga se mantiene tanto su fecha de descarga del ejecutable como de su uso.

La última medida de seguridad son los registros. En caso de que un tercero accediera de forma indebida al sistema, podría ser identificado mediante los logs de acceso que se mantienen. Y si llegará a ejecutar alguna operación sobre alguna máquina, se podrían obtener evidencias de ello con los registros de operación que también se mantienen.

6 Implementación

En esta sección se detallan los aspectos técnicos que hay detrás de las diferentes características del sistema. También se explicarán las decisiones tomadas en relación al diseño interno del programa, así como el uso que se les da a las tecnologías previamente mencionadas.

Dado que el código se encuentra anexo a este trabajo y es de libre acceso para cualquiera, en esta sección no se citará el código en su totalidad salvo en los casos cuya presencia resulte imperativa para la adecuada comprensión de la explicación.

6.1 Conexión inversa del cliente

En el contexto de las conexiones remotas a equipos, generalmente se usan conexiones directas. En estas el controlador al otro lado de la conexión toma la iniciativa en la comunicación y, de forma activa, indica al equipo remoto que debe hacer. Para que esto sea posible, es imperativo que en la red del equipo remoto estén los puertos necesarios abiertos y no haya cortafuegos muy restrictivos de por medio.

Como el principal objetivo del cliente es que pudiera ser ejecutado por cualquier persona, sin requerir de conocimientos avanzados por parte del usuario final. Por ello se ha intentado eximir al usuario final de toda posible configuración que de la red y el equipo que el cliente pudiera requerir. Entonces, con el objetivo de simplificar su uso, se ha optado por usar una conexión inversa.

En las conexiones inversas, a diferencia de las conexiones normales, no es el servidor quien toma la iniciativa en la comunicación sino el cliente. El cliente cada cierto tiempo preguntará al servidor que acción debe ejecutar.

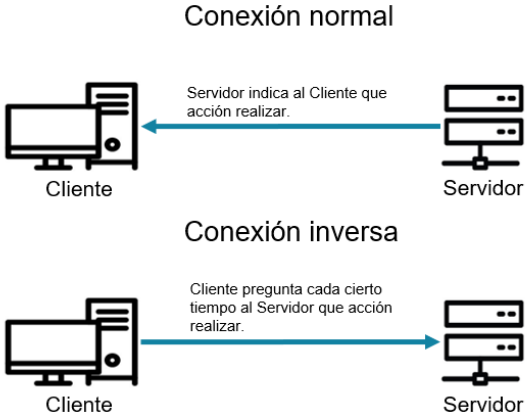


Ilustración 20: Comparativa entre conexión normal e inversa. (Fuente: Elaboración propia)

El servidor implementa cinco endpoints de monitorización de acciones, que, al ser solicitados por el cliente, le devuelve un código de operación que codifica la acción remota a realizar. Los códigos de operación se estructuran como una cadena de caracteres, donde los tres primeros caracteres de cada uno identifican el tipo de operación. Los tipos de operaciones disponibles son los siguientes:

- PNG: Realizar test de conexión con el servidor.
- MSG: Mostrar mensaje por pantalla.
- PRC: Recuperar y enviar al servidor información sobre los procesos en ejecución.
- CMD: Acciones relacionadas con la consola remota (gestión de sesiones o ejecución de comandos).
- NET: Captura del tráfico de red.
- DEV: Recuperar y enviar al servidor los dispositivos conectados.
- SYS: Calcular datos sobre el rendimiento del equipo y transmitirlos al servidor.

La cabecera de la operación puede venir seguida de parámetros, que variaran en función del tipo de operación. Se puede encontrar en el anexo una lista con todos los parámetros y opciones disponibles para cada operación.

Cada código de operación es monitorizado en paralelo por subprocesos diferentes en el cliente. Esto permite la ejecución simultánea de varias operaciones al mismo tiempo. Salvo el código de operación *OPCODE2*, que puede terminar la ejecución de todos los procesos, ejecutan la misma función de monitorización. El diagrama de flujo de esta función es el siguiente:

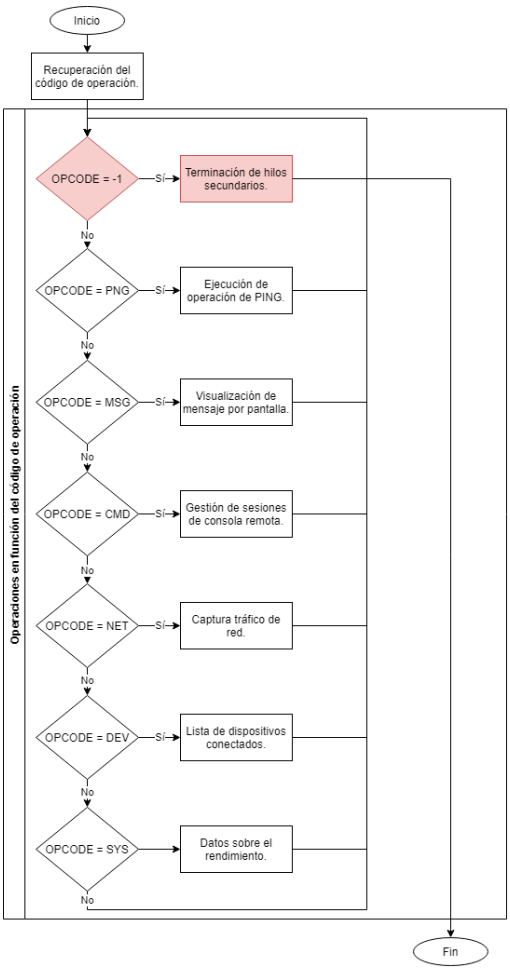


Ilustración 21: Diagrama de flujo para la monitorización de OPCODE2. El resto son iguales exceptuando los elementos en rojo. (Fuente: Elaboración propia)

Gracias a esta implementación se consigue evitar la apertura de puertos en la red del equipo comprometido, ya que es el cliente quien se conecta al servidor y no al revés. También se evita incomodar al usuario final, que, si requiere asistencia remota debido a un ciberataque, lo último que querrá son más vulnerabilidades en su entorno.

Otra alternativa que se consideró durante la implementación, fue la de usar una red VPN para poder realizar una conexión directa con el equipo remoto. Usar este método hubiera requerido de infraestructuras y costes adicionales que no añadían más valor que la conexión inversa, por lo que no se aplicó.

6.2 Recuperación, transmisión e intercambio de la información

El recopilado de información es una funcionalidad vital para el funcionamiento del sistema. En lo que se refiere a información estática, aquella que reside o es proporcionada directamente por el equipo remoto, el proceso de obtención y procesado es muy similar en todos los casos.

Una vez se ha identificado el tipo de operación, el cliente procede a su ejecución. Para ello y en función de que operación se trate recurre a las siguientes librerías para obtener los datos:

- psutil: Usada tanto en la recuperación del software y hardware del equipo, así como para la monitorización del rendimiento y los procesos.
- pyudev / pywinusb: Elaboración del listado de dispositivos conectados. Para lograr la compatibilidad entre plataformas se usa la primera para sistemas Linux y la segunda para Windows.

A partir de las funciones y atributos que proporciona cada librería se elabora un objecto JSON, esencialmente un formato de texto sencillo, para enviar la información al servidor. Estos objetos de texto JSON se estructuran internamente como parejas clave – valor, característica que hace que construir un analizador sintáctico para su parseado se mucho más sencillo. Una vez codificada la información en JSON, se envía de vuelta al servidor por endpoints específicos.

La problemática con que se encuentra el sistema en este punto, es la de cómo hacer llegar la información a la interfaz. En otros sistemas este proceso se realiza de forma directa mediante una comunicación activa entre servidor e interfaz. Sin embargo, tal y como se ha estructurado el sistema, esto no es posible.

En la arquitectura del sistema, tal y como se ha comentado en el apartado de diseño, el servidor se sitúa entre cliente e interfaz. Toda comunicación cliente – interfaz debe pasar por el servidor y se lleva a cabo mediante peticiones HTTP a endpoints específicos de la API. Para que esta comunicación sea

factible hay tratar dos factores clave: el almacenamiento temporal de los datos en el servidor y el sincronismo con la interfaz.

Para lo primero se usa una base de datos Redis como una memoria temporal. Los datos en las tareas de monitorización se transfieren constantemente, por lo que guardarlos en una base de datos convencional seria innecesario y muy costoso en cuanto a inserciones / eliminaciones de filas en las tablas de la base de datos. En su lugar el sistema usa una base de datos de Redis como si de una memoria cache temporal se tratase. Cuando el cliente envía información al servidor, este los guarda en Redis con una clave, que facilita su posterior recuperación. En el caso de la lista de dispositivos, está clave tiene el siguiente formato: “D_LST<DIRECCIÓN_MAC_EQUIPO>”.

Para el segundo problema se usan los canales de Pusher. Una vez el servidor ha guardado los datos en Redis, notifica mediante este servicio a la interfaz, que solicitará mediante un request al servidor los datos que se han guardado. El servidor los recuperará con la clave guardada y se los devolverá a la interfaz.

```
32  class DevicesInfoReceiver(Resource):
33
34      @auth.login_required(role=['machine'])
35      def post(self, mac):
36          machine = MachineModel.find_by_mac(mac)
37          if machine:
38              machine.opcode2 = 0
39              machine.save_to_db()
40              r.set("D_LST" + mac, request.data)
41              pusher.trigger("private-" + machine.mac_address.replace(":", ""), "devices_update",
42                             {"message": "new post"})
43              return {"message": "Data received"}, 200
44          return {"message": "Machine not found"}, 404
45
46      @auth.login_required(role=['support', 'admin'])
47      def get(self, mac):
48          machine = MachineModel.find_by_mac(mac)
49          if machine:
50              ret = json.loads(r.get("D_LST" + mac))
51              return ret, 200
52          return {"message": "Machine not found"}, 404
53
```

Ilustración 22: Fragmento de código de la transferencia de datos cliente - servidor y servidor - interfaz. (Fuente: squid-remote)

El fragmento de código anterior reside en el servidor y corresponde al endpoint que usa el cliente para enviar la información sobre los dispositivos conectados. En las líneas 40 y 41 se realiza el guardado en Redis y la notificación a la interfaz, respectivamente. El canal por el cual se notifica a la interfaz es único y privado para cada máquina.

Por su parte, la interfaz está conectada al mismo canal privado de la máquina que el servidor y además, tiene asociada una función a la notificación que envía el servidor (en la figura anterior, "*devices_update*"). El método asociado a este evento es el que recupera los datos que el servidor ha guardado en Redis.

```
126     devicesMonitoringChannel() {
127       var channel = this.pusher.subscribe('private-' + this.machine.mac_address.replace(/:/g, ""))
128       // eslint-disable-next-line no-unused-vars
129       var handler = function () {
130         this.getDevices()
131       }
132       channel.bind("devices_update", this.getDevices)
133     },
```

Ilustración 23: Fragmento de código de la sincronización por notificaciones en la interfaz (Fuente: squid-remote)

El proceso de comunicación e intercambio de datos implementado en el sistema se resume con el siguiente diagrama:

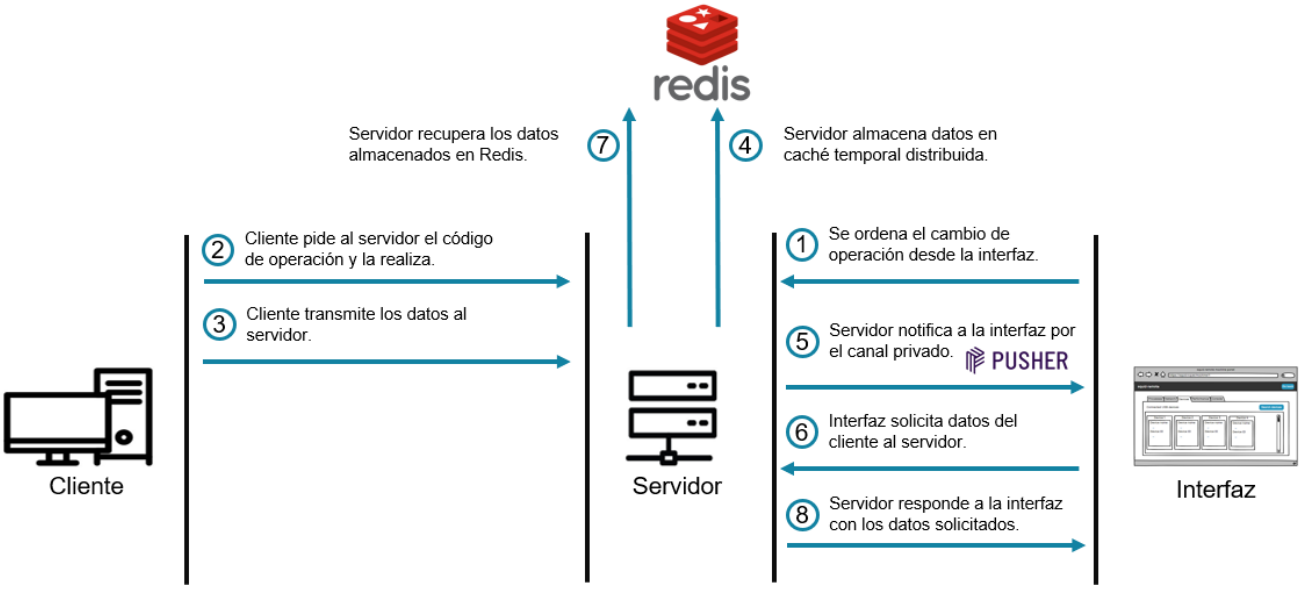


Ilustración 24: Diagrama del proceso completo de intercambio de datos cliente - servidor - interfaz. (Fuente: Elaboración propia)

6.3 Captura del tráfico de red

Esta funcionalidad permite capturar los paquetes individuales que se transmiten en la red en la que el equipo remoto está conectado. En el contexto del sistema y durante la realización de un ciber-análisis, poder interceptar el tráfico saliente y entrante de una red podría ayudar a localizar puertas traseras o incluso software malicioso en un equipo.

La implementación interna de esta función no difiere mucho del resto que también recuperan información del equipo. En ambos casos los datos recopilados se codifican en JSON y se transmiten al servidor. La diferencia con el resto de procesos está en que, en este caso, para recabar los datos, es necesario aportar una cierta configuración.

El formato de los códigos de operación usado en las operaciones de red puede variar ya que además de los parámetros de configuración existen tres subfunciones dentro de este bloque: listar tarjetas de red instaladas, captura de paquetes por tiempo y captura de paquetes por cantidad. Para las tres subfunciones, el formato es el siguiente:

NET-<SUBFUNCIÓN>-<ID_TARJETA_RED>-<PARÁMETRO1>-<PARÁMETRO2>

Los posibles valores de subfunciones pueden ser los siguientes:

- LST: Listará todas las tarjetas de red conectadas. Cada tarjeta recibe un identificador único incremental. No hace uso de ningún parámetro, por lo que el resto de campos se llenan con "X". Ejemplo: "*NET-LST-X-X-X*".
- TCP: Función de captura de paquetes por tiempo. Hace uso del parámetro uno, que interpretará como los segundos durante los que permanecerá capturando el tráfico.
- PCP: Función de captura por cantidad de paquetes. Se hace uso del parámetro uno que se interpreta como la cantidad de paquetes a capturar.

En todas las funciones el sistema se sirve de las funciones del programa Scapy para la captura de paquetes y la identificación de las tarjetas de red. En el caso de Windows, también se apoya en la librería Npcap.

La subfunción de listar las tarjetas de red se ejecuta automáticamente tras la conexión remota al equipo, ya que para la selección desde la interfaz de la tarjeta de red es necesario tener la lista. Está subfunción y la de capturar paquetes por tiempo usan funciones directamente proporcionadas por Scapy. La captura por tiempo, sin embargo, no es una función existente en Scapy. Esta se implementa en el sistema con la función de sniffer de Scapy seguido de un delay tras el cual se detiene la captura de paquetes.

En ambos modos de captura, se requieren los parámetros uno, dos y el identificador de captura de red. El significado del parámetro uno, como se ha explicado un par de párrafos atrás, varía en función de la operación. El parámetro dos tiene el mismo significado para ambas subfunciones y permite aplicar filtros a la captura de paquetes. En el momento de redactar esta memoria, los filtros existentes aplican las capas de paquete TCP, ICMP, DNS y UDP. Cada filtro se separa con una barra baja dentro del código de operación. Por ejemplo, el código de operación "*NET-PCP-1-10-TCP_ICMP_DNS"*, indica que se trata de una captura de diez paquetes con capas TCP, ICMP y DNS.

6.4 Consola remota

La consola remota permite la ejecución de comandos en el equipo remoto. Dos retos planteados en la implementación de esta funcionalidad han sido el mantenimiento de varias consolas remotas al mismo tiempo y el mantenimiento del directorio de trabajo de cada sesión.

En todas las consolas remotas, de código abierto y que he encontrado por Github, el directorio de trabajo no se mantiene entre comandos. Es decir, se introduce un comando para acceder a un directorio y al introducir el siguiente, se vuelve a estar en el directorio inicial. También solo podían mantener una consola remota a la vez. Estos dos problemas, han sido convertidos en las medidas de éxito de esta funcionalidad.

La implementación se divide en dos partes: una de gestión y otra de ejecución. La parte de gestión es común para todas las plataformas y se encarga de crear, reiniciar o terminar las sesiones de consola remotas. La operación de estás funciones de gestión se realizan desde unos endpoints específicos y reservados para este mismo motivo.

La parte de gestión se opera desde los códigos de operación, que el cliente monitoriza. Como en el caso de la captura del tráfico de red, también es posible con este tipo de operación enviar parámetros. El formato de los códigos de operación de gestión son los siguientes:

CMD-<SUBFUNCIÓN>-<ID_SESIÓN>

Las posibles subfunciones que se pueden usar son las siguientes:

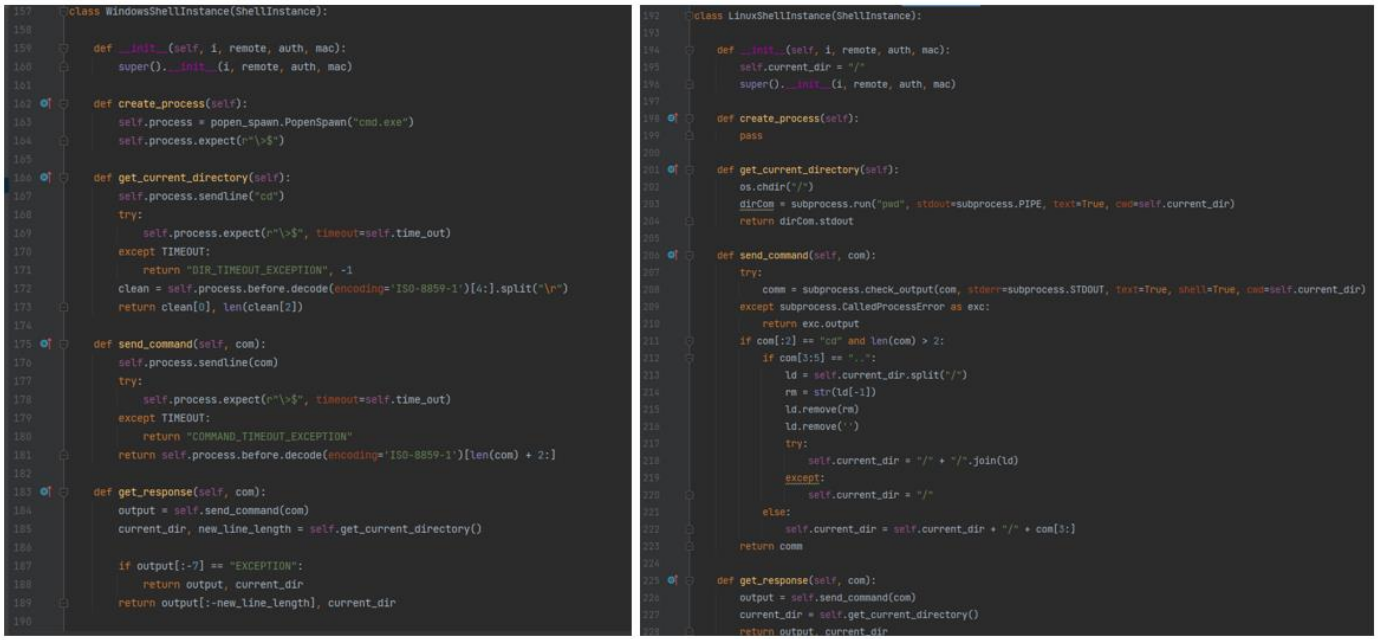
- NEW: Crea una nueva sesión de consola remota.
- RES: Reinicia una sesión de consola remota determinada.
- FIN: Termina la ejecución de una sesión de consola remota.
- TER: Termina la ejecución de todas las sesiones remotas. No necesita identificador de sesión, así que el parámetro se rellana con “X”. Ejemplo: “*CMD-TER-X*”.

Como se puede intuir, el identificador de sesión indicado en la creación de una consola remota, permite identificarla entre el resto y ejercer acciones concretas contra esta.

La otra gran parte es la de ejecución, que es la que interpreta y ejecuta los comandos en el equipo remoto. El comando a ejecutar que se introduce en la consola de la interfaz es enviado al servidor que lo guardará en la base de datos. En concreto en la tabla de máquinas y en la fila correspondiente a la máquina conectada. Se usan los campos COMMAND1, COMMAND2, COMMAND3, COMMAND4 y COMMAND5. Se infiere por el número de campos para comandos que el límite de sesiones remotas simultaneas es de cinco.

Cada sesión de consola remota, en su propio proceso, monitoriza el campo de comando que le corresponde en función de su identificador. En cuanto se encuentra algún valor en el campo que le corresponde, se interpreta como comando y se ejecuta. La salida de los comandos se transmite al servidor siguiendo el proceso explicado en el apartado 6.2.

Como cada plataforma tiene su propia sintaxis para los comandos, existe una implementación distinta para cada una. En Windows se usa la librería pexpect mientras que en Linux se usa subprocess. El mantenimiento del directorio de trabajo en Windows se da de forma nativa, ya que la instancia de la consola remota ejecuta directamente el proceso "*cmd.exe*", correspondiente al intérprete de comandos de Windows, que ya se encarga de la gestión del directorio. En Linux esto no es posible, puesto que cada distribución puede llamar distinto al proceso del intérprete de comandos. Para mantener el directorio en este caso el cliente mantiene el directorio, a parte del sistema, tras cada comando de acceso o salida de un directorio ("*cd <directorio>*" y "*cd ..*" respectivamente).



```
class WindowsShellInstance(ShellInstance):

    def __init__(self, i, remote, auth, mac):
        super().__init__(i, remote, auth, mac)

    def create_process(self):
        self.process = popen_spawn.PopenSpawn("cmd.exe")
        self.process.expect(r"\>$")

    def get_current_directory(self):
        self.process.sendline("cd")
        try:
            self.process.expect(r"\>$", timeout=self.time_out)
        except TIMEOUT:
            return "DIR_TIMEOUT_EXCEPTION", -1
        clean = self.process.before.decode(encoding='ISO-8859-1')[4:].split("\n")
        return clean[0], len(clean[2])

    def send_command(self, com):
        self.process.sendline(com)
        try:
            self.process.expect(r"\>$", timeout=self.time_out)
        except TIMEOUT:
            return "COMMAND_TIMEOUT_EXCEPTION"
        return self.process.before.decode(encoding='ISO-8859-1')[len(com) + 2:]

    def get_response(self, com):
        output = self.send_command(com)
        current_dir, new_line_length = self.get_current_directory()

        if output[:-7] == "EXCEPTION":
            return output, current_dir
        return output[:-new_line_length], current_dir
```

```
class LinuxShellInstance(ShellInstance):

    def __init__(self, i, remote, auth, mac):
        self.current_dir = "/"
        super().__init__(i, remote, auth, mac)

    def create_process(self):
        pass

    def get_current_directory(self):
        os.chdir("/")
        dirCom = subprocess.run("pwd", stdout=subprocess.PIPE, text=True, cwd=self.current_dir)
        return dirCom.stdout

    def send_command(self, com):
        try:
            comm = subprocess.check_output(com, stderr=subprocess.STDOUT, text=True, shell=True, cwd=self.current_dir)
        except subprocess.CalledProcessError as exc:
            return exc.output
        if com[:2] == "cd" and len(com) > 2:
            if com[3:5] == "..":
                ld = self.current_dir.split("/")
                rm = str(ld[-1])
                ld.remove(rm)
                ld.remove('')
                try:
                    self.current_dir = "/" + "/".join(ld)
                except:
                    self.current_dir = "/"
            else:
                self.current_dir = self.current_dir + "/" + com[3:]
        return comm

    def get_response(self, com):
        output = self.send_command(com)
        current_dir = self.get_current_directory()
        return output, current_dir
```

Ilustración 25: Comparativa entre la implementación de la consola remota en Windows y en Linux. (Fuente: squid-remote)

Otro problema derivado de la utilización de una conexión inversa y que afecta a la consola remota es que no es posible saber de forma activa cuando un comando termina su ejecución. Si un comando tiene un tiempo de ejecución muy largo, puede que la sesión remota se quede colgada. Para evitar esto se aplican dos medidas. La primera consiste en buscar un retorno de carro en el último carácter de la salida del comando. Si se encuentra, significa que el comando ha terminado correctamente. Si no se encuentra se aplica un timeout, que por defecto es de diez segundos.

Tanto el timeout como las acciones de gestión de una sesión concreta pueden ser usados desde la consola desde la interfaz, usando unos comandos reservas y específicos para estas tareas. Son los siguientes:

- help: Muestra la pantalla de ayuda en la interfaz.
- clear: Limpia las salidas de los comandos de la sesión de consola de la pantalla.
- terminate: Termina la ejecución de la sesión de consola remota.
- restart: Reinicia la sesión de consola remota.
- set-timeout N: Sirve para establecer el tiempo de timeout. El tiempo debe expresarse en segundos y colocarse en lugar de la N. Por ejemplo, para tener un timeout de 20 segundos habría que entrar "*set-timeout 20"*.

La consola emulada en la interfaz se basa en el proyecto "*vue-shell*" de Salah Eddine (HalasProject en Github). Este proyecto de código abierto y con licencia MIT, proporciona un componente cuya fachada simula una consola de un equipo. Este proyecto se ha usado como base para la interfaz "*squid-shell"* implementada en el sistema. Dado que "*vue-shell*" era únicamente una fachada, toda la lógica, rediseño y adaptación ha sido implementado para este trabajo.

La licencia MIT, permite el uso comercial, la modificación, distribución y el uso privado del proyecto sin aportar ninguna responsabilidad o garantía por parte del autor del proyecto. La única condición que impone esta licencia es que se mantenga la licencia original adjunta al nuevo código fuente. Atendiendo a este requisito, la licencia original puede encontrarse en el directorio raíz del proyecto, en el fichero "*LICENSE_vue-shell*".

6.5 Test de conexión con equipos remotos

La utilidad de diagnóstico *ping*, llamada test de conexión en la implementación del sistema, sirve para comprobar el estado de la red y la comunicación del equipo anfitrión local con uno o varios remotos que ejecuten IP. Esta función se implementa normalmente como un envío de un determinado número de paquetes con una secuencia de números específica y que deben ser recibidos tal y como se han enviado para validar el correcto funcionamiento de la red.

Como la arquitectura del sistema no contempla una conexión remota literal a la red del equipo comprometido no es posible enviar paquetes al equipo remoto, y, por ende, ejecutar la función de ping literalmente.

Como solución, el sistema usa los códigos de operación, OPCODE, que se han visto en otros puntos de la implementación. Cuando desde la interfaz se solicita ejecutar esta función, el servidor coloca en el OPCODE0 el valor "*PNG-5"*. Este código de operación, cuya cabecera es "*PNG*" identifica la operación de ping. El cinco indica el número de repeticiones a ejecutar.

El cliente, que está monitorizando constantemente los códigos de operación, recibe la orden y la devuelve con la repetición decrementada. Es decir, recibe "*PNG-5*" y devuelve "*PNG-4"*, y así

sucesivamente hasta llegar a “*PNG-0*”. Por cada respuesta del cliente, en que la repetición es mayor a cero, el servidor lo interpreta como una respuesta del cliente y notifica a la interfaz de que el cliente está conectado y respondiendo a los mensajes.

La interfaz usa esta notificación para mostrar una visualización de la acción en pantalla. Estas son visibles desde el panel de control de la máquina o desde el listado de todas las máquinas.

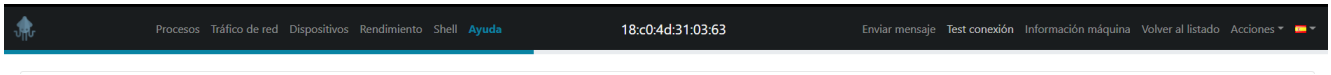


Ilustración 26: Barra de progreso del proceso de test de conexión a una máquina remota, visto desde el panel de control de la máquina. (Fuente: Elaboración propia)

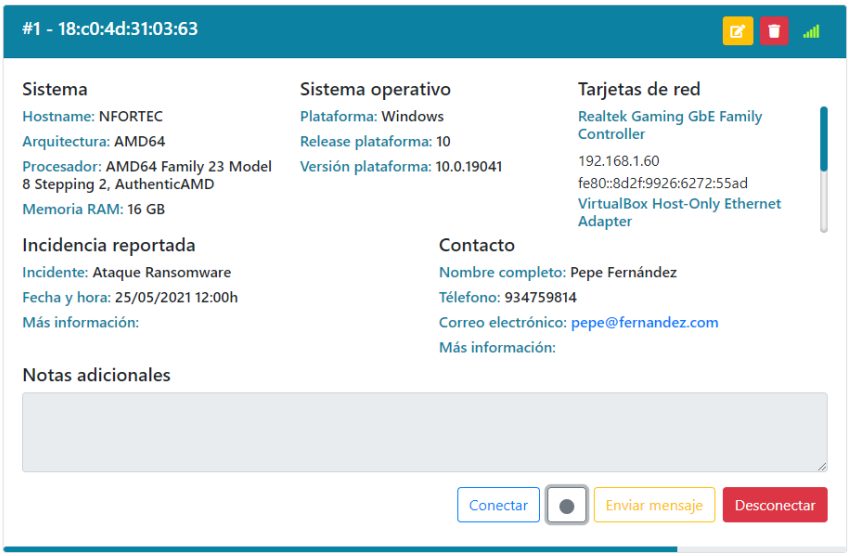


Ilustración 27: Barra de progreso del proceso de test de conexión a una máquina remota, visto desde la página principal. (Fuente: squid-remote)

6.6 Autentificación de usuarios y mantenimiento de sesiones

La protección de los endpoints de la API se gestiona por roles y usando una autentificación básica. El servidor tiene una clave secreta que usa para generar los tokens de usuario. El token entonces, consiste en el nombre de usuario del usuario encriptado. Este token se usa como contraseña de la autenticación básica usuario – contraseña, donde se usa el identificador de cuenta como usuario y el token como contraseña.

El servidor hace usado de la librería *Flask-HTTPAuth* para gestionar la autenticación de los endpoints. Internamente cuando el servidor recibe una petición a un endpoint protegido, primero comprueba si el token ha caducado. Si no ha caducado comprueba que el nombre de usuario de la ID proporcionada coincide con el que el token encripta. Por último y si coincide, consulta en base de datos el rol del usuario. Si el usuario no tiene los permisos necesarios o el token ha expirado o no coincide con el ID, se deniega la petición, retornando el código de error 403 (*Forbidden)* de HTTP.

Pese a ser un sistema de autentificación relativamente sencillo, es más que suficiente para evitar los posibles problemas derivados de un ataque de spoofing. Además, como en el despliegue todas las peticiones se realizan por el puerto 443 (HTTPS), interceptar el token se hace prácticamente imposible.

El mantenimiento de la sesión, sin embargo, se implementa a nivel de interfaz y para los usuarios técnicos o administradores que la usan. En cuanto el usuario inicia sesión en la interfaz, se establecen dos temporizadores de 55 y 60 minutos. El primero se encargará de dar la opción al usuario de extender la sesión. El segundo expulsa al usuario del sistema.

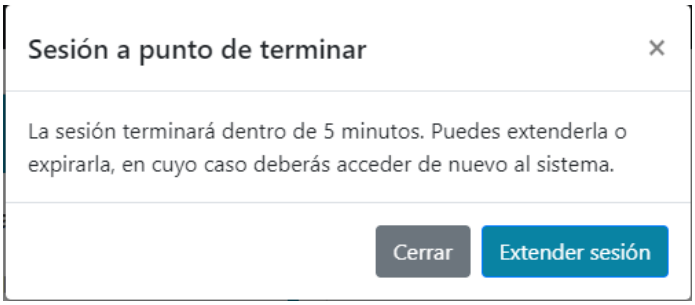


Ilustración 28: Mensaje de aviso de terminación de sesión con la opción a extenderla. (Fuente: squid-remote)

Si el usuario decide extender su sesión, el servidor generará un nuevo token que la interfaz sustituirá por el anterior. Al ampliar la duración de la sesión, los temporizadores se resetean debido a la extensión de la sesión.

Tal y como el sistema lo implementa, una extensión de sesión no es lo mismo que un acceso nuevo, puesto que en la API estas dos funciones se distribuyen en endpoints separados. Se ha implementado así ya que un acceso nuevo no requiere autenticación, ya que es el punto en que el servidor devuelve el token inicial. Al extender la sesión, pero, el usuario ya está dentro del sistema, por lo que el endpoint sí que va autenticado y verifica que el usuario tiene permisos para hacerlo.

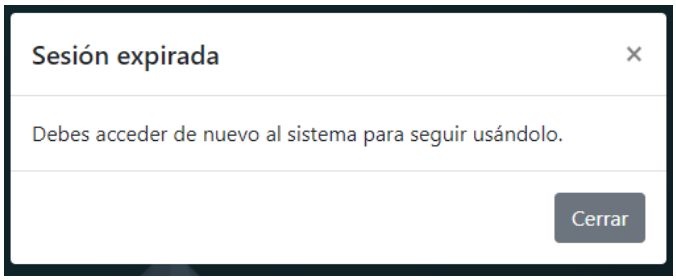


Ilustración 29: Mensaje de sesión caducada. (Fuente: squid-remote)

Durante la duración del token, la interfaz mantiene la sesión iniciada mediante el uso de cookies. Las cookies están encriptadas cuando el sistema se carga sobre HTTPS. Esto protege la manipulación de los datos guardados. Si el sistema en lugar de desplegarse de forma segura se despliega sobre HTTP, la sesión no se mantiene puesto que no es posible encriptar las cookies al no haber ningún certificado

disponible. La única desventaja de esto es que al recargar la página será necesario introducir usuario y contraseña de nuevo.

6.7 Traducción del sistema

Para facilitar la comprensión de las instrucciones y facilitar el uso por parte de los usuarios finales, todo el sistema tiene los mensajes localizados en un fichero independiente al resto del código. Gracias a esto es posible añadir traducciones de forma sencilla sin necesidad de tener que hacer grandes modificaciones en el código.

En el cliente el fichero que contiene las traducciones se llama "*messages.py*". Todos los mensajes guardados se indexan en formato diccionario de dos niveles. Desde el nivel más externo, la primera clave identifica el idioma. Las claves dentro de estas, contienen las traducciones. La selección del idioma se hace clicando una tecla determinada, tal y como explica el cliente al ser ejecutado.

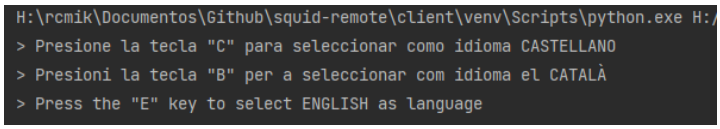


Ilustración 30: Selección del idioma en el cliente mediante la pulsación de ciertas teclas. (Fuente: squid-remote)

La interfaz usa el plugin *Vue I18n*, un plugin para la internalización de las aplicaciones web desarrolladas en Vue.js. Similar al caso del cliente, la interfaz también se sirve de un fichero externo para la contención de las traducciones. También se guarda el texto en formato diccionario (clave – valor) más dispone de un nivel de profundidad adicional. Desde el más externo, las claves indican idioma, componente de la interfaz que lo usa y traducción, por ese orden. Este nivel adicional facilita la modularidad de las traducciones, ya que el texto se separa por cada componente. El fichero que contiene las traducciones en la interfaz es "*msgs.js*", dentro del directorio assets.

Para mantener el sistema en el idioma seleccionado la interfaz usa cookies. Al igual que con la sesión, las cookies usadas se encriptan sobre HTTPS. Cargar el sistema sobre HTTP puede suponer que las preferencias de idioma no se guarden.

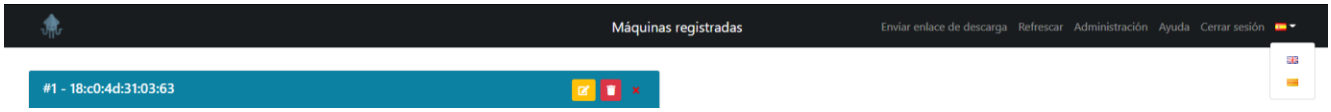


Ilustración 31: Selección del idioma de la interfaz, en la parte superior derecha. (Fuente: squid-remote)

Los correos electrónicos también pueden ser traducidos creando nuevas plantillas y traduciéndolas a mano. Para esto, aunque pocas líneas, si es necesario modificar ligeramente el código.

6.8 Triple factor de autenticación

Aunque el objetivo del sistema es asistir a los usuarios que han sufrido algún ciberataque, en las manos equivocadas puede usarse como una herramienta maliciosa para acceder al equipo de una víctima. Para evitar esta situación el sistema implementa un protocolo de tres pasos para cerciorarse que todas las conexiones son legítimas. Los pasos son los siguientes:

1. **Servidor:** Registra el correo electrónico del usuario y la plataforma de su sistema. Tras esto, le envía un correo electrónico con el enlace de descarga de un solo uso del cliente y del código de descarga para la conexión.
2. **Cliente:** El usuario se descarga el cliente e introduce su correo y el código que se le ha enviado por correo electrónico. El cliente envía ambos al servidor que valida la petición.
3. **Servidor:** Si el código de descarga está registrado con el correo electrónico introducido y todavía no se ha usado, genera y envía al usuario un código de seis dígitos por correo electrónico.
4. **Cliente:** El usuario introduce el código de seis dígitos. El cliente lo envía al servidor.
5. **Servidor:** Recibe el código y si es legítimo, autoriza la conexión de la máquina.

Si en cualquier etapa del proceso se recibe un error, bien porque el correo electrónico y/o el código de descarga no están registrados o no coinciden en base de datos, porque el código de descarga ya se ha usado o porque el código de seis dígitos no coincide, el sistema deniega toda conexión.

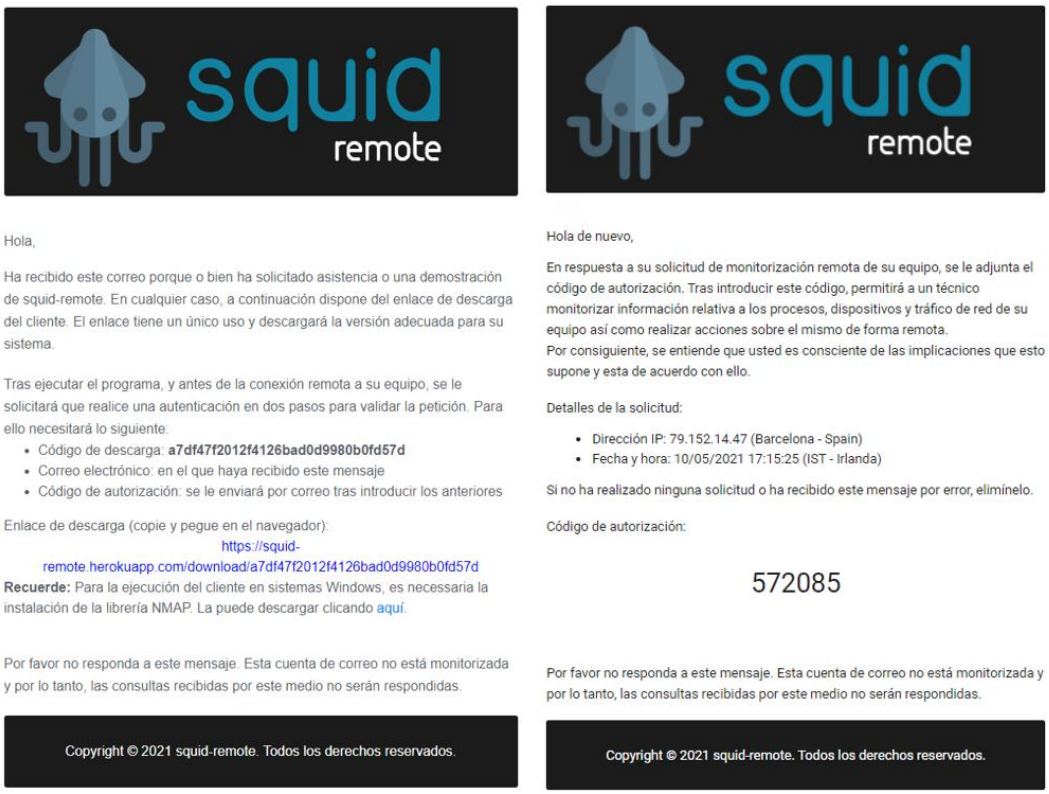


Ilustración 32: A la izquierda, correo electrónico con el código de descarga e instrucciones del cliente. A la derecha, correo electrónico con el código de autorización. (Fuente: squid-remote)

6.9 Gestión de datos estáticos

La gestión de los datos estáticos se realiza desde la interfaz usando la REST-API que implementa el servidor e incluye los datos de registros, cuentas, códigos de descarga y datos de caso. Este implementa varios endpoints cuya función es tratar con estos siguiendo los estándares existentes. Se usan los métodos de requests GET, POST, PUT y DELETE para recuperar los datos, crearlos, actualizarlos o eliminarlos, respectivamente.

En el caso de los registros, por cuestiones de diseño y seguridad, solo es posible su alteración. No tendría ningún sentido poder manipularlos al libre albedrio puesto que perderían su utilidad. Por este mismo motivo, el servidor solo implementa el método GET tanto para los registros de acceso como los de operaciones.

Para el resto de situaciones, el sistema es más flexible. En los códigos de descarga, por ejemplo, es posible activarlos y desactivarlos sin restricciones. Esto se ha hecho así por si una intervención remota se prolonga a varios días no sea necesario enviarle un nuevo código de descarga al cliente.

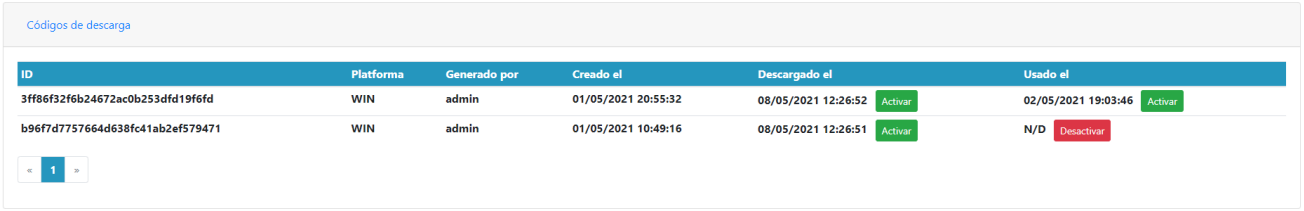


Ilustración 33: Administración de códigos de descarga en el sistema. (Fuente: squid-remote)

Los datos de las cuentas de usuario también pueden ser modificados, aunque solo por el administrador. Desde el panel de control es posible cambiar todos los campos de la cuenta excepto el rol y la contraseña. También es posible desactivarlas y cambiarles la contraseña. En este último caso, la contraseña es enviada por correo electrónico al usuario por cuestiones de privacidad. Las cuentas no pueden ser eliminadas del sistema desde la interfaz.

Ilustración 34: Correo electrónico de nueva cuenta de usuario. (Fuente: squid-remote)

Por último, la información sobre las máquinas registradas. Los datos que se guardan de cada máquina relacionados con el caso investigado pueden ser modificados desde la interfaz, a excepción de los que especifican hardware y software del equipo. Las máquinas pueden ser borradas, en cuyo caso son eliminadas directamente de la base de datos, a excepción de sus registros que permanecen.

Ilustración 35: Gestión de la información de las máquinas registradas. (Fuente: squid-remote)

7 Despliegue

En total durante el desarrollo del proyecto se han usado dos entornos: uno de desarrollo y otro de producción. El entorno de desarrollo se ha usado para validar y probar los cambios durante el sprint. Al acabar el sprint y publicar la reléase y nueva versión del sistema, los cambios pasaban al entorno de producción.

Ambos entornos están alojados en Heroku, tecnología que se ha mencionado anteriormente. Aunque ambos tienen el mismo funcionamiento y objetivos, a nivel de recursos el entorno de producción dispone de más prestaciones, tanto a nivel de disponibilidad como de cuota de tráfico de red.

Los despliegues se han automatizado usando Travis CI, también mencionada al comienzo de este documento. Travis sirve de puente entre el repositorio de Github, donde el código reside, y los entornos de despliegue. También se ha encargado de compilar y generar los ficheros finales de la interfaz, que son enviados conjuntamente con el código del servidor al despliegue.

En Github se ha trabajado usando dos ramas principales, la de desarrollo y la de producción. Este modo de trabajo se ha trasladado a Travis usando su configuración de modo que, todos los cambios publicados en la rama de desarrollo y producción se publican automáticamente en los entornos de desarrollo y producción, respectivamente.

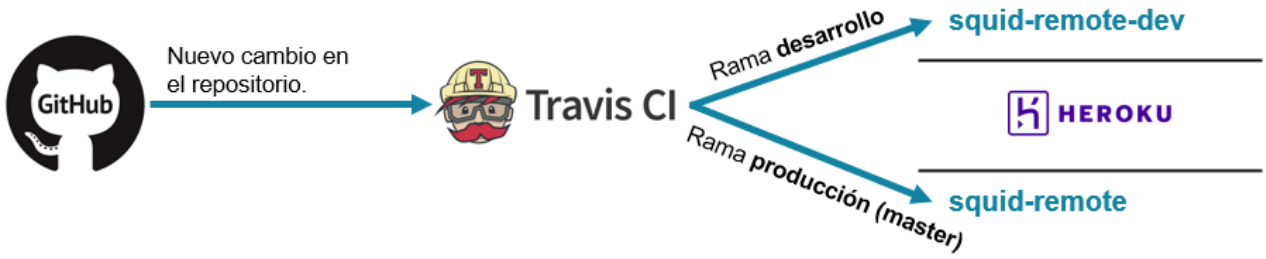


Ilustración 36: Diagrama de flujo de despliegue usado durante el desarrollo. (Fuente: Elaboración propia)

Durante el desarrollo, la base de datos usado ha sido *SQLite*, que pese a depender de un fichero único ha tenido una gestión muy simple. Para las fases de testeo final y producción, ambos entornos se han migrado a una base de datos de *PostgreSQL*, que aporta un mejor escalado vertical.

Las direcciones de ambos despliegues son las siguientes, para producción y desarrollo respectivamente:

https://squid-remote.herokuapp.com/ o https://squid.rcq.es/

https://squid-remote-dev.herokuapp.com/

Tras la finalización del desarrollo del sistema, ambos entornos sirven como producción, actuando el de desarrollo como espejo del de producción. El de desarrollo tiene una carga inicial más lenta debido a la diferencia de prestaciones con el de producción.

Para crear los binarios del cliente se ha usado *pyinstaller*.

8 Pruebas y resultados

8.1 Pruebas realizadas

Al usar la metodología de trabajo ágil Scrum, tras cada iteración se disponía de un prototipo funcional del sistema. Esto ha permitido realizar pequeñas pruebas de usabilidad durante las etapas de desarrollo, y que, pese a no hacerse con un producto acabado, han permitido identificar errores con antelación, evitando así la acumulación o agravamiento de los mismos.

Las pruebas más significativas se han realizado con el sistema terminado y en entornos de actuación reales. Para estas se han usado equipos de las dos plataformas soportadas, Windows y Linux, y en el caso de esta última con varias distribuciones distintas. Además, los equipos remotos han usado directamente los endpoints de producción.

Sobre la interfaz también se han realizado pruebas de robustez y adaptación. En estas pruebas se ha comprobado su compatibilidad con dispositivos variados incluyendo smartphones, tablets y ordenadores.

De todo el conjunto de pruebas se han identificado y solucionado varios problemas, los más graves han sido los siguientes:

Problema	Explicación	Solución
La captura de paquetes en Windows no funciona y muestra errores en la pantalla del cliente.	En Windows no existe una librería nativa para la captura de paquetes, por lo que la librería usada no funciona correctamente. En Linux sin embargo sí que existen estas librerías de bajo nivel.	Se establece como requisito para la ejecución del cliente la instalación de las librerías de Npcap.
La interfaz no es capaz de recuperar las tarjetas de red de los equipos Windows.		
Error en el despliegue que impide el envío de correos electrónicos.	Heroku tiene algunas políticas para evitar las aplicaciones que envían spam masivo. Aplica a aquellas que envían mensajes directamente por correo electrónico.	Se añade el servicio de Sengrid para el envío de los correos electrónicos del sistema. Al ser externo a Heroku, los correos no son bloqueados.
Cuando un correo electrónico tiene más de un código de descarga la triple autenticación falla.	El servidor busca el código de descarga únicamente por el correo electrónico. Cuando hay más de uno no puede verificar	Actualización del triple factor de autenticación para que el servidor use el correo electrónico y el código de descarga para la verificación.

	cual es el correcto porque no lo conoce.	
Los enlaces de descarga funcionan más de una vez.	El servidor tenía configurado un tiempo de cache bastante amplio para los ficheros que sirve.	Se elimina el tiempo de caché de los ficheros del servidor.
Carga excesiva del servidor cuando varias máquinas se monitorizan de forma simultánea.	Cada segunda cada máquina realiza un mínimo de cinco peticiones al servidor. Esto multiplicado por cinco máquinas resulta algo sobrecogedor para el servidor.	A ciertos códigos de monitorización más secundarios se les añade un retraso. De esta forma se realizan cada 10-20 segundos y no cargan el servidor de forma excesiva.

8.2 Sistema resultante

8.2.1. Cliente

El ejecutable que el usuario afectado ha quedado como se muestra en la imagen. En la imagen se puede ver el proceso de autenticación, la recepción de un mensaje y la desconexión remota de la máquina.

Ilustración 37: Ejemplo de ejecución del cliente. (Fuente: squid-remote)

8.2.2. Interfaz

Página de acceso al sistema

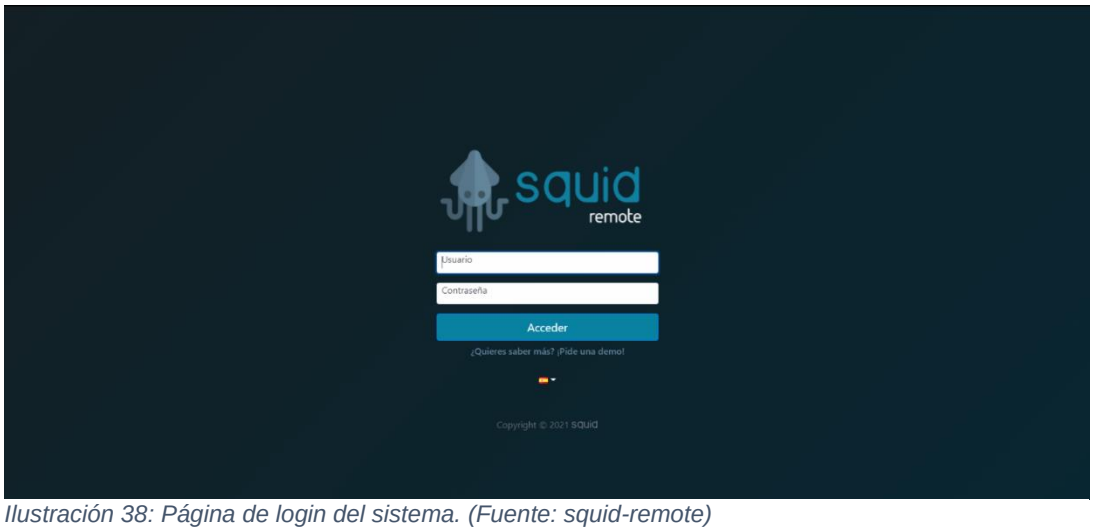


Ilustración 38: Página de login del sistema. (Fuente: squid-remote)

Página principal del sistema

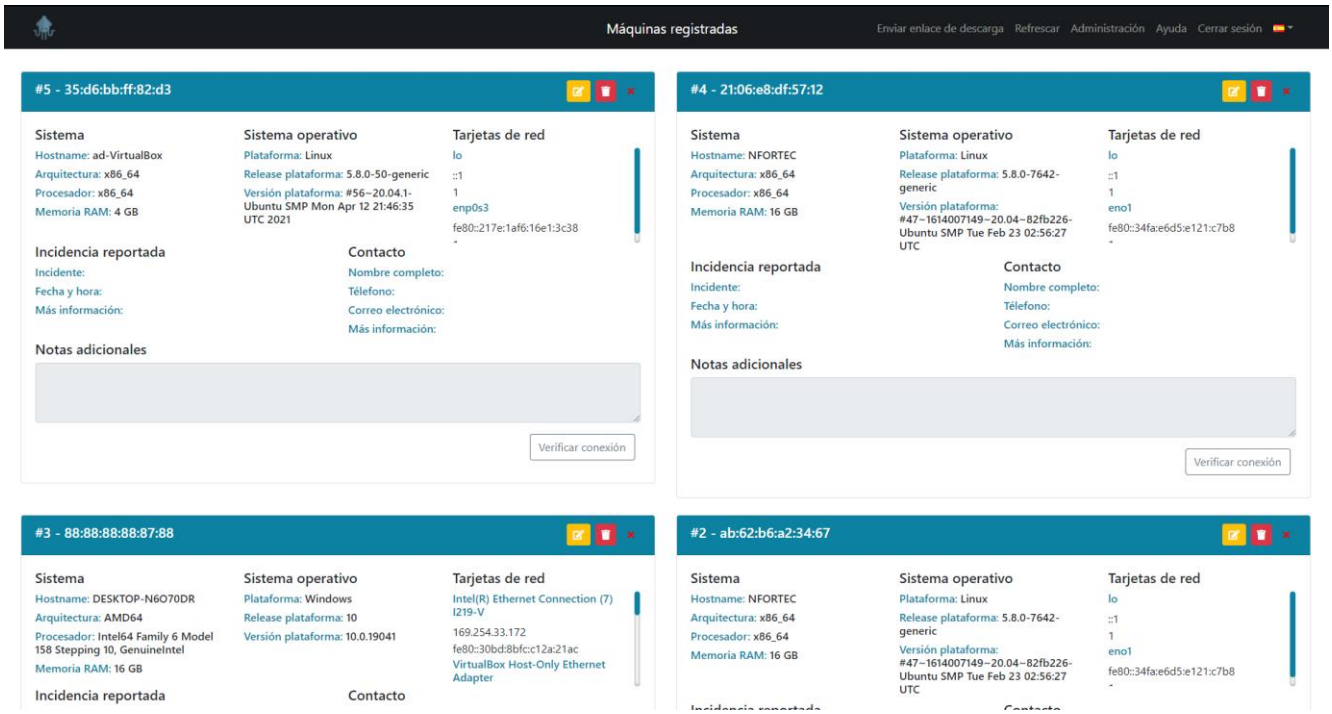


Ilustración 40: Pantalla principal del sistema. Muestra todas las máquinas registradas con su información. Desde la barra de navegación superior es posible enviar enlaces de descarga o acceder a la sección de administración. (Fuente: squid-remote)

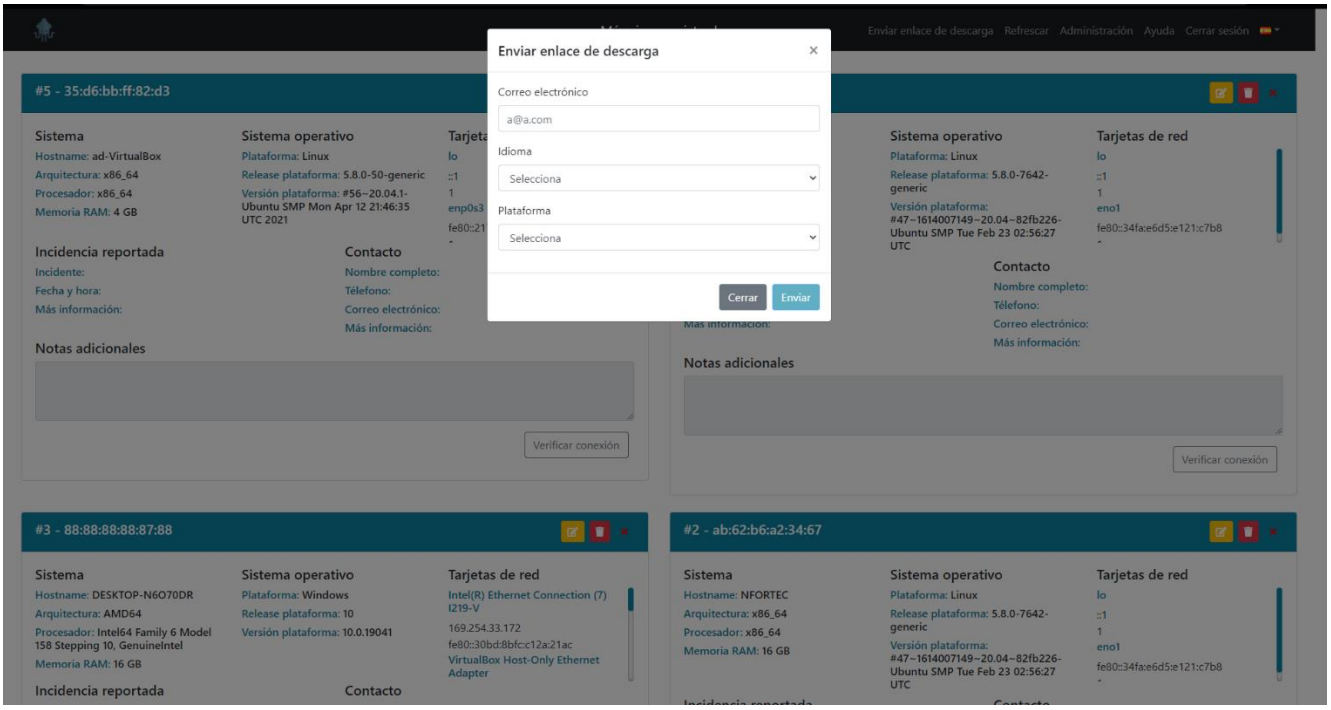


Ilustración 39: Ventana de creación de enlaces de descarga. Es posible seleccionar el idioma y el ejecutable que el usuario podrá descargar. (Fuente: squid-remote)

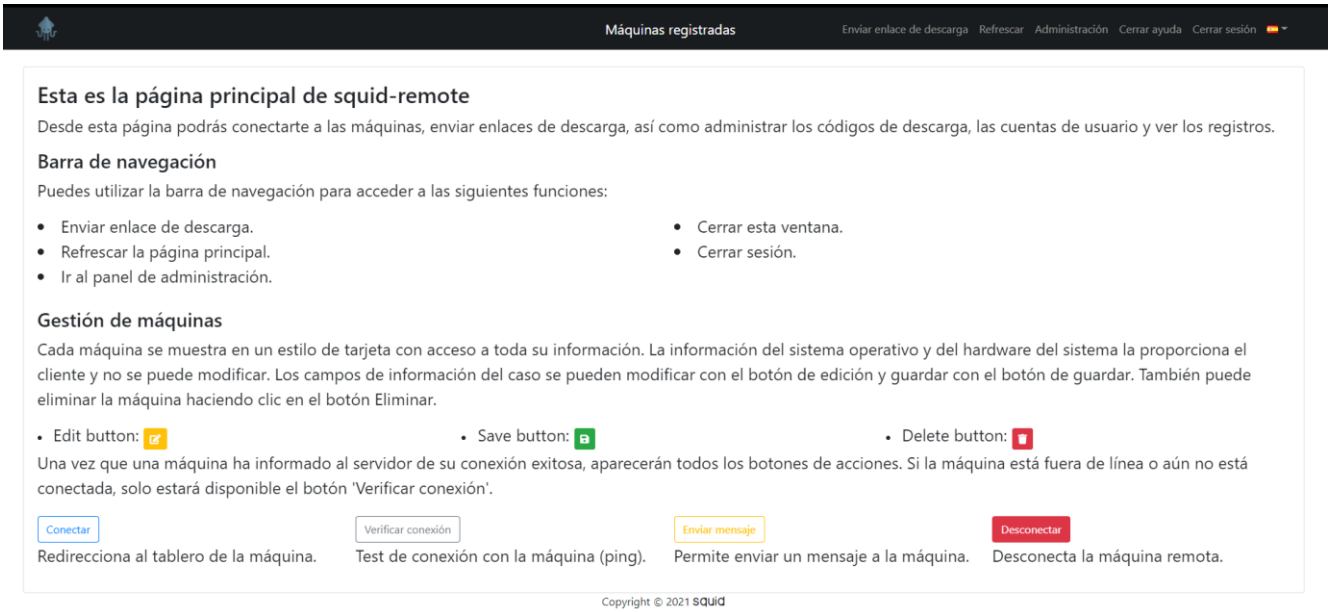


Ilustración 42: Página de ayuda de la pantalla principal. (Fuente: squid-remote)

Ilustración 41: Visualización de los controles de una máquina conectada desde la pantalla principal. (Fuente: squid-remote)

Panel de administración

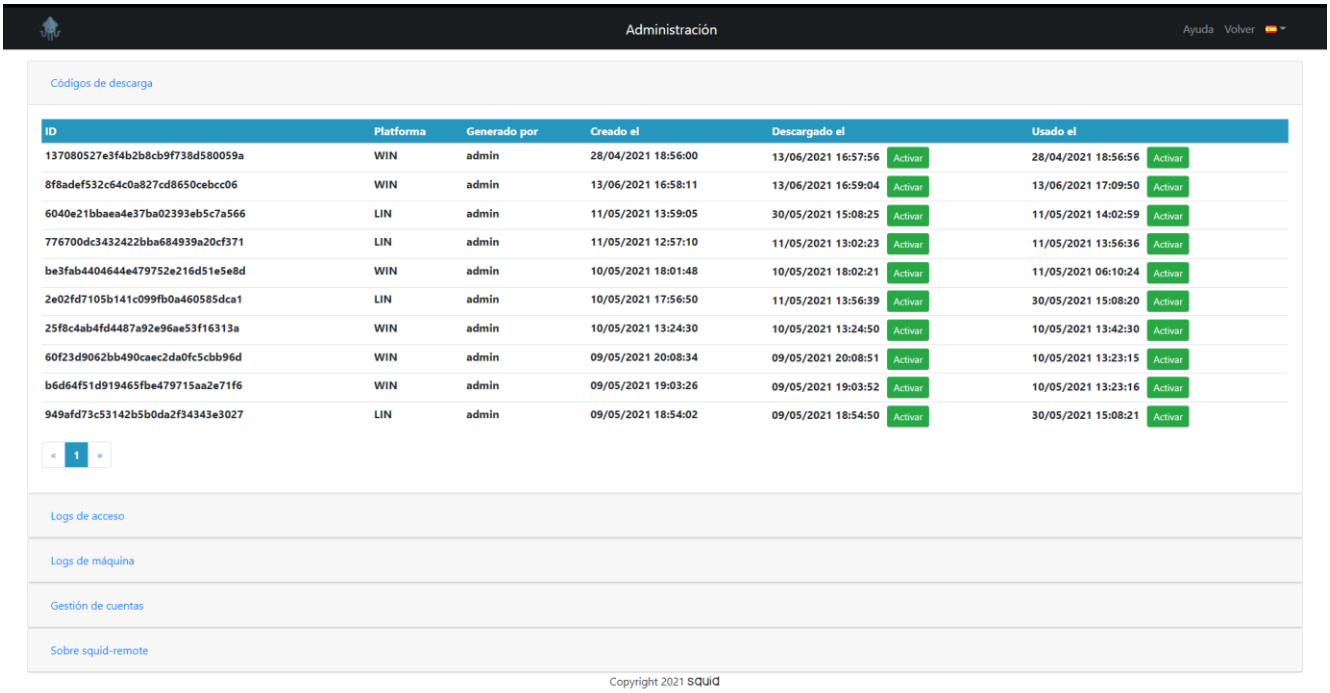


Ilustración 44: Sección de administración de códigos de descarga. El sistema permite reactivar su descarga o uso desde la sección de administración. (Fuente: squid-remote)

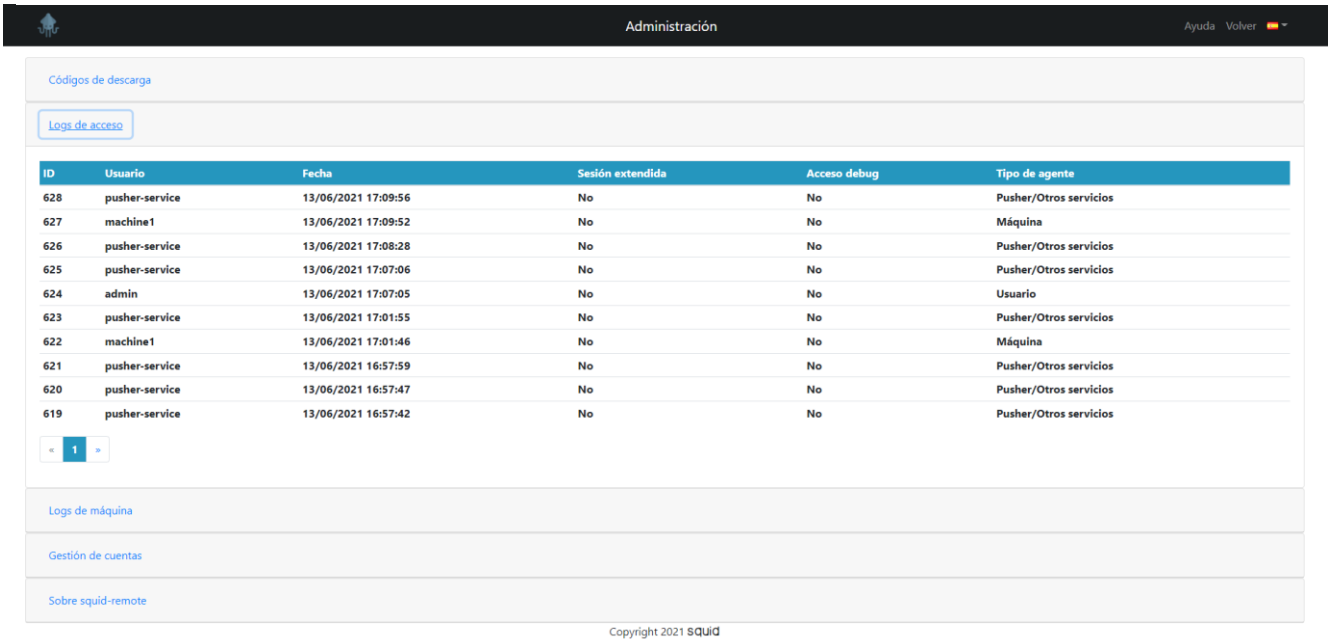


Ilustración 43: Visualización de los logs de acceso en la sección de administración.

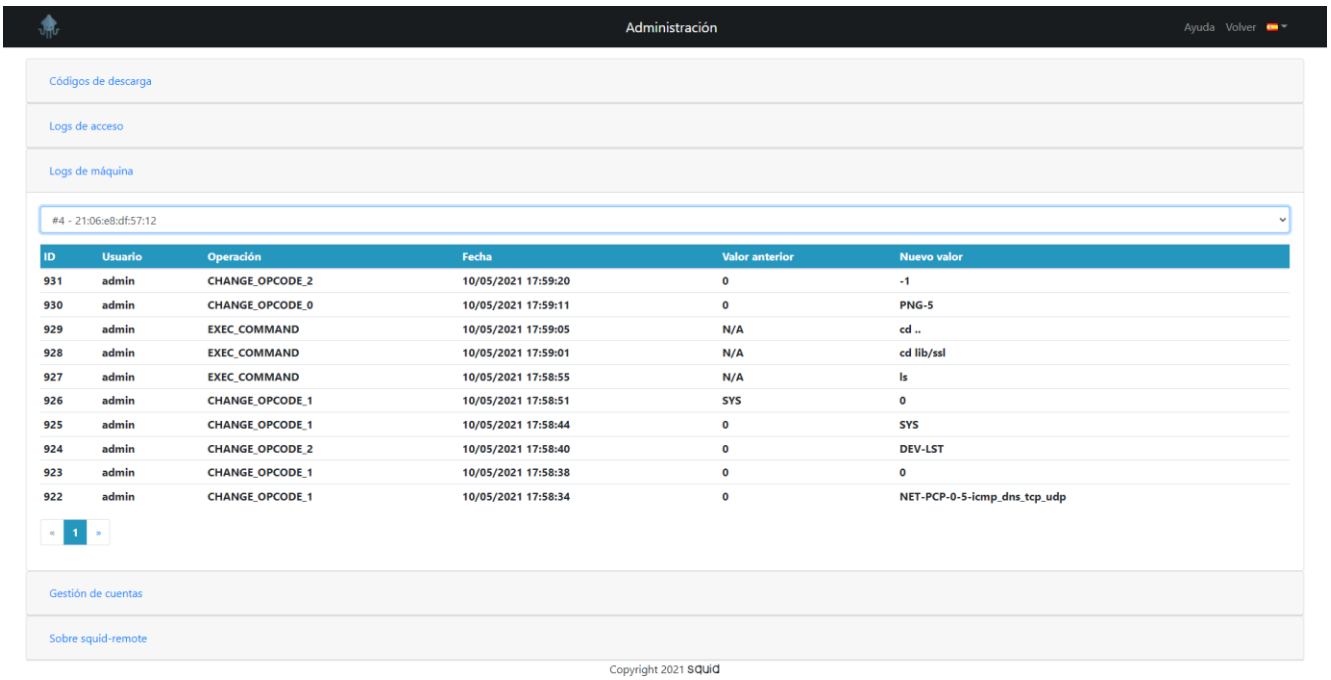


Ilustración 46: Visualización de los registros de operaciones sobre todas las máquinas. Toda operación realizada sobre una máquina queda registrada en el sistema. (Fuente: squid-remote)

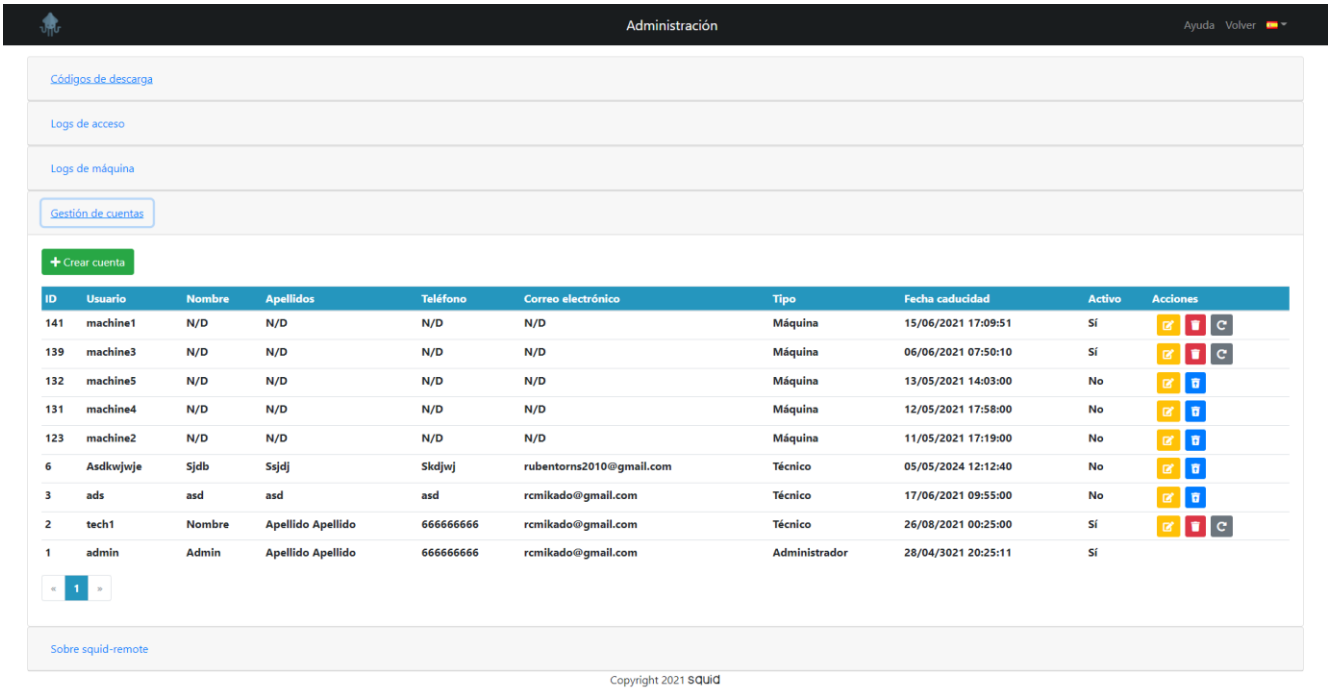


Ilustración 45: Sección de administración de cuentas. Es posible modificar los datos de cada cuenta, desactivarlas o generar una nueva contraseña. (Fuente: squid-remote)

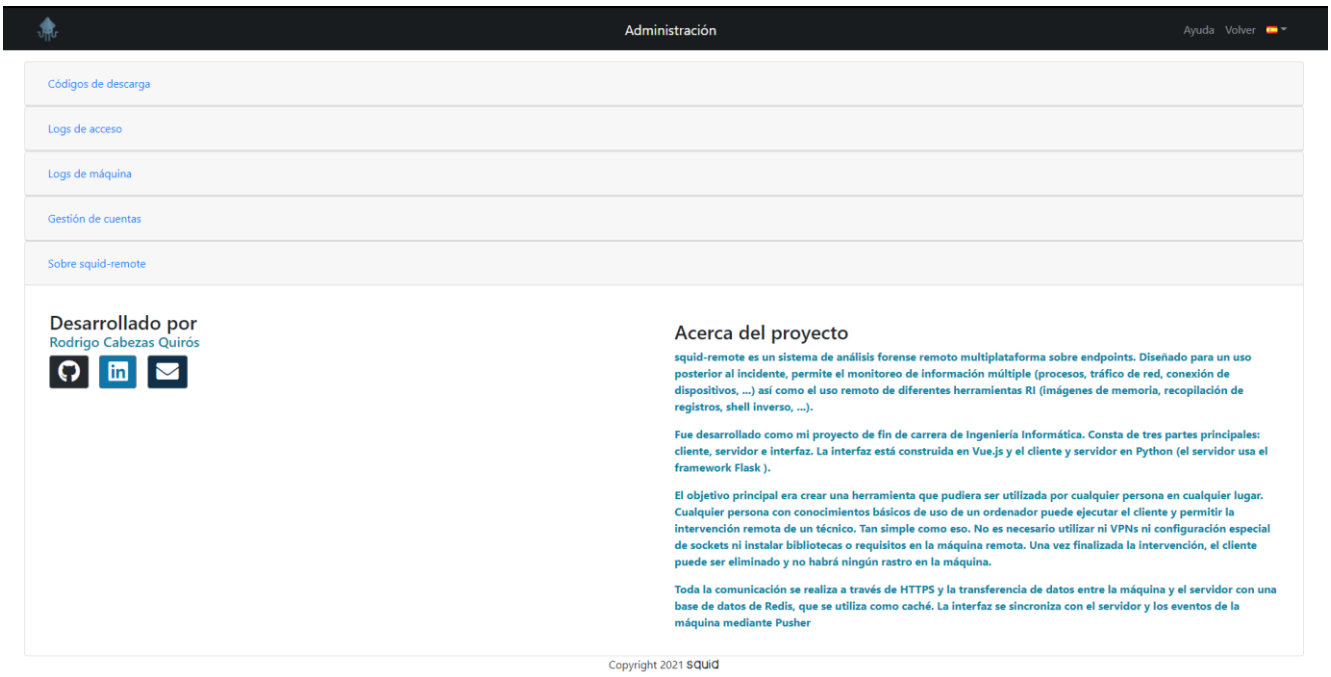


Ilustración 47: Información acerca del sistema, visible en la sección de administración. (Fuente: squid-remote)

Panel de control de máquina

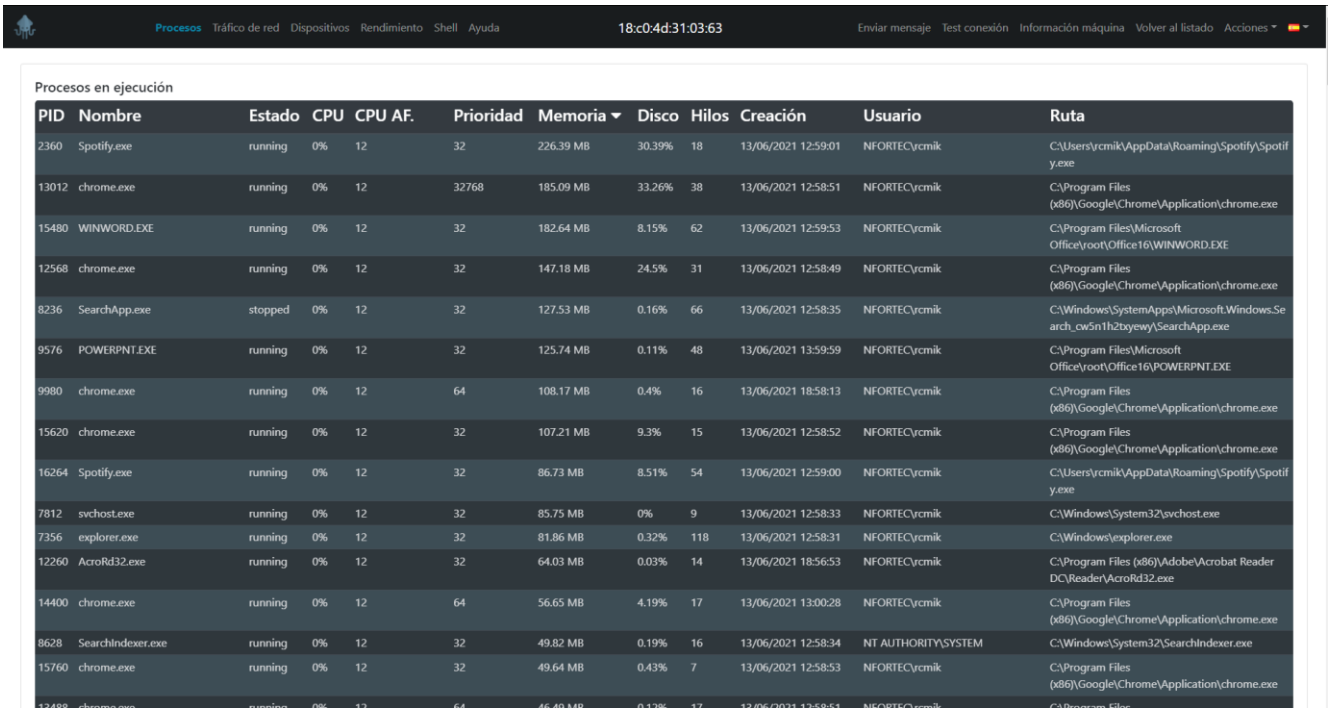


Ilustración 48: Vista de los procesos en ejecución en la máquina remota. Los procesos pueden ordenarse en función del valor de la columna clicando sobre esta. (Fuente: squid-remote)

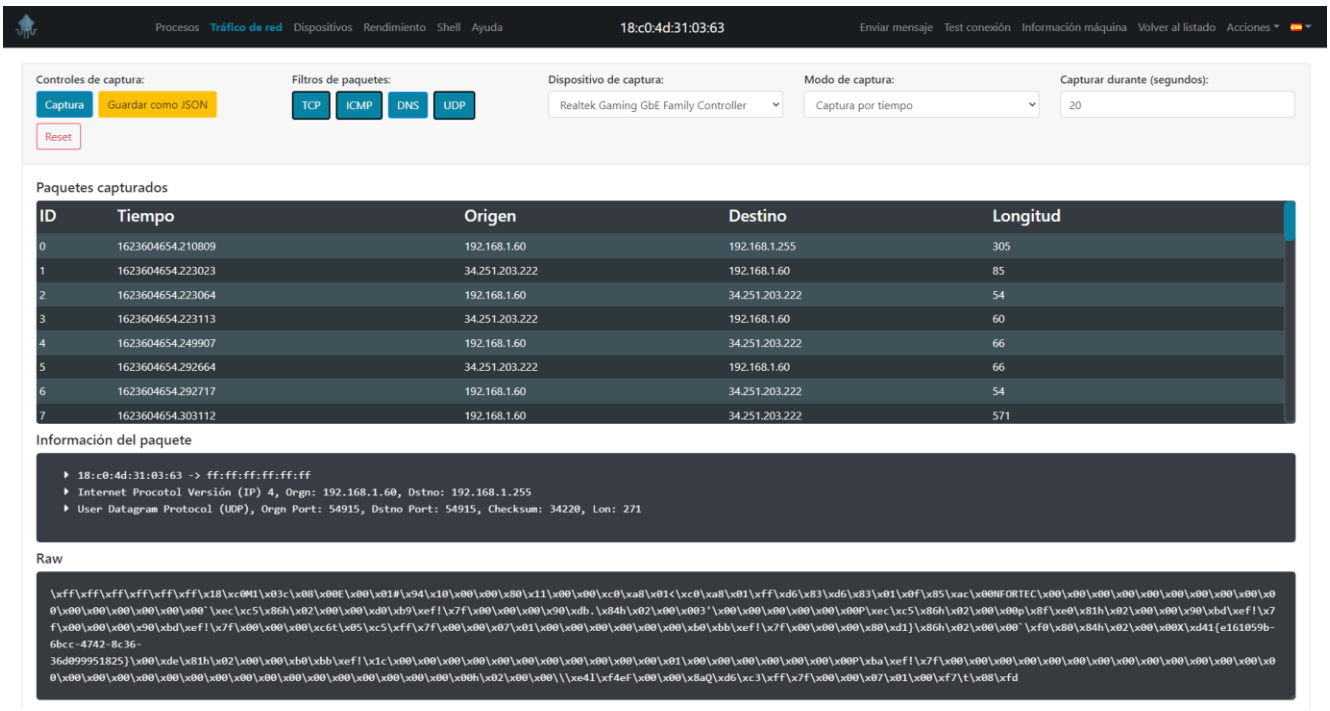


Ilustración 50: Funcionalidad de captura del tráfico de red. En la parte superior se encuentran los controles de captura, que permiten elegir tarjeta de red, modo de captura y filtros a aplicar. La primera tabla muestra los paquetes capturados, y en clicar sobre uno, se muestra en las dos últimas tablas información sobre el mismo. (Fuente: squid-remote)

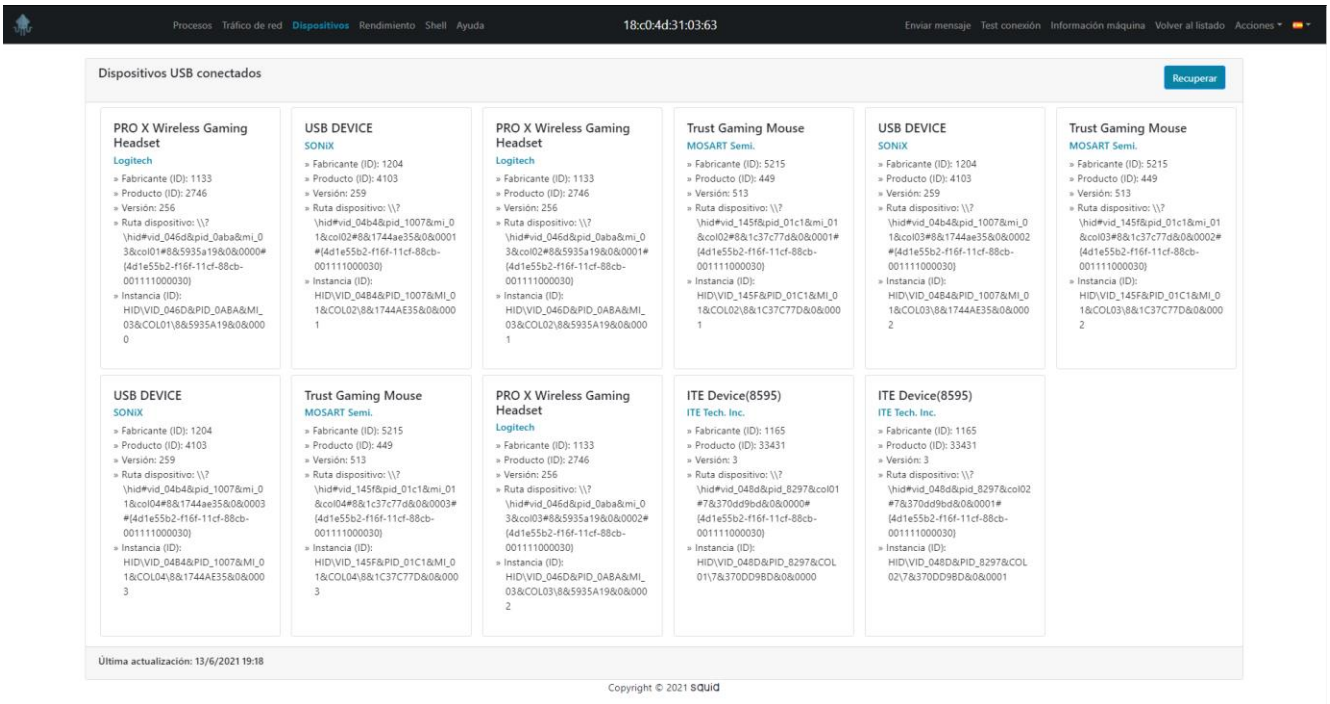


Ilustración 49: Sección de dispositivos conectados. Se genera el listado pulsando en el botón "Recuperar". (Fuente: squid-remote)

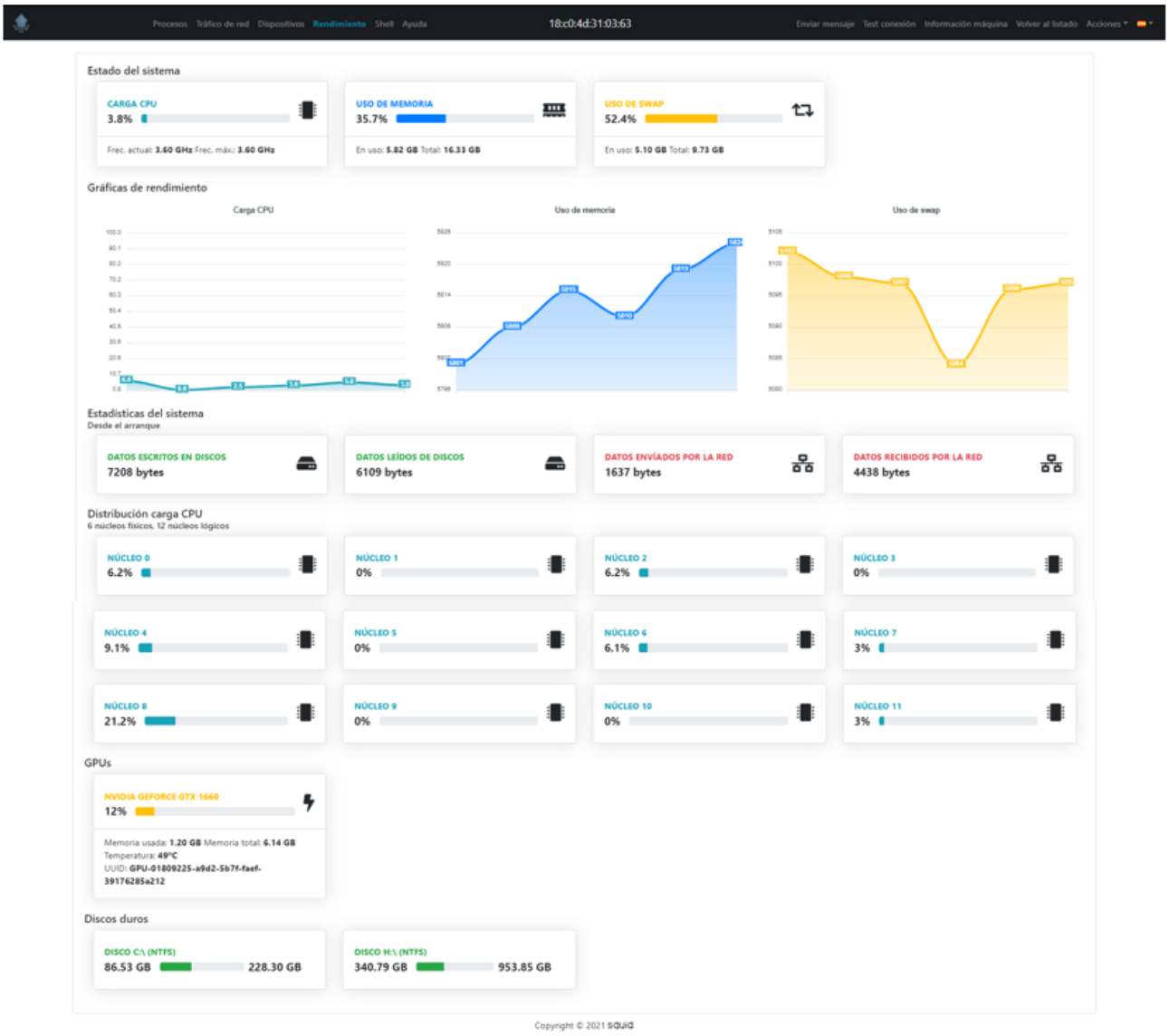


Ilustración 52: Sección de monitorización del rendimiento de la máquina. Se actualiza automáticamente y muestra el estado del sistema y sus recursos en tiempo real. (Fuente: squid-remote)

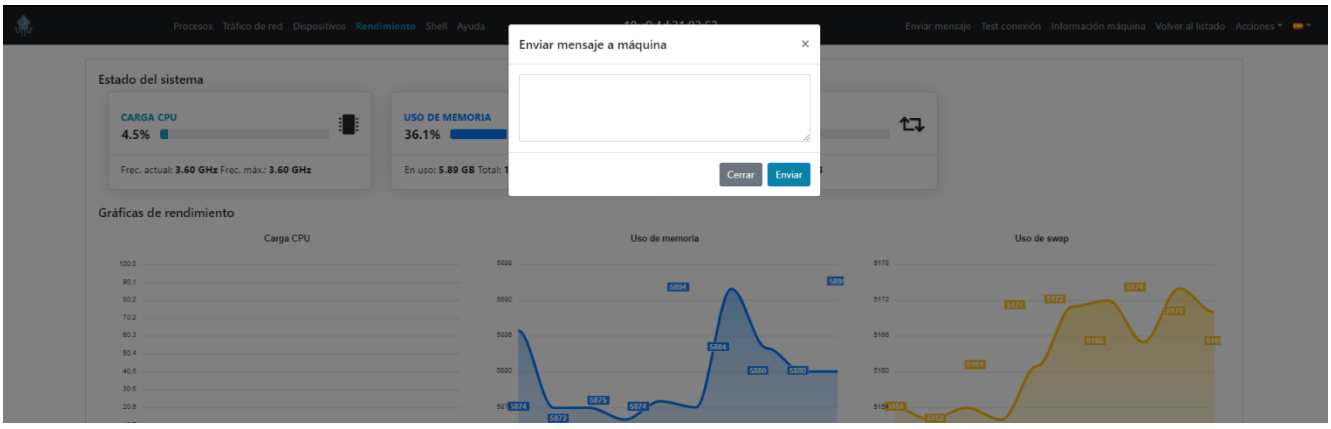


Ilustración 51: Opción de mandar un mensaje a la máquina remota. Está disponible para su uso en cualquier momento desde el panel de control de la máquina conectada. (Fuente: squid-remote)

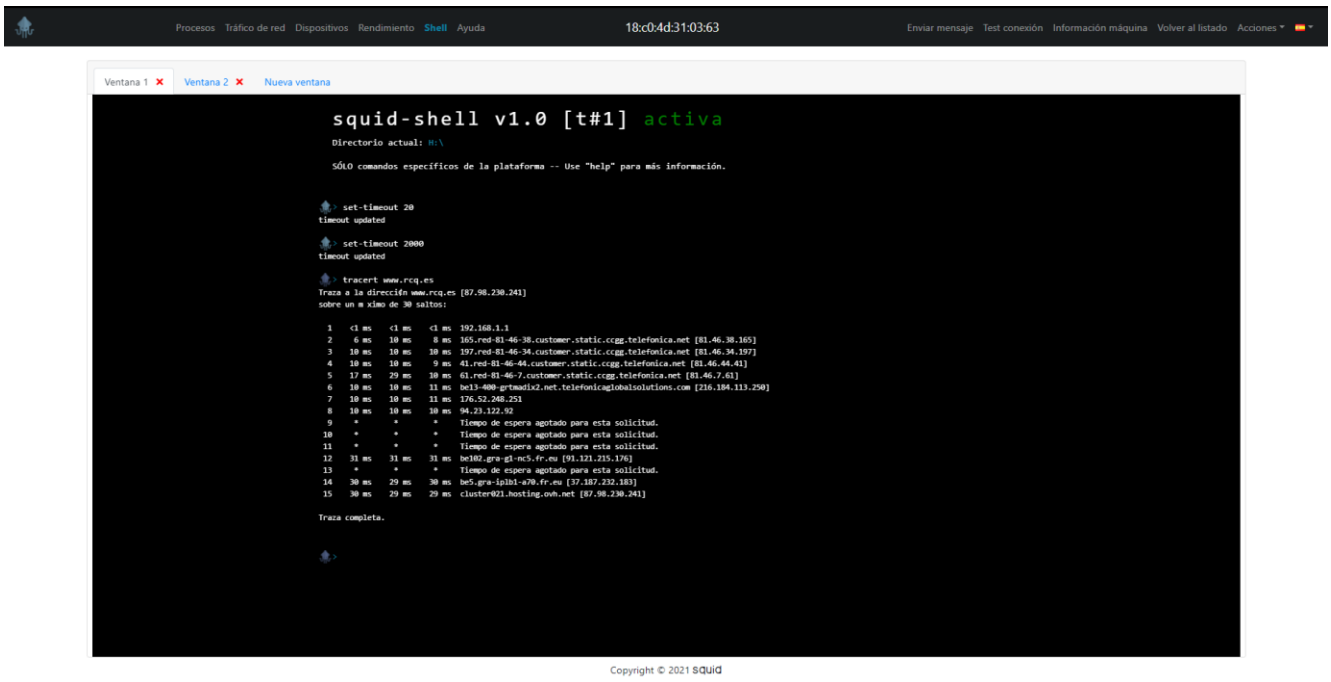


Ilustración 54: Funcionalidad de consola remota. En la imagen es posible ver dos terminales en ejecución. (Fuente: squid-remote)

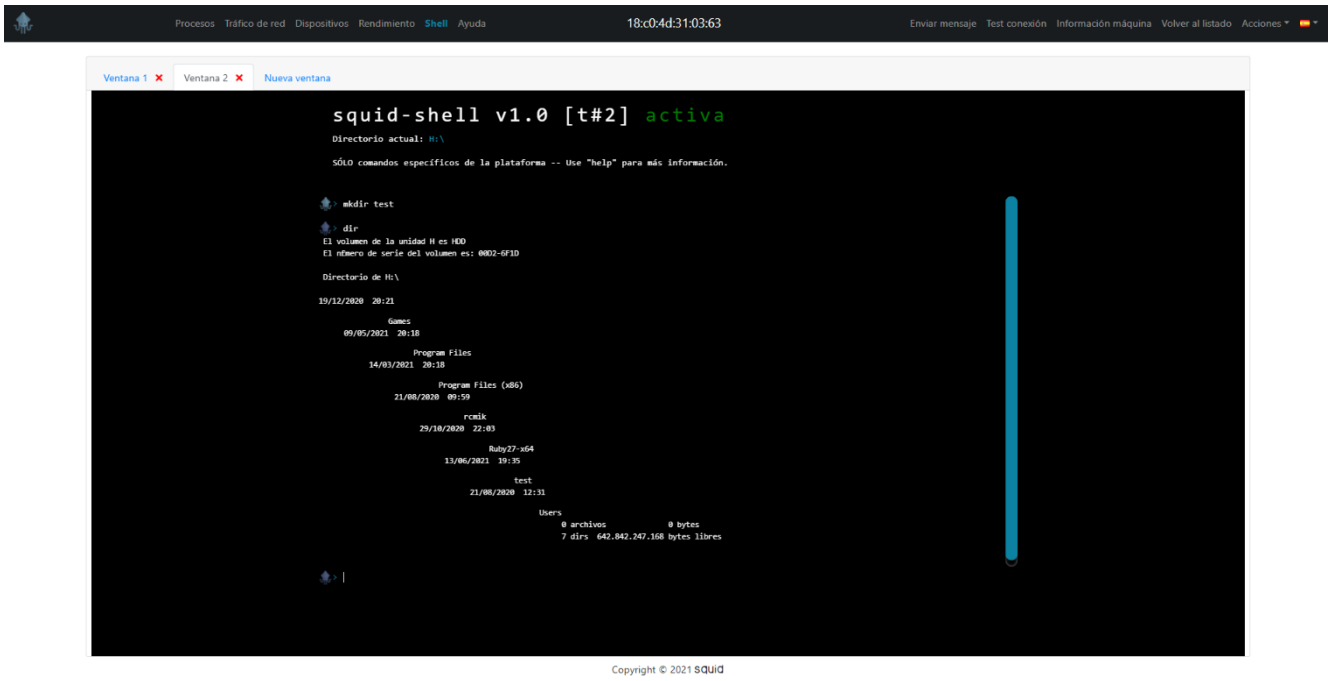


Ilustración 53: Funcionalidad de consola remota. Vista de una segunda sesión activa. (Fuente: squid-remote)

Ilustración 56: Página de ayuda del panel de control. (Fuente: squid-remote)

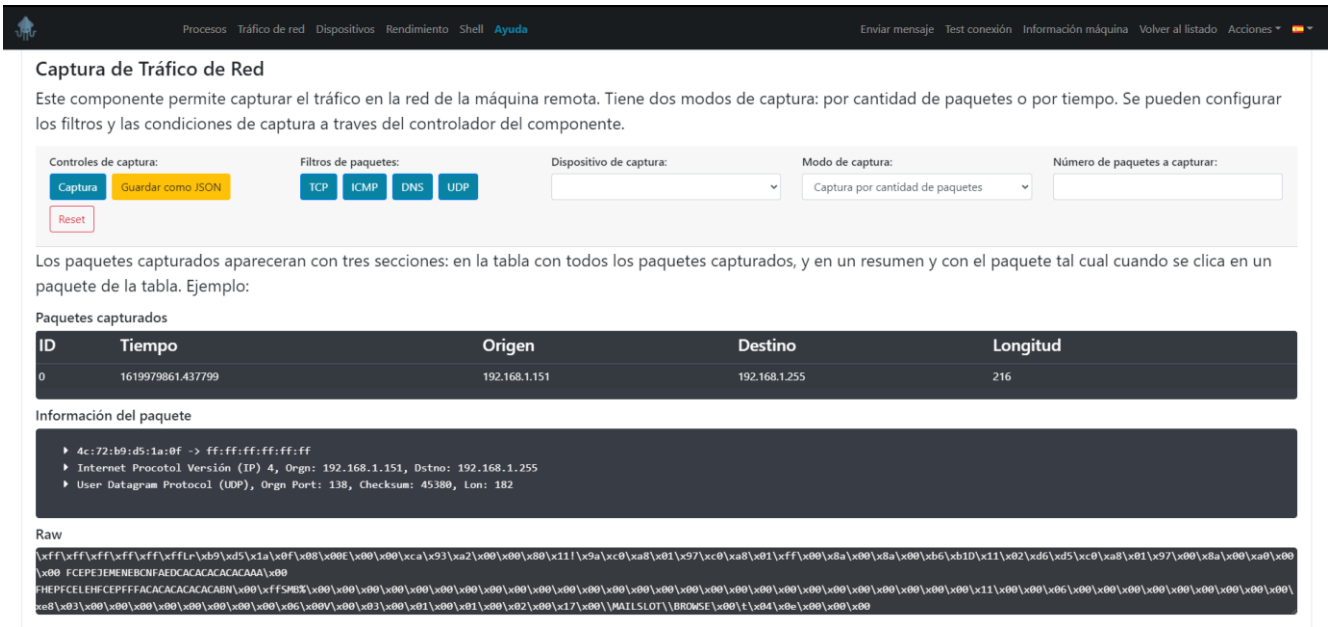


Ilustración 55: Página de ayuda del panel de control. (Fuente: squid-remote)

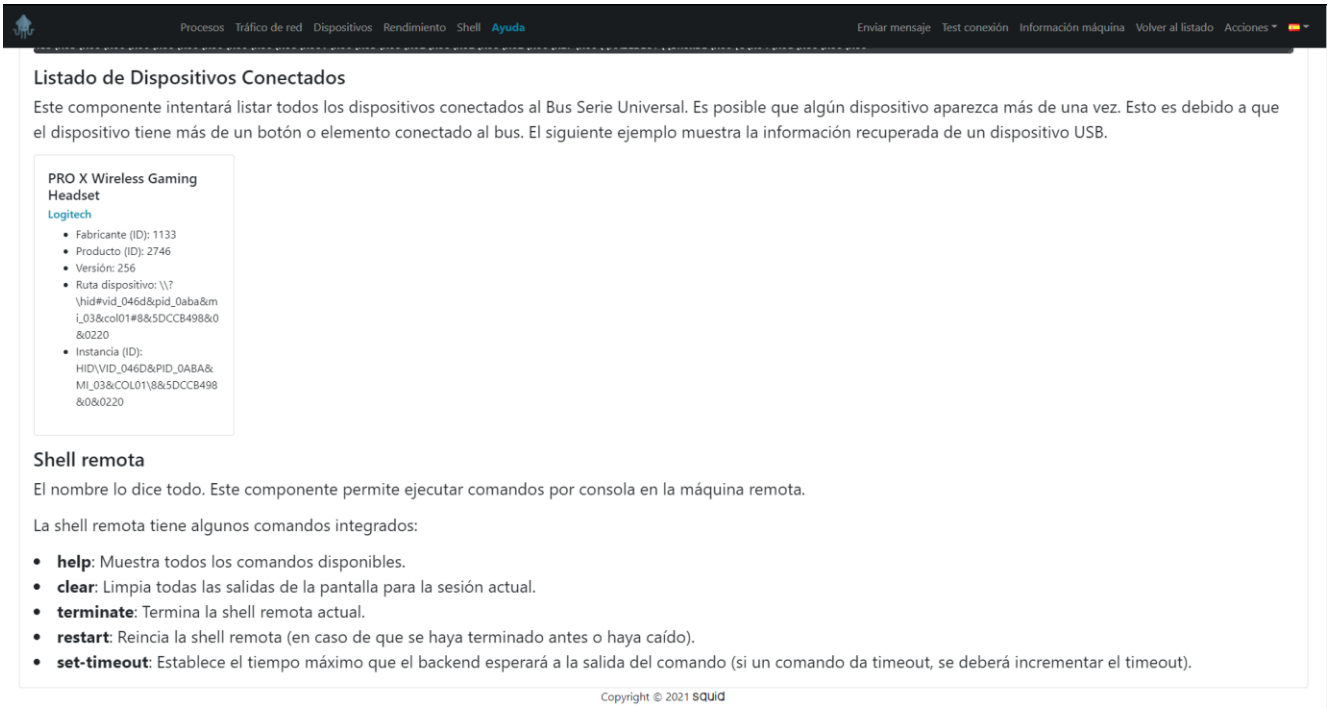


Ilustración 57: Página de ayuda del panel de control. (Fuente: squid-remote)

Ilustración 58: Desde la barra de navegación superior es posible acceder a la información de la máquina conectada en cualquier momento. (Fuente: squid-remote)

9 Costes

A nivel de software, los costes incluirían el coste del despliegue en Heroku y el servicio de integración continua de Travis CI. El primero, usado en el despliegue de producción (plan Hobby Dev de Heroku) tiene un coste de 5.78€ al mes. El entorno de desarrollo ha usado el plan gratuito por lo que no ha generado ningún gasto.

El servicio de integración continua también tiene un plan gratuito que proporciona 10000 créditos gratuitos, tras los cuales se debe contratar una cuota mensual de 69 dólares, 56.98€ al cambio. Se han hecho un total de 215 despliegues, contando todos los entornos. Cada despliegue ha requerido una media de un minuto y diez segundos de construcción. Cada minuto de uso en una máquina basada en Linux (la más barata) cuesta 10 créditos. Teniendo en cuenta todos estos factores, se han usado aproximadamente 2509 créditos que, al no pasar del cupo, tampoco han derivado en ningún coste.

Para poder calcular el coste de las horas de desarrollo, al comenzar el trabajo instalé un programa para hacer el seguimiento de las horas invertidas. En total, se ha contabilizado un total de 318 horas. Esta cantidad es aproximada, puesto que se han contado solo horas de programación en Windows, y no en Linux o las horas dedicadas a la investigación de errores o de lectura de documentación. Por lo que el total de horas se situaría más bien alrededor de las 408 horas.

Si el desarrollo lo hiciera un único ingeniero de software junior, como ha sido el caso, suponiendo que trabajando en una consultora y con un coste medio a la hora rondaría los 25€, el total en tiempo de desarrollo serían 10.200€. En caso de ser un ingeniero de software senior, el coste a la hora se sitúa entre los 45 y 90 euros, que elevarían el coste del proyecto a los 18.360€ – 36.720€. Con estas estimaciones se intenta conseguir una cifra genérica, por lo que están basadas en datos extraídos de internet, y excluyen factores como por ejemplo las horas extra (si aplicase).

En total, el desglose y coste total de desarrollo del proyecto es el siguiente:

	Concepto	Precio
	408 horas de desarrollo.	25€ / hora
	Suscripción durante 7 meses al plan Hobby Dev de Heroku.	5.78€ / mes
Total + IVA (21%)		**12.390,96€**

10 Trabajo futuro

Durante el transcurso del desarrollo del sistema han surgido algunas ideas nuevas y mejoras que podrían aplicarse en futuras versiones del mismo. Algunas de estas son las siguientes:

- Apagado y reinicio del equipo remoto: Está función sería útil en aquellas intervenciones en que tras realizar un cambio en la configuración del sistema se requiera un reinicio del equipo. En lugar de empezar toda la conexión de nuevo, se ordenaría el reinicio remoto del equipo y este al arrancar de nuevo se conectaría automáticamente al servidor.
- Modo oscuro en la interfaz: Muchos de los IDEs y aplicaciones web incorporan un modo oscuro que cambia o invierte los colores por unos con menos intensidad y que ayudan a reducir la fatiga visual.
- Ejecutable del cliente firmado: El archivo ejecutable del cliente no está firmado con un certificado reconocido por Microsoft. Esto supone que la protección *SmartScreen* de Windows impida su ejecución directa, requiriendo en su lugar que se ejecute desde un terminal elevado o con requisitos de administrador. El coste de estos certificados es muy elevado (300€ - 7000€) por lo que únicamente seria justificable si se llevará el sistema a una producción comercial.
- Desarrollo del cliente en C++: El cliente se ha desarrollado en Python, que pese ser rápido y fácil de desarrollar compromete la velocidad de ejecución. Una buena mejora seria migrar el código del cliente a C++, que ofrece unos ejecutables más livianos y una velocidad más alta.
- Implementar cuentas de organización: Esta mejora permitiría ofrecer squid-remote como un *Software as a service* (Saas). Cada organización tendría su propio espacio dentro del entorno de producción para sus cuentas de usuario y máquinas. El desarrollo de esta mejora está más enfocado hacia una explotación comercial del sistema.
- Soporte para gráficas AMD: La librería usada para la monitorización del rendimiento del equipo solo tiene en cuenta las tarjetas gráficas fabricadas por Nvidia. Como trabajo futuro podría ampliar el soporte a otros fabricantes.

11 Conclusiones

El objetivo principal del trabajo era diseñar un sistema capaz de permitir el análisis forense remoto de un equipo, mostrando la información crítica relativa a los procesos en ejecución, tráfico de red, dispositivos conectados y al rendimiento del equipo, y, en definitiva, squid-remote no solo cumple con los objetivos, sino que además excede toda expectativa inicial.

Se ha conseguido diseñar un sistema distribuido capaz de operar en equipos de cualquier plataforma, en cualquier lugar y sin ningún requisito de configuración especial sobre la red. También se ha logrado desarrollar una interfaz web sencilla, intuitiva, moderna y adaptable para el control de los equipos remotos y cuya calidad de detalle y calidad es comparable con proyectos profesionales.

Para satisfacer los requisitos definidos inicialmente se han usado las últimas tecnologías existentes, buscando la máxima seguridad en el sistema y la aplicación de los modelos de trabajo y desarrollos modernos que además de aportar robustez al sistema me han proporcionado experiencia. Todo ello sin comprometer las futuras líneas de trabajo por las que el sistema puede ser extendido.

Ha sido un proyecto que me ha permitido como estudiante demostrar y consolidar las capacidades para analizar, diseñar y desarrollar aplicaciones y sistemas de forma segura y eficiente. He podido también aprender a trabajar con una gran cantidad de tecnologías que he aplicado e integrado en el contexto del sistema aplicando paradigmas y arquitecturas vistos a lo largo del grado.

Por todo ello, se puede concluir que el proyecto ha resultado muy satisfactorio e interesante en todas sus iteraciones y ámbitos. Desde un punto de vista personal me ha aportado experiencia y nuevos conocimientos que puedo sumar al aprendizaje tecnológico y profesional. Por último y tras cumplir con expectativas, objetivos y requisitos definidos inicialmente, se puede dar por completado (no acabado, puesto que todavía hay posibles extensiones de trabajo futuras) este proyecto, que concluye el grado y marca el final de una etapa.

12 Referencias

Bootstrap. (s.f.). *Bootstrap Documentation.* Obtenido de https://getbootstrap.com/docs/4.1/getting-started/introduction/

Eddine, S. (s.f.). *vue-shell.* Obtenido de https://github.com/HalasProject/vue-shell#readme

guru99.com. (s.f.). *Flask vs Django: What's the Difference Between Flask & Django?* Obtenido de https://www.guru99.com/flask-vs-django.html#3

Hennion, N. (s.f.). *Glances - An eye on your system.* Obtenido de https://github.com/nicolargo/glances#readme

Hunt-Walker, N. (2 de Abril de 2018). *opensource.com.* Obtenido de An introduction to the Flask Python web app framework: https://opensource.com/article/18/4/flask

Kamluk, V. (s.f.). *Welcome to Bitscout.* Obtenido de https://bitscout-forensics.info/intro/what-is-bitscout

Kaspersky.com. (s.f.). *Kaspersky Lab Researcher Creates Free Software Tool for Collecting Remote Evidence After Cyber-Attacks.* Obtenido de https://www.kaspersky.com/about/press-releases/2017_kaspersky-lab-researcher-creates-free-software-tool-for-collecting-remote-evidence-after-cyber-attacks

Kazupon. (s.f.). *Vue I18n Documentation.* Obtenido de https://kazupon.github.io/vue-i18n/introduction.html

Microsoft. (25 de Mayo de 2021). *Windows Sysinternals.* Obtenido de https://docs.microsoft.com/en-us/sysinternals/

NMAP.ORG. (s.f.). *Guía de refenrencia de Nmap (Página de manual).* Obtenido de https://nmap.org/man/es/

OpenHardwareMonitor.org. (s.f.). *OpenHardwareMonitor Documentation.* Obtenido de https://openhardwaremonitor.org/

Packtpub.com. (s.f.). *HTTP reverse shell.* Obtenido de https://subscription.packtpub.com/book/networking_and_servers/9781788838979/1/ch01lvl1sec12/http-reverse-shell

palletsprojects.com. (s.f.). *Flask Documentation.* Obtenido de https://flask.palletsprojects.com/en/2.0.x/

Pusher Community. (s.f.). *What is Pusher?* Obtenido de https://pusher-community.github.io/real-time-laravel/introduction/what-is-pusher.html

Pusher.com. (s.f.). *Pusher Channels Documentation.* Obtenido de https://pusher.com/docs/channels/getting_started/javascript/?ref=docs-index

Raqet Project. (s.f.). *Raqet Project Documentation.* Obtenido de http://www.raqet.org/

readthedocs.io. (s.f.). *Pexpect version 4.8.* Obtenido de https://pexpect.readthedocs.io/en/stable/

readthedocs.io. (s.f.). *psutil documentation.* Obtenido de https://psutil.readthedocs.io/en/latest/

readthedocs.io. (s.f.). *Scapy Documentation.* Obtenido de https://scapy.readthedocs.io/en/latest/introduction.html

redis.io. (s.f.). *Redis Documentation.* Obtenido de https://redis.io/documentation

Rockikz, A. (Marzo de 2021). *How to Make a Process Monitor in Python.* Obtenido de https://www.thepythoncode.com/article/make-process-monitor-python

Sharma, A. (s.f.). *ptop - An awesome task manager written in python.* Obtenido de https://github.com/darxtrix/ptop

Shootback: herramienta en python para realizar un túnel TCP inverso. (25 de Febrero de 2017). Obtenido de https://www.hackplayers.com/2017/02/shootback-herramienta-tunel-tcp-inverso.html

SourceForge. (s.f.). *Pusher Beams vs Pusher Channels Comparison Chart.* Obtenido de https://sourceforge.net/software/compare/Pusher-Beams-vs-Pusher-Channels/

stackoverflow.com. (13 de Enero de 2021). *Why do i get a 503 in my front-end with SendGrid when email sends?* Obtenido de https://stackoverflow.com/questions/65498946/why-do-i-get-a-503-in-my-front-end-with-sendgrid-when-email-sends

Twilio SendGrid. (29 de Junio de 2020). *Sending Emails from Python Flask Applications With Twilio SendGrid.* Obtenido de https://sendgrid.com/blog/sending-emails-from-python-flask-applications-with-twilio-sendgrid/

Twilio SendGrid. (s.f.). *How to Send an SMTP Email.* Obtenido de https://sendgrid.com/docs/for-developers/sending-email/getting-started-smtp/

Vuejs.org. (s.f.). *Vue.js Docs.* Obtenido de https://v3.vuejs.org/guide/introduction.html

13 Anexos

Notas para la evaluación

Si el tribunal lo prefiere, puede disponer de acceso a los entornos en los que el sistema se encuentra desplegado y listo para su uso usando las siguientes URLs:

https://squid.rcq.es/

https://squid-remote-dev.herokuapp.com/

El segundo entorno es el de desarrollo, que, por cuestiones de redundancia, puede usarse como alternativa en caso de indisponibilidad del entorno de producción. Las siguientes cuentas otorgan acceso al sistema hasta el 31 de julio de 2021:

Tipo	Usuario	Contraseña
Administrador	admin2	1ds8RRe2Wv1mssb
Administrador	admin3	Jf6fR37TG4qxCX5
Técnico	tech2	PfVJBQGxnRjrVNU
Técnico	tech3	BaPAJeWyHRD1kRe

En caso que la evaluación quiera realizarse ejecutando el sistema de forma local, se requería una instalación y configuración completa del sistema. Esta última puede ahorrarse solicitando por correo electrónico el fichero de configuración de squid-remote tanto al tutor de este trabajo (raulroca@ub.edu) como a su autor (rcq@rcq.es).

La caducidad de las cuentas de usuario puede ser también extendida solicitándolo por correo electrónico al autor del trabajo.

Glosario

API: Una interfaz de programación de aplicación, *Application Programming Interface* por sus siglas en inglés, es un tipo de interfaz de software que ofrece un conjunto de protocolos y funciones a otras piezas de software y que se rige por unas especificaciones.

Backend: Parte de un sistema que se encarga de toda la lógica ajena al usuario y que se encarga por ejemplo de las comunicaciones con los servidores o del procesado de datos.

Cliente: Componente de hardware o software que se instala en un equipo y permite acceder a unos servicios o productos.

Consola: Tipo de interfaz de usuario de un ordenador que permite a los usuarios dar instrucciones a un programa o sistema operativo por medio de líneas de texto simples.

Diagrama de clases: Diagrama estático que describe la estructura de un sistema mostrando las clases del sistema junto con sus atributos, funciones y relaciones entre sí.

Framework: Conjunto de herramientas y módulos que definen e implementan funciones y partes de código que pueden ser usarse múltiples veces sin necesidad de repetir líneas de código.

Frontend: Componente de un sitio o aplicación web con la que un usuario interactúa.

HTTP / HTTPS: *Hypertext Transfer Protocol,* protocol de comunicación que permite la transferencia de información en internet. HTTPS es su homologo con medidas adicionales de seguridad.

JSON: Formato ligero de intercambio de datos que se caracteriza por su fácil compresión para los programadores y sencilla generación e interpretación para las máquinas.

Modelo (base datos): Tipo de modelo de datos que define la estructura lógica interna de una base de datos y esencialmente, la manera en que los datos se almacenan, organizan y manipulan.

RESTful: Conjunto de principios arquitectónicos por los que se pueden diseñar servicios web que principalmente centrados en los recursos de un sistema.

Software as a service: Modelo de distribución de software en que tanto el soporte lógico como los datos que se usan se alojan en una compañía y al que se accede desde internet mediante un cliente.

Unified Threat Management (UTM): Enfoque hacia la seguridad de la información en que un solo componente de software o hardware proporciona múltiples funciones de seguridad.

Guía de instalación

Para la ejecución del cliente como del servidor es necesaria la instalación de Python 3.9.5. Una vez Python está instalado, cada componente requiere la instalación de sus librerías. Esto se puede hacer de forma automática ejecutando el siguiente comando en los directorios de sendos componentes:

pip install -r localRequirements.txt

La interfaz requiere la instalación de Node.js y npm. Es probable que con la instalación del primero se instale el segundo conjuntamente. Tras la instalación de Node y npm, es necesario instalar las dependencias de la interfaz. Esto puede hacerse ejecutando el siguiente comando en el directorio de la interfaz:

npm install

Guía de configuración

La configuración del sistema se realiza en el fichero "*Config.py"* ubicado en el directorio raíz del servidor. Para que el sistema funcione correctamente es necesario disponer de una base de datos tipo SQL, una base de datos Redis, una cuenta en Sendgrid, una cuenta de correo electrónico, una cuenta de Pusher y una API key para IPStack, que se encarga de localizar las direcciones IP.

La lista de campos a rellenar en el fichero son los siguientes:

- **PUSHER_APP_ID**
- **PUSHER_KEY**
- **PUSHER_SECRET**
- **PUSHER_CLUSTER**
- **MAIL_SERVER**
- **MAIL_PORT**
- **MAIL_USERNAME**
- **MAIL_PASSWORD**
- **MAIL_DEFAULT_SENDER**
- **REDIS_HOST**
- **REDIS_PORT**
- **REDIS_PASSWORD**
- **REPLY_EMAIL**
- **IPSTACK_API_KEY**
- **SECRET_KEY** (DevelopmentConfig)
- **SQL_ALCHEMY_DATABASE_URI** (Si se usa una base de datos propia)

Si se dispone del fichero de configuración, tan solo es necesario sustituirlo por el existente en el directorio.

En caso de tratarse de una instalación limpia desde cero, para crear las primeras cuentas, incluyendo la de administrador, puede usarse el script ubicado en el fichero "*setup.py*". Este script creará las cuentas de usuario manualmente en base de datos a partir de los parámetros especificados para cada una. Se recomienda eliminar este fichero tras su uso.

Comandos para la ejecución local

- Cliente:
  python main.py –debug
- Servidor:
  python app.py
- Interfaz:
  vue-cli-service serve --mode debug

La interfaz no necesita ser ejecuta como tal, puesto que sus ficheros compilados ya se encuentran junto con el servidor. En acceder a la dirección proporcionada por el servidor tras su ejecución, la interfaz será cargada como si del entorno desplegado se tratara.

Listado de valores de operación en registro

- **CHANGE_OPCODE_[X]**: Cambio de el código de operación X en la máquina indicada.
- **CREATE_MACHINE:** Registro en el sistema de una máquina nueva.
- **UPDATE_MACHINE:** Actualización de los datos de una máquina existente en el sistema.
- **EXEC_COMMAND:** Ejecución de un comando por la consola remota de la máquina indicada.
- **UPDATE_INFO:** Actualización de los datos de caso de una máquina del sistema.

Listado de códigos de operación

- **PNG-5:** Ejecución de operación de Ping.
- **NET-LST-X-X-X:** Devolver listado de tarjetas de red del equipo.
- **CMD-NEW-[X]:** Creación de sesión de consola remota con identificador X.
- **PRC:** Devolver información de los procesos en ejecución.
- **NET-PCP-[X1]-[X2]-[X3]:** Captura de tráfico de red sobre la tarjeta de red con identificador X1 durante hasta un máximo de X2 paquetes, aplicando los filtros X3 (cuyos valores pueden ser una combinación de icmp, dns, tcp o udp con una separación del carácter "_" entre cada uno).
- **NET-TCP-[X1]-[X2]-[X3]:** Igual que el anterior pero con captura de tiempo durante X2 segundos.
- **DEV-LST:** Devolver listado de dispositivos conectados.
- **SYS:** Devolver información de rendimiento del sistema.
- **CMD-FIN-[X]:** Terminar la ejecución de la consola remota con identificador X.
- **CMD-TER-X:** Terminar la ejecución de todas las consolas remotas.
- **CMD-RES-[X]:** Reiniciar la consola remota con identificador X.

Manual de usuario

Gestión de máquinas registradas:

Cada máquina se muestra en un estilo de tarjeta con acceso a toda su información. La información del sistema operativo y del hardware del sistema la proporciona el cliente y no se puede modificar. Los campos de información del caso se pueden modificar con el botón de edición y guardar con el botón de guardar. También puede eliminar la máquina haciendo clic en el botón Eliminar.

- Edit button:
- Save button:
- Delete button:

Una vez que una máquina ha informado al servidor de su conexión exitosa, aparecerán todos los botones de acciones. Si la máquina está fuera de línea o aún no está conectada, solo estará disponible el botón 'Verificar conexión'.

Redirecciona al tablero de la máquina.

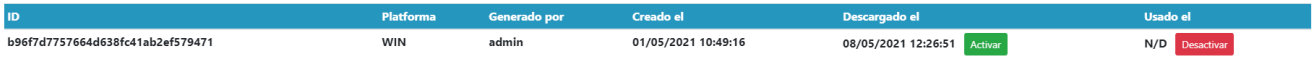
Test de conexión con la máquina (ping).

Permite enviar un mensaje a la máquina.

Desconecta la máquina remota.

Gestión de códigos de descarga

Los códigos de descarga actúan como clave de acceso en una conexión del cliente. Se envían por correo electrónico a través de la página principal. Cada código está vinculado a un correo electrónico, que debe introducirse en la máquina remota para autorizar la conexión. Si el correo electrónico del código de descarga y el introducido no coinciden, la conexión es rechazada. Cada código permite una descarga de cliente para la plataforma dada y una conexión. En esta sección se muestra el registro de cuándo se usaron los códigos. Desde esta sección también puede activar o desactivar los privilegios de descarga y de conexión de cada código.



Registros de sistema

Se mantienen dos tipos de registros: registros de inicio de sesión y registros de máquina. Los registros de inicio de sesión mantienen un registro de todos los usuarios / servicios que se han autenticado con la aplicación (esto incluye máquinas y Pusher). El registro de máquinas mantiene un registro de todos las acciones realizadas sobre una máquina específica.

Gestión de cuentas de usuario

Cada cuenta tiene una fecha de vencimiento, después de la cual el usuario pierde la capacidad de iniciar sesión. Hay tres tipos de cuentas:

- **Cuenta de administrador:** Rol maestro. Caduca a los 1000 años.
- **Cuenta de técnico:** Similar al administrador, pero sin la capacidad de administrar cuentas. Caduca a los 3 años.
- **Cuenta demo:** Igual que un técnico. Caduca 2 días después de la creación.

Se pueden realizar diferentes acciones para una cuenta determinada (excepto en cuentas de administrador):

- Editar la información de la cuenta y la fecha de vencimiento.
- Desactivar una cuenta.
- Reactivar una cuenta.
- Regenerar la contraseña de la cuenta (una nueva enviada por correo electrónico).
- Desactivar una cuenta.

Estas acciones se pueden realizar utilizando los siguientes botones (que se adjuntan a cada fila):


Botón editar


Botón guardar


Botón reactivar cuenta


Botón descativar cuenta


Botón de regenerar contraseña

Panel de control

Parte superior izquierda

- Monitorización de procesos.
- Captura de tráfico de red.
- Listado de dispositivos conectados.
- Monitor de rendimiento del sistema.
- Shell remota.

Parte superior derecha

- Enviar mensaje a máquina remota.
- Test de conexión con la máquina.
- Información de la máquina y del caso.
- Volver a listado de máquinas.
- Botón de acciones (desconectar).

Monitorización de procesos

La monitorización de procesos mostrará la siguiente información de cada proceso que se esté ejecutando en la máquina remota:

PID	Nombre	Estado	CPU	CPU AF.	Prioridad	Memoria	Disco	Hilos	Creación	Usuario	Ruta

- **PID**: Identificador de proceso.
- **Nombre**: Nombre del proceso.
- **Estado**: Estado de ejecución actual (Running, Stopped).
- **CPU**: Uso en % de la CPU.
- **CPU Af.**: Cantidad de CPUs que el proceso tiene permitido usar.
- **Prioridad**: La prioridad que tiene el proceso.
- **Memoria**: Uso de memoria actual.
- **Disco**: Uso en % de discos.
- **Hilos**: Número de hilos creados por el proceso.
- **Creación**: Fecha y hora de creación del proceso.
- **Usuario**: Usuario que instanció el proceso.
- **Ruta**: Directorio del proceso.

Captura del tráfico de red

Este componente permite capturar el tráfico en la red de la máquina remota. Tiene dos modos de captura: por cantidad de paquetes o por tiempo. Se pueden configurar los filtros y las condiciones de captura a traves del controlador del componente.

Controles de captura: Captura | Guardar como JSON | Reset

Filtros de paquetes: TCP | ICMP | DNS | UDP

Dispositivo de captura:

Modo de captura: Captura por cantidad de paquetes

Número de paquetes a capturar:

Los paquetes capturados apareceran con tres secciones: en la tabla con todos los paquetes capturados, y en un resumen y con el paquete tal cual cuando se clica en un paquete de la tabla. Ejemplo:

Paquetes capturados

ID	Tiempo	Origen	Destino	Longitud
0	1619979861.437799	192.168.1.151	192.168.1.255	216

Información del paquete

```
▸ 4c:72:b9:d5:1a:0f -> ff:ff:ff:ff:ff:ff
▸ Internet Procotol Versión (IP) 4, Orgn: 192.168.1.151, Dstno: 192.168.1.255
▸ User Datagram Protocol (UDP), Orgn Port: 138, Checksum: 45380, Lon: 182
```

Raw


```
\xff\xff\xff\xff\xff\xffLr\xb9\xd5\x1a\x0f\x08\x00E\x00\x00\xca\x93\xa2\x00\x00\x80\x11!\x9a\xc0\xa8\x01\x97\xc0\xa8\x01\xff\x00\x8a\x00\x8a\x00\xb6\xb1D\x11\x02\xd6\xd5\xc0\xa8\x01\x97\x00\x8a\x00\xa0\x00
\x00 FCEPEJEMENEBCNFAEDCACACACACACAAA\x00
FHEPFCELEHFCEPFFFACACACACACACABN\x00\xffSMB%\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x11\x00\x00\x06\x00\x00\x00\x00\x00\x00\x00\x00\x00\
xe8\x03\x00\x00\x00\x00\x00\x00\x00\x00\x06\x00V\x00\x03\x00\x01\x00\x01\x00\x02\x00\x17\x00\\MAILSLOT\\BROWSE\x00\t\x04\x0e\x00\x00\x00
```


Listado de dispositivos conectados

Este componente intentará listar todos los dispositivos conectados al Bus Serie Universal. Es posible que algún dispositivo aparezca más de una vez. Esto es debido a que el dispositivo tiene más de un botón o elemento conectado al bus. El siguiente ejemplo muestra la información recuperada de un dispositivo USB.

PRO X Wireless Gaming Headset

Logitech

- Fabricante (ID): 1133
- Producto (ID): 2746
- Versión: 256
- Ruta dispositivo: \\?\hid#vid_046d&pid_0aba&mi_03&col01#8&5DCCB498&0&0220
- Instancia (ID): HID\VID_046D&PID_0ABA&MI_03&COL01\8&5DCCB498&0&0220

Consola remota

El nombre lo dice todo. Este componente permite ejecutar comandos por consola en la máquina remota.

La shell remota tiene algunos comandos integrados:

- **help**: Muestra todos los comandos disponibles.
- **clear**: Limpia todas las salidas de la pantalla para la sesión actual.
- **terminate**: Termina la shell remota actual.
- **restart**: Reincia la shell remota (en caso de que se haya terminado antes o haya caído).
- **set-timeout**: Establece el tiempo máximo que el backend esperará a la salida del comando (si un comando da timeout, se deberá incrementar el timeout).

Documentación API

AccountsList			
Descripción	Devuelve listado de todas las cuentas existentes en el sistema		
Endpoint	/api/accounts	Roles	Admin
Body	N/A		
GET			
200	{“accounts”: […]}		
401	{“message: “Unauthorized Access”}		

RegenerateAccountPassword			
Descripción	Genera una nueva contraseña para una cuenta de usuario.		
Endpoint	/api/account/<int:idd>/password/regen	Roles	Admin
Body	N/A		
POST			
200	{“message”: “Email sent correctly”}		
500	{“message”: “Something went wrong”}		
404	{“message”: “User not found”}		
401	{“message: “Unauthorized Access”}		

AccountManagement			
Descripción	Gestión de las cuentas de usuario.		
Endpoint	/api/account/<int:idd> /api/account	Roles	Admin
Body	{“fname”: “str”, “lname”: “str”, “email”: “str”, “phone”: “str”, “expiration_date”: “str”, “active”: “bool”}		
POST			
409	{"message": "User with given username already exists"}		
200	{"message": "Email sent correctly"}		
201	{"message": "Account created ok"}		
500	{'message': 'Something went wrong'}		
401	{“message: “Unauthorized Access”}		
PUT			
403	{"message": "Not allowed to change admin account parameters"}		
200	{"message": "Account updated ok"}		
404	{"message": "Account not found"}		
401	{“message: “Unauthorized Access”}		

DELETE			
200	{"message": "Account deactivated ok"}		
404	{"message": "Account not found"}		
401	{“message: “Unauthorized Access”}		

Auth			
Descripción	Login para los usuarios.		
Endpoint	/api/auth	Roles	Todos
Body	{“username”: “str”, “password”: “str”}		
POST			
401	{"message": "Account has expired"}		
200	{"token": “…”, "id": <int>, "type": <int>}		
400	{"message": "Incorrect password"}		
404	{"message": "Account not found"}		

DebugAuth			
Descripción	Login para el desarrollo, omite triple autenticación.		
Endpoint	/api/debug/auth	Roles	Admin
Body	{“username”: “str”, “password”: “str”, “mac”: “str”}		
POST			
200	{"uname": “str”, "pass": “str”, "account_id": <int>}		
400	{"message": "Invalid request"}		
404	{"message": "Account not found"}		

PusherAuth			
Descripción	Login para el servicio de Pusher.		
Endpoint	/api/pusher/auth	Roles	Admin, Técnico
Body	{“channel_name”: “str”, “socket_id”: “str”}		
POST			
200	{"auth ": “str”}		

ExtendSession			
Descripción	Extiende la sesión de un usuario.		
Endpoint	/api/session/extend	Roles	Todos
Body	{“id”: <int>, “token”: “str”}		
POST			
409	{"message": "Token does not match, session can't be extended"}		

200	{"token": "…", "id": <int>, "type": <int>}
410	{"message": "Token is expired and is no longer valid"}
404	{"message": "Account not found"}
401	{“message: “Unauthorized Access”}

HardwareMonitorReceiver			
Descripción	Intercambio de datos de rendimiento del equipo.		
Endpoint	/api/machine/<string:mac>/hardware_monitor	Roles	GET: Admin, Técnico POST: Máq.
Body	{“data”: {…}}		
GET			
200	{“data”: {…}}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		
POST			
403	{"message": "Action forbidden"}		
200	{"message": "Data received"}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		

NetworkTrafficReceiver			
Descripción	Intercambio de datos del tráfico capturado.		
Endpoint	/api/machine/<string:mac>/traffic	Roles	GET: Admin, Técnico POST: Máq.
Body	{“traffic”: […]}		
GET			
200	{“traffic”: […]}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		
POST			
403	{"message": "Action forbidden"}		
200	{"message": "Data received"}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		

DevicesInfoReceiver			
Descripción	Intercambio de datos de dispositivos conectados.		
Endpoint	/api/machine/<string:mac>/devices	Roles	GET: Admin, Técnico POST: Máq.
Body	{“devices”: […]}		
GET			
200	{“devices”: […]}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		
POST			
403	{"message": "Action forbidden"}		
200	{"message": "Data received"}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		

NetworkInterfacesListReceiver			
Descripción	Intercambio de datos de tráfico de red capturado.		
Endpoint	/api/machine/<string:mac>/ifaces	Roles	GET: Admin, Técnico POST: Máq.
Body	{“ifaces”: […]}		
GET			
200	{“ifaces”: […]}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		
POST			
403	{"message": "Action forbidden"}		
200	{"message": "Data received"}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		

ProcessInfo			
Descripción	Intercambio de datos sobre los procesos en ejecución.		
Endpoint	/api/machine/<string:mac>/processes	Roles	GET: Admin, Técnico POST: Máq.

Body	{“processes”: […]}
GET	
200	{“processes”: […]}
404	{"message": "Machine not found"}
401	{“message: “Unauthorized Access”}
POST	
403	{"message": "Action forbidden"}
200	{"message": "Data received"}
404	{"message": "Machine not found"}
401	{“message: “Unauthorized Access”}

ShellOutputReceiver			
Descripción	Intercambio de datos sobre las consolas remotas.		
Endpoint	/api/machine/<string:mac>/shell_output /api/machine/<string:mac>/shell_output/<int:idd>	Roles	GET: Admin, Técnico POST: Máq.
Body	{"output": "str", "directory": “str”, "window_id": <int>}		
GET			
200	{"output": "str", "directory": “str”, "window_id": <int>}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		
POST			
403	{"message": "Action forbidden"}		
200	{"message": "Data received"}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		

DemoRequestManagement			
Descripción	Gestión de las peticiones de cuentas de demostración.		
Endpoint	/api/demo_request	Roles	Todos
Body	{“lname”: “str”, “lname”: “str”, “email”: “str”}		
POST			
200	{"message": "Email sent correctly"}		
500	{'message': 'Something went wrong'}		

DownloadCodesList			
Descripción	Devuelve listado de todos los códigos de descarga.		
Endpoint	/api/downloads	Roles	Admin, Técnico
Body	N/A		
GET			
200	{“download_codes”: […]}		
401	{“message: “Unauthorized Access”}		

DownloadCodeUpdate			
Descripción	Activación / desactivación de un código de descarga.		
Endpoint	/api/download	Roles	Admin, Técnico
Body	{“id”: “str”, “download_enable”: <bool>, “connect_enable”: <bool>}		
GET			
200	{"message": "Download code updated correctly"}		
404	{"message": "Download code not found"}		
401	{“message: “Unauthorized Access”}		

DownloadCodeManagement			
Descripción	Creación de códigos de descarga y descarga del cliente.		
Endpoint	/download/<string:idd> /download	Roles	POST: Admin, Técnico GET: Todos
Body	{“platform”: “str”, “email”: “str”, “language”: “str”}		
POST			
200	{"message": "Download code updated correctly"}		
500	{"message": "Download code not found"}		
401	{“message: “Unauthorized Access”}		
GET			
200	Redirección a página estática de código caducado.		
201	Fichero ejecutable del cliente.		

LoginLogRetriever			
Descripción	Devuelve listado de todos los registros de acceso.		
Endpoint	/api/login/logs	Roles	Admin, Técnico
Body	N/A		
GET			
200	{“login_logs”: […]}		
401	{“message: “Unauthorized Access”}		

MachineLogRetriever			
Descripción	Devuelve listado de todos los registros de operaciones.		
Endpoint	/api/machine/log/<string:mac>	Roles	Admin, Técnico
Body	N/A		
GET			
200	{“machine_logs”: […]}		
401	{“message: “Unauthorized Access”}		

MachineStatusAcknowledge			
Descripción	Gestión del estado de conexión de una máquina.		
Endpoint	/api/machine/<string:mac>/status	Roles	GET/POST:Admin, Técnico PUT: Máquina
Body	{“status”: <int>}		
GET			
404	{"message": "Machine not found on system"}		
200	{"status ": <int>}		
401	{“message: “Unauthorized Access”}		
PUT			
403	{"message": "Action forbidden"}		
200	{"message": "New status received"}		
404	{"message": "Machine not found on system"}		
401	{“message: “Unauthorized Access”}		
POST			
404	{"message": "Machine not found on system"}		
200	{"message": "Machine status updated"}		
401	{“message: “Unauthorized Access”}		

MachineListResource			
Descripción	Devuelve listado de todas las máquinas conectadas.		
Endpoint	/api/machines	Roles	Admin, Técnico
Body	N/A		
GET			
200	{“machines”: […]}		
401	{“message: “Unauthorized Access”}		

MachineOpcodeMonitor			
Descripción	Gestión de los OPCODEs de una máquina		
Endpoint	/api/machine/<string:mac>/<int:n> /api/machine/<string:mac>	Roles	GET/POST:Admin, Técnico, Máquina PUT: Máquina
Body	{“opcode”: “str”, “account_id”: <int>}		
GET			
404	{"message": "Machine not found"}		
200	{"opcode": “…”}		
401	{“message: “Unauthorized Access”}		
POST			
409	{"message": "Conflict with non-existing account"}		
200	{"message": "Opcode updated"}		
404	{"message": "Machine not found on system"}		
401	{“message: “Unauthorized Access”}		
PUT			
403	{"message": "Action forbidden"}		
200	{"message": "Opcode updated"}		
404	{"message": "Machine not found on system"}		
401	{“message: “Unauthorized Access”}		

MachineAuthCodeValidator			
Descripción	Ejecución del triple factor de autenticación.		
Endpoint	/api/auth/<string:mac> /api/auth	Roles	GET: Admin, Técnico PUT/POST: Todos
Body	{“auth_code”: <int>, “download_code”: “str”, “email”: “str”}		
GET			
200	{“code”: <int>}		
404	{"message": "Machine not found on system"}		
401	{“message: “Unauthorized Access”}		
POST			
403	{"message": "Not allowed"}		
200	{"uname": “str”, "pass": “str”, "account_id": <int>}		
406	{"message": "Connection not authorized"}		
404	{"message": "Machine not found on system"}		
PUT			
404	{"message": "Download not registed for given parameters"}		

500	{'message': 'Something went wrong'}
403	{"message": "Values do not match with recorded"}
404	{"message": "Machine not found on system"}
200	{"message": "Email sent correctly"}

MachineResource			
Descripción	Gestión datos de caso de máquina y eliminación de máquinas.		
Endpoint	/api/machine/<int:idd> /api/machine	Roles	POST: Todos GET/PUT/DELETE: Admin, Técnico
Body	{“mac_address”: “str”, “platform”: “str”, ”platform_release”: “str”, ”platform_version”: “str”, ”architecture”: “str”, ”hostname”: “str”, ”ip_address”: “str”, ”processor”: “str”, ”ram”: “str”, ”network_cards”: “str”, ”incident”: “str”, ”incident_date”: “str”, ”incident_info”: “str”, ”contact”: “str”, ”contact_tel”: “str”, ”contact_email”: “str”, ”contact_info”: “str”, ”account_id”: <int>, ”additional_info”: “str”}		
POST			
201	{"machine": {…} }		
200	{"message": "Machine already on system"}		
PUT			
409	{"message": "Conflict with non-existing account"}		
200	{"message": "Data updated correctly"}		
404	{"message": "Account not found"}		
401	{“message: “Unauthorized Access”}		
DELETE			
200	{"machine ": {…}}		
404	{"message": "Machine not found in system"}		
401	{“message: “Unauthorized Access”}		
GET			
200	{"message": "Machine deleted"}		
404	{"message": "Machine not found in system"}		
401	{“message: “Unauthorized Access”}		

MessageRetriever			
Descripción	Transferencia de mensajes interfaz – servidor - cliente.		
Endpoint	/api/machine/<string:mac>/message	Roles	GET: Máquina PUT/POST: Admin, Técnico
Body	{“message”: “str”}		
GET			
200	{“message”: “…”}		
404	{"message": "Machine not found in system"}		
403	{"message": "Action forbidden"}		
401	{“message: “Unauthorized Access”}		
POST			
200	{"message": "Update correctly"}		
404	{"message": "Machine not found"}		
401	{“message: “Unauthorized Access”}		
PUT			
404	{"message": "Machine not found in system"}		
200	{"message": "Message reset successful"}		
403	{"message": "Action forbidden"}		
401	{“message: “Unauthorized Access”}		

CommandManagement			
Descripción	Gestión datos sesiones remotas y transferencia de comandos y sus salidas entre cliente, servidor e interfaz.		
Endpoint	/api/machine/<string:mac>/shell/<int:idd>	Roles	POST: Admin, Técnico GET/PUT/DELETE: Máquina
Body	{“command: “str”, “account_id”: <int>, “code_ack”: “str”}		
POST			
404	{"message": "Machine not found"}		
200	{"message": "Command updated"}		
409	{"message": "Conflict with non-existing account"}		
401	{“message: “Unauthorized Access”}		
PUT			
409	{"message": "ACK is not correct"}		
200	{"message": "ACK is correct"}		

404	{"message": "Machine not found"}
401	{“message: “Unauthorized Access”}
DELETE	
200	{"message": "Command reset ok"}
404	{"message": "Machine not found"}
401	{“message: “Unauthorized Access”}
GET	
404	{"message": "Machine not found"}
200	{"command ": “…”}
401	{“message: “Unauthorized Access”}

ServerStatusCheck			
Descripción	Test de conexión con el servidor.		
Endpoint	/api/time	Roles	Todos
Body	N/A		
GET			
200	{“date-time”: “…”}		