ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

DESPLIEGUE DE INFRAESTRUCTURA Y SERVICIOS DE RED EN LA RESIDENCIA UNIVERSITARIA "ALBERTO JIMÉNEZ FRAUD": DESPLIEGUE DE INFRAESTRUCTURA, VIRTUALIZACIÓN EN CONTENEDORES Y DESARROLLO DEL FRONTEND DE LA APLICACIÓN DE GESTIÓN "PANEL DEL RESIDENTE"

NETWORK INFRASTRUCTURE AND SERVICES DEPLOYMENT AT THE MÁLAGA UNIVERSITY'S DORMITORY "ALBERTO JIMÉNEZ FRAUD": NETWORK INFRASTRUCTURE DEPLOYMENT, CONTAINER VIRTUALIZATION AND FRONTEND DEVELOPMENT OF THE MANAGEMENT APPLICATION "PANEL DEL RESIDENTE"

Realizado por

MELCHOR ALEJO GARAU MADRIGAL

Tutorizado por

PEDRO MERINO GOMEZ

ALMUDENA DIAZ ZAYAS

VICTORIANO FRANCISCO GIRALT GARCÍA

Departamento

TECNOLOGIA ELECTRONICA

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE 2018

Fecha de defensa:

El secretario del Tribunal

Resumen:

En la Residencia Universitaria “Alberto Jiménez Fraud”, se tratará la renovación de la instalación de internet por cable y el diseño y configuración de una instalación inalámbrica (WiFi), que facilite la conexión a internet de los residentes, con la adición de nuevos servicios para los mismos y, además, de mejorar la administración de la red. Aprovechando algunos servicios existentes, que se fueron añadiendo sobre la marcha, y otros nuevos, fruto de este trabajo y del trabajo complementario a este, se realizará un despliegue de servicios virtualizados de fácil administración. De entre los nuevos servicios, destacamos el “Panel del residente” que permitirá gestionar y centralizar los servicios actuales y los nuevos por venir, para uso de los residentes.

La aplicación de la nueva infraestructura de red está dando buenos resultados entre los residentes. Para los administradores de la red, la gestión y monitorización de todo el equipamiento se ha simplificado usando un solo panel de administración. Los servicios virtualizados han sido simplificados de cara al mantenimiento, actualización y mejora de éstos, de tal manera que son más ágiles realizar cambios en ellos sin tener que realizar grandes cortes en el despliegue final.

Palabras clave:

Despliegue de red, redes inalámbricas, WiFi, despliegue de servicios, contenedores, docker, aplicación web, residencia, unifi

Abstract:

In the University’s Residence “Alberto Jiménez Fraud”, we will make not only an upgrade to the wire network infrastructure, but also a design and configuration of a brand new wireless network infrastructure (WiFi) in which residents will be able to use internet easily. New services will be added for the residents, and the network administration will be improved too. With that new network infrastructure and benefiting from some already existing services, ones created over the days and others being new, as a result of this project and the complementary project, a deployment of virtualised services will be done, and will be easily administrable.

Between the new services, “Panel del residente” is the most remarkable which will allow to manage and centralise the current services and the new ones yet to come, whose target users are the residents.
The results of applying the new network infrastructure are being really positive among the residents. For the network administrators, the new infrastructure can now be managed from one dashboard which allow them to simplify the tasks of monitoring and configuring all the equipment. The maintenance, update and improvement of the services have been simplified finally, allowing more agile changes in the services and avoiding big outages of the services in the production environment.

Keywords:

Network deployment, wireless networks, WiFi, services deployment, containers, docker, web application, residence, unifi

1. Introducción

Contexto

En los últimos años, el avance en las tecnologías de la información y la comunicación (TIC) ha sido enorme indudablemente. Tal es que, actualmente, la mayoría de trabajos dependen de ellos (de alguna forma), e incluso nuestras vidas se han visto influenciadas por las TIC. Además, en el área de la educación, las nuevas tecnologías están entrando en las metodologías de enseñanza, haciendo imprescindible su uso en las aulas, bibliotecas y en los hogares de los estudiantes.

La modernización de las tecnologías en las empresas e instituciones publicas es necesario para seguir la ola de "*la modernización*" y sobrevivir en este mundo informatizado en constante cambio. Los estudiantes universitarios dependen, por tanto, de estas tecnologías y requieren que los espacios para ellos tengan de una conexión suficiente para sus tareas. Además las innovaciones ayudan al control de las instalaciones y facilitan los despliegues de WiFi y telefonía VoIP.

La Residencia Universitaria Alberto Jiménez Fraud, sobre la que se centra la actividad de este proyecto, alberga unos 250 miembros de la Universidad. La mayoría son estudiantes de grado, pero también hay otros estudiantes y profesores. En un lugar como este resulta impensable no tener una instalación TIC adecuada para el uso y disfrute de los residentes, para poder trabajar y, además, para poder vivir. Un entorno como este, permite ofrecer un conjunto de servicios a los estudiantes, y dan cabida a renovar otro tipo de servicios que actualmente no ofrecen nada desde el punto de vista informático.

Antes del proyecto, la residencia (de esta forma nos referimos a la residencia mencionada anteriormente) no disponía de un servicio WiFi suficiente para tantas personas y el servicio de telefonía, que se proporcionaba a través de la tradicional red de telefonía conmutada. En cambio, la instalación de cable de internet era bastante aceptable, pero su administración era algo manual. Todo esto hacía muy complejo la administración y el mantenimiento del despliegue de comunicaciones de la residencia.

Motivación y objetivos

Mi compañero Antonio y yo solicitamos una beca que ofrece la Universidad para trabajos informáticos en la residencia de la Universidad, de la cual ambos éramos usuarios, para mejorar la instalación WiFi y ofrecer algo a los residentes con lo que pudieran trabajar y pasar el rato. Después de hablar con el encargado de las TIC en la universidad, nos planteó un proyecto bastante ambicioso y que permitiría renovar, en esta área, la residencia. Además, de poderlo enmarcar dentro de un trabajo fin de grado en grupo, siendo el grupo Antonio y Melchor (autor de esta memoria). Se trataría renovar la instalación cableada y mejorar la de WiFi, desplegar un servicio de telefonía VoIP y un sistema de gestión de toda la infraestructura de telecomunicaciones.

Concretamente los objetivos marcados dentro del marco del proyecto son los siguientes:

- **Configuración y despliegue de la infraestructura de red**
- **Despliegue de servicios en una infraestructura más cercana a la "cloud"**
- **Despliegue del servicio VoIP**
- **Sistema software de gestión**

De los cuales, el punto de VoIP y la mitad del ultimo punto se podrá ver en el trabajo «*Despliegue de infraestructura, implantación de telefonía IP y desarrollo del backend de la aplicación de gestión "Panel del residente" en la residencia universitaria de la Universidad de Málaga*» de Antonio Ángel, como parte de la división de tareas del proyecto. Por lo tanto, en este trabajo, la estructura que se seguirá será:

- **Diseño, configuración y despliegue de la infraestructura red**
  - *Análisis de la situación inicial, búsqueda de solución y alternativas, y diseño final*
- **Despliegue de servicios en una infraestructura más cercana a la "cloud"**
  - *Los servicios que existan, más los nuevos, se convertirán a contenedores, como servicio de virtualización y despliegue*
- **Sistema software de gestión**
  - *Requisitos, modelo de base de datos y frontend del software de gestión*

Estado del arte

El grupo de estándares IEEE 802.11 ha evolucionado mucho durante estos años. De inicialmente usar en nuestros hogares una conexión máxima de 54Mbps (estándar 802.11g) a actualmente tener velocidades de hasta 145Mbps o 300Mbps (estándar 802.11n dependiendo del dispositivo y del punto de acceso) y de hasta 867Mbps o casi 2Gbps (estándar 802.11ac dependiendo del dispositivo y del punto de acceso). Tecnologías como MU-MIMO (*Multiple User Multiple-Input Multiple-Output*), nuevas modulaciones (como 256-QAM) y un amplio ancho de canal (80MHz) para el estándar 802.11ac, le permiten alcanzar altas tasas de datos[32].

La acogida de este estándar ha sido enorme: se pueden encontrar ahora mismo en muchos hogares del país (incluso usted puede tener un aparato con 802.11ac ahora mismo); y a nivel empresarial, los fabricantes han apostado rápidamente por este estándar. Fácilmente se puede observar que todos los puntos de acceso (o los "routers" de casa) que se distribuyen incluyen soporte para 802.11g, n, an y ac.

Aunque existen nuevos campos de investigación para la mejora de esta tecnología, como el 802.11ax[1], actualmente el 802.11ac parece que va a quedarse para algunos años más. Y eso se puede aprovechar para nuestro objetivo en la residencia.

Actualmente, los servicios de internet se montan sobre una infraestructura en la nube. La disponibilidad de internet hace que muchas de nuestras aplicaciones puedan estar conectados y compartir información con otros usuarios. Todo eso requiere de un despliegue, y deben ser fáciles de mantener y de mejorar.

La virtualización es el mejor método para poder distribuir los servicios entre diversos servidores y aprovechar su potencia para el fin de la empresa. Ya sean máquinas virtuales o contenedores, la virtualización se usa en prácticamente cualquier servicio que se pueda imaginar. En general, los servicios en la nube se montan sobre diversas capas. Si contrata un servicio de una capa, solo se maneja esa capa y las superiores. De acuerdo con [60], la implementación de esos servicios se suele realizar sobre virtualización para obtener aislamiento. Las capas mas comunes en la nube son:

- **Infrastructure as a Service (IaaS)** ofrece servicios de virtualización con máquinas virtuales, virtualiza máquinas y su hardware

- **Platform as a Service (PaaS)** ofrece un entorno de despliegue para aplicaciones, que pueden añadirse servicios extra a necesidad del desarrollador
- **Software as a Service (SaaS)** ofrece una plataforma para que usuarios puedan usar cierto software desde cualquier lugar sin tener que instalarlo en sus dispositivos

Y los despliegues en la nube pueden hacerse en:

- **Nubes privadas** la infraestructura es propia de la empresa o institución, ellos ponen ponen todo para tener el despliegue en marcha
- **Nubes públicas** la infraestructura puede ser propia o ofrecida por un tercer (como Amazon Web Services o Google Cloud) y es accesible públicamente
- **Nubes híbridas** es una mezcla de los conceptos anteriores

Como se puede observar, la computación en la nube es muy amplia y requiere de mucha tecnología para hacerla funcionar. De los tipos de nube, la que más se ajusta a nuestras necesidades es *SaaS*, ya que podría ayudar, tanto a nosotros como a los futuros becarios de la residencia, a manejar el despliegue de servicios y poderlo probar parte de él fuera de la infraestructura real. Los requisitos de la *SaaS* a desplegar vendrán dados por los servicios que se quieran utilizar.

Tecnologías

Para el **despliegue de la red** se usará las tecnologías comunes:

- El grupo de estándares 802.11 para la red inalámbrica, en su versión 802.11ac
- Para el cable, Ethernet 802.3 con uso de par trenzado 1Gbps y fibra óptica 1Gbps
- Se usará VLANs para la separación de diversas redes (802.1Q)
- Para una de las redes WiFi, se usará 802.1X y WPA2-Enterprise, el resto serán WPA2-PSK

Para el **despliegue de servicios** se usará la tecnología de contenedores Docker[2].

Para el **frontend del software de gestión** se usará los lenguajes JavaScript ES6 + TypeScript[4] con react.js[5] y redux[6] para la presentación y lógica de la aplicación web, y bootstrap[7] como framework de diseño y estilo.

2. Diseño, configuración y despliegue de la infraestructura de red

Descripción del estado inicial de la red

En este apartado se describirá el estado inicial de la red de la residencia, tal y como estaba antes de hacer los nuevos despliegues, que se irán describiendo en este capítulo.

En primera instancia, se hablará del estado de la **instalación cableada**, que sustenta a todo lo demás. El diseño de esta instalación fue realizada por un equipo del Servicio Central de Informática a principios de siglo y se ha mantenido sin apenas modificación hasta 2017.

Distribución de zonas de conexión cableada en la residencia - mapa basado en [30]

La infraestructura estaba encabezada por un servidor, que se describirá posteriormente, que se conectaba con un pequeño switch de 8 puertos.
Éste repartía la conexión a dos switches de 24 puertos y a dos conversores de fibra óptica. Los dos switches de 24 puertos daban conexión directa a 46 apartamentos y dos apartamentos iban conectados directamente al switch de 8 puertos. Eso conforma una mitad de la residencia, la izquierda mirando desde la entrada. Le llamaremos a partir de ahora “lado izquierdo”.

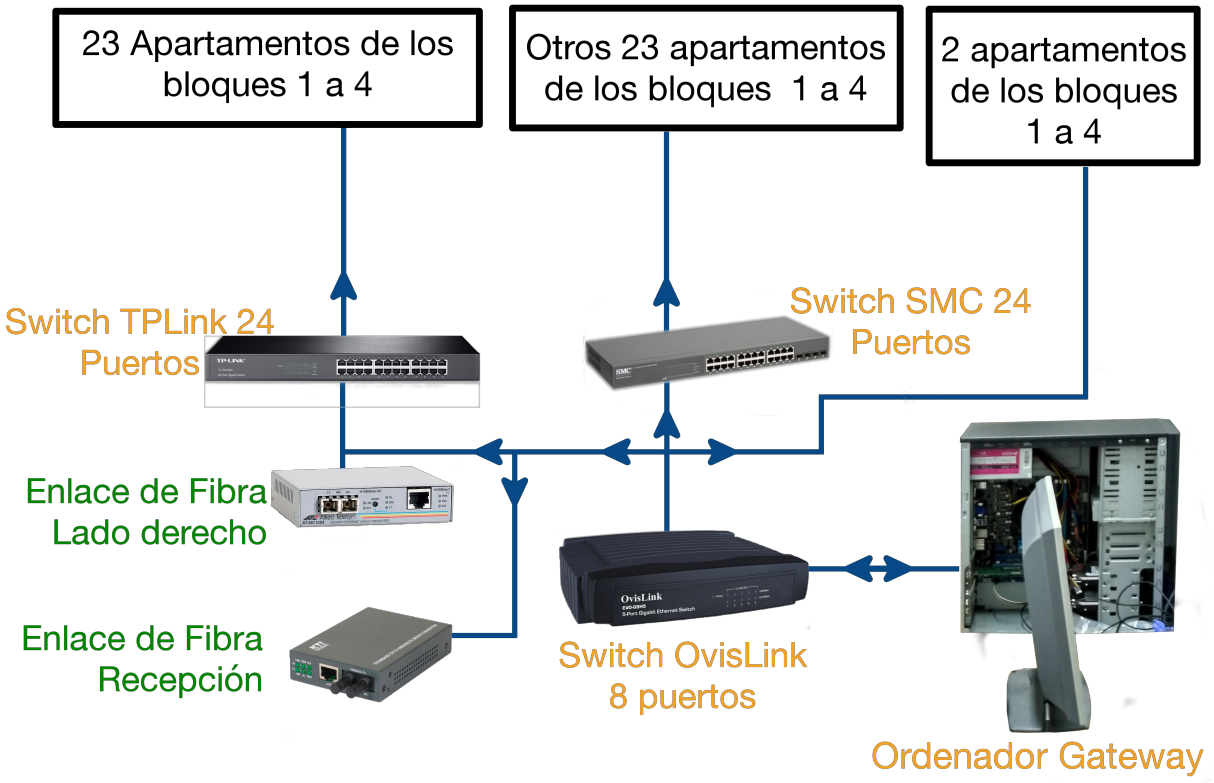


Gráfico que ilustra la parte de infraestructura comentada anteriormente

De uno de los enlaces de fibra, el que está etiquetado como “lado derecho”, va a otra sección con otros dos switches de 24 puertos cada uno y un receptor de fibra óptica. Dicho lado derecho es la mitad derecha de la residencia, que se puede apreciar en el mapa anterior. Al igual que en lado izquierdo, los switches de 24 puertos conectan directamente los apartamentos de esta mitad, que en este caso son menos que en la primera mitad.

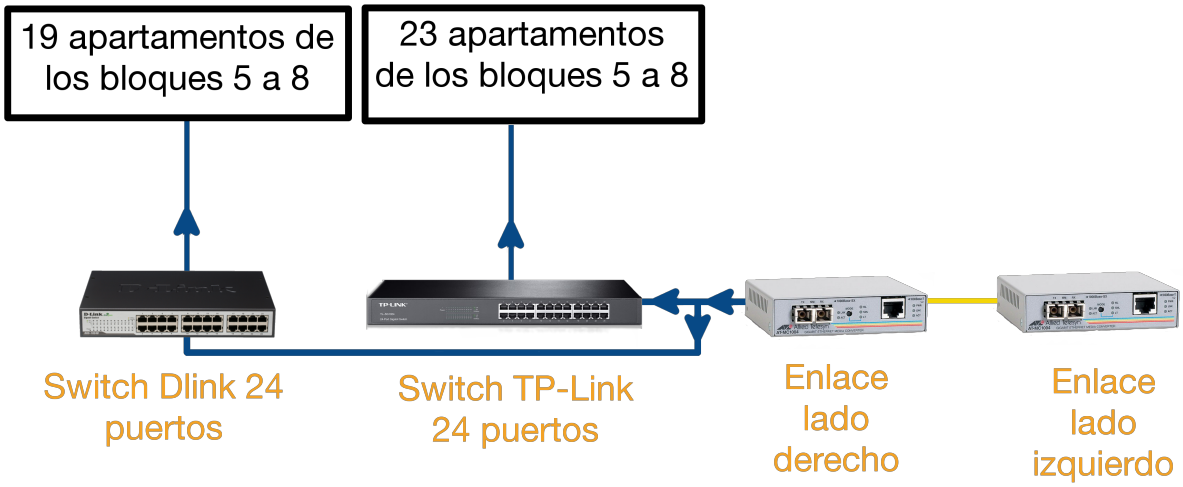


Gráfico que ilustra la siguiente parte comentada anteriormente

Queda por explicar el otro enlace de fibra etiquetada como “recepción”. Éste conecta con la recepción de la residencia para dar internet a la sala de estudio/ biblioteca y al salón de actos. Ahí se podia encontrar inicialmente un pequeño switch que proporcionaba conectividad al salón de actos y a la sala de estudios, estando el ramal que se dirigía hacia el salón de actos estropeado. En la sala de estudio se podia encontrar un punto de acceso para dar WiFi en esta zona y alrededores.

Gráfico que ilustra la ultima parte (recepción) comentada anteriormente

Antes se ha mencionado un **servidor en la infraestructura**. Éste es el encargado de ofrecer la conexión de internet a toda la residencia. El servidor tenia instalado el sistema operativo Linux cuya distribución era Ubuntu Server 16.04 LTS, y dentro del software instalado en dicho servidor, nos encontramos con:

- **Firewall**: mediante *firehol*[8] (interfaz sencilla para *iptables*[9]), se configuraba para crear un NAT entre la conexión interna y la externa, bloquear los puertos externos por seguridad y aplicar políticas de seguridad ante ciertos ataques.
- **Calidad del servicio**: mediante *fireqos*[8] (interfaz sencilla para *tc*[10]) para ofrecer una configuración de calidad del servicio y dar prioridad a cierto tráfico para maximizar el rendimiento cuando se sature la conexión.
- **DHCP**: el servidor se usaba con *dnsmasq*[11], con la siguiente configuración:
  - Rango de IPv4: 10.10.10.5 - 10.10.12.254
  - Mascara de red: 255.255.0.0
  - Puerta de enlace: 10.10.10.1
  - Servidor DNS: 10.10.10.1

- **Servidor DNS:** el propio *dnsmasq* hace de servidor DNS también, incluye también los dispositivos conectados, cada uno obtenía un dominio propio local.

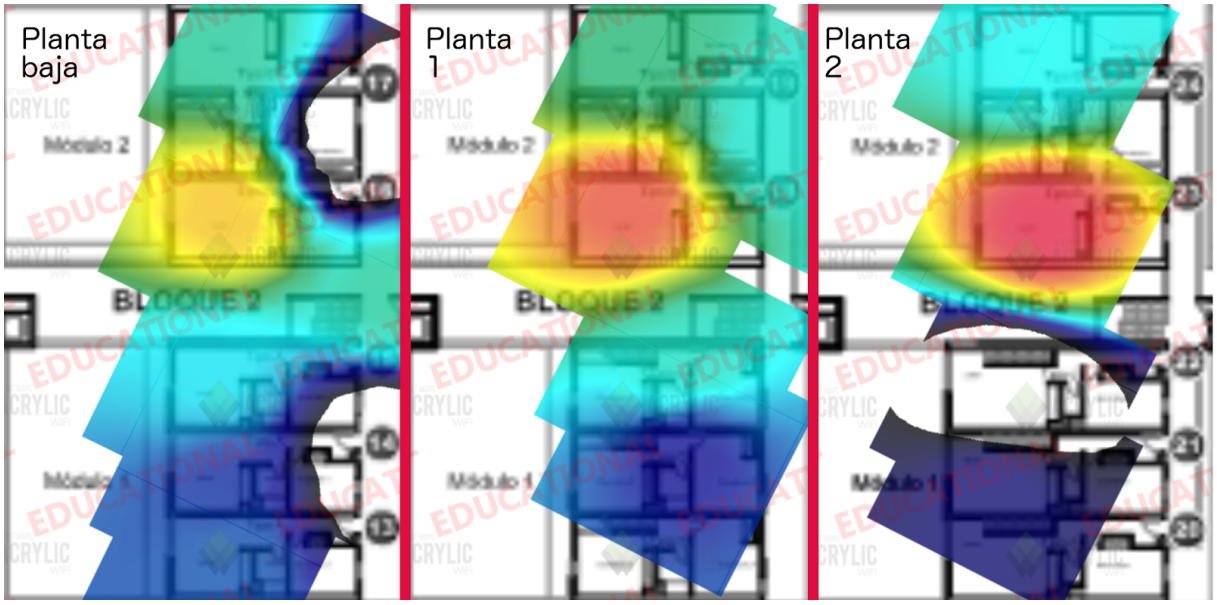
Como se puede observar, el servidor necesita dos tarjetas de red, y así es. Una es la que conecta con la infraestructura interna, anteriormente comentada; y la otra es la que conecta con el servicio de internet ofrecido por Ono/Vodafone. Este servicio inicialmente eran 50Mbps simétricos y se consiguió ampliar a 200Mbps simétricos, ambos de fibra óptica.

A modo de ejemplo, para ilustrar la infraestructura, un usuario que se conecta por cable o por alguno de los puntos de acceso (que se comentará en breve), recibía una dirección IP del servidor DHCP. Al enviar un paquete, pasaba por los switches hasta llegar al servidor, que aplicaba las reglas de NAT (traducir la IP de origen y el puerto) y emitía el paquete hacia la puerta de enlace de Ono y volvía a aplicar otro NAT, para acabar finalmente en internet.

Tratemos ahora la situación del **despliegue de WiFi**. Había repartidos por varios apartamentos unos routers neutros como puntos de acceso. Con *router neutro* nos referimos a routers que uno puede comprar en cualquier lugar, que no son de ninguna empresa telefónica y que sirven para cualquier tipo de servicio que trabaja a nivel de capa de red. Los *puntos de acceso* son los aparatos que trabajan sólo hasta nivel de enlace y emiten la señal inalámbrica. Se conectan al router a través de un cable Ethernet, y en algunos casos, a través de otro punto de acceso.

Estos routers neutros tienen incorporados también la funcionalidad de punto de acceso (o *AP* para acortar). Repartidos por la residencia, se podían encontrar puntos de acceso de diversos modelos: la mayoría eran TP-Link WR841N(D), algunos TP-LINK WR1043ND que son mas potentes, y algún que otro OvisLink (de modelo desconocido) de peor potencia que el resto. La colocación de éstos era aleatoria, incluso algunos los movían los propios residentes sin notificarlo. En ambos casos, hacían que en algunas zonas no hubiera cobertura a penas, y en otras, los APs se interfirieran entre ellos.

Para finales del primer cuarto de 2016, se consiguió comprar unos puntos de acceso Unifi AP-LR. Con ellos, se consiguió mejorar la cobertura, pero aún así no era suficiente.



Heatmap de cobertura - Rojo mas potencia, Azul menos potencia

En el gráfico superior se puede observar lo que se denominan como *heatmaps* de cobertura de WiFi en uno de los mejores bloques en cuanto a cobertura. El gráfico junta las tres plantas de un bloque (numerado como el 2º) y solo muestra la potencia de señal de todos los puntos de acceso pertenecientes a la residencia. Como se puede observar, una mitad del bloque se ve bastante desfavorecida por una calidad de señal muy inferior; mientras que en la otra mitad, la señal llegaba a casi todos los lugares. Otro factor importante a tener en cuenta es que los apartamentos de la izquierda (en el gráfico, los que están más abajo) están dando junto a la calle y las interferencias de los otros puntos de acceso de los bloques de pisos hace que la conexión sea peor.

Los modelos de routers neutros que se han mencionado anteriormente, su objetivo no es el mercado profesional como se demanda en un lugar como este. En palabras de la web oficial del TP-Link WR841N, ”Excelente capacidad para mitigar la pérdida de datos a larga distancia y a través de obstáculos en una pequeña oficina o en una vivienda grande”[12], aunque no describe el modelo que se tenia en la residencia, da una idea de que el objetivo no es el uso profesional. Por tanto, estos routers neutros no eran los ideales para ofrecer una conexión por WiFi en la residencia.

Como medida temporal, se permitió a los residentes traer sus propios aparatos para que ellos se pudieran conectar por WiFi, ante la falta de una instalación adecuada. Esta decisión traía consigo una serie de problemas, pasando por que la mayoría de

residentes no saben como instalarlo y hacerlo funcionar, además, si alguno se desconfiguraba, podia hacer que dejara de ir la conexión de internet en parte o toda la residencia.

Los dueños, muchos de ellos, solían traernos sus aparatos para que se lo dejáramos preparado para usar, pero otros lo enchufaban directamente y provocaban "el caos". En el caso de que alguno se desconfigurase (incluyendo a los APs de la residencia) podían provocar el mencionado "caos". Con esto nos referimos a que dichos aparatos volvían a su configuración inicial, donde tenían un DHCP ya que actuaban como routers. Para actuar como APs, se tenia que conectar el cable ethernet en el switch para LAN (justo lo contrario a lo común que es conectarlo en el puerto WAN o de internet). La configuración por defecto es ofrecer un servidor DHCP que funciona tanto por su WiFi como por su switch LAN, haciendo que los equipos de la residencia fueran a conectarse a este router (sin internet) en lugar del servidor de la residencia.

Esta situación hace inviable el modelo actual de instalación WiFi, haciendo trabajar a los becarios en exceso, cuando las alternativas profesionales no requieren de tanto trabajo. Como ejemplo, los APs Unifi, aquellos mencionados anteriormente que se compraron, se manejan desde un panel de administración web centralizado, ayudando enormemente en el trabajo de administración de la red.

La red WiFi de la residencia se ofrecía sobre una red abierta, y las redes de cada residente que traía su propio aparato era a elección de él mismo, aunque la mayoría eran con contraseña. El mayor problema de estas redes es que al ser abiertas no tienen ningún cifrado y "la captura de estos paquetes de información es muy sencilla; utilizan programas que capturan los paquetes que viajan por la red, y que no son complicados de utilizar"[13]. Incluso se pueden realizar "honeypot"s que consiste en una persona (*ladrón*) que "crea una red WiFi abierta, muchas veces haciéndose pasar por un comercio o lugar conocido, para que nos conectemos a ella. E intentará robarnos la mayor cantidad de información posible"[14].

Por último, aunque el equipamiento para la infraestructura de cable era bueno, alguno de los switches dejaron de funcionar momentáneamente, en ciertas ocasiones. Además, uno de los conversares de fibra no era gigabit (1000Mbps), lo que podia provocar un cuello de botella.

Obtención de requisitos de la nueva instalación

Después de analizar el estado inicial de la instalación, sus puntos positivos y negativos, se puede realizar una búsqueda de requisitos sobre como debería quedar la instalación final, que permitirá guiar en las decisiones para la implementación de la instalación.

Los usuarios finales de la instalación serán los residentes, a cada uno de los cuales se les quiere proporcionar conectividad Wi-Fi, Ethernet y un teléfono VoIP. Suponiendo que cada residente tendrá al menos dos dispositivos (un ordenador y un teléfono), y sabiendo que la residencia puede albergar unas 250 personas, la instalación debería soportar 500 dispositivos conectados simultáneamente.

Los residentes, como se ha comentado anteriormente, no solo estudian, también viven, por lo que la instalación debería ser capaz de soportar a residentes con casos de uso como videojuegos online o videos y música en streaming. La navegación por internet también es un requisito, pero requiere menos recursos que lo anterior por lo que se considera ya incluido.

La seguridad es uno de los puntos mas importantes actualmente. La red WiFi, que se preve que será la más usada, debe estar protegida con WPA2-PSK o WPA2-Enterprise (protocolos de protección y cifrado de una red WiFi).

Para evitar un flujo de paquetes innecesarios entre dispositivos por WiFi, reduciendo el uso de la red, se debería poder usar un mecanismo de aislamiento entre dispositivos y bloquear los mensajes de "*broadcast*" para éstos. Al ser una red grande, mantener el uso de los APs al mínimo es importante para tener una red estable y usable. Así se evitarían problemas como el del Chromecast a principios de 2018[15], en el que saturaba la red WiFi de los hogares por enviar demasiados paquetes por la red WiFi, saturándola.

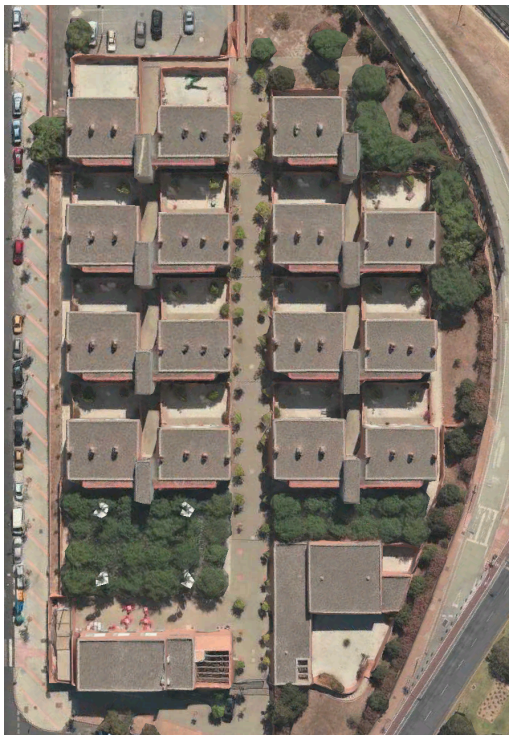
Con el objetivo de separar el tráfico de cada una de las redes y aumentar la seguridad, se debería implementar diversas VLANs, aislando los dispositivos por diversos segmentos. Dicha separación podría ser un segmento para la instalación VoIP, otra para el manejo de los switches y APs, y una última para el acceso a internet de los usuarios. Con las VLANs, se pueden crear redes virtuales que viajan

por la misma instalación pero dando la sensación de que son instalaciones distintas y separadas con un único punto en común.

Toma de decisiones en base a los requisitos

Conforme a los requisitos anteriores, en este apartado se detallarán las decisiones mas técnicas para el despliegue que influirán en los siguientes apartados.

Lo primero que se pensó fue que había que pensar cuantos puntos de acceso iban a hacer falta y de que marca y modelo se comprarían. Y sobre lo primero dependenderian las necesidades de los switches. Teniendo claro estas decisiones, se podría empezar a pensar el resto de apartados de diseño y configuración.



Residencia Universitaria[30]

Se va a describir la estructura de los edificios de la residencia brevemente para que se entienda la gran problemática de una instalación WiFi en ella. La residencia se distribuye en 8 bloques: 7 bloques de 12 apartamentos y uno de 6 apartamentos. Cuatro van a una mitad (izquierda del mapa), los otros cuatro a la otra (derecha del mapa); separados por un pasillo ancho. Siete de los ocho bloques está dividido en dos, y en medio de la separación, se encuentra un pequeño pasillo.

Los lados izquierdos de cada bloque (mirando desde la entrada, abajo de la imagen hacia arriba) tiene tres apartamentos en la planta baja (de 2 personas) y tres apartamentos en la segunda planta (de 4 personas), con dos plantas cada uno (dúplex). En los lados derechos de cada bloque, en cambio, hay dos apartamentos en la planta baja, dos en la primera planta y dos en la segunda; todos de tres personas. El bloque con una sola mitad, sigue la estructura del lado izquierdo.

En cada apartamento, solo hay una toma de pared de cable Ethernet. Eso incluye a los apartamentos de dos plantas, que solo hay toma en la planta superior.
Resumiendo estos datos, tenemos que la residencia puede albergar como máximo a unas 260 personas como máximo en 90 apartamentos.
Este es el reto a solventar: analizar cual seria la cantidad de puntos de acceso necesarios para una instalación a la altura para esta residencia, con tantos huecos y tan esparcida.

Una de las primeras ideas que se barajó, fue instalar un punto de acceso de pared en cada apartamento, al igual que algunos hoteles acaban haciendo (según informó un comercial de Ono/Vodafone). De esta forma, se conseguiría una gran cobertura en todos los apartamentos. En contra tenemos que al ser algo que está en la pared, y se debería colocar en un lugar próximo a la toma de cable, tenemos el problema de que podría sufrir muchos golpes de los residentes. Otro motivo por lo que se estuvo en contra es que la mayoría de ellos (de los modelos que se observaron) emiten la señal WiFi principalmente para una planta, por lo que los residentes de los dúplex (los apartamentos de dos plantas) podrían sufrir de mala cobertura en la planta inferior. Por lo que se descartó la idea de este tipo de puntos de acceso.

Decidimos que usaríamos modelos de APs mas *tradicionales*, que se parezcan a los que la propia Universidad está usando actualmente. Con estos modelos, los APs se podrían colocar en lugares donde sufran menos de golpes accidentales, incluso deberían bastar para llegar a la planta inferior de los dúplex. Este tipo de APs son bastante mas potentes que los anteriores, incluyendo los modelos pensados para baja densidad de dispositivos. Esto nos llevó a la idea de que un punto de acceso por apartamento iba a ser demasiado, ya que estos modelos emiten con mayor potencia, podrían llegar a más lugares, pero a la vez podría haber mucha interferencia entre ellos.

Siguiendo con el mismo modelo de punto de acceso, plantemos si poner un punto de acceso por cada dos apartamentos o un punto de acceso por cada tres. Ahora entra en juego como seria la cobertura de los puntos en un entorno real para saber responder a esta pregunta, por lo que hay que decidir marca y modelo de AP para poder ir a la siguiente fase.
Aunque partíamos de la idea de usar Cisco Meraki para los puntos de acceso, acabamos decidiendo que iba a ser mejor opción usar Ubiquity Unifi por tener ya algún equipamiento de esa empresa de antes, además de haber comprobado que su rendimiento era el adecuado para el tipo de despliegue que se necesitaba.

Ahora quedaba elegir qué switches íbamos a usar para el despliegue, para complementar la instalación. Inicialmente se sugirió usar switches de Cisco, pero como los puntos de acceso iban a ser de Ubiquity Unifi, iba a ser mejor opción usar switches de la misma marca para tener una mejor integración en la instalación y en el panel de administración de Ubiquity.
Un requisito indispensable para los switches es que debían de ser de 48 puertos mínimo, y tener 4 de estos. El porqué se debe a que, como se comentó en el apartado "*Descripción del estado inicial de la red*", hay que repartir internet a 48 apartamentos para una mitad y otros 42 para la otra mitad de la residencia. Además, al añadir puntos de acceso, se necesitarán más cables para éstos. Poniéndonos en el peor caso de un AP para cada dos apartamentos, se necesitarían 24 cables extra en la primera mitad y 21 cables extra para la segunda. El total se resume en la siguiente tabla:

	Mitad 1	**Mitad 2**
Apartamentos	48	42
Puntos de acceso	24	21
Total puertos en switches	72	63

Por tanto, dos switches para cada mitad: uno dedicado a apartamentos y otro dedicado a los puntos de acceso.

Además, existen dos puntos de acceso fuera de los apartamentos: uno en la sala de estudios y otra en el salón de actos. Para hacer llegar internet a esta sección, con un switch de 8 puertos bastaba (es lo más pequeño que se puede encontrar).

Resumiendo, se decidió lo siguiente respecto a la instalación nueva de red:

- 1 AP por cada dos o tres apartamentos
- APs de Ubiquity Unifi
- 4 switches de 48 puertos de Ubiquity Unifi - 2 por mitad
- 1 switch de 8 puertos de Ubiquity Unifi

Diseño y configuración de la red fija

Este es uno de los puntos más importantes de la red, ya que es la parte de la infraestructura que soporta al resto. Por lo que un buen diseño hará que dé un buen resultado la red por cable.

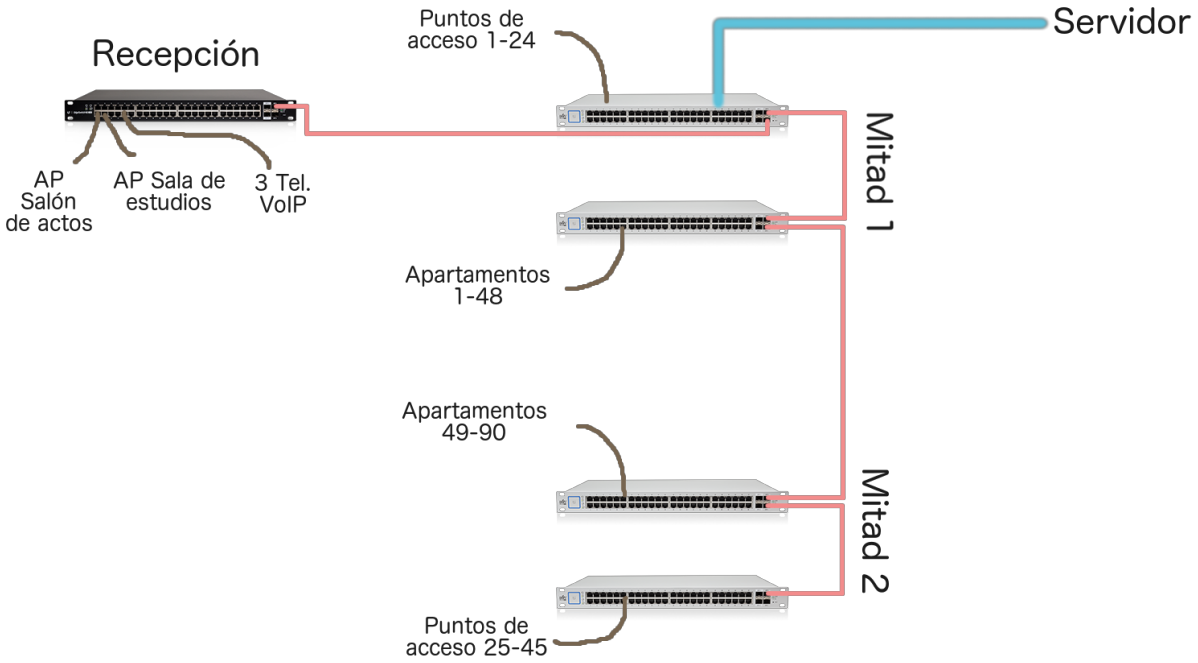


Gráfico que describe la distribución de la red fija (o de cable)

Como se comentó al principio de la sección "*Descripción del estado inicial de red*", el diseño inicial fue creado por un grupo del Servicio Central de Informática de la

Universidad, por lo que vimos adecuado replicar la estructura para el nuevo despliegue, pero adaptándolo a los nuevos dispositivos y sus posibilidades.
La nueva distribución, que se describe visualmente en el gráfico de la anterior, consiste en que el servidor se conecta al primer switch de la primera mitad. Éste se decidió que repartiera la conexión a los 48 APs de la misma mitad. Por enlace de fibra, se conectaba con el segundo switch de esta primera mitad y con el switch de recepción (que luego se comentará). Los switches de este modelo tienen todos 4 puertos adicionales. Por lo que, estos puertos se usan para los enlaces de fibra switch-switch.
El segundo switch tenia todas sus puertos ocupados con los 48 apartamentos. Por lo que se conecta con el primer switch de la otra mitad de la residencia mediante enlace de fibra, usando esos puertos adicionales.
El de la segunda mitad reparte conexión a los 42 apartamentos de esta mitad de la residencia. Y, por enlace de fibra, se conectaba con el segundo switch.
El segundo switch de esta mitad reparte la conexión a los puntos de acceso de esta segunda mitad.
Por último, el switch de recepción. Como se comentó, iba a ser inicialmente un switch de pocos puertos, pero a la residencia apareció un Unifi EdgeSwitch de 48 puertos. Éste es de otra linea de productos, por lo que no se conecta con el panel de Unifi, y se configura manualmente mediante su propio panel. Continuando con la descripción de la topología, este switch por fibra recibe la conexión y la reparte a los puntos de acceso de la sala de estudios y el salón de actos, además de tres teléfonos VoIP.

Respecto a la configuración de la red, distinguimos 4 redes, que se implementan sobre VLANs:

- **Red de administración**: es la red que usan los switches, puntos de acceso y el controlador Unifi para comunicarse y controlar la red.
- **Red WiFi:** es la red que usan todos los usuarios que se conectan por WiFi.
- **Red cableada:** es la red que usan todos los usuarios que se conectan por cable al teléfono del apartamento[16], los teléfonos ofrecen un puerto extra para conectarse por cable a la red, y éste etiqueta el tráfico de ese puerto de forma distinta al del VoIP.

- **Red VoIP:** los teléfonos VoIP son los que usarán esta red para la comunicación con la centralita y entre los teléfonos.

Una vez descritos el objetivo de las redes, se va a describir la configuración más técnica de cada una, mediante la siguiente tabla:

	Administración	WiFi	Cableada	VoIP
Red IPv4	10.10.10.0/24	10.10.0.0/21	10.10.20.0/24	10.10.11.0/24
Puerta de enlace	10.10.10.254	10.10.0.1	10.10.20.1	10.10.11.254
VLAN	No	10	15	7
Aislamiento	No	Si	No	No
DHCP Estático	Si	No	No	Si

En la tabla anterior, se puede ver cuales son las redes IPv4 (descritas por IP de la red y máscara), la puerta de enlace, el numero de la VLAN, si los switches hacen aislamiento entre dispositivos y si el DHCP es estático. Con esto último se está refiriendo a que el DHCP en modo estático no ofrece una IP si no está dentro de una lista definida manualmente. De esta forma, es más difícil que una persona se conecte a una de esas redes y que le funcione.

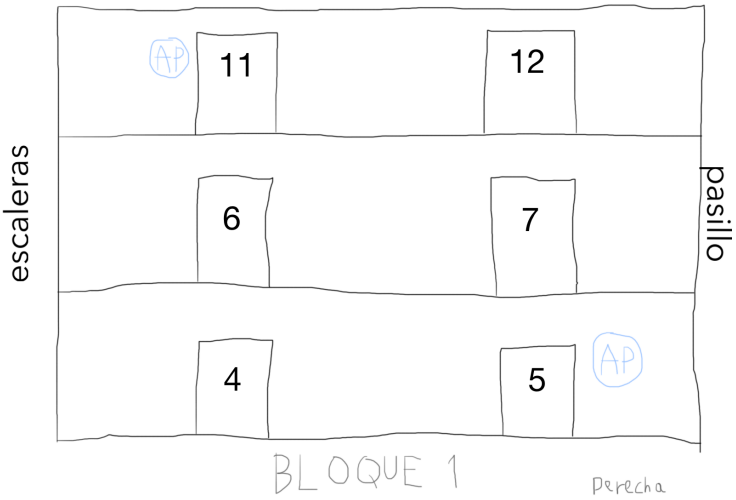
Diseño de la red WiFi

Para empezar con esa fase, el Servicio Central de Informática nos proporcionó un mapa de cobertura que realizaron mediante una simulación con el software Ekahaut sobre la banda 2,4GHz, suponiendo que los puntos de acceso serian los Unifi AC Lite. Además, nos recomendaron usar este modelo de puntos de acceso ya que veían que iban a ser idóneos para el despliegue.

Sobre este modelo de APs, uno solo es capaz de soportar a 200 clientes en condiciones normales tanto en 2,4GHz como en 5GHz, según el documento de especificaciones del modelo[17], además de que su alimentación se realiza sobre el propio cable de internet (Power over Ethernet).

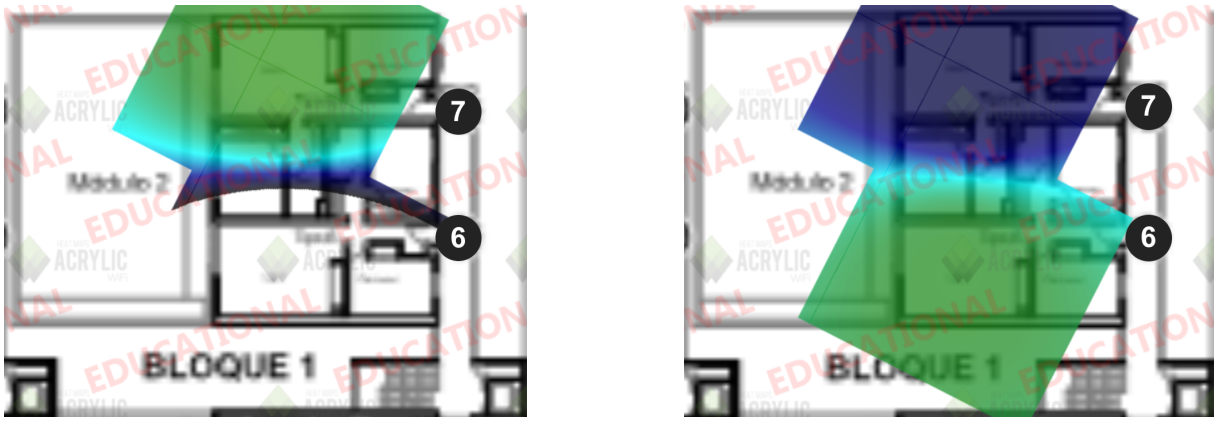
La simulación, realizada por el SCI, consistía en colocar dos puntos de acceso por módulo formando una diagonal. Con módulo se refiere a una mitad de un bloque, y la diagonal es colocar uno en la planta baja y en un lado del módulo, y el otro en la segunda planta en el lado opuesto del módulo. El dibujo de la esta siguiente página

resume esta distribución sobre un módulo. Los puntos de acceso se colocarían en el techo apuntando uno hacia abajo y otro hacia arriba (el del apartamento 11 y 5 respectivamente).



Dibujo hecho el 19 de abril de 2017 representando la prueba de campo de la simulación

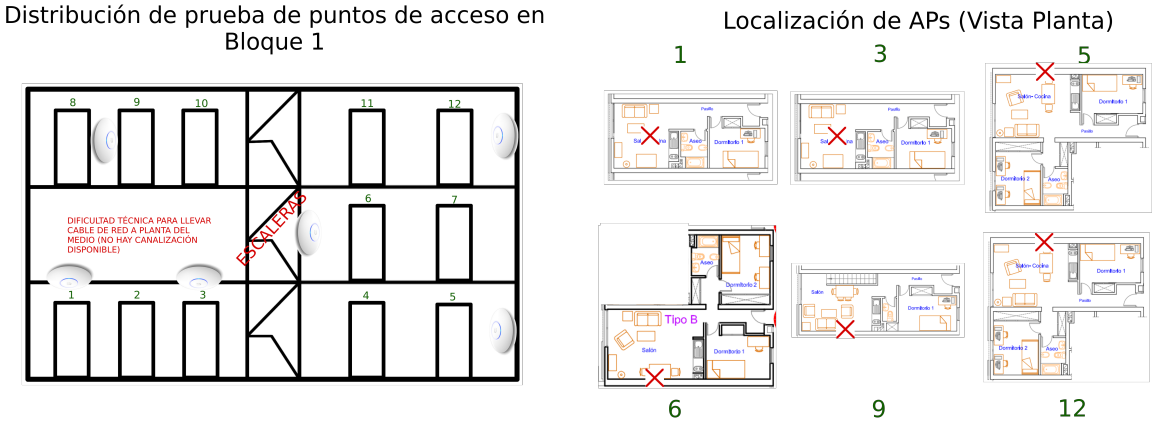
Sobre la simulación se hizo una prueba de campo con un punto de acceso que nos prestaron. El objetivo era ver como llegaba la señal del punto de acceso sobre todos los apartamentos del módulo de la derecha del bloque 1 (ver imagen de nuevo) colocando primero el punto de acceso en el apartamento 5 y luego en el apartamento 11. Las pruebas se hicieron sobre cada apartamento, tanto en 2,4GHz y 5GHz, usando una licencia educativa de Acrylic WiFi Heatmaps para la obtención de estos mapas.



Heatmaps *en la primera planta de las pruebas de cobertura sobre el AP en el apartamento 5 (izquierda) y el AP en el apartamento 11 (derecha) en la banda 5GHz*

Como se puede observar en los "*mapas de calor*" de arriba, los peores casos se situaron sobre la primera planta en la banda 5GHz, al que tenemos gran interés al ser mucho mas "nueva". Se puede apreciar que el punto de acceso colocado en cada lado da una señal suficiente para el apartamento colocado en dicho lado (arriba para el 5, abajo para el 11). Pero en cambio en el otro apartamento la señal es casi nula o nula. Como la prueba solo se hizo en este módulo, nos llevó a pensar

que en el otro módulo, al haber tres apartamentos en la planta baja y otros tres en la segunda (y en la primera hallarse la planta de abajo de los dúplex), esta situación haría difícil el uso de la banda de 5GHz porque, usando este mismo ejemplo, la señal sería casi nula.



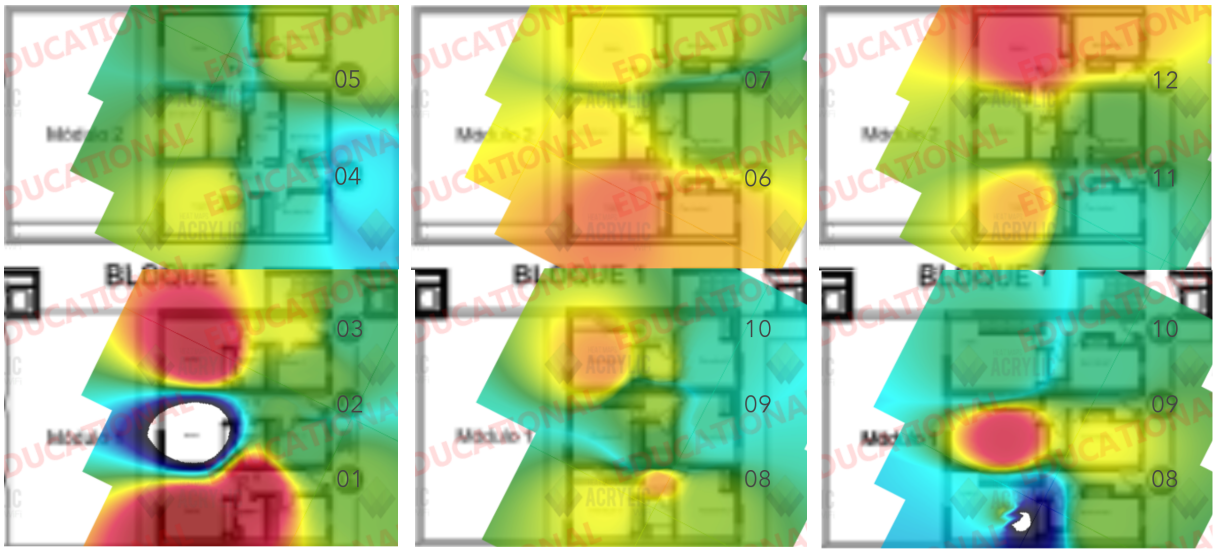
Dos gráficos que representan la distribución de los APs sobre un bloque entero, visto desde delante y desde arriba de un bloque - Antonio Ángel Cruzado Castillo

Por tanto, tuvimos que pensar en otra forma de distribuir los puntos de acceso entre los apartamentos y dependiendo del módulo (o mitad de un bloque). Para la mitad derecha (la que aparece en las pruebas anteriores) fue rápido la distribución, con la siguiente idea de Antonio: planta baja en el apartamento de la derecha, planta primera en el apartamento de la izquierda, planta segunda en el apartamento de la derecha; todos colocados en la pared en lugar del techo. Viendo que los resultados en el techo para esta mitad eran bastante buenos, la colocación de esta forma ayudaría a dar mejor cobertura en esta mitad. La distribución seguía la idea de colocar los APs en diagonal.

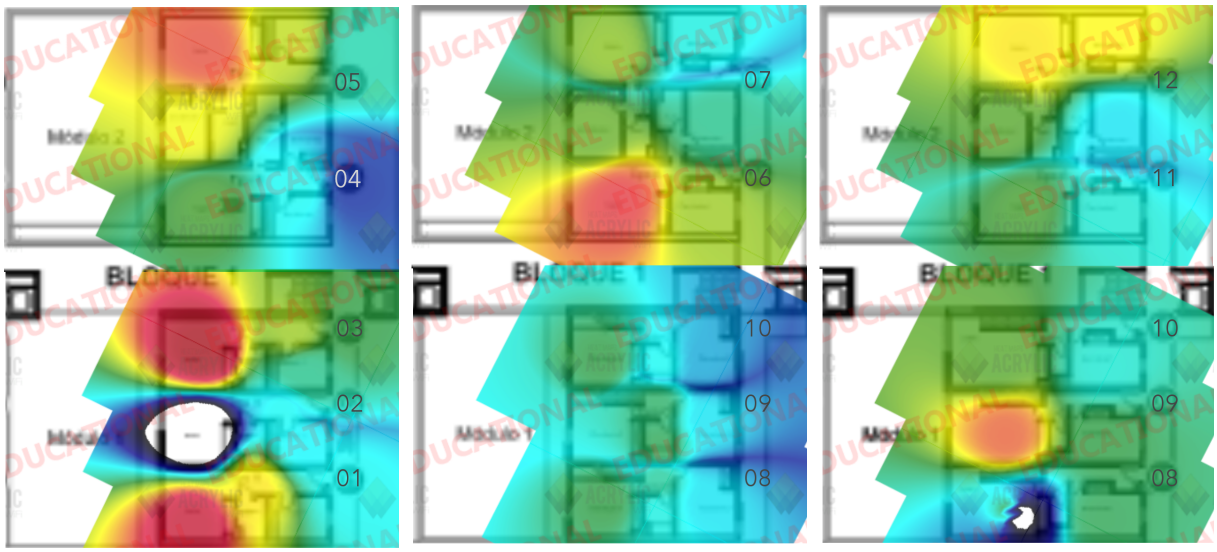
Pero para la otra mitad es más difícil. Primero, como ya se ha comentado, solo hay 6 apartamentos (tres en la planta baja, tres en la segunda planta) y la primera planta forma parte de los apartamentos de la segunda (dúplex). Por lo que en esta planta no se encuentra disponible un cable de internet. Además, los techos y suelos son de hormigón, por lo que hacer un simple agujero no es tarea fácil. Por lo que había que pensar como distribuir los APs, partiendo de que la distribución anterior (para la mitad derecha) no podría aplicarse en esta mitad izquierda y que había que ofrecer una mejor cobertura para la banda de 5GHz.

Finalmente, después de varias ideas, Antonio se le ocurrió una distribución que prometía: colocar dos APs en la planta baja y en el techo, uno en el de la izquierda y

otro en el de la derecha, y luego el tercer AP en la planta segunda en el apartamento de la segunda planta de en medio.
Esta distribución fue validada usando tres puntos de acceso.



Heatmaps *para la banda 2,4GHz, usando la distribución descrita - (izquierda) planta baja, (centro) planta primera, (derecha) planta segunda*



Heatmaps *para la banda 5GHz, usando la distribución descrita - (izquierda) planta baja, (centro) planta primera, (derecha) planta segunda*

Por tanto, con esta distribución se hicieron pruebas de calidad de señal (*heatmaps*) para demostrar si esta distribución funcionaria o no.
Quitando el problema de el circulo de no señal (un error de software), podemos ver que la calidad de la señal había mejorado enormemente para esta banda de 2,4GHz, y debería dar suficiente para todos los residentes.

Para esta banda de los 5GHz (y obviando de nuevo el agujero de no señal comentado antes), ofrecía una mejor cobertura por los apartamentos, aunque también sabíamos que para la zona de la planta primera de la mitad izquierda, la señal ahi de 5GHz era baja y es posible que no funcionara, pero habiendo señal de 2,4GHz no podría ser un problema enorme.
Los resultados obtenidos en las pruebas fueron proporcionados al Servicio Central de Informática, el cual nos dio su aprobación para realizar el despliegue de esta forma. Además, ya sabiendo la cantidad de APs necesarios, se podría conocer la cantidad de cable nuevo que iba a ser necesario añadir para éstos.

Despliegue de nuevo cableado para el WiFi

Para que estos nuevos puntos de acceso pudieran funcionar, se requería pasar nuevo cable sobre la instalación actual para que no interfiriera con el despliegue de telefonía VoIP. Por lo que también se ha planificado cuanto cable haría falta aproximadamente y por donde hacerlo pasar.

Lo primero que se hizo fue calcular aproximadamente cuántos metros de cable extra iba a hacer falta sabiendo que tenían que ir de las salas donde se ubicaban los switches hasta los apartamentos. Por lo que, con la ayuda de los planos de la residencia y tomando algunas medidas en los apartamentos, llegamos a la siguiente tabla, resumida por Antonio Ángel:

Apart.	Metros	Apart.	Metros	Apart.	Metros	Apart.	Metros	Apart.	Metros
1	40,3	21	52,6	41	75,1	67	51,4	63	62,6
3	31,8	24	50,3	42	71,15	69	42,9	66	60,3
5	33,5	25	61,4	45	83,1	71	44,6	49	81,9
6	29,55	27	52,9	48	80,8	72	40,65	51	73,4
9	41,5	29	54,6	79	70,9	75	52,6	53	83,1
12	39,2	30	50,65	81	62,4	78	50,3		
13	51,4	33	62,6	83	64,1	55	61,4		
15	42,9	36	60,3	84	60,15	57	52,9		
17	44,6	37	81,9	87	72,1	59	54,6		

Apart.	Metros	Apart.	Metros	Apart.	Metros	Apart.	Metros	Apart.	Metros
18	40,65	39	73,4	90	69,8	60	50,65		

Tabla que describe los metros necesarios de cable para cada apartamento - Antonio Ángel Cruzado Castillo, 14 de julio 2017

En total, harían falta 2750m aproximadamente.

Para algunos apartamentos, el cable tenia que “viajar” por la pared al descubierto para llegar de la salida del cable de los tubos hasta el AP. Por lo que se usarían canaletas de PVC para proteger el cable.

El despliegue efectivo de tanto estos cables nuevos, como de los APs y los switches corría a cargo de la Universidad que decidió formar un grupo de voluntarios que dedicaron Julio entero y la primera semana de Septiembre para realizar toda la instalación, la parte física del trabajo.

Configuración de la red WiFi

En este apartado, se especificará cual es la configuración final de la red WiFi. Algunos de las configuraciones se han basado en una guía de Unifi[59].

La red WiFi es la que mas dispositivos va a tener conectados simultáneamente, por lo que debe prepararse para tal. Para ello, lo mas importante es tener despejado lo máximo posible los canales de emisión de los puntos de acceso. Un truco que viene perfecto para ello es bloquear el tráfico *multicast* y *broadcast* de la red WiFi a la LAN y viceversa. Este tipo de tráfico suele saturar la red WiFi cuando hay muchos dispositivos, ya que todos ellos se comunican entre ellos y envían información para todos, y al ser la mayoría paquetes pequeños, acaban haciendo muchas interferencias y ralentizando la red entera[18].

Otros elementos que ayudan a mejorar la red WiFi es aplicar aislamiento entre dispositivos, que se aplica a nivel de switch. Esto evita tráfico innecesario entre los dispositivos conectados al WiFi, reduciendo el tráfico en la red inalámbrica. Este

sistema también se está usando en la red Universitaria, ya que es fácil comprobar que en la red dos dispositivos no se pueden ver (no funciona *ping* por ejemplo).
Para mejorar la calidad de la red, también se ha aplicado un límite de "mínimo de RSSI", que viene a ser el mínimo de potencia de señal para que el AP te permita conectarte. Para la banda de 2,4GHz está puesto a -75dBm, en cambio para 5GHz está a -80dBm. Llegamos a estas conclusiones después de probar con diversos valores, en cada banda, y ver qué resultados daban.
La potencia de señal para la banda 2,4GHz está a la mitad porque en esta banda la señal penetra más que en la 5GHz, y como se ha demostrado antes, para esta banda no hacían falta tantos APs. Además, se ha podido comprobar que una mayor potencia de señal da muchos problemas en los dispositivos. Entre estos problemas se encuentran las interferencias, o algunos dispositivos que no son capaces de cambiar a otro AP con mejor señal hasta que se desconectan de él. La única excepción a esta regla es el punto de acceso del salón de estudios, que al estar más alejado de todo, no sufre de tantas interferencias, y ayuda a los usuarios a tener una mejor conexión.
También descubrimos que los APs de Unifi soportan una opción que ellos llaman "*Band Steering*". Esta opción "empuja" a los dispositivos a conectarse a la banda 5GHz, pero no obliga a ello. Así se intenta siempre que se use esta banda lo máximo posible.

Relacionado con el tema de los SSID (o las redes WiFi), se van a usar dos: "*eduroam*" para cualquiera relacionado con la universidad (incluyendo los residentes) y una red de invitados "*ResiInvitados*" para gente externa a la residencia o los residentes que aun no hayan configurado la red "*eduroam*".

La red "*eduroam*" permite a estudiantes, profesores, investigadores de la UMA y de otra universidad europea (que esté adherido al sistema *eduroam*) conectarse con sus credenciales de la universidad y disfrutar de conexión a internet fácilmente allá donde haya esta red. La Universidad puso especial énfasis en la implantación de esta red ya que, como se puede deducir, es bastante cómoda de manejar. Es más, actualmente la Universidad ofrece "*eduroam*" para todos estos grupos de personas, en contra del anterior sistema con varios SSID.

Para su funcionamiento, se requiere de un servidor RADIUS, que permite identificar al usuario. Este servidor lo provee la Universidad, y los puntos de accesos intentan autenticar directamente con éste. La red *eduroam* usa un sistema de autenticación por universidades, y para conectarse con otros países, se hace a través de los servidores del NREN (*National Research and Education Network*) del país correspondiente. Por tanto, nuestro servicio se conecta a un servidor RADIUS que redirecciona las peticiones al servidor correspondiente dependiendo del dominio que tenga el email del usuario.

Para los invitados, o aquellas personas que no hayan configurado aún la red *eduroam*, se implementará un portal cautivo en esta red para poder identificar a cada uno de los usuarios y limitar el uso que hacen de la red.

Todo el tráfico de la red WiFi va por la VLAN 10, que se explicó en "Diseño y configuración de la red fija".

3. Requisitos del sistema software de gestión

Análisis

Vivimos en un mundo donde todo está en internet, que nos ofrece la comodidad de poder hacer o ver cosas desde tu ordenador o móvil, centralizado en nuestros dispositivos. Para la residencia, poder ofrecer un conjunto de servicios y que se puedan ver desde tu apartamento es posible gracias a la nueva instalación de cable e inalámbrica, y las facilidades que ofrecen los contenedores. Además, agrupar diversos servicios, inicialmente separados, en un solo lugar ayuda a visibilizarlos. Y poder informatizar algún servicio que actualmente no lo es, permite simplificar las gestiones de ellas y dar comodidad de uso.

Actualmente tenemos datos de uso de tráfico por dispositivo, un servicio de partes informáticos, el blog de temas relacionados con la residencia, o nuevos datos como los que ofrece el servicio de telefonía ahora son elementos que podrían estar presentes en dicho software que pueden interesar a los residentes. Tenerlos en un mismo lugar puede facilitar a los usuarios encontrar todo esto y poder ver esa información en cualquier lugar, sea en la residencia o fuera.

Además, este software no debería ser cerrado, si en el futuro se quiere extender, debería poderse extender sin mucha dificultad. Para este trabajo no se va a poder abarcar muchas de las ideas que puedan surgir, este punto es importante para que los futuros estudiantes que trabajen en el área de informática de la residencia puedan ampliarlo con las ideas no contempladas en dicho trabajo o con sus propias ideas.

Metodología del desarrollo software

Para esta fase del trabajo, se usará una metodología basada en Scrum. La toma de requisitos se realizará mediante obtención de lo que se denominan "*User Stories*", que pueden ser uno o varios requisitos de las metodologías mas clásicas. De los requisitos, se obtendrá un modelo para la base de datos que servirá para el *backend* que realiza Antonio para almacenar información. Con el modelo realizado, se definirá un conjunto de *endpoints* de la API para que no haya discrepancias entre

el *backend* y el *frontend*, que realizo yo. Esos *endpoints* son las rutas disponibles en el servidor que permiten realizar el intercambio de información y realizar acciones entre el *backend* y el *frontend*.
El proceso de desarrollo del servidor web se podrá encontrar en la sección «*Desarrollo del backend de la apliación "Panel del residente"*» del trabajo de Antonio Ángel. El trabajo del *frontend*, en cambio, se encontrará en la siguiente sección. De esta forma, cada uno trabajará independientemente en cada una de sus tareas, pero podrá dar opinión sobre las *historias de usuario* o la especificación de los *endpoints* en cualquier momento.
En la ultima fase, habrá que hacer un proceso de pruebas manuales para comprobar que todo funciona correctamente y que la integración entre ambos componentes es correcta.

Requisitos

En este apartado, se expondrán los requisitos recabados para esta aplicación web. Al seguir una metodología Scrum, los requisitos serán “historias de usuario”, que son puntos más genéricos, que incluyen más información sobre lo que se pide, pero permite poder modificarlos o extenderlos más fácilmente que los requisitos clásicos. Estos requisitos están muy bien estructurados y definidos, pero que en muchas ocasiones realizar un cambio sobre ellos conlleva a hacer una reestructuración de algunos de ellos, tal y como hemos comprobado en los trabajos de los estudios de la carrera.

Desde la primera versión hasta la final, han habido diversos cambios y mejoras en estas historias de usuario, siguiendo la agilidad de la metodología SCRUM, que han permitido obtener el siguiente estado de ellos. Estos son los “requisitos” finales de la aplicación:

- **Aspectos generales de la aplicación web**
  - La aplicación web debe ser modular, permitiendo su extensión fácilmente.
  - La aplicación debe poder usarse desde una navegador web en cualquier dispositivo.

- La aplicación está perfectamente en dos partes, se debe garantizar la interoperabilidad entre ambas mediante el estilo arquitectónico REST y usando el formato de intercambio de datos JSON:
  - La parte visual - Interfaz de Usuario - con la lógica de control de la interfaz, a la que llamaremos frontend.
  - La parte de lógica de negocio y base de datos, con las operaciones específicas de ésta, a la que llamaremos backend.

- **Permisos de la aplicación**
  - En la aplicación se distinguen varios roles, cada uno de ellos determinará las acciones que el usuario puede llevar a cabo en esta. Los roles principales serán:
    - Administrador
    - Residente
    - Personal de la UMA no residente
    - Personal de administración
  - Los permisos se distinguirán cada módulo, dentro de un módulo habrá páginas y dentro de cada página habrá uno o varios permisos. Por tanto, se dirá que un rol tendrá X permiso sobre el módulo A y la página B.
  - La visibilidad de los módulos y los distintos apartados dentro de un módulo dependiendo de los permisos que tenga el usuario debe poderse obtener mediante la API.
- **Módulo de consumo de internet**
  - La aplicación permitirá ver el consumo de internet de cada uno de los dispositivos ligados a un residente. Cada usuario podrá identificar a la aplicación cuales son sus dispositivos entrando a este módulo con cada uno de sus dispositivos y con su cuenta. Se mostrarán gráficos de consumo de internet por cada uno de sus dispositivos identificados e información relacionada a cada uno, como consumo agregado, dirección IP, etc.
  - Los administradores pueden ver todos los dispositivos asociados, para ver información sobre ellos o realizar acciones sobre ellos.
  - Si un usuario reclama que su dispositivo, siendo suyo, no puede encontrarlo en la página, porqué alguien se ha apropiado de él, podrá notificarlo a los becarios de informática y realizar el cambio de “dueño”.

    - Sobre consumo agregado, se podrá ver dicho consumo en un rango de fechas, que va de 1 día hasta 1 semana, con una precisión de 1h hasta 1 día, o de 1 día hasta 1 mes con precisión de 1 día.
    - Para los datos en tiempo real, el rango oscilará entre dos valores X e Y. X será del momento actual hasta 1 día, 23 horas y 59 minutos antes; el valor de Y será de 1h antes hasta 2 días. La diferencia en el rango debe ser mínimo de 1 minuto. La precisión será de 1s hasta 1 hora.
- **Módulo de resumen para el WordPress**
    - Una sección que muestre un resumen de las últimas noticias del blog, cargando el RSS de éste, y mostrando un enlace para acceder a él. También se mostraría los métodos para mantenerse informado.
- **Módulo de historial de llamadas**
    - Un usuario residente será capaz de ver el historial de llamadas de su apartamento.
    - El personal de administración debe ser capaz de ver el historial de llamadas de cualquier apartamento.
    - En cualquier caso, debe haber un filtrado por fecha y número de teléfono del otro extremo.
- **Módulo de incidencias informáticas**
    - La notificación y gestión de incidencias informáticas debe hacerse a través de la aplicación. Existirá un formulario que un usuario del sistema pueda rellenar para notificar la avería o el problema, o simplemente hacer una consulta informática. Los administradores pueden ir actualizando el estado de la incidencia y los usuarios, hacer su seguimiento.
    - En la página de una incidencia, los administradores serán capaces de modificar el estado del mismo. Al realizar un cambio de estado, deben escribir un mensaje que explique el motivo.
    - El usuario puede descartar en cualquier momento una incidencia, explicando la razón de tal descarte.
    - Además, los administradores pueden ver cuales son las incidencias activas y resueltas.
    - Los estados que puede tener una incidencia son:
        - Pendiente: *nueva incidencia, esperando a que algún becario lo vea*

    - Esperando: *incidencia lista para ser arreglada, pero no pueden ir aún (no pueden ir por no ser la hora de disponibilidad o no están disponibles ningún becario)*
    - Trabajando: *algún becario está trabajando en ello*
    - Esperando a usuario: *se espera alguna respuesta del usuario por algún motivo*
    - Resuelto: *la incidencia se ha resuelto*
    - Descartado: *el usuario ha descartado la incidencia*
    - No se arreglará: *por algún motivo (razonable), la incidencia no se va a arreglar*
- **Notificaciones y módulo de notificaciones**
  - La aplicación enviará notificaciones a los usuarios antes distintos eventos que ocurran en el sistema. Esta contará con distintos métodos de notificación:
    - Correo institucional UMA, método de notificación por defecto.
    - Bot de telegram, el bot mandará a los usuarios con los que mantiene conversación las notificaciones oportunas
  - Cada usuario puede seleccionar qué medios de notificación desea, y tener varios (o ninguno activo). Como mínimo se tendrán las notificaciones in-app.
  - Dentro de la aplicación habrá un “centro de notificaciones”, desde donde se pueden consultar todas las notificaciones (pendientes, vistas…).
  - Una notificación ya vista será borrada al cabo de un tiempo prudente. (1-3 meses).
- **Gestión de la sesión**
  - Cualquier usuario de la aplicación a excepción del personal de administración, debe identificarse en la aplicación usando sus credenciales de iDUMA. El personal de administración empleará unas credenciales para la autentificación usando un directorio propio.
  - La sesión se mandará en cada petición en la cabecera HTTP en la forma de token JWT. La validez de un JWT será de 2 días.
    - En una respuesta HTTP se mandará una cabecera de extensión cuando el token esté a punto de expirar, esto es, cuando le queden 1h.

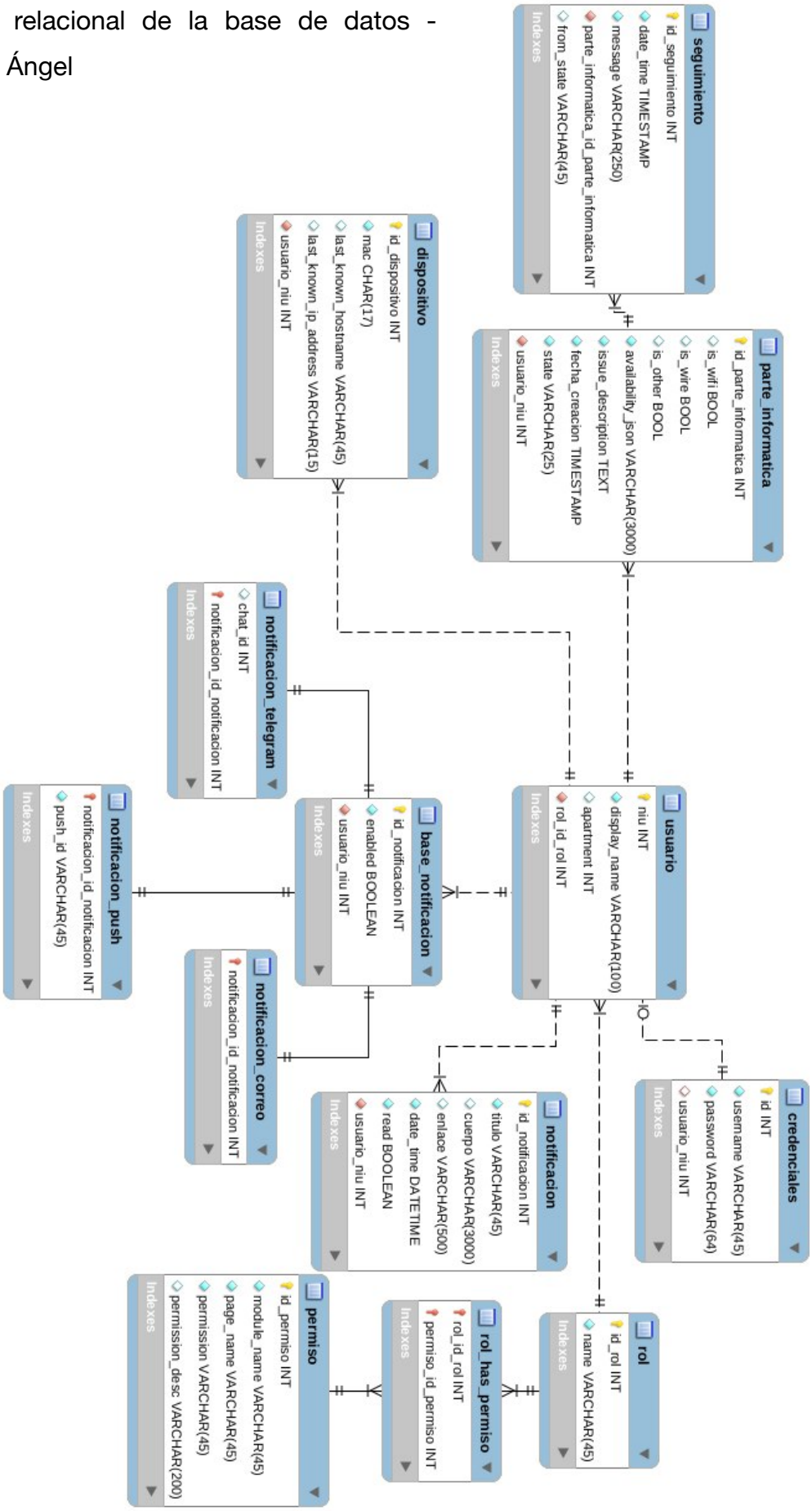
  - Se usará una cabecera HTTP con valor a 1, y aconseja al cliente que pida un token nuevo en una ruta específica (descrita más abajo) y empleando el token que tiene aún válido.
  - La ruta será */renew_token*, y a esta se le hará simplemente una petición GET (con el campo de la cabecera *Authorization: Bearer JWT_TOKEN*) y la respuesta será un JSON: *{ token: NEW_TOKEN }*
  - Habrá una ruta disponible que le permitirá renovar el token al cliente, empleando *access token* para entonces aún válido. El token nuevo que se proporcione, tendrá una validez de 2 horas desde el momento de su expedición. Si el token ya ha sido renovado, el aviso de expiración se hará cuando a este le queda 30min en el caso del token de 2h.
- El mecanismo de persistencia del token JWT en el cliente será empleando la API local storage/session storage del navegador web.
  - local storage permite mantener la sesión incluso si cierra la pestaña o el navegador
  - session storage, en cambio, cierra la sesión al cerrar las pestaña de la web
  - Permite implementar el mecanismo de “recuerdame”
- Si un cliente realiza una petición al servidor para realizar una acción, pero el usuario que lo realiza no tiene permisos, debe fallar la petición indicando que no tiene permisos para realizar dicha acción.
- En el frontend, si intenta acceder a una página que no tiene permisos, aparecerá que la página no existe.

Modelo

Para este proyecto, el modelo de datos debe poder permitir almacenar toda la información descrita anteriormente en una base de datos. Pero, los datos de consumo y llamadas telefónicas se encuentran disponibles en otras base de datos, a las que la aplicación se conectará para obtener esa información.

El modelo siguiente solo representará los datos que manejará directamente la aplicación web, otros datos como el historial de llamadas o consumos de internet se encuentran en otras bases de datos.

Modelo relacional de la base de datos - Antonio Ángel



El modelo lo ha realizado Antonio Ángel, aunque el modelo inicial fue un trabajo en equipo.
El centro del modelo es la entidad *Usuario*. Esta entidad almacena la suficiente información del usuario para que pueda usar la aplicación. El NIU, su nombre y, si es residente, el número de apartamento son los datos necesarios. Para los usuarios de la administración de la residencia, al no ser personal de la universidad, se debe ofrecer un inicio de sesión alternativo. Eso está recogido en la entidad *Credenciales,* cuyo NIU será uno cualquiera que no interfiera con los de la Universidad (como por ejemplo números bajos como el 1, 2, 3…).

Los permisos están recogidos en las entidades *Rol* y *Permiso* (con la relación de muchos a muchos representada por la entidad *rol_has_permiso*). Por tanto un rol determinado tendrá un conjunto de permisos, y esos permisos se pueden ir repitiendo entre los diversos roles. A su vez, cada *Usuario* está relacionado con el rol al que pertenece. Los permisos contienen tres claves que permiten determinar a donde pertenece el permiso y cual es. Los atributos *module_name* y *page_name* identifican donde pertenece el permiso *permission*. El módulo identifica un grupo de datos y acciones relacionados, como puede ser los partes de informática. La página, que está más relacionado con el frontend, permite poder entrar en una o varias secciones del módulo.

Sobre notificaciones, estas se almacenan en la base de datos en la entidad *Notificacion*. En ella se almacenan el título, la fecha y si se ha leído o no, además de a quién va dirigido la notificación. Opcionalmente se puede almacenar un cuerpo, con contenido que describa más la notificación, y un enlace, en caso de que se conozca el origen de la notificación.

El sistema de notificaciones tiene diversos canales para distribuirlas, como se ha comentado ya. Para configurarlas, se han creado cuatro entidades de las cuales tres “heredan” (usando el mismo concepto de herencia de Orientación a Objetos) de una. Esta super-entidad es *base_notificación* que relaciona un usuario con una configuración, además de indicar si está activo o no ese canal. De esta, sale una configuración para Telegram, otro para los Correos y una última para notificaciones

PUSH (aunque no se usen, el sistema esta preparado para añadir este nuevo canal). Para el canal de correos electrónicos no se almacena ningún dato adicional, pero para poderlo identificar dentro del *backend*, se debe crear dicha entidad. Para Telegram, hay que almacenar el *chat_id* (o *user_id*, que son el mismo identificador) para poder enviar mensajes directos al usuario.

En el sistema de partes informáticos también se debe modelar, y lo encontramos con las entidades *ParteInformatica* y *Seguimiento*. El primero, *ParteInformatica*, almacena la información necesaria para un parte. Contiene el tipo de parte (*is_wifi, is_wire* y *is_other*) donde se puede seleccionar más de un tipo a la vez. También incluimos una descripción del problema o consulta, que es obligatoria, la fecha de creación, el estado en el que se encuentra el parte y una disponibilidad en forma de horarios almacenados como json (*availability_json*). Este último permite al usuario que crea el parte dejar anotado su disponibilidad horaria para que se pueda resolver lo antes posible. El formato del JSON se especificará en la API. Por último, cada parte está relacionado con un usuario, que es su creador. El *seguimiento* sirve para ver el historial de modificaciones del estado del parte, y poder visualizar los cambios del mismo. En un seguimiento, almacenamos la fecha de cuando se realizó esa modificación, un mensaje que indica el motivo del cambio, y, si hay un cambio de estado, se almacena también el estado en el que se encontraba el parte antes de modificar. De esta forma, se puede observar el flujo de transiciones de estado del parte.

La última entidad es *Dispositivo.* Esta entidad permite relacionar un usuario con un dispositivo de la residencia. Permite ofrecer la utilidad de ver los consumos de internet al saber cuales son los dispositivos del usuario. Lo importante de la entidad es la MAC (o dirección MAC) que es la que permite la identificación. Los otros dos atributos sirven como caché, para acceso rápido de esos datos.

Descripción de la API

La base común entre el *backend* y el *frontend* es la API que expone el primero que consumirá el segundo. Se describirá cual es esta especificación, que

funcionamiento tiene cada uno de los *endpoints* de la API y el proceso para llegar hasta el resultado final.

La especificación se ha realizado siguiendo el estándar OpenAPI Specification 2.0/ swagger[55]. Permite describir APIs REST mediante un fichero YAML, fácilmente entendible para las máquinas y humanos mediante generadores de código y documentación (respectivamente). De esta forma, se podrá especificar la API del *backend* y se podrá revisar y mejorar sin tener que implementar nada. La especificación en *yaml* la pueden encontrar en el archivo adjunto *swagger.yaml,* y, subiendo el fichero a https://editor.swagger.io, se puede visualizar la documentación que genera dicha especificación de forma interactiva.

Esta especificación se ha basado en el modelo de la base de datos y en la especificación de las historias de usuario. Desde la primera versión, ha ido recibiendo cambios y mejoras para obtener el resultado final. Y algunos de esos cambios han repercutido en un cambio en las historias de usuarios.
La primera especificación la realizó Antonio Ángel al completo, y sobre eso, se han ido haciendo iteraciones para ir mejorando la API. Alguno de los dos observábamos mejoras que se podían realizar o fallos en la especificación y luego se discutían como aplicar esas modificaciones. Para mantener claro todas los cambios que se habían de aplicar, usamos una aplicación web llamada Trello que permite definir columnas y tarjetas, y dichas tarjetas pueden contener una descripción y comentarios. De esta forma, hemos podido seguir todos esos cambios y realizar comentarios sobre ellos. Prácticas típicas de metodologías ágiles como la que aplicamos para este trabajo.

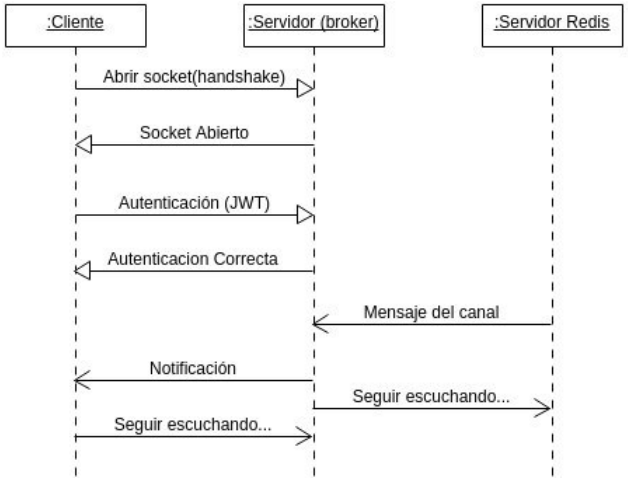


Diagrama de secuencia del WebSocket - Antonio Ángel

De la especificación de OpenAPI 2.0, hay un elemento que no está detallado al ser un WebSocket. Su ruta es */notifications/ws*, y permite

en tiempo real enviar al navegador una notificación justo en el momento en el que se crea. Para ello, el cliente debe conectarse al WebSocket y enviar el token de acceso en JSON con la siguiente estructura: *{"token": “<TOKEN_DE_SESION>”}*. Una vez que la autenticación es correcta, el canal queda abierto listo para recibir cualquier mensaje de nueva notificación.

En el diagrama se observa también un actor “servidor redis” que es el que el *backend* usa para poder saber cuando se ha creado una nueva notificación y, a su vez, permite enviarla al destinatario correspondiente de dicha notificación. En el trabajo de Antonio Ángel (sección *5.2.4. Sistema de notificaciones en tiempo real*) se pueden ver más detalles sobre este “servidor redis” y el sistema de notificaciones.

4. Desarrollo del frontend del software de gestión

En esta sección se describirá el proceso de desarrollo del frontend, las tecnologías usadas para ello, la separación de los módulos distintos, las decisiones y los problemas en el desarrollo y qué hacer para poder desarrollar si alguien desea continuar el trabajo.

Los módulos que se van a realizar, obtenidos a partir de los requisitos, son los siguientes:

- Página inicial - *para iniciar sesión*
- Página resumen del blog en WordPress
- Módulo para consumo de internet
- Módulo para historial de llamadas
- Módulo para partes informáticos
- Módulo para notificaciones

Tecnologías, modularidad y consideraciones

Para el desarrollo del *frontend* se usará las siguientes tecnologías:

- React.js[5]: *biblioteca JavaScript para el desarrollo de interfaces web*
- redux[6]: *biblioteca que permite definir comportamiento predecible independiente del lugar de uso*
- bootstrap[7]: *framework para diseño web que define componentes de uso común*
- SASS/SCSS[50]: *lenguaje que simplifica diseñar en CSS con algunos extras*
- TypeScript[4]: *el lenguaje de programación que se usará, que es JavaScript con tipado*
- create-react-app, permite crear una plantilla para trabajo con react.js y herramientas para el desarrollo sencillo en él. Se usará una modificación de create-react-app-typescript[46] para añadir redux y bootstrap, con algunas dependencias extra y poner un código base común a la aplicación

El motivo de la elección de react.js con redux es la simplicidad que ofrecen ambas tecnologías juntas de mantener un estado de la aplicación (redux) y la representación visual (react.js) separadas. También facilita la modificación de

elementos de la interfaz de forma casi transparente y ayuda mucho poder tener componentes reutilizables (al igual que hace el framework Angular[47]) pero de forma mucho más ligera (en contraposición a Angular). Después de haber trabajado con react.js + redux y Angular, opino que el rendimiento que da el primero es mejor que el segundo, aunque sea más difícil de aprender que Angular (ya que sigue el patrón Modelo-Vista-Controlador). En algunas ocasiones, creo que react.js es más fácil que Angular por tener mezclado la representación (HTML) con parte de la lógica de la interfaz, donde Angular se tiene separado la plantilla que genera la representación (HTML) con el código de la lógica. Y los componentes nuevos son más fáciles de añadir en react.js que en Angular, en mi experiencia.
El motivo de elegir TypeScript en lugar de JavaScript puro es el de poder evitar problemas de tipos, típicos de JavaScript, y al usarlo en un IDE como WebStorm[48] o Visual Studio Code[49], te pueda ayudar a auto completar y con sugerencias de código que aceleran el desarrollo. Además, TypeScript es una extensión de JavaScript, todo código de JavaScript, con mínimas modificaciones (generalmente añadir tipos), funciona en TypeScript. Además un código JavaScript se puede integrar con uno TypeScript definiendo un archivo de tipos. Dicho archivo permite a TypeScript entender los tipos de clases, métodos y funciones. Al final, es una ayuda al desarrollo que no supone una sobrecarga grande al archivo final.

Para el diseño de las páginas, bootstrap es una gran elección. Llevo usando desde muchos años este *framework*, y creo que es una buena ayuda para mantener un diseño moderno, independiente al tamaño de pantalla y fácilmente extensible. Además bootstrap desarrolla su código en SASS, por lo que las utilidades de código que ellos ofrecen se pueden aprovechar para el diseño de la web, usándolos en nuestro código SASS. Esto último se refiere a que SASS permite la definición y uso de funciones para definir estilos de CSS, por lo que las utilidades que definen en dicho framework se pueden usar para el diseño del *frontend*.

create-react-app es una aplicación que permite instalar una base de configuraciones y código inicial para empezar a desarrollar. Para ello, usa un paquete que contiene esa configuración, que para usar TypeScript hay que usar create-react-app-typescript. Pero estos paquetes solo instalan los paquetes básicos

para el desarrollo, otras dependencias como redux o bootstrap hay que instalarlas por separado. Para no tener que repetir el mismo proceso cada vez que se quiera crear una nueva página con estas tecnologías, se ha realizado una modificación de la versión con TypeScript para añadirle el código inicial preparado para nuestras necesidades, y las dependencias que se van a usar.

El *frontend* dividirá cada módulo en un proyecto distinto para mantener la modularidad. De esta forma, un cambio en una página de un modulo, solo requerirá *recompilar* dicho módulo. Dentro de cada módulo, a parte del código y dependencias comunes, se dispondrá de un módulo común con componentes reutilizables y el estilo base de la aplicación. Dicho módulo se encuentra en un repositorio separado y se importa en cada proyecto como un submódulo[54] de git (dentro de un repositorio, se importa otro repositorio en una carpeta). El único punto en contra del módulo común es que está integrado en cada uno de los módulos, por lo que dicho código estará duplicado en cada módulo, y actualizar el módulo común implica tener que recompilar todos los módulos.
Para esta fase inicial de desarrollo, lo importante es dejar el módulo común lo más completo posible para evitar tener que modificar mucho dicho módulo en el futuro. En el desarrollo, se ha usado otro método para compartir dicho modulo común entre los repositorios: *bind mount*[51]. Disponible en sistemas operativos basados o similares a Unix (como Linux[52] o macOS[53]), permite “montar” (como si fuera un USB externo) una carpeta existente en otra ubicación. Si modificas la carpeta original, los puntos de montaje se verán modificados también, y viceversa. De esta forma, es más cómodo poder ir actualizando ese código en desarrollo sin tener que usar control de versiones e ir replicando ese cambio a todos.

Cada módulo será encargado de implementar las restricciones de permisos que existen, aunque el servidor de la API también hará esas comprobaciones. De esta forma, el usuario podrá notar de que no puede acceder a dicha página por algún motivo. La forma en el que la web mostrará la falta de permisos sobre una página o módulo se hará mostrando el error de “página no encontrada”. Además, dichas páginas del módulo no se cargarán en el navegador para evitar que un usuario pueda ver el código de estas páginas, o que algún código se ejecute aun no

teniendo permisos. Para ello, se usará una funcionalidad incluida en create-react-app llamada "*code splitting*" que permite separar elementos en diversos archivos JavaScript a la hora del despliegue y cargarlos bajo demanda.

La sesión se obtiene mediante un token que envía el *backend* al acabar con el proceso de inicio de sesión. Dicho token se debe almacenar en el lugar correspondiente del navegador (*localStorage* o *sessionStorage*) dependiendo de si quiere mantener la sesión con la opción de "recordarme" o no (respectivamente). Además, se comprobará la validez del token, y si hiciera falta, se renovará mediante el método de renovación del token. Si el token ha caducado, el navegador irá de nuevo a la página de inicio para que inicie sesión, previo aviso al usuario.

Hay dos métodos para iniciar sesión, ambos terminan de la misma forma, descrita en el anterior párrafo. El primero es el inicio de sesión a través de la Universidad, que para ello, se redirige el navegador al módulo SAML y éste se encarga de terminar el proceso, pasando por el backend que generará el token. El segundo método es mediante el tradicional usuario/contraseña, siendo un método alternativo para la administración de la Residencia.

Descripción de la interfaz de los módulos

En esta sección se describirá brevemente el funcionamiento de cada una de las páginas de cada módulo, con capturas de las paginas. Por cada módulo, se mostrarán las capturas primero y luego se describirá.

‣ Módulo común

Este modulo contiene el diseño base de la aplicación web y prepara un mecanismo sencillo para poder crear el resto de módulos con un sistema de páginas.

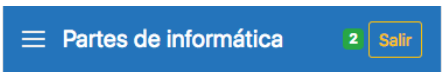
Tiene el manejo de la sesión. Al entrar a un módulo, comprueba si la sesión es válida y qué permisos tiene sobre dicho módulo. En el caso de que la sesión sea inválida o no exista, se encarga de llevar el navegador a la página principal (donde un usuario puede iniciar sesión). También se encarga de renovar el token de la sesión si la API envía la cabecera que indica que debería renovar el token.

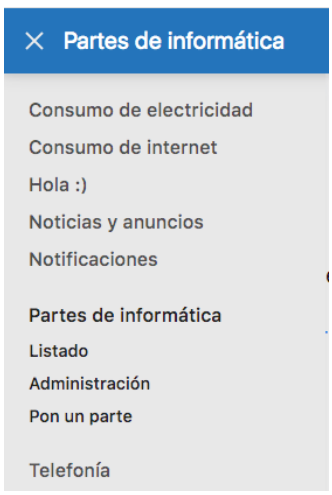
Para React, se han definido algunos componentes que se pueden reutilizar tantas veces como sea necesario. Algunos son componentes de Bootstrap convertidos para su fácil uso en React, como botones[56], modal[57] o collapse[58].

Para comunicarse con la API del *backend*, se ha definido una pequeña biblioteca que simplifica la comunicación con éste. El uso de estos métodos dentro de una acción de redux (código que permite modificar el estado del módulo) debe meterse dentro de un decorador (del patrón de diseño *decorador*) que captura los errores y los almacena en redux, de esta forma se evita repetir código para errores, y cualquier componente puede conocer que ha habido un error usando siempre el mismo mecanismo (ver el estado de los errores).
Este módulo, además, escucha a las nuevas notificaciones que puedan aparecer mientras el usuario usa la aplicación. Ya que es un comportamiento que se va a compartir entre todos los módulos, se ha decidido colocarlo en este componente común. Al recibir una nueva notificación, la mostrará en una esquina del navegador (como una notificación), a la vez que incrementará el contador de notificaciones no leídas (se encuentra en la parte superior de una página, a la derecha).

Sobre esta cabecera, el botón de "*Salir*" permite cerrar sesión en cualquier momento. El texto de la izquierda "*Inicio*" indica el nombre del módulo en el que te encuentras. Esta cabecera siempre aparece en todos los módulos.

También se ha colocado una barra de navegación entre los distintos módulos, y el mismo módulo en el que uno se encuentra. Se encuentra a la izquierda del navegador (captura derecha abajo), y en los móviles se muestra mediante un botón que aparece al lado del texto de la izquierda de la cabecera (captura derecha arriba).



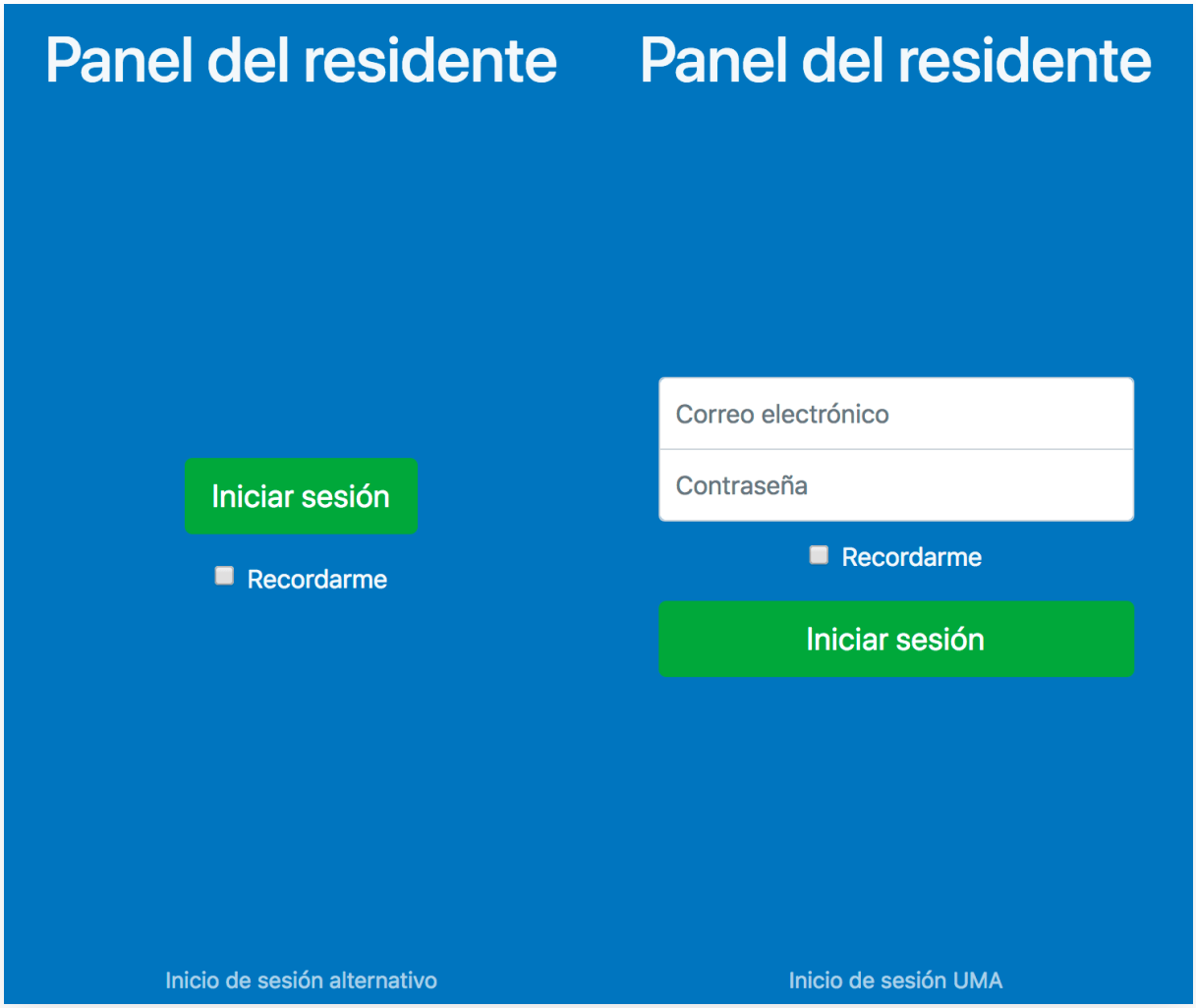


Por último, también se encarga de preparar la detección del idioma del navegador para poder mostrar los textos en

el idioma del usuario (si está disponible).

Hay un componente llamado *App* que es el encargado de realizar la mayoría de las tareas descritas anteriormente. La barra de navegación, la cabecera, las notificaciones, los estilos base (CSS), el sistema de rutas, que muestra las páginas dependiendo de la ruta del navegador, y la comprobación de permisos son tareas que realiza este componente. *App* se muestra siempre y es antecesor de cualquier página del sistema de rutas.
En el caso de que una página no exista (o el usuario no la pueda ver), la página que se muestra también esta incluida en este módulo. También, durante la comprobación de los permisos y de la sesión, si ocurre un error, se mostrará otra página que está incluida en este modulo común también.

‣ Página inicial - Página de inicio de sesión



(↑) Página de inicio de sesión, con las dos modalidades
(↓) Página inicial después del proceso de inicio de sesión

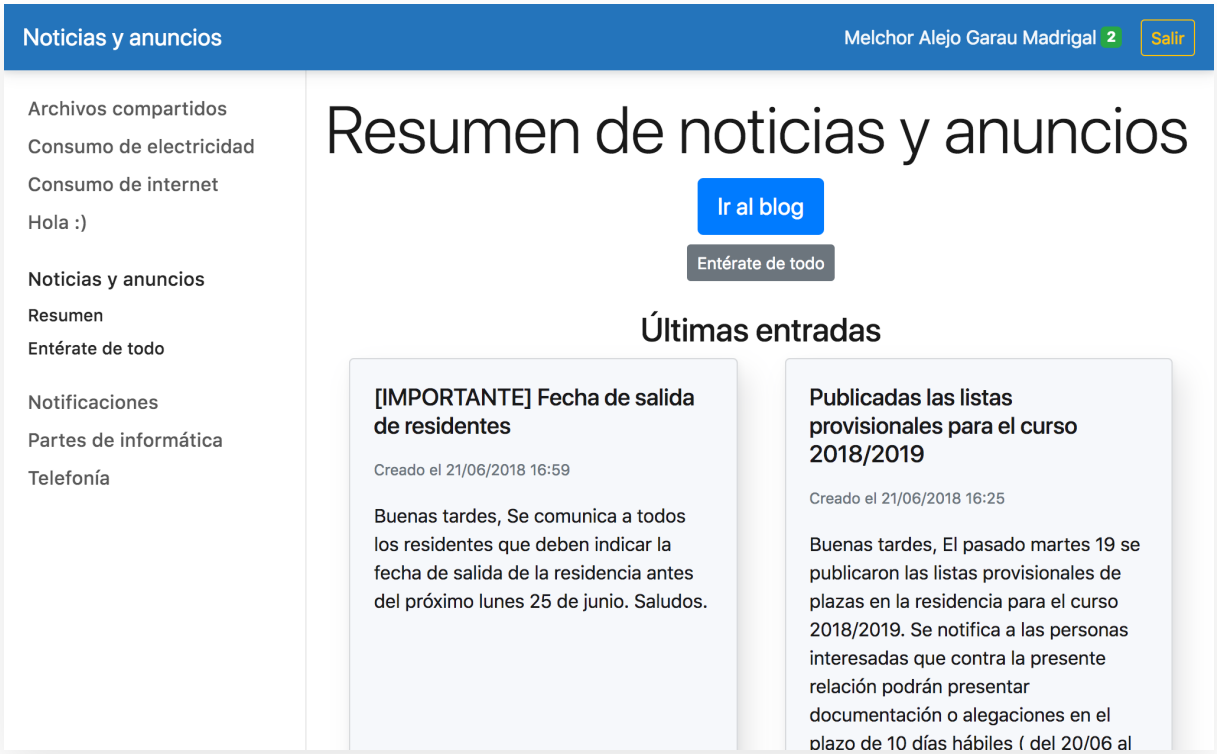
Para que los usuarios puedan iniciar sesión a través del mecanismo de autenticación SAML de la Universidad, se ha preparado un botón que permite empezar el proceso de autenticación. Con un sencillo click, redirigirá a la ruta /*alberginia* que es el que empieza todo el proceso. Al final del proceso, como se ha descrito, la aplicación web recibirá un token dentro de los parámetros de la URL, que la aplicación almacenará dentro del *localStorage* o *sessionStorage,* dependiendo de si se ha apretado el botón de recordar o no (respectivamente). En el caso de que el usuario viniera de algún otra página y haya acabado en la página de inicio de sesión por haberle caducado la sesión, se intentará redirigir a la página en la que se encontraba anteriormente.

En la página de inicio de sesión también encontramos un modo alternativo de inicio de sesión. Este modo es para la administración de la residencia, ya que no son usuarios de la Universidad, necesitan de un método alternativo de inicio de sesión. Este método es el tradicional usuario y contraseña, con las mismas funcionalidades que el anterior método, aunque el token de la sesión se obtiene en la misma respuesta de comprobación de las credenciales. Esa comprobación se realiza directamente al *backend* de la aplicación web.

El botón de abajo de ambas capturas permite cambiar entre ambos métodos de inicio de sesión, que se realiza al momento.

Si hay ya un usuario que haya iniciado sesión, esta página se convierte en una especie de escritorio con enlaces a los diversos módulos y páginas donde el usuario tiene permisos para acceder, además de un módulo extra con enlaces rápidos a páginas de la web de la Universidad que pueden ser útiles a los usuarios.

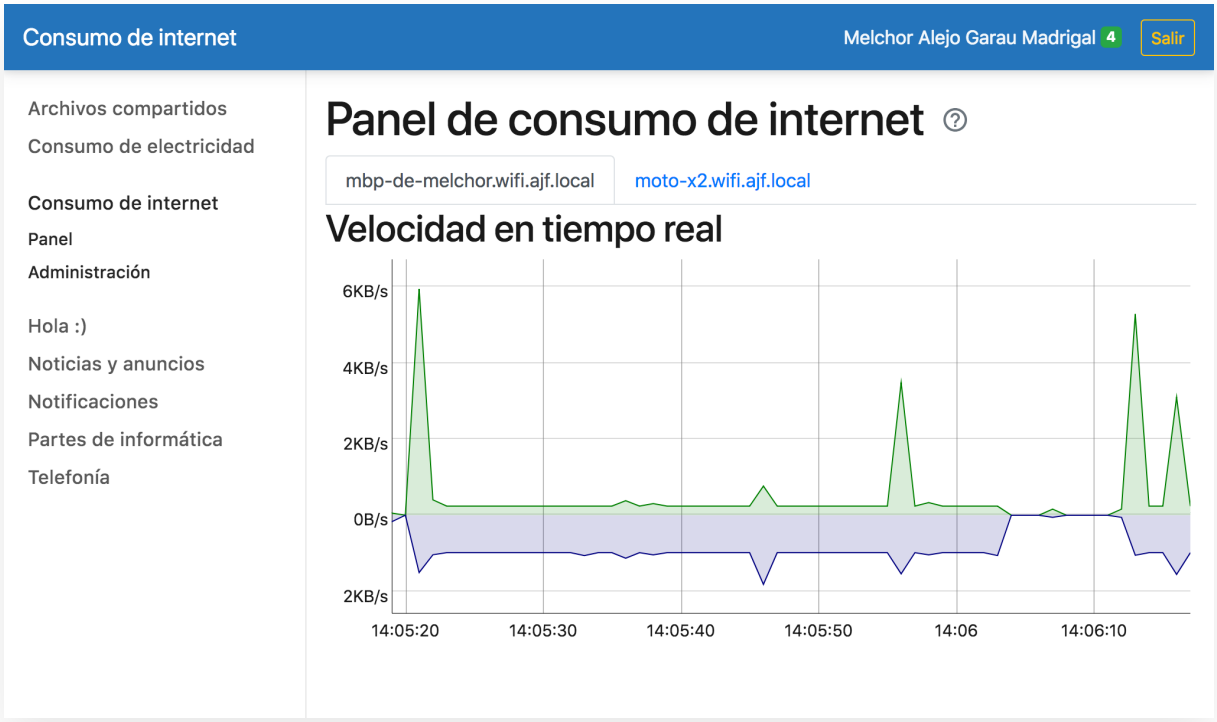
‣ Página resumen del blog



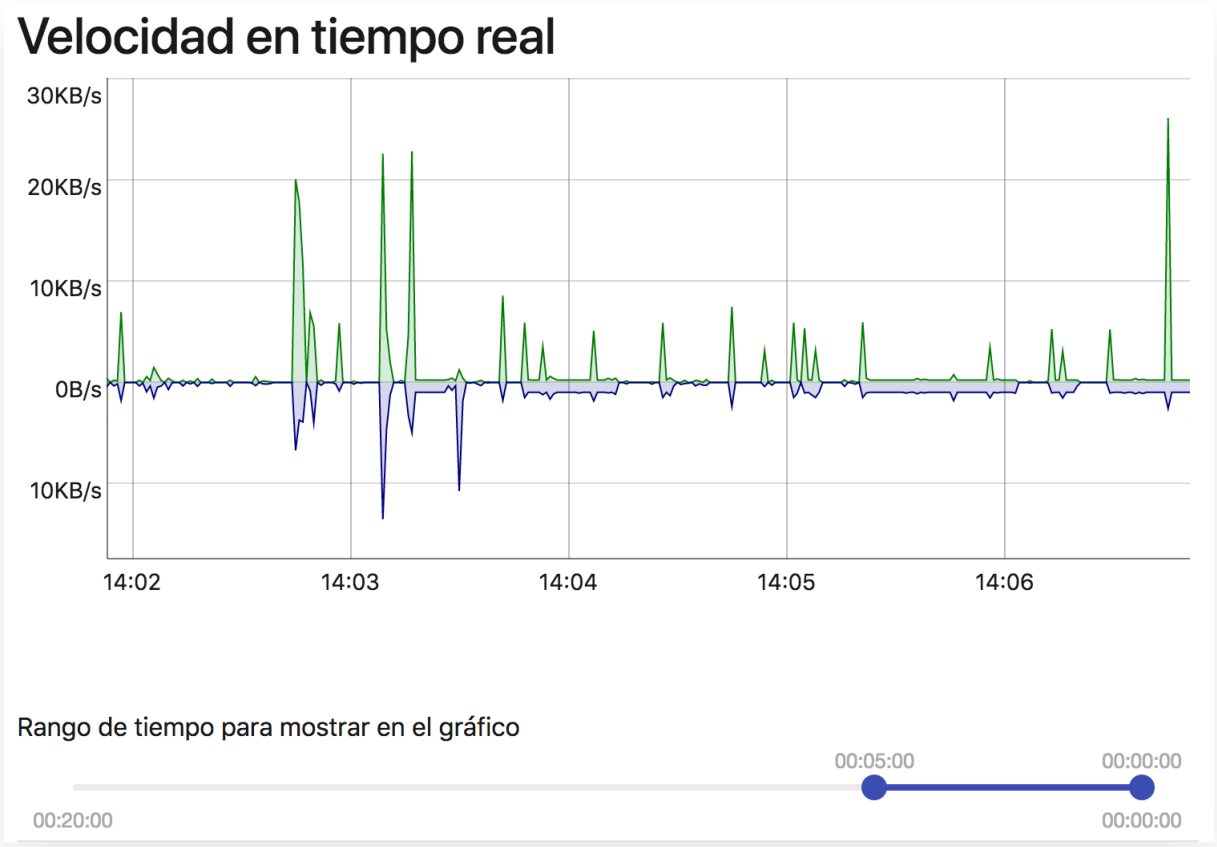
Este módulo es un acceso para el blog en WordPress. Al entrar, se le mostrará al usuario una lista de últimas entradas del blog, a modo de resumen sin tener que entrar en el blog para visualizarlos. Al clickar en una “tarjeta” de un post, se abrirá una nueva pestaña con la entrada completa. Encima de este listado, se encuentran dos botones: grande y azul para ir al blog, mas pequeño y gris para ir a otra página.

Esta otra página explica al usuario diversas maneras de recibir notificaciones cada vez que se publica una entrada.

‣ Módulo para consumo de internet

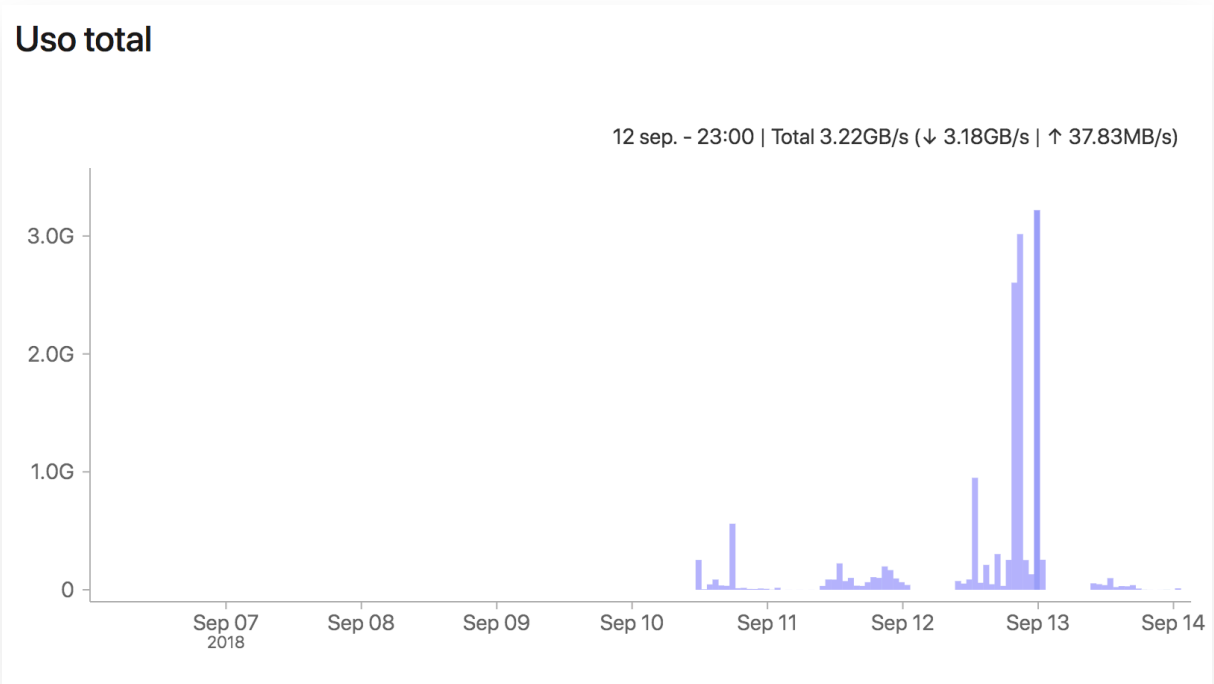


Panel con dispositivos y gráficos

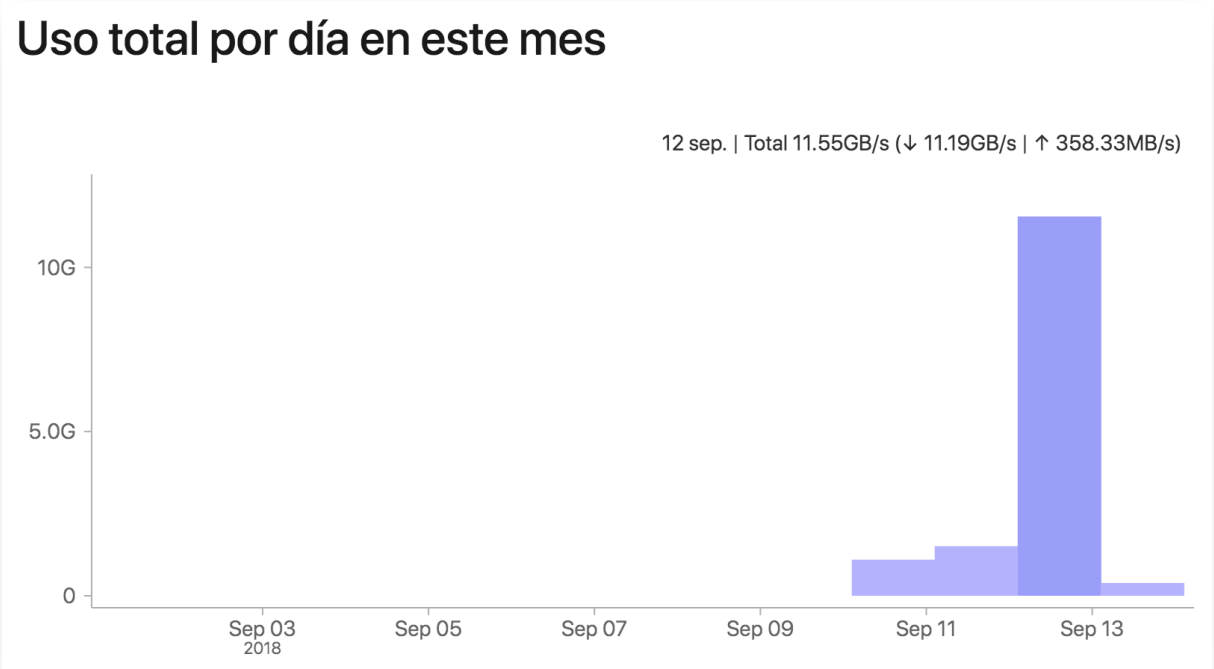


Gráfica de tiempo real con selector de rango de tiempo

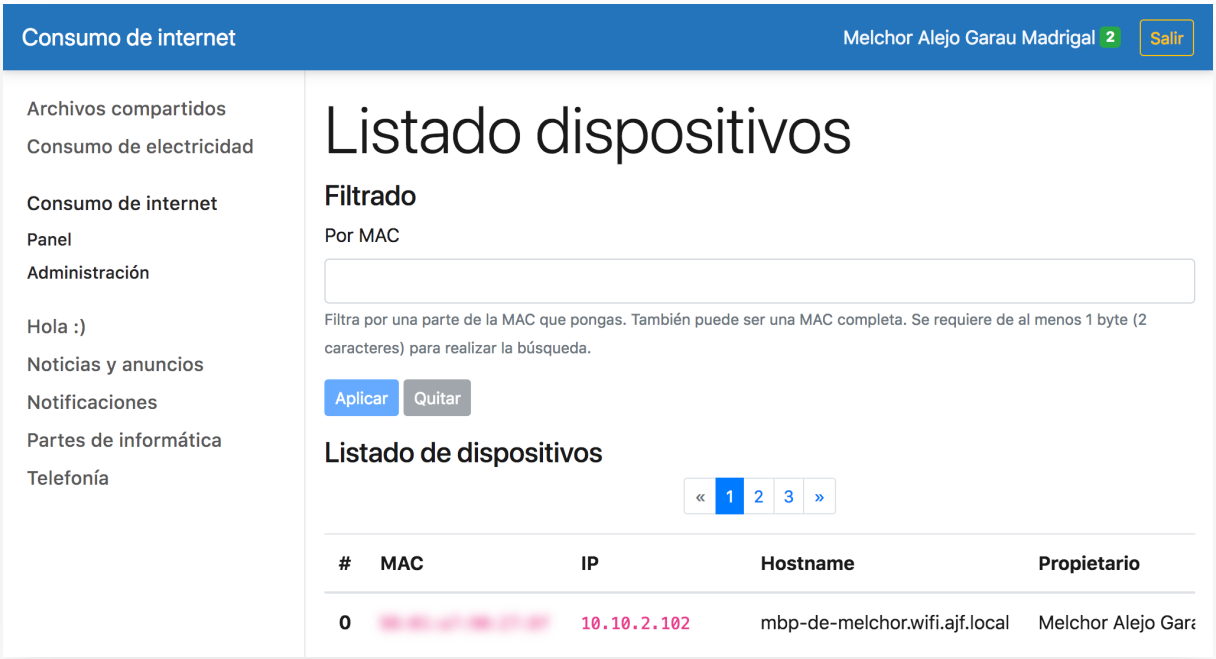
Datos hora a hora de la última semana sobre velocidad media



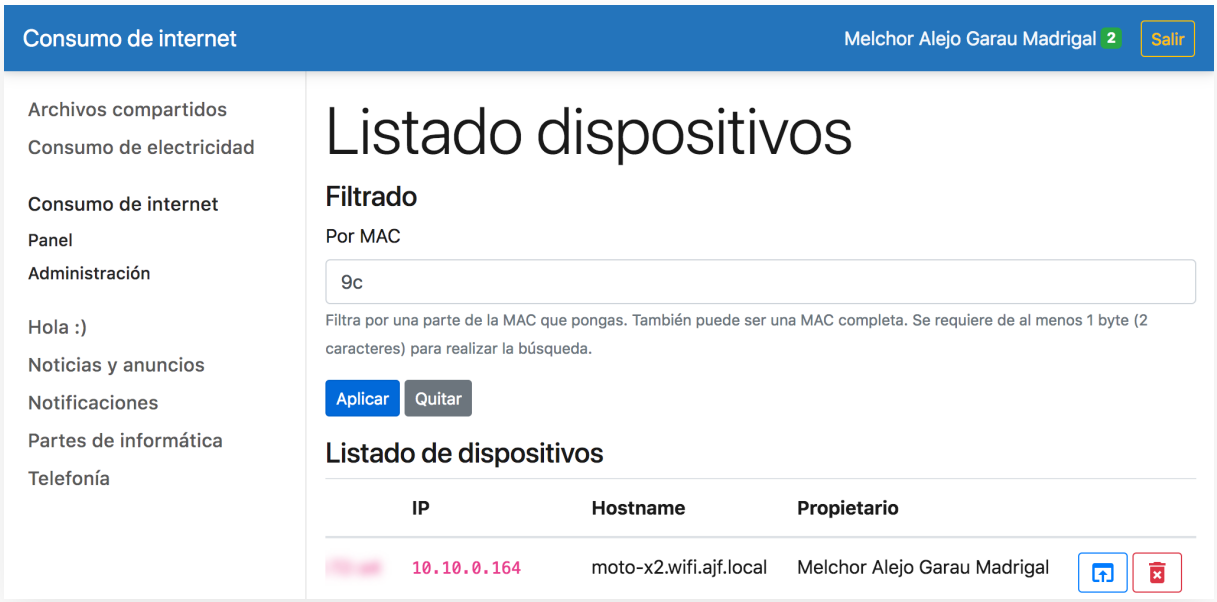
Datos hora a hora de la última semana sobre consumo agregado



Gráfica día a día del mes sobre consumo agregado



Panel de administración (listado de dispositivos)



Panel de administración con filtrado (más botones de la tabla)

En el módulo de consumo de internet, un residente es capaz de visualizar diversas gráficas sobre los datos que se obtienen del uso de internet que realizan sus dispositivos. En la parte superior de la página encontrará una lista de pestañas con cada uno de los dispositivos asociados a su cuenta. Al hacer click en alguno, aparecerán las gráficas. Además, la URL se modificará permitiendo recargar la página o llevarla a otra pestaña (copiando y pegando) sin perder el dispositivo seleccionado. En caso de entrar con una URL inválida o con un dispositivo que no es del residente, no hará nada.

La primera gráfica que encontramos muestra en tiempo real el consumo de ancho de banda del dispositivo. En la gráfica, hay un campo que permite seleccionar el rango de fechas a mostrar en el gráfico. El rango usa fechas relativas al momento actual, este es el motivo por el cual la fecha más próxima a la actual está más a la derecha, y la más lejana a la izquierda, expresando así el paso del tiempo y siguiendo el mismo esquema que la gráfica. El campo escala sus valores máximo y mínimo de acuerdo a los valores actuales, permitiendo dar mayor precisión cuando ambos valores del campo son próximos, y menor precisión cuando la diferencia es mayor. El valor más antiguo que se puede obtener es de 1 día.

Después tenemos dos gráficas relacionadas con datos de la última semana. El primer gráfico muestra la media de velocidad de uso de internet hora a hora. El segundo gráfico muestra el consumo agregado (suma del total consumido tanto en descarga como subida), hora a hora también. En este último, al colocar el ratón encima de una de las barras, te permite ver el consumo por separado (subida y descarga) a parte del total.

La última gráfica representa el consumo agregado desde que empezó el mes hasta el día anterior. Los datos son día a día y al igual que el anterior de consumo agregado, son la suma de la descarga y la subida, y con opción de ver los datos por separados al poner el ratón encima de una de las barras.

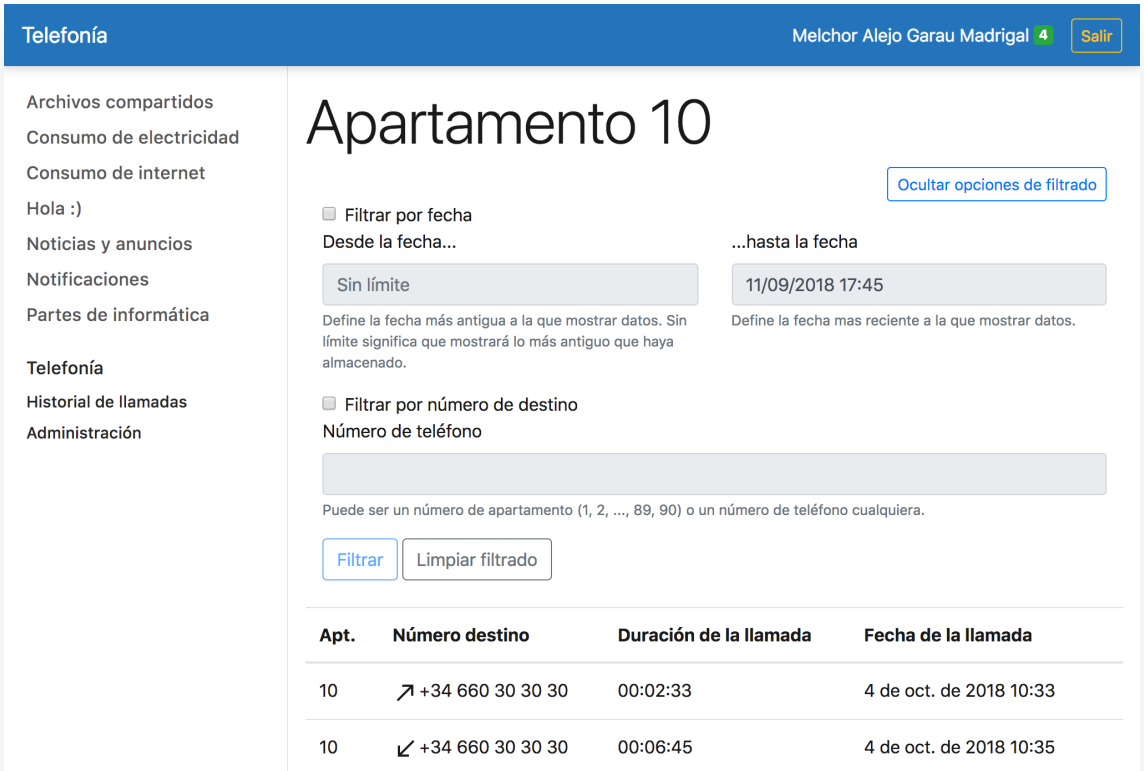
Si no hay datos para una fecha determinada, en cualquiera de los gráficos se mostrará como 0 (no consumido).

Para los gráficos de velocidad, se puede hacer zoom en las gráficas seleccionando, con el click izquierdo, una región. Para poder mover la gráfica de un lado a otro, se puede realizar apretando la tecla Mayúscula (o *Shift*) y arrastrar con el botón izquierdo del ratón. Para salir de este modo basta con apretar dos veces el botón izquierdo.

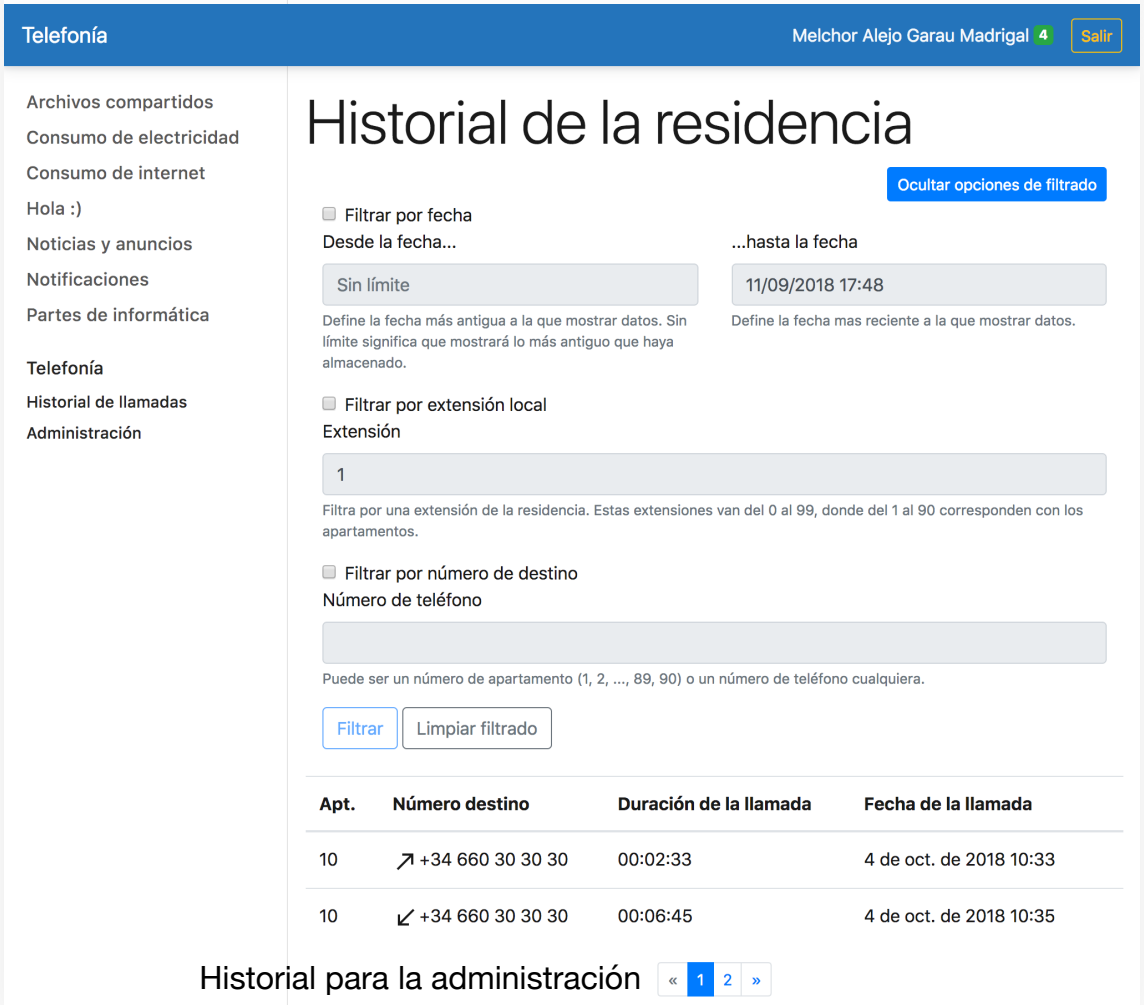
En la página de página de administración de este módulo encontramos una lista dispositivos registrados en el sistema. Esta lista es la que permite al panel, descrito anteriormente, saber qué dispositivos mostrar para el usuario de la sesión. En este panel, se puede filtrar por una dirección MAC (o parte de ella).

Dentro de la tabla, al final, hay unos botones que permiten realizar dos acciones (ultima captura). El primer botón permite ver los gráficos para ese dispositivo. El segundo, en cambio, permite eliminar la asociación de la tabla.

- Módulo para historial de llamadas



Historial para el apartamento del residente

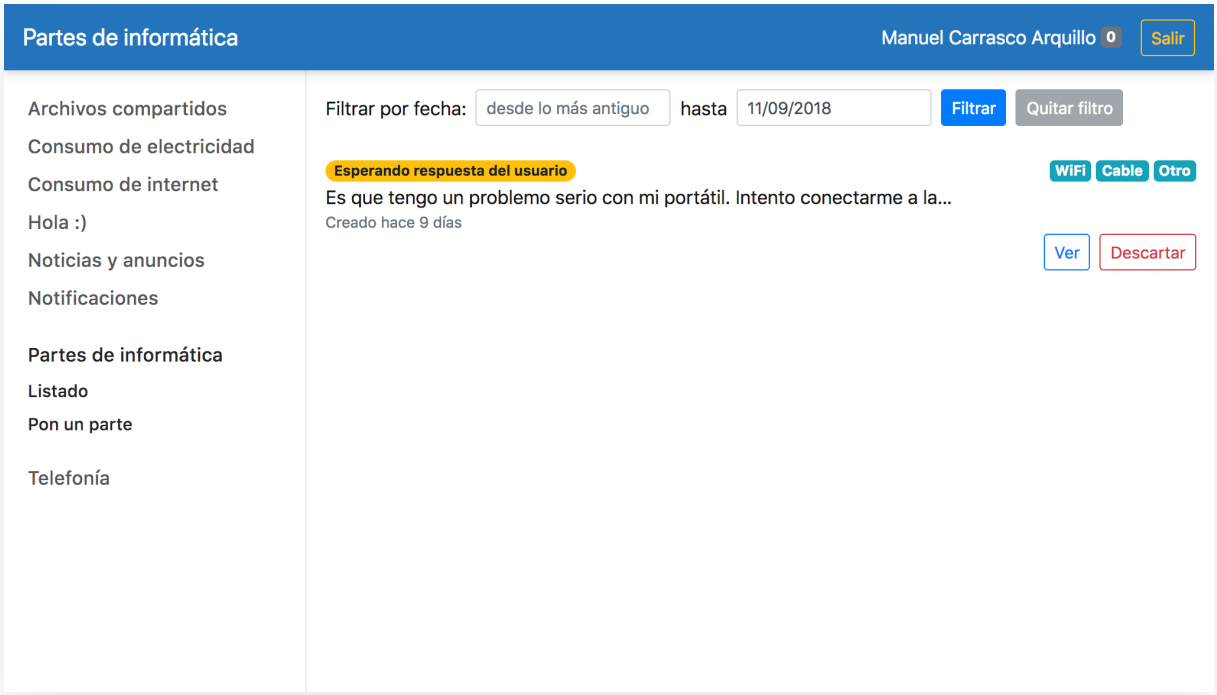


Historial para la administración

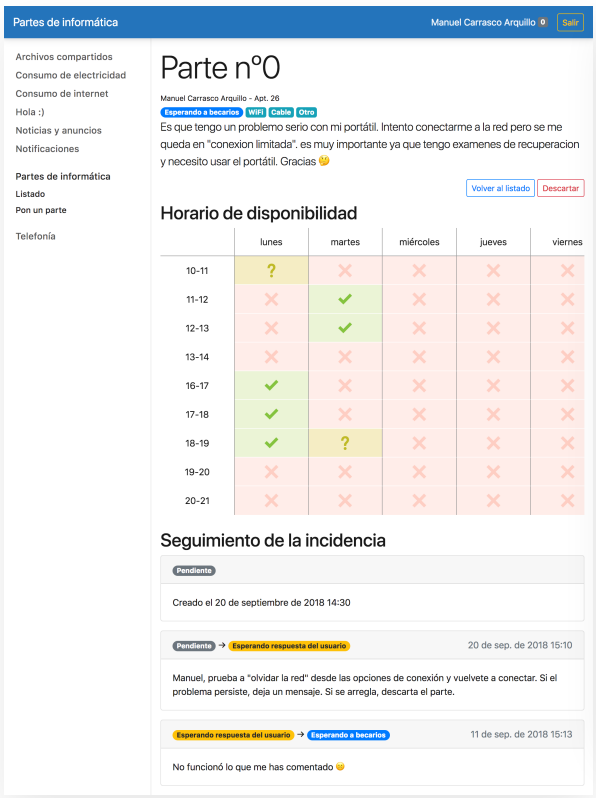
En este módulo, los residentes y los administradores de la residencia pueden ver el historial de llamadas de su apartamento o teléfono de recepción, respectivamente. Incluyen opciones de filtrado por fechas (por rango o anteriores a una fecha) y /o por un teléfono externo. Todos los resultados están ordenados de más nuevo a más antiguo, y aparecerán dentro de un sistema de paginación. Las opciones de filtrado aparecen ocultas por defecto y con el botón, que hay debajo del titulo “Apartamento 10” (en el caso de la captura primera), es posible mostrarlo y volverlo a ocultar. Además, las páginas y las opciones de filtrado se ven reflejados en la URL, de tal forma forma que actualizar la página o abrirla en otra pestaña o navegador permite mantener las opciones de filtrado y la página en la que se encontraba el usuario.

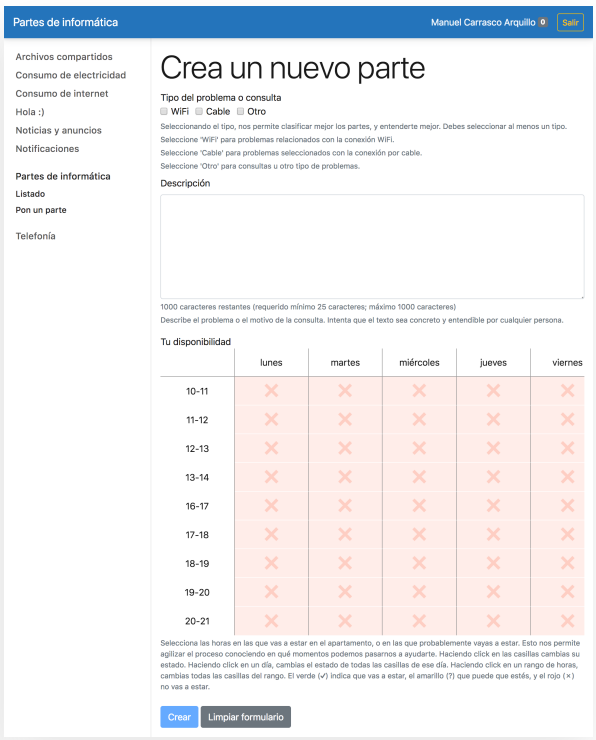
Para la administración de la residencia, de recepción, también tienen una página (segunda captura) que les permite ver el historial de toda las extensiones de la residencia. Sobre las opciones de filtrado, se mantienen todas, y se añade la opción filtrar por extensión. También incluye el botón para mostrar y ocultar esas opciones de filtrado, y se mantiene el formato de URL que permite recuperar esas opciones al actualizar o cambiar de navegador.

‣ Módulo para partes informáticos



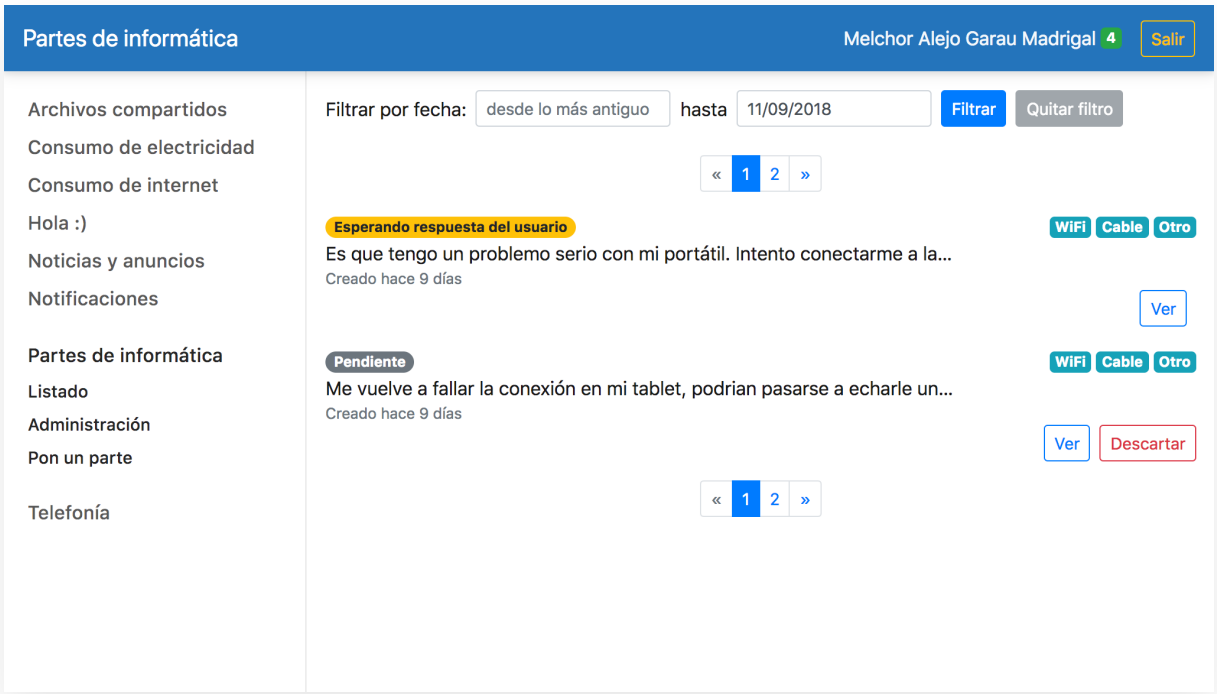
Listado de partes de un residente





(izquierda) Página de un parte

(derecha) Página de creación de un parte



Listado de partes desde el punto de vista de un becario

Con el objetivo de informatizar el sistema de partes informáticos, los residentes pondrán sus problemas relacionados con el internet o la informática en la web, en lugar de hacerlo en papel como se ha estado haciendo hasta ahora. En el módulo, los residentes (primera captura) podrán ver un listado de todos los partes que han creado con el estado de cada uno de ellos. Además, se tendrá la opción de poder hacer un filtrado por fecha.

Cada parte se podrá visualizar de forma más completa (segunda captura, izquierda), con la que se incluirá un seguimiento del parte, que son los estados por los que ha pasado el parte hasta el momento y la tabla de disponibilidad del creador del parte.

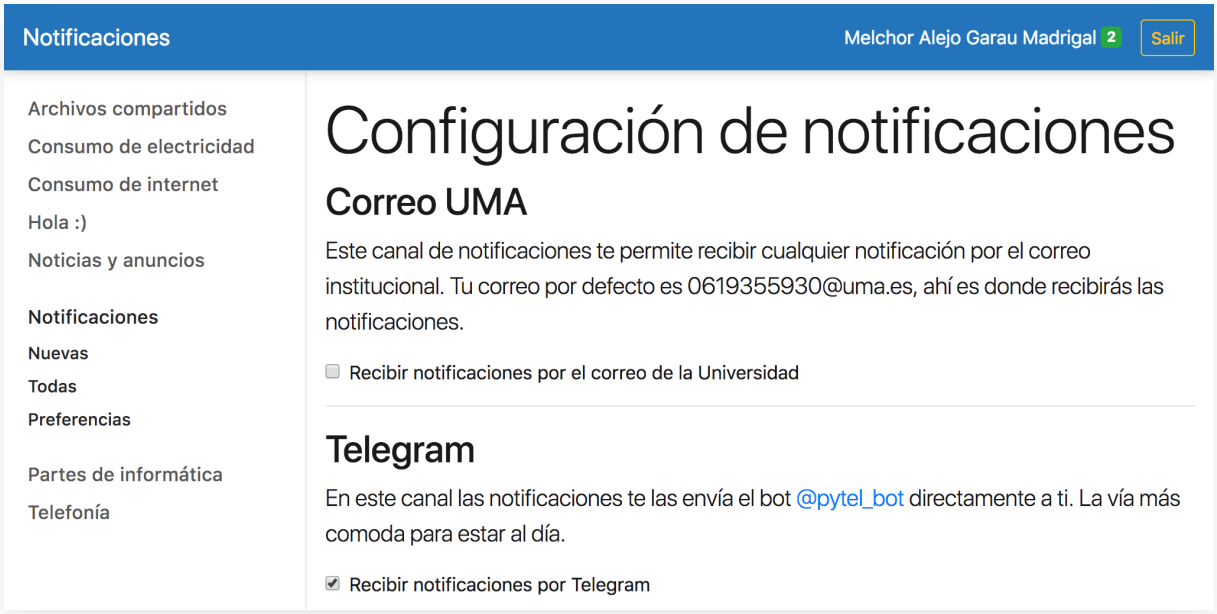
En las anteriores páginas, se puede observar un botón rojo con texto “Descartar”. Ese botón permite al residente, creador del parte, descartarlo. Al descartar, el residente debe especificar el motivo de tal acción, y el estado del parte cambiará a “descartado”. El mensaje aparecerá en el seguimiento del parte.

En el caso de que un parte esté en el estado “Esperando al usuario”, el creador del parte deberá poner un mensaje, desde la página del parte, para responder a la petición del becario.

Cualquier residente puede poner un parte con un sencillo formulario que le permiten expresar su problema y la disponibilidad suya para que los becarios le puedan arreglar el problema (tercera captura, derecha). En el formulario, debe indicar el tipo de problema que tiene, con opción a escoger más de un tipo. Luego, debe rellenar un campo de texto con la descripción del problema. Y, al final, se le pide al residente que rellene la tabla de disponibilidad con las horas en las que estará o puede que esté disponible, para agilizar la gestión del problema. El formulario no es más que la versión informática del formulario en papel original.

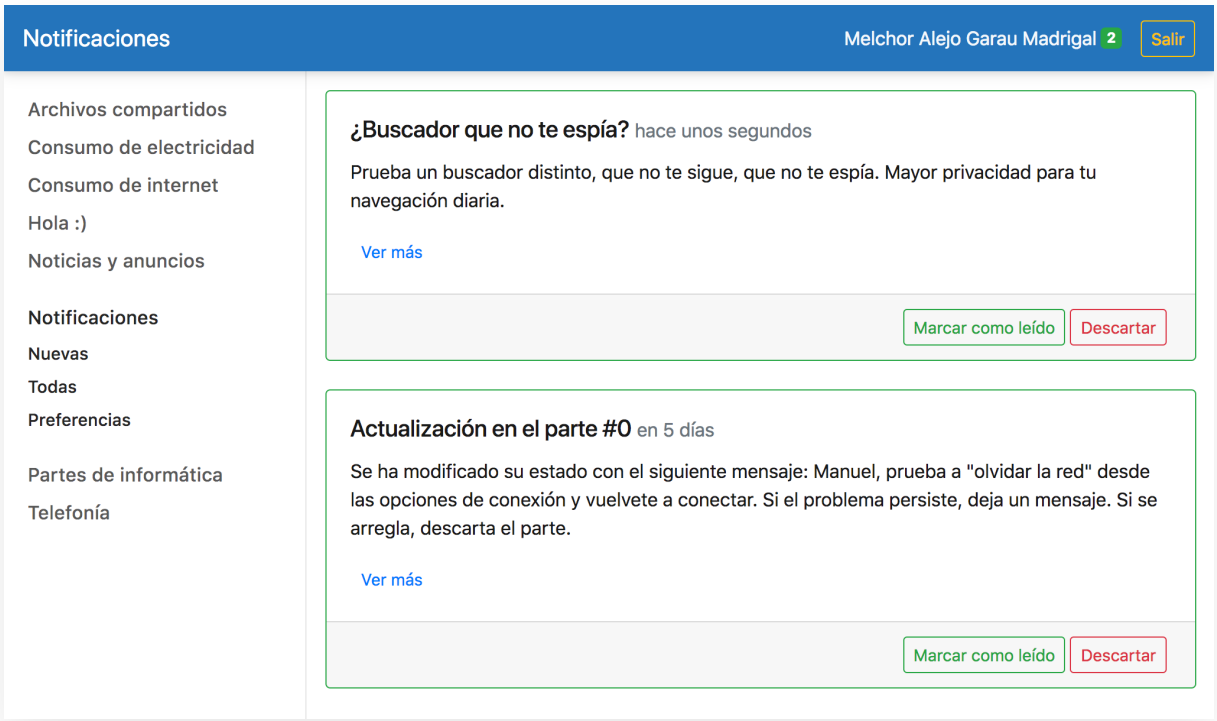
Por último, los becarios de informática tienen la opción de poder administrar todos esos partes que rellenan los residentes, en una página propia (cuarta y última captura). El diseño y funcionalidad es la misma que en el listado de los partes de un usuario, pero muestra todos los partes, en su lugar. Además, siendo becario, puede ver cualquier parte y modificar el estado del mismo, siempre que no se haya dado como terminado (en estado *descartado*, *no se arreglará* o *solucionado*) o se esté esperando la respuesta del residente.

- Módulo para notificaciones

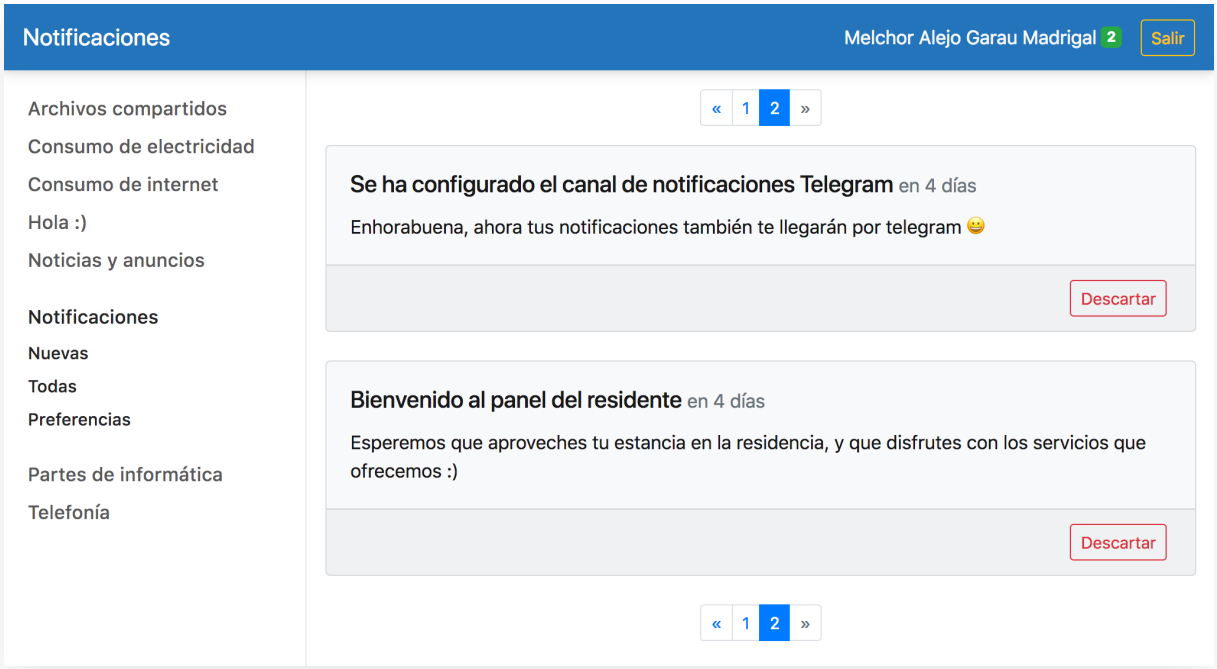


Configuración de notificaciones

Configuración de notificaciones, con el código de activación para Telegram



Listado de notificaciones sin leer



Listado de todas las notificaciones

El módulo de notificaciones permite al usuario configurar los canales de notificaciones y ver sus notificaciones, sin leer o todas.

En la configuración, con un simple click se puede activar o desactivar los canales de notificación disponibles. En cada canal se incluye una breve descripción de a donde se envían las notificaciones y debajo la casilla para activar o desactivar dicho canal. En el caso de Telegram, antes de poder usar ese canal hay que configurarlo. Para ello, tal y como se muestra en la segunda captura, el usuario debe hablar al bot (*el que aparece es un ejemplo, no es uno real*) y le debe enviar el código que aparece en pantalla, y después debe apretar el botón de "*Comprobar activación*" que le notificará si el proceso ha funcionado correctamente.

En el listado de notificaciones sin leer, que corresponde con la tercera captura, permite ver dichas notificaciones e interactuar con ellas. Si las notificaciones contienen un enlace, se mostrará como "*Ver más*". Al apretar el enlace, se marcará como leído. En el resto de notificaciones, se puede marcar como leído con el botón verde. El botón rojo de "*Descartar*" permite eliminar la notificación completamente del sistema.

En el listado de notificaciones, el usuario puede ver todas las notificaciones (leídas y sin leer), y hacer las mismas operaciones que en el listado de notificaciones sin leer.

5. Despliegue de servicios

Estado actual de los servicios

Empezaremos describiendo los servicios actuales en la residencia, como están instalados y funcionando, y como podría ayudar el uso de virtualización mediante contenedores.

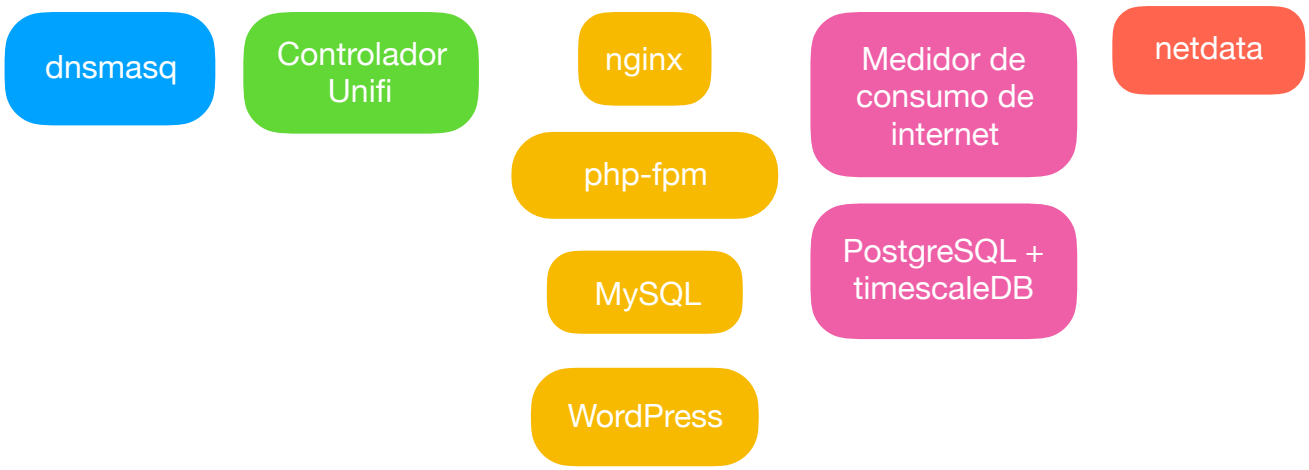


Gráfico con los servicios actuales que se pueden virtualizar, agrupados en columnas por relación

Actualmente, en la residencia, tenemos un conjunto de servicios que sustentan la residencia o complementan las actividades de los residentes. Una buena parte son servicios web, pero otros son servicios de sistemas y redes.

- dnsmasq

Tenemos primero el servicio de DHCP + Caché DNS que, como se ha mencionado diversas veces en la sección 2 (*Diseño, configuración y despliegue de la infraestructura de red*). Es un servidor que ofrece DHCP (para IPv4) a las distintas redes de la residencia, para que los dispositivos reciban su IP asignada (estática o dinámicamente). Además, tiene un servidor DNS que almacena localmente copias de entradas DNS para una mayor respuesta a nivel local, además de tener en su registro los dispositivos que se les ha asignado una IP por DHCP (y hayan dado un nombre de dispositivo - conocido como *hostname*). Es un elemento esencial de la red que permite a todos poder configurar sus dispositivos automáticamente, incluyendo teléfonos VoIP, switches y APs.

Virtualizado serviría para hacerlo portable, fácilmente instalable y se reducirían los tiempos de despliegue en caso de fallo. Pero nos quitaría de actualizaciones

automáticas de seguridad que vienen con Ubuntu Server y la estabilidad que ofrece este paquete, además de la fuerte dependencia con las interfaces de red del sistema host. La configuración del servidor, teniendo una copia de los archivos de configuración, solo sería *copiar y pegar*. Con estos pros y contras, veo que no seria necesario virtualizar el servicio ya que actualmente como está, va bien y no da problemas.

‣ Controlador Unifi

El siguiente servicio es el controlador Unifi. Es el que se encarga de gestionar los APs y los switches (menos el de la zona de recepción). Además nos permite poder ver el estado de todos ellos, configurarlos, actualizar su firmware…, a través de un panel web. El controlador usa diversos puertos para el control de los dispositivos.
También podría ser buen candidato para virtualizar, pero debido a que, al igual que el servidor DHCP, necesita acceso directo a la red de los switches y APs, dificulta su proceso. Existe dos métodos para poder usar este servicio en un contenedor: uno es usar las mismas redes que el del sistema (exactamente igual que como está ahora) y que no es compatible con orquestadores de contenedores como Kubernetes o Docker Swarm, o usar una red macVLAN[19] que permite conectar el contenedor directamente a una red real y es compatible con orquestadores. Aunque, en mi opinión, posiblemente sea más difícil montar todo el sistema virtualizado que dejar el servicio tal y como está. Existe un ejemplo de contenedor que permite ambos modos de red[20].

‣ Servidor Web

También tenemos una web. La web está dividido en diversos elementos, por lo que primero describiremos el servidor web al que se conectan todos los usuarios. El servidor es un nginx[21] que sirve archivos estáticos que se encuentran en la carpeta de archivos web. Además tenemos un WordPress donde se publican algunas circulares, información que ofrecen los becarios (si lo desean) o ciertas noticias relacionadas con la residencia. Este usa también nginx, aunque delega gran parte del trabajo a PHP por ser una aplicación web hecha en PHP.
Este servicio se podría dividir en un proxy inverso (un servidor web que redirecciona peticiones web a otros servidores dependiendo de cual sea), otro para los archivos

estáticos de la web y otro para WordPress con un apache + mod_php (ya que WordPress se lleva muy bien con este tipo de configuraciones). Para el proxy inverso se usaría træfik[22] que se define como "*The Cloud Native Edge Router*"[22]. Este software permite el auto-descubrimiento de servicios y su configuración mediante parámetros que se ofrecen dentro del mismo descubrimiento. Con este træfik, añadir y quitar partes de la web será tan fácil como ejecutar y parar contenedores. Para los archivos estáticos, se podría usar nginx también, ya que es bastante ligero. Para WordPress[23], ya se ha descrito que contenedor se podría usar. Además éste requiere de una base de datos MySQL[24] o MariaDB[25], por lo que añadir uno al despliegue es bastante fácil: con solo escribirlo en un archivo que describe los servicios es suficiente (más adelante se comentará sobre esto).

Dentro del servidor web, también tenemos una pequeña web que muestra el consumo de ancho de banda que está haciendo el dispositivo con el que accedes al mismo, siempre que el dispositivo está dentro de la residencia. Para su funcionamiento, requiere de una aplicación que capture el tráfico, lo clasifique, calcule el consumo y lo almacene en una base de datos. Este servicio pasará a ser parte de la aplicación web que se describirá más adelante en este trabajo.
Por lo que la parte de la web, se describirá en esa sección como quedaría, pero la aplicación que mide el uso se va a colocar en un contenedor en el servidor/puerta de enlace. Además, esta aplicación requiere de una base de datos PostgreSQL[26] con la extensión timescaleDB[27], ya que almacena la información con tiempo y dicha extensión permite manejar los datos de una forma muy cómoda para agrupar por tiempo.

‣ Monitorización de servidores

Para monitorizar el estado del servidor, usamos una aplicación llamada netdata[28]. Es capaz de obtener métricas de todo tipo y mostrarlas en una web.
Existe una versión para contenedor de este software. Aunque este contenedor no es capaz de acceder a toda la información que nos podría interesar de la puerta de enlace (no es capaz de leer datos de las tarjetas de red)[29]. Por lo que para este servidor se dejará como se encuentra actualmente, sin contenedor, y se enlazaría con træfik (el proxy inverso) mediante configuración manual. Si se instala un nuevo

servidor, es posible que sea más cómodo para éste usar la version del contenedor, ya que no interesaría tener tanta información más allá de los propios contenedores y el estado del sistema.

Para resumir, se va a exponer una tabla con los componentes que estarán virtualizados y los que no, con lo que llevamos descrito hasta el momento:

En contenedores	**Fuera de contenedores**
træfik (proxy inverso - c*loud native edge router*)	dnsmasq (DHCP + DNS)
nginx (para archivos estáticos)	Controlador de Unifi
WordPress con apache + php	netdata en la puerta de enlace
MariaDB o MySQL	
Medidor de uso de internet	
PostgreSQL + timescaleDB	
netdata en otros servidores	

Nuevos servicios

Sobre la marcha, han ido apareciendo nuevos servicios que son necesarios para algunas partes que se describirán en este trabajo posteriormente o en el trabajo de Antonio Ángel Cruzado Castillo.

‣ Servidor VoIP

El primer servicio que se va a añadir es el de la centralita VoIP usando el programa Asterisk. Este servicio es el que permite a los teléfonos VoIP poderse comunicar con otros teléfonos de la residencia y con el servicio de telefonía de Vodafone (al exterior de la residencia). Más detalles sobre este servicio lo podrán encontrar en la sección «*Configuración de asterisk*» del mencionado trabajo de Antonio Ángel.

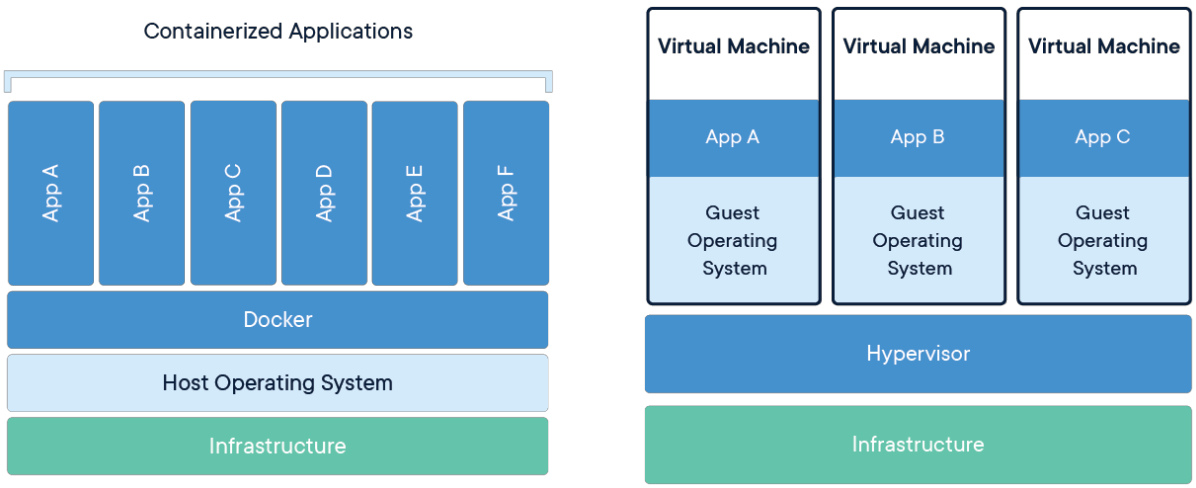
Y, ¿podría ser virtualizado este servicio? Como un servicio que puede ir 100% sobre la red de internet, podría ser virtualizado sin problemas. Un detalle a tener en cuenta es que, por como es el protocolo SIP, es posible que asterisk requiera de muchos puertos efímeros. Aunque no debería ser problema ya que se pueden configurar qué puertos efímeros usar y se pueden “publicar” todos ellos fácilmente. O usar una configuración de red “host”, donde las redes del contenedor son las mismas que las

del servidor, pero disminuyendo el aislamiento. Además, para este servicio, también requiere de una base de datos MySQL (o MariaDB) para almacenar cierta configuración y datos sobre las llamadas.

Otro servicio que va a hacer falta es el *backend* de la aplicación web. También añadir el sistema de autenticación con la UMA, que requerirá de un servidor redis.
Este servicio está pensado para virtualizarse en contenedores. No requiere de más extras a parte de ponerse detrás del proxy inverso (*træfik*). También se requerirá una base de datos PostgreSQL para almacenar datos de la aplicación, y para obtener los datos del medidor de uso de ancho de banda. Y también necesitará acceso a la base de datos de Asterisk para alguno de sus servicios. Además de la mencionada base de datos redis, que se usa en el sistema de autenticación.
Un elemento importante del *backend* es el módulo de inicio de sesión con la Universidad mediante SAML[44], que se desplegará por separado.

Migración de servicios existentes a Docker

En este apartado se describirá la metodología seguida para pasar los servicios actuales a imágenes de Docker, se explicará el funcionamiento que tiene Docker y algunos de los problemas que han ocurrido en el proceso.



Comparativa entre contenedores y máquinas virtuales[31]

Antes de empezar con la migración, se explicará como funciona Docker para que se entienda antes de explicar el trabajo realizado para migrarlos.
Docker es un servicio de contenedores, que no son mas que una especie de máquinas virtuales, pero más ligeros. Los contenedores virtualizan parte del sistema operativo, en contraposición a las máquinas virtuales que virtualizan hardware. Todos comparten el mismo *kernel,* pero solo son capaces de ver lo que el creador de la imagen y del contenedor quieren que vean éstos. Esto permite virtualizar servicios y que se ejecuten en segundos, que puedan ser fácilmente escalables (aumentar o reducir réplicas del mismo servicio) e independientes del sistema operativo en el que se ejecutan (Linux o Windows en caso de Docker).
La base de los contenedores son los que Docker llama "*imágenes*". Es el sistema de archivos inicial que tendrá el contenedor cuando se ejecute por primera vez. Contiene una base de bibliotecas y ejecutables que permiten al contenedor funcionar con lo que se desee. Las imágenes se montan sobre capas, por lo que permite tener varias imágenes que comparten una misma base, y reducir espacio. Por lo que un "*contenedor*" no es más que una imagen en ejecución. En un contenedor se puede configurar qué archivos puede ver y donde almacenarlos, además de qué redes puede ver. También ofrece la posibilidad de limitar el uso de CPU y de memoria de cada uno de ellos.
Con la ayuda de un orquestador, como Kubernetes o Docker Swarm, Docker puede elevar su funcionalidad a multiples nodos (servidores que ejecutan contenedores) que trabajan conjuntamente para ejecutar todos los contenedores, permitiendo escalabilidad y alta disponibilidad. Con un orquestador, uno podría montar un "*cloud*" como los que se encuentran en los servicios PaaS (*Platform as a Service*).
Para nuestro caso, no se usará dicha funcionalidad porque la infraestructura no es tan grande que la diferencia entre usar un orquestador y no usarlo son pocas.

Tratando el tema de cómo migrar lo servicios, se seguirá estos pasos:

1. Analizar los componentes que conforman el servicio (analizado en los apartados anteriores)
2. Buscar si existen imágenes en https://store.docker.com[33] con el componente listo para usarse y analizar si valen la pena

3. Preparar el archivo para generar la imagen (*Dockerfile*) y los archivos extra para configurar el componente
4. Probar su funcionamiento

A la hora de buscar imágenes existentes, se intentarán que todos se basen en la misma imagen, que suele ser Debian Stretch, en la fecha de escritura de la memoria.

El primer componente que se convertirá es el servidor nginx. En la tienda de Docker, se encuentra la imagen oficial de nginx[34] y con una versión en Debian. Esta imagen viene con una configuración por defecto del servidor, según se puede leer en la misma página [34].

Para nuestro caso, tenemos que añadirle algún que otro archivo de configuración para poder personalizar los archivos de error (errores como estos códigos de estado HTTP: 404, 401, 500…)[35]. El servidor actual tiene soporte para PHP mediante la versión *php-fpm* que usa el protocolo FastCGI, por lo que hay que añadirlo también a la imagen. Por lo que, se necesitarán dos archivos de configuración: uno que contenga la configuración para que nginx se pueda conectar con php-fpm (fastcgi-php.conf), otro con la configuración por defecto de nginx con los añadidos (default.conf).

El archivo que genera la imagen se quedaría de la siguiente forma:

```
FROM nginx:latest

COPY default.conf /etc/nginx/conf.d/default.conf
COPY fastcgi-php.conf /etc/nginx/

EXPOSE 80
```

Describiendo lo que hace el archivo, la primera linea indica cuál es la base de esta nueva imagen, que es nginx. Nótese que al final está *:latest*, que indica qué etiqueta (versión) de la imagen se va a usar. Los dos siguientes son los que copian los archivos de configuración al interior de la imagen. El último indica que en el puerto 80 (tcp) nginx va a estar escuchando para peticiones.

Estos archivos se pueden encontrar adjunto, dentro de la carpeta de *web-nginx* en *repositorios*.

Para poder probar este servidor web, necesitamos que esté funcionando también el servidor de PHP, por lo que se describirá el proceso de crear este y en la prueba se juntarán ambos servidores.

El servidor php-fpm también está disponible como imagen oficial de Docker[36]. En la página de la tienda, hay disponibles muchas versiones de la imagen. En nuestro caso nos interesa una versión de PHP 7 con el motor de FastCGI. Lo encontramos con la etiqueta *fpm* que está basado también en Debian.
Acorde con [36], en la sección "*How to Install more PHP extensions*", la imagen viene con diversas extensiones incluidas, pero para algún futuro uso vendría bien tener la extensión *gd* de manipulación de imágenes, *mysqli* para conectarse a base de datos MySQL y *opcache* para acelerar la ejecución de los scripts PHP. Este último es bastante importante para mejorar el rendimiento de PHP.
Por tanto, el archivo que genera la imagen seria el siguiente:

```
FROM php:fpm

RUN apt-get update && apt-get install -y \
        libfreetype6-dev \
        libjpeg62-turbo-dev \
        libpng-dev \
    && apt-get clean \
    && docker-php-ext-configure gd --with-freetype-dir=/usr/include --with-png-dir=/usr/include --with-jpeg-dir=/usr/include
    && docker-php-ext-install -j$(nproc) gd mysqli opcache \
    && rm -rf /var/lib/apt/lists/*; \
# set recommended PHP.ini settings
# see https://secure.php.net/manual/en/opcache.installation.php
    { \
        echo 'opcache.memory_consumption=128'; \
        echo 'opcache.interned_strings_buffer=8'; \
        echo 'opcache.max_accelerated_files=4000'; \
        echo 'opcache.revalidate_freq=2'; \
        echo 'opcache.fast_shutdown=1'; \
        echo 'opcache.enable_cli=1'; \
    } > /usr/local/etc/php-fpm.d/opcache-recommended.ini

EXPOSE 9000
```

En este archivo, nos basamos en la versión de *php:fpm*. Instalamos las extensiones mencionadas (y sus dependencias) siguiendo [36]. También añadimos un archivo de configuración para la extensión *opcache*, con la configuración que recomiendan desde el manual de PHP[37]. La instrucción RUN permite ejecutar comandos en un shell en la imagen, mientras se genera.
Estos archivos se pueden encontrar en *web-php-nginx* dentro de *repositorios*.
Para poder probar el servidor web basta con ejecutar los dos contenedores con una carpeta común montada que contenga algunos archivos (y algún script de PHP). Para ello, un sencillo archivo de docker-compose[38] puede ayudar a simplificar la prueba. Esta herramienta nos permite definir diversos componentes, las necesidades que tengan y desplegarlos todos a la vez.

El archivo de despliegue que se puede observar en el lateral, permite hacer una prueba con los dos servicios, montando la carpeta *web,* que se encuentra junto al archivo de despliegue, dentro de cada uno de los contenedores.

```yaml
version: 3

services:
  nginx:
    image: web-nginx
    volumes:
      - ./web:/usr/share/nginx/html:ro
      - ncache:/var/cache/nginx
    depends_on:
      - php-fpm
    ports:
      - "80:80"

  php-fpm:
    image: web-php-fpm
    volumes:
      - ./web:/var/www/html

volumes:
  ncache:
```

En la primera prueba, se observó que los scripts PHP fallaban porque las rutas dentro de ambos contenedores no coinciden: en nginx *es /usr/share/nginx/html* donde se encuentran los archivos, mientras que en php-fpm es */var/www/html.*

Para la segunda prueba, se modificó el archivo de configuración de nginx para que usara */var/www/html* como ruta donde encontrar archivos web, y también el archivo de despliegue la ruta se debía cambiar. El resultado fue positivo ya que los scripts PHP ya funcionaban.

Para referencia, el script de php solo contenía lo siguiente:

```php
<?php phpinfo();
```

El siguiente es el servidor web Apache con PHP. Este servidor no existía inicialmente porque se usaba el propio nginx para ello, pero para WordPress es mejor usar el servidor web primero, ya que tiene mejor integración con él.

Curiosamente, en la misma página de la imagen de PHP[36] existe una versión que incluye el servidor web Apache cuya etiqueta es *apache*. Por tanto, se aprovechó el archivo *Dockerfile* de *php-fpm* para generar esta versión, solo cambiando la imagen base a *php:apache*.

A la hora de probar esta nueva imagen, simplemente usando el archivo de despliegue que se encuentra a la derecha es suficiente. Los archivos fueron los mismos que en la prueba de nginx con php-fpm.

```yaml
version: 3

services:
  wordpress:
    image: web-apache-php
    volumes:
      - ./web:/var/www/html
    ports:
      - "80:80"
```

Por experiencia con esta imagen, durante el tiempo el servidor de apache tiene una tendencia a acumular muchos procesos abiertos, por lo que, a modo preventivo, se ha

añadido un archivo de configuración (*mpm_prefork.conf*) a la imagen para evitar que abra tantos procesos.

Para la base de datos MySQL (o MariaDB), se escogió usar MariaDB. Se usará para el blog en WordPress y el servidor SIP Asterisk. En la tienda, podemos encontrar que existe una imagen oficial de MariaDB[39]. En este caso, las versiones existentes se basan en Ubuntu 18.04 (bionic), por lo que en este caso no se podrá aprovechar el sistema de capas de Docker.
Para el uso que se le estaba dando la base de datos, no se requería de ninguna configuración especial, por lo que no será necesario crear una imagen a partir de la oficial. En cambio, habrá que persistir los archivos que se generen dentro del contenedor, a la vez que ejecutar unos scripts iniciales para rellenar la base de datos. Estos aspectos se comentarán más adelante, ya que pertenecen a la configuración de despliegue.

La siguiente base de datos será la de redis. Al igual que con MariaDB, existe una imagen oficial de Docker[40]. Esta base de datos es un NoSQL del tipo clave-valor, útil para almacenar datos sencillo temporalmente, como servidor de caché de datos o para crear canales Publicador/Subscriptor (un cliente publica mensajes y otros los escuchan).
En la página podemos encontrar que la versión por defecto es la que se basa en Debian. Al igual que con MariaDB, no se requiere de ninguna configuración extra, por lo que se puede usar la base de datos directamente.

Todo esto son los componentes existentes que tenían que virtualizarse. Componentes nuevos como træfik o postgreSQL no son existentes actualmente por lo que se describirá en la siguiente sección.

Preparando nuevos servicios para virtualizar

En esta sección se comentarán los nuevos servicios y componentes que no existían inicialmente y se han añadido para los nuevos casos de uso. Esto incluye algunos de los componentes mencionados en la tabla del final de la sección "*Estado de los*

servicios actuales” pero que no se han incluido en la anterior sección y los servicios mencionados en la sección “*Nuevos servicios*”. Se seguirá la misma metodología que en la sección anterior.

‣ Træfik

El primer componente que se hablará es el de træfik. Esta aplicación es el punto en común para exponer muchos servicios web a internet. Es un proxy inverso que dependiendo de unas condiciones, reenvía la petición al servicio interno correspondiente para que él se encargue de dar respuesta. Es idóneo para nuestro sistema con diversos componentes separados que se quieren exponer, como el blog WordPress o el backend de la aplicación web de este trabajo. Además, tiene un servicio de descubrimiento de componentes que permite en vivo reconfigurar træfik simplemente iniciando o parando componentes. Este servicio es indispensable para el funcionamiento del resto de servicios en la residencia.
Para usar dicho servicio, solo basta con usar su contenedor oficial, tal y como mencionan al final de la web del producto[22].

‣ Medidor de uso de internet

El siguiente componente es el medidor de uso de internet. Este componente es un proyecto que hice cierto tiempo atrás a modo de experimento, que ha resultado funcionar bastante bien y cobrar cierto interés para añadirlo en la aplicación web parte de este trabajo.
El código fuente se encuentra en GitHub[43] y se provee también de una imagen en Docker para su uso. Este componente solo se puede usar en el servidor/puerta de enlace ya que es ahí de donde se obtiene esa información, y se almacena en una base de datos basado en series de tiempo, como InfluxDB o postgreSQL con timescaleDB, que para nuestro caso será postgreSQL.

Habiéndose mencionado postgreSQL y timescaleDB, es momento de buscar como poder usarlo en nuestro despliegue. timescaleDB[27] es una extensión de postgreSQL[26] que le añade la funcionalidad de base de datos de series de tiempo. Una serie de tiempo es un conjunto de datos donde el campo tiempo está presente. Ejemplo de datos de este tipo seria la temperatura que ha hecho en un lugar la

semana anterior, donde el campo tiempo es un factor importante. Este tipo de base de datos para la obtención de medidas sobre el tiempo es idónea por las operaciones que puedes hacer sobre ellas, como agrupación sobre el tiempo.

Para el despliegue, vamos a necesitar una imagen de esta base de datos con la extensión, que, buscando por la tienda de Docker, encontramos una imagen que crean y mantienen el mismo equipo de timescaleDB, que se encuentra en [41]. Las dos versiones que ofrecen están basadas en Alpine Linux, una distribución de Linux ligera, cuya imagen ocupa solo 5MB. Aunque no se puede aprovechar el sistema de capas de imágenes para reducir las capas comunes, el espacio que ocupa la imagen base es mínima, por lo que no genera mucho problema. Además, usan la imagen oficial de postgreSQL[42] para generar la de timescaleDB.

- Autenticación SAML

```
FROM debian:stretch

ARG ID_URL=...
ARG JQ_VERSION=jq-1.5

RUN apt-get update && \
    apt-get install -y apache2 libapache2-mod-auth-mellon curl redis-tools && \
    apt-get clean && \
    rm -rf /var/lib/apt && \
    mkdir -p /var/www/html/alberginia && \
    curl -sSL ${ID_URL} > /etc/apache2/id.xml && \
    curl -sSL https://github.com/stedolan/jq/releases/download/${JQ_VERSION}/jq-linux64 > /usr/bin/jq && \
    chmod +x /usr/bin/jq

COPY melon.conf /etc/apache2/conf-available/melon.conf
COPY index.shtml show.sh /var/www/html/alberginia/

RUN a2enmod auth_mellon include cgi rewrite headers proxy proxy_http proxy_wstunnel && \
    a2enconf melon && \
    #Changes the logs of the files to stdout and stderr
    echo CustomLog /proc/self/fd/1 combined >> /etc/apache2/apache2.conf && \
    echo ErrorLog /proc/self/fd/2 >> /etc/apache2/apache2.conf && \
    #Also in the 000-default conf file
    sed -ri \
        -e 's!^(\s*CustomLog)\s+\S+!\1 /proc/self/fd/1!g' \
        -e 's!^(\s*ErrorLog)\s+\S+!\1 /proc/self/fd/2!g' \
        /etc/apache2/sites-available/000-default.conf && \
    chown -R www-data:www-data /var/www/html/alberginia

#Environment variables borrowed from `/etc/apache2/envvars`
ENV APACHE_RUN_USER=www-data
ENV APACHE_RUN_GROUP=www-data
ENV APACHE_PID_FILE=/var/run/apache2/apache2.pid
ENV APACHE_RUN_DIR=/var/run/apache2
ENV APACHE_LOCK_DIR=/var/lock/apache2
ENV APACHE_LOG_DIR=/var/log/apache2
ENV LANG=C

VOLUME /mellon
EXPOSE 80

CMD apache2 -DFOREGROUND
```

Por último, el módulo de autenticación con la Universidad mediante SAML. Este modulo no es más que un servidor web apache con un módulo para autenticación mediante SAML llamado mellon[45]. A través de la ayuda desde el Servicio Central de Informática, se montó una imagen de Docker con todo lo necesario para realizar la autenticación y pasar la información relevante al *backend* de la aplicación web.
La imagen está basada en Debian, y se instala, mediante el gesto de paquetes de la distribución, el servidor web, el módulo mellon y la herramienta por terminal para hacer peticiones a un servidor redis; luego configura el servidor para activar el módulo en la ruta */alberginia*. En esta ruta, al acabar la autenticación, un script guarda en una base de datos todos los datos y redirecciona al *backend* del servidor web para acabar el proceso de autenticación, con un código para que pueda obtener esa información de redis.

Sobre la creación de la imagen de asterisk, se puede ver detallado en «*Configuración de asterisk*» del trabajo de Antonio Ángel. Para la imagen del *backend* del servidor web, en el mismo trabajo, se puede ver detallado en «*Desarrollo del backend de la aplicación "Panel del residente"*».

Despliegue de los servicios

En esta sección se explicará como se ha distribuido los distintos componentes, como se persiste el almacenamiento de diversos componentes, el uso de la

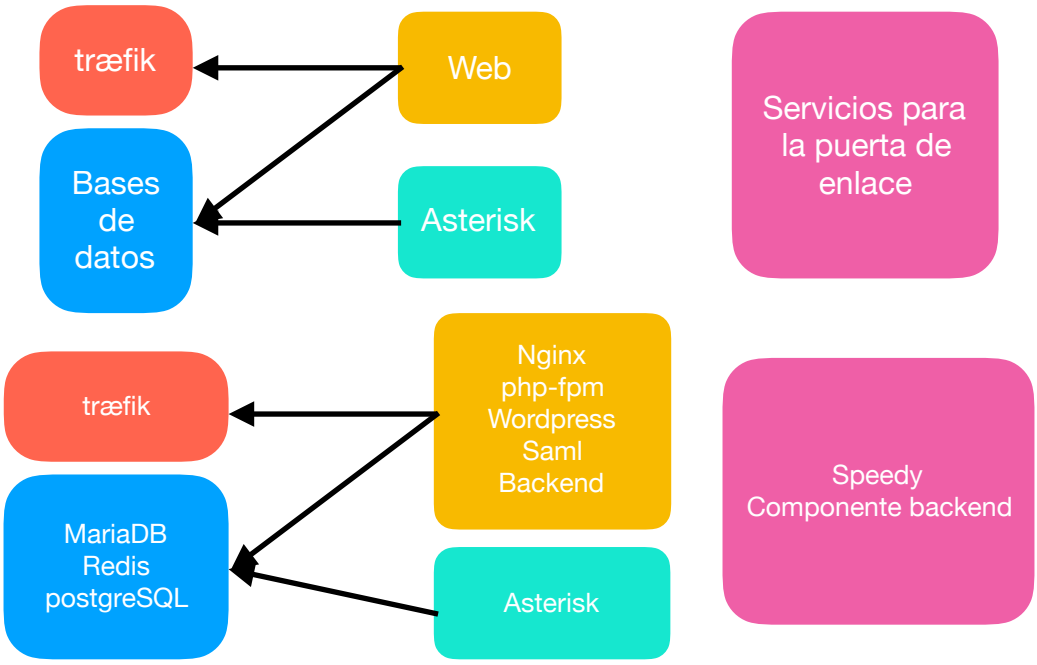
```
#T H E  D O M A I N
export DOMAIN=

#The path where the usual http files are located
export HTTP_PATH=
#The path where Wordpress files are located
export WORDPRESS_PATH=

#The path where the certificate `cert.pem` and the private key `key.key` are located
export SAML_MELLON_PATH=

#Root password for MariaDB
export MYSQL_ROOT_PASSWORD=
#The path where the database files will be stored
export MYSQL_DATA_PATH=
#Same as above, but for another DB
export REDIS_DATA_PATH=
#postgres password for PostgreSQL
export POSTGRES_PASSWORD=
#The path where the database files will be stored
export POSTGRES_DATA_PATH=
```

herramienta *docker-compose* y como usarlo en un despliegue en local y en uno en la residencia.

docker-compose es una herramienta de Docker que permite definir componentes y sus necesidades, y relacionar todos entre ellos, usando un simple archivo de definición. Con sencillos comandos, puedes ejecutar varios contenedores, y actualizarlos cuando haya cambios de configuración o una nueva imagen. Es una forma sencilla de poder ejecutar diversos contenedores y agruparlos.

Para este trabajo, se usará dicha herramienta para hacer los despliegues tanto de pruebas como el entorno real.

Para permitir distinguir entre diversos entornos donde se hace el despliegue, *docker-compose* nos permite usar variables de entorno para dar valor a partes del archivo de definición, con lo que para ciertos valores de los componentes que dependen de donde se despliegue se definirán con esas variables. De esta forma, no hay que modificar los archivos *docker-compose.yaml* (los de despliegue) para adaptarlos a cada entorno. Se usará un archivo plantilla, como el de la imagen anterior, que contenga todas las variables de entorno sin rellenar para su copia y uso en cada entorno.



Gráficos de dependencias entre grupos de servicios (arriba) y los servicios dentro de cada grupo (abajo)

Para el despliegue, se dividirán 5 grupos los distintos componentes de la siguiente forma:

- **Bases de datos**:
  - MariaDB
  - Redis
  - postgreSQL
- **Træfik**
- **Web**
  - nginx (para archivos estáticos y el *frontend* de la aplicación web)
  - php-fpm (para ejecutar scripts en PHP en nginx)
  - wordpress (apache + PHP para usarlo en Wordpress)
  - saml (modulo de autenticación SAML con la Universidad)
  - Backend aplicación web
- **asterisk**
- **Servicios para la puerta de enlace**
  - speedy (medidor de ancho de banda por dispositivo)
  - Componente externo del backend de la aplicación web

Cada grupo de servicios corresponde un un fichero *docker-compose.yaml* para el despliegue de los mismos.

‣ Træfik

El archivo de despliegue para træfik es ciertamente sencillo, pero crea la base para que los componentes que quieran publicar un servicio a través del proxy inverso, puedan hacerlo.

```yaml
version: '3'

services:
  traefik:
    image: traefik
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - ./traefik.toml:/etc/traefik/traefik.toml:ro
    ports:
      - "80:80"
      - "443:443"
      - "8080:8080"
    networks:
      - net

networks:
  net:
```

Hablamos de una red interna entre contenedores que conecta dichos servicios con træfik. De esta forma, dichos servicios solo son accesibles directamente (sin tener que pasar por træfik) desde los contenedores conectados a dicha red (llamada *traefik_net*).

El contenedor tiene un archivo de configuración (montado como volumen) que te permite preparar lo que la aplicación denomina "*entry points*" y el auto-descubrimiento de los servicios. Con *entry points* nos referimos a qué puertos va a escuchar el proxy inverso para ofrecer sus servicios. En este caso, usaremos el estándar puerto 80/tcp para HTTP, 443/tcp para HTTPS y el puerto alternativo 8080/tcp para el panel de métricas de træfik (definido en la sección *ports* en la captura de código). En cambio, el auto-descubrimiento permite reconfigurar el proxy inverso cuando aparece un nuevo contenedor con la configuración correcta y desconfigurarlo cuando se para.

Para su uso en Docker, hay que etiquetar los contenedores (en web se explicará cuáles son) con cierta configuración que incluye la regla que debe aplicarse a una petición para redirigir a ese servicio, la red interna (si el contenedor tiene más de una) y una para activar dicho contenedor para que el proxy inverso redirija las peticiones - en la configuración está puesta para que no haga eso por defecto, y permite filtrar contenedores con servicios web de los que no. Y para que todo esto funcione, necesitamos pasar el *socket* de Docker al contenedor de træfik, de ahí que en la sección de volúmenes se monte el *socket*.

▸ Bases de datos

Para el grupo de contenedores de las base de datos, se irá explicando por cada base de datos, y mostrando la parte correspondiente del archivo *docker-compose.yaml* para reducir las medidas de la imagen.

```yaml
mariadb:
  image: mariadb
  command: ["--character-set-server=utf8mb4", "--collation-server=utf8mb4_unicode_ci"]
  networks:
    - net
  volumes:
    - "${MYSQL_DATA_PATH}:/var/lib/mysql"
    - "./sql:/docker-entrypoint-initdb.d"
  environment:
    MYSQL_ROOT_PASSWORD: "${MYSQL_ROOT_PASSWORD}"
```

La base de datos MariaDB se despliega con la configuración de la imagen superior. Para empezar, la base de datos se inicia con unos argumentos (colocados en *commands*) para que use UTF8 como codificación de texto por defecto, en

concreto la codificación *utf8mb4*, que tiene soporte completo de la codificación, a diferencia de otros modos que en MySQL (y MariaDB) no soportan todo el estándar.
Para persistir los datos de la base de datos, usamos un volumen, en concreto lo que se denomina *bind mount* ya que enlaza algo existente dentro del contenedor. La ruta de los datos se encuentra dentro de una variable de entorno que se puede encontrar en el archivo comentado anteriormente.
También se monta la carpeta *sql* que se encuentra en el mismo lugar del archivo de despliegue. Los ficheros SQL de esa carpeta se ejecutarán en el servidor cuando se estén creando los archivos de la base de datos (cuando es la primera ejecución por ejemplo). Permite dar un valor inicial a la base de datos.
Por último, el usuario por defecto es *root* y la contraseña se define dentro de otra variable de entorno.

```
postgres:
  image: timescale/timescaledb:latest-pg10
  environment:
    POSTGRES_PASSWORD: "${POSTGRES_PASSWORD}"
  networks:
    - net
  volumes:
    - "${POSTGRES_DATA_PATH}:/var/lib/postgresql/data"
    - "./psql:/docker-entrypoint-initdb.d"
```

La siguiente base de datos es la de postgreSQL con la extensión de timescaleDB.
La configuración es parecida a la de MariaDB: una ruta donde persistir los datos, una carpeta para que al crear las estructuras de almacenamiento de la BD se ejecuten diversos SQL y la contraseña del usuario por defecto. En este caso, el usuario por defecto es *postgres*.

Por último tenemos redis. Al ser una base de datos tan sencilla, no hace falta contraseña. El volumen montado es de uso temporal ya que redis por defecto no persiste el contenido de la base de datos.

```
redis:
  image: redis
  networks:
    - net
  volumes:
    - "${REDIS_DATA_PATH}:/data"
```

Si hiciera falta, en [40] te explican como activarlo.

La red que crea este archivo de despliegue, para su uso fuera de él, es *databases_net*. Se usará para conectar otros contenedores a la red de bases de datos, tal y como veremos ahora en los próximos párrafos.

‣ Servicios web

Ahora describiremos la configuración de despliegue de la parte de la web. Al igual que en la anterior parte, se irá dividiendo en partes por cada uno de los contenedores.

```
nginx:
  image: web-nginx
  volumes:
    - "${HTTP_PATH}:/var/www/html:ro"
    - "ncache:/var/cache/nginx"
  networks:
    - webnet
    - traefik_net
  depends_on:
    - php-fpm
  labels:
    - "traefik.enable=true"
    - "traefik.backend=web"
    - "traefik.frontend.rule=Host:${DOMAIN}"
    - "traefik.docker.network=traefik_net"
```

El servidor nginx, para archivos estáticos, se define tal y como aparece en la imagen superior.

Lo primero son los volúmenes. El primero es la ruta, fuera del contenedor, donde se encontrarán los archivos estáticos que se servirán en nginx, cuyo valor se define mediante una variable de entorno. El segundo volumen es para una cache que tiene configurado nginx. En este caso usamos un volumen que gestiona enteramente Docker ya que no es necesario acceder a esos datos ni hacer copias de seguridad. El nombre *ncache* es el nombre que tiene el volumen.

En la sección de redes (*networks*) definimos a qué redes tiene acceso el contenedor, que en este caso es la red que conecta con træfik y una red interna de este despliegue. El último sirve para conectarse con php-fpm.
La sección *depends_on* define que este contenedor debe ejecutarse después de lo que haya dentro (en este caso, php-fpm).
La sección *labels* define las etiquetas del contenedor (siempre que no se despliegue en Docker Swarm o Kubernetes). Aquí definimos la configuración que usará træfik para configurar lo que ellos denominan el *backend* y el *frontend*. El *frontend* es la configuración que permite definir un servicio y el *backend* son los servidores disponibles que sirven ese contenido o servicio. De esta forma, puede haber más de un servidor para el mismo servicio que permite hacer distribución de carga y estar mejor protegido ante fallos. En lo que se puede observar de la imagen, la primera etiqueta dice a træfik que ese contenedor se puede usar para configurar un nuevo *backend* y/o *frontend*. La segunda le da un nombre al backend (para poderlo identificar). La tercera es la regla que permitirá identificar qué peticiones van directos a este servicio. *Host:${DOMAIN}* indica que todas las peticiones cuyo host (el dominio al que se accede) sea el de la variable de entorno *DOMAIN* entonces irá a ese servicio. El orden de aplicación de reglas, por defecto, es del más largo al más pequeño, y este al ser una regla pequeña se comprobará en las ultimas posiciones (menor prioridad). La última etiqueta sirve para decir cuál red es la que se conecta con træfik, para poder acabar de configurar el *backend*.

Para php-fpm, no se requiere de mucha configuración para hacerlo funcionar.

```
php-fpm:
  image: web-php-fpm
  volumes:
    - "${HTTP_PATH}:/var/www/html"
  networks:
    - webnet
    - databases_net
```

Montamos la misma ruta de archivos que en nginx, por si se desea ejecutar scripts en PHP, de esta forma coinciden las rutas entre nginx y php.
Las redes que se conecta son la red interna (para conectarse con nginx) y el de base de datos, por si se desea conectar con alguna de las bases de datos.

```
wordpress:
  image: web-apache-php
  volumes:
    - "${WORDPRESS_PATH}:/var/www/html/noticias-rafj"
  networks:
    - databases_net
    - traefik_net
  labels:
    - "traefik.enable=true"
    - "traefik.backend=wordpress"
    - "traefik.frontend.rule=Host:${DOMAIN};PathPrefix:/noticias-rafj"
    - "traefik.frontend.errors.network.status=404,500"
    - "traefik.frontend.errors.network.backend=web"
    - "traefik.frontent.errors.network.query=/errores/{status}.html"
    - "traefik.docker.network=traefik_net"
```

Para wordpress, usamos la siguiente configuración.

Primero, se monta la carpeta donde se encuentren los archivos de WordPress. Nótese que la ruta dentro del contenedor acaba en *noticias-rafj*. La ruta está mal escrita (*rafj* en lugar de *rajf* que viene de Residencia Alberto Jiménez Fraud) ya que cuando se configuró inicialmente, ni a Antonio ni a mi nos dimos cuenta del fallo y ahora cambiar la ruta de un WordPress funcionando es muy complicado.

Las redes a las que se conecta son la de las base de datos, para conectarse con MariaDB, y la de træfik, para poderse publicar.

Sobre las etiquetas, muchas ya se conocen, aunque se van a explicar los mas diferentes. Primero, la regla es más larga y contiene ahora un *PathPrefix*. Entre ambos, se separa mediante un punto y coma (;) que significa Y. Por tanto, esta regla al completo es "*todas las peticiones al dominio* ${DOMAIN} *y cuya ruta empiecen por* /noticias-rafj".

Las tres siguientes etiquetas, después de la regla, son para configurar unos archivos de error. Así, cuando salta un error 404 (no encontrado), un error 500 (error interno) o 502 (cuando el contenedor está parado - no aparece en la captura) se servirá lo que resulte de hacer la petición al nginx en la ruta */errores/404.html*, */errores/500.html* y */errores/502.html* respectivamente.

Ahora es el turno del módulo de autenticación SAML.

```
saml:
  image: saml
  volumes:
    - "${SAML_MELLON_PATH}:/mellon"
  networks:
    - databases_net
    - traefik_net
  labels:
    - "traefik.enable=true"
    - "traefik.backend=saml"
    - "traefik.frontend.rule=Host:${DOMAIN};PathPrefix:/mellon,/alberginia"
    - "traefik.frontend.errors.network.status=404,500"
    - "traefik.frontend.errors.network.backend=web"
    - "traefik.frontent.errors.network.query=/errores/{status}.html"
    - "traefik.docker.network=traefik_net"
```

En este contenedor, el volumen es para el certificado necesario para el correcto funcionamiento de la autenticación SAML. Se generan con un simple comando en la herramienta por terminal de OpenSSL, por ejemplo, y con eso ya sería suficiente. El certificado debe llamarse *cert.pem*, y la clave privada, *key.key*.

Las redes vuelven a ser la de base de datos y la de træfik. Como la autenticación usa redis para almacenar temporalmente unos datos, debe poderse conectar a él.

Por último, las etiquetas todas son conocidas, aunque se comentará la etiqueta de la regla. En este caso la parte de *PathPrefix* tiene dos valores separados por una coma (,) que viene a decir O. Esta regla significa "*las peticiones al dominio* ${DOMAIN} *que tengan como prefijo de ruta* /mellon *o* /alberginia".

```
backend:
  image: backend
  networks:
    - databases_net
    - traefik_net
  labels:
    - "traefik.enable=true"
    - "traefik.backend=api-backend"
    - "traefik.frontend.rule=Host:${DOMAIN};PathPrefix:/app-api"
    - "traefik.docker.network=traefik_net"
  restart: always
```

El backend de la aplicación web (que podéis ver más detalles en «*Desarrollo del backend de la apliación "Panel del residente"*» en el trabajo de Antonio Ángel) necesita conectarse a todas las base de datos desplegadas (postgreSQL, MariaDB y redis), por lo que se le ha asociado a la red de base de datos. Además, como se debe publicar la API dentro de la web, se tiene que añadir las etiquetas correspondientes (*labels*) para que træfik lo detecte y configure un camino para poder publicar dicha API.

```
networks:
  webnet:
  traefik_net:
    external: true
  databases_net:
    external: true

volumes:
  ncache:
```

Por completitud, comentaré también la parte final del archivo *docker-file.yaml.*
En él podemos ver como se definen las distintas redes, donde la de træfik y la de las bases de datos se marcan como externas ya que no los maneja este a través de este grupo.
Los volumenes que maneja Docker se tienen que definir también en el archivo.

- Asterisk

```
version: 3

services:
  asterisk:
    image: asterisk
    network_mode: host
    cmd: ["asterisk", "-fv"]
    environment:
      MYSQL_HOST: "127.0.0.1"
      MYSQL_USER: "asterisk"
      MYSQL_PASS: "4st3r1sk"
      MYSQL_DATABASE: "asterisk"
    restart: always
```

La centralita de telefonía por IP (VoIP) asterisk se desplegará como su propio grupo de servicios. Un requisito indispensable para el funcionamiento correcto de asterisk es que debe usar el modo de red *host* (opción *network_mode*), en el que realmente es deshabitar el aislamiento de redes y el contenedor tiene acceso a las redes del

servidor directamente. El contenedor ejecuta por defecto una shell de comandos, por lo que para ejecutar asterisk tenemos que decirle exactamente que debe ejecutar. Eso se consigue en la opción *cmd*. El valor es una lista de argumentos que el contenedor interpretará y ejecutará. Con la opción "*-f*" de asterisk conseguimos que funcione sin que el contenedor se detenga, ya que el servidor se ejecuta en lo que se conoce como "*foreground*" (el proceso que ejecutas es el servidor, en lugar de crear otro proceso que se ejecutará de fondo que realizará esa tarea). En caso de no ponerlo, asterisk crea un segundo proceso donde ejecutará el servidor de fondo, y el que se ha creado por motivo de ejecutar el comando se cierra. Al cerrar ese proceso, que en Docker es el principal, el contenedor se para ya que solo tiene en cuenta dicho proceso como comprobación del estado, e ignora procesos secundarios. Por último, queda configuración para que la centralita se pueda conectar con la base de datos MariaDB, necesaria para almacenar cierta información.

Más detalles sobre la configuración de la centralita la podéis encontrar en «Configuración de asterisk» del trabajo de Antonio Ángel.

▸ Servicios en la puerta de enlace

```
version: '3'

services:
  speedy:
    image: melchor9000/speedy
    cap_add:
      - NET_RAW
      - NET_ADMIN
    network_mode: host
    command: speedy -device enp4s5 -db timescaledb -db-url postgres://postgres:${POSTGRES_PASSWORD}@10.10.10.253
/speedy?ssl=disable
    restart: always

  backend-gw-part:
    image: backend-gw-part
    build:
      context: ./
    command: -l /dnsmasq.leases -a 10.10.10.253
    volumes:
      - "/var/lib/misc/dnsmasq.leases:/dnsmasq.leases:ro"
    network_mode: host
```

Para terminar, hablaremos de los contenedores que se desplegarán en el servidor/puerta de enlace, que es distinto al resto de contenedores. En este, como se comentó, tiene el medidor de tráfico por dispositivo (*speedy*) y, como se puede observar en la imagen, también tiene un trozo del *backend* de la aplicación web.

El primer contenedor, para su correcto funcionamiento, debe poder ver las tarjetas de red del servidor, de ahí que el modo de red (*network_mode*) sea host. Además, para que sea capaz de capturar el tráfico, hay que añadirle unos permisos especiales para ello. Estos se encuentran en *cap_add*.
Por ultimo, la forma de configurar el programa es mediante argumentos. En el valor *command* tenemos la lista de argumentos completa.

El segundo contenedor, una parte del *backend*, permite al *backend,* que programa Antonio, obtener cierta información solo disponible en dicho servidor. La imagen la genera en el momento de hacer el despliegue (por tener el atributo *build*).
También necesita unos argumentos para funcionar, pero en este caso no se incluye el nombre del programa porque en la imagen se fijó que siempre se ejecutará dicho programa con el contenedor (seria usar *ENTRYPOINT* en lugar de *CMD* en el archivo Dockerfile).
También usamos la red de tipo host, pero en este caso el motivo es que por tener este servidor un firewall, Docker no es capaz de crear sus reglas propias y no es capaz de funcionar correctamente si no se usa esa red.

6. Conclusiones y trabajos futuros

En la realización de este trabajo tanto mi compañero de grupo Antonio Ángel como yo hemos aprendido mucho en su realización. Hemos visto como partir de no tener una infraestructura inalámbrica que dificultaba la conexión a internet (y los estudios), a tener un sistema completo, con una conexión bastante buena y con servicios que ayudarán a los residentes con sus necesidades y problemas.

Primeramente, el diseño de la instalación de cable e inalámbrica ha sido un éxito. La parte más compleja y novedosa era la instalación inalámbrica. En la que, partiendo de una instalación prácticamente nula, hemos aplicado conocimientos de la asignatura optativa *Redes Inalámbricas* y la ayuda ofrecida desde el Servicio Central de Informática de la Universidad para poder realizar un diseño de calidad. El resultado ha sido bastante satisfactorio, tanto para nosotros como para los residentes que han podido probarlo. Incluso tiene una cobertura inalámbrica en el exterior de los apartamentos bastante suficiente para casos de uso como mensajería, que inicialmente no se tenía en cuenta. El sistema al completo, con los switches y puntos de acceso, se puede controlar desde un mismo panel, algo que tampoco teníamos pensado inicialmente entre los objetivos, pero que ha dado mayor comodidad para configurar todo y, sobre todo, monitorizar.

Con esos servicios básicos de conectividad funcionando, dan cabida al uso de otros servicios de todo tipo que usen la infraestructura de red. Con lo que, agrupamos y mejoramos los servicios que estaban disponibles en la residencia, dando más comodidad al residente al acceder a todos ellos. Con la aplicación web, hemos dado un aspecto común a todos los servicios, sin contar WordPress que tiene un diseño algo distinto; además hemos podido protegerlos para que lo puedan usar usuarios de la comunidad universitaria y residentes. Con el diseño modular de la parte *frontend* de la aplicación, permite fácilmente añadir futuros módulos e incluso permite usar otros servidores para APIs distintas (siempre que definan los permisos correspondientes para que los usuarios puedan acceder a la página). Durante el desarrollo, usando el sistema modular, cada vez que creaba un módulo, solo tenía que encargarme de programar las páginas de este y la forma de comunicarse con la API. De una forma cómoda y sin muchas complicaciones, se ha

podido ir definiendo páginas nuevas y módulos nuevos, aunque las tecnologías usadas no son lo más fácil del panorama actual. Esperemos que los residentes nuevos disfruten con esta reorganización de servicios y se encuentren más cómodos con todos ellos.

Por último, el despliegue de los servicios es algo importante. Cada vez más, los despliegues tornan más complejos y dependen de muchos componentes. El uso de la virtualización con contenedores ha ayudado a reducir la dificultad de la infraestructura de servicios y su mantenimiento. Además de que permite poder probar algunos de los componentes en otros ordenadores (en entornos de pruebas y desarrollo) para mejorarlos, arreglar fallos o hacer pruebas. La actualización de los paquetes está a manos de los administradores del despliegue, por lo que uno tiene tiempo para poder probar los cambios antes de aplicarlos, para evitar problemas. Este es un motivo por lo que muchas veces un software deja de funcionar o da fallos al actualizar el sistema operativo (o distribución Linux). Al ser paquetes independientes al sistema operativo, el software siempre se ejecutará dentro de ese entorno enjaulado. En mi opinión, ayuda mucho a poder mantener un despliegue con menor dificultad, una vez que ya lo tienes montado.

Dentro de este proyecto se abre una nueva etapa para los futuros becarios de informática en la Residencia Universitaria Alberto Jimenez Fraud. Estas personas tendrán una infraestructura fácil de gestionar y de actualizar. Aunque tengan que aprender bastante de la infraestructura, no será tan difícil como en nuestros años como becarios (Antonio Ángel y yo). Además, si desean añadir nuevos servicios a la aplicación web, ahora lo tienen más fácil que nunca, y es algo que invitamos a hacer, ya que en la residencia, hay bastantes cosas que se pueden informatizar, pero también habrán nuevas ideas a añadir igual de interesantes o más que estarían bien añadir. Es un mundo que se está informatizando enormemente, poder acceder a información, realizar gestiones o ver contenido desde el movil u ordenador es increíblemente cómodo y nos ahorra tiempo.

7. Referencias

[1] arXiv:1501.01496 [cs.NI] - Bellalta, B. (2015). IEEE 802.11ax: High-Efficiency WLANs.

[2] https://www.docker.com - Web de Docker (*Sistema de ejecución de contenedores*)

[3] https://docs.docker.com/engine/swarm/ - Documentación de Docker swarm

[4] https://www.typescriptlang.org - Web de TypeSript (*lenguaje de programación basado en JavaScript, pero con tipos*)

[5] https://reactjs.org - Web de react.js (*biblioteca para interfaces web en JS*)

[6] https://redux.js.org - Web de redux (*biblioteca de lógica para interfaces web reactivo*)

[7] http://getbootstrap.com - Web de Bootstrap (*framework para diseño web*)

[8] https://firehol.org - Web para firehol y fireqos (*herramientas que simplifican la creación y manejo de firewall y QoS en Linux*)

[9] Para iptables, se puede consultar esta entrada de la wiki de ArchLinux - https://wiki.archlinux.org/index.php/Iptables_(Español) - o la web del proyecto *netfilter* - https://netfilter.org

[10] La herramienta *tc* es parte del paquete de herramientas de red del kernel de linux, en esta entrada de la wiki de ArchLinux puede informarse de como funciona https://wiki.archlinux.org/index.php/Advanced_traffic_control

[11] http://www.thekelleys.org.uk/dnsmasq/doc.html - Web de dnsmasq

[12] Descripción del producto, ver sección “Mejor Rendimiento Inalámbrico” - https://www.tp-link.com/es/products/details/TL-WR841N.html#overview - 3 de agosto de 2018

[13] Monica Beloki, Cuidado con las Conexiones WIFI, 30 septiembre de 2015 - http://grupodatcon-norte.com/red-wifi-abierta/

[14] Ver referencia de noticia: David Pérez, WiFi gratis y redes públicas: posibles problemas y consejos para estar seguros, 17 abril de 2017 - https://elandroidelibre.elespanol.com/2017/04/redes-wifi-publicas-peligros-consejos.html

[15] Ver referencia de noticia: Jorge Sanz Fernández, ¿Tienes problemas con el WiFi al usar Chromecast? Google ya tiene la solución, 12:03 28 enero de 2018 - https://cincodias.elpais.com/cincodias/2018/01/18/lifestyle/1516269614_473828.html

[16] Aunque no se detallará en este trabajo, en el trabajo de Antonio Ángel Cruzado Castillo se explica el por qué y el funcionamiento de los teléfonos VoIP.

[17] Información obtenida de las especificaciones oficiales, página 8 del documento, visto el 10 de agosto 2018 - https://dl.ubnt.com/datasheets/unifi/UniFi_AC_APs_DS.pdf

[18] Multicast WiFi Problem Statement, M. McBride & C. Perkins, 26 octubre 2017 - https://tools.ietf.org/id/draft-mcbride-mboned-wifi-mcast-problem-statement-01.html

[19] Más información sobre la red de tipo *macvlan* de Docker: https://docs.docker.com/network/macvlan/

[20] Referencia de imagen de Docker con las características de adopción en capa 2: https://github.com/jacobalberty/unifi-docker#layer-2-adoption

[21] http://nginx.org/en/ - Web de nginx (*Servidor y proxy inverso web*)

[22] https://traefik.io - Web de træfik o traefik (*Proxy inverso para la nube*)

[23] https://es.wordpress.org - Web de WordPress (*aplicación web de gestión de contenido* - CMS)

[24] https://www.mysql.com - Web de MySQL (*base de datos relacional*)

[25] https://mariadb.com - Web de MariaDB (*base de datos relacional basado en MySQL*)

[26] https://www.postgresql.org - Web de PostgreSQL (*base de datos relacional*)

[27] https://www.timescale.com - Web de timescaleDB (*extensión de base de datos basado en datos temporales para PostgreSQL*)

[28] http://netdata.firehol.org - Web de netdata (*monitor en tiempo real en un ordenador y sus servicios*)

[29] *titpetric* en la página de la imagen de Docker, 16 de agosto 2018 - https://hub.docker.com/r/titpetric/netdata/

[30] Mapa de la aplicación *Mapas* de Apple - 5 agosto 2018

[31] Docker, 17 de agosto 2018 - https://www.docker.com/resources/what-container

[32] R. V. Nee, "Breaking the Gigabit-per-second barrier with 802.11AC," in IEEE Wireless Communications, vol. 18, no. 2, pp. 4-4, April 2011. doi: 10.1109/MWC.2011.5751287

[33] Servicio de Docker para la búsqueda de imágenes oficiales o de la comunidad

[34] https://store.docker.com/images/nginx - Página de la tienda de Docker para la imagen de nginx

[35] Ver sección 6.1.1. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext transfer protocol--HTTP/1.1* (No. RFC 2616).

[36] https://store.docker.com/images/php - Página de la tienda de Docker para la imagen de PHP

[37] https://secure.php.net/manual/en/opcache.installation.php

[38] https://docs.docker.com/compose/overview/ - Documentación de docker-compose (herramienta que permite definir diversos componentes y conectarlos entre ellos)

[39] https://store.docker.com/images/mariadb - Página de la tienda de Docker para la imagen de MariaDB

[40] https://store.docker.com/images/redis - Página de la tienda de Docker para la imagen de redis

[41] https://store.docker.com/community/images/timescale/timescaledb - Página de la tienda de Docker para la imagen de timescale/timescaledb

[42] https://store.docker.com/images/postgres - Página de la tienda de Docker para la imagen de postgreSQL

[43] https://github.com/melchor629/speedy - Repositorio del software medidor del uso de ancho de banda

[44] https://wiki.oasis-open.org/security/FrontPage - Web/Wiki de SAML

[45] https://github.com/Uninett/mod_auth_mellon - Repositorio del módulo de Apache HTTPD para autenticación mediante SAML

[46] https://github.com/wmonk/create-react-app-typescript - Repositorio de create-react-app-typescript

[47] https://angular.io - Web de Angular (*framework de interfaces de usuario, que integra todo el código y las herramientas necesarias para hacer una interfaz basado en el patrón Modelo-Vista-Controlador*)

[48] https://www.jetbrains.com/webstorm/ - IDE de JetBrains para el desarrollo de interfaces web

[49] https://code.visualstudio.com/ - IDE de Microsoft genérico, al que se le pueden insertar complementos para desarrollo web

[50] https://sass-lang.com - *CSS with superpowers*

[51] https://unix.stackexchange.com/a/198591 - What is bind mount? (respuesta en Unix & Linux Stack Exchange)

[52] http://man7.org/linux/man-pages/man8/mount.8.html - Manual de *mount* en Linux, incluye sección para *bind mounts* llamada "*Bind mount operation*"

[53] https://bindfs.org - Implementación de *bind mount* para sistemas otros operativos mediante FUSE (útil para macOS o FreeBSD)

[54] https://git-scm.com/docs/gitsubmodules - Descripción y documentación de los submódulos en el sistema de control de versiones git

[55] https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md - Especificación de OpenAPI 2.0/swagger

[56] http://getbootstrap.com/docs/4.1/components/buttons/

[57] http://getbootstrap.com/docs/4.1/components/modal/

[58] http://getbootstrap.com/docs/4.1/components/collapse/

[59] https://help.ubnt.com/hc/en-us/articles/115002806907-UniFi-High-Density-WLAN-Scenario-Guide (6 de agosto 2018)

[60] Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.