Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Desarrollo de una aplicación web para la gestión fuera de banda de un laboratorio de redes de datos

Estudante: Alfonso Torralba Mantiñán
Dirección: Francisco Javier Nóvoa de Manuel
José Carlos Dafonte Vázquez

A Coruña, febreiro de 2021.

A mis padres, por estar siempre ahí

Agradecimientos

Agradecer en primer lugar a mis padres y abuelos, por toda su paciencia y tolerancia hacia mí desde que tengo uso de razón, así como por su cariño y apoyo altruista. Otorgar un gran reconocimiento a Manolo, Nati, Toño, Ramón, Erick, Mercedes y Diego, por ser como una segunda familia para mí. También agradecer a mis tutores del proyecto, Francisco y Carlos, por toda su ayuda, consejos y enseñanzas transmitidas. A Noe, por ser mi mejor amiga y estar siempre tanto en las buenas como en las malas. Por último, pero no por ello menos importante, a todos mis amigos del gimnasio, facultad y demás conocidos, con especial mención a Igor, David y Carlos Miguel. Muchas gracias a todos, os lo agradezco de corazón.

Resumen

La configuración de dispositivos de red requiere una conexión física directa del administrador en determinadas circunstancias como, por ejemplo, cuando los dispositivos pierden su configuración o su sistema operativo y, por lo tanto, no es posible llevar a cabo conexiones remotas, que precisan conectividad TCP/IP. En estas circunstancias es necesario utilizar un puerto especial de administración, denominado habitualmente ”consola”, que permite el establecimiento de una sesión de terminal directa en capa física. Para ello, es necesario que el ordenador que se utiliza para realizar la configuración establezca una conexión física serie, mediante un puerto de comunicaciones RS-232 o un puerto USB, con el fin de enviar directamente caracteres al citado puerto de consola.

En el Máster de Ciberseguridad que se imparte conjuntamente entre la Universidade da Coruña y la Universidade de Vigo se dispone de un laboratorio de redes de datos. Dicho laboratorio está formado por diferentes dispositivos de red (switches, switches multicapa, routers y firewalls) que son utilizados por los estudiantes para la realización de prácticas. La actual situación pandémica dificulta que los alumnos puedan acceder físicamente a estos dispositivos. Los continuos confinamientos imposibilitan que los estudiantes accedan físicamente a los equipos. Si bien es posible que accedan a la gestión de los equipos de forma remota, por ejemplo mediante SSH, un fallo en la configuración provoca que los dispositivos pierdan la conectividad IP y, por lo tanto, los alumnos el acceso remoto a los dispositivos. De esta problemática surge la necesidad de buscar una alternativa de trabajo, un nuevo método que permita la gestión remota de los dispositivos de red, sin precisar que los dispositivos administrados disponga de conectividad IP.

En el presente proyecto se propone una solución a este problema, incorporando un equipo de gestión intermedio que permite el acceso remoto a los dispositivos de red utilizando el puerto de consola. Se ha diseñado una interfaz web que facilita el acceso. La solución permite controlar la conexión simultánea a las consolas de múltiples usuarios de forma remota. Además, proporciona la posibilidad de crear diferentes entornos de trabajo compuestos por distintas combinaciones de equipos de red, lo que da flexibilidad y modularidad a este producto.

Actualmente, la aplicación se encuentra completamente operativa en un laboratorio de pruebas. La herramienta se encuentra disponible en el siguiente repositorio Git: https://github.com/torralba98/application-out-of-band-management

Abstract

Network devices configuration requires a direct physical connection from the administrator in certain circumstances, such as when devices lose their configuration or operating system and therefore remote connections are not possible, which require TCP/IP connectivity. In these circumstances it is necessary to use a special management port, usually called ”console”, which allows the establishment of a direct terminal session at the physical layer. For this, it is necessary for the computer used to perform the configuration to establish a physical serial connection, through an RS-232 communications port or a USB port, in order to directly send characters to the aforementioned console port.

In the Master of Cybersecurity that is taught jointly between the University of Coruña and the University of Vigo, there is a data network laboratory. This laboratory is made up of different network devices (switches, multilayer switches, routers, and firewalls) that are used by students for practical work. The current pandemic situation makes it difficult for students to physically access these devices. Continual lockdowns make it impossible for students to physically access equipment. Although it is possible for them to access the management of the computers remotely, for example via SSH, a failure in the configuration causes the devices to lose IP connectivity and, therefore, the students remote access to the devices. From this problem arises the need to find a work alternative, a new method that allows remote management of network devices, without requiring that the managed devices have IP connectivity.

In this project a solution to this problem is proposed, incorporating an intermediate management equipment that allows remote access to network devices using the console port. A web interface has been designed for easy access. The solution allows to control the simultaneous connection to the consoles of multiple users remotely. In addition, it provides the possibility of creating different work environments composed of different combinations of network equipment, which gives flexibility and modularity to this product.

Currently the application is fully functional and operational in a testing laboratory. The developed tool is available in the following Git repository: https://github.com/torralba98/application-out-of-band-management

Palabras clave:

- Placa de bajo coste
- Dispositivos de red
- Gestión fuera de banda

Keywords:

- Raspberry Pi
- Network devices
- Out-of-band management

Hardware y software utilizado

- Low cost plate
- Multi Hub USB
- Dispositivos de red
- Cables de consola
- Apache Server
- MariaDB
- Node.js
- PHP
- HTML
- JavaScript

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introdución

El pasado año 2020, como bien es sabido, llegó cargado de muy amargas sorpresas y malos momentos para toda la humanidad. Todos somos conscientes de la gran crisis mundial ocasionada por el coronavirus SARS-CoV-2 (**COVID-19**) y las repercusiones que este conllevó. Todos los sectores de la sociedad fueron afectados, incluso, el de la informática y las telecomunicaciones. Empresas, universidades, colegios y otras numerosas instituciones tuvieron que recurrir a la vía telemática ante para continuar su actividad.

El correcto funcionamiento de los sistemas de información y las telecomunicaciones pasaron a convertirse en un elemento fundamental en una gran cantidad de organizaciones. La disponibilidad de los servicios paso de ser una características importante a crítica. Las herramientas de gestión en línea, fundamentadas en la conectividad IP de los dispositivos (VPNs, accesos remotos basados en SSH, escritorios remotos, etc.) pueden sufrir incidencias que provoquen los equipos administrados se queden aislados. En circunstancias normales, un operador se desplazaría físicamente hasta dicho dispositivo para restaurar la conectividad, pero en las circunstancias actuales es necesario disponer de herramientas de gestión alternativas.

Concretamente, en el ámbito de la administración de dispositivos de red, la configuración a través de una interfaz de línea de comandos es el método de gestión más utilizado para la configuración inicial de dichos dispositivos y para las tareas de operación y mantenimiento. Para acceder a dicha línea de comandos se pueden utilizar dos métodos de conexión, que se denominan respectivamente "fuera de banda" y "en banda". El primero de ellos se basa en una conexión física serie a través de un conector específico denominado habitualmente "puerto de consola". En las especificaciones técnicas del dispositivo de red se indican los parámetros que se deben usar para enviar caracteres al dispositivo administrado a través de este puerto. Es necesario indicar la velocidad en bits por segundo (por ejemplo 9600 bps), el número de bits que conforman el caracter (por ejemplo 8 bits), los bits de separación entre caracteres (por ejemplo, 1 bit de parada), la paridad y el control de flujo. El terminal de gestión hace uso de un puerto de comunicaciones, de forma clásica un RS-232 y actualmente puertos USB,

para a través de un cable específico conectarse al puerto de consola del dispositivo de red. Se establece una conexión orientada al carácter que permite el intercambio de texto directo entre el terminal, que puede ser una aplicación que se ejecuta en un PC, como por ejemplo Teraterm, Putty, secureCRT, MobaXterm, etc. El envío de datos a través de esta conexión se realiza transmitiendo caracteres directamente a través de la conexión física. La ventaja que tiene este modo de gestión es la independencia de la configuración del dispositivo de red, es decir, no es necesario que dicho dispositivo tenga conectividad a través de una red IP. El inconveniente de este método es que es necesario estar físicamente conectado a dicho dispositivo, lo que en muchas ocasiones no es posible o es muy costoso. A este tipo de conexión de gestión se la denomina fuera de banda porque no precisa de una red IP subyacente para llevarse a cabo.

Debido a las limitaciones que presenta el método de gestión "fuera de banda", existe el método "en banda", que permite el acceso remoto a la línea de comandos. Tal es el caso del uso de SSH, SNMPv3 o incluso más recientemente el uso de APIs Netconf o Restconf. La ventaja principal que aportan estas tecnologías es el acceso remoto y el mayor inconveniente es que todas necesitan conectividad completa a través de TCP/IP. Debido a esto, se dice que estas tecnologías son de tipo "en banda".

En un escenario habitual de operación y mantenimiento de dispositivos de red es habitual llevar a cabo tareas en las que, si se produce un error, se pierde la conectividad. Esta situación es completamente indeseable (supongamos un operador ubicado en A Coruña haciendo cambios en un router ubicado en Miami). En estos casos es necesaria la intervención de un operador local, con conexión de consola, si no se toman las precauciones necesarias.

En el caso de un laboratorio de prácticas de redes de comunicaciones, esta situación es especialmente habitual, debido a que los administradores son estudiantes en período de aprendizaje que cometen errores que dejan aislados a los equipos que pretenden configurar. En una coyuntura como la producida por la COVID-19 es clave que los alumnos puedan seguir accediendo a sus dispositivos y realizar prácticas en sus equipos.

En este contexto surge este proyecto, que busca darle una solución al problema plantaeado. Se propone un sistema de bajo coste que facilita el trabajo a distancia, pero independiente del estado de configuración y conectividad de los dispositivos administrados.

1.1 Objetivos del proyecto

El objetivo principal de este proyecto será la elaboración de un sistema que proporcione una interfaz web que facilite el acceso remoto a las consolas de los dispositivos de red. El sistema estará basado en una placa de bajo coste con un hub USB, que permitirá la conexión

simultánea de múltiples desde la placa de bajo coste o equipo de salto a diferentes dispositivos de red.

Para lograr este objetivo principal será preciso tener en cuenta y completar una serie de objetivos concretos.

- **Proporcionar acceso a las conexiones de consola desde la interfaz web.** La aplicación permitirá a los usuarios el acceso remoto a las consolas de los dispositivos.
- **Controlar el acceso en base a usuarios.** Se tratará de una aplicación multiusuario. En ella, cada alumno podrá registrarse y disponer de sus propias credenciales de inicio de sesión. Dentro de la aplicación, cada una de las consolas de los dispositivos solo podrá ser accedida por un usuario a la vez. Será necesaria la gestión del acceso de los usuarios a dispositivos pertenecientes al mismo grupo.
- **Facilitar la creación de grupos de usuarios y dispositivos.** La aplicación dispondrá de herramientas que permitan la gestión tanto de grupos de usuarios como de dispositivos de red.
- **Objetivos no funcionales.**
  - **Gestión de los recursos de la placa de bajo coste.** Será necesario prestar especial atención a como son empleados todos los recursos de la placa de bajo coste, evitando saturar esta.
  - **Almacenamiento de la información.** Debido a la persistencia de la información, será obligatorio evaluar las más adecuadas alternativas disponibles para la conservación de esta.

1.2 Organización de la memoria del proyecto

- **Introducción.** Se explica la motivación que llevó al desarrollo del trabajo y los objetivos del mismo. Se incluye adicionalmente un breve resumen de la estructura de la memoria del proyecto.
- **Fundamentos teóricos y estudio de alternativas.** Se explican aspectos fundamentales relacionados con la administración de redes y se analizan aspectos técnicos detallados relacionados con el trabajo. Se pretenden exponer los conceptos básicos que deben conocerse previamente antes de abordar la realización del proyecto. También se incluye información acerca de diferentes alternativas a nuestro trabajo existentes en el mercado actual.

- **Fundamentos tecnológicos.** Se detallan las diferentes tecnologías empleadas en el desarrollo del proyecto. Se analiza el hardware, software y entorno de desarrollo empleado.
- **Metodología.** Se expone la metodología empleada en el proyecto, así como todos sus conceptos asociados.
- **Planificación y estudio económico.** Se da a conocer el calendario de realización del proyecto y una evaluación del coste económico derivado del desarrollo del mismo.
- **Trabajo desarrollado.** Se describe la aplicación desarrollada. Se analizan diversos aspectos relacionados con su diseño, implementación, resultados finales y pruebas.
- **Conclusiones y líneas futuras.** Se realiza un estudio de posibles líneas futuras de mejora de nuestra aplicación. Se acompaña también de las conclusiones finales de la realización del proyecto, así como del contraste de objetivos y de las lecciones aprendidas durante su desarrollo.
- **Apéndices.** Se acompaña a la memoria de manuales de usuario y de instalación de la aplicación. Se incluye un apéndice dedicado a la explicación de los casos de uso de la aplicación web.

Capítulo 2

Fundamentos teóricos y estudio de alternativas

Nuestro proyecto está enmarcado en el ámbito de la administración "fuera de banda" de equipos de red. Para contextualizar adecuadmente el entorno, es necesario explicar previamente como se estructuran en planos las diferentes funcionalidades de los dispositivos de red. Éstos son: el plano de datos, el plano de control y el plano de gestión. A continuación se detallan las funcionalidades asociadas con cada uno de ellos.

2.1 Planos de los dispositivos de red

- **Plano de Datos**. Aquel en el que se llevan a cabo las operaciones necesarias para el enrutamiento y la conmutación del tráfico de red. En un switch el plano de datos se encarga de conmutar tramas en base a la dirección MAC de destino. En un router las funciones de este plano consisten en determinar la interfaz de salida de los paquetes y encapsular el paquete IP en la trama de capa de enlace de datos adecuada. También son características de este plano otras funciones como gestión de colas y la priorización del tráfico. En los firewalls, además es necesario aplicar los filtros establecidos en las listas de control de acceso, que sirven para implementar las politicas de seguridad perimetral.

- **Plano de Control**. En el plano de control se ejecutan todas las funciones relacionadas con el mantenimiento de las estructuras de datos utilizadas en el plano de datos, para la toma de decisiones de envío. En este plano se encuentra funciones como creación de VLANs y la operación del protocolo Spanning Tree en los switches y el funcionamiento de los protocolos de enrutamiento, que permiten mantener la información de las tablas de rutas actualizada, en el caso de los routers.

- **Pano de Gestión**. Como bien indica su nombre, es en el que ejecutan todas las funcio-

nes que permiten administrar el dispositivocuestión. Protocolos tales como Telnet, SSH, HTTP, HTTPS o SNMP trabajan en este plano. Mencionar que a la hora de administrar un dispositivo de red, este se comporta como si de un dispositivo final se tratase.

Es importante tener bien clara la diferencia entre los tres planos anteriores. Son importantes a la hora de evaluar el rendimiento de la red, al definir políticas de seguridad o a la hora de evaluar el equipamiento a adquirir para nuestra red pues, dependiendo del diseño de esta, requerirá unas funciones específicas en cada plano. En la figura 2.1 (página 6) podemos ver gráficamente todo lo explicando anteriormente, a fin de reforzar los conceptos recientemente explicados.

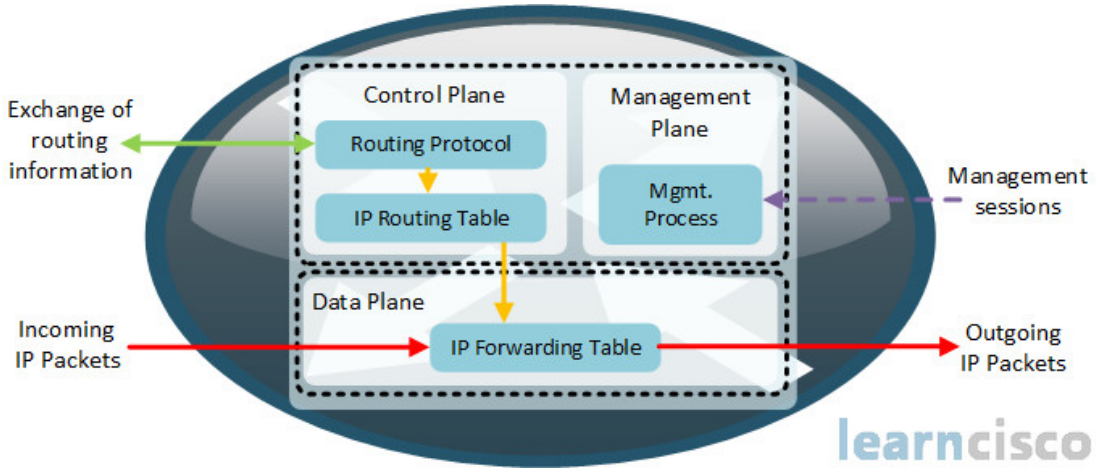


Figura 2.1: Planos de un dispositivo de red.
Fuente: www.learncisco.net/courses/iins/security-on-cisco-routers/cisco-network-foundation-and-protection-framework.html

2.2 Gestión de los dispositivos de red

Como bien se mencionó en el "*plano de Gestión*" (sección 2.1, página 5), algunos ejemplos de protocolos usados para la gestión remota de los dispositivos de red son SSH, Telnet y SNMP. Estos protocolos se ejecutan en la capa de aplicación de la pila de protocolos TCP/IP y del modelo OSI. Por lo tanto, es imprescindible la conectividad mediante TCP/IP en este tipo de conexiones. A esto se le conoce como "*gestión en banda*". En caso de que se pierda esta conectividad TCP/IP, perderemos la capacidad de poder gestionar los dispositivos. Para solucionar este problema surge la conocida como "*gestión fuera de banda*". Al igual que con los diferentes planos de los dispositivos de red, es preciso tener bien la clara la diferencia entre estos dos tipos de gestión, que según Intel [1].

- **Gestión en banda**. Aquella gestión que precisa que los dispositivos estén encendidos y que sean capaces de conectarse a los servicios de TI antes de poder gestionarse.

- **Gestión fuera de banda**. Con este tipo de gestión, los administradores de los dispositivos de red pueden acceder a los dispositivos incluso aunque estos no dispongan de una configuración de red. Este tipo de gestión tiene una serie de ventajas sobre la anterior como otorgar un mejor acceso y funcionalidad, una reducción notoria de los costes de gestión de TI y una mejora considerable en los tiempos de mantenimiento de los dispositivos, con lo que aumenta la productividad de estos.

Para nuestro proyecto utilizaremos la "*gestión fuera de banda*". En un escenario típico, para llevar a cabo este tipo de gestión es necesaria la conexión física directa a los puertos de consola de los dispositivos de red. Esta conexión es la que permite el intercambio de caracteres entre el dispositivo de red y nuestra computadora. Obviamente, no se precisa para nada de una conectividad de red, sin embargo, presenta una gran limitación, que es la obligatoriedad de ubicarnos físicamente junto al dispositivo. La solución más eficiente y rápida para este problema es el empleo de un tercer dispositivo, como elemento de salto intermedio. En este escenario, el nuevo dispositivo esta conectado físicamente a los puertos de consola de cada uno de los dispositivos de red. Sin embargo, aunque estos últimos carezcan de conexión a Internet, es preciso que el dispositivo alternativo si disponga de ella. Esto permite la conexión remota a este equipo de salto, utilizando algún protocolo de administración remota (por ejemplo, SSH o HTTPS), para así gestionar los dispositivos de red desde cualquier lugar.

En la figura 2.2 (página 8) podemos observar cómo sería un escenario que aplica el método de "*gestión en banda*". En él podemos ver una serie de dispositivos (routers y switches para este ejemplo) interconectados. Todos ellos cuentan con su propia configuración de red y son accesibles desde Internet. Por lo tanto, desde otro lugar, un usuario puede conectarse, a través de Internet, a cualquiera de los dispositivos conformantes de la red. En esto se basa la "*gestión en banda*", en otorgar una manera simple y rápida de gestionar equipos remotamente. Este escenario tiene un gran problema. Si nos fijamos en las capas en las que se lleva a cabo la comunicación de gestión, podemos percibir la existencia de una gran dependencia del correcto funcionamiento de las capas superiores (capas de aplicación, presentación, sesión, ...) de la arquitectura TCP/IP. En este caso, si por ejemplo el dispositivo no contase con una correcta configuración de red, no podríamos conectar por SSH(Capa de Aplicación).

Por otra parte, en la figura 2.3 (página 9) podemos ver el escenario referido al método de "*gestión fuera de banda*". En este nuevo ejemplo observamos los mismos dispositivos que en el ejemplo anterior, también interconectados. La principal diferencia es que en este caso no estarán conectados a Internet ni será necesario que presenten una configuración de red válida.

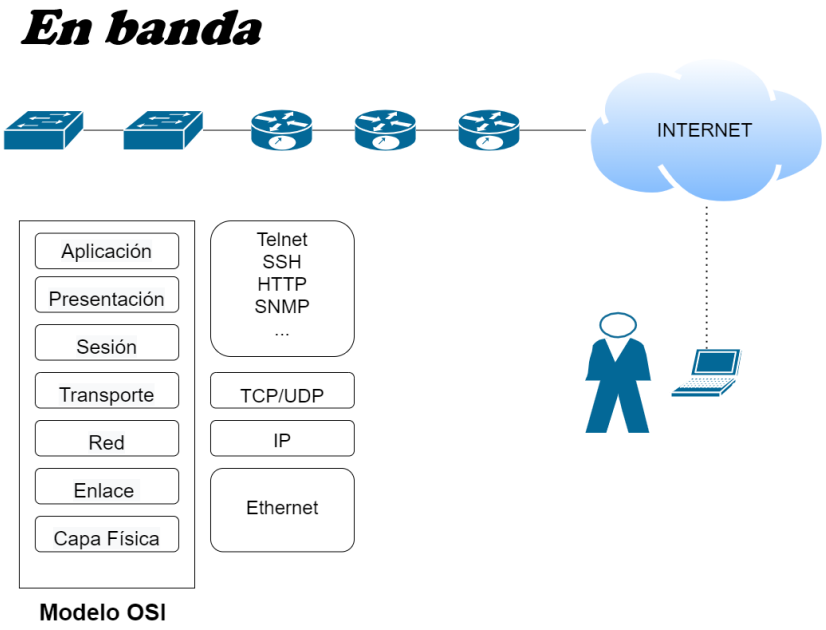


Figura 2.2: Diagrama de representación de la "gestión en banda".

En contrapartida, el usuario deberá conectarse físicamente (por ejemplo, con conectores RJ45) con su equipo a los dispositivos miembros de la red. Desde el punto de vista del modelo OSI ahora solo trabajamos con la Capa Física, en concreto con los puertos de consola de cada dispositivo. Para esta nueva situación es preciso definir previamente al establecimiento de la comunicación, para cada dispositivo, los parámetros que definirán como se transmitirán los caracteres (velocidad de transmisión, bits de datos, ...) físicamente.

Para entender mejor este último escenario explicado, es preciso que entendamos perfectamente el significado de la "*comunicación orientada al carácter*". En este tipo de protocolos, se intenta garantizar, mediante todos los controles, la calidad de los caracteres en la comunicación. Los protocolos orientados al carácter son aquellos en los que los mensajes están compuestos por conjuntos de caracteres pertenecientes a un determinado código. Para cada uno de estos caracteres, contará con su propio significado específico y único. Un ejemplo de estos protocolos es el caso de UART, que es el empleado en nuestro proyecto. Para explicarlo, se toma como referencia lo enunciado en la web oficial de Analog Devices Inc. [2]: "UART, o receptor-transmisor asincron universal, es uno de los protocolos de comunicación de dispositivo a dispositivo más utilizados.", el cual, correctamente configurado, es capaz de trabajar con diferentes tipos de protocolos serie que implican la transmisión y recepción de datos serie. La estructura de la paquetería transmitida con este protocolo la podemos apreciar en la figura 2.4 (página 9). Cada paquete consta de:

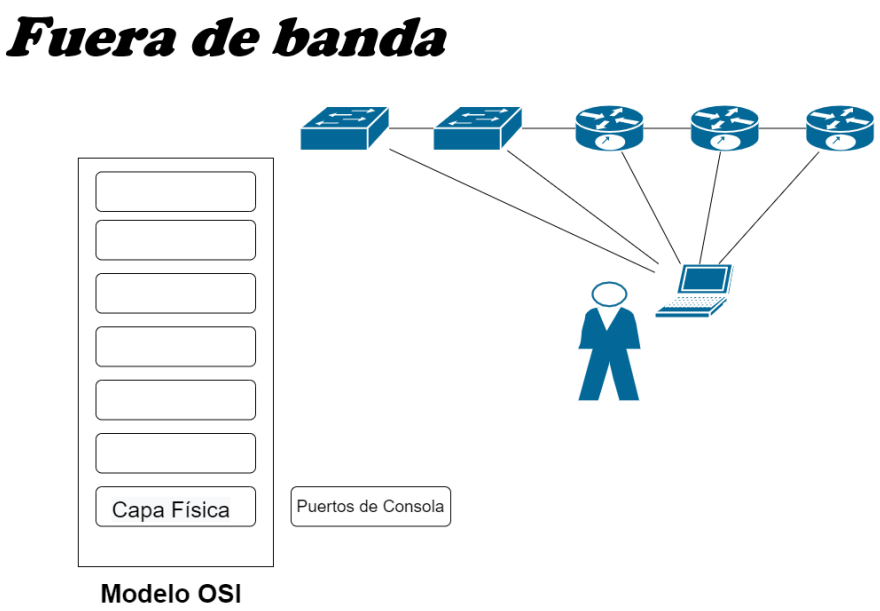


Figura 2.3: Diagrama de representación de la "gestión fuera de banda".

- **Bit de inicio**. Este bit es el empleado para marcar el inicio de la transmisión de la paquetería.
- **Data frame**. Contiene los datos reales que se desean transmitir. Puede ser desde 5 bits hasta 8 bits en caso de emplear un bit de paridad. De no emplear dicho bit de paridad, la trama de datos puede llegar a componerse de 9 bits.
- **Bits de paridad**. Bit que permite conocer si algún dato ha cambiado durante la transmisión, es decir, si no se han producido errores.
- **Bits de parada**. Marcan el final de cada paquete de datos.

Figura 2.4: Representación de la estructura de la paquetería de UART.
Fuente: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html#

Por último, mencionar que, en nuestro proyecto, la transmisión de los datos se realizará a través de un único cable y esta se realizará carácter a carácter.

2.3 Estudio de alternativas

Existen numerosas alternativas a nuestro trabajo en el mercado actual. Ejemplos claros de estas podrían ser las ofertadas por las empresas Opengear [3], Lantronix [4] o Raritan [5]. Los productos ofertados por todas cumplen una función similar a la de nuestra propuesta, permitir la gestión fuera de banda de dispositivos de red. El principal obstáculo que nos topamos es su elevado coste. La mayoría suponen elevados pagos periódicos para poder disfrutar de sus ventajas. Con este trabajo de fin de grado demostramos que es posible desarrollar una alternativa de bajo coste que cumpla con prácticamente las mismas funcionalidades.

Opengear, ofrece entre sus productos el conocico como *Infraestructura Manager* que podemos ver en la figura 2.5 (página 10), el cual permite al personal de operaciones de red el acceso de forma segura a los diferentes dispositivos de red críticos desde el centro de operaciones de red. Proporciona una administración remota, a través de un enlace seguro, de los dispositivos de red, que estarán conectados físicamente.

Figura 2.5: IM7200 Infrastructure Manager
Fuente:
https://opengear.com/products/im7200-infrastructure-manager/

Para el caso de Lantronix proporciona soluciones de gestión fuera de banda para las infraestructuras de TI, ofreciendo una gestión remota optimizada de la infraestructura de red, servidores y energía, con lo que garantiza la continuidad del negocio y proporciona un acceso seguro y confiable. En la figura 2.6 (página 11) podemos ver uno de sus productos ofertados para temas de gestión, tanto fuera de banda como en banda, de los diferentes dispositivos de red de una empresa.

Figura 2.6: Lantronix SLC 8000.
Fuente: https://www.lantronix.com/products/lantronix-slc-8000/

Finalmente, la empresa Raritan oferta una amplia variedad de conmutadores KVM a través de IP que aprovechan las redes Ethernet y TCP/IP actuales para proporcionar acceso remoto, control y gestión en cualquier momento y lugar, de los dispositivos de red. En la figura 2.7 (página 11 vemos un ejemplo de uno de sus productos ofertados. Para este caso, se trata del *DOMINION® KX III*, el cual proporciona acceso a 1,2,4 u 8 a controlar, de forma remota a nivel de BIOS, de 8 a 64 servidores.

Figura 2.7: DOMINION® KX III
Fuente: https://www.raritan.com/products/kvm-serial

Capítulo 3

Fundamentos tecnológicos

3.1 Hardware

Es de vital importancia para el desarrollo de este trabajo la selección de una placa de bajo coste con las características adecuadas. Hemos empleado una Raspberry Pi 3 Model B Plus Rev. 1.3, que cuenta con 1 GB de memoria RAM. Esta Raspberry está conectada físicamente un hub USB 3.0 de tipo Industrial de 16 conectores. Desde dicho hub se conectan los cables a los puertos de consola (sección 3.1.1) de todos los dispositivos de red disponibles.

Es el dispositivo de salto, que está habilitado para permitir el acceso remoto y, de esta forma, gestionar dichos dispositivos de red. Se optó por este modelo de Raspberry en concreto, debido a que cuenta con un procesador lo suficientemente capaz de soportar toda la carga de trabajo de nuestra aplicación web y que a la vez presenta un coste económico asequible.

Bien es cierto que, para la realización del proyecto no es imprescindible el uso de una Raspberry Pi. Alternativas a esta podría ser el uso de un ordenador de mesa o de un ordenador portátil. De cualquier forma, no resulta factible ninguna de estas alternativas para la implementación de pruebas o en producción pues, debería estar disponible de forma ininterrumpida proporcionado acceso la aplicación. Resulta más práctico y económico el uso de una Raspberry.

3.1.1 Puertos COM

Los también conocidos como puertos serie o puertos de consola, son unas interfaces de E/S, presentes en los dispositivos de red, que permiten establecer una conexión física entre estos y los dispositivos que los gestionan. Si bien hasta finales del siglo pasado era común con los PCs incorporasen este tipo de puertos (puertos COM) con una interfaz RS-232, la mayor parte de los ordenadores modernos ya no disponen de estos puertos, que pueden ser emulados a través de adaptadores USB-RS232.

En los puertos serie, la información se transmite bit a bit, lo que otorga un método de comunicación simple y confiable. Para cada dispositivo de red a administrar se debe definir la velocidad de transmisión, el número de bits que componen cada caracter, los bits de separación entre caracteres, la paridad y debe indicarse si se implementa la funcionalidad de control de flujo. Tanto el puerto del ordenador de gestión como el puerto de consola del dispositivo administrado deben estar configurados con las mismas características. En el caso de este proyecto: 9.600 bps, 8 bits de datos por caracter, 1 bit de separación, sin paridad y sin control de flujo. En la figura 3.1 (página 14) podemos ver el que sería el puerto de consola de un dispositivo de red.

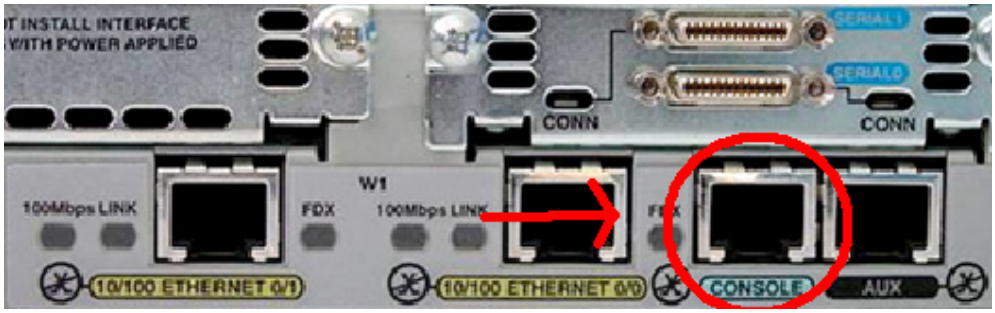


Figura 3.1: Puerto de consola de un dispositivo de red.

3.2 Software

3.2.1 Sistema Operativo

El sistema operativo empleado será el propio de la Raspberry que, para nuestro caso será *Raspbian GNU/Linux 10 (buster)*. Este es una distribución del sistema operativo GNU/Linux, que está basada en Debian. Para desplegar la interfaz web que facilita el acceso remoto a las consolas será necesario contar con las siguientes tecnologías y componentes software.

3.2.2 Node.js

Según la web oficial de Node.js [6], este "*es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome*". Es de código abierto y además es posible ejecutarlo en Mac OS X, Windows y Linux. Cuenta con una gran velocidad de ejecución, así como una increíble capacidad para gestionar grandes cantidades de clientes asíncronamente.

El problema más común de las aplicaciones a la hora de trabajar asíncronamente es que estás se basaban en la programación basada en hilos. Este tipo de programación conlleva el

uso de un excesivo espacio de memoria, el cual se incrementa exponencialmente a medida que iba aumentando el número de clientes conectados a la aplicación. Node le puso solución a este problema pues, en lugar de ir generando nuevos hilos para cada nuevo cliente, cada nueva conexión disparaba la ejecución de un nuevo evento dentro del propio proceso del motor Node, permitiendo así que un único servidor pueda soportar múltiples conexiones. Se eligió como tecnología para desarrollar el servidor encargado del intercambio de caracteres entre el equipo de salto, implementado en una Raspberry Pi, y los diferentes dispositivos de red.

3.2.3 Apache HTTP Server

Apache es uno de los servidores web de código abierto más empleados en el mundo a día de hoy. Como bien especifica la documentación oficial de su página web [7], la principal función de un servidor Apache es la de proporcionar las páginas web alojadas en el propio servidor a los diferentes exploradores web (Google Chrome, Mozilla Firefox, Internet Explorer...). Apache es capaz de garantizar una comunicación fluida y constante entre cliente y servidor. A mayores, proporciona servicios HTTP a la par que respeta los estándares HTTP actuales.

Bien cierto es que el rendimiento de nuestra web podría verse notablemente afectado en caso de que la concurrencia de usuarios fuese alta. En cualquier caso este proyecto está planteado para un reducido número de usuarios.

El servidor Apache está directamente relacionado con el servidor Node. Cuando un usuario emite una petición HTTP a la página referente a la consola de un dispositivo en red, la aplicación ejecutará automáticamente el servidor Node correspondiente a este último.

3.2.4 Sistema de gestión de base de datos

De vital importancia es el almacenamiento y gestión de la información por parte nuestra aplicación. Como sistema de gestión de base de datos se decidió emplear MariaDB. Acorde con su web oficial [8], "*MariaDB Server es uno de los servidores de bases de datos más populares del mundo. Está hecho por los desarrolladores originales de MySQL y garantiza que se mantendrá de código abierto.*".

Se optó por esta alternativa al ser la más conveniente para nuestro proyecto. Se trata de un gestor rápido y escalable que, sumado a que se desarrolla como una base de datos relacional y ser un software de código abierto, lo convierten en el gestor idóneo para nuestra situación.

3.3 Entorno de desarrollo

Como entorno de desarrollo se empleó un ordenador portátil Medion Erazer X7833, propiedad del alumno. Este cuenta con un sistema operativo Windows 10 Home, en el cuál se llevaron a cabo todas las operaciones de desarrollo del proyecto. Entre sus características destacamos el sistema operativo de 64 bits, 16 GB de RAM, 4 núcleos y un procesador Intel Core i7-4710MQ CPU @ 2.50Ghz.

Como parte final del proceso de desarrollo de la aplicación, se desplegó esta misma en una Raspberry. Fueron necesarias algunas pequeñas modificaciones y adaptaciones en la implementación debido a notorias incompatibilidades entre comandos propios de Windows 10 y los de Raspberry Pi OS. Para trabajar contra esta, se optó por el establecimiento de conexiones remotas a través del protocolo SSH, empleando una VPN. Esto ha de ser imprescindible debido a la necesidad de realizar el trabajo remotamente, evitando la utilización de dispositivos periféricos adicionales (monitor, teclado, ratón, etc.) para ello.

3.3.1 Lenguajes de desarrollo

Como lenguajes de desarrollo han sido empleados los siguientes.

- **SQL**. Empleado en todas las funciones relacionadas con la gestión de la información almacenada en la base de datos. Entre estas operaciones destacamos operaciones de recuperación, agregación, modificación o eliminación de datos de la base de datos.
- **HTML, CSS y JavaScript**. Estos tres lenguajes fueron los utilizados para el desarrollo de la interfaz web en la que correrá la aplicación final. Con HTML establecimos la estructura de las página web, con CSS organizamos la presentación y aspecto de esta. Por último, pero no menos importante, JavaScript fue empleado como complemento a los dos anteriores. La principal función de este fue la de posibilitar la gestión de eventos causados por el usuario tales como pulsaciones de botones, modificaciones del DOM, detección de errores en formularios, entre otros. En este grupo incluimos a mayores la implementación del servidor Node.js, pues este está basado en el lenguaje JavaScript.
- **PHP**. El principal motivo de su empleo fue el de favorecer la conexión entre los servidores y la interfaz de usuario. Fue especialmente útil a la hora de realizar peticiones a la base de datos desde la propia página web, el gestionar el envío de formularios, la forma en la que se tramitaron los datos enviados por estos últimos o hasta el envío de correos electrónicos por parte de la web.

Capítulo 4

Metodología

Se entiende por metodología "el conjunto de procedimientos racionales utilizados para alcanzar un objetivo que requiera habilidades y conocimientos específicos [9]". La elección de una metodología constituye una de las partes primordiales a la hora de realizar cualquier proyecto, tanto de desarrollo software como de otro tipo. Partiendo de un enfoque teórico se definirán una serie de técnicas concretas para el correcto cumplimiento de los objetivos. En pocas palabras, será el método empleado que marcará los pasos a seguir para finalizar exitosamente las tareas constituyentes de nuestro proyecto.

4.1 Elección de la metodología

Para nuestro proyecto se ha decidido seguir la metodología Scrum. Para nuestro caso, se trata de la metodología perfecta, puesto que es necesario que la aplicación se adapte a las necesidades del usuario. Durante toda la realización del proyecto se ha intentado seguir fielmente todas las pautas enunciadas en la guía de Scrum [10], aunque en algunos casos presentamos discrepancias debido a que contamos con un anteproyecto y, en consecuencia, se conocerá el objetivo final del proyecto con lo que para nuestro caso se tratará de un Scrum "adaptado".

4.1.1 Conceptos previos

Antes de comenzar es preciso que se tengan claros una serie de términos que emplearemos a lo largo de la memoria.

- El **Scrum Team** o Equipo Scrum es el grupo formado por el "Scrum Master", el "Product Owner" y los Desarrolladores. Abordaremos este tema más profundamente en la sección 4.1.2.

- Los **Sprint** son considerados iteraciones de duración breve (aproximadamente de 2 a 4 semanas). En cada uno de estos se pretenderá completar con éxito los objetivos del proyecto, definidos en el "Product Backlog", además de añadir funcionalidades al producto.
- El **Product Backlog** es el documento público contenedor de la lista de las funcionalidades, descritas previamente, que se espera que contenga el producto final. Están también incluidas las prioridades de cada una de las funcionalidades. Solo puede ser modificado por el "Product Owner".
- Las **Tareas** son aquellos trabajos cuya realización nos ayudará a progresar hacia el objetivo final. Su duración suele ser de unas 16 horas.
- El **Sprint Backlog** constituye el conjunto de tareas asociadas a cada "Sprint". Serán los desarrolladores los encargados de repartirse la carga de trabajo.

4.1.2 Roles dentro del Scrum Team

Dentro de todo Scrum Team encontramos los roles explicados a continuación.

- El **Scrum Master**. Su principal cometido es el de garantizar el correcto cumplimiento de los objetivos, mediante la eliminación de todo obstáculo que surja, sin la necesidad de tener que organizar al resto del equipo.
- El **Product Owner** representa al cliente, no obstante, este rol puede ser asumido por el lider del proyecto. Su tarea será la de proporcionar el mayor valor posible al producto final.
- Los **Desarrolladores** constituyen el equipo de trabajo. Deberán contar con conocimientos técnicos adecuados para la correcta realización del proyecto. Suelen estar compuestos por entre 3 y 9 miembros.

4.1.3 Reuniones

Según la guía de Scrum, es precisa la realización de una serie de reuniones entre todo el "Scrum Team" a fin de exponer los objetivos logrados, problemas surgidos o cualquier temática relacionada con la realización del proyecto. Distinguimos cuatro tipos de reuniones, definidas a continuación.

- **Diaria**. Reunión de una breve duración (aproximadamente un cuarto de hora). Como su nombre indica, se llevará a cabo diariamente. En ella se tratan temas diversos como son los problemas surgidos, avances realizados y objetivos a cumplir antes de la próxima reunión.

- **Planificación del Sprint**. Evento de una duración estimada máxima de 8 horas. Se realiza siempre al comienzo de cada Sprint. En ella son definidas tareas y objetivos a alcanzar a lo largo del ”Sprint” en cuestión.

- **Revisión del Sprint**. Evento realizado a la finalización de cada ”Sprint”. Tendrá una duración máxima de unas 4 horas. En él se revisará todo el trabajo abordado a lo largo del Sprint, tanto finalizado como sin finalizar.

- **Retrospectiva**. Al igual que el anterior, se realiza tras la finalización del ”Sprint”. Esta reunión es organizada por el ”Scrum Master”. En ella se trata de identificar todos los aspectos, tanto positivos como negativos del proceso de realización del ”Sprint”, a fin de mejorar la ejecución de la metodología en futuros ”Sprints”.

4.1.4 Scrum aplicado a nuestro proyecto

Pese a que Scrum es la metodología más adecuada para realizar este trabajo, es necesario adaptarla a las características y limitaciones de un trabajo de fin de grado. A continuación, se explica cómo se ha aplicado Scrum en el desarrollo de este proyecto.

Roles

- Alfonso Torralba es el único desarrollador. Es el responsable de diseñar el sistema e implementarlo. Esto, obviamente, va en concreta de los principios de la guía de Scrum, la cual recomienda un mínimo de 3 miembros.

- Carlos Dafonte es el “Product Owner” dado que es la persona de la que partió la idea sobre la temática del proyecto. Es el principal responsable de comunicar cuál iba a ser el objetivo del proyecto y de su correcta realización, además de proporcionar el material necesario para su realización.

- Francisco Nóvoa es el “Scrum Master”, responsable de asegurar la efectividad de todo el equipo Scrum y de ayudar al equipo en la creación de incrementos de alto valor para el producto. También propuso ideas referidas a la mejora de la aplicación tales como herramientas útiles para el usuario o de implementación.

Reuniones

- **Diaria**. En este caso no se cumple con la guía de Scrum. Las reuniones diarias no fueron posibles debido a la ajustada agenda de los directores del proyecto y del alumno. No obstante, se organizaron reuniones periódicas semanales y no periódicas, siempre que fue necesario para solventar dudas o resolver problemas.

- **Planificación del Sprint**. Realizadas al comienzo de cada uno de los Sprints. En ellas se dictaban tareas y objetivos a cumplir hasta la próxima reunión. También se establecía la duración de la iteración que para nuestro caso eran de entre una semana y media o dos.
- **Revisión del Sprint**. Realizada al final de cada uno de los Sprints. En ella se revisaba todo el trabajo realizado a lo largo de todo el Sprint.

Capítulo 5

Planificación y estudio económico

A continuación se profundiza en las diferentes fases que componen el proceso de desarrollo del proyecto. Se expone la planificación seguida y se explica "grosso modo" qué tareas se llevaron a cabo en cada una de sus fases. Adicionalmete, se realiza el estudio económico del mismo, para estimar el coste del proyecto.

5.1 Calendario del proyecto

En la figura 5.1 (página 22) podemos ver una representación de la planificación expresada mediante un diagrama de Gantt. Inicialmente se estimó una duración similar para cada una de las tareas formantes, la cuál era de unas 2 a 3 semanas y el proyecto debería tomar cerca de 27 semanas en finalizarse. Por diversos motivos algunas tareas se alargaron en el tiempo como es el caso, por ejemplo, de la tarea "Product Backlog", la cual se alargó inevitablemente por motivo de las vacaciones de verano. Otro ejemplo es el "Sprint 3", cuya duración se alargó aproximadamente un mes debido a la excesiva carga de trabajo. De todos modos, a pesar de estos retrasos en algunas tareas, que en consecuencia ralentizaron la realización de todo el proyecto, otras supusieron un adelanto para este. Casos como el de las tareas "Sprint 1" y "Sprint 8", como resultado de una mayor velocidad de trabajo, aparición de menos contratiempos durante la implementación, entre otros, obtuvieron una notable reducción en su duración y, en consecuencia, se logró adelantar tiempo en la realización del proyecto. Como resultado final, a pesar de ligeros contratiempos, se consiguió terminar el proyecto en el plazo inicialmente establecido.

Cada una de las partes constituyentes del calendario de nuestro proyecto serán explicadas a continuación.

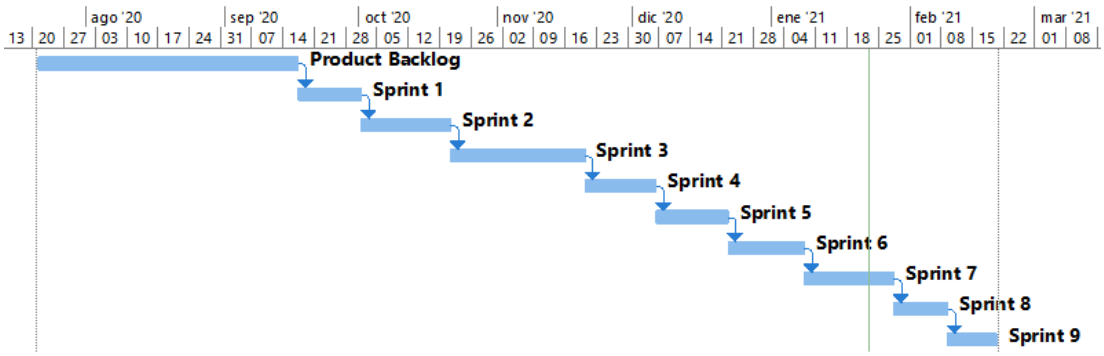


Figura 5.1: Diagrama de Gantt de la planificación del proyecto.

5.1.1 Product Backlog

21 de julio - 16 de septiembre

En estas primeras reuniones se buscó la temática más adecuada, tanto para el alumno como para el profesorado, para la realización del proyecto. Esta iteración tuvo una duración excesiva en consecuencia de las vacaciones de verano. Como tarea para la siguiente iteración se estableció el indagar sobre las Raspberry y los puertos de consola de los dispositivos de red, elementos con los que el alumno nunca había tenido contacto. Se definieron todos los objetivos a alcanzar durante el desarrollo de la aplicación, bien sean implementaciones, herramientas u otros relacionados.

5.1.2 Sprint 1: Preparación del entorno y familiarización con este

17 de septiembre - 30 de septiembre

El objetivo de este "Sprint" es verificar la factibilidad del proyecto y seleccionar las tecnologías adecuadas para llevarlo a cabo. Es necesario además construir el entorno de desarrollo, que requiere de determinados componentes hardware para realizar pruebas de funcionalidad durante el desarrollo. Al comienzo de esta iteración el alumno recibe dicho material imprescindible para la realización de la primera fase del proyecto: un router Cisco 4300 Series, un concentrador multipuerto USB y el cableado necesario (adaptadores USB-RS232 y cables de consola).

En este Sprint se llevaron a cabo todas las tareas relacionadas con la preparación del entorno de desarrollo. Al utilizar Windows 10, se decide emplear XAMPP. Este es un paquete de software libre que consiste en un sistema de gestión de bases de datos, el servidor Apache y el intérprete para el lenguaje PHP. A su vez, se indagó sobre diferentes tecnologías que podrían resultar de utilidad durante el desarrollo.

En este Sprint el alumno comenzó con la elaboración de la memoria, la cual se fue actualizando a lo largo de todos los Sprint según se iba implementando la práctica.

5.1.3 Sprint 2: Toma de contacto con Node.js

1 de octubre - 20 de octubre

Se desarrolla la primera versión de software desplegada en un servidor Node.js, que permite intercambiar caracteres entre el entorno de desarrollo y el dispositivo de red. Fue necesario recopilar y analizar diferentes librerías y seleccionar las más adecuada. Se diseña también una web simple, alojada en el servidor Apache, donde se muestra la consola del dispositivo de red para comprobar el correcto funcionamiento del código desarrollado, las librerías y el servidor Node.js. En este ”Sprint” se valida la selección de tecnologías.

5.1.4 Sprint 3: Comienzo del desarrollo web

21 de octubre - 19 de noviembre

Se comienza con el desarrollo la aplicación web que sirve de interfaz al sistema. En esta iteración se implementan los procesos de ”login”, el registro y la ventana de inicio. Es primordial que la aplicación sea multiusuario y la interfaz sea sencilla y ligera.

Se corrigen también algunos errores detectados en el código del servidor Node.js, y se añaden algunas mejoras, entre las que destacamos un sistema de logs. Por último, pero no menos importante, creamos la base de datos, con la tabla de usuarios, en la que se almacena la información relativa a los nuevos usuarios registrados.

Como se puede observar, la duración de esta iteración es se incrementa debido a la incorporación de funcionalidades no previstas inicialmente, como el servicio de logs.

5.1.5 Sprint 4: Optimización y continuación del desarrollo web

20 de noviembre - 5 de diciembre

En esta nueva iteración se añaden mejoras con respecto a la implementación anterior. Ahora la aplicación verifica la autenticidad de los usuarios mediante el envío de correos electrónicos, verificando que se trata de miembros de la Universidade da Coruña. Se añaden nuevas comprobaciones en los campos de los formularios de ”login” y registro.

Como nueva herramienta, se añade un panel de administración a la web. Dentro de este nuevo panel, se implementan las herramientas relacionadas con la gestión de usuarios.

Con respecto software de intercambio de caracteres entre las conexiones de consola y la web, desplegado en el servidor Node.js, se añade una mejora importante para garantizar la disponibilidad y el control de acceso a los dispositivos: el servidor es capaz de detectar inactividad por parte del usuario o incluso si el usuario ha perdido la conexión con el servidor, en cuyo caso este se autocerrará y redirigirá al usuario, en caso de haberlo, a la página de inicio de sesión.

5.1.6 Sprint 5: Mejoras del panel de administración y adaptación de la base de datos

6 de diciembre - 21 de diciembre

Se añaden nuevas funcionalidades al panel de administración. Entre otras, destacamos las que permiten gestionar los diferentes dispositivos de red disponibles, que constituyen el POD ("Pool of Devices") para cada grupo de alumnos. Se implementa también una nueva utilidad que permite a los administradores crear asignaciones entre grupos de usuarios y grupos de dispositivos o PODs. Como consecuencia de esto, fue necesario añadir dos nuevas tablas en la base de datos; una asociada a los dispositivos y otra asociada a las asignaciones entre grupos de dispositivos y de usuarios.

Se incorpora también al panel de administración el apartado de logs, en el que se pueden revisar los registros de actividad de los diferentes dispositivos de red.

Como parte adicional, se comienza a indagar sobre las diferentes alternativas disponibles para realizar el completo despliegue de nuestra aplicación en una Raspberry Pi.

5.1.7 Sprint 6: Combinación de Nodejs y Apache

22 de diciembre - 7 de enero

En este apartado se integra el software desplegado en el servidor Node.js con la aplicación web. Para lograrlo, es necesario realizar una serie de modificaciones en la web, a fin de que esta sea capaz de invocar los servicios Node.js cuando es necesario. Se diseña también un fichero XML que contiene las configuraciones de los servicios de intercambio para cada dispositivo de red.

Se prueban diferentes configuraciones con routers Cisco (incluidas conexiones en diferentes puertos del concentrador USB) para verificar el correcto funcionamiento de la nueva implementación. Se añaden también nuevas medidas de seguridad para esta nueva funcionalidad, entre las que destacamos la limitación de un único usuario por cada consola de los dispositivos de red o el uso de " Tokens" para evitar el acceso fraudulento a las mismas.

5.1.8 Sprint 7: Despliegue de la aplicación web en la Raspberry Pi

8 de enero - 27 de enero

Una vez concluidas todas las implementaciones, incluidas en el Product Backlog, era necesario al despliegar la aplicación en el entorno de producción, es decir, en ordenador de bajo coste. Como la Raspberry no se encontraba físicamente ubicada junto al alumno, se decidió utilizaron diferentes herramientas para acceder al dispositivo remotamente de forma segura: una VPN de acceso remoto para el acceso seguro a la red de laboratorio donde se ubica el equipo de salto (Raspberry) y protocolos de comunicaciones seguros como FTPS y SSH para la transferencia de ficheros y la administración remota, respectivamente.

Una vez estuvieron establecidas las vías de acceso, se preparó el nuevo entorno. Se instaló Apache Server, el servidor de base de datos MariaDB y el resto de componentes necesarios para el correcto funcionamiento de la aplicación.

Además, fue necesario realizar una serie de adaptaciones en las implementación realizada en el entorno de desarrollo (entorno Windows) para desplegarlo en un entorno Linux utilizado en la Raspberry.

Por último se procedió al despliegue de la aplicación. Los pasos detallados del proceso de instalación se pueden ver en el anexo correspondiente.

5.1.9 Sprint 8: Revisión

28 de enero - 8 de febrero

Al concluir con el despliegue en la Raspberry se pasó a la fase de revisión. En esta nueva iteración se comprobó si todas las funcionalidades, incluidas en el Product Backlog estaban correctamente implementadas. Esta revisión fue realizada tanto por el alumno como por los directores, teniendo estos últimos la potestad de decidir si realizar algún cambio, tanto de diseño como de implementación.

En esta iteración se decide hacer un añadido de última hora. Se creó un nuevo sistema de logs para controlar los accesos no autorizados a las consolas de los diferentes dispositivos de red.

5.1.10 Sprint 9: Pruebas

9 de febrero - 19 de febrero

En este último Sprint se realizaron pruebas de unidad de la aplicación. El alumno y los directores del proyecto, realizaron pruebas, como si de una situación real se tratase, de todas

las herramientas disponibles. Durante la realización de esta iteración se detectaron pequeños errores de implementación, los cuáles fueron correctamente solventados. Así mismo, el alumno concluyó con la elaboración de la memoria.

5.2 Evaluación de costes

A continuación se detalla el cálculo de costes asociado con la elaboración de este proyecto.

Primero comenzaremos analizando el caso de los recursos humanos. Para el caso del alumno se calcula que invirtió una media de 5 horas al día en la realización del trabajo, pues era dedicaba la totalidad de su tiempo disponible a su realización. El alumno trabajó, por lo general, de lunes a viernes, lo que hace un total de 25 horas a la semana. El proyecto en sí, tuvo una duración estimada de 27 semanas, sin contar los meses de vacaciones de verano. Multiplicando estas 27 semanas por las 25 horas a la semana invertidas por el alumno hacen un total de 675 horas. El convenio colectivo [11] establece que la duración de la jornada ordinaria máxima de trabajo efectivo será de 1.800 horas anuales. Este mismo convenio, establece un salario anual bruto de 25.189,02 € para un analista de sistemas. Haciendo los cálculos oportunos obtenemos un salario de 13,99 € por hora trabajada. Con esto podemos obtener la remuneración total para nuestro proyecto que viene a ser 675 horas por 13,99 € por hora, que hacen un total de **9.443,25 €**.

Ahora analizaremos el caso de los directores del proyecto. Para nuestro proyecto, ambos directores trabajaron en conjunto en todo momento. Debido a compaginar sus tutorías y clases con el proyecto, se estima una media semanal de 4 horas invertidas en él por ambos. Esta media multiplicada por las 27 semanas que duró el proyecto hace un total de 108 horas. A la hora de realizar el cómputo de la remuneración tenemos que tener en cuenta la ocupación profesional de cada director. Primero, para el caso de Carlos Dafonte. Carlos es un profesor titular de universidad, y como bien se refleja en la tabla de retribuciones correspondiente [12], su salario anual bruto es de 35.095,54 €. Según las condiciones de trabajo del profesorado de educación superior de España [13], se establece una jornada laboral de 1.685 horas anuales para los centros educativos con ánimo de lucro. De esto podemos calcular su salario por hora, el cual es de 20,83 €. Multiplicando esta cifra por el total de horas invertidas en la realización del proyecto obtendremos el coste total por el trabajo de Carlos, que es de **2.249,64 €**. Para calcular el coste asignado al trabajo realizado por Francisco Nóvoa es exactamente idéntico al de Carlos, a excepción de que Francisco es un profesor contratado doctor. Revisando su tabla correspondiente de remuneraciones [14] obtenemos un salario anual bruto de 34.963,22 €. Dividiendo este salario anual entre las 1.685 horas de jornada anual obtenemos una remuneración de 20,75 € por hora. Multiplicando de nuevo esta cifra por el total de horas invertidas en el proyecto obtendremos el coste asociado al trabajo de Francisco, que asciende a un total

de **2.241 €**.

En segundo lugar, haremos un análisis de los costes materiales:

- **Raspberry Pi 3**. Al tratarse de un modelo relativamente antiguo se estima un coste aproximado de unos **50 €**.
- **Hub USB 3.0**. El multipuerto tiene un costo aproximado de **70 €**.
- **Cables de consola**. Cada unidad vale en torno a 10 €. Para nuestro proyecto usamos 5 unidades, lo que hace un costo total de **50 €** aproximadamente.

Sumando todos los costos anteriores, tanto humanos como materiales, obtendremos una aproximación de lo que supondría económicamente producir nuestro proyecto. El coste total para nuestro proyecto sería de unos **14.103,89 €** brutos.

Capítulo 6

Trabajo desarrollado

En este apartado entraremos en detalle en los aspectos más importantes de nuestro proyecto. Se profundizará en temas tales como la arquitectura de alto nivel del sistema, el diseño de detallado del software, la integración de todas las tecnologías empleadas, así como ejemplos de implementación, funcionalidades y pruebas desarrolladas.

6.1 Diseño arquitectónico

Podemos apreciar la arquitectura de nuestro escenario en la figura 6.1 (página 30). Como se puede apreciar a simple vista, se trata una combinación de los dos tipos de gestión de dispositivos de red explicados en el apartado 2.2 (página 6). Por un lado, podemos apreciar la gestión "fuera de banda" en la Raspberry, la cual está directamente conectada a Internet y a la que tendrán acceso remoto cualquier usuario autorizado. La gestión "en banda" la podemos apreciar en la parte superior de la ya mencionada figura. En ella podemos apreciar un conjunto de dispositivos de red interconectados entre sí y, a su vez, a la Raspberry[1].

Otro punto muy importante a la hora de analizar la arquitectónica de nuestro proyecto es analizar la forma en la que interaccionan entre sí las tecnologías utilizadas. Esto podemos verlo de una forma muy simplificada en la figura 6.2 (página 30). Nuestro servidor Apache, contenedor de nuestra aplicación web, será capaz de autoejecutar los servidores Node cuando se precisen. A su vez, este y los servidores Node interaccionarán con la base de datos, haciendo las peticiones pertinentes cuando sea preciso.

[1] Nuestra aplicación cuenta con una arquitectura escalable. El número de dispositivos de red conectados dependerá en gran medida del hardware disponible, es decir, a más puertos USB disponibles, más dispositivos de red se podrán conectar.

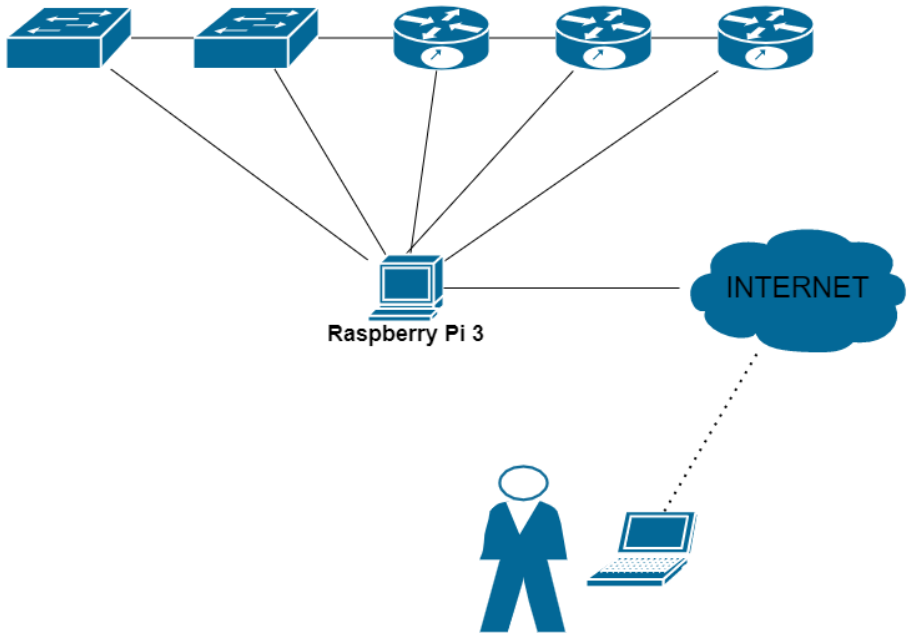


Figura 6.1: Diseño arquitectónico del proyecto.

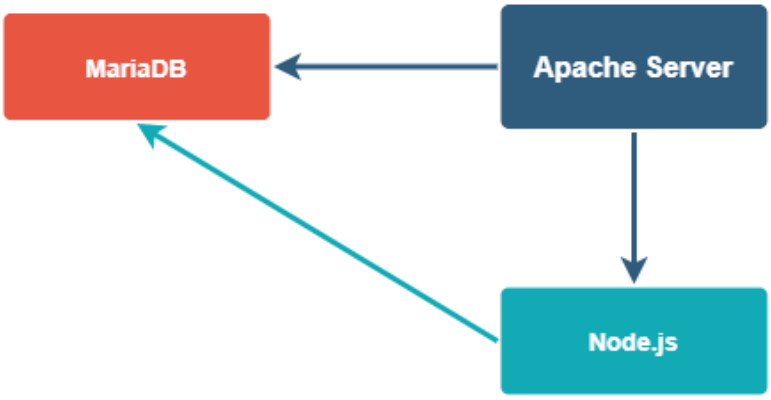


Figura 6.2: Interrelación de las tecnologías empleadas.

6.2 Diseño: Base de datos, XML y diagramas de secuencia

El diseño es un punto muy importante a tener en cuenta a la hora de desarrollar una aplicación web. En este nuevo apartado abordaremos temas tales como el análisis del modelado de la base de datos, incluyendo sus debidos diagramas y explicaciones, así como el estudio de diversos diagramas de secuencia que harán referencia a algunas funcionalidades de la aplicación, a nuestro parecer, las más destacables y de más complicada comprensión.

6.2.1 Esquema relacional de la base de datos

Diseño lógico

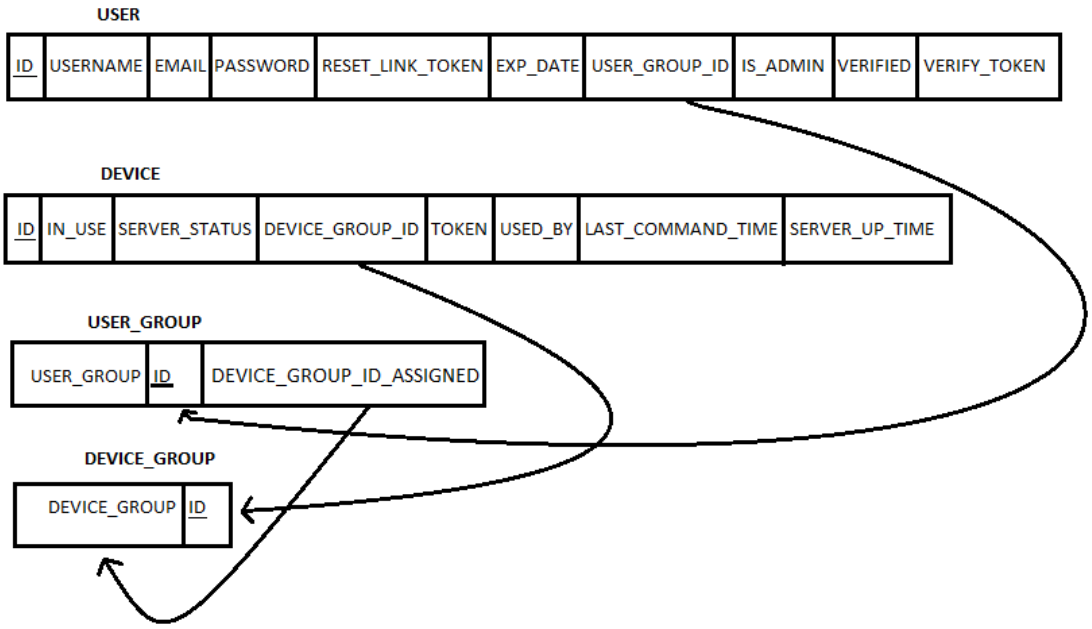


Figura 6.3: Diseño lógico de la base de datos.

Diseño conceptual

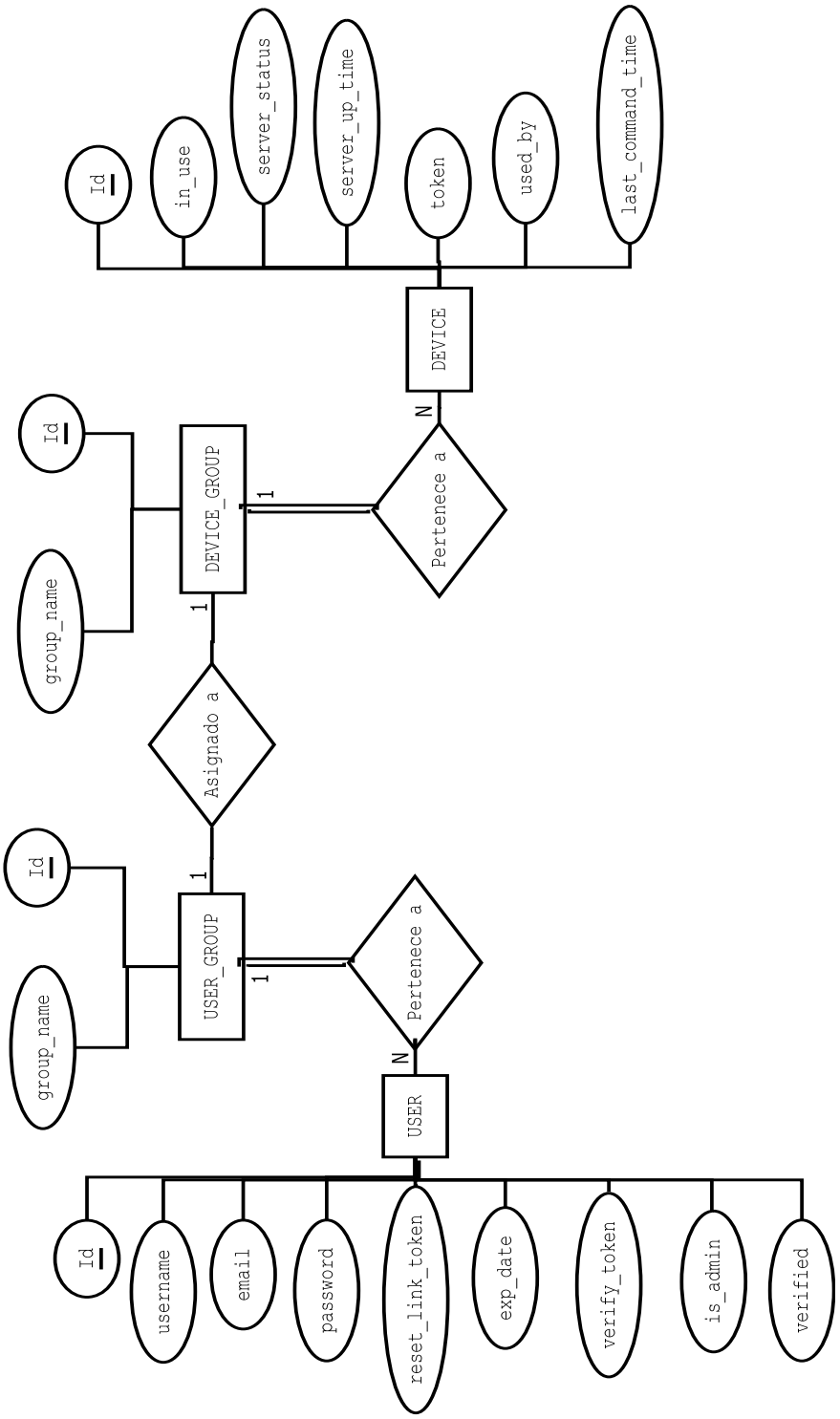


Figura 6.4: Diseño conceptual de la base de datos.

Descripción del modelado

Se buscó un diseño para la base de datos lo más sencillo posible, pero a su vez, lo más eficiente para poder soportar toda la carga de trabajo asociada a nuestra aplicación. Nuestra base de datos está compuesta por 4 tablas explicadas a continuación:

- **USER**. Contendrá información relativa a los usuarios registrados. Entre esta información destacamos su nombre de usuario, email, contraseña y varios "tokens", estos últimos necesarios para verificar la identidad del usuario.
- **USER_GROUP**. Esta almacenará información referida a los diversos grupos de usuarios existentes. Esta tabla solo contendrá como atributos su Id, el propio nombre del grupo y el Id del grupo de dispositivos al que estará asignado, en caso de estarlo.
- **DEVICE**. Contendrá información relativa a los servidores de los dispositivos de red registrados en la aplicación. Entre sus atributos más destacables mencionamos la hora en la que se inició por última vez su servidor Node, el estado de este servidor (Encendido/Apagado), quién lo está usando, en caso de estar este encendido y un "token" para evitar accesos fraudulentos.
- **DEVICE_GROUP**. Esta almacenará información referida a los diversos grupos de dispositivos de red existentes. Esta tabla solo contendrá como atributos su Id y el propio nombre del grupo.

En cuanto a la relación existentes entre las presentes tablas es muy sencilla de entender. Un usuario podrá pertenecer a único grupo de usuarios, mientras que cada grupo podrá tener un número ilimitado de usuarios. Aquí podemos ver la primera relación entre las tablas *USER* y *USER_GROUP*. Esta misma explicación nos vale para el caso de los dispositivos. Un dispositivo podrá pertenecer a un solo grupo de dispositivos, mientras que estos últimos podrán estar compuestos de múltiples dispositivos. Aquí vemos la segunda relación entre las tablas *DEVICE* y *DEVICE_GROUP*. Por otra parte, podrán existir asignaciones entre grupos de usuarios y dispositivos, que serán las que permitirán organizar a que dispositivos de red tendrá acceso cada usuario. Aquí podemos ver la última de las relaciones entre las tablas *DEVICE_GROUP* y *USER_GROUP*, siendo esta última la que almacenará el atributo que marcará dichas asignaciones.

6.2.2 Fichero XML

Como bien se explicó en el apartado anterior, la tabla *DEVICE* almacena información relativa más al servidor Node del propio dispositivo, que al propio dispositivo. Se decidió tomar como alternativa el diseño de un documento XML, a fin de almacenar la información más

específica de cada dispositivo de red. Este método resulta mucho más sencillo para el administrador a la hora de modificar cierta información acerca del dispositivo. Comparando esta con otras opciones disponibles, como es la situación de almacenar dicha información en la propia tabla de dispositivos de la base de datos, se evita la necesidad de hacer múltiples consultas, siendo todo el proceso mucho más rápido y directo.

La estructura de nuestro XML contará con 10 atributos explicados a continuación:

- **name**. Nombre del dispositivo de red.
- **port**. Puerto en el que correrá el servidor Node del dispositivo de red en cuestión.
- **com**. Puerto de consola al que estará conectado el dispositivo de red.
- **description**. Descripción o breve comentario relacionado con el dispositivo de red.
- **baudRate**. Velocidad de transmisión medida en bits por segundo.
- **dataBits**. Longitud de los bits de datos transmitidos.
- **stopBits**. Bits de parada. Marcan el final de cada paquete transmitido.
- **flowControl**. Booleano para establecer el control de flujo.
- **lock**. Booleano utilizado para prevenir que otros procesos abran el puerto.
- **idDb**. Id de la base de datos del propio dispositivo de red, para así mantener una relación directa con la información de su servidor Node.

Un ejemplo de la estructura del fichero XML, para el caso de un único dispositivo de red, puede ser apreciada a continuación.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <devices>
4    <device>
5      <name>ISP</name>
6      <port>8080</port>
7      <com>/dev/ttyUSB2</com>
8      <description>Dispositivo ubicado en la planta 2 de la Facultad
       de Informática de A Coruña.</description>
9      <baudRate>9600</baudRate>
10     <dataBits>8</dataBits>
11     <stopBits>1</stopBits>
12     <flowControl>false</flowControl>
```

```
    <lock>false</lock>
    <idDb>5</idDb>
  </device>
</devices>
```

6.2.3 Diagramas de secuencia de la aplicación

A modo de afianzar los conocimientos sobre las diferentes tecnologías empleadas y como interactúan entre ellas, se proporcionan a continuación una serie de diagramas de secuencia. Se aportarán diagramas relacionados con funcionalidades interesantes del trabajo o cuyo entendimiento pueda resultar confuso a primera vista. Adicionalmente, se agregará para cada uno su debida explicación pertinente.

Diagrama de secuencia Nº 1 - Acceso a la consola de un dispositivo de red.

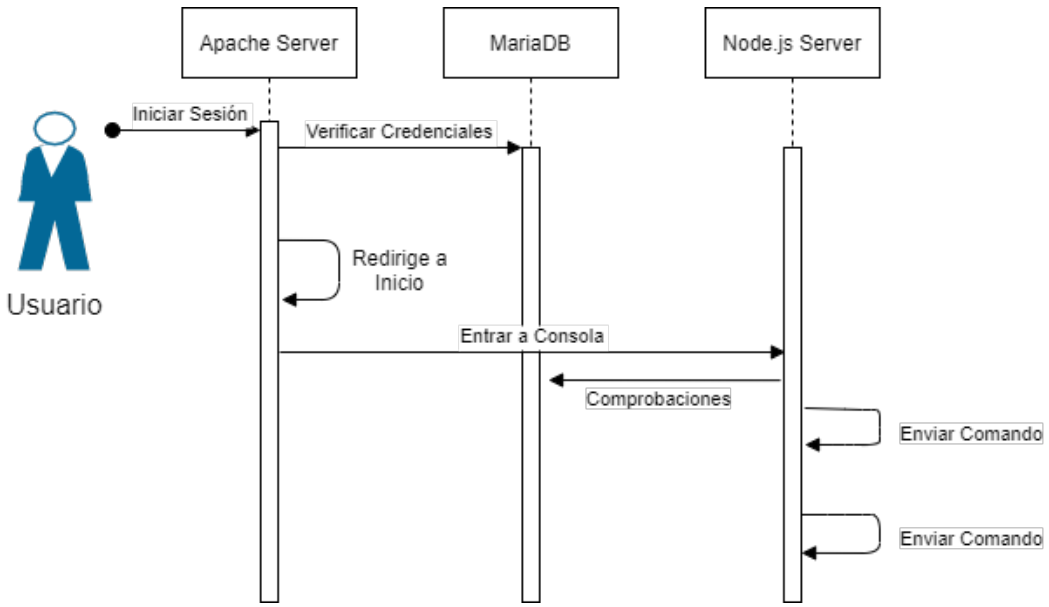


Figura 6.5: Diagrama de secuencia Nº 1 - Acceso a la consola de un dispositivo de red.

En el presente diagrama se refleja la situación de un usuario que desea acceder a la consola de un dispositivo de red e interaccionar con ella. En primer lugar, el usuario deberá iniciar sesión en la aplicación, para esto interactúa contra el servidor Apache el cual, a su vez, se comunica con la base de datos para corroborar que las credenciales proporcionadas por el usuario sean las correctas. Una vez verificada la identidad del usuario, el servidor Apache lo redirigirá a su página de *Inicio*. Dentro de esta, el usuario seleccionará, de entre sus dispositivos de red asignados, al que desee acceder a su consola. Ahora el servidor Apache cede

el control al servidor Node correspondiente. Este realizará los chequeos oportunos contra la base de datos (revisar si la consola ya se encuentra en uso, revisar el "token" de acceso para evitar accesos fraudulentos, entre otros). Si supera esta fase de comprobaciones el usuario ya tendrá acceso completo a la consola y podrá enviar tantos comandos como desee mientras dure su sesión.

Diagrama de secuencia Nº 2 - Administrador crea nuevo grupo de usuarios.

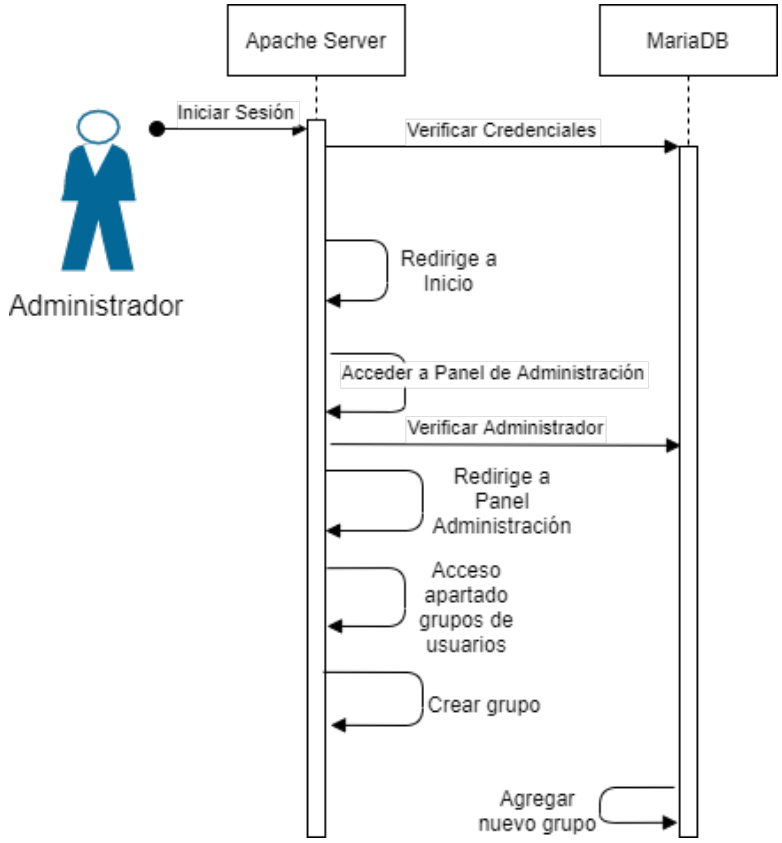


Figura 6.6: Diagrama de secuencia Nº 2 - Administrador crea nuevo grupo de usuarios.

En esta nueva situación nos encontramos con un administrador que desea crear un nuevo grupo de usuarios. Nuevamente, primero ha de iniciar sesión en la aplicación. Este paso es exactamente idéntico al del diagrama previo. Una vez haya iniciado sesión el usuario se dispondrá a acceder al Panel de Administración. Como se puede comprobar, de momento toda interacción del usuario se realiza contra el servidor Apache, excepto el caso de verificar las credenciales de inicio de sesión, en la que se conecta el servidor Apache con la base de datos. Antes de que el administrador tenga acceso completo al panel, el servidor Apache ha de asegurarse de que realmente se trata de un administrador y, para ello, se comunicará con la base de datos. Una vez se ha otorgado el acceso al Panel de Administración, el usuario se dirige

al apartado correspondiente de creación de grupos. Finalmente, la base de datos agregará el nuevo grupo en su registro.

Diagrama de secuencia Nº 3 - Usuario se registra en la aplicación web.

En este último ejemplo podemos ver el escenario relacionado con el registro de un nuevo usuario. Primero de todo, el usuario interacciona contra el servidor Apache, accediendo a su página de registro. El usuario completará el formulario con sus datos y los enviará, datos los cuales serán chequeados contra la base de datos para comprobar su validez. En caso de así serlo, el servidor Apache enviará al usuario un correo electrónico de confirmación de la cuenta y la base de datos almacenará los datos del nuevo usuario. Una vez el usuario reciba el e-mail y acceda al hipervínculo, en él incluido, será redirigido a la aplicación web y, por detrás, el servidor Apache interaccionará con el servidor de base de datos para actualizar el estado de ese usuario a "Verificado".

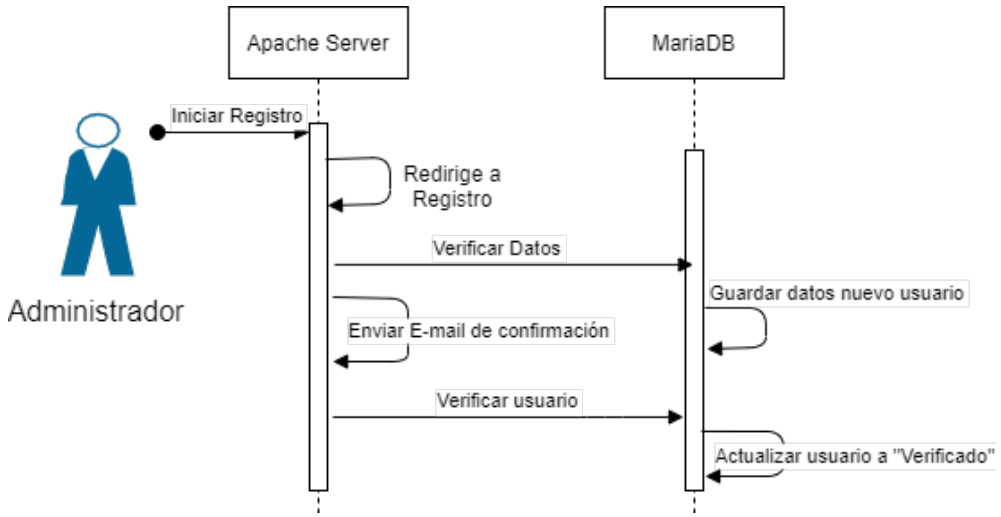


Figura 6.7: Diagrama de secuencia Nº 3 - Usuario se registra en la aplicación web.

6.3 Implementación

En este nuevo apartado abordaremos profundamente todos los detalles de implementación relacionados con nuestra aplicación web[2]. Se explicará que función realiza cada uno de los archivos constituyentes de la carpeta raíz de Apache, incluyendo los archivos del servidor Node.js. Para el caso de los archivos del servidor Apache, se explicarán los métodos empleados

[2] Se abordarán fundamentalmente las codificaciones utilizando PHP, pues todo el código HTML y CSS empleado no da lugar a duda alguna.

para llevar a cabo funciones como son el caso de gestionar formularios enviados o la comunicación y realización de peticiones hacia la base de datos. En el caso del servidor Node, se explicará la relación guardada entre los archivos que lo forman, así como información sobre su implementación para comprender su funcionamiento. Por último, se hará una explicación más detallada de los casos de uso[3] desarrollados en los diagramas de secuencia del apartado anterior, explicando paso a paso y con referencias al código, como va ocurriendo cada evento.

6.3.1 Directorio raíz del servidor Apache

Cuando un usuario accede a nuestra web y carga la página de "*Index*", lo que está haciendo por detrás el servidor Apache es gestionar esa petición por parte del usuario y devolverle el contenido de la página solicitada. Todas las páginas de nuestra aplicación web están almacenadas en el directorio "*/var/www/html*", que es la carpeta raíz de nuestro servidor Apache. Mencionar que empleamos ficheros HTML con código PHP embebido. Se ha elegido esta aproximación para el desarrollo con el objetivo de obtener una aplicación web lo más ligera posible, puesto que se va a albergar en un hardware con recursos limitados. Por esta misma razon se descarta el empleo de una arquitectura Modelo-vista-controlador, pues penalizaría mucho el rendimiento en la Raspberry.

Los archivos o directorios que podemos encontrar en dicho directorio son los siguientes:

- **.htaccess**. Archivo en el que se definen las directivas de configuración para cada directorio.
- **index.php**. Contiene toda la información acerca del "*Index*" de nuestra página.
- **inicio.php**. Aquí es donde se llevará a cabo todo lo relacionado con la verificación de credenciales del usuario una vez inicie sesión. En caso de que sean válidas, se mostrará al usuario su página de inicio con los dispositivos de red que le hayan sido asignados. En caso de tratarse de un administrador se mostrarán todos los dispositivos de red registrados en la aplicación.
- **register.php**. Contiene todo lo relativo al apartado de registro de nuestra web.
- **forbidden.php**. Contiene la página de prohibido el acceso (Error 403), el cuál saltará cuando un usuario intente acceder a la consola de un dispositivo de red de forma fraudulenta.
- **forgot.php**. Contiene todo lo referido al apartado de "*Olvidar contraseña*", donde el usuario podrá recuperar sus credenciales de inicio de sesión solo aportando el correo electrónico con el que se registró.

[3] En los apéndices se incluye un profundo análisis de todos los casos de uso presentes en la aplicación.

- **logout.php**. Contiene el código PHP necesario para permitir que los usuarios puedan cerrar su sesión en nuestra página web.
- **verify.php**. El apartado de "*Verificar usuario*" estará contenido dentro de este fichero. Cuando el usuario sea redirigido por el e-mail de verificación de cuenta, será llevado a esta página, la cual realizará unos chequeos previos (nombre de usuario, token..) para verificar la identidad del usuario.
- **Directorio *web***. En este directorio están contenidos el "*Header*" y *Footer* de todos los apartados de nuestra web. Destacar que el "*Header*" será capaz de detectar si un usuario se trata de un administrador, en cuyo caso mostrará la opción de acceder al *Panel de Administración*.
- **Directorio *server_node***. Aquí estará contenido todo lo relacionado con los servidores Node.js, que será explicado con más detalles en el siguiente apartado.
- **Directorio *css***. Contendrá todos los ficheros de hoja de estilos en cascada empleados para el diseño de todos los apartados de nuestra web. Nótese que uno de ellos es pertenencia de Bootstrap.
- **Directorio *web_config***. En este directorio encontraremos dos archivos. El primero de ellos, "*configuration_properties.php*", será el que contenga toda la información de acceso a la base de datos y al cual recurrirán el resto de ficheros PHP siempre que sea preciso realizar alguna operación contra esta. El segundo de los ficheros es "*devices_info.XML*". Este se trata de un fichero XML, el cual ya fue explicado con anterioridad en otro apartado, el cuál contendrá información relativa sobre los dispositivos de red registrados en la aplicación.
- **Directorio *images***. Contendrá las imágenes empleadas en algún momento por los ficheros PHP. Ejemplos de estas pueden ser el logo de la Universidad de A Coruña o el propio fondo de nuestra página web.
- **Directorio *PHPMailer***. Descargado desde su página oficial de Git. Es una biblioteca de códigos para enviar correos electrónicos de manera fácil y segura a través de un código PHP desde un servidor web. Para nuestro caso, será empleado para enviar los e-mails de verificación de cuenta y de recuperación de contraseñas.
- **Directorio *jquery***. Descargado desde la página oficial de JQuery. Se prefirió trabajar contra esta localmente debido a problemas de conectividad ocasionados por la VPN utilizada para establecer un túnel seguro contra la Raspberry. Se trata de una biblioteca multiplataforma de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, entre otras funcionalidades.

- **Directorio *admin***. Contiene todos los archivos que conforman el "*Panel de Administración*" de nuestra web. Dentro de este directorio encontramos:
  - **admin-pan.php**. Albergará la página de inicio del "*Panel de Administración*".
  - **devices.php**. Contenedor del apartado "*Dispositivos*". Aquí el administrador podrá ver los dispositivos registrados en la aplicación y dispondrá de opciones para poner editarlos o eliminarlos.
  - **edit-device.php**. Contiene todo el código necesario para mantener el apartado *Editar Dispositivo*, desde el cuál el administrador podrá modificar la información relativa a este.
  - **new-device.php**. Fichero PHP que contendrá el apartado "*Crear Dispositivo*", desde el cual el administrador podrá registrar nuevos dispositivos en la web.
  - **new-device-manual.php**. Albergará la página de creación de dispositivos manualmente. La funcionalidad de esta es similar a la anterior, solo que en este nuevo caso la web realizará automáticamente la creación de fila correspondiente en la base de datos. A continuación, devolverá al administrador un código XML que deberá modificar e insertar en el fichero "*devices_info.XML*" mencionado anteriormente.
  - **device-groups.php**. El apartado "*Grupos de Dispositivos*" estará contenido dentro de este fichero. Desde este el administrador tendrá la capacidad de visualizar los grupos de dispositivos existentes, ver sus dispositivos miembros y, en caso de así desearlo, eliminar alguno de estos del grupo.
  - **new-device-group.php**. En este fichero estará contenida la página "*Crear Grupo de Dispositivos*", la cual, como su propio nombre indica, permitirá al administrador la creación de un nuevo grupo de dispositivos.
  - **device-group-summary.php**. Contenedor del apartado web "*Grupos de Dispositivos Existentes*". En este, el administrador podrá ver un resumen de los grupos de dispositivos existentes con sus dispositivos miembros correspondientes.
  - **registered-users.php**. El apartado "*Usuarios Registrados*" será el desarrollado en este fichero PHP. En dicho apartado, el administrador podrá ver una lista del conjunto de usuarios registrados en la aplicación. Para cada uno de ellos, se le presentará la opción de otorgar o retirar el rol de administrador.
  - **user-groups.php**. Contiene la implementación necesaria para poder presentar el apartado *Administrar Grupos de Usuario*. Su contenido es similar al del fichero *device-groups.php* y permite también al administrador poder gestionar los diferentes grupos existentes de usuarios.

- **new-user-group.php**. Contiene el apartado "*Crear Grupo de Usuarios*". Igual que en el fichero anterior, su contenido es similar al de su equivalente en lo referido a dispositivos, en concreto al fichero "*new-device-group.php*". Permitirá al administrador la creación de nuevos grupos de usuarios.
- **user-group-summary.php**. En este fichero está desarrollado el apartado *Grupos de Usuario Existentes*. Es el equivalente al fichero *device-group-summary.php* para el caso de los dispositivos. En él, el administrador podrá ver un resumen de todos los grupos de dispositivos existentes con sus respectivos dispositivos miembros.
- **assignments.php**. Contendrá el apartado "*Asignaciones*" del "*Panel de Administración*". Desde este el administrador será capaz de crear nuevas asignaciones entre grupos de dispositivos y de usuarios.
- **existing-assigns.php**. Alberga el código necesario para proporcionar el apartado *Administrar Asignaciones Existente*. En este apartado el administrador podrá ver una lista con todas las asignaciones existentes, teniendo la opción de remover alguna si así lo desea.
- **logs.php**. En este fichero PHP esta albergado el código de la página de *Logs*. Aquí el usuario podrá revisar el registro de la actividad de todas las consolas de los servidores de red registrados en la aplicación.
- **reset-devices.php**. Fichero contenedor del apartado "*Resetear Dispositivos*". Aquí el administrador dispondrá de la opción de resetear el estado de los servidores Node de los diferentes dispositivos de red. Con esta funcionalidad lo que se pretende es actualizar, para cada uno de los dispositivos de red registrados, los valores de estado de sus servidores en la base de datos. Esto resulta especialmente útil en situaciones de fallas eléctricas. A modo de ejemplo, si se da el caso de que algún usuario se encontrase trabajando con la consola de algún dispositivo antes de producirse una falla eléctrica, al momento de reiniciar la Raspberry, la base de datos entenderá que el usuario todavía se encuentra trabajando contra el servidor. Con esta herramienta ponemos solución a este problema de una manera rápida y sencilla, restaurando el estado del servidor.
- **Directorio *css***. Directorio que alberga los archivos de hoja de estilos en cascada necesarios para definir la presentación de todas las páginas miembro del *Panel de Administración*. Aquí solo estarán contenidos los CSS propios del panel, aunque bien es cierto que en algunos casos son empleados los CSS propios del directorio padre.

Una vez explicados todos los archivos constituyentes de nuestra práctica, es hora de explicar puntos más concretos acerca de la implementación de los mismos. La manera en la que se

implementó el código de todos los ficheros PHP es muy similar. Como consecuencia de esto, en los subapartados posteriores serán explicados conceptos, como la forma en la que se llevó a cabo la gestión de eventos o las conexiones a la base de datos, de manera que sirvan como explicación global para todas las implementaciones.

Gestión de los eventos dentro de los ficheros PHP

Con gestión de eventos nos estamos refiriendo a la forma en la que trabajamos con los diferentes formularios y los datos emitidos por estos. Cada vez que un usuario completa todos los campos de un formulario y lo envía, la web por detrás está enviando una petición POST conteniendo todos los campos del formulario. Nosotros vamos a trabajar con estas peticiones POST. Utilizando PHP capturaremos estas peticiones POST y podremos realizar los ajustes pertinentes con los valores aportados por el usuario a través del formulario, bien sea contra la base de datos u otros. Podemos ver ahora un ejemplo claro de como capturaríamos un formulario recién enviado.

```
1 <?php
2 if ($_SERVER['REQUEST_METHOD'] == "POST") {
3     $variableCampo1 = $_POST['campo1'];
4     $variableCampo2 = $_POST['campo2'];
5
6     ...  // Code
7 }
8 ?>
```

Nótese en las líneas 4 y 5 la forma en la que podríamos recuperar los valores de los diferentes campos del formulario, asignándolos a unas nuevas variables a fin de que resulte más sencillo trabajar con ellas (por ejemplo, para actualizar algún atributo de alguna tabla de la base de datos).

Hasta aquí todo bastante sencillo, pero, ¿cómo podemos actuar en caso de tener dos formularios en el mismo apartado web? Observemos los siguientes fragmentos de código.

```
1     <form action='' method='post'>
2         <label for="fname">Nombre:</label><br>
3         <input type="text" id="fname" name="fname"><br><br>
4         <input type='submit' name='form1' value='ENVIAR' />
5     </form>
6
7     <form action='' method='post'>
8         <label for="sname">Apellido:</label><br>
```

```
9           <input type="text" id="sname" name="sname"><br><br>
10          <input type='submit' name='form2' value='ENVIAR' />
11      </form>
```

```
1  <?php
2      if($_SERVER['REQUEST_METHOD'] == "POST" and
       isset($_POST['form1'])) {
3      $nombre = $_POST['fname'];
4
5          ...  // Code
6
7      }
8
9      if($_SERVER['REQUEST_METHOD'] == "POST" and
       isset($_POST['form2'])) {
10     $apellido = $_POST['sname'];
11
12         ...  // Code
13
14     }
15 ?>
```

En el primero de los dos fragmentos podemos ver un ejemplo simple de representación de dos formularios en HTML. En el segundo fragmento veremos la forma en la que PHP trataría cada formulario enviado. La principal diferencia con lo explicado anteriormente es que, si nos fijamos en la implementación de cada formulario, para el elemento "*input*" asociado al botón de enviar en ambos formularios, hemos defifino la variable "*name*". Esta será el que permita al código PHP averiguar qué formulario es el que se ha enviado. Si nos fijamos en el segundo fragmento de código, en las líneas 3 y 10, concretamente en la segunda condición, vemos como hace una comprobación de si se han definido dichas variables, las cuales solo estarán definidas al momento del envío del formulario en cuestión.

Como observamos, es bastante sencillo de entender. En toda la aplicación, solo se complica en el apartado "*Inicio*", aunque la idea base sea la misma, este caso aislado contará con una pequeña modificación.

Cada uno de los dispositivos de red, asignados a cada usuario, mostrados en la pantalla de "*Inicio*", constituirá un formulario por sí mismo. A continuación, podemos ver un ejemplo de esta implementación.

```
1  <?php
2  echo "<form action='' method='post'>";
3      echo "<p style='margin-left: 2em'><b> • NombreDispositivo </b>";
```

```
4     echo "<input style='background-color:red;' name='namedevice'
      value='NombreDispositivo' hidden/>";
5     echo "<input type='submit' class='large green button'
      name='enterConsole' value='Entrar a la Consola    '/><br>";
6     echo "<label style='padding-top: 8px;'>Descripción del
      Dispositivo</label>";
7 echo "</form><br>";
8 ?>
```

```
1 <?php
2 if($_SERVER['REQUEST_METHOD'] == "POST" and
      isset($_POST['enterConsole'])) {
3     initiateServer($conn, $_POST['namedevice'], $username);
4 }
5 ?>
```

Estos ejemplos de código están obtenidos directamente del archivo "*inicio.php*". Se han realizado algunas pequeñas modificaciones en los mismos, a fin de que sea mucho más sencillo de entender. Para entrar en contexto, el primer fragmento de código estaría incluido dentro de un bucle "*While*" que imprimirá todos los dispositivos de red asignados al usuario en cuestión. El segundo fragmento nos muestra la forma en la que PHP captura cuando un usuario quiere entrar a la consola de algún dispositivo.

Si nos fijamos en el primer fragmento, en la línea 5 concretamente, observamos que hay un elemento de tipo "*Input*" con el atributo "*hidden*", es decir, que no aparecerá a la vista del usuario. Este elemento tiene como valor predefinido el nombre del dispositivo en cuestión (para ese formulario concreto). Una vez sea enviado el formulario, el código PHP captará dicha petición POST y, gracias a este elemento, será capaz de ejecutar el servidor Node correcto.

En el segundo fragmento destacamos la llamada a la función "*initiateServer*", a la cual se le pasarán como parámetros la variable contenedora del objeto asociado a la conexión contra la base de datos, el nombre del usuario en cuestión y el nombre del dispositivo de red, del cual se ha de iniciar su servidor Node.

Interacción entre los ficheros PHP y la base de datos

Para explicar cómo se establecen las conexiones a la base de datos desde los ficheros PHP contenidos en el servidor Apache, primero es preciso explicar el contenido del fichero "*configuration_properties.php*", que puede ser visualizado a continuación.

```php
<?php
  // Connection variables
  $dbhost = "localhost";    // localhost or IP
  $dbuser = "root";     // database username
  $dbpass = "<password>";   // database password
  $dbname = "db";         // database name
  $web_url = "http://10.51.1.44";     // Web URL
  $emailPHP = "cibersecudc@gmail.com";  // email used by PHPMailer
  $emailPHPpass = "<password>"; // email password
?>
```

Estas variables serán las que se deban configurar para establecer la URL de nuestra web, el correo electrónico desde donde enviara e-mails automaticamente y para permitir el acceso, desde el servidor Apache, a la base de datos.

- **$dbhost**. Nombre del host donde se encuentre localizado el servidor de base de datos. Para nuestra práctica será "*localhost*".
- **$dbuser**. Nombre del usuario de la base de datos. Por lo general usaremos *root*.
- **$dbpass**. Contraseña de la base de datos para el usuario anterior.
- **$dbname**. Nombre de la base de datos donde estarán contenidas todas las tablas necesarias para nuestra aplicación.
- **$web_url**. URL de nuestra web.
- **$emailPHP**. Dirección de correo electrónico desde donde la librería PHPMailer enviará correos electrónicos.
- **$emailPHPpass**. Contraseña del correo electrónico anterior.

Una vez entendido esto, pasaremos a hablar sobre el proceso de establecimiento de conexiones. Vemos a continuación un código sencillo de conexión y realización de una petición a la base de datos. Se proporciona este ejemplo a modo de entendimiento generalizado de cómo se realizan todas las peticiones a la base de datos desde el servidor Apache.

```php
<?php
  // Connection info. file
  include './web_config/configuration_properties.php';

  // Connection establishment
```

```
6   $conn = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);
7
8   // Check connection
9   if (!$conn) {
10     die("Connection failed: " . mysqli_connect_error());
11  }
12
13  // Query sent to database
14    $result = mysqli_query($conn, "SELECT username FROM user ORDER
      BY username ASC");
15
16    if (mysqli_num_rows($result)==0)
17        printf("<p style='margin-left: 2em; color:red'> <b>No hay
      usuarios.</b></p>");
18
19    while ($row = mysqli_fetch_array($result)) {
20        printf("<p style='margin-left: 2em; color:green'> <b>No hay
      usuarios.</b></p>",$row[0]);
21    }
22
23    mysqli_close($conn);
24 ?>
```

Es muy importante que en cada fichero PHP, antes de establecer una conexión contra la base de datos, referenciemos el fichero "*configuration_properties.php*". Para ello lo referenciaremos con un *include* tal y como podemos observar en la línea 4 del ejemplo. El primer paso, una vez realizado esto, es establecer la conexión contra la base de datos[4], lo cual podemos ver en la línea 7. Dicha función devolverá un objeto que representará la conexión contra la base de datos, el cual almacenaremos en la variable "*$conn*". A partir de momento, y en cualquier función que realice peticiones contra la base de datos, será necesario acompañarla con esta variable como parámetro. Seguidamente, será preciso verificar la correcta conexión contra esta, tal y como aparece representado desde la línea 10 a la 12. En caso de éxito en el establecimiento de la conexión ya se podrá empezar a realizar peticiones. En la línea 15 vemos un ejemplo de una operación "*SELECT*", la cual guardará los valores devueltos en la variable *$result*.

Una vez realizada la operación, es muy sencillo recuperar los valores devuelvos por la base de datos. En la línea 17 podemos ver la sentencia en cuestión. Con la función "*mysqli_num_rows*" y pasándole como atributo la variable "*$result*", obtendremos el número de elementos devueltos por la base de datos. Esto es muy útil para muchas situaciones en las que,

[4] Siempre que trabajemos contra la base de datos desde los ficheros contenidos en Apache, emplearemos la clase "*mysqli*" de PHP, la cual nos permite operar contra ella.

en caso de no devolver ningún resultado ocurra un evento, o en caso de devolver al menos uno, ocurra otro. Para nuestro ejemplo, en caso de no devolver ningún resultado, la aplicación mostrará en rojo el mensaje "*No hay usuarios.*", en caso contrario, devolverá en color verde, una lista de todos los nombres de usuario contenidos en la base de datos.

Como parte final, es preciso darnos cuenta que todas las conexiones que abramos contra la base de datos deben ser cerradas una vez realizadas las operaciones pertinentes. El cierre de esta conexión es sencillo, basta con la función "*mysqli_close*" a la que le pasaremos como parámetro la variable "*$conn*", que si recordamos contiene el objeto que representa la conexión a la base de datos.

Inicio, mantenimiento y cierre de sesiones con PHP

A la hora de gestionar las sesiones de usuario es muy importante tener en cuenta si un usuario ya ha sido autenticado o no. A continuación, podemos ver un ejemplo de código empleado para gestionar las sesiones desde el "*Index*" de la página web.

```
1  <?php
2
3  session_start();
4
5  if (isset($_SESSION['username'])){
6    if (time() - $_SESSION['start'] < 3600)
7          header('Location: inicio');
8     else {
9      session_unset($_SESSION['username']);
10     session_destroy;
11    }
12 }
13
14 ?>
```

Como primer paso, con la función "*session_start()*" iniciaremos una nueva sesión, o en caso de ya existir una, reanudaremos la existente. A continuación, comprobaremos si efectivamente ya existe una sesión creada o no. En caso de existir una, haremos una comprobación previa al redireccionamiento de usuario a la ventana de "*Inicio*". Como norma general se establece una duración para cada sesión de 1 hora y, en caso de superar este tiempo, se entenderá como que la sesión ha caducado y el usuario deberá iniciar sesión de nuevo.

Siguiendo este ejemplo es muy sencillo comprender como se realiza la gestión de las sesiones en todos los apartados de la aplicación.

Para el caso de acceder a los diferentes apartados del "*Panel de Administración*", existe una diferencia. Un ejemplo de código para este caso lo podemos ver a continuación.

```
<?php
session_start();
include '../web_config/configuration_properties.php';

$conn = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);
if (!$conn)
  die("Connection failed: " . mysqli_connect_error());

if (isset($_SESSION['username'])){
  if (time() - $_SESSION['start'] > 3600) {
       session_unset($_SESSION['username']);
       session_destroy();
       header("Location: ../index");
       die();
  } else {
    $username = $_SESSION['username'];
    $isAdm = mysqli_query($conn, "SELECT is_admin FROM user WHERE
    username = '$username'");
    while ($isAdmRow = mysqli_fetch_array($isAdm)) {
      if ($isAdmRow[0] == 0){
        header('Location: ../index');
        die() ;
      }
    }
  }
}else{
header('Location: ../index');
 die() ;
}
mysqli_close($conn);
?>
```

Los primeros pasos son similares a los ya explicados. La principal diferencia con el anterior, es la conexión contra la base de datos. Primero incluiremos la referencia al fichero contenedor de las variables de acceso a la base de datos. Acto seguido, se establecerá conexión contra esta, utilizando las mencionadas variables.

Una vez iniciada la sesión continuaremos con el siguiente paso, otra vez similar con el explicando previamente, comprobar si ya existe una sesión previa y que no haya excedido el tiempo máximo de sesión. Lo nuevo para este caso viene a continuación, en caso de que exista una sesión activa y no se haya superado el tiempo máximo de sesión, se realizará una

comprobación contra la base de datos. Se chequeará si el usuario es un administrador donde, en caso de no serlo, será redireccionado el usuario a la página de "*Index*"[5] de nuestra web. En caso de tratarse realmente de un administrador, se continuará con el acceso normal a la pestaña de inicio del panel. Finalmente, se cerrará esa conexión contra la base de datos.

Por último, es necesario explicar cómo se cerraría la sesión de usuario, en caso de que este último lo solicite voluntariamente. Este código se halla en el archivo "*logout.php*" y es el mostrado a continuación.

```
<?php

    session_start();
    session_unset($_SESSION['username']);
    session_unset($_SESSION['pop-up']);
    session_destroy();

    header('location: index');

?>
```

Este caso es realmente fácil de entender. Primero recuperamos la sesión que deseamos cerrar para a continuación, liberar las variables de sesión correspondientes[6]. Acto seguido, se destruirá la sesión y por último redireccionado el usuario al "*Index*" de la aplicación.

Inicialización de un servidor Node.js desde el servidor Apache

Este es uno de los puntos más interesantes a conocer de la aplicación. Nuestro servidor Apache es capaz de autoejecutar los servidores Node, de cada dispositivo de red, siempre que sean requeridos. ¿Cómo se consiguió esto? Revisemos el código que se muestra a continuación.

```
function initiateServer($conn, $device, $username){

  include "./web_config/configuration_properties.php";
    $devices =
    simplexml_load_file($web_url."/web_config/devices_info.xml");

  foreach($devices as $device1)
```

[5] Nótese que la pestaña de "*Index*" redireccionará automáticamente al usuario al "*Inicio*" en caso de haber una sesión existente.

[6] Con las variables de sesión se diseñó el "pop-up" de la ventana de "*Inicio*". Para cada sesión iniciada, el "pop-up" aparecerá una única vez. Una vez el usuario lo cierre, no volverá a aparecer hasta que vuelva a iniciar sesión en la aplicación web.

```
7       if ($device1->name == $device) {
8       $deviceId = $device1->idDb;
9     }
10
11  $result = mysqli_query($conn, "SELECT Id, in_use FROM device
      WHERE Id = '$deviceId' AND in_use = 'NO'");
12
13  if (!mysqli_num_rows($result)==0) {
14
15    foreach($devices as $device1)
16      if ($device1->idDb == $deviceId) {
17        $name = $device1->name;
18        $com = $device1->com;
19        $port = $device1->port;
20        $baudRate = $device1->baudRate;
21        $dataBits = $device1->dataBits;
22        $stopBits = $device1->stopBits;
23        $flowControl = $device1->flowControl;
24        $lock = $device1->lock;
25          }
26
27      $server_status = mysqli_query($conn, "SELECT Id, in_use FROM
      device WHERE Id = '$deviceId' AND server_status = 'OFF'");
28
29    echo "<img src='images/loading.gif' class='pageCover'>";
30      echo "<div class='pageCover'></div>";
31
32    $token2 = md5(time()).rand(10,9999);
33
34      $add2 = mysqli_query($conn, "UPDATE device SET token =
      '$token2' WHERE Id = '$deviceId'");
35
36    if (!$add2)
37      die('Invalid query: ' . mysql_error());
38
39    If (!mysqli_num_rows($server_status)==0) {
40
41      $rootDir = getRootDir();
42      $rootDir = $rootDir . "server_node/server.js";
43        $processId = shell_exec("DISPLAY=:0.0 xterm -hold -e 'node
      $rootDir $com $port $baudRate $dataBits $stopBits $flowControl
      $lock $deviceId $name' > /dev/null 2>&1 & echo $!");
44
45      flush();
46      ob_flush();
47      sleep(3.2);
```

```
48
49      }
50
51      flush();
52      ob_flush();
53      sleep(4.3);
54
55      $in_use = mysqli_query($conn, "UPDATE device SET in_use = 'YES'
        WHERE Id = '$deviceId'");
56
57      echo "<script type='text/javascript'>
58      window.top.location='console$deviceId?pidConsole=
59      $processId&user=$username&port=$port&token=$token2
60      &deviceName=$device';</script>";
61
62    }
63  }
64  ?>
```

Este nuevo fragmento de código se corresponde con un resumen de la función "*initiateServer*", la cuál será ejecutada cuando un usuario quiera entrar a la consola de algún dispositivo. En él se incluyen aquellas líneas de código más interesantes a explicar de acuerdo con la funcionalidad a obtener.

Primeramente, guardaremos en la variable "*$device*" el contenido del fichero "*devices_info.xml*" tal y como aparece en la línea 3. Una vez cargado, el siguiente paso será recorrer todos los elementos contenidos en el XML, a fin de recuperar aquel cuyo nombre coincida con el parámetro de entrada "*$device*". Una vez obtenido, recuperaremos su Id correspondiente de la base de datos.

Acto seguido, realizaremos una petición contra la base de datos, empleando el mencionado Id, para comprobar si la consola del dispositivo ya está en uso por otro usuario. En caso negativo, continuará la ejecución del código. El paso siguiente es recuperar todos los valores, para el dispositivo en cuestión, existentes en el archivo XML, los cuales emplearemos más adelante.

Una vez realizado todo lo anterior, se realizará una nueva petición a la base de datos, con el objetivo de comprobar si el servidor Node de ese dispositivo ya está en funcionamiento o no. El resultado de esta petición se guardará en la variable "*$server_status*". Las líneas subsecuentes (líneas 29 y 30) harán que al usuario se le muestre una pantalla de carga. En la línea 32 generaremos un "token", que será actualizado en la fila correspondiente de la base de datos en la línea 34. Este "token" será el que empleemos para otorgar seguridad a las conexiones

contra las consolas.

En la última parte del código comprobaremos el valor de la variable "*$server_status*". En caso de que el servidor Node, para este caso, no esté en funcionamiento, se ejecutará a continuación. Si observamos la línea 43 podemos ver la forma en la que esto es posible. Aprovechando la función "*shell_exec()*" de PHP, ejecutaremos una nueva ventana de Xterm, a la que pasaremos como parámetros los necesarios para ejecutar el servidor Node. De esta forma, cada servidor Node correrá en una ventana independiente a los demás. El resultado devuelvo por esta funcion ("*shell_exec*") será almacenado en la variable "*$processId*", pues contendrá el Id del proceso Xterm recién creado y que emplearemos para finalizarlo cuando sea preciso.

Por último, añadir que el código empleado de las líneas 45 a la 47, representa un mero tiempo de espera. Realizando varias pruebas de cálculo de los tiempos de inicio de los servidores Node, se obtuvo una estimación media del tiempo de inicialización de estos. Como resultado, establecemos un tiempo de espera durante el cual el usuario solo podrá visualizar la pantalla de carga, mientras que por detrás el servidor Node se estará iniciando. Una vez transcurrido este tiempo, se actualizará en la base de datos que el dispositivo de red pasa a estar "en uso" y el usuario será redirigido a la consola del mismo (líneas 56 a 59).

Envío de correos electrónicos utilizando la biblioteca PHPMailer

El envío de correos electrónicos desde PHP es relativamente sencillo empleando la biblioteca PHPMailer.

```
1  <?php
2
3  use PHPMailer\PHPMailer\PHPMailer;
4  use PHPMailer\PHPMailer\Exception;
5
6  require 'PHPMailer/Exception.php';
7  require 'PHPMailer/PHPMailer.php';
8  require 'PHPMailer/SMTP.php';
9
10 include "./web_config/configuration_properties.php";
11
12 $mail = new PHPMailer(true);
13
14 $mail->CharSet =  "utf-8";
15 $mail->SMTPDebug = 0;
16 $mail->IsSMTP();
17 $mail->Host = "smtp.gmail.com";
18 $mail->SMTPAuth = true;
19 $mail->Username = $emailPHP;
```

```
20 $mail->Password = $emailPHPpass;
21 $mail->SMTPSecure = "tls";
22 $mail->Port = 587;
23
24 $mail->setFrom($emailPHP, 'Soporte CiberSec - UDC');
25 $mail->AddAddress($username . "@udc.es");
26
27 $mail->IsHTML(true);
28 $mail->Subject  =  'Máster CiberSec - Verifica tu Cuenta';
29 $mail->Body    = '¡Enhorabuena, te has registrado con éxito!';
30
31 $mail->Send();
32
33 ?>
```

Inicialmente será preciso definir todas las librerías empleadas (líneas 3 a 8). A continuación, será preciso definir todas las variables para el correcto envío de los correos electrónicos.

- **$mail->CharSet**. Formato de codificación. Para nuestro caso usaremos "*UTF-8*".
- **$mail->SMTPDebug**. Variable para habilitar el modo verboso.
- **$mail->IsSMTP()**. Enviar usando SMTP.
- **$mail->Host**. Servidor SMTP a través del cual se enviará el e-mail, en nuestro caso usaremos el de Gmail. En caso de emplear otro, será preciso actualizar al servidor SMTP correspondiente.
- **$mail->SMTPAuth**. Habilitar la autenticación en SMTP.
- **$mail->Username**. Correo electrónico del servidor SMTP.
- **$mail->Password**. Contraseña del correo anterior.
- **$mail->SMTPSecure**. Habilitar encriptación TLS.
- **$mail->Port**. Puerto TCP en el que correrá.
- **$mail->setFrom**. Remitente del e-mail.
- **$mail->addAddress**. Destinatario del e-mail.
- **$mail->isHTML**. Definir el formato del e-mail a HTML.
- **$mail->Subject**. Asunto del e-mail.

- **$mail->Body**. Cuerpo del mensaje.

Una vez esté todo en regla, el e-mail será enviando con la función "*$mail->Send()*".

Modificaciones del fichero XML

En numerosas implementaciones de nuestra aplicación está presente la edición del fichero "*devices_info.xml*". En este nuevo apartado nos centraremos en la explicación acerca del todo el proceso que hay por detrás.

```
1  <?php
2
3  include "./web_config/configuration_properties.php";
4  $devices = simplexml_load_file("../web_config/devices_info.xml");
5
6  foreach($devices->children() as $device) {
7
8      if ($device->idDb == $id){
9          $dom = dom_import_simplexml($device);
10         $dom->parentNode->removeChild($dom);
11         $device = $devices->addChild("device");
12         $device->addChild("name",$name);
13         $device->addChild("port",$port);
14         $device->addChild("com",$com);
15         $device->addChild("description",$description);
16         $device->addChild("baudRate",$baudRate);
17         $device->addChild("dataBits",$dataBits);
18         $device->addChild("stopBits",$stopBits);
19         $device->addChild("flowControl",$flowControl);
20         $device->addChild("lock",$lock);
21         $device->addChild("idDb",$id);
22
23         $dom = new DOMDocument("1.0");
24         $dom->preserveWhiteSpace = false;
25         $dom->formatOutput = true;
26         $dom->loadXML($devices->asXML());
27         $dom->save("../webconfig/devices_info.xml");
28 }
29
30 ?>
```

En este código estamos viendo un ejemplo de edición de algún elemento miembro del fichero XML. Lo primero será cargar en la variable "*$devices*" el contenido del fichero XML, tal y como se hace en la línea 3, empleando la función de PHP "*dom_import_simplexml*". Para este

ejemplo, haremos una búsqueda por todos los elementos del XML, tomando como referencia un Id, pasado por parámetro previamente. En caso de éxito durante la búsqueda, los siguientes pasos serán sencillos.

Con el elemento resultante de la búsqueda (perteneciente a la clase "SimpleXML"), aplicando sobre él la función "*dom_import_simplexml*", lo convertirá a un nodo "DOMElement". Para el caso de editar un elemento optamos por diseñarlo, como normal general, de la siguiente manera: primero buscar el elemento a editar, hacer una copia de este, eliminarlo del XML, modificar con los nuevos valores y por último insertar el elemento editado. Atendiendo a estos pasos, el que viene a continuación es eliminar dicho elemento, lo cual se hará en la línea 9. Es preciso indicar que para este caso no se hizo copia, sino que los nuevos valores para cada atributo del elemento ya estarán guardados dentro de sus respectivas variables. El paso a continuación será crear un nuevo elemento XML, lo cual está implementado desde las líneas 10 a la 20. En la línea 10 crearemos el que será el futuro nuevo elemento a insertar en el fichero XML. En las líneas posteriores, utilizando la función "*addChild()*", agregaremos a este nuevo elemento cada una de las variables correspondientes.

El último paso será insertar el elemento modificado en el XML. Este paso estará implementado desde las líneas 22 a la 26. La función "*$dom->save()*" será la que inserte el elemento definitivamente.

Reglas de redireccionamiento del servidor Apache

Es muy importante tener en cuenta que tanto el servidor Apache, como cada uno de los servidores Node, correrán en puertos independientes. Es por esta misma razón por la que es preciso realizar una redirección de puertos al momento de intentar entrar a la consola de algún dispositivo. En el código a continuación mostrado, vemos un ejemplo de cómo se realizaría la redirección de puertos.

```
1  Options +FollowSymLinks -MultiViews
2
3  RewriteEngine on
4
5  RewriteCond %{REQUEST_URI}  ^/console(.*)         [NC]
6  RewriteCond %{QUERY_STRING} ^pidConsole=(.*)       [NC]
7  RewriteCond %{QUERY_STRING} ^user=(.*)       [NC]
8  RewriteCond %{QUERY_STRING} ^port=(.*)       [NC]
9  RewriteCond %{QUERY_STRING} ^token=(.*)       [NC]
10 RewriteCond %{QUERY_STRING} ^deviceName=(.*)       [NC]
11 RewriteCond %{QUERY_STRING} transport=websocket    [NC]
12 RewriteRule /(.*)           ws://10.51.1.44:$3/$1/$2/$3/$4/$5 [P,L]
13
```

```
14 RewriteCond %{REQUEST_URI}  ^/terminal(.*)         [NC]
15 RewriteCond %{QUERY_STRING} transport=websocket    [NC]
16 RewriteCond %{QUERY_STRING} ^port=(.*)        [NC]
17 RewriteCond %{QUERY_STRING} ^username=(.*)        [NC]
18 RewriteCond %{QUERY_STRING} ^token=(.*)        [NC]
19 RewriteRule /(.*)    ws://10.51.1.44:$1/console.html/$1/$2/$3 [P,L]
```

```
1  ProxyPreserveHost on
2
3 ProxyPass /console1 http://10.51.1.44:8080/
4 ProxyPassReverse /console1 http://10.51.1.44:8080/
5
6 ProxyPass /terminal1 http://10.51.1.44:8080/console.html
7 ProxyPassReverse /terminal1 http://10.51.1.44:8080/console.html
```

En el primer fragmento observamos el contenido del archivo "*.htaccess*". En este archivo es preciso destacar la implementación desde las líneas 18 a la 32. En este apartado se trata la forma en la que se trabaja con los atributos incluídos en las diferentes URLs con las que trabajaremos. Esto resulta especialmente útil para intercambiar información entre el servidor Apache y los servidores Node.js.

En el segundo fragmento observamos una parte del archivo "*000-default.conf*" ubicado en el directorio "*/etc/apache2/sites-available*". En este archivo es donde marcaremos la forma en la que queremos redireccionar los puertos. En el ejemplo mostrado podemos ver el caso para el dispositivo de red Id = 1 en la base de datos. Al servidor Node de este dispositivo le corresponde el puerto 8080. Las líneas 3 y 4 corresponden a la redirección de la URL a la que nos redirige el Apache cuando deseemos entrar a alguna consola. Las líneas 6 y 7 corresponden a una redirección de URL realizada por el servidor Node. El método de funcionamiento de todas es similar. Centrémonos en el caso para la URL "*/console1*".[7] Cuando se detecte una petición GET hacia la URL en cuestión, esta será redirigida a la del servidor Node respectivo que, no podemos olvidar, estará corriendo en un puerto alternativo.

Visualización de los ficheros de log

En el presente y último subapartado abordaremos el tema de la visualización de los logs de los diferentes dispositivos de red. Estos ficheros no son más que simples documentos de texto, con lo resultará bastante sencillo extraer su contenido y mostrarlo en la aplicación web. Atendamos a los ejemplos de código siguientes.

```
1 <?php
2 $dir = opendir("../server_node/logs/");
```

[7] Mencionar que este "1", al final de la URL, indica el Id, en la base de datos, del dispositivo de red en cuestión.

```
    while ($elemento = readdir($dir)){
        if( $elemento != "." && $elemento != ".."){
            echo "<form action='' method='post'>
                <p style='margin-left: 2em'> • <input type='submit'
                name='logdevice' value='$elemento' /></p>";
        }
    }
?>
```

En este ejemplo podemos apreciar como recuperaríamos todos los archivos de log existentes, para su posterior exhibición en la pestaña de "*Logs*" de la aplicación. Es bastante fácil. Primero abriremos el directorio contenedor de los logs y guardaremos su contenido en la variable "*$dir*". Una vez abierto, la recorreremos[8] imprimiendo los nombres de cada uno de los archivos de log existentes, para que después el usuario pueda elegir libremente aquel del que desee ver su log.

```
<?php

$file = fopen("../server_node/logs/" . $logdevice . ".txt", "r");

while(!feof($file)) {
    $traer = fgets($file);
    echo nl2br($traer);
}

fclose($file);

?>
```

En este segundo ejemplo vemos como sería el proceso de imprimir por pantalla todos los registros de un archivo de log. Con la función "*fopen()*" abriremos el fichero y guardaremos el contenido del log en cuestión, dentro de la variable "*$file*". Seguidamente, con un bucle, se recorrerá todo su contenido y se irá mostrando por pantalla. A la hora de escribir los logs dentro de cada archivo, se fueron separando con la cadena "*\r\n*", la cual simboliza un salto de línea. Con la función "*nl2br()*", convertimos esta cadena en saltos de línea reales. Finalmente, cerraremos el fichero que abrimos inicialmente empleando la función "*fclose()*".

6.3.2 Directorio raíz de Node.js

Cuando un usuario accede a una consola de un dispositivo de red, lo que está haciendo la web por detrás es redirigir al usuario al "*Index*" de la consola, ubicado en el directorio Node

[8] Los directorios "padre" y el propio directorio en sí, los cuales se ignorarán, pues no tendría sentido imprimirlos.

(directorio "*server_node*"). En este mismo directorio también están ubicados todos los archivos que permiten la correcta ejecución y funcionamiento de los servidores Node.

Bien es cierto que no debería estar ubicado dentro del directorio raíz del servidor Apache. No obstante, se decidió implementar de esta manera debido a la estrecha relación que guarda con este.

Los archivos y directorios que se encuentran en la carpeta raíz de Node.js son los siguientes:

- **client.js**. Código JavaScript relativo a la página de consola.
- **console.html**. Contiene todo el código HTML relativo a la página de las consolas de los dispositivos de red.
- **package.json**. Contiene los metadatos referidos a nuestro proyecto y ayuda para administrar los módulos de Node.js dependientes del mismo.
- **package-lock.json**. Archivo generado automáticamente. Se crea al realizar cualquier operación con npm, describiendo el árbol exacto que se generó para cada una de estas operaciones.
- **server.js**. El servidor Node. En él se encuentran todas las sentencias necesarias para su correcto funcionamiento. Codificado en lenguaje JavaScript.
- **Directorio css**. Contiene las hojas de estilos en cascada necesarias para darle su diseño a la página de las consolas.
- **Directorio logs**. Contiene los logs de las consolas de cada uno de los diferentes dispositivos de red.
- **Directorio node_modules**. Almacenará, para cada paquete instalado, una carpeta con el nombre de este junto a sus ficheros y dependencias necesarias. Directorio creado automáticamente al instalar paquetes o dependencias utilizando npm.

Conexión y desconexión de usuarios e inicialización del servidor web

El proceso de establecimiento de nuevas conexiones al servidor de una consola es sencillo de entender. Primero, si recordamos lo explicado anteriormente, cuando un usuario quiere acceder a la consola de un dispositivo de red, la web por detrás lo redirigirá a la URL correspondiente (atendiendo a las diferentes directivas de Apache). Inicialmente será redirigido a la URL "*/consoleX*", siendo "X" el Id en la base de datos para el dispositivo en cuestión, acompañada de un conjunto de parámetros. Esta nueva URL será la del fichero "*console.html*"

del servidor Node. Cuando el servidor detecte esta nueva conexión contra la web, primero guardará los valores incluídos en la URL en sus variables correspondientes y, por temas de seguridad, redirigirá al usuario a la URL "*/terminalX*", la cual irá acompañada de parámetros con escasa utilidad para el usuario.

```
const express = require("express");
const app = express();

app.get("/", function (req,res) {
  if (pidCons == -1)
    pidCons = req.query.pidConsole;
  username = req.query.user;
  if (token == "")
    token = req.query.token;
  if (deviceId == "")
    deviceId = req.query.deviceId;
  if (deviceName == "")
    deviceName = req.query.deviceName;
  res.redirect("/terminal" + process.argv[2] + "?port=" +
    process.argv[3] + "&username=" + username + "&token=" +
    req.query.token);
});
```

En el ejemplo anterior representa el fragmento de código encargado de redireccionar al usuario a la nueva URL y, a su vez, guardar las variables proporcionadas como parámetros de la URL inicial. Nótese que para hacer esto posible es preciso utilizar "*express*", el cual es un marco de aplicación web de back-end para Node.js, diseñado para crear aplicaciones web.

Para el proceso de detección de nuevas conexiones (y de la misma manera, desconexiones) de usuario, emplearemos la biblioteca de JavaScript conocida como "**Socket.IO**". Esta permite la comunicación bidireccional en tiempo real entre clientes y servidores web. Tiene dos partes: una biblioteca del lado del cliente que se ejecuta en el navegador (archivo "*client.js*") y una biblioteca del lado del servidor para Node.js (archivo "*server.js*"). A continuación, se proporciona un ejemplo claro de cómo sería un posible caso de detección de conexiones o desconexiones. Así mismo, dicho ejemplo también muestra la inicialización del servidor, a partir de la cuál, el servidor comenzará a escuchar peticiones al puerto preconfigurado.

```
const server = require("http").Server(app);
const io = require("socket.io")(server);

io.on("connection", (socket) => {
  ...  // Code A
```

```
6
7   socket.on("disconnect", () => {
8     ...  // Code B
9   });
10 });
11 server.listen(process.argv[3], function(){
12   ...  // Code C
13 });
```

Primero de todo nos centraremos en la inicialización del servidor. Para ello será preciso definir la constante "*server*" requiriendo la biblioteca "*http*'", que nos permitirá definir un servidor HTTP. Con esta nueva constante, tal y como vemos en la función de la línea 11, podremos poner al servidor a escuchar en un puerto determinado (Para este ejemplo "*process.argv[3]*" representa el puerto en el que escuchará el servidor, y será pasamado como parámetro de inicialización del servidor Node). En lo relativo a la gestión de las conexiones se deberá definir la constante "*io*", la cual requerirá la biblioteca "*socket.io*", que nos permitirá trabajar con Sockets. En la línea 4 vemos como se captura una conexión. El servidor recibirá un socket con el mensaje "*connection*" a partir del cual se podrá realizar todo el "Codigo A". En caso de que este mismo usuario se desconecté, se enviará al servidor un nuevo socket, esta vez con el mensaje "disconnect", y es cuando se podrá ejecutar el Código B. Mencionemos que tanto el código A, como el B, como el C, son implementaciones a gusto del usuario. Estas implementaciones podrían ser por ejemplo, actualizaciones de la base de datos, mostrar algún mensaje por consola, etc.

Es necesario mencionar que cada servidor Node solo permitirá una única conexión simultánea por servidor. Para conseguir esto empleamos manejamos diversos comodines como puede ser el caso de una variable que almacena el número de usuarios conectados o, para los usuarios que estén en la ventana de "*Inicio*" de la aplicación web, el atributo "*in_use*" de la base de datos, el cuál informará si la consola está siendo usada o no (este valor se irá actualizando cuando el servidor Node detecte conexiones o desconexiones).

Gestión de eventos empleado web sockets

Es realmente importante explicar la relación guardada entre los archivos "*console.html*", "*server.js*" y "*client.js*". En la figura 6.8 (página 61) podemos ver una representación de esta.

Para lograr esta conducta empleamos el módulo "*Socket.io*", que nos permite establecer comunicación vía sockets entre las partes cliente y servidor. Para nuestro proyecto, los archivos "*console.html*" y "*client.js*" constituirán la parte cliente, mientras que el fichero "*server.js*" será la parte servidora. El proceso de comunicación entre ambas partes es sencillo de entender. El primer paso sería poner a escuchar la parte servidora a correr contra un puerto específico,

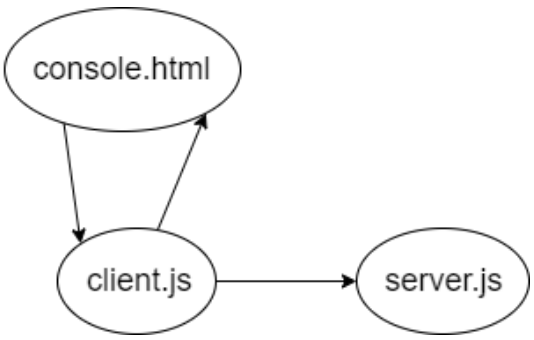


Figura 6.8: Relación entre los ficheros de Node.js.

tal y como se explicó en el subapartado previo. Lo siguiente será configurar la parte cliente. En el archivo "*console.html*" será preciso realizar las debidas importaciones de los archivos "*socket.io.js*" y "*client.js*". El primero será para poder comunicarnos con el servidor, el segundo será el que actuará de intermediando entre cliente y servidor. Tendremos que preparar el archivo "*client.js*" para poder también trabajar contra el mismo puerto que el servidor. Para afianzar los conocimientos trasladados se proporcionan a continuación ejemplos de código al respecto.

```
// Fragmento del archivo console.html

  <script> var web_url = "http://10.51.1.44"; </script>

  <script id="socket"></script>
  <script id="client"></script>

  ... // Code

  document.getElementById("client").src = url + ":" + port +
    "/client.js";
  document.getElementById("socket").src = url + ":" + port +
    "/socket.io/socket.io.js";

  var socket = io(web_url + ':' + port + '/');

  <script>

      function setJsPort() {
              stablishPortConnection(port, username, web_url);
              getName();
            }

   </script>
```

```
23
24    ... // Code
25
26    <body background="/images/background.jpg" onload="setJsPort();">
27
28    ... // Code
```

```
1  // Fragmento del archivo client.js
2  function stablishPortConnection(port, username, web_url) {
3
4      ... // Code
5
6      socket = io.connect(web_url + ":" + port, { "forceNew": true});
7
8      ... // Code
9  }
```

Los fragmentos de código previos, son representaciones reales sacadas de nuestro proyecto. Para el primer caso, vemos como se hacen referencia a los archivos "*client.js*" y "*socket.io.js*", como bien se explicó con autoridad y como se comienza a escuchar en el mismo puerto que el servidor (línea 11). En la línea 13 podemos apreciar como se establece la conexión contra la URL y el puerto en cuestión. Ahora bien, en el momento en que finaliza la carga del cuerpo de la web de la consola, se hará una llamada a la función "*setJsPort()*", la cual a su vez, hará una llamada a la función "*stablishPortConnection()*" contenida en el archivo "*client.js*", que notificará al mencionado fichero de que puede establecer conexión contra la URL y puerto susodichos.

En el segundo fragmento vemos una parte de la implementación del fichero "*client.js*". Aquí vemos la función que será llamada, llegado su momento, desde el "*console.html*". En esta función, concretamente en la línea 7, establecemos que este fichero también trabaje contra el mismo puerto que el servidor. Llegados a este punto ya tenemos los 3 archivos interconectados. A continuación, se mostrará un ejemplo de cómo sería un posible caso de comunicación entre ellas. Para entrar en contexto, nos ubicamos en el escenario de que el dispositivo de red se encuentre enviando sentencias de caracteres, las cuales serán captadas por el servidor Node.

```
1  // Fragmento del archivo server.js
2      ... // Code
3
4      io.sockets.emit("messages", routerMessage);
5
6      ... // Code
```

```
1 // Fragmento del archivo client.js
2     ... // Code
3
4     socket.on("messages",function(data){
5       render(data);
6     });
```

En el primero de los dos fragmentos se nos presenta la situación en la que el servidor Node ya captó la secuencia de caracteres enviada por el dispositivo y la almacenó en la variable "*routerMessage*". Ahora, para enviarla a la web y poder mostrarla al usuario, primero ha de enviarsela al fichero "*client.js*" de la forma que se muestra en la línea 5. El fichero "*client.js*" captará el mensaje enviado por el primer archivo de la manera reflejada en la línea 5. El paso a continuación será pasar el antedicho mensaje como parámetro a una función ("*render()*") que lo imprimirá por pantalla al usuario final.

Conectividad contra la base de datos

Para el establecimiento de conexiones contra la base de datos precisaremos del uso del módulo "*mysql*". Veamos un ejemplo más detallado de una situación real.

```
1  const mysql = require('mysql');
2
3  var con = mysql.createConnection({
4    host: db_host,
5    user: db_username,
6    password: db_password,
7    database: db_name
8  });
9
10 con.connect(function(err) {
11   if (err) throw err;
12   console.log("Connected to DB!");
13 });
14
15 server.listen(process.argv[3], function(){
16   var sql = "UPDATE routers SET state = 'Up' WHERE Id = '" +
       deviceId +"'";
17   con.query(sql, function (err, result) {
18     if (err) throw err;
19     console.log("Router DB state updated to UP!");
20   });
21 });
```

En la línea 3, con la función "*createConnection()*" realizaremos la conexión contra la base de datos. Será preciso pasar por parámetros el host donde está alojado el servidor de base de datos, nombre de usuario para este último, su contraseña correspondiente y el propio nombre de la base de datos. Una vez establecida la conexión, ya podremos realizar peticiones contra esta. Vamos a ver un ejemplo de esto en la línea 10. Como se observa, es la misma operación de inicialización del servidor web explicada en el subapartado previo. En este caso en la línea 11, actualizaremos el valor de la base de datos, correspondiente al dispositivo de red con el que estemos trabajando, indicando que el servidor está encendido. Todas las peticiones a la base de datos se realizarán de la misma manera que esta.

Conexión al puerto de consola

Para realizar la interacción entre el servidor y la conexión en el puerto de consola, será preciso emplear el módulo "*serialport*". Estblecer la conexión es relativamente sencillo, basta con hacer uso del constructor "*SerialPort*", al que le pasarán como parámetros aquellas variables que resulten de utilidad en el establecimiento de la conexión. El resultado de esto será guardado en la variable "*SerialPort*", la cual usaremos para enviar mensajes contra el dispositivo de red o gestionar la recepción de los mismos. A continuación, se muestra un ejemplo de lo explicado.

```
1 const SerialPort = require("serial");
2
3 var port = new SerialPort(process.argv[2], {
4     baudRate: parseInt(process.argv[4]),
5     databits: parseInt(process.argv[5]),
6     stopBits: parseInt(process.argv[6]),
7     flowControl: process.argv[7],
8     lock: process.argv[8]
9 });
```

Captura y envío de mensajes al dispositivo de red

El proceso de captura y envío de mensajes estará basado en lo anteriormente explicado. Para la captura de secuencias de caracteres provenientes del dispositivo de red emplearemos el parser "*@serialport/parser-readline*", el cuál como es obvio a simple vista, pertenece al módulo "*serialport*".

```
1 const parser = port.pipe(new Readline({ delimiter: ' ' }));
2
```

```
parser.on("data", (line) => {
   ... // Code
}
```

Con las sentencias anteriores prepararemos al servidor Node para capturar todas aquellas secuencias de caracteres que envía el dispositivo. Para trabajar con ellas establecimos los espacios en blanco como delimitadores. Esto fue debido a que los dispositivos de red enviaban las sentencias de tal manera que resultaba complicado trabajar con ellas. A medida que se reciban secuencias, se irán concatenando hasta que se detecta una sentencia de salto de línea después de la cual, el mensaje será mostrado[9] al usuario en la web.

Para el caso de envío de mensajes emplearemos la función "*write*".

```
  port.write(comando);
```

Donde "*comando*" es la variable contenedora del mensaje a enviar al dispositivo de red.

Detección de conexiones fraudulentas

Como se mencionó con anterioridad, se hizo especial énfasis en el tema de la seguridad de la aplicación. En esta ocasión hablaremos sobre como gestiona Node los intentos de acceso fraudulento a sus servidores. Atendamos al código siguiente.

```
socket.on('token', ({ token, socketId, usuario }) => {

    var checkToken = "SELECT token from routers WHERE Id = '" +
    deviceId + "'" ;
    con.query(checkToken, function (err, result) {
      if (err) throw err;
      if (result[0].token != token){

        if (loggerAttempIntrusion == null)
          loggerAttempIntrusion = fs.createWriteStream(rootDir +
    "/logs/AttempIntrusionConsole.txt", {
              flags: 'a'
          });

          io.to(socketId).emit('redirect', '/forbidden');
          loggerAttempIntrusion.write("El usuario " + usuario + "
    intentó entrar en el dispositivo " +  device + ". (" +
    getCurrentDate() + "h.)\r\n");
```

[9] Las diferentes concatenaciones serán tratadas de diversas formas dentro del método parseador, a fin de obtener un comportamiento similar al de la consola original del dispositivo.

```
15            connectionRejected = true;
16        }
17    });
```

¿Cómo funciona el código anterior? Muy fácil. Si recordamos, los "token" de cada conexión a los dispositivos de red, irán incluidos en la URL. Cada vez que el servidor Node detecte una nueva conexión de algún usuario, necesitará comprobar su token, para ello mandará un mensaje (vía socket) a la parte cliente, solicitando el ya mencionado "token". Estos se lo reenviarán con otro token, el cuál capturará la parte servidora en el código mostrado de ejemplo. Ahora lo que hará será comprobar el token suministrado por la parte servidora con el contenido en la base de datos. En caso de no coincidir, se guardará registro de ello incluyendo la hora y el nombre el usuario.

Sistema de autoapagado

Esta es una de las funcionalidades más útiles implementadas. Es necesario tener un control de cómo se manejan los recursos de la Raspberry, por eso se diseñó un sistema capaz de autocerrar los servidores Node, en caso de que no se estén empleando.

```
1  const cron = require('node-cron');
2
3  cron.schedule('5 * * * *', function() {
4    var sql = "SELECT last_command_time from routers WHERE Id = '" +
       deviceId + "'" ;
5    con.query(sql, function (err, result) {
6      if (err) throw err;
7      var diff = Math.abs(parseDate(getCurrentDate()) -
       parseDate(result[0].last_command_time));
8      var minutes = Math.floor((diff/1000)/60);
9
10     if ((minutes >= 10 || userNum == 0) && serverOn) {
11       io.sockets.emit('serverOff', '');
12       shutdownServer();
13     }
14   });
15 });
```

Para esta implementación empleamos el módulo "*node-cron*", que nos permitirá ejecutar funciones periódicamente. En la línea 3 podemos ver la función en cuestión, la cual se ejecutará cada segundo 5 de cada minuto. Dentro de ella realizaremos comprobaciones como si el servidor no tiene ningún usuario conectado o si este lleva más de 10 minutos sin enviar

ningún comando. Para esta última condición aprovecharemos la variable de la base de datos que contiene la hora del último comando enviado.

En caso de que el usuario lleve más de 10 minutos ausente o no haya nadie conectado, la función enviará un socket con el mensaje "*serverOff*", que será captado por la parte cliente, notificando al usuario (de haberlo) en cuestión de que el servidor va a ser apago y él redirigido a la página de "*Inicio*". A su vez, se realizará la llamada a la función "*shutdownServer()*", la cual, aprovechando el PID del proceso, enviado inicialmente como parámetro de URL desde el servidor Apache, finalizará el proceso Xterm que contenga el servidor Node en cuestión, además de actualizar los correspondientes atributos de la base de datos que hagan referencia al estado del servidor.

Escritura de logs

Se decidió implementar un sistema de logs por motivos de seguridad. Es preciso guardar un registro de todo aquel comando enviado, así como sus conexiones/desconexiones, por parte de cada usuario dentro de los dispositivos de red, por si de algún uso malintencionado de estos se produjese. El método de implementación de este sistema podemos apreciarlo en el ejemplo a continuación.

```
1 const fs = require('fs');
2
3 logger = fs.createWriteStream(rootDir + "/logs/Router" + device +
      ".txt", {
4         flags: 'a'
5     });
6
7 logger.write(username + " is connected. (" + getCurrentDate() +
      "h.)\r\n");
```

Para poder establecer el mentado sistema de registros será preciso recurrir al módulo "*fs*", que nos permitirá interactuar con el sistema de ficheros de la Raspberry. Primero de todo, será preciso crear un Stream de escritura. Esto podemos apreciarlo en la línea 3, utilizado la función "*createWriteStream*", a la que pasaremos como parámetro el "PATH" del que será el fichero de log (en caso de no existir, lo creará automáticamente). El flag "a", significarán que cada vez que se inserte una cadena de texto, no sobrescribirá al contenido, sino que se concatenará a este.

Para escribir logs en el mencionado fichero, emplearemos la función "*write()*", tal y como apreciamos en la línea 7, pasando por parámetro el mensaje a introducir en el fichero. Nótese que cada mensaje finalizará con la secuencia "*\r\n*", que simboliza saltos de línea, para su

posterior exhibición en el apartado "*logs*" de la web.

6.4 Resultados finales

Aplicando todo lo anteriormente explicado se consiguió desarrollar, exitosamente, la aplicación web completa. En la figura 6.9 (página 68) podemos ver la apariencia final de la pestaña de "*Inicio*" para un usuario con el rol de administrador. El resultado final del "*Panel de Administración*" podemos apreciarlo en la figura 6.10 (página 68).

Figura 6.9: Resultado final de la pestaña de *Inicio* para los usuarios con rol de administador.

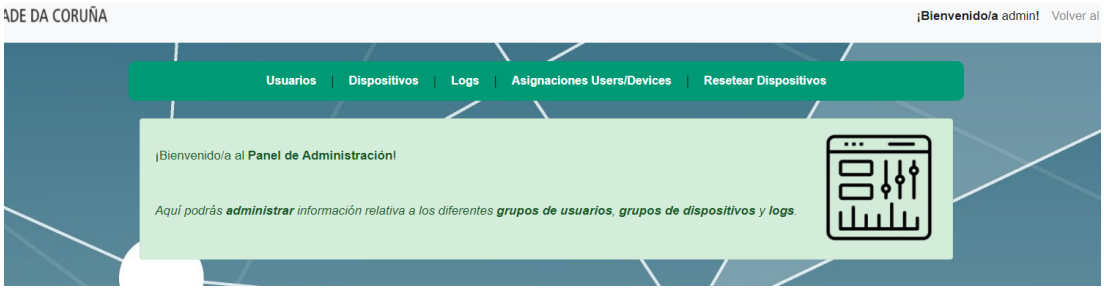


Figura 6.10: Resultado final de la pestaña *Panel de Administración.*

A continuación, ya teniendo claros todos los conceptos previos, se procederá a realizar un análisis más denso de uno de los casos de uso representados en el apartado 6.2.3, página 35, concretamente el de "*Administrador crea nuevo grupo de usuarios*", con objeto de entender más profundamente todo el proceso realizado por la aplicación.

En este escenario, el primer paso realizado por el usuario será el de autenticarse en la web. Este paso no tiene mucho misterio, pues consiste simplemente en que usuario rellene el formulario de inicio de sesión, lo envíe y el sistema corrobore los datos proporcionados con los ya existentes en la base de datos. Acto seguido, el usuario será redirigido a la pestaña de "*Inicio*". En esta nueva pestaña se cargarán todos los dispositivos registrados en la aplicación, pues se trata de un usuario con el rol de administrador. Para ello el servidor Apache hará una petición a la base de datos solicitando todos los dispositivos existentes. Con los resultados devueltos, obtendrá su información del fichero XML (aprovechando el Id de cada dispositivo) y los mostrará por pantalla. Lo siguiente que hará el usuario será dirigirse al "*Panel de Administración*". Nuevamente, antes de redirigir al usuario, el servidor Apache chequeará contra la base de datos que el usuario se trate efectivamente de un administrador. Una vez en él, el administrador se dirigirá al apartado correspondiente de creación de nuevos grupos de usuario.

En este nuevo apartado, primero se le indicará al usuario que se va el crear el grupo, con un nombre definido automáticamente por la aplicación. El nombre será de la forma "*GrupoX*", siendo X el número del grupo, el cual lo obtendrá Apache preguntándole a la base de datos cuál es el Id del último grupo insertado. En la figura 6.11 (página 69) podemos ver un ejemplo de cómo se le mostraría al usuario.

Figura 6.11: Pantalla principal del apartado *Crear Grupo de Usuarios*

.

En la siguiente ventana se le pedirá al administrador que seleccione de una lista aquellos usuarios que desee incluir en el grupo. Esta lista será generada por el servidor Apache, el cuál de nuevo, con una petición SQL solicitará a la base de datos que le devuelva aquellos usuarios sin grupo asignado. A continuación, se muestra esta sentencia en cuestión.

```
1 $result = mysqli_query($conn, "SELECT username FROM user WHERE
    user_group_id IS NULL AND username != 'admin'");
```

Una vez seleccionados, el usuario creará el grupo. La forma de actuar que tiene el sistema por detrás es realmente sencilla de entender. Primero, gracias a PHP detectará el envío del formulario. Después, se creará el grupo en la base de datos. Por útlimo, empleando un bucle,

se irá actualizando la información de cada usuario[10] seleccionado (se le asignará al nuevo grupo) y se notificará de la creación satisfactoria del grupo. El código referido a lo explicado en este párrafo puede ser visualizado a continuación.

```
if($_SERVER['REQUEST_METHOD'] == "POST" and
    isset($_POST['nameUsuarios'])) {

     $usuariosArray = explode("|", $_POST['nameUsuarios']);

     $add1 = mysqli_query($conn, "INSERT INTO
    user_group(Id,group_name) VALUES('$newGroup','Grupo$newGroup')");
     if (!$add1)
         die('Invalid query: ' . mysql_error());

     foreach($usuariosArray as $selected){
       $add = mysqli_query($conn, "UPDATE user SET user_group_id =
    '$newGroup' WHERE username = '$selected'");
       if (!$add)
         die('Invalid query: ' . mysql_error());
     }

    ...  // Code

 }
```

Se aportan también capturas de la base de datos conforme se creó el nuevo grupo de usuarios y se actualizó la información de aquellos usuarios involucrados. [11]. Podemos apreciar esto en la figura 6.12 con las imágenes 6.12a y 6.12b.

```
MariaDB [db]> select * from user_group;
+----+------------+--------------------------+
| Id | group_name | device_group_id_assigned |
+----+------------+--------------------------+
|  1 | Grupo1     |                     NULL |
+----+------------+--------------------------+
```

(a) Grupo de usuarios creado

.

```
MariaDB [db]> select username,user_group_id from user;
+----------+---------------+
| username | user_group_id |
+----------+---------------+
| admin    |          NULL |
| admin2   |             1 |
| admin3   |             1 |
+----------+---------------+
```

(b) Usuarios con grupo de usuarios actualizado.

Figura 6.12: Capturas de la base de datos.

En la figura 6.12a podemos ver la situación la tabla "*USER_GROUP*" con el nuevo grupo "*Grupo1*" recién creado. En la segunda figura (6.12b) observamos la tabla "*USER*" actualizada.

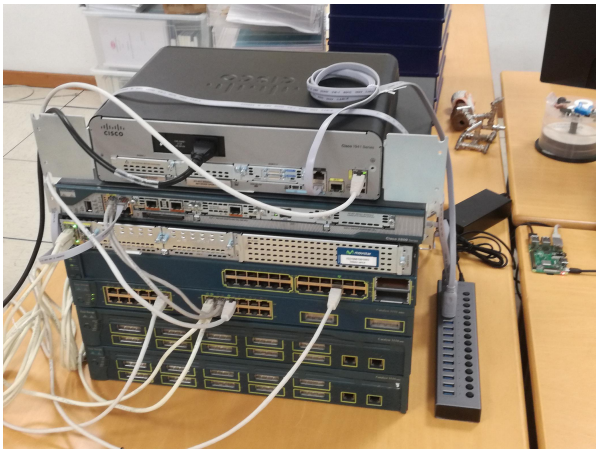
[10] Los usuarios seleccionados estarán contenidos dentro de la variable "*$_POST['nameUsuarios']", la cual será enviada por el formulario y captada por el código PHP.*

[11] Mencionar que, para lo mostrado en las capturas relativas a la base de datos, se creó el "*Grupo1*", a diferencia de la figura 6.11, en la que aparece "*Grupo5*".

En esta observamos 3 usuarios de prueba. Para el nuevo grupo creado, solo se añadieron los usuarios "*admin2*" y "*admin3*", lo cual está reflejado en la propia figura. A estos usuarios en cuestión, se les actualizó el atributo "*USER_GROUP_ID*" con el Id del nuevo grupo creado.

Por último, se mostrarán imágenes de la instalación Hardware de la aplicación en un laboratorio de pruebas. En la figura 6.13 podemos ver varias fotografías tomadas desde varios ángulos.

(a) Imagen frontal.



(b) Imagen trasera.

(c) Imagen superior (1).

(d) Imagen superior (2).

Figura 6.13: Disposición del hardware de la aplicación.

Se puede apreciar perfectamente como todos los dispositivos de red están conectados al Hub USB, mientras que este último está conectado directamente a la placa de bajo coste.

6.5 Pruebas realizadas

A modo de verificar el correcto funcionamiento de toda la aplicación, se llevaron a cabo numerosas pruebas simulando entornos reales de trabajo. Ambos directores del proyecto, en conjunto con el alumno, verificaron la correcta funcionalidad de todas las herramientas disponibles. A modo de pruebas de caja blanca se realizaron múltiples pruebas de unidad. Para este ámbito, se desarrollaron diversos casos de prueba que produzcan la ejecución de cada posible ruta de la implementación. Estas, han de garantizar que se ejecuten, al menos una vez, todos los caminos independientes de cada módulo (bucles, decisiones lógicas, estructuras internas de datos para asegurar su validez, etc.), verificando a su vez, el correcto funcionamiento de cada unidad de código. En lo relacionado al servidor Apache, se comprobaron todos los posibles caminos de implementación para cada una de las herramientas desarrolladas. Se hace un seguimiento del código fuente según se va ejecutando los casos de prueba, determinándose de manera concreta todas las instrucciones y bloques que han sido ejecutados por los casos de prueba. Un ejemplo puede ser a la hora de registrar un nuevo usuario, situación en la que se chequearán todos los posibles casos existentes, tanto de error como de éxito en el registro. Para el caso de los servidores Node se realizaron pruebas basadas en lo anteriormente mencionado. Por ejemplo, en el caso de acceder a la consola de un servidor Node, verificar paso a paso todas sus funcionalidades desarrolladas. Otro ejemplo podría ser el establecimiento de conexión por parte de un usuario, en el que se seguiría muy de cerca todo el proceso, verificando el correcto funcionamiento de todos los posibles caminos implementados.

Por otra parte, también se llevaron a cabo pruebas de caja negra llevando a cabo diversas pruebas de integración. En lo relativo al servidor Apache, se realizaron múltiples pruebas funcionales de registro, inicio de sesión y pruebas de todas las herramientas incluidas en el panel de administración, simulando situaciones de trabajo reales. En cuanto a los servidores Node, se verificó el correcto funcionamiento del mismo, así como de sus consolas (envío de comandos, recepción de comandos de dispositivo de red, etc.). Se realizaron pruebas de establecimiento simultáneo de conexiones contra servidores independientes, verificar el sistema de autoapagado de cada servidor, el intento de acceso fraudulento a los mismos o el intento de acceso a la consola de un dispositivo que ya se encuentra en uso. Así mismo, se realizaron pruebas de envío de comandos y recepción de mensajes por parte del dispositivo de red. Se verificó también la correcta comunicación entre todos los componentes de la aplicación (Apache server, Node.js, mariaDB y sus derivados).

Capítulo 7

Conclusiones y líneas futuras

Capítulo final de la memoria, donde se presentará la situación final del trabajo incluyendo conclusiones y un estudio de diversas líneas futuras de ampliación del mismo. Así mismo, se expondrán las lecciones aprendidas por el alumno, así como las capacidades adquiridas durante toda la elaboración del proyecto.

7.1 Conclusiones

El proyecto tenía como objetivo principal el desarrollo de una aplicación web para la gestión fuera de banda un laboratorio de redes de datos. Dado que se alcanzo exitosamente el objetivo inicialmente formulado, se puede afirmar que los resultados finales han sido satisfactorios. Se logró diseñar una aplicación con una interfaz sencilla, de cara al usuario final, pero que a la vez cumple con todas las expectativas. De hecho, se implementaron más funcionalidades de las inicialmente previstas, otorgando un importante incremento en las capacidades de la aplicación. Se destaca la capacidad de autoverificación de las cuentas de usuario registradas ahorrándonos todo el proceso de verificación manual. En cuanto a las consolas de los dispositivos de red funcionan a la perfección. Se realizaron múltiples pruebas, simulando situaciones de trabajo real, todas con resultados satisfactorios. A mayores, se hace un uso óptimo de los recursos disponibles en la placa de bajo coste (Raspberry) desarrollando un sistema que autoejecuta y autocierra los servidores Node, asociados a cada dispositivo de red, siempre que sea preciso. En resumen, actualmente contamos con un prototipo funcional, carente de errores, dentro de un entorno de pruebas, el cual es muy aplicable para la situación actual de confinamiento.

7.1.1 Contraste de objetivos

Como veremos a continuación, se cumplió exitosamente con todos los objetivos iniciales. A mayores, se agregaron nuevas funcionalidades que resultan en una aplicación más intere-

sante, funcional y con mayor rendimiento.

Objetivos iniciales fijados en el anteproyecto

- **Permite el acceso a las conexiones de consola desde la aplicación web.** Gracias a los servidores Node implementados se es capaz de capturar todas las sentencias de caracteres, enviadas desde el dispositivo de red a la placa de bajo coste, y mostrarlas por pantalla al usuario. De la misma manera, también somos capaces de enviar comandos en sentido contrario, desde la placa de bajo coste hacia los dispositivos de red para así lograr una comunicación completa bidireccional.

- **Controlar el acceso en base a usuarios.** Gracias al sistema de base de datos se logró diseñar una aplicación multiusuario en la que cada usuario dispondrá de sus propias credenciales de acceso. Mismamente, gracias al servidor de base de datos se logró controlar el acceso de cada usuario a los diferentes dispositivos de red mediante los grupos de usuarios y sus asociaciones a grupos de dispositivos. También se consigue controlar de esta misma manera, la restricción de un único usuario empleando cada consola de red en cada en cada momento.

- **Facilitar la creación de grupos de usuarios y dispositivos.** Con la creación del panel de administración y sus correspondientes herramientas y haciendo uso de la base de datos, se logró implementar un sistema de creación de grupos de usuarios y dispositivos realmente sencillo de manejar, pero verdaderamente efectivo.

Objetivos adicionales

- **Gestión de los recursos de la placa de bajo coste.** El manejo de los recursos en la placa de bajo coste es el óptimo en nuestra aplicación. Primero de todo se decidió implementar un código con poco peso, para no afectar en gran medida al rendimiento de la misma. Lo que más recursos consume sin duda alguna son los servidores Node. Para ello se decidió un sistema capaz de ejecutarlos y cerrarlos automáticamente siempre que sea necesario.

- **Almacenamiento de la información.** Además del sistema de gestión de base de datos, se diseñaron alternativas para el almacenamiento de información de utilidad relacionada con la aplicación. Por una parte, tenemos los ficheros de log, los cuales irán guardando un registro de todo lo que los usuarios hagan en las consolas de los dispositivos. Como se estimó que el volumen de datos almacenado sería relativamente grande, se optó por no almacenarlos en la base de dato, sino en ficheros de texto. Por último, la información referida a los dispositivos de red como nombre, descripción, entre otros,

se almacenó en un fichero XML, a fin de facilitar a los administradores la edición de los mismos sin la necesidad de realizar múltiples consultas contra la base de datos.

- **Autoverificación de nuevos usuarios.** Para evitar la verificación manual de todos aquellos usuarios registrados se diseñó un mecanismo de autoverificación de los mismos, basado en el envío de correos electrónicos de verificación. Aprovechando el requisito inicial de que todos los correos han de ser del tipo "*@udc.es*" y que cada usuario solo puede disponer de uno de estos, se diseñó un sistema mediante el que, la propia aplicación enviará correos electrónicos de verificación automáticamente.

- **Interfaz de uso intuitiva y sencilla de usar.** El objetivo principal del proyecto es el trabajo remoto contra las consolas de los dispositivos de red. De esto la necesidad de prestar especial énfasis en el desarrollo de una interfaz web realmente sencilla y fácil manejo de cara al usuario final.

7.1.2 Lecciones aprendidas

A lo largo de todo el desarrollo del proyecto se han adquirido capacidades tales como la autoorganización. Se aprendió también a gestionar adecuadamente todo el tiempo disponible para la realización el proyecto. A su vez, se desarrolló la capacidad para trabajar con tecnologías, hasta el momento nunca empleadas por el alumno, como son los casos del lenguaje de programación PHP o el entorno en tiempo de ejecución multiplataforma Node.js.

7.2 Líneas futuras

Aun habiendo alcanzado exitosamente la totalidad de objetivos definidos en el anteproyecto, existen funcionalidades adicionales las cuales se encuentran fuera del alcance definido inicialmente, pero que deben ser tenidas en cuenta a la hora de realizar futuras optimizaciones en la aplicación desarrollada. Estas son las siguientes:

- **Administradores pueden visualizar, en tiempo real, las consolas de los dispositivos de red.** Esta funcionalidad resultaría especialmente utilidad en empresas donde el administrador sea el jefe de un equipo de trabajo y los usuarios los demás integrantes del mencionado grupo. En caso de que alguno de los integrantes tuviese algún problema con la configuración de algún dispositivo de red, el administrador podría visualizar la situación en tiempo real y así ofrecer soporte de manera más eficiente.

- **Empleo de la base de datos propio de la empresa.** Para nuestro proyecto empleamos un servidor de base de datos propio. En caso de grandes empresas, las cuáles cuenten

con su propio sistema de base de datos, en la que los empleados cuenten con sus credenciales, sería mucho más óptimo emplear el servidor ya existente, evitando así la necesidad de registro.

- **Expandir la aplicación mucho más allá de los dispositivos de red.** Bien es cierto que actualmente la aplicación solo ofrece soporte para trabajar contra las consolas de los diferentes dispositivos de red. Sería interesante ampliar esta limitación más allá, por ejemplo, poder configurar remotamente los diferentes servidores existentes en los CPD. Otra posible ampliación sería el control remoto del sistema eléctrico del CPD, aunque para esto sería preciso aportar Hardware a mayores.

Apéndices

Apéndice A

Casos de uso de la aplicación web

Según la página web de la Junta de Andalucía [15], "un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software". Cada caso de uso nos especifica como debería ser la interacción entre el usuario y el sistema, a fin de alcanzar un objetivo específico. Para nuestro proyecto, se decidió dividir los casos de uso en dos subgrupos, unos orientado a los usuarios y otro a los administradores.

A.1 Casos de uso de los usuarios

En la figura A.1 (página 79) podemos ver la representación del diagrama de casos de uso de los usuarios para nuestro proyecto. A continuación, se hace una enumeración de todos los casos de usos existentes con una referencia a su tabla correspondiente, en la que se explicará cada uno más en detalle.

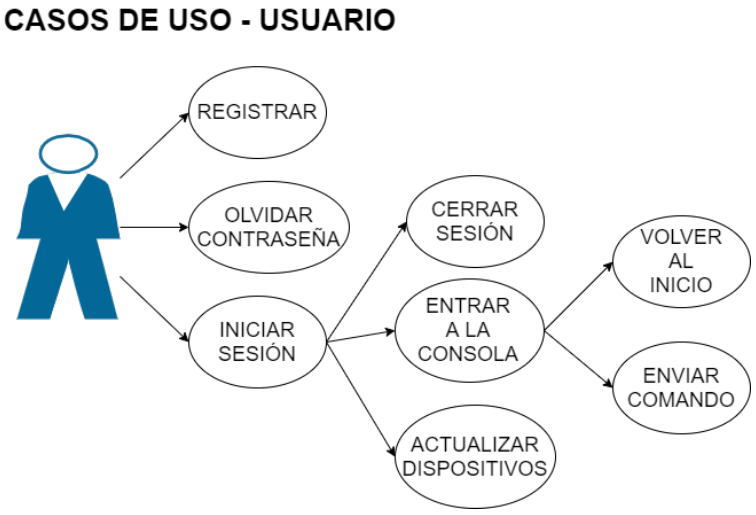


Figura A.1: Diagrama de casos de uso de usuarios.

- **CU 1 - Registro**. La información relativa a este se puede observar en la tabla A.1.

Actores	Usuario sin registrar.
Descripción	Nuevos usuarios podrán registrarse en la aplicación web.
Flujo básico	· El usuario hace click en *Regístrate ahora*. · La aplicación web muestra un formulario de registro. · El usuario completa los campos del formulario mostrado. · El sistema comprueba que todos los campos sean correctos y crea el nuevo usuario.
Flujo alternativo	El formulario de registro arroja un error. El usuario deberá volver a rellenar de nuevo todos los campos correctamente.
Precondiciones	· No puede haber mas de un usuario existente con el mismo correo electrónico. · El correo electrónico ha de ser el propio de la universidad. · La contraseña ha de tener un mínimo de 6 caracteres.
Postcondiciones	Usuario creado.

Tabla A.1: Caso de uso 1 - Registro.

- **CU 2 - Iniciar sesión**. La información relativa a este se puede observar en la tabla A.2.

Actores	Usuario registrado.
Descripción	Los usuarios registrados podrán iniciar sesión en la aplicación web.
Flujo básico	· El usuario rellenará los campos del formulario de inicio de sesión. · El usuario hace click en *Iniciar Sesión*. · El sistema verifica los datos y autentica al usuario.
Flujo alternativo	Los formularios arrojan un error. El usuario deberá volver a rellenar de nuevo todos los campos correctamente.
Precondiciones	El usuario debe estar registrado en la aplicación web.
Postcondiciones	Usuario redirigido a la ventana de *Inicio* de la aplicación.

Tabla A.2: Caso de uso 2 - Iniciar sesión.

- **CU 3 - Olvidar contraseña**. La información relativa a este se puede observar en la tabla A.3.

Actores	Usuario registrado.
Descripción	Los usuarios registrados podrán recuperar su contraseña en caso de perderla.
Flujo básico	· El usuario hace click en *¿Olvidaste tu contraseña?*. · La aplicación web muestra un formulario. · El usuario completa los campos del formulario mostrado. · El sistema comprueba que todos los campos sean correctos y envía un correo a ese e-mail. · El usuario revisará su bandeja de correo electrónico, accederá al nuevo correo recibido por parte de la web y hará click en el hipervínculo que se le mostrará. · El usuario será redirigido a la web donde se le mostrará un nuevo formulario solicitando la nueva contraseña. · Una vez completados los campos, el sistema verificará los mismos y actualizará la contraseña del usuario.
Flujo alternativo	Los formularios arrojan un error. El usuario deberá volver a rellenar de nuevo todos los campos correctamente.
Precondiciones	· El correo electrónico introducido ha de corresponderse con el de un usuario registrado. · La nueva contraseña introducida tendrá un mínimo de 6 caracteres
Postcondiciones	Contraseña actualizada.

Tabla A.3: Caso de uso 3 - Olvidar contraseña.

- **CU 4 - Cerrar sesión**. La información relativa a este se puede observar en la tabla A.4.

Actores	Usuario autenticado.
Descripción	Los usuarios autenticados podrán cerrar su sesión en la aplicación web.
Flujo básico	· El usuario hace click en *Cerrar Sesión*. · El sistema cierra su sesión.
Precondiciones	El usuario debe estar autenticado en la aplicación web.
Postcondiciones	Usuario redirigido a la ventana de *Inicio de sesión*.

Tabla A.4: Caso de uso 4 - Cerrar sesión.

- **CU 5 - Entrar a la consola**. La información relativa a este se puede observar en la tabla A.5.

Actores	Usuario autenticado.
Descripción	Los usuarios autenticados podrán acceder a las consolas de los dispositivos de red que tengan asignados.
Flujo básico	· El usuario hace click en *Entrar a Consola*. · El sistema lo redirige a la consola del dispositivo de red en cuestión.
Precondiciones	El usuario debe estar autenticado y tener asignados dispositivos de red.
Postcondiciones	Usuario redirigido a la ventana de la consola del dispositivo de red.

Tabla A.5: Caso de uso 5 - Entrar a la consola.

- **CU 6 - Actualizar dispositivos**. La información relativa a este se puede observar en la tabla A.6.

Actores	Usuario autenticado.
Descripción	Usuarios podrán actualizar la ventana de *Inicio* de la aplicación a fin de actualizar el estado de sus dispositivos de red asignados.
Flujo básico	· El usuario hace click en *Actualizar Disp.* · El sistema actualiza la ventana de *Inicio* de la aplicación.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	Ventana de *Inicio* de la aplicación actualizada.

Tabla A.6: Caso de uso 6 - Actualizar dispositivos.

- **CU 7 - Volver al inicio**. La información relativa a este se puede observar en la tabla A.7.

Actores	Usuario autenticado presente en la consola de algún dispositivo de red.
Descripción	Los usuarios autenticados que se encuentren en la ventana de consola de algún dispositivo de red, podrán volver a la ventana de *Inicio* de la aplicación.
Flujo básico	· El usuario hace click en *Volver al Inicio*. · El sistema lo redirige a la ventana de *Inicio* de la aplicación.
Precondiciones	El usuario debe estar autenticado y encontrarse dentro de alguna consola de algún dispositivo de red.
Postcondiciones	Usuario redirigido a la ventana de *Inicio* de la aplicación web.

Tabla A.7: Caso de uso 7 - Volver al inicio.

- **CU 8 - Enviar comando**. La información relativa a este se puede observar en la tabla A.8.

Actores	Usuario autenticado presente en la ventana de consola de algún dispositivo de red.
Descripción	Los usuarios autenticados que se encuentren en la ventana de consola de algún dispositivo de red, podrán enviar comandos al dispositivo de red.
Flujo básico	· El usuario escribe el comando en el campo habilitado para ello. · El usuario hace click en *Enviar*. · El sistema envía el comando al dispositivo de red.
Precondiciones	El usuario debe estar autenticado en la aplicación web y encontrarse dentro de alguna consola de algún dispositivo de red.
Postcondiciones	Sistema envía comando al dispositivo de red.

Tabla A.8: Caso de uso 8 - Enviar comando.

A.2 Casos de uso de los administradores

A la hora de hablar de los casos de uso de los *administradores*, entenderemos a estos últimos como usuarios especializados. Es decir, este tipo de usuarios contarán, además de con sus propios casos de uso, con los ya explicados anteriormente pertenecientes a los usuarios normales. En la figura A.1 (página 79) podemos ver gráficamente los casos de uso para este tipo de usuarios. A fin de aprovechar mejor el espacio disponible, solo se referenciará al caso de uso *Iniciar Sesión* del apartado previo, pues es el de que parten todos los nuevos casos.

A continuación, se hace una enumeración explicada de todos los casos de usos existentes para los administradores.

- **CU 9 - Acceso a panel de administración**. La información relativa a este se puede observar en la tabla A.9.

Actores	Administrador autenticado.
Descripción	Administradores podrán acceder al panel de administración.
Flujo básico	· El administrador hace click en *Panel de Administración*. · La web lo redirige.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	La aplicación web redirige al administrador al panel de administración.

Tabla A.9: Caso de uso 9 - Acceso a panel de administración.

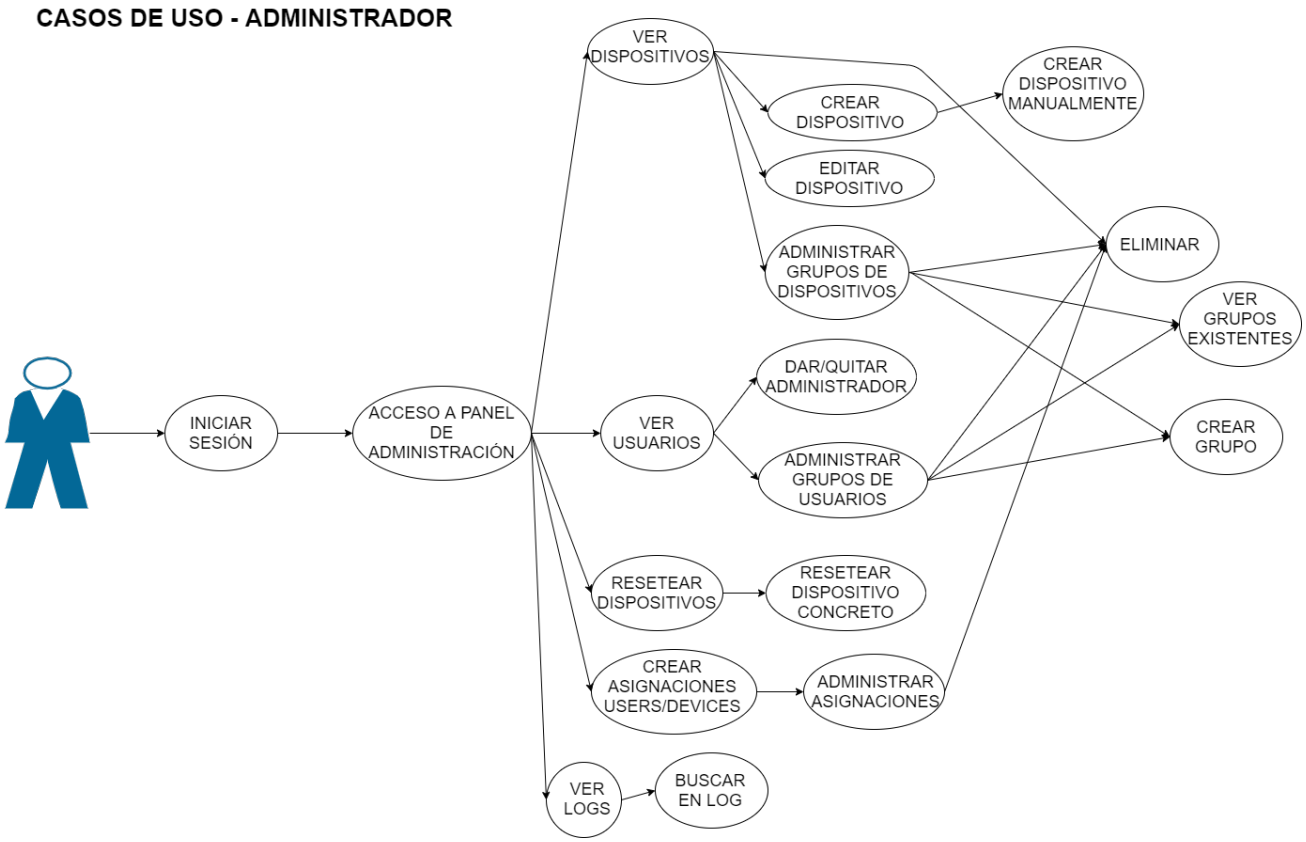


Figura A.2: Diagrama de casos de uso de administradores.

- **CU 10 - Ver Dispositivos**. La información relativa a este se puede observar en la tabla A.10.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán ver todos los dispositivos disponibles.
Flujo básico	· El administrador hace click en *Dispositivos*. · La web lo redirige.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	La aplicación web redirige al administrador al panel de administración.

Tabla A.10: Caso de uso 10 - Ver dispositivos.

- **CU 11 - Crear dispositivo**. La información relativa a este se puede observar en la tabla A.11.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán registrar nuevos dispositivos de red en la aplicación web.
Flujo básico	· El administrador hace click en *Crear Dispositivo*. · La web lo redirigirá a una nueva ventana donde habrá generado un formulario. · El administrador rellenará todos los campos del formulario. · El sistema comprobará si los campos fueron correctamente cubiertos. · El sistema creará el nuevo dispositivo.
Flujo alternativo	El formulario arroja un error. El usuario deberá volver a rellenar de nuevo todos los campos correctamente.
Precondiciones	· El administrador ha de estar autenticado. · No deben existir dos dispositivos con el mismo nombre.
Postcondiciones	Dispositivo de red creado.

Tabla A.11: Caso de uso 11 - Crear dispositivo.

- **CU 12 - Crear dispositivo manualmente**. La información relativa a este se puede observar en la tabla A.12.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán registrar nuevos dispositivos de red en la aplicación web.
Flujo básico	· El administrador hace click en el hipervínculo referido a la creación manual. · El administrador hace click en *Crear Entrada*. · El sistema creará una nueva fila en la tabla correspondiente de la base de datos. · El sistema devuelve instrucciones y un código XML que el administrador deberá completar e introducir donde se especifique en la propia página.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Dispositivo de red creado.

Tabla A.12: Caso de uso 12 - Crear dispositivo manualmente.

- **CU 13 - Editar dispositivo**. La información relativa a este se puede observar en la tabla A.13.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrá editar la información relativa a un dispositivo de red.
Flujo básico	· El administrador hace click en *¿Editar?*. · El sistema le devolverá un formulario con información referida al dispositivo de red en cuestión, la cual podrá ser editada. · El sistema comprobará que los campos del formulario sean correctos. · El sistema actualiza la información del dispositivo de red.
Flujo alternativo	El formulario devuelve un error. El usuario deberá volver a rellenar de nuevo todos los campos correctamente.
Precondiciones	· El administrador ha de estar autenticado. · No pueden existir dos dispositivos con el mismo nombre.
Postcondiciones	Dispositivo de red actualizado.

Tabla A.13: Caso de uso 13 - Editar dispositivo.

- **CU 14 - Administrar grupos de dispositivos**. La información relativa a este se puede observar en la tabla A.14.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrá administrar grupos de dispositivos de red.
Flujo básico	· El administrador hace click en *Administrar Grupos de Disp.* · El sistema lo redirige.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema redirige al administrador.

Tabla A.14: Caso de uso 14 - Administrar grupos de dispositivos.

- **CU 15 - Ver grupos existentes**. La información relativa a este se puede observar en la tabla A.15.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán visualizar grupos de usuarios o dispositivos.
Flujo básico	· El administrador hace click en *Ver Grupos Existentes.* · El sistema lo redirige.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema redirige al administrador.

Tabla A.15: Caso de uso 15 - Administrar grupos de dispositivos.

- **CU 16 - Eliminar**. La información relativa a este se puede observar en la tabla A.16.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán eliminar dispositivos y asignaciones, así como grupos de usuarios o dispositivos.
Flujo básico	· El administrador hace click en *Eliminar.* · El sistema elimina el elemeno en cuestión de su registro.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema elimina el elemento de su registro.

Tabla A.16: Caso de uso 16 - Eliminar.

- **CU 17 - Crear Grupo**. La información relativa a este se puede observar en la tabla A.17.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán crear nuevos grupos de usuarios o dispositivos.
Flujo básico	· El administrador hace click en *Crear Grupo*. · El sistema muestra una lista con dispositivos o usuarios sin grupo asignado, para que seleccione aquellos que precise añadir al nuevo grupo. · El sistema crea el nuevo grupo.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema crea el nuevo grupo de usuarios o dispositivos.

Tabla A.17: Caso de uso 17 - Crear Grupo.

- **CU 18 - Ver usuarios**. La información relativa a este se puede observar en la tabla A.18.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán ver los usuarios registrados en la aplicación web.
Flujo básico	· El administrador hace click en *Usuarios*. · El sistema devuelve una lista con los usuarios registrados.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema devuelve una lista con todos los usuarios registrados.

Tabla A.18: Caso de uso 18 - Ver usuarios.

- **CU 19 - Administrar grupos de usuarios**. La información relativa a este se puede observar en la tabla A.19.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán administrar los diferentes grupos de usuarios.
Flujo básico	· El administrador hace click en *Administrar Grupos de Users*. · La web muestra una lista con todos los grupos disponibles y el administrador selecciona uno. · La web devuelve toda la información disponible acerca de ese grupo.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema permite administrar los grupos de usuarios.

Tabla A.19: Caso de uso 19 - Administrar grupos de usuarios.

- **CU 20 - Dar/Quitar administrador**. La información relativa a este se puede observar en la tabla A.20.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán ver los usuarios registrados en la aplicación web.
Flujo básico	· En la lista de usuarios registrados el administrador podrá marcar o desmarcar la casilla *¿Administrador?*. · El sistema actualizará el rol del usuario en cuestión.
Precondiciones	· El administrador ha de estar autenticado. · No se puede quitar el rol *administrador* al usuario *admin*.
Postcondiciones	Sistema otorga o retira el rol de administrador al usuario en cuestión.

Tabla A.20: Caso de uso 20 - Dar/Quitar administrador.

- **CU 21 - Resetear dispositivos**. La información relativa a este se puede observar en la tabla A.21.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán resetear el estado de los dispositivos de red.
Flujo básico	· El administrador hace click en *Resetear Dispositivos*. · El sistema resetea todos los dispositivos.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema resetea el estado de todos los servidores de los dispositivos de red.

Tabla A.21: Caso de uso 21 - Resetear dispositivos.

- **CU 22 - Administrar asignaciones**. La información relativa a este se puede observar en la tabla A.22.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores gestionar las diferentes asignaciones existentes.
Flujo básico	· El administrador hace click en *Administrar asignaciones existentes*. · El sistema devuelve una lista con todas las asignaciones existentes.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema devuelve lista con asignaciones existentes.

Tabla A.22: Caso de uso 22 - Administrar asignaciones.

- **CU 23 - Resetear dispositivo concreto**. La información relativa a este se puede observar en la tabla A.23.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán resetear el estado del servidor de un dispositivo de red en concreto.
Flujo básico	· El administrador hace click en hipervínculo referido al reseteo manual. · El sistema devuelve una lista con todos los dispositivos de red registrados. · El administrador selecciona uno y el sistema lo resetea.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema resetea el estado del servidor de un dispositivo de red concreto.

Tabla A.23: Caso de uso 23 - Resetear dispositivo concreto.

- **CU 24 - Crear asignaciones users / devices**. La información relativa a este se puede observar en la tabla A.24.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán resetear el estado del servidor de un dispositivo de red en concreto.
Flujo básico	· El administrador hace click en *Asignaciones Users/Devices*. · El sistema devuelve una lista con todos los grupos de usuarios existentes. · El administrador elegirá uno. · El sistema devuelve una lista con todos los grupos dispositivos de red registrados. · El administrador elegirá uno de estos para asignárselo al grupo de usuarios previamente seleccionado. · El sistema crea la asignación.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema crea una nueva asignación entre un grupo de usuarios y un grupo de dispositivos.

Tabla A.24: Caso de uso 24 - Crear asignaciones users / devices.

- **CU 25 - Ver logs**. La información relativa a este se puede observar en la tabla A.25.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán revisar los logs de todos los dispositivos de red.
Flujo básico	· El administrador hace click en *Logs*. · El sistema devuelve una lista con todos los archivos de logs disponibles de cada dispositivo de red. · El administrador selecciona uno y el sistema le muestra su contenido.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema permite ver los registros de los servidores de los dispositivos de red.

Tabla A.25: Caso de uso 25 - Ver logs.

- **CU 26 - Buscar en log**. La información relativa a este se puede observar en la tabla A.26.

Actores	Administrador autenticado y dentro del panel de administración.
Descripción	Administradores podrán revisar los logs de todos los dispositivos de red.
Flujo básico	· El administrador entra al log de un dispositivo de red. · El sistema muestra, además de sus logs, un campo de búsqueda. · El administrador completa ese campo. · El sistema devuelve los registros de log que coincidan con esa búsqued5.
Precondiciones	El administrador ha de estar autenticado.
Postcondiciones	Sistema devuelve registros de log que coincidan con los criterios de búsqueda.

Tabla A.26: Caso de uso 26 - Buscar en log.

Apéndice B

Manual de Instalación

La aplicación web está diseñada para instalarse en sistemas operativos basados en Linux. El presente manual está enfocado a su instalación en entornos con Ubuntu 18.04. Para otras distribuciones será necesario adaptar algunas de las secuencias de comandos de manera apropiada.

B.1 Prerrequisitos de instalación: configuración Hardware

- Entorno con Ubuntu 18.04.
- Puertos de consola correctamente identificados.
- Conexiones eficaces a los puertos de consola.

B.2 Instalación manual

El primer paso será la instalación de aquellas dependencias indispensables para el correcto funcionamiento de la aplicación web. Es preciso que se instalen las versiones especificadas para evitar problemas de incompatibilidad.

B.2.1 Actualización del índice de paquetes

El primer paso antes de la instalación de las dependencias del software, será la actualización del índice de paquetes del servidor:

```
1 sudo apt update
```

B.2.2 Descarga de la aplicación desde el repositorio de GitHub

El primer paso será hacer una copia en local de la aplicación contenida en el repositorio de GitHub, para lo que nos valdremos del comando "*git*".

```
git clone
    https://github.com/torralba98/application-out-of-band-management
```

B.2.3 PHP

Será preciso instalar la versión "*7.4 (cli)*". Para ello ejecutaremos en un terminal la siguiente secuencia de comandos de instalación:

Descargaremos y almacenaremos el repositorio PPA. En concreto añadiremos el repositorio "*ppa:ondrej/php*", que contiene los más recientes paquetes de compilación de PHP.

```
sudo apt-get update
sudo apt -y install software-properties-common
sudo add-apt-repository ppa:ondrej/php
sudo apt-get update
```

Finalmente, instalaremos PHP y las dependencia que permitirá a la aplicación trabajar contra ficheros XML y contra la base de datos.

```
sudo apt -y install php7.4 php7.4-xml php7.4-mysql
```

Una vez instalado necesitaremos habilitar la extensión "*mysqli*", que nos permitirá trabajar contra gestores de bases de datos desde los archivos PHP. Para ello tendremos que abrir un terminal y ubicarnos en el directorio "*/etc/php/7.4/apache2*". Una vez en él, deberemos editar, con privilegios de administrador, el archivo "*php.ini*". Dentro de este deberemos encontrar la línea "*;extension=mysqli*" y descomentarla (quitando ";").

B.2.4 Node.js

Será preciso instalar la versión "*v14.x*". Para ello ejecutaremos en un terminal el siguiente comando de instalación:

```
sudo curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E
    bash -
```

```
2
3 sudo apt-get install -y nodejs
```

Una vez instalado, iniciaremos con la configuración de los futuros servidores Node. En un terminal nos ubicaremos en el directorio "*server_node*" y ejecutaremos el siguiente comando (deberemos tener instalado "*npm*"):

```
1 npm init
```

Se nos harán una serie de preguntas y pulsaremos en todas "ENTER", pues estos datos se han de modificar a continuación. En el mismo directorio se creará el fichero "*package.json*", lo abriremos y lo editaremos para indicar que el punto de entrada a nuestra App es el fichero "*server.js*". Para ello modificaremos la siguiente línea y la dejaremos tal que así: " *"main": "server.js"* ".

Ahora instalaremos todas aquellas dependencias necesarias con la siguiente secuencia de comandos:

```
1 npm install express@4.17.1 --save
2 npm install mysql@2.18.1 --save
3 npm install node-cron@2.0.3 --save
4 npm install serialport@9.0.4 --save
5 npm install socket.io@2.3.0 --save
```

B.2.5 MariaDB

Para ello ejecutaremos en un terminal el siguiente comando de instalación:

```
1 sudo apt install -y mariadb-server
```

Configuración de MariaDB

El comando a continuación, se empleará para cambiar algunas de las configuraciones predeterminadas carentes de seguridad. Se usarán para el bloqueo de conexiones root remotas, entre otros.

```
1 sudo mysql_secure_installation
```

Una vez introducido en un terminal, se visualizarán una serie de solicitudes, gracias a las que podremos actualizar las opciones de seguridad de MariaDB. Primero se nos solicitará introducir la contraseña *root* de la base de datos actual, al no haber configurado ninguna bastará con que pulsemos *ENTER* (valor por defecto "*none*").

A continuación, se preguntará si se desea configurar una contraseña *root* de la base de datos. Deberemos escribir "N" y pulsar "ENTER". Esto es para evitar que una actualización de paquetes dañe el sistema de base de datos eliminando el acceso a la cuenta administrativa, debido a que la cuenta *root* de MariaDB está vinculada al mantenimiento del sistema automatizado. Más tarde se explicará cómo realizar este cambio de contraseña.

Después escribiremos "Y" y luego "ENTER" para aceptar los valores predeterminados de todas las preguntas siguientes. Así eliminaremos todos aquellos usuarios anónimos y la base de datos de prueba y serán deshabilitadas las credenciales de inicio de sesión remoto de *root*.

Con motivo de que el servidor utiliza la cuenta "*root*" para tareas como rotación de registros y el inicio y detención del servidor, se recomienda no cambiar los detalles de autenticación root. Por ello, en su lugar crearemos una nueva cuenta llamada "*admin*" con los mismos privilegios que la cuenta root. Para ello haremos lo siguiente:

```
sudo mysql
```

Primero abriremos la instrucción de MariaDB desde el terminal, y a continuación creamos el nuevo usuario y le asignaremos los privilegios de root.

```
MariaDB [(none]> GRANT ALL ON *.* TO 'admin'@'localhost' IDENTIFIED
    BY '<password>' WITH GRANT OPTION;
```

En la sentencia anterior deberemos cambiar "*<password>*" por la que será la nueva contraseña de autenticación. Por último, vaciaremos los privilegios para asegurar que se guarden y cerraremos el shell de MariaDB.

```
MariaDB [(none]> FLUSH PRIVILEGES;
MariaDB [(none]> exit;
```

Ahora nuestro sistema de gestión de base de datos estará correctamente configurado y funcional. A continuación, iniciaremos sesión con el nuevo usuario "*admin*".

```
sudo mysql -u admin -p
```

Introduciremos la contraseña definida para el usuario "*admin*" y ya podremos empezar a trabajar contra el gestor de base de datos.

Instalación de la base de datos de la aplicación

En el repositorio de Git se incluye un script de creación de la base de datos de la aplicación. Este script creará automáticamente la base de datos con sus tablas correspondientes. Finalmente creará el usuario "*admin*". Primero de todo será preciso modificar este script de creación y actualizar la contraseña y e-mail por defecto del administrador. No se deberá modificar el nombre "*admin*". Una vez listo, deberemos abrir un nuevo terminal y ubicarnos en el mismo directorio que el script. Iniciaremos sesión contra el gestor de base de datos y escribiremos lo siguiente:

```
MariaDB [(none]> source script.sql;
```

Ahora ya tendríamos nuestro gestor de base de datos con la base de datos completamente funcional.

B.2.6 Apache Server

Para ello ejecutaremos en un terminal el siguiente comando de instalación (la dependencia incluida será para que nuestro servidor Apache pueda trabajar con ficheros PHP):

```
sudo apt-get install -y apache2 libapache2-mod-php7.4
```

Para garantizar que el servidor Apache trabaje correctamente con PHP es necesaria la siguiente secuencia de comandos:

```
sudo a2dismod mpm_event
sudo a2enmod mpm_prefork
sudo a2enmod php7.4
```

Una vez instalado, todos los archivos descargados del repositorio de Git se deberán mover a la ruta "*/var/www/html*". Para ello con un terminal nos ubicaremos en el directorio donde descargamos la aplicación del repositorio de GitHub y ejecutaremos el siguiente comando:

```
sudo mv application-out-of-band-management/* /var/www/html
```

Para que la aplicación pueda trabajar con direcciones URL alternativas a las dinámicas, generadas por la programación de nuestro sitio, será preciso habilitar el módulo "*mod_rewrite*". A continuación, será preciso reiniciar el servidor Apache. Ejecutaremos los siguientes comandos en un terminal.

```
sudo a2enmod rewrite
```

Acto seguido, con el mismo terminal nos dirigiremos al directorio "*/etc/apache2/sites-available*". Aquí editaremos el archivo "*000-default.conf*". Recordar que deberemos editarlo con privilegios de administrador. Una vez abierto el archivo, deberemos agregar las siguientes líneas justo después de "*DocumentRoot /var/www/html*":

```
<Directory "/var/www/html">
AllowOverride All
</Directory>
```

Para lograr el correcto redireccionamiento desde el servidor Apache a los servidores Node, que corren en puertos independientes, será preciso habilitar el módulo proxy de Apache para poder trabajar con un proxy inverso. Este último se trata de un tipo de servidor proxy que recupera recursos en nombre de un cliente. A modo de ejemplo, el cliente hará una petición al puerto 80 del proxy, y éste es el que realizará esta petición al servidor web, que por lo general estará en una red interna no accesible desde el cliente. Para hacer esto funcional será preciso ejecutar los siguientes comandos en un terminal:

```
sudo a2enmod proxy
sudo a2enmod proxy_http
sudo systemctl restart apache2
```

También deberemos verificar el archivo "*dir.conf*", ubicado en el directorio "*/etc/apache2/mods-available*. En la línea que comienza por "*DirectoryIndex...*" deberemos asegurarnos que el primer elemento especificado sea siempre "*index.php*" para que así se cargue correctamente nuestro index al momento de entrar a la página web. Por lo general, el primer elemento será "*index.html*", lo más fácil será intercambiar su posición con la de "*index.php*".

Por último, para que los servidores Node sean capaces de establecer conexiones contra los puertos de consola son necesarias unas pequeñas configuraciones. El acceso a los puertos de consola está restringidos al usuario "*root*" y a aquellos miembros del grupo "*dialout*". Los servidores Node y el servidor Apache tendrán como propietario al usuario "*www-data*". En-

tonces, para solventar este problema haremos que este último usuario pase a ser miembro del susodicho grupo "*dialout*". Por último, será preciso reiniciar el servidor Apache.

```
sudo usermod -a -G dialout www-data
sudo systemctl restart apache2
```

Configuración de los archivos de la aplicación

Se deberán configurar los accesos a la base de datos para que la aplicación pueda trabajar contra ella. Primero de todo será necesario editar el archivo "*configuration_parameters.php*" contenido en el directorio "*web_config*". El archivo debería quedar similar al siguiente ejemplo.

```
<?php
    // Connection variables
    $dbhost = "localhost";      // localhost or IP
    $dbuser = "admin";          // database username
    $dbpass = "<PASSWORD>";     // database password
    $dbname = "db";             // database name
    $web_url = "<URL>";         // Web URL
    $emailPHP = "<E-MAIL>";     // e-mail used by PHPMailer
    $emailPHPpass = "<E-MAIL_PASSWORD>"; // e-mail password
?>
```

Nótese que el usuario deberá sustituir "*<PASSWORD>*" por su correspondiente contraseña para el usuario "*admin*", creado previamente en el gestor MariaDB. También deberá actualizar los campos "*<URL>*", "*<E-MAIL>*" y "*<E-MAIL_PASSWORD>*". El primero hace referencia a la URL de la página web que deberá ser de la forma: "*http://URL*" o en su defecto "*https://URL*". El segundo hace referencia al correo electrónico que empleará la librería PHPMailer para el envío de e-mails y el último hace referencia a la contraseña del propio correo electrónico.[1]

De esta manera la aplicación contenida en el servidor Apache ya tiene acceso a la base de datos. Para darle el mismo privilegio a los servidores Node será preciso actualizar la siguiente información del fichero "*configuration_properties.js*" contenido en el directorio "*server_node/config*". Deberemos buscar el siguiente bloque y editarlo de una manera similar a la siguiente:

[1] En caso de usar una cuenta de Gmail será preciso habilitar la opción "*Acceso de aplicaciones poco seguras*" de la misma.

```
// DB configuration

const db_host = 'localhost';
const db_username = 'admin';
const db_password = '<PASSWORD>';
const db_name = 'db';

module.exports = db_host;
module.exports = db_username;
module.exports = db_password;
module.exports = db_name;
```

Donde la variable "*host*" se corresponde con el host del servidor de la base de datos, "*username*" con el nombre de nuestro usuario de la base de datos, "*password*" con la contraseña de este último y "*database*" con el nombre de la base de datos.

A continuación, se gestionarán los permisos asociados a los diferentes directorios y carpetas para que puedan ser modificados por la aplicación web.

El primero de ellos será el fichero XML. Abriremos un terminal y nos ubicaremos en el directorio contenedor del archivo en cuestión ("*/var/www/html/web_config*"). Deberemos ejecutar los siguientes comandos para otorgarle los permisos necesarios.

```
sudo chmod a+rwx devices_info.xml
```

Así mismo, se deberá dar los permisos correspondientes al directorio de logs ("*server_node/logs*") para que puedan ser accedidos desde la aplicación web y desde los servidores Node. Desde el directorio raíz del servidor Apache ejecutaremos:

```
sudo chmod a+rwx server_node/logs
```

En último lugar será preciso editar el fichero "*.htaccess*".

```
Options +FollowSymLinks -MultiViews

RewriteEngine on

RewriteRule ^$ /index [R=301,L]
RewriteRule ^/$ /index [R=301,L]

## hide .php extension
```

```
# To externally redirect /dir/foo.php to /dir/foo
RewriteCond %{THE_REQUEST} ^[A-Z]{3,}\s([^.]+)\.php [NC]
RewriteRule ^ %1 [R=302,L]

# To internally forward /dir/foo to /dir/foo.php
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME}.php -f
RewriteRule ^(.*?)/?$ $1.php [L]

RewriteCond %{REQUEST_URI}  ^/console(.*)         [NC]
RewriteCond %{QUERY_STRING} ^pidConsole=(.*)       [NC]
RewriteCond %{QUERY_STRING} ^user=(.*)       [NC]
RewriteCond %{QUERY_STRING} ^port=(.*)       [NC]
RewriteCond %{QUERY_STRING} ^token=(.*)       [NC]
RewriteCond %{QUERY_STRING} ^deviceName=(.*)       [NC]
RewriteCond %{QUERY_STRING} transport=websocket    [NC]
RewriteRule /(.*)           ws://<URL>:$3/$1/$2/$3/$4/$5 [P,L]

RewriteCond %{REQUEST_URI}  ^/terminal(.*)        [NC]
RewriteCond %{QUERY_STRING} transport=websocket    [NC]
RewriteCond %{QUERY_STRING} ^port=(.*)       [NC]
RewriteCond %{QUERY_STRING} ^username=(.*)       [NC]
RewriteCond %{QUERY_STRING} ^token=(.*)       [NC]
RewriteRule /(.*)   ws://<URL>:$1/console.html/$1/$2/$3 [P,L]
```

Aquí se deberá reemplazar "*<URL>*" por la URL de la aplicación web, obviando la parte de "*http://*" o en su defecto "*https://*".

B.2.7 Xterm

Para ello ejecutaremos en un terminal el siguiente comando de instalación:

```
sudo apt install -y xterm
```

Para que los terminales Xterm, futuros contenedores de los servidores Node, puedan ser ejecutados desde la aplicación web, ha de ser necesario emplear Xhost.

El programa Xhost es empleado para agregar y eliminar nombres de host a la lista permitida para hacer conexiones al servidor X. Nos permite establecer qué equipos pueden acceder al servidor gráfico de forma remota, es decir, qué máquinas cliente pueden lanzar una aplicación para ser presentada en el servidor. En consecuencia, deberemos ejecutar el siguiente comando en un terminal:

```
1 xhost +local:
```

En caso de reinicio del host, será preciso volver a ejecutar del susodicho comando.

B.3 Instalación automática mediante un script

En el propio repositorio Git se encuentra el directorio "*installation_script*", la cuál es contenedor de un script interactivo que instalará automáticamente la aplicación web. Este script está diseñado para sistemas con Ubuntu 18.04[2].

El primer paso será hacer una copia en local del repositorio Git. En un terminal ejecutaremos:

```
1 git clone
    https://github.com/torralba98/application-out-of-band-management
```

Una vez descargado, en el mismo terminal nos ubicaremos dentro del directorio "*installation_script*". Ahora ejecutaremos el script con privilegios root:

```
1 sudo installation_script.sh
```

Ahora solo resta esperar. Se nos harán algunas preguntas referidas a configuraciones de la aplicación (base de datos, URL, contraseñas...) las cuales deberemos responder. Una vez completada la instalación ya tendremos nuestra aplicación web funcional. Si en el momento de instalación se le especificó que la URL de nuestra aplicación sería "*http://localhost*", ahora bastará con abrir dicha URL en un navegador y verificar que la aplicación fue instalada correctamente.

Se recomienda no ejecutar el script más de una vez, pues podría ocasionar problemas. En caso de cometer errores en las configuraciones solicitadas por el mismo deberán ser modificadas manualmente en los archivos pertinentes.

[2] Sería preciso que se tratase de un entorno virgen (sin Apache Server, MariaDB, PHP y sus dependencias instaladas), para evitar posibles fallas durante la instalación.

Apéndice C

Manual de Usuario

C.1 Usuario

C.1.1 Inicio de sesión

En la figura C.1 (página 105) podemos ver el formulario de inicio de sesión que se le mostrará al usuario (sin autentificar) al entrar a la aplicación web. Aquí el usuario rellenará los campos existentes con sus credenciales de inicio de sesión y, una vez cubiertos, hará click en "*Iniciar Sesión*. En caso de error con los datos proporcionados por el usuario, la aplicación se lo notificará apropiadamente, en caso contrario, se redirigirá el usuario a la pestaña de "*Inicio*".

Figura C.1: Formulario de inicio de sesión.

C.1.2 Registro

En la anterior figura C.1 (página 105), vemos que se nos da la opción de registro, en caso de no contar todavía con una cuenta. Aquí se hará click en "*Regístrate ahora*" y la aplicación nos redirigirá a la pestaña de "*Registro*". En esta nueva ventana, se nos mostrará el formulario de registro, que podemos ver en la figura C.2 (página 106).

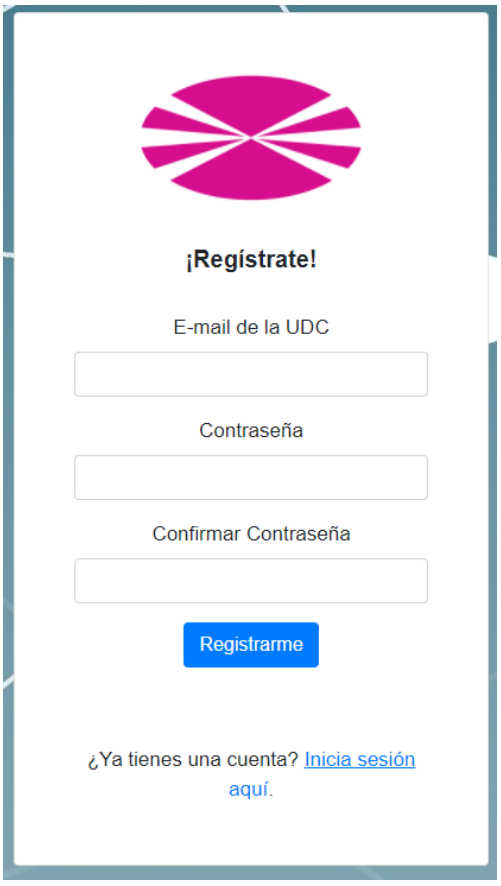


Figura C.2: Formulario de registro.

Aquí el usuario completará los campos del formulario y hará click en "*Registrarme*". En caso de error, el formulario notificará al usuario para que corrija aquellos errores detectados. Si todo está correcto, la aplicación registrará al nuevo usuario y enviará un correo electrónico de verificación de la cuenta al correo proporcionado. Podemos ver el formato de este e-mail de verificación en la figura C.3 (página 107). Este correo vendrá acompañado de un hipervínculo con el que podrá verificar su cuenta. Una vez acceda a este se le redirigirá a la aplicación web y se le notificará que su cuenta ha sido correctamente verificada. La notificación de verificación que se mostrará podemos verla en la figura C.4 (página 107).

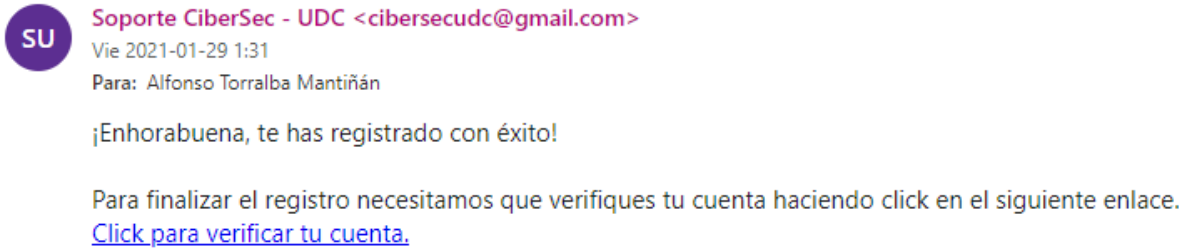


Figura C.3: Correo electrónico de verificación.

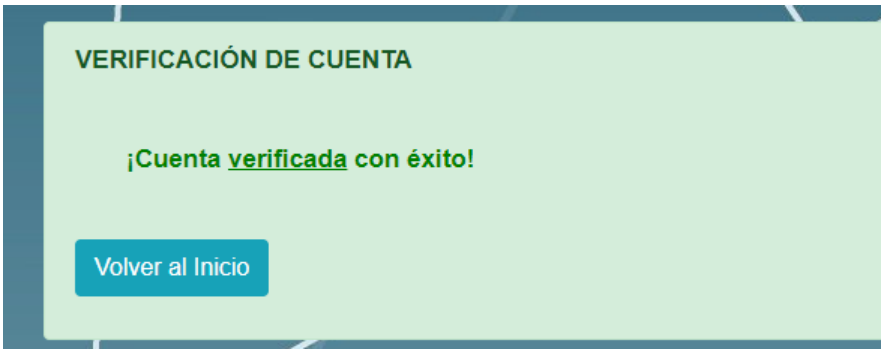


Figura C.4: Notificación cuenta verificada.

C.1.3 Olvidar contraseña

En la figura inicial C.1 (página 105) podemos ver la opción que nos permite restablecer la contraseña de inicio de sesión en caso de olvido de la misma. Para ello el usuario hará click en el botón "*¿Olvidaste tu contraseña?*" y se le redirigirá a la nueva ventana con el formulario de recuperación de contraseña, el cual podemos ver en la figura C.5 (página 108). El usuario introducirá el correo electrónico con el que se registró en la aplicación web. Una vez se envie el formulario, la aplicación le enviará un correo para verificar la identidad del solicitante. Podemos ver el formato de este correo en la figura C.6 (página 108).

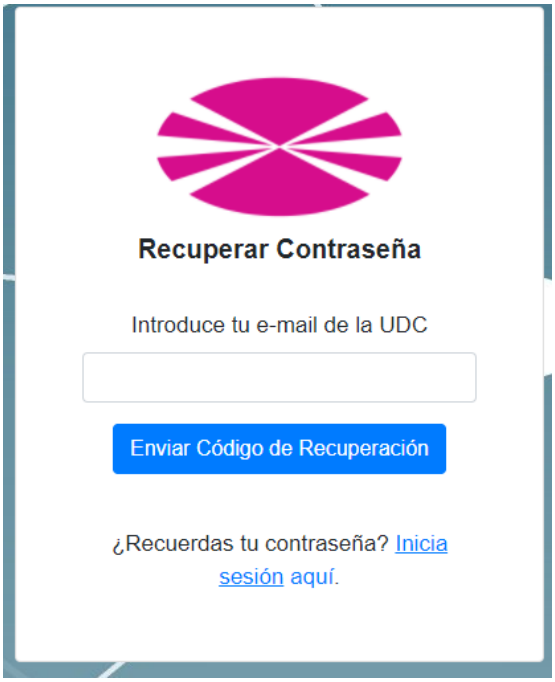


Figura C.5: Formulario inicial de recuperación de contraseña.

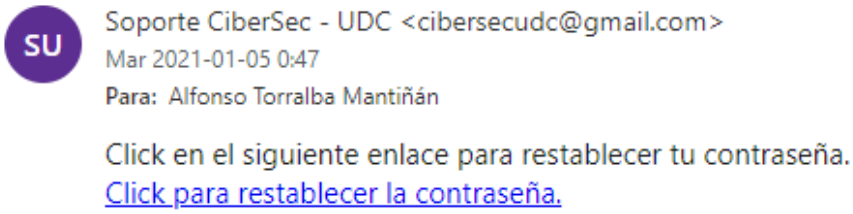


Figura C.6: Correo electrónico de recuperación de contraseña.

Este correo también vendrá acompañado de un hipervínculo, el cuál redirigirá de nuevo al usuario a la aplicación web donde podrá visualizar un nuevo formulario. En la figura C.7 (página 109) podemos ver este nuevo formulario mostrado. Aquí el usuario completará los campos con la nueva contraseña. Una vez enviado el formulario, ya estará actualizada la contraseña.

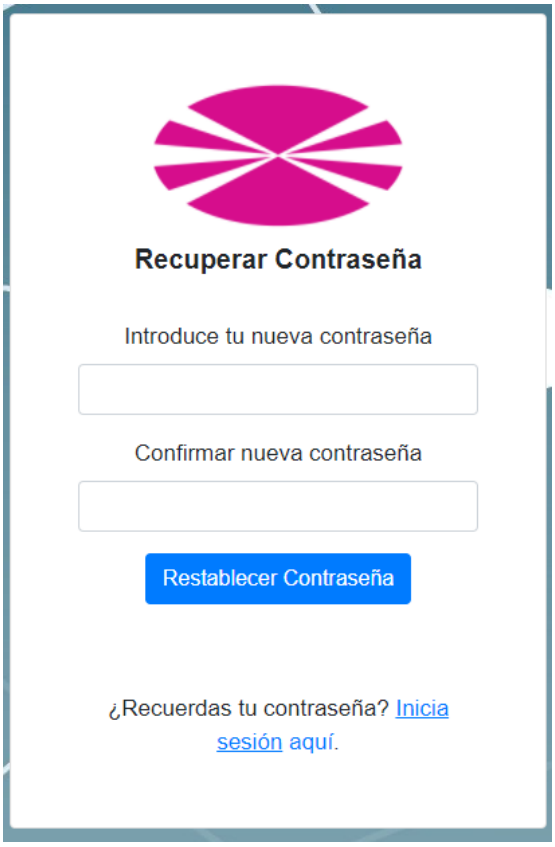


Figura C.7: Formulario final de recuperación de contraseña.

C.1.4 Cerrar sesión

Una vez el usuario inicie sesión en la aplicación web podrá cerrarla cuando lo requiera. En la parte superior derecha de todas las ventanas de la aplicación se le dará la opción al usuario de "*Cerrar Sesión*". Bastará con que el usuario haga click para que el sistema cierre su sesión. Podemos ver este botón reflejado en la figura C.8 (página 109).

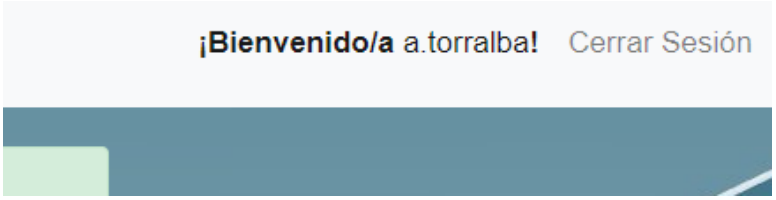


Figura C.8: Botón cierre de sesión en la aplicación web.

C.1.5 Entrar a las consolas de los dispositivos de red y manejo de estas

Al momento de iniciar sesión en la aplicación web esta nos redirige a la pestaña de "*Inicio*". En esta ventana aparecerán todos aquellos dispositivos de red que tengamos asignados. Cada dispositivo de red vendrá acompañado de su correspondiente botón de acceso a su consola. Podemos ver esto en la figura C.9 (página 110). Una vez el usuario haga click en este botón será redirigido a la consola del dispositivo de red.

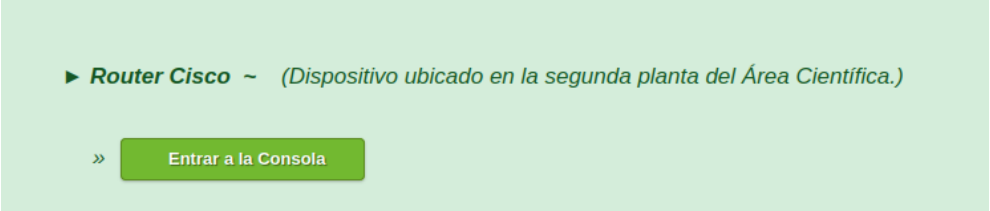


Figura C.9: Botón para entrar a la consola de los dispositivos de red.

En esta nueva pestaña podemos interactuar contra la consola del dispositivo de red en cuestión. En la parte inferior podemos ver un recuadro en el que introducir comandos para poder enviárselos al dispositivo. En la figura C.10 (página 110) podemos ver un ejemplo de interacción con un dispositivo.

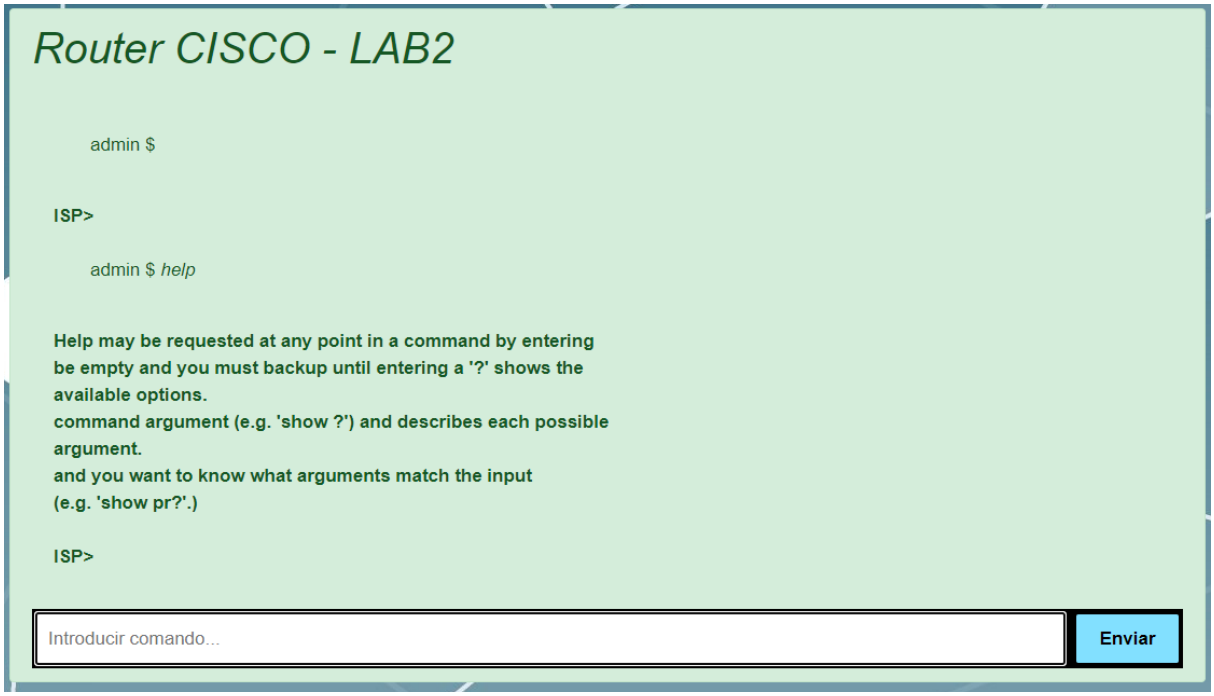


Figura C.10: Interacción contra la consola de un dispositivo de red.

C.2 Administrador

En este segundo manual se explicará el método de empleo de todas aquellas herramientas disponibles en el panel de administración de la aplicación web.

Menú del panel de administración

En todos los apartados del panel de administración siempre estará visible un menú con accesos directos a las diferentes herramientas disponibles. Podemos apreciarlo en la figura C.11 (página 111).

Figura C.11: Menú de accesos directos del panel de administración.

C.2.1 Ver usuarios registrados, gestionar administradores y administrar grupos de usuarios

En el menú de la figura C.11 (página 111), el administrador hará click en "*Usuarios*". Se le redirigirá a la pestaña correspondiente donde el administrador podrá visualizar una lista con la totalidad de los usuarios registrados en la aplicación. Cada usuario vendrá acompañado a su derecha de una opción llamada "*¿Administrador?*". Con esta opción se podrán designar nuevos administradores del sitio web. En la figura C.12 (página 111) se puede visualizar lo que se encontraría un administrador en caso de acceder a este apartado.

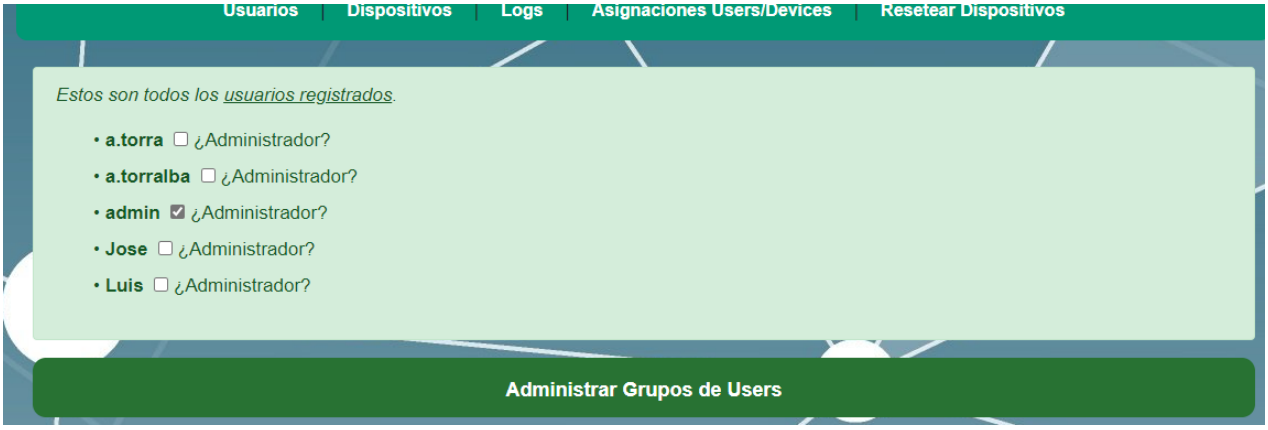


Figura C.12: Ejemplo del apartado *Usuarios* del panel de administración.

Si observamos en la parte inferior de la figura C.12 (página 111) vemos el botón "*Administrar Grupos de Users*". Este será el que redirija al administrador al apartado relativo a la

administración de los grupos de usuario. En la figura C.13 (página 112) podemos ver este nuevo apartado.

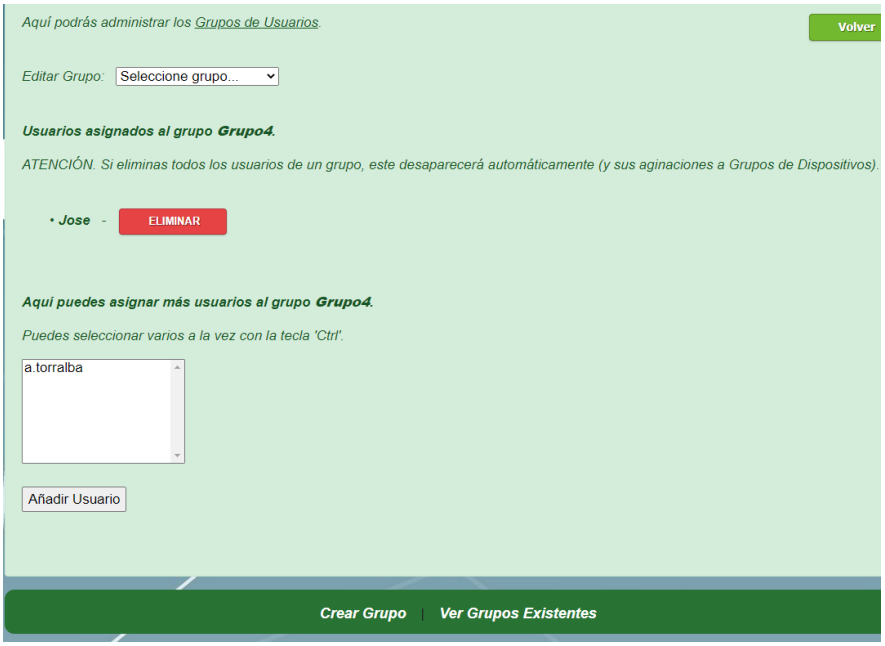


Figura C.13: Apartado administración de grupos de usuarios.

De primeras se nos ofrece la opción de seleccionar alguno de los grupos de usuarios existentes. Para cada uno se nos desplegará un menú con aquellos usuarios miembros del grupo o, en su defecto, aquellos usuarios sin grupo asignado, que podremos asignar al grupo seleccionado. Así mismo, cada usuario miembro del grupo vendrá acompañado del botón "*Eliminar*" con el que podremos expulsarlo del grupo.

En la parte inferior de la misma figura observamos un menú con las opciones "*Crear Grupo*" y "*Ver Grupos Existentes*". El primero de los dos le otorgará a la administración la opción de crear un nuevo grupo de usuarios. La primera ventana que visualizará la administración en esta herramienta será la que vemos en la figura C.14 (página 112). Aquí se notificará cual será el nombre del nuevo grupo que se creará.

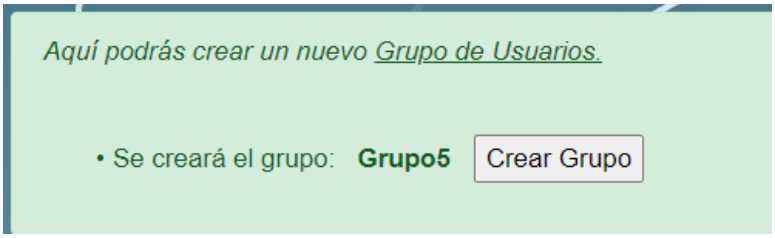


Figura C.14: Ventana inicial de creación de un nuevo grupo de usuarios.

Una vez el administrador confirme, se le pedirá que seleccione de una lista aquellos usuarios (sin grupo asignado) que desee añadir al nuevo grupo. Podemos visualizar esta situación en la figura C.15 (página 113). Finalmente, el administrador hará click en "*Crear Grupo*" y se creará el nuevo grupo.

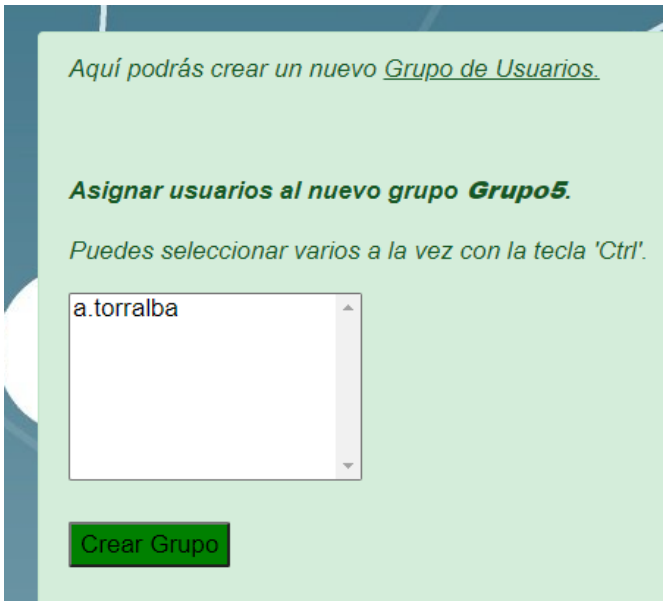


Figura C.15: Ventana final de creación de un nuevo grupo de usuarios.

La segunda opción del menú inferior de la figura C.13 (página 112), "*Ver Grupos Existentes*", le permite al administrador visualizar un resumen te todos los grupos de usuarios existentes, con sus usuarios asignados, así como aquellos usuarios sin ningún grupo. Podemos ver en la figura C.16 (página 113) un ejemplo de esta ventana.

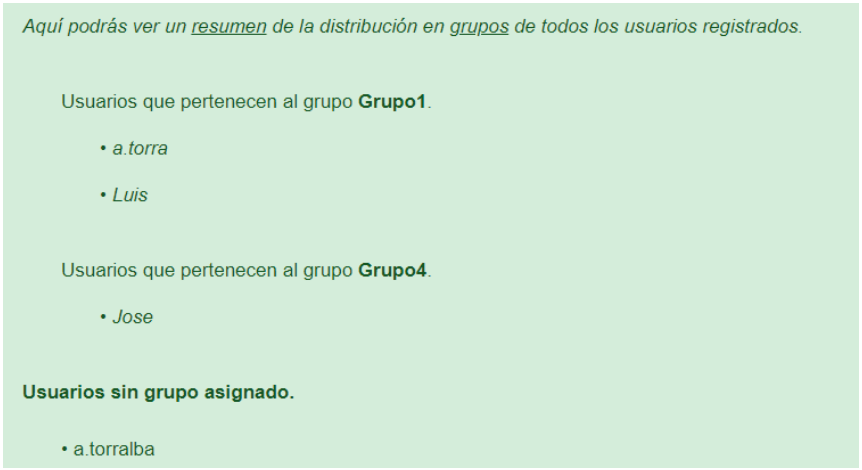


Figura C.16: Ventana de visualización de todos los grupos de usuarios existentes.

C.2.2 Ver dispositivos disponibles, gestionarlos y administrar grupos de dispositivos

En el menú de la figura C.11 (página 111), el administrador hará click en "*Dispositivos*". Se le redirigirá a la pestaña correspondiente donde el administrador podrá visualizar una lista con la totalidad de los dispositivos de red disponibles en la aplicación. En la figura C.17 (página 114) se puede visualizar lo que se encontraría un administrador en caso de acceder a este apartado.

Figura C.17: Ejemplo del apartado *Dispositivos* del panel de administración.

Cada dispositivo vendrá acompañado a su derecha de las opciones "*¿Administrador?*" y "*¿Eliminar?*". Con la primera opción se podrá editar la información del dispositivo. Con la segunda eliminaremos completamente ese dispositivo del registro. La primera opción redirigirá al administrador a una nueva ventana con un formulario con la información del propio dispositivo, la cual podrá ser modificada por el administrador en caso de necesitarlo. En caso de modificación de alguna información, el administrador deberá hacer click en el botón "*Actualizar Dispositivo*" para hacer efectiva la modificación. Un ejemplo de este formulario podemos verlo en la figura C.18 (página 115).

En la parte inferior de la figura C.13 (página 112) se visualiza un menú con las opciones "*Crear Grupo*" y "*Ver Grupos Existentes*". La primera redirigirá al administrador al apartado de registro de nuevos dispositivos de red, en el cual se le mostrará un formulario como el de la figura C.19 (página 116). El administrador rellenará todos los campos adecuadamente y, al hacer click en el botón "*Crear Dispositivo*", se registrará este en el sistema. En caso de detectarse algún error en algún campo del formulario, el sistema notificará apropiadamente al administrador. El botón "*Restablecer*" se usará en aquellas situaciones donde deseemos resetear todos los campos del formulario al estado inicial (campos vacíos). Es importante destacar el campo "*COM*", donde el administrador deberá especificar el puerto de consola al que se conecta la placa de bajo coste. Con el comando "*ls /dev*" podemos ver aquellos dispositivos de red conectados a la placa. En nuestro caso están conectados mediante cables USB, así que al hacer el "*ls*" estos dispositivos serán mostrados de la forma: "*/dev/ttyUSBx*".

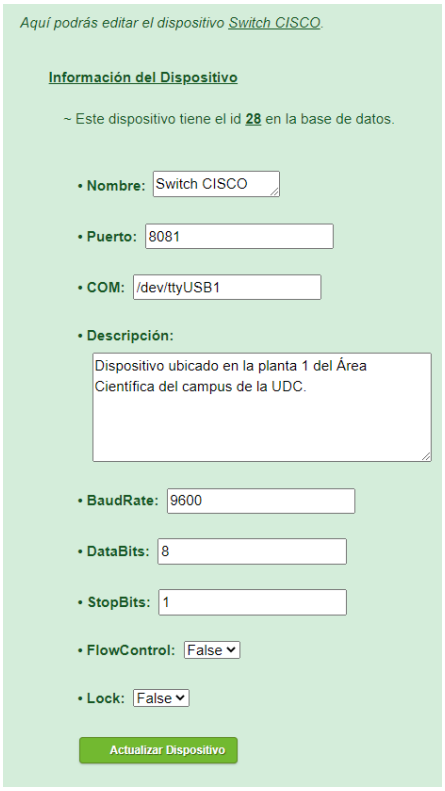


Figura C.18: Formulario de modificación de la información relativa a un dispositivo de red.

En la parte superior de la misma figura se nos ofrece la opción de registro manual de nuevos dispositivos. En esta nueva situación se nos mostrará una ventana como la mostrada en la figura C.20 (página 116).

Una vez el administrador haga click en el botón "*Crear Entrada*", el sistema creará la nueva entrada, correspondiente a ese dispositivo, en la base de datos. Acto seguido, se le proporcionará al administrador un código XML, el cual ha de ser editado adecuadamente y posteriormente insertado en el archivo "*devices_info.XML*".

Actualización de las reglas del servidor Apache al registrar nuevos dispositivos

Tanto en el registro "automático" como el manual de nuevos dispositivos de red, será preciso una modificación de las reglas del servidor Apache. Si recordamos, los servidores Node de los dispositivos de red y el servidor Apache corren en puertos diferentes, con lo que será preciso ajustar las reglas de Apache para hacer posible la redirección de puertos. Esta actualización será manual, garantizando así la integridad y seguridad de la aplicación, pues solo podrán llevarlo a cabo aquellos con acceso autorizado al host del servidor web.

Para este manual se explicará el proceso de modificación para los sistemas operativos ba-

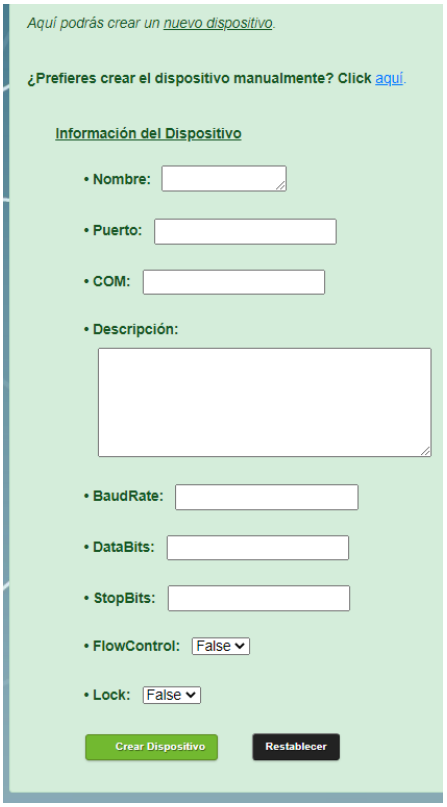


Figura C.19: Formulario de registro de un nuevo dispositivo de red.

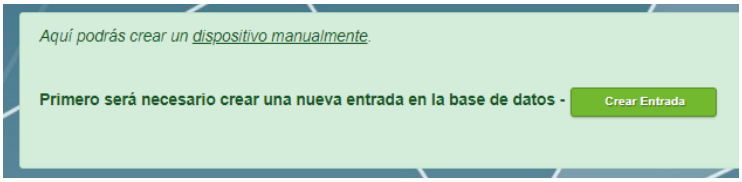


Figura C.20: Ventana de registro manual de un nuevo dispositivo de red.

sados en Linux. El administrador deberá dirigirse a la carpeta raíz de Apache, en concreto a este directorio "*/etc/apache2/sites-available*". Aquí será preciso modificar el archivo "*000-default.conf*". Este archivo deberá editarse con privilegios root. Una vez dentro, habrá que añadir al final del bloque "*<VirtualHost>*" la línea "*ProxyPreserveHost on*. Una vez listo, definiremos las reglas de dirección. Para ello habrá que añadir lo siguiente:

```
1 ProxyPass /consoleX <URL>:YYYY/
2 ProxyPassReverse /consoleX <URL>:YYYY/
3
4 ProxyPass /terminalX <URL>:YYYY/console.html
5 ProxyPassReverse /terminalX <URL>:YYYY/console.html
```

Para cada nuevo dispositivo de red, será preciso añadir las 4 líneas anteriores[1] a continuación de las ya existentes (de haberlas) en el fichero. Nótese que la "X" indica el Id en la base de datos del dispositivo en cuestión que se esté registrando. "<URL>" ha de ser sustituido por la URL de la web. "YYYY" indica el puerto en el que correrá el servidor Apache en cuestión, es importante que este puerto coincida con el establecido, para el mismo dispositivo, en el fichero XML. Por ejemplo, si trabajamos con el dispositivo con Id 4 en la base de datos, un posible ejemplo sería:

```
ProxyPreserveHost on

ProxyPass /console2 http://10.51.1.44:8080/
ProxyPassReverse /console2 http://10.51.1.44:8080/

ProxyPass /terminal2 http://10.51.1.44:8080/console.html
ProxyPassReverse /terminal2 http://10.51.1.44:8080/console.html

ProxyPass /console4 http://10.51.1.44:8081/           // NUEVA
ProxyPassReverse /console4 http://10.51.1.44:8081/    // NUEVA

ProxyPass /terminal4 http://10.51.1.44:8081/console.html //NUEVA
ProxyPassReverse /terminal4 http://10.51.1.44:8081/console.html
                                                          //NUEVA
```

Una vez modificado el fichero en cuestión y guardados los cambios, solo resta reiniciar el servidor Apache con el comando: "*sudo systemctl restart apache2*".

Antes de registrar cualquier nuevo dispositivo en la aplicación, para evitar fallas, es muy **importante** tener claro:

- Asegurarse de que el host este correctamente conectado al puerto de consola del dispositivo de red.
- El dispositivo está encendido y el administrador de red realizó su configuración inicial, previamente al establecimiento de conexión con el servidor.

C.2.3 Ver logs de los dispositivos de red

En el menú de la figura C.11 (página 111), el administrador hará click en "*Logs*". Se le redirigirá a la pestaña correspondiente donde el administrador podrá visualizar una lista con todos los archivos de registro de cada dispositivo de red. En la figura C.21 (página 118) se

[1] Destacar que si la operación fuese la de eliminar un dispositivo del registro, estas líneas deberían ser eliminadas.

puede visualizar lo que se encontraría un administrador en caso de acceder a este apartado.

Figura C.21: Ventana de visualización de logs de los dispositivos de red.

El usuario solo tendrá que hacer click encima de aquel log que desee ver y el sistema se lo mostrará. El administrador también podrá ver un registro de todos aquellos accesos fraudulentos a los diferentes dispositivos haciendo click en el botón, que aparece en la misma figura en la parte inferior, llamado "*Click aquí*". En caso de que un administrador se dirija a ver un registro de un dispositivo de red se le mostrará algo similar a lo de la figura C.22 (página 118).

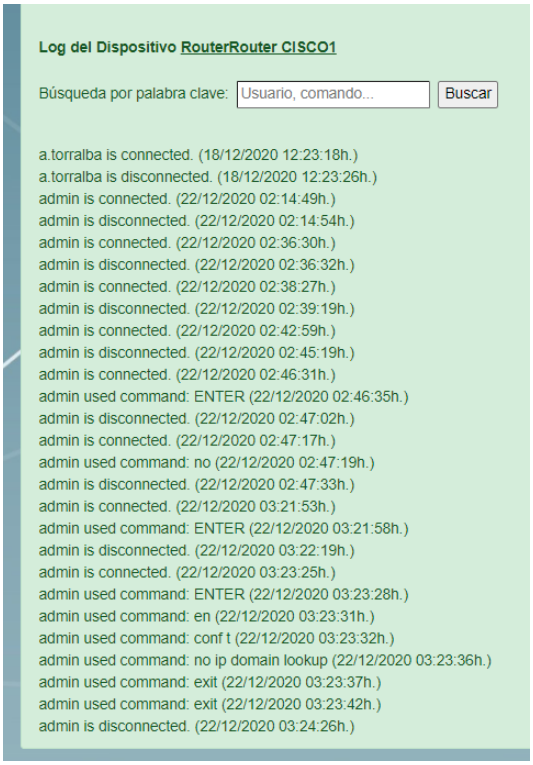


Figura C.22: Ejemplo de visualización de un log de un dispositivo de red.

El administrador también dispondrá de un campo de búsqueda por palabras clave. En este, introducirá el concepto a buscar y se le devolverán todas aquellas líneas de registro que coincidan con el criterio de búsqueda.

C.2.4 Administrar asignaciones entre grupos de usuarios y de dispositivos

En el menú de la figura C.11 (página 111), el administrador hará click en "*Asignaciones Users/Devices*". Se le redirigirá a la pestaña correspondiente donde el administrador podrá administrar las diferentes asignaciones existentes. De primeras se le ofrecerá al administrador la posibilidad de crear una nueva asignación. Se le mostrará primero un menú desplegable, contenedor de todos los grupos de usuario existentes, en el que deberá seleccionar aquel que desee asignar a un grupo de dispositivos. Acto seguido se le solicitará que seleccione el grupo de dispositivos en cuestión. Finalmente hará click en el botón "*Crear Asignación*" y esta se creará. En la figura C.23 (página 119) podemos ver la fase final de este proceso (tras haber seleccionado ambos grupos), antes de crear una nueva asignación.

Figura C.23: Ejemplo de visualización de un log de un dispositivo de red.

En la misma figura, en la parte inferior se visualiza un menú con el botón "*Administrar Asignaciones Existentes*". Este redirigirá al administrador a una nueva ventana donde podrá visualizar una lista con el conjunto de todas aquellas asignaciones existentes. Cada una vendrá acompañada de su correspondiente botón "*Eliminar*", con el que la administración podrá deshacer la asignación en cuestión. Podemos visualizar la pestaña de "*Administración de asignaciones existentes*" en la figura C.24 (página 120).

C.2.5 Resetear el estado, en la base de datos, de los servidores de los dispositivos de red

En el menú de la figura C.11 (página 111), el administrador hará click en "*Resetear Dispositivos*". Se le redirigirá a la pestaña correspondiente donde el administrador podrá resetear el estado de todos los servidores Node de los dispositivos de red. Para ello bastará con que el

Figura C.24: Ejemplo de visualización de un log de un dispositivo de red.

administrador hará click en "*Resetear Dispositivos*". Podemos ver una captura de pantalla de este apartado en la figura C.25 (página 120).

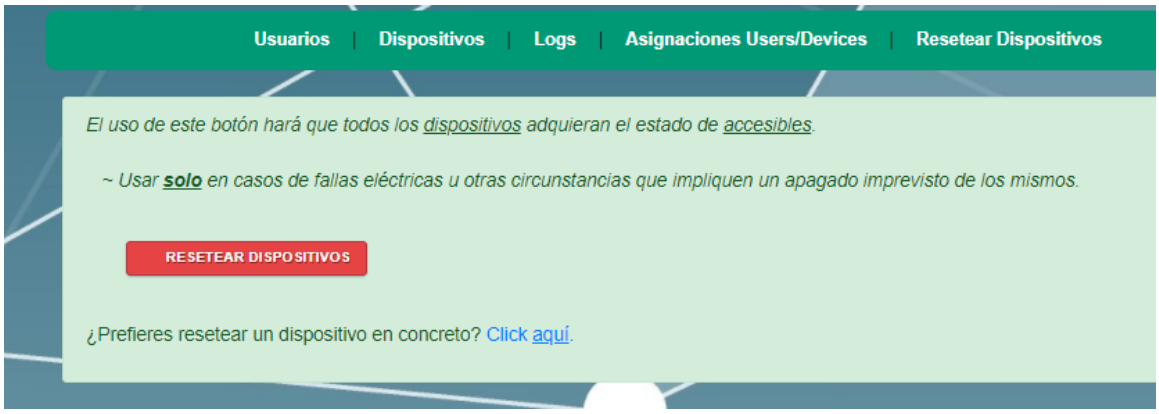


Figura C.25: Apartado de reseteo del estado de los servidores de todos los dispositivos de red.

En caso de que el administrador desee resetear el estado del servidor de un dispositivo de red en particular, deberá hacer click en el hipervínculo "*Click aquí*", que podemos ver en la parte inferior de la misma figura. El sistema lo redirigirá. En esta nueva ventana primero se le mostrará un menú desplegable en el que podrá elegir el dispositivo con el servidor que desee restablecer. Finalmente hará click en el botón "*Resetear Dispositivo*" para restablecerlo a su estado inicial. Podemos ver en la figura C.26 (página 121) la última fase del proceso manual de reseteo de un dispositivo.

Figura C.26: Apartado de reseteo del estado de los servidores de todos los dispositivos de red.

Glosario

Bootstrap Biblioteca multiplataforma de herramientas de código abierto para diseño de sitios y aplicaciones web. 39

conmutadores KVM Dispositivos que permiten a un usuario el control de múltiples dispositivos utilizando en un mismo monitor, ratón, teclado y dispositivo de audio. 11

CPD Centro de Proceso de Datos. Espacio donde se ubican aquellos recursos necesarios para el procesamiento de la información de una organización. 76

DOM Document object model. Interfaz de programación de aplicaciones que define el modo en que se accede y manipula un documento y la estructura lógica del mismo. 16

HTTP Protocolo de transferencia de hipertexto. Protocolo de comunicación que permite las transferencias de información a través de archivos en la World Wide Web. 6

HTTPS Protocolo seguro de transferencia de hipertexto. Protocolo de aplicación, basado en HTTP, que permite la transferencia segura de datos de hipertexto. 6

npm Sistema de gestión de paquetes por defecto para Node.js. 58

parseador Programa de computación que lleva a cabo la tarea de analizar una secuencia de símbolos a fin de determinar su estructura gramatical definida. 65

parser Un analizador sintáctico es un programa informático que analiza una cadena de símbolos de acuerdo a las reglas de una gramática formal. 64

PATH Variable de entorno de los sitemas operativos POSIX y Microsoft, en la que se especifica las rutas en las que el intérprete de comandos debe buscar los programas a ejecutar. 67

POST Método de petición HTTP. Utilizado para enviar una entidad a un recurso en específico, resultando en un cambio en el estado del servidor. 42

pruebas de caja blanca Técnica de pruebas de software que se centra en los detalles procedimentales del software, por lo que su diseño está vinculado al código fuente de la aplicación. 72

pruebas de caja negra Técnica de pruebas de software en las que se verifica la funcionalidad sin tener en cuenta la estructura interna de código o implementación del software. 72

Raspberry Pi Computador de placa simple de bajo coste. 13

SMTP Protocolo para transferencia simple de correo electrónico. 53

SNMP Protocolo simple de administración de red. Protocolo de la capa de aplicación que permite el intercambio de información de administración entre dispositivos de red. 6

Socket Concepto abstracto por el cual dos procesos pueden intercambiar cualquier flujo de datos. 60

SSH Protocolo que permite el acceso remoto a un dispositivo por medio de un canal seguro y cifrado. 6

Stream Secuencia de elementos de datos disponibles a lo largo del tiempo. 67

TCP/IP Conjunto de protocolos que permiten la comunicación entre los diferentes dispositivos miembros de una red. 6

Telnet Protocolo de red que permite el control remoto de un dispositivo, sin la necesidad de ubicarnos físicamente junto a este. 6

TLS Protocolo criptográfico que proporciona comunicaciones seguras a través de una red. 53

Token Aplicado a la seguridad de los datos, es el proceso de sustitución de un elemento de datos sensible por un equivalente no sensible, el cual no tiene un significado explotable. 24

UART Universal Asynchronous Receiver-Transmitter. El dispositivo encargado de controlar los puertos y dispositivos serie. 8

XML Lenguaje Extensible de Etiquetado. Es un lenguaje marcado para documentos que contienen información estructurada. 24

Xterm Emulador de terminal. 52

Bibliografía

[1] "Gestión fuera de banda (oobm)," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://www.intel.es/content/www/es/es/business/enterprise-computers/out-of-band-management.html

[2] E. Peňa and M. G. Legaspi, "Uart: A hardware communication protocol understanding universal asynchronous receiver/transmitter," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html#

[3] "Secure remote management of your critical network infrastructure," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://opengear.com/products/

[4] "Enabling the internet of things," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://www.lantronix.com/products/

[5] "Página oficial de raritan," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://www.raritan.com/

[6] "Node.js," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://nodejs.org/es/

[7] "Apache," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://httpd.apache.org/

[8] "Mariadb server," consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://mariadb.org/about/

[9] E. G. Maida and J. Pacienzia, "Metodologías de desarrollo de software," 2015, consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://repositorio.uca.edu.ar/handle/123456789/522

[10] K. Schwaber and J. Sutherland, "The definitive guide to scrum: The rules of the game," 2020, consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[11] "Boe-a-2018-3156," 2018, consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf

[12] "Retribucions p. d. i. udc - persoal funcionario," 2020, consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://www.udc.es/export/sites/udc/goberno/_galeria_down/xerencia/servizos/retribucions_seguridade_social_e_accion_social/taboas_retributivas/taboas2020/PDI-funcionario.pdf_2063069294.pdf

[13] "Condiciones de trabajo del profesorado de educación superior," 2021, consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://eacea.ec.europa.eu/national-policies/eurydice/content/conditions-service-academic-staff-working-higher-education-72_es#:~:text=profesorado%20con%20jornada%20completa%20exclusiva,clases%20y%20tutor%C3%ADas%2C%20etc.)

[14] "Retribucions p. d. i. udc - persoal laboral," 2020, consultado el 23 de febrero de 2021. [En línea]. Disponible en: https://www.udc.es/export/sites/udc/goberno/_galeria_down/xerencia/servizos/retribucions_seguridade_social_e_accion_social/taboas_retributivas/taboas2020/pdi-laboral.pdf_2063069294.pdf

[15] "Guía para la redacción de casos de uso," consultado el 23 de febrero de 2021. [En línea]. Disponible en: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416