UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

18|19

Trabajo fin de grado

Solución P2P embebida

Sergio Navarro Sánchez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Solución P2P embebida

En colaboración con Vaelsys formación y desarrollo S.L

Autor: Sergio Navarro Sánchez
Tutor: Jorge Abrines Bendayan

junio 2019

Sergio Navarro Sánchez
Solución P2P embebida

Sergio Navarro Sánchez

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres y mi novia

Cualquier tonto puede escribir código
que un ordenador entiende.
Los buenos programadores escriben código
que los humanos pueden entender.

Martin Fowler

Agradecimientos

Me gustaría destacar todo el apoyo que me han dado mis padres y mi novia en los momentos difíciles a la hora de realizar este proyecto y a lo largo de la carrera.

A mis compañeros, que me han acompañado a lo largo de todos estos años y hemos podido compartir buenos y malos momentos.

También destacar a mis profesores que me han aportado los conocimientos necesarios para poder finalizar el proyecto y afianzar una base de los mismos que me servirán a la hora de incorporarme al mundo laboral.

A mi tutor de proyecto que me ha prestado la ayuda necesaria para poder cumplir todos los objetivos previstos.

Por último, al resto de mis familiares y amigos por preocuparse y apoyarme en mis estudios, en especial a Poncho que siempre ha estado a mi lado en los momentos complicados.

Resumen

El tráfico de datos en la red se ha convertido en una tarea cotidiana en el mundo actual, debido a este efecto, es necesario que las tecnologías de red evolucionen de forma continua en función de los requisitos solicitados por los usuarios.

Uno de los principales obstáculos que podemos encontranos en el uso de la red es la estructura en sí misma de Internet: el equipo que participó en su desarrollo no la concibió como una red extensa, por lo que únicamente reservaron 32 bits de direccionamiento público, cifra que a día de hoy queda muy lejos del número de usuarios que la utilizan. Para solucionar esta limitación, se han ido desarrollando protocolos de red que nos permiten aprovechar al máximo los instrumentos de los que disponemos. Además, debemos tener en cuenta otros factores, como el de ofrecer una total seguridad a la hora de tratar y enviar los datos, asegurándonos de que éstos llegan de forma correcta a su destinatario, sin haber sido interceptados o modificados por personas ajenas en el transcurso de la transmisión de dicha información.

A pesar del uso de los recursos de la red de los que ya se disponían, en los últimos años se ha producido entre las empresas desarrolladoras de software la tendencia de dar una mayor relevancia a los problemas de seguridad, ya que han ido surgiendo al incrementarse el uso de la red; así como el conocimiento sobre ella entre los usuarios. Por ello, ha aumentado tanto la concienciación como la necesidad de los equipos de desarrollo a la hora de implementar o actualizar sus productos para subsanar estos problemas.

Al estar realizando el Trabajo de Fin de Grado con una empresa desarrolladora de software, he adquirido una mayor conciencia de las preocupaciones que actualmente existen y de reconocer la importancia de las mismas, permitiéndome dar un enfoque diferente al trabajo.

Si nos centramos en las actividades que realiza la empresa, podemos resumir los objetivos del proyecto en el desarrollo de una librería que permita la conexión de dispositivos a través de la red indistintamente de los recursos de los que se disponga, evitando así el riesgo que se genera en el intercambio de datos.

Para ello dispondremos de un cliente, que permitirá al usuario utilizar la librería de manera transparente; y un servidor de descubrimiento, que le ofrecerá al cliente la dirección IP y puerto con el objetivo de localizar el servidor donde se encuentran los periféricos y sus datos generados. En el desarrollo del cliente se ha empleado el framework de Python Kivy; el servidor de descubrimiento se ha implementado con el framework de Python Django y por último; para el servidor y la librería se ha empleado el framework de C++ Qt.

Palabras clave

Red, Internet, direcciones limitadas, seguridad, Trabajo Fin Grado, Python, Kivy, Django, C++, Qt

Abstract

Data traffic on the network has become a daily task in today's world, due to this effect, network technologies need to evolve continuously depending on the requirements requested by users.

One of the main obstacles that we can find in the use of the network is the structure itself of the Internet: the team that participated in its development did not conceive it as an extensive network, so they only reserved 32 bits of public address, a figure that is far from the number of users who use it today. To solve this limitation, network protocols have been developed to allow us to take full advantage of the instruments available to us. In addition, we must take into account other factors, such as providing total security when processing and sending the data, ensuring that the data arrive correctly at their destination, without having been intercepted or modified by outsiders transmission of such information.

Despite the use of the network resources that were already available, in recent years there has been a tendency among software developers to give greater relevance to security problems, as they have emerged as the use of the network; as well as knowledge about it among users. Therefore, both awareness and the need for development teams to implement or update their products to address these problems have increased.

By doing the Final Degree Project with a software developer, I have become more aware of the concerns that currently exist and to recognize the importance of them, allowing me to take a different approach to work.

If we focus on the activities carried out by the company, we can summarize the objectives of the project in the development of a library that allows the connection of devices through the network regardless of the resources available, thus avoiding the risk generated in data exchange.

To do this we will have a client, which will allow the user to use the library in a transparent way; and a discovery server, which will provide the client with the IP address and port in order to locate the server where the peripherals are located and their generated data. The Kivy Python framework has been used in client development; the discovery server has been deployed with the Django Python framework and finally; the C++ Qt framework has been used for the server and library.

Keywords

Network, Internet, limited directions, security, Final Degree Project, Python, Kivy, Django, C++, Qt

Índice

Listas

Lista de algoritmos

Lista de códigos

Lista de cuadros

Lista de ecuaciones

Lista de figuras

Lista de tablas

Lista de cuadros

1

Introducción

1.1. Motivación

Hoy en día vivimos en un mundo interconectado a través de la red. Por ello tenemos que lidiar con los obstáculos que existen en Internet desde su creación, la cantidad de direcciones disponibles y la seguridad de los datos transmitidos a través de ella y las conexiones entre los dispositivos. El mismo Vint Cerf, uno de los creadores de Internet, hizo alusión a estas carencias: “Es posible que estos hayan pasado a ser problemas después de un crecimiento inesperado o que nadie pensara que el producto iba a ser usado de esa forma” [1]. Por lo que intentar solucionar o reducir en la mayor medida posible dicha problemática supone un reto interesante como objetivo principal del proyecto.

Cuando hablamos sobre la limitación de recursos de la red, nos referimos a que en IPV4 disponemos de 32 bits de direccionamiento para IPs públicas, por lo que existen aproximadamente 4.000 millones de direcciones, cifra muy pequeña en comparación con el número de dispositivos que emplean la red. Este efecto, es denominado “agotamiento de IPs públicas”, problema que se ha ido agravando con la llegada del internet de las cosas. Como solución, se diseñó el protocolo NAT, que consiste en mapear la dirección IP pública asignada frente a un rango de direcciones privadas. Las ventajas que ofrece el protocolo, se pueden resumir en: permite la conexión de múltiples dispositivos con una única IP pública; ofrece una capa de seguridad extra a nivel IP, ya que las IP privadas solo son visibles en el interior de la red y por último; posee un mantenimiento sencillo, dado que solo es necesario gestionar la tabla de enrutamiento. La utilización del protocolo lleva consigo un problema: antes de enviar un paquete a la red, es necesario modificar la IP y puerto de origen privado por los de la dirección pública, siendo necesario modificar la cabecera cada vez que se transmite o recibe un paquete de la red.

Cuando tratamos la seguridad de las conexiones, estamos mencionando a los puntos de acceso a la red, estos son zonas vulnerables a los ataques cibernéticos, que si no se controlan y se configura de forma correcta es probable que nos puedan robar la información a través de ellos. La esencia de este proyecto es emplear la red P2P para que abra los puertos solo cuando sea necesario y cada vez uno diferente, lo que incrementa el nivel de seguridad frente a estos ataques.

A partir de toda la problemática expuesta, se quiere encontrar una solución enfocada al ámbito de la videovigilancia. La finalidad del proyecto es ofrecer un sistema que permita realizar conexiones entre clientes y grabadores o cámaras, ofreciendo un punto de acceso sencillo y seguro de utilizar, para que los clientes pertenecientes al sistema puedan recuperar las grabaciones e imágenes de seguridad disponibles en los grabadores.

Para ello, las características que deberá poseer el sistema serán: robustez, se debe evitar el acceso de personas ajenas al sistema; eficiente, ha de ofrecer un servicio que pueda ser desplegado mediante una única dirección IP; y transparente, ha de brindar un método de configuración sencillo, enfocado a personas sin conocimientos de la materia.

1.2. Objetivos

El objetivo principal del proyecto es permitir la conexión de un cliente con un servidor de manera transparente, esto es de manera segura y óptima sin la necesidad de disponer de conocimientos sobre el funcionamiento de la red. Centrándonos en los requisitos que deben cumplir los componentes, éstos deben ser:

-Cliente:

Control de acceso: Para utilizar la aplicación el usuario deberá estar registrado en la misma, evitando que personas ajenas accedan a la plataforma.

Definición de roles: Se dispondrá de dos tipos de usuario: el usuario (básico) que sólo podrá utilizar los servicios de la aplicación; y el administrador, que podrá utilizar los servicios y además tendrá permiso para gestionar tanto usuarios como grabadores.

Uso de la biblioteca: La aplicación deberá ser capaz de utilizar todas las funcionalidades de la biblioteca, así como de gestionar las diferentes salidas de la misma.

Gestionar la información: Se deberá almacenar la información (datos, usuarios y grabadores) en el sistema de manera eficiente y organizada.

Interfaz intuitiva y amigable: La aplicación deberá ser fácil de utilizar por cualquier usuario.

Multiplataforma: La aplicación podrá ser utilizada en diferentes sistemas operativos (SSOO).

-Biblioteca:

Transparencia: Esta parte debe ser invisible a ojos del usuario, de manera que

no tiene conocimiento sobre la existencia ni la funcionalidad de la misma.

Funcionalidad: Realizará las acciones de intermediario entre el usuario, el servidor de descubrimiento y el cliente.

-Servidor (Grabador):

Registro: Debe ser capaz de registrarse en el servidor de descubrimiento, ofreciendo un nombre que lo identifique y una manera para localizarlo, esto es a través de su dirección IP y puerto.

Mantenerse actualizado: En el momento en el que uno de los atributos anteriores varíe, este cambio deberá notificarse al servidor de descubrimiento para que actualice ese atributo.

Gestión de la información: Se almacenarán de manera eficiente y organizada los datos (vídeos e imágenes) generados por las cámaras de vigilancia.

Distribución de la información: El servidor deberá atender cualquier petición generada por un cliente: le ofrece el recurso correspondiente a esa petición siempre que exista; y en caso contrario, le indica el motivo por el que no es capaz de proporcionarle ese recurso.

-Servidor de descubrimiento:

Gestión de los datos: Deberá almacenar todos los datos empleados en los componentes citados anteriormente.

Gestión de los servidores: Ofrece al cliente la IP y el puerto donde localizar al servidor y el estado en el que se encuentra en el momento previo a la conexión.

Gestión de las conexiones de datos: Deberá generar la conexión de datos con el grabador, obteniendo un punto de acceso al mismo por el que el cliente solicitará los recursos.

1.3. Organización de la memoria

La memoria se compone de las siguientes secciones, exponiéndose los pasos que se han realizado hasta llegar al desarrollo final del proyecto.

Estado del arte: en la primera sección se documenta el trabajo de investigación realizado para la toma de decisiones: se ha tenido en cuenta aplicaciones ya existentes que intentan solucionar el problema de internet para obtener una idea inicial sobre cómo abordar el diseño del sistema y la implementación de posibles mejoras; los frameworks que usan; librerías y otros programas en los que se basan, permitiéndome tomar elecciones sobre

los lenguajes de programación, frameworks y librerías que se adaptan mejor al objetivo del proyecto.

Diseño: esta sección contiene la documentación básica sobre el proyecto. En ésta se describen tres pilares del proyecto: el ciclo de vida; la definición de los requisitos y las pautas a seguir para cumplirlos.

Desarrollo: la siguiente sección expone cómo se van a desarrollar las decisiones tomadas en la fase de diseño y el funcionamiento de cada uno de los componentes.

Integración, pruebas y resultados: en esta sección se analiza si el producto final cumple los requisitos definidos, a partir del uso de pruebas de caja negra y blanca de cada uno de los componentes, tanto de manera individual como global.

Conclusiones y trabajo futuro: en la última sección se detallan posibles mejoras del proyecto, optimizando su funcionamiento o aportando nuevas funcionalidades para asegurar la vida del mismo.

2

Estado del arte

Para el desarrollo de esta sección, se ha dividido en dos bloques:

En el primero se ofrece un estudio del mercado, a partir del cual se extraerá información sobre aplicaciones similares que emplean la misma tecnología que se quiere aplicar en el desarrollo del proyecto. De esta forma, se podrá obtener una idea global de los posibles pasos a seguir durante la implementación del proyecto.

Por otro lado, el segundo bloque nos aportará los conocimientos necesarios sobre las tecnologías existentes que puedan adaptarse a los objetivos inicialmente propuestos. Para ello se tendrá en cuenta que las herramientas ofrezcan la funcionalidad requerida y sean compatibles entre ellas, simplificando y optimizando el funcionamiento del sistema de la mejor forma posible.

2.1. Redes peer to peer

Arquitectura de red distribuida donde todos los elementos son vistos como nodos, generando que la percepción de cliente/servidor desaparezca, ya que cada uno de ellos realiza las dos funciones. Estas redes fueron diseñadas para compartir tanto información como recursos de cualquier tipo (audio, vídeo, texto, ...) sin la necesidad de disponer de un sistema central o servidor, ya que cada nodo comparte sus recursos con el resto de la red al mismo tiempo que solicita al resto de nodos los recursos que poseen.

Figura 2.1: Ejemplo de red P2P [2]

Las ventajas que ofrecen este tipo de redes son:

Redes descentralizadas: eliminan la necesidad de disponer de un servidor central que distribuya la información, debido a que solo existen nodos con las mismas funciones y estos pueden situarse en cualquier punto de la red.

Robustez: en caso de que algún nodo se desconecte o falle, no afecta al funcionamiento de la red, aunque en casos excepcionales sí puede afectarse: si ese nodo es el único que dispone del recurso solicitado.

Optimización de recursos: Eliminan el problema de cuello de botella que se genera en los servidores, ya que al estar entrelazados los nodos, permite administrar y optimizar el ancho de banda de cada uno de ellos, dado que el recurso se solicita a varios nodos.

Escalabilidad: Cuanto mayor sea el número de nodos del que disponga la red, mejor será el funcionamiento de la misma, debido a que la redundancia de los recursos aumenta, permitiendo solicitar los recursos a mayor número de nodos.

Los principales inconvenientes de las redes P2P son:

Seguridad: es muy complicado determinar si un nodo de la red es malicioso, esto es: que aprovecha este tipo de redes para ofrecer contenido infectado; espiar al resto de nodos u ofrecer recursos con derecho de autor no regulados.

Consumo excesivo del ancho de banda: si no se configura de manera correcta, todo el ancho de banda se asignará a la red generando que el resto de procesos se ralenticen o se queden sin servicio.

Tamaño de la red: Si el número de nodos es muy reducido, no se puede garantizar la disponibilidad de la red, ya que los nodos se conectan y desconectan cuando el usuario lo solicita.

Direcciones IP dinámicas: Los nodos se conectan desde redes locales, por lo que es necesario configurar la NAT y el router para especificar un punto de acceso.

2.1.1. Tipos de redes P2P

Se pueden clasificar en función de su arquitectura o grado de centralidad, permitiendo decidir si la gestión de la red se realiza a través de uno o varios nodos centrales, o por el contrario, todos los nodos realizan las mismas tareas, y ellos mismos son los encargados de gestionar sus conexiones entrantes y salientes:

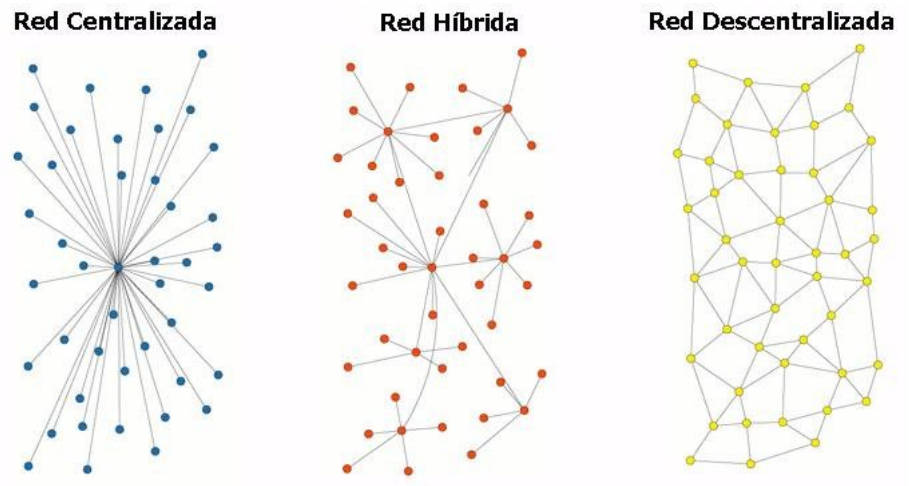


Figura 2.2: Tipos de redes P2P [2]

Centralizadas

Diseñada con una arquitectura monolítica, formada por nodos exteriores y un nodo central. El cometido del nodo central es almacenar la localización de cada nodo, junto con los recursos que dispone. Esto permite realizar las conexiones entre pares de nodos exteriores en función de los recursos solicitados y la cercanía de los mismos. Sus principales desventajas son: la escalabilidad, ya que toda la red depende del nodo central; y la robustez, debido a que, si el nodo central queda fuera de servicio, toda la red queda incomunicada. [3]

Híbridas

Es una implementación derivada de la centralizada 2.1.1, su funcionamiento se basa en la creación de subredes compuestas por nodos exteriores y uno central. En este tipo de arquitecturas, el nodo central actúa como hub, administrando los recursos de ancho de banda, acceso al resto de subredes y comunicación entre nodos, sin la necesidad de almacenar la información de los nodos exteriores, ya que son éstos los encargados de almacenar dicha información cuando se genera una conexión directa entre ellos. Esta arquitectura ofrece mayor robustez: en caso de que el servidor central falle, los nodos exteriores conectados a éste siguen funcionando, debido a la conexión directa que existe entre ellos. [3]

Descentralizada

Este tipo de arquitectura es la más extendida, ya que no es necesaria la implementación de un servidor central que actúe como gestor de la red o subred. Su funcionamiento se basa en la conexión directa de los nodos exteriores: cuando ésta se produce, cada nodo deberá mantenerla abierta y almacenar la información del nodo al que se conecta. [3]

2.1.2. Conclusión

Tras haber realizado el estudio de las diferentes arquitecturas, la que más se ajusta a las necesidades del proyecto es la centralizada. La necesidad de generar una red que permita controlar quién accede a un determinado nodo, se adapta a la funcionalidad que ofrece el servidor central. Sus funciones serán: gestionar las conexiones de la red y mantener la persistencia de los datos referentes a los nodos exteriores. Además, como las redes que se van a generar son de tamaño mínimo, no se va a tener la necesidad de generar otras subredes.

2.2. Técnicas para conectar nodos

Tras haber decidido el tipo de arquitectura a emplear, la siguiente tarea es decidir la manera en que los nodos van a generar la conexión de datos entre ellos.

2.2.1. Port knocking

También denominado llamar al puerto: consiste en abrir un determinado puerto, previamente cerrado por el firewall del SO. Su funcionamiento consiste en definir una secuencia de puertos a modo de contraseña de acceso. En el momento que un nodo intente acceder al sistema, para que el firewall abra el puerto, deberá realizar intentos de conexión sobre los puertos definidos como password en un orden y franja de tiempo establecidos en la configuración del firewall. [4]

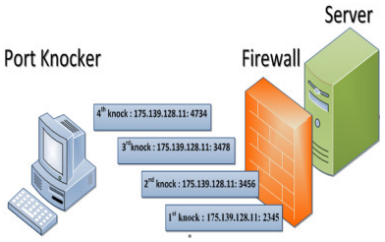


Figura 2.3: Ejemplo de acceso por port knocking [5]

Las ventajas que ofrece esta técnica son:

Autenticación silenciosa: la apertura del puerto se produce mediante conexiones al sistema.

Robustez: aunque se haga un rastreo de puertos, no será posible obtener los puertos implicados o su orden, ya que se debe realizar en un tiempo y orden específico, si no hay que volver a realizar la secuencia completa.

Sencillez de uso: una vez establecida la secuencia y el tiempo, no es necesario añadir o

configurar el servicio conectado a dicho puerto.

Los principales inconvenientes son:

Seguridad: esta técnica está pensada para el acceso a servicios privados, no es recomendable para puertos comunes como el 80 (HTTP) o 25 (SMTP).

Rendimiento: en las situaciones donde la tasa de acceso al sistema sea muy elevada, el consumo de recursos por parte de éste crecerá de manera exponencial, ya que deberá almacenar los datos de acceso de cada cliente.

Acceso: cuando se realice el acceso a través de una red local, cualquier dispositivo que pertenezca a la red local y tenga acceso a la red podrá acceder al servicio.

Compatibilidad: esta técnica no es recomendable emplearla en redes con alta seguridad por parte del firewall, ya que podría dejar al servicio sin acceso o generar un punto débil por el que acceder.

2.2.2. Universal Plug and Play (UPnP)

Es un conjunto de protocolos de comunicación, que permite a los dispositivos de una red privada realizar solicitudes al router para abrir puertos. La principal utilidad de estos protocolos, es que las solicitudes se realizan de manera transparente al usuario, sin la necesidad de configurar el router. Cuando un dispositivo solicita el acceso, el router abre el puerto, y cuando éste se desconecta de la red, el puerto se cierra automáticamente. El mayor inconveniente de este método es la vulnerabilidad de seguridad que se produce cuando un puerto está abierto, por lo que solo es recomendable emplearla en redes domésticas donde la probabilidad de un ataque sea mínima. [6] [7] [8]

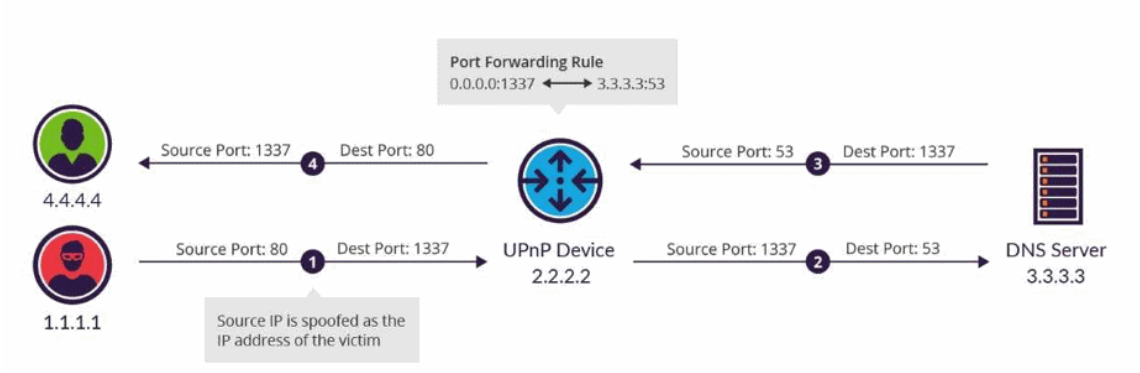


Figura 2.4: Protocolo UPNP [6]

2.2.3. Sockets TCP

Es un método de comunicación entre procesos que se encuentran en diferentes sistemas de la red: ofrecen un punto por el que transmitir o recibir información. Es interesante emplearlos, ya que, al estar orientados a la conexión de dispositivos, garantizan que la información va a llegar a su destino. Para este proyecto se deberían configurar como no bloqueantes, ya que es necesario que la comunicación sea bidireccional e instantánea en el momento que sea necesario transmitir o recibir datos.

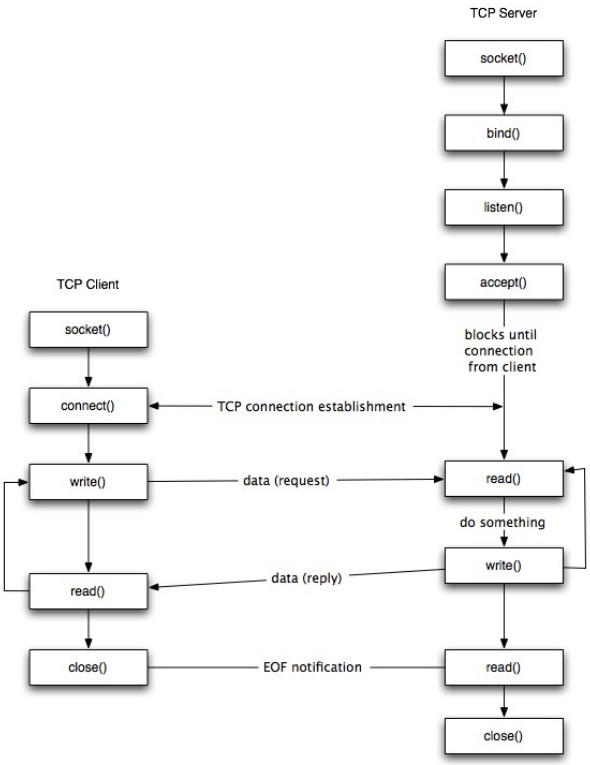


Figura 2.5: Sockets TCP [9]

2.3. Análisis de aplicaciones con la misma tecnología

Para obtener una idea general del estado en el que se encuentran las tecnologías relacionadas con este trabajo, se realizará un estudio de mercado que nos permitirá conocer los servicios que ya ofrecen dichas aplicaciones y un punto de partida sobre la posibilidad de añadir nuevas funcionalidades o mejoras. A continuación, se realizará una introducción sobre las empresas y sus productos más relevantes.

2.3.1. Hikvision

(a) Logo de la empresa [10] (b) Logo de la aplicación [10]

Figura 2.6: Figuras correspondientes a los elementos analizados en este apartado.

Es una empresa china, según cita su web "Hikvision es el mayor proveedor de soluciones de seguridad y productos de vigilancia". Cuenta con más de 34.000 empleados, sigue la misma política que su directa competidora 2.8: dedica más de la mitad de su plantilla al desarrollo de I+D permitiéndola ofrecer productos con las últimas tecnologías.

En relación con la temática del proyecto, esta empresa posee una plataforma P2P con almacenamiento en su nube, ofreciendo al usuario un cliente multiplataforma que se ha desarrollado en los siguientes formatos: aplicación de escritorio desarrollada para Windows y iOS; aplicación móvil compatible con Android y iPhone OS; y servicio web accesible por los navegadores más empleados en la actualidad (Firefox, Chrome, Safari, ...). Se ha tomado como referencia su versión Android para realizar el estudio de funcionalidad, lo más destacable es: [10]

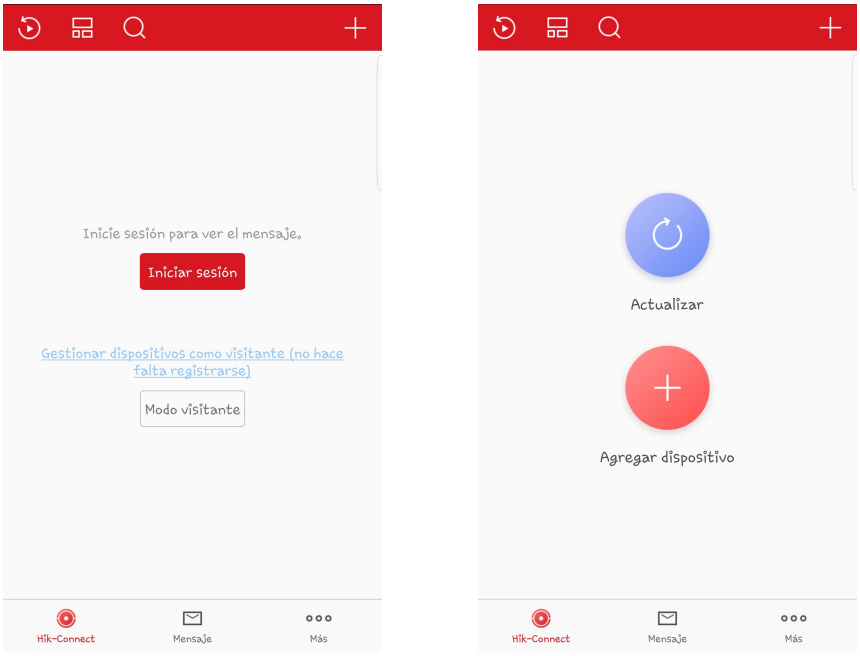
1- Solicita el país desde el que se accede a la aplicación: permite configurar el idioma de la aplicación y activar los plugins con los que se operan en dicha región, dado que estos pueden variar en función de la legislación.

2- En la vista de login se ofrecen dos tipos de acceso: usuario registrado o visitante, se puede extrapolar que la aplicación como mínimo posee dos roles diferentes.

3- Al acceder como visitante, ésta nos ofrece la posibilidad de añadir un nuevo dispositivo gracias a la lectura de su código QR; filtra los dispositivos mediante su estado de conexión; acceder a la bandeja de mensajes; conexión online con cualquier cámara activa y acceso al contenido descargado.

4- Si accedemos con el rol de usuario registrado, nos ofrece las mismas funcionalidades que en el modo visitante, además de darnos la opción de eliminar dispositivos o recursos previamente accesibles.

5- Su interfaz gráfica es muy intuitiva y fácil de manejar, ya que en pocos pasos se puede realizar cualquier acción de las citadas anteriormente.



(a) Vista de login (b) Vista principal

Figura 2.7: Principales vistas de la aplicación.

2.3.2. Dahua

(a) Logo de la empresa [11] (b) Logo de la aplicación [11]

Figura 2.8: Figuras correspondientes a los elementos analizados en este apartado.

Nos encontramos ante otra empresa china que se dedica al desarrollo de servicios tecnológicos inteligentes en el campo de la grabación de vídeo. En la actualidad opera en más de 180 países y cuenta con más de 16.000 empleados. Dedica entorno a 10.000 empleados a I+D, permitiéndole desarrollar hardware y software propio, y así tener mayor control sobre todos sus productos para no depender de terceros. Entre sus proyectos en el ámbito de la videovigilancia, caben destacar los que se llevaron a cabo en los Juegos Olímpicos de Río; la Cumbre del G20 en Hangzhou; el metro de Brasil; así como la instalación del sistema de seguridad pública en diferentes ciudades.

Centrándonos en este trabajo, sus aplicaciones dedicadas al acceso de grabadores o captura en vivo de imágenes o vídeo, se basan en una arquitectura P2P junto con un cliente multiplataforma en

el mismo formato que su directo competidor 2.3.1. Para hacer un estudio comparativo entre ambos, también se empleará su versión móvil para Android. En lugar de enumerar todas sus funcionalidades, ya que son similares, se van a describir las principales diferencias respecto a la aplicación del apartado anterior: [11]

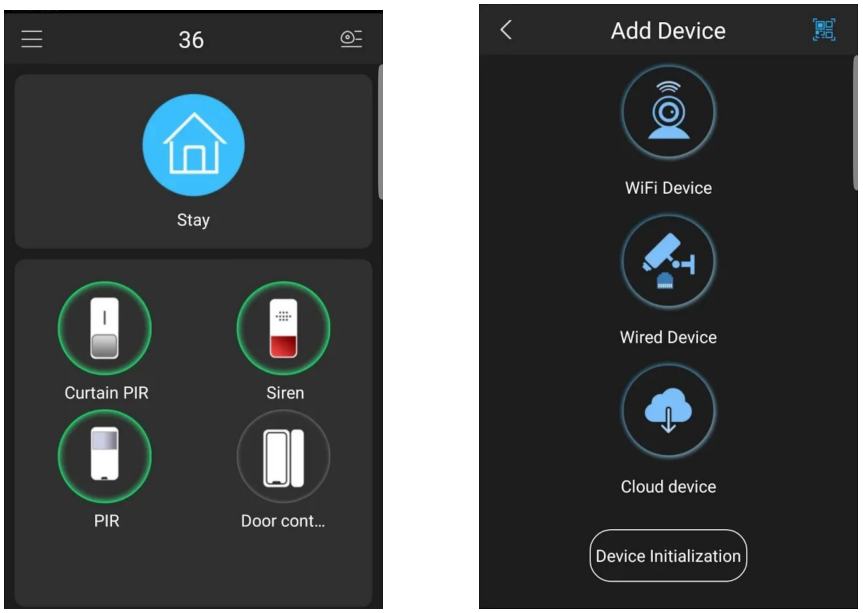
1- La configuración sobre el idioma hay que hacerla de forma manual.

2- En vez de ofrecer acceso como visitante o usuario, existen dos versiones de la aplicación, en la cada una se simula uno de los dos roles.

3- Tras acceder como visitante, no permite la posibilidad de añadir un nuevo dispositivo mediante la lectura de su código QR, se tiene que introducir el código de la cámara de forma manual; filtra los dispositivos mediante su estado de conexión; acceder a la bandeja de mensajes; conexión online con cualquier cámara activa y acceso al contenido descargado.

4- Si empleamos la aplicación con usuario registrado, nos ofrece las mismas funcionalidades que su versión reducida, además de eliminar dispositivos o recursos previamente accesibles.

5- La interfaz gráfica es menos amigable e intuitiva, ya que las imágenes de los botones y sus leyendas abreviadas no representan con exactitud la funcionalidad del botón.



(a) Vista principal (b) Vista añadir dispositivo

Figura 2.9: Principales vistas de la aplicación.

2.3.3. Conclusión

La principal diferencia entre los productos de las dos empresas, es que Hikvision 2.6 ofrece un servicio en la nube, evitando al usuario almacenar las imágenes y vídeos en sus dispositivos. Tras haber realizado la comparación anteriormente descrita de las aplicaciones existentes, me ha permitido confirmar que el desarrollo del proyecto es viable, además de ofrecerme un posible punto de partida: la vista global del Trabajo de Fin de Grado será desarrollar una red P2P; como nodos emplearemos para los clientes una aplicación de escritorio; y para los grabadores un servidor que atienda las peticiones de los clientes.

2.4. Análisis de las tecnologías

2.4.1. Modelo

Al tener la necesidad de generar una interfaz gráfica que permita al usuario interactuar con la red P2P, hay que determinar el modelo que va a seguir su implementación: En este caso la elección fue sencilla puesto que a lo largo de la carrera hemos desarrollado alguna aplicación con interfaz gráfica. La experiencia me ha llevado a la elección de que la más adecuada es el Modelo Vista Controlador (MVC), ya que es un modelo que ofrece un grado de modularidad elevado que nos permite separar las vistas de la lógica de la aplicación. [12]

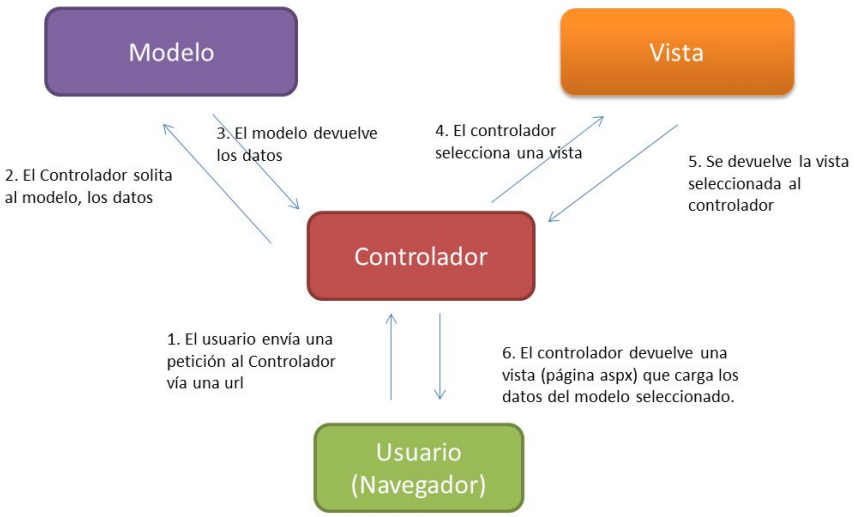


Figura 2.10: Esquema Modelo Vista Controlador

2.4.2. Ciclo de vida

Para realizar un desarrollo apropiado del sistema, se tiene que tomar una decisión sobre las fases por las que pasa el proyecto a lo largo de su vida. Para ello, se describen dos tipos de ciclos válidos: el modelo cascada iterativo y el modelo iterativo ágil. Tras estudiar sus diferencias, me he decantado por el incremental iterativo, ya que al ser una única persona quien está desarrollando el proyecto, da la facilidad de revisar cualquier parte del proyecto tras la finalización de cada iteración, lo que permite añadir o eliminar cualquier elemento e incluso modificar o mejorar algún aspecto de los mismos.

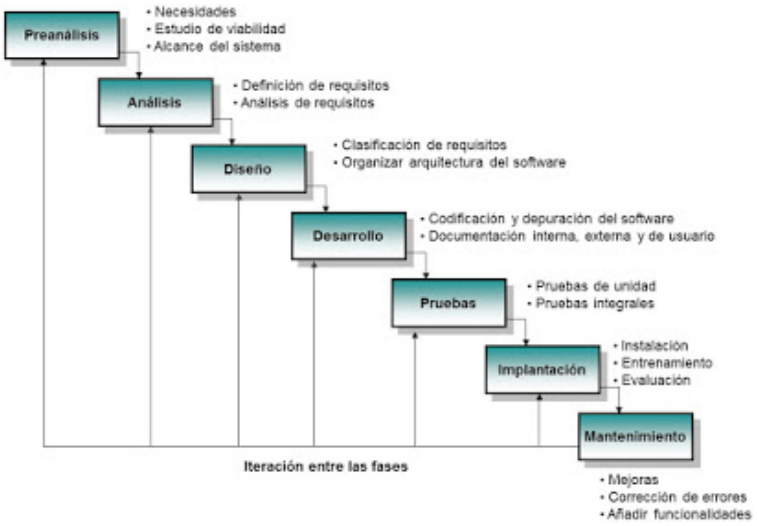


Figura 2.11: Esquema Modelo Cascada Iterativo

2.4.3. Herramientas y tecnologías

Las bases para tomar una decisión fueron:

1- Seleccionar un lenguaje de programación adecuado para la creación de aplicaciones de escritorio con interfaz gráfica, con portabilidad mínima a cualquier distribución de Linux y Windows.

2- Elegir un lenguaje de programación que permita la creación de servicios web adaptados a REST.

3- Escoger un lenguaje de programación para el desarrollo de la red P2P.

4- Manuales y recursos que posibiliten su uso y comprensión.

5- Librerías y frameworks disponibles.

6- Compatibilidad con los sistemas gestores de bases de datos.

7- Cada una de las elecciones debe ser código abierto y licencia GPL.

Kivy

(a) Logo del lenguaje de programación [13]

(b) Logo del framework [13]

Figura 2.12: PYTHON y su framework de desarrollo Kivi

Es un framework de Python que permite el desarrollo ágil de aplicaciones de escritorio con interfaz gráfica muy potentes con un consumo mínimo de recursos del sistema operativo. Dicho framework permite el uso de cualquier librería de Python combinado con su lenguaje nativo Kivy. Este lenguaje es fácil de usar debido a que está basado en la utilización de un conjunto de etiquetas, similar a CSS, por lo que al ser reducido permite obtener una curva de aprendizaje muy elevada. Una de sus principales características es el desarrollo de interfaces gráficas táctiles, por lo que además de las aplicaciones de escritorio, permite desarrollar aplicaciones enfocadas a dispositivos móviles. Otro punto destacable es que, al ser una distribución de código abierto, posee una wiki [13] muy bien documentada, y una comunidad extensa que la respalda. [14]

Qt

(a) Logo del lenguaje de programación [15]

(b) Logo del framework [15]

Figura 2.13: C++ y su framework de desarrollo Qt

Framework de C++ que posee su propio lenguaje declarativo llamado QML. Está basado en la utilización de señales y ranuras que permiten realizar una comunicación orientada a eventos. Ofrece su propio IDE de desarrollo, llamado QtCreator; además posee su propia API, que permite al programador obtener información de uso de manera eficaz y sencilla, ya que está muy bien documentada; posee su propio sistema de compilación Qmake, sin embargo, esto no es un problema dado que su funcio-

namiento es similar a la utilización de Make; y por último, permite desarrollar aplicaciones de escritorio multiplataforma, excepto para dispositivos móviles. [15].

Elección del lenguaje programación y framework del cliente

Como los dos frameworks se adaptan bien a los requisitos solicitados, se optó por el desarrollo de dos aplicaciones sencillas de prueba, permitiéndome comparar sus resultados, usabilidad y curva de aprendizaje. La decisión final fue la utilización del framework Kivy, ya que éste ofrece mayor portabilidad; y su uso es más sencillo e intuitivo; además de tener a su disposición toda la funcionalidad que ofrece Python, lenguaje con mayor utilización en la actualidad.

IDE

Figura 2.14: Pycharm [16]

Al escoger Python como lenguaje de programación, decidí usar Pycharm, uno de los IDEs más completos de Python: cada vez que se crea un proyecto, se genera por defecto un entorno virtual que permite instalar las versiones de los paquetes necesarias para la aplicación, sin generar conflictos con el resto de proyectos. Es un editor inteligente, que permite completar y generar código; a la hora de detectar un error ofrece una traza muy detallada del mismo y en muchos casos una posible solución; y por último, posee una terminal que facilita ejecutar los proyectos y así depurarlos de forma simple [16].

Django Rest

Figura 2.15: Logo Django Rest [17]

Al haber seleccionado Python para el desarrollo del cliente, lo más aconsejable es buscar un framework en el mismo lenguaje de programación basado en servicios web REST, puesto que permite su uso sin la necesidad de implementar vistas (con la excepción de las vistas del administrador). Para ello

es recomendable utilizar Django Rest, ya que permite la gestión de acceso de usuarios a la aplicación y al servidor; también la conexión con la base de datos de manera simplificada; y además posee la opción de emplear como salida el formato Json, que es una estructura de datos muy eficiente a la hora de trabajar con los mismos [17].

MySql

Figura 2.16: Logo MySql [18]

Considerado como uno de los mejores gestores de bases de datos relacionales, entre sus características destacan las siguientes: posee funcionamiento multiplataforma; es multi-thread, permite realizar varias tareas de forma paralela; dispone de librerías optimizadas y depuradas; ofrece una interfaz gráfica sencilla de comprender; tiene implementado un sistema de roles, permitiendo atribuir diferentes privilegios a los usuarios; es compatible con Djago Rest; posee un manual de usuario muy detallado; tiene gran aprobación en la comunidad de desarrolladores.

Apache

Figura 2.17: Servidor web Apache [19]

Es el primer servidor web que apareció, por lo que es el más conocido y empleado. Tiene a su favor que existe mucha información sobre su uso y configuración; está soportado por todos los SSOO y permite dotar al servidor de diversas configuraciones. En su contra cabe destacar que genera un nuevo proceso cada vez que le llega una solicitud, por lo que consume muchos recursos del sistema en el momento de atender demasiadas peticiones, siendo su principal desventaja. [19].

Nginx

Figura 2.18: Servidor web Nginx [20]

Es el gran competidor de Apache gracias al buen rendimiento que ofrece: consume poca memoria al emplear hilos en vez de procesos; tiene una tasa de servicio muy elevada y su tendencia de uso cada vez es mayor. Sin embargo, en ocasiones las configuraciones que ofrece son muy escasas, no funciona en todos los SSOO y hay partes de su documentación que no están bien redactadas [20].

Herramientas para desplegar los servidores de descubrimiento y conexiones

(a) Logo de Gunicorn [21] (b) Logo deSupervisord [22]

Figura 2.19: Herramientas para el despliegue

Gunicorn es un servidor WSGI HTTP desarrollado para trabajar con Python y además posee gran compenetración con el framework Django, por lo que su elección es la idónea para el propósito del presente proyecto. Su principal funcionalidad será la de gestionar las posibles peticiones simultáneas, definidas como workers(hilos) [23]. Sigue la siguiente ecuación:

$$WORKERS = (CPUs * 2) + 1 \tag{2.1}$$

Describiendo la fórmula, el número de peticiones (workers) que podemos atender al mismo tiempo, es directamente proporcional al número de CPUs que posea el hardware del servidor que albergará el sistema.

Supervisord es un gestor de procesos para Linux, que de manera complementaria a la ejecución de Gunicorn, permite monitorizar el estado del servidor durante el tiempo que esté desplegado. Los

datos más útiles que permite monitorizar son: Identificardor de proceso; nombre del servicio; usuario que ha lanzado el servicio; estado del servicio; y el tiempo que lleva en ejecución el servicio [24].

VirtualBox

Figura 2.20: VirtualBox [25]

Es un software de virtualización de SSOO desarrollado por Oracle Corporation [26]. Permite al sistema anfitrión albergar uno o varios sistemas operativos de forma independiente. Entre una de sus múltiples herramientas, destaca la que permite crear entre las máquinas generadas una red de telecomunicaciones: simula que cada una de ellas pertenece a una subred distinta, lo que permitirá realizar pruebas de manera eficiente en la etapa de depuración del sistema.

3

Diseño

3.1. Esquema básico de funcionamiento

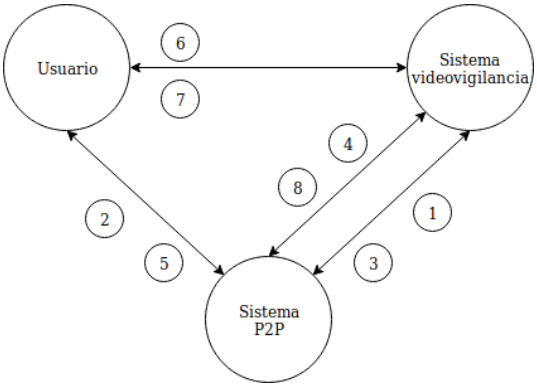


Figura 3.1: Esquema básico

3.1.1. Desripción del flujo

A partir del siguiente esquema, se describen todos los pasos que el sistema debe realizar para que se produzca la comunicación, suponiendo que solo hay un único usuario solicitando recursos:

1- El sistema de videovigilancia se registra en el sistema P2P, creando la conexión de control.

2- El usuario solicita al sistema P2P la IP y el puerto del sistema de videovigilancia.

3- El sistema de P2P indica al sistema de videovigilancia que genere una conexión de datos.

4- El sistema de videovigilancia genera la conexión de datos y la mantiene abierta de forma permanente.

5- El sistema de P2P indica al cliente el puerto de acceso al sistema de videovigilancia.

6- El cliente se conecta al sistema de videovigilancia y solicita recursos requeridos.

7- El cliente cierra la conexión con el sistema de videovigilancia.

8- El sistema de videovigilancia cierra la conexión de datos con el sistema P2P.

3.2. Esquema general del proyecto

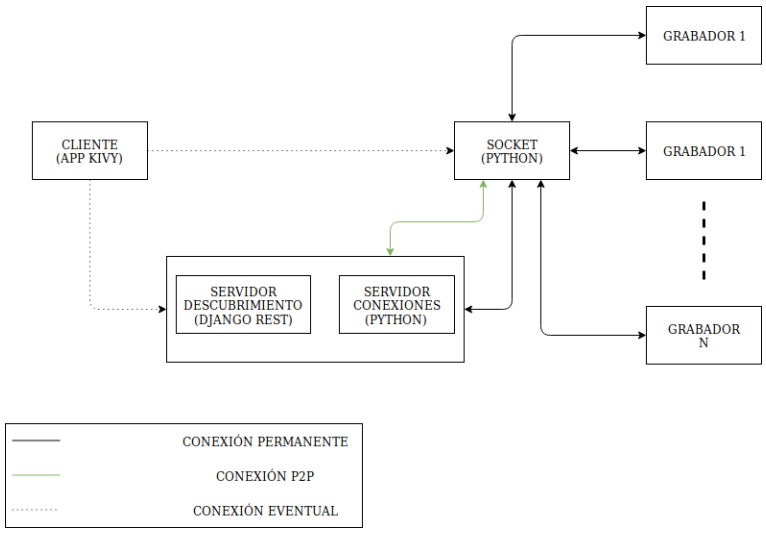


Figura 3.2: Esquema general

3.2.1. Desripción de los componentes

A continuación, se introducen los componentes del sistema, ofreciendo una vista global, para posteriormente ofrecer una explicación más detallada de los mismos.

Cliente: es la aplicación que ofrece al usuario la interfaz gráfica, permitiéndole interactuar con el resto de componentes de la red.

Servidor de descubrimiento: su finalidad es la de gestionar todos los datos del sistema; ofrecer una capa de seguridad mediante login; y atender las peticiones generadas por el socket o el cliente.

Servidor de conexiones: su función es la de gestionar toda la red P2P, permitiendo la interactuación entre los clientes y los grabadores mediante la solicitud de una conexión al grabador correspondiente, generando que se abra un nuevo puerto por el que el cliente envía sus peticiones.

Socket: pasarela que gestiona el acceso a todos los grabadores mediante la creación de una nueva conexión por la que el cliente enviará sus peticiones.

Grabadores: sistemas que ofrecen los vídeo o imágenes. No son elementos importantes, puesto que solo envían datos.

3.2.2. Descripción del funcionamiento

Suponiendo un sistema recién instalado, el primer paso a realizar será generar como mínimo un usuario administrador en el sistema de descubrimiento: éste deberá introducir en el sistema todos los usuarios, grabadores y cámaras que formen parte de la red P2P. Posteriormente, se activará cada uno de los grabadores que enviará un mensaje al distribuidor de peticiones (socket) con el objetivo de solicitar al servidor de conexiones que les ponga en estado activo.
Cada vez que el servidor de conexiones reciba una petición de conexión por parte del socket, éste comprobará que el grabador está registrado en el sistema de descubrimiento: si lo está, mantendrán una conexión permanente hasta que uno de los dos componentes quede fuera de servicio; y en el caso que no lo esté, rechazará la conexión.
En este aspecto el sistema está preparado para recibir peticiones por parte de los clientes. Cuando una de éstas llega, solicitará al servidor de conexión el acceso a uno de los grabadores; éste comprobará si está activo; en caso de que no, se lo notificará al cliente; y en caso de que sí, verificará si hay ya creada una conexión de datos frente dicho grabador. En caso de existir, se indicará al cliente el puerto dónde se localiza dicho acceso de datos; en caso negativo, se almacenará el nombre del cliente y el grabador a acceder hasta obtener la respuesta de la solicitud por parte del socket. Además, enviará la señal de abrir nueva conexión al socket para que se genere un punto de acceso al grabador especificado.
Cuando el socket recibe una petición de este tipo, deberá generar la nueva conexión contra el servidor de conexiones e indicarle que se trata de una conexión de datos, para que éste acceda a los clientes en espera y les envíe el puerto por el que puede solicitar al grabador uno de los recursos disponibles en el mismo.
Transcurrido un tiempo, el socket comprobará si el cliente ha terminado de solicitar y descargar recursos, si es así, deberá cerrar la conexión para que el puerto se cierre de nuevo.

3.3. Diagrama de flujo del cliente

El objetivo de este diagrama 3.3 no consiste en representar un esquema UML detallado, si no ofrecer los diferentes flujos de datos que se pueden generar dentro de la aplicación de cliente gracias a las diferentes funcionalidades que ofrece la misma a partir del tipo de cliente que se registra en la aplicación.

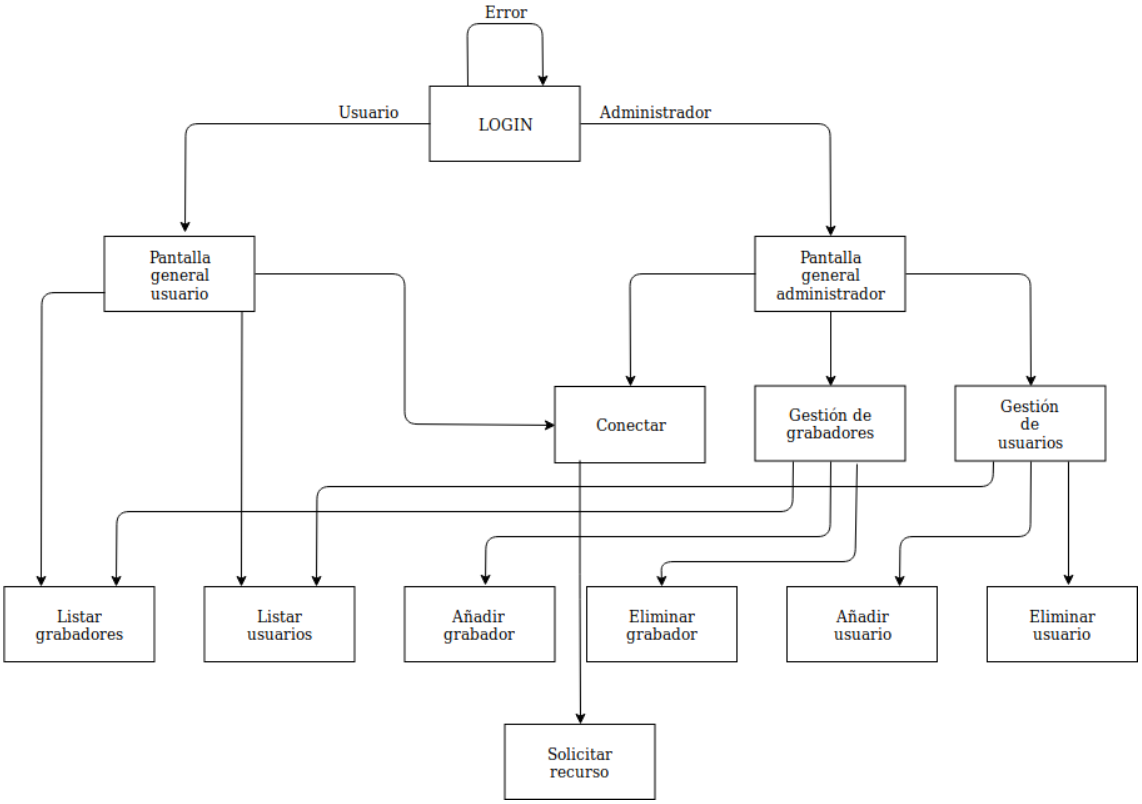


Figura 3.3: Diagrama de flujo del cliente

3.4. Esquema de datos

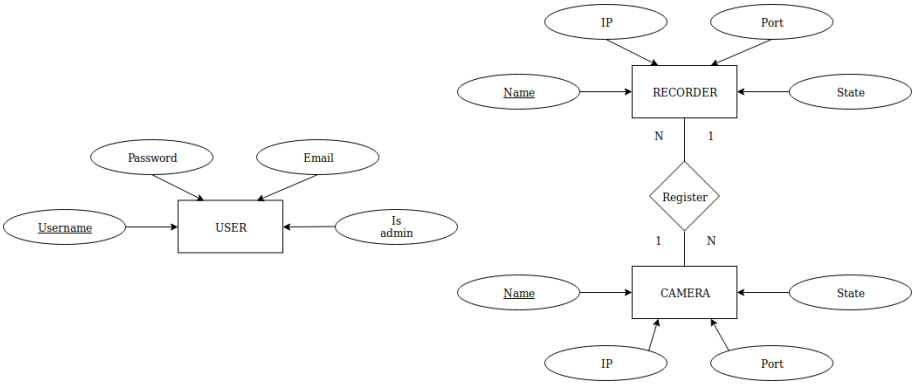


Figura 3.4: Diagrama entidad relación

A partir de diagrama **??** se introducirán todos los datos que debe ser capaz de almacenar y gestionar el sistema P2P, cada uno de ellos consiste en:

User: representa a los usuarios que pueden acceder a la aplicación del cliente.

Username: nombre del usuario.

Password: contraseña del usuario.

Email: correo del usuario.

Is admin: indica los permisos del usuario dentro del cliente.

Recorder: modelo de datos que contiene la información de los grabadores.

Name: nombre del grabador.

IP: dirección dónde localizar al grabador.

Port: puerto dónde localizar al grabador.

State: indica si está activo el grabador.

Camera: objeto que almacena la información de las cámaras.

Name: nombre de la cámara.

IP: dirección dónde localizar la cámara.

Port: puerto dónde localizar la cámara.

State: indica si está activa la cámara.

4

Desarrollo

En este apartado se ofrecerá una descripción detallada de la implementación de todos los componentes.

4.1. Servidor de descubrimiento

Para este apartado podría realizar una guía sobre cómo implementar una aplicación Rest, pero para ello podemos acceder a su documentación [17], gracias a un tutorial con tal propósito. Los más destacable de este apartado es detallar el diseño de la base de datos y la especificación de las URLs para la gestión del sistema.

De cara a la base de datos que está implementada en MySql 2.4.3 y siguiendo las especificaciones del diseño 3.4, no hay ningún detalle de implementación relevante a la hora de abordar este tema. En cuanto a la especificación de las URLs accesibles por el navegador, disponemos de las siguientes:

discoverServer/admin/: acceso al entorno de administración de los datos del servidor.

discoverServer/api/: acceso al entrono principal de Django REST: nos muestra un listado con todos los objetos de la base de datos que puede representar en formato Json.

discoverServer/api/user/ muestra el listado de todos los usuarios en formato Json.

discoverServer/api/user/<username>/ muestra los datos del usuario indicado en formato Json.

discoverServer/api/recorder/ muestra el listado de todos los garabadores en formato Json.

discoverServer/api/recorder/<name>/ muestra los datos del grabador indicado en formato Json.

discoverServer/api/camera/ muestra el listado de todas las cámaras en formato Json.

discoverServer/api/camera/<name>/ muestra los datos de la cámara indicada en formato Json.

El resto de operaciones se deberán realizar mediante la terminal del SO a través del comando

CURL con el siguiente formato:

```
curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X POST –
data username=<username>&password=<password>&email=<email>
http://discoverServer/api/user/
```

Cuadro 4.1: Creación de un usuario con el comando CURL

***El resto de comandos se encuentran en el anexo C**

4.2. Cliente

Como ya se introdujo en el estado del arte, el modelo a emplear es MVC 2.4.1, por lo que el primer paso ha sido generar una biblioteca desarrollada en Python 2.4.3 para poder acceder a los servidores de descubrimiento y conexiones. Las tareas que nos permite realizar sobre el servidor de descubrimiento son las de crear, eliminar o listar usuarios y grabadores; y sobre el de conexiones, solicitar y descargar un recurso. Posteriormente se ha implementado el modelo, aportando la lógica de la aplicación, ya que éste es el encargado de utilizar la funcionalidad de la biblioteca e introducir los datos o acciones dentro de la ejecución de la aplicación. Al estar implementado en Kivy, sólo ha sido necesario crear una clase llamada AccessLibrary, que contiene los siguientes métodos:

checkUser: comprueba si el usuario existe y el rol que posee.

addUser: verifica los datos introducidos en el formulario y realiza una petición al servidor de descubrimiento para añadir un usuario.

deleteUser: envía una petición al servidor de descubrimiento para eliminar al usuario.

getUsers: solicita todos los datos de todos los usuarios en formato Json.

getUser: solicita los datos de un solo usuario en formato Json.

addRecorder: verifica los datos introducidos en el formulario y realiza una petición al servidor de descubrimiento para añadir un grabador.

deleteRecorder: envía una petición al servidor de descubrimiento para eliminar al grabador.

getRecorders: solicita todos los datos de todos los grabadores en formato Json.

getRecorder: solicita los datos de un solo grabador en formato Json.

connectRecorder: solicita al servidor de conexiones el puerto de grabador indicado, para crear una conexión TCP, que servirá de conexión directa entre el cliente y el grabador.

listItems: solicita al grabador la lista de todos sus recursos.

getItem: solicita un recurso al grabador.

Por último, se implementaron las vistas. Para ello se comenzó por el desarrollo del elemento más pequeño hasta generar la vista completa, permitiendo que la APP sea lo más homogénea posible: en lenguaje Kivy se generaron botones personalizados de return, logout, realizar una acción, confirmar, login; y el siguiente paso fue definir las dimensiones de las vistas (márgenes, posición de los elementos comunes, ...) generando una clase padre denominada GenericView, de la que heredaban tres vistas más: MainWindow, MenuView, ListView, produciéndose el siguiente esquema de herencias.

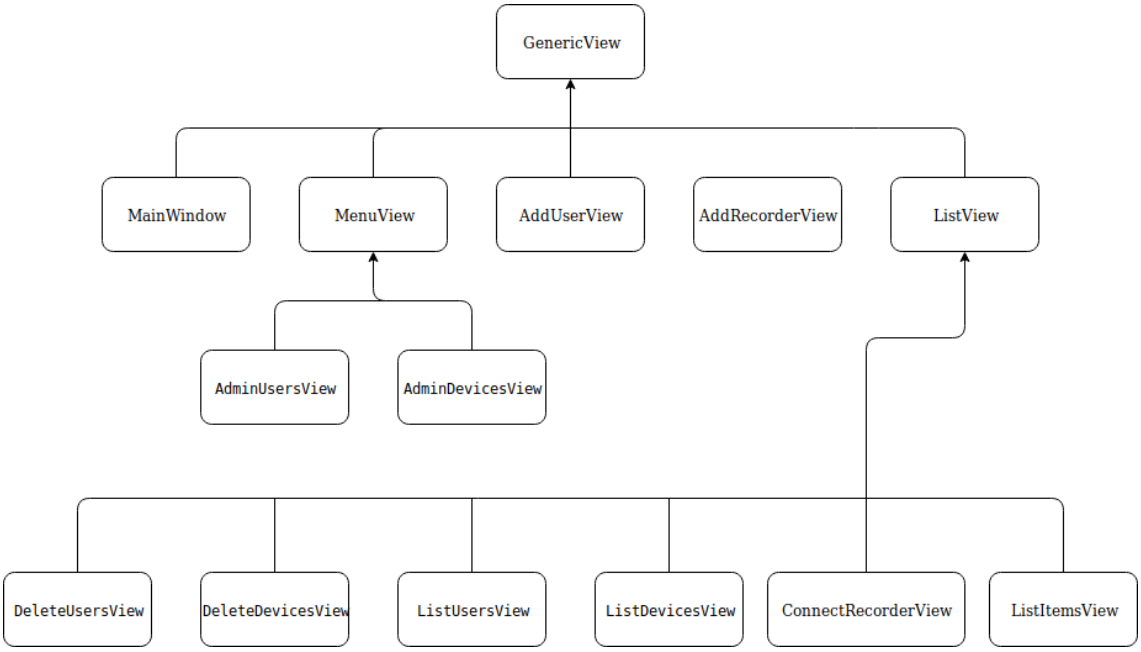


Figura 4.1: Diagrama de herencia de las vistas

La parte del controlador se genera de manera automática por parte del framework de Kivy, definiendo el método que ejecuta cada elemento de la vista al ser instanciado, por lo que esta parte del código no es necesaria de implementar.

4.3. Servidor de conexiones

Sistema implementado en Python [27], en concreto en su versión 3.0, ya que la 2.0 se está quedando sin soporte de forma gradual. Es una de las piezas clave en la implementación del sistema P2P, puesto que es el encargado de: gestionar las conexiones de control con los grabadores; solicitar a estos las conexiones de datos; y atender las peticiones de conexión por parte de los clientes. Para su desarrollo se han empleado 3 hilos: el primero se encargará de atender las peticiones nuevas; el segundo procesará las peticiones de los clientes y les devolverá el resultado de la misma; y el tercero gestionará las conexiones de datos y las mantendrá abiertas, permitiendo que los clientes utilicen el

puerto abierto en éstas para poder comunicarse con los grabadores. El proceso que se crea al ejecutar el servidor después de crear los hilos, generará una entrada de datos por terminal para ejecutar comandos internos de debug o de final de ejecución.

Para generar las conexiones se han empleado sockets TCP no bloqueantes, permitiendo que el proceso no quede a la espera de recibir datos y así poder gestionar todas las conexiones por medio de las esperas activas. De manera secuencial se irá comprobando si hay alguna petición por parte de algún cliente, previamente aceptado en el servidor.

4.3.1. Descripción del funcionamiento

Tras ejecutar el servidor, la primera tarea será generar un socket TCP. Para cumplir con las necesidades del proyecto, se configurará en modo no bloqueante y se definirá un tamaño de cola prudente, así se evitará que el sistema se sature. Posteriormente se instanciarán los hilos de aceptar cliente y procesar cliente, ambos en modo demonio, para evitar que el proceso principal quede bloqueado. Tras este proceso, todos los hilos quedarán a la espera. Centrándonos en el hilo principal, se comprobará si se introduce una cadena de texto por teclado, principalmente se empleará para debug, los comandos que atiende son: listar todos los grabadores conectados, listar todos los clientes activos, listar conexiones de datos y listar todas las peticiones pendientes de abrir conexión de datos. Por otro lado, cuando llegue una conexión nueva, el hilo de aceptar cliente realizará el accept y comprobará si existe una conexión anterior del cliente: si se cumple esta condición, será sustituida por la nueva; y en caso contrario, la configurará como no bloqueante y almacenará su descriptor de socket en una lista. Así de manera sucesiva hasta que el servidor quede fuera de servicio. Mientras tanto, el hilo de procesar cliente estará recorriendo la lista de clientes, y cuando detecte que hay un mensaje de un cliente, comprobará el mismo y ejecutará la funcionalidad correspondiente para atender la petición. Los verbos que es capaz de entender el servidor son los siguientes:

CONNECT: señal que enviará un grabador en el momento que se active: permite al sistema conocer que el grabador está activo, actualizando su estado a activo en el servidor descubrimiento, posteriormente se añadirá a la lista de servidores activos del servidor de conexiones.

ALIVE: permite al servidor detectar cuándo un grabador le manda la señal de que está activo: cuando reciba una solicitud de este tipo, el servidor actualizará la IP y el puerto almacenados en el servidor de descubrimiento en caso de que hayan variado.

CLOSE: señal que permite saber cuándo un cliente o un grabador se va a desconectar, permitiendo al servidor de conexiones eliminar su socket; y si se trata de un grabador, cambiar su estado a ofline en el servidor de descubrimiento.

OPEN_HOLE: solicita al servidor de conexiones el puerto de datos de un determinado grabador para solicitarle los recursos: si ya existe, se le devolverá el puerto directamente; y si no, se añadirá su petición a la lista de peticiones de conexión de datos y se solicitará al grabador que genere una conexión de datos contra el servidor de conexiones. Una vez realizada, se enviará al cliente el puerto para que realice la conexión y se eliminará de la solicitud.

PORT_HOLE: señal por la que el grabador indicará al servidor de conexiones que está realizando una conexión de datos y que almacene el puerto para que los clientes puedan solicitarle recursos.

4.4. Socket

Sistema que actúa de pasarela entre las conexiones del grabador con el servidor de conexiones y los clientes con el grabador. Para su implementación se ha empleado el lenguaje de programación Python, con la misma filosofía que el servidor de conexiones 4.3, la utilización de espera activa con sockets TCP no bloqueantes. Para su implementación se ha empleado el proceso principal como terminal para debuguear o cerrar el sistema, junto con tres hilos encargados de: realizar la conexión de control con el servidor de conexiones; enviar la señal de alive cada cierto período de tiempo; y generar la conexión de datos siempre que se solicite acceso a sus recursos.

4.4.1. Descripción del funcionamiento

En el momento de iniciar el sistema, el propio socket deberá enviar una señal de connect al servidor de conexiones para ponerse en estado activo dentro del sistema y generar la conexión de control. Tras este momento, el socket estará comprobando la entrada de datos por teclado y la conexión de control. Cuando éste reciba una señal de abrir conexión de datos, activará el hilo destinado para este propósito y éste creará un socket TCP por el que enviará el comando de conexión de datos, permitiendo que el servidor de conexiones detecte que es de este tipo y pueda gestionar las solicitudes pendientes. Mientras dicha conexión permanezca abierta, cualquier cliente podrá conectarse a través de ésta hasta que todos los clientes se desconecten; entonces cerrará la conexión y se lo indicará al servidor de conexiones, volviendo a quedar en el mismo estado que al inicio de su funcionamiento.

Los comandos que es capaz de atender son:

CONNECT: envía esta señal para indicar que está activo en el sistema.

ALIVE: indica al servidor de conexiones que sigue estando activo.

CLOSE: indica al servidor de conexiones que el sistema se va a desactivar.

CLOSE_DATA: indica al servidor de conexiones que cierre la conexión de datos.

OPEN_HOLE: señal que activa el hilo de conexiones de datos, permitiendo que éste genere una conexión TCP no bloqueante.

PORT_HOLE: indica al servidor de conexiones el puerto por el que se debe acceder.

GET_FILES: un cliente solicita la lista de recursos del grabador.

GET_FILE: un cliente solicita un determinado recurso al grabador.

5

Integración, pruebas y resultados

El desarrollo de esta sección ha posibilitado evaluar todos los aspectos del funcionamiento del sistema, permitiéndome valorar: si se han cumplido todos los requisitos propuestos en la fase de análisis; cerciorarme que cada uno de los bloques implementados funciona de forma correcta bajo cualquier circunstancia; y determinar el grado de fiabilidad y calidad del sistema desarrollado. Para su correcta comprobación, se han realizado tanto pruebas de carácter unitario, donde se comprueba cada componente de forma individual [28], como pruebas de integración [29], cuya función es comprobar el acoplamiento de todos los componentes del sistema. En el desarrollo de los bloques de pruebas, se han empleado las siguientes metodologías: pruebas de caja blanca [30], cuyo cometido es verificar el correcto funcionamiento de los flujos de datos que se generan en un componente o en la integración de todos; y pruebas de caja negra [31], empleadas para comprobar las salidas que debe ofrecer cada elemento del sistema de forma individual y grupal. También se han realizado pruebas no funcionales [32], cuyo fin es el de verificar la usabilidad y escalabilidad del sistema mediante la simulación de ejecuciones en máquinas virtuales, o solicitando a personas ajenas al proyecto que lo utilicen y lo valoren, permitiendo analizar los resultados obtenidos.

5.1. Demostración de funcionamiento

Para este apartado se ha realizado un sistema de pruebas mediante el uso de tres máquinas virtuales: la primera contendrá la aplicación de cliente; la segunda los servidores de descubrimiento y conexión; y la tercera el socket. De esta manera podemos simular un acceso a cada componente empleando el router de la red. En cuanto a su configuración, cada una de las máquinas deberá configurarse como adaptador tipo puente, esta opción permite que el SO anfitrión genere un adaptador de red ethernet o wifi para cada máquina, permitiendo que cada una de ellas obtenga la dirección IP directamente desde la puerta de enlace, ofreciendo la misma funcionalidad que si estuviéramos en un sistema físico. [33] [34]

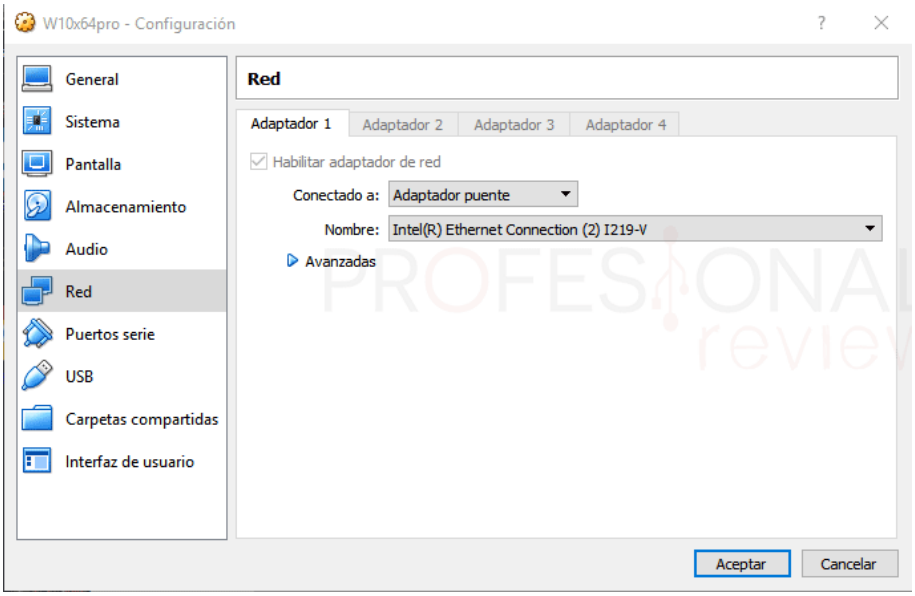


Figura 5.1: Configuración tipo puente [33]

5.1.1. Pruebas realizadas

Comprobación de la IP asignada: mediante el comando de Linux ifconfig, se obtendrá la IP asignada a cada máquina.

Ping: mediante este comando se comprobará que las maquinas son accesibles a través de la red.

Creación de los datos del sistema: se crearán, eliminarán y modificarán objetos de datos mediante los siguientes métodos: acceso por navegador a la interfaz de administrador; utilizando el comando CURL C; y mediante la aplicación de cliente D

Registro de un grabador en el servidor de conexiones: se comprobará la correcta comunicación entre los dos servicios y se verificará que en el servidor de descubrimiento el estado del grabador pasa a estar activo.

Desconexión de un grabador del servidor de conexiones: se comprobará que todos los recursos son liberados y que el estado del grabador en el servidor de conexiones pasa a inactivo.

Solicitud de acceso por parte de un cliente a un grabador no conectado: se deberá comprobar que el cliente no queda a la espera de conexión.

Solicitud de acceso por parte de un cliente a un grabador conectado: se deberá comprobar que el cliente queda a la espera hasta que el servidor de conexiones envíe el puerto para la conexión o pase el tiempo definido de espera.

Solicitud de conexión de datos al grabador: se deberá comprobar que la conexión TCP se genera de forma correcta y es accesible al servidor de conexiones y a cualquier cliente.

Conexión del cliente con el grabador: se comprobará la conexión entre los dos elementos.

Descarga de datos del grabador: se comprobará que el cliente recibe todos los recursos disponibles del sistema o el recurso seleccionado.

conexión correcta del cliente: cuando un cliente termine de solicitar recursos, se deberá comprobar que cierra correctamente los sockets conectados al grabador y al servidor de conexiones.

6

Conclusiones y trabajo futuro

6.1. Conclusiones

La finalización del sistema P2P, me ha permitido afianzar mi confianza de cara a la toma de decisiones en un entorno de trabajo, ya que a lo largo del desarrollo del proyecto surgieron una serie de problemas y decisiones que afectarían drásticamente al resultado final del mismo. Para obtener los mejores resultados en estas situaciones, la rutina fue siempre la misma: recabar información; sintetizar los puntos más relevantes que podría aplicar al proyecto; y en caso de duda, consensuar con el tutor la mejor manera de llevarlo a cabo.

Centrándome en el ámbito educacional, la experiencia ha sido muy enriquecedora, puesto que me ha permitido poner en práctica bastantes conocimientos adquiridos a lo largo de la carrera de forma conjunta, como base para el desarrollo del sistema. Mediante la profundización de las diversas materias necesarias, he obtenido mayor grado de compresión, permitiéndome tener más herramientas a la hora de afrontar proyectos con la misma problemática. Gracias a su implementación, he adquirido unas mecánicas de trabajo que en mi opinión me aportarán muchos beneficios de cara a mi inserción en el mundo laboral.

Desde un punto de vista más detallado sobre el proyecto, durante todas sus etapas de desarrollo, mi meta siempre ha sido la de obtener el mejor resultado posible, con la finalidad de que pueda ser reutilizado en otros proyectos. Por lo que, si lo dividiera en etapas, la primera sería la más larga, puesto que fue un período de reflexión donde el objetivo principal fue asentar las bases y organizar de forma precisa las etapas y caminos que debería seguir el sistema. Una vez planteado y organizado el proyecto, había que buscar una serie de tecnologías y herramientas que se adaptaran lo mejor posible a las decisiones tomadas anteriormente. Teniendo todas las bases y herramientas decididas, lo único que faltaba era dedicar todo el esfuerzo al desarrollo del sistema, la metodología seguida fue la de dividir el proyecto en pequeños problemas y tratarlos en un principio de forma individual; ésto me permitió desarrollar y probar cada uno de ellos de una manera ágil y simplificada.

Como conclusión final, espero que el esfuerzo, la dedicación y tiempo invertidos en este trabajo, genere una sensación positiva sobre el mismo, y permita captar el potencial frente a todas las posibles

aplicaciones que se podrían dar de cara a su implementación en otros proyectos. A modo de ejemplo, éste se ha centrado en servidores de vídeo, pero podría adaptarse a cualquier tipo de servidor.

6.2. Trabajo futuro

Lo más importante será seguir las pautas y métodos empleados para el desarrollo del sistema, de forma que las posibles actualizaciones o mejoras estarán en concordancia con el resto de sistemas. Como potenciales mejoras de la aplicación, podrían sugerirse:

Añadir un sistema de cloud: permitiría eliminar la necesidad de que la base de datos y los recursos solicitados tengan que ser almacenados de manera local.

Aplicación cliente para dispositivos móviles: ofrecería al cliente acceder en cualquier momento a sus datos de una manera online desde cualquier lugar, esta mejora iría ligada con la citada anteriormente.

Añadir un reproductor de vídeo a la aplicación cliente: permitiría al cliente reproducir Íos recursos solicitados sin la necesidad de poseer una aplicación externa que le permita visualizarlos.

Mejora en la seguridad 1: adaptar al sistema para que funcione con HTTPS.

Mejora en la seguridad 2: encriptar los datos que envía el socket a los clientes, esto permitiría que individuos ajenos a los mismos puedan acceder a ellos.

Mejora de rendimiento 1: comprimir los datos que envía el socket a los clientes, lo que generaría la reducción de la cantidad de datos que hay que enviar por la red.

Solicitudes: añadir a la aplicación cliente un apartado de solicitudes, donde se podrían introducir sugerencias o problemas que han tenido los usuarios de cara al uso del sistema.

Bibliografía

[1] C. Valero, "Internet nació con dos grandes problemas según uno de sus creadores." https://www.adslzone.net/2019/01/21/dos-problemas-internet-creador/, 2019. [Estado: Activo].

[2] R. Steinmetz, *What Is This Peer-to-Peer?*, pp. 9–16. Springer, first ed., 2005.

[3] L. Castro, "Qué son los programas P2P y cómo funcionan." https://www.aboutespanol.com/que-son-los-programas-p2p-y-como-funcionan-157981, 2016. [Estado: Activo].

[4] G. Fernández, "Knock, knock, knockin' on server's ports – Port knocking con ejemplos." https://poesiabinaria.net/2017/08/knock-knock-knockin-on-servers-ports-port-knocking-ejemplos/#beneficios-de-esta-teacutecnica, 2018. [Estado: Activo].

[5] Admin, "Port knocking." https://wiki.archlinux.org/index.php/Port_knocking_(Espa%C3%B1ol), 2018. [Estado: Activo].

[6] Microsoft, "Using the UPnP Control Point API." https://docs.microsoft.com/en-us/previous-versions/windows/embedded/ms898948(v=msdn.10), 2012. [Estado: Activo].

[7] Admin, "¿QUÉ ES UPnP Y PARA QUE SIRVE ESTA FUNCIÓN DEL ROUTER?." https://pcpro.es/guias/que-es-upnp-y-para-que-sirve-esta-funcion-del-router/, 2018. [Estado: Activo].

[8] G. G. Richard, *Service and Device Discovery : Protocols and Programming*. McGraw-Hill Professional, 2005.

[9] I. O. for Standardization, "ISO/IEC 29341-1:2011 Information technology – UPnP Device Architecture." https://www.iso.org/standard/57195.html, 2012. [Estado: Activo].

[10] Hikvision, "Web de Hikvision." https://www.hikvision.com/es/, 2019. [Estado: Activo].

[11] D. technology, "Web de Dahua." https://www.dahuasecurity.com/es/, 2010. [Estado: Activo].

[12] U. de Alicante, "Modelo–vista–controlador." https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html, 2019. [Estado: Activo].

[13] Kivy, "Kivy wiki." https://kivy.org, 2019. [Estado: Activo].

[14] A. Mozco, "Kivy: El Framework de código abierto para el desarrollo rápido de aplicaciones en Python." https://alfonsomozkoh.github.io/2018/06/16/kivy-el-framework-de-codigo-abierto-para-el-desarrollo-rapido-de-aplicaci
html, 2019. [Estado: Activo].

[15] Qt, “Qt wiki.” https://www.qt.io, 2019. [Estado: Activo].

[16] Pycharm, “Pycharm wiki.” https://www.jetbrains.com/pycharm/, 2019. [Estado: Activo].

[17] DjangoRest, “Django Rest wiki.” https://www.django-rest-framework.org/, 2019. [Estado: Activo].

[18] MySQL, “MySQL wiki.” https://www.mysql.com/, 2019. [Estado: Activo].

[19] Apache, “Apache wiki.” https://www.apache.org/, 2019. [Estado: Activo].

[20] Nginx, “Nginx wiki.” https://www.nginx.com/, 2019. [Estado: Activo].

[21] Gunicorn, “Gunicorn wiki.” https://gunicorn.org/, 2019. [Estado: Activo].

[22] Supervisord, “Supervisord wiki.” https://www.pythoniza.me/gunicorn/, 2019. [Estado: Activo].

[23] A. Dzul, “Gunicorn.” https://www.pythoniza.me/gunicorn/, 2017. [Estado: Activo].

[24] Admin, “Supervisord: Un gestor de procesos para linux.” https://javierin.com/supervisord-un-gestor-de-procesos-para-linux/, 2010. [Estado: Activo].

[25] Oracle, “ViertualBox wiki.” https://www.virtualbox.org/, 2019. [Estado: Activo].

[26] O. Corporation, “Oracle Corporation.” https://es.wikipedia.org/wiki/Oracle_Corporation, 2019. [Estado: Activo].

[27] Python, “Python Wiki.” https://www.python.org/download/releases/3.0/, 2019. [Estado: Activo].

[28] Microsoft, “Conceptos básicos de las pruebas unitarias.” https://docs.microsoft.com/es-es/visualstudio/test/unit-test-basics?view=vs-2019, 2019. [Estado: Activo].

[29] E. S. de Informática (UCLM), “Ejemplo de pruebas unitarias y de integración.” http://www.inf-cr.uclm.es/www/mpolo/asig/is4/madumSistema.pdf, 2019. [Estado: Activo].

[30] U. P. de Valencia, “Pruebas de caja blanca. Técnica del camino básico.” https://www.youtube.com/watch?v=O6Cg4ing5bo, 2017. [Estado: Activo].

[31] U. P. de Valencia, “Pruebas de caja negra. Tecnica de particion equivalente.” https://www.youtube.com/watch?v=pAVc6SY__cA, 2017. [Estado: Activo].

[32] U. del Valle, “Pruebas no funcionales.” https://campusvirtual.univalle.edu.co/moodle/pluginfile.php/1210928/mod_resource/content/1/2017_PruebasNoFuncionales-ParteI.pdf, 2018. [Estado: Activo].

[33] P. review, “Formas de conectar dos máquinas virtuales en red VirtualBox.” https://www.profesionalreview.com/2018/12/16/conectar-maquinas-virtuales-en-red-virtualbox/, 2018. [Estado: Activo].

[34] U. P. C. III, “Simular redes independientes en Virtualbox.” https://www.youtube.com/watch?v=0BU469zRB2g, 2017. [Estado: Activo].

Definiciones

arquitectura monolítica la responsabilidad del funcionamiento de la red recae en el servidor central.

base de datos relacionales es una colección de elementos de datos organizados en un conjunto de tablas formalmente descritas desde la que se puede acceder a los datos o volver a montarlos de muchas maneras diferentes sin tener que reorganizar las tablas de la base.

entorno virtual es un ambiente creado con el objetivo de aislar recursos como librerías y entorno de ejecución, del sistema principal o de otros entornos virtuales.

firewall programa informático que controla el acceso de una computadora a la red y de elementos de la red a la computadora.

Json es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.

plugin complemento de una aplicación que se relaciona con otra para aportarle una función nueva.

router dispositivo de hardware que permite la interconexión de ordenadores en red.

Acrónimos

API Application Programming Interface.

CSS Cascading Style Sheets.

GPL General Public License.

HTTP Hypertext Transfer Protocol.

I+D Investigation and Development.

IDE Integrated Development Environment.

IP Internet Protocol.

MVC Model View Controller.

NAT Network Address Translation.

P2P Peer To Peer.

QR Quick Response.

REST Representational State Transfer.

SMTP Simple Mail Transfer Protocol.

WSGI Web Server Gateway Interface.

Apéndices

A

Análisis de requisitos

Debido a que el sistema consta de varios elementos, la mejor forma de detallar este documento es tratándolos como componentes individuales.

A.1. Cliente

Requisitos funcionales:

RF1: La aplicación deberá disponer un sistema de registro, para controlar el acceso a la misma.

RF2: Se deberá emplear un sistema de roles donde exista un rol de usuario y otro de administrador.

RF3: Listar usuario.

RF4: Registrar usuario.

RF5: Eliminar usuario.

RF6: Mostrar los detalles de usuario (nombre, password, email).

RF7: Listar grabador.

RF8: Eliminar grabador.

RF9: Registrar grabador.

RF10: Mostrar los detalles de grabador (nombre, ip, puerto, estado).

RF11 Conectar con un grabador.

RF12: Listar los recursos que dispone un grabador.

RF13: Solicitar recurso.

RF14: Almacenar el recurso en el sistema.

NOTA: Al disponer de dos roles, el administrador engloba rodos los requisitos citados, pero a el usuario solo le afectan los RF's: 1, 7, 10, 11, 12, 13, 14.

Requisitos no funcionales:

-Seguridad:

RNF1: La contraseña de usuario deberá ser encriptada antes de ser almacenada en el sistema.

RNF2: Los roles de usuario solo podrán ser modificados por el administrador.

-Portabilidad:

RNF3: La aplicación deberá ser compatible con Widows y Linux.

-Usabilidad:

RNF4: La aplicación debe ser persistente de cara a los datos empleados.

RNF5: La interfaz gráfica deberá mostrar mensajes informativos al usuario.

RNF6: En caso del uso erróneo de la aplicación por parte de un usuario, se deberá generar un mensaje dónde se le indicará el error y una breve descripción.

-Interfaz:

RNF7: Debe ofrecer un aspecto intuitivo y amigable para el usuario.

RNF8: El diseño deberá ser estéticamente atractivo.

RNF9: Para facilitar su uso, deberá contener botones ilustrados.

A.2. Servidor de descubrimiento

Requisitos funcionales:

RF1: La aplicación deberá disponer un sistema de registro, para controlar el acceso a la misma.

RF2: Se deberá emplear un sistema de roles donde exista un rol de usuario y otro de administrador.

RF3: Listar usuario.

RF4: Registrar usuario.

RF5: Eliminar usuario.

RF6: Modificar usuario.

RF7: Mostrar los detalles de usuario (nombre, password, email).

RF8: Listar grabador.

RF9: Eliminar grabador.

RF10: Registrar grabador.

RF11: Modificar grabador.

RF12: Mostrar los detalles de grabador (nombre, ip, puerto, estado).

Requisitos no funcionales:

-Seguridad:

RNF1: La contraseña de usuario deberá ser encriptada antes de ser almacenada en el sistema.

RNF2: Los roles de usuario solo podrán ser modificados por el administrador.

-Portabilidad:

RNF3: La aplicación deberá poder ser desplegada como servicio web.

-Usabilidad:

RNF4: La aplicación debe ser persistente.

RNF5: La interfaz gráfica del administrador deberá estar bien estructurada.

RNF6: Deberá emplear un formato de datos eficiente y fácil de gestionar por la aplicación cliente.

A.3. Servidor de conexiones

Requisitos funcionales:

RF1: El servidor solo registrará a los servidores que pertenezcan al sistema.

RF2: Solo se atenderán peticiones de usuarios registrados en el sistema.

RF3: Deberá mantener una conexión permanente contra los servidores activos.

RF4: Se encargará de solicitar las conexiones de datos a los servidores activos.

RF5: Deberá notificar al usuario el puerto donde se localiza la conexión de datos.

RF6: Almacenará las conexiones de datos activas.

A.4. Socket

Requisitos funcionales:

RF1: El servidor enviará las peticiones de registro en el servidor de conexiones.

RF2: Clasificar las peticiones entrantes.

RF3: Solicitar al grabador correspondiente el recurso solicitado.

RF4: Generará la conexión de datos frente al servidor de conexiones.

RF5: Mantendrá una conexión de control permanente frente al servidor de conexiones.

B

Manual de uso de la librería

A continuación, se describe toda la funcionalidad que aporta la librería de gestión de datos para permitir la comunicación entre los diversos subsistemas y la base de datos.

```
Usage: ./ClientLibrary [options]
Test helper

Options:
  -h, --help                          Displays this help.
  -v, --version                       Displays version information.
  -C, --check_user                    Check if the user is registered in the
                                      app, if it is returned 1, if not 0.
  -A, --add_user                      Create new user, if create returned 1,
                                      if not 0.
  -D, --delete_user                   Delete user, if delete returned 1, if
                                      not 0.
  -G, --get_users                     Get users, return jsondocument.
  -B, --get_user                      Get user, return jsondocument.
  -E, --add_recorder                  Create new recorder, if create returned
                                      1, if not 0.
  -J, --delete_recorder               Delete recorder, if delete returned 1,
                                      if not 0.
  -H, --get_recorders                 Get recorders, return jsondocument.
  -I, --get_recorder                  Get recorder, return jsondocument.
  -R, --connect_recorder              Connect recorder, if connected returned
                                      1, if not 0.
  -u, --username <username>           Username.
  -p, --password <password>           Password.
  -n, --new_username <new_username>   Username to create, delete or list.
  -c, --new_password <new_password>   New password.
  -e, --email <email>                 Email.
  -r, --superuser <superuser>         User rol (user or admin).
  -x, --recorder_name <recorder_name> Recorder name to create, delete or list.
  -y, --recorder_ip <recorder_ip>     Ip to connect.
  -z, --recorder_port <recorder_port> Port to connect.
```

Figura B.1: Manual librería de datos

C

Manual de comandos CURL

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X POST –data username=<username>&password=<password>&email=<email> http://discoverServer/api/user/

Cuadro C.1: Crear usuario

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X GET http://discoverServer/api/user/<username>/

Cuadro C.2: Listar un usuario

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X GET http://discoverServer/api/user/

Cuadro C.3: Listar todos los usuarios

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X DELETE http://discoverServer/api/user/<username>/

Cuadro C.4: Eliminar usuario

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X PATCH –data üsername=<username>&password=<password>&email=<email>" http://discoverServer/api/user/<username>/ (En data solo se añade lo que queramos actualizar)

Cuadro C.5: Actualizar usuario

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X POST –data name=<name>&ip=<ip>&port=<port> http://discoverServer/api/recorder/

Cuadro C.6: Crear garbador

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X GET http://discoverServer/api/recorder/<name>/

Cuadro C.7: Listar un garbador

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X GET http://discoverServer/api/recorder/

Cuadro C.8: Listar todos los garbador

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X DELETE http://discoverServer/api/recorder/<name>/

Cuadro C.9: Eliminar garbador

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X PATCH –data "name=<name>&ip=<ip>&port=<port>" http://discoverServer/api/recorder/<name>/ (En data solo se añade lo que queramos actualizar)

Cuadro C.10: Actualizar garbador

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X POST –data name=<name>&ip=<ip>&port=<port> http://discoverServer/api/camera/

Cuadro C.11: Crear cámara

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X GET http://discoverServer/api/camera/<name>/

Cuadro C.12: Listar una cámara

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X GET http://discoverServer/api/camera/

Cuadro C.13: Listar todas las cámara

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X DELETE http://discoverServer/api/camera/<name>/

Cuadro C.14: Eliminar cámara

curl -H 'Accept: application/json; indent=4' -u <username>:<password>-X PATCH –data "name=<name>&ip=<ip>&port=<port>" http://discoverServer/api/camera/<name>/ (En data solo se añade lo que queramos actualizar)

Cuadro C.15: Actualizar cámara

D

Manual de uso del cliente

Mediante este documento se pretende realizar una visita orientada hacia un correcto uso de la aplicación. Por simplificar, en la guía de uso no se mostrarán todos los mensajes de error o la vista de usuario normal, ya que está contenida en la funcionalidad de administrador.

D.1. Login

Al iniciar la aplicación, la primera ventana que aparecerá será la de login, donde el usuario deberá introducir sus credenciales y pulsar sobre el botón de login, pudiendo darse dos escenarios: si el registro es erróneo, se lanzará una ventana emergente indicando el error; y si no, permitirá el acceso a la aplicación.

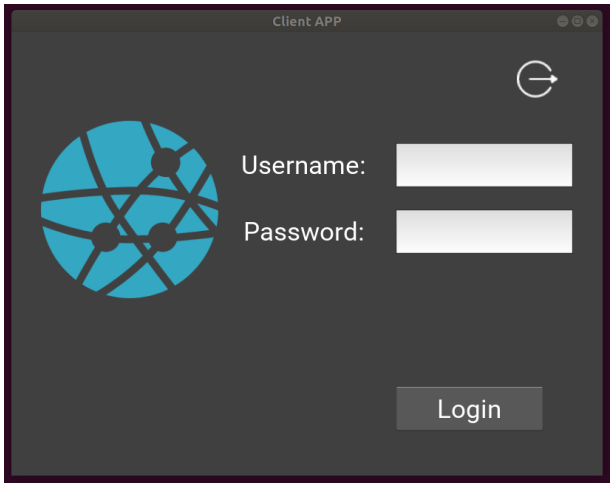


Figura D.1: Login

D.2. Vista principal en modo administrador

En esta vista, el administrador tendrá la opción de pulsar tres botones: el primero le permitirá gestionar los usuarios; el segundo, gestionar los grabadores; y el tercero, conectarse con algún grabador

para solicitar algún recurso.

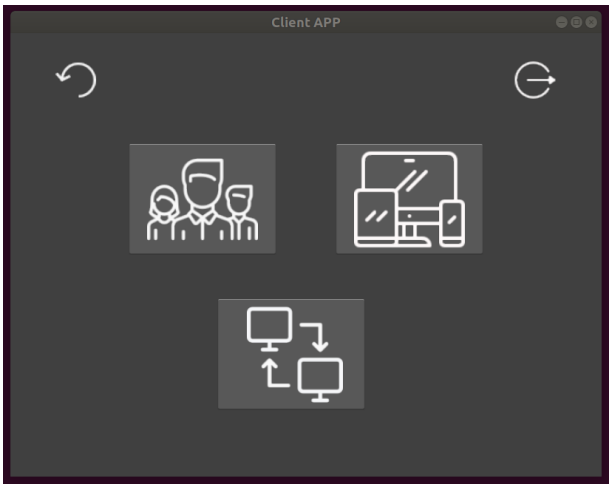


Figura D.2: Vista principal en modo administrador.

D.3. Vista gestión de usuarios

La gestión de usuarios permite al administrador: crear nuevos usuarios, eliminar usuarios registrados en la plataforma, o listar todos los usuarios para poder acceder a sus datos.

Figura D.3: Vista gestion de usuarios.

D.3.1. Añadir usuario

Lo más destacable de esta vista, son los datos a introducir en el formulario:

Username: nombre del usuario, su principal requisito es que no pueden existir dos usuarios con el mismo nombre.

Password: clave del usuario para acceder a la aplicación.

Email: dirección de correo para contactar con el usuario.

Rol: permisos que se le conceden al usuario.

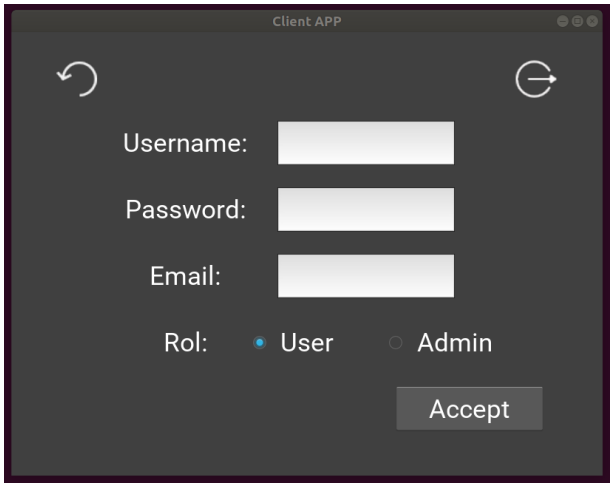


Figura D.4: Vista para añadir un usuario.

Tras rellenar todos los campos, el administrador deberá pulsar sobre el botón de aceptar, registrando el usuario en la aplicación si todos los datos introducidos son correctos. Posteriormente, se mostrará una ventana emergente para informar sobre el resultado de la acción y se cargará la vista de gestión de usuarios.

D.3.2. Eliminar usuario

Dicha vista nos mostrará un listado de todos los usuarios en forma de botón: cuando el usuario pulse sobre alguno de ellos, la aplicación eliminará al usuario correspondiente y cargará la vista de gestión de usuarios.

D.3.3. Listar usuarios

La vista es generada igual que la de eliminar usuarios, la única diferencia es la funcionalidad de los botones: cuando se pulse uno de ellos, se generará una ventana emergente mostrando los datos del usuario seleccionado.

D.4. Vista gestión de grabadores

Esta vista es idéntica a la de gestión de usuarios, nos permite realizar las mismas operaciones sobre los grabadores (crear, eliminar o listar).

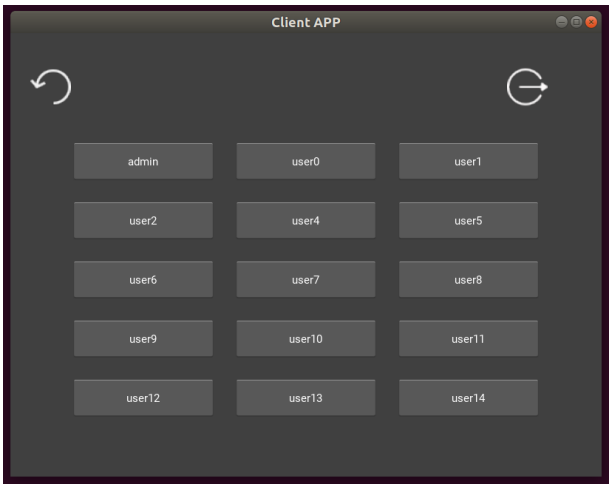
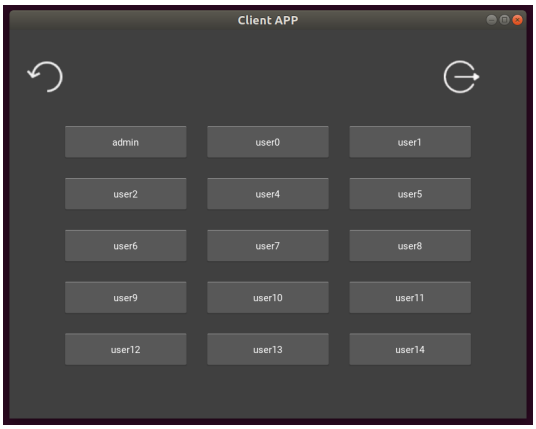
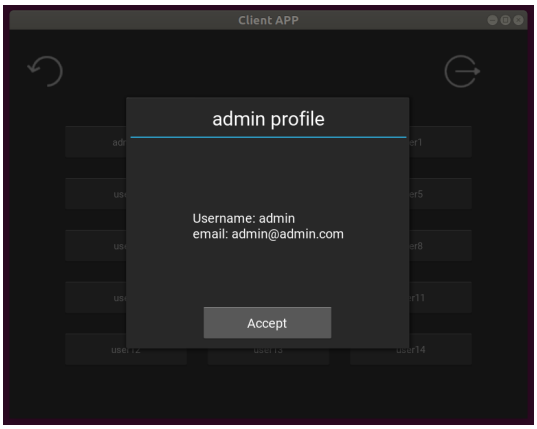


Figura D.5: Vista para eliminar un usuario.



(a) Listado de usuarios



(b) Detalle de un usuario

Figura D.6: Vistas para mostrar el detalle de un usuario

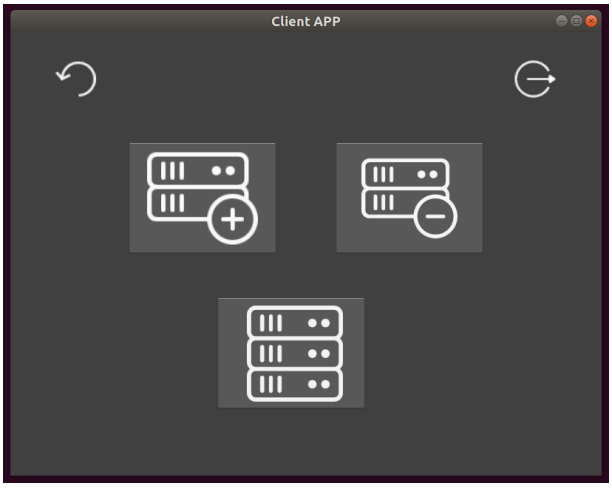


Figura D.7: Vista gestion de grabadores.

D.4.1. Añadir grabador

Al igual que en la vista de añadir usuario, lo más destacable son los datos que el cliente introduce en el formulario:

Nombre: nombre del grabador, su principal requisito es que no pueden existir dos grabadores con el mismo nombre.

IP: debe ser la dirección pública, se debe introducir con el siguiente formato AAA.BBB.CCC.DDD, donde cada bloque de letras es un número comprendido entre 0 y 255.

Puerto: número de puerto abierto en el router para acceder a la dirección IP indicada.

Tras rellenar todos los campos, el administrador deberá pulsar sobre el botón de aceptar, registrando el grabador en la aplicación si todos los datos introducidos son correctos. Posteriormente, se mostrará una ventana emergente para informar sobre el resultado de la acción y se cargará la vista de gestión de grabadores.

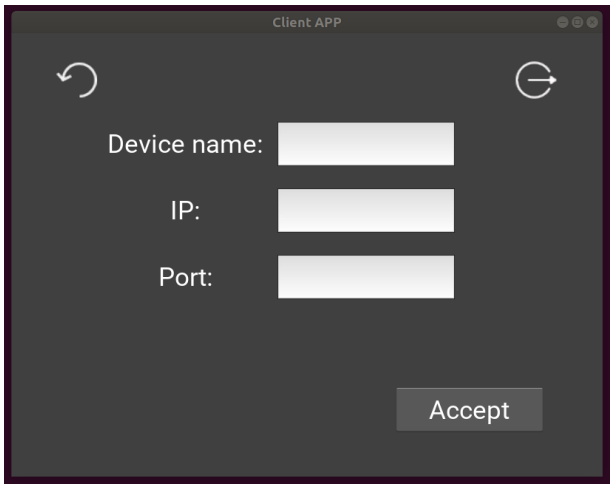


Figura D.8: Vista para añadir un grabador.

D.4.2. Eliminar grabador

Dicha vista nos mostrará un listado de todos los grabadores en forma de botón: cuando el usuario pulse sobre alguno de ellos, la aplicación eliminará al grabador correspondiente y cargará la vista de gestión de grabadores.

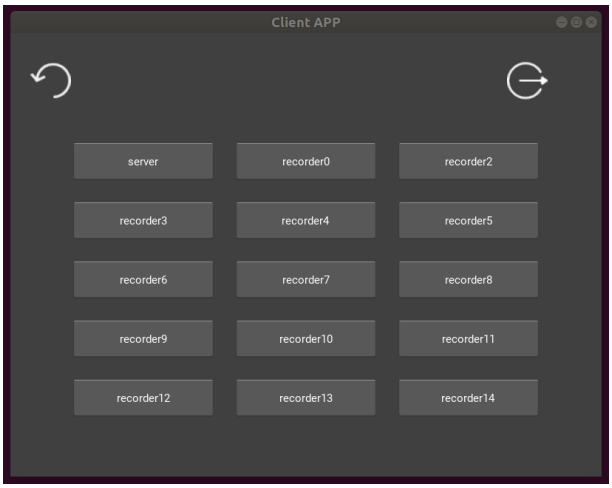
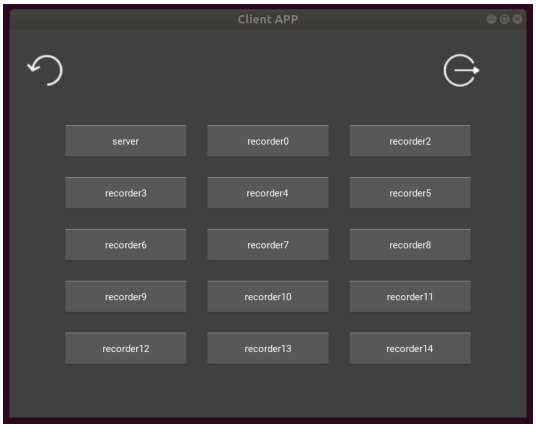


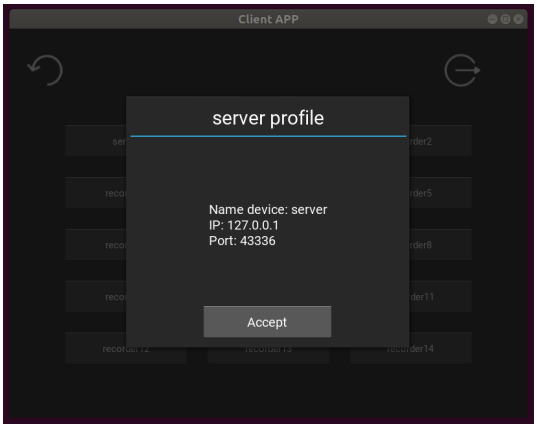
Figura D.9: Vista para eliminar un grabador.

D.4.3. Listar grabadores

La vista es generada al igual que la de eliminar grabadores o usuarios, la única diferencia es la funcionalidad de los botones: cuando se pulse uno de ellos, se generará una ventana emergente mostrando los datos del grabador seleccionado.



(a) Listado de grabadores

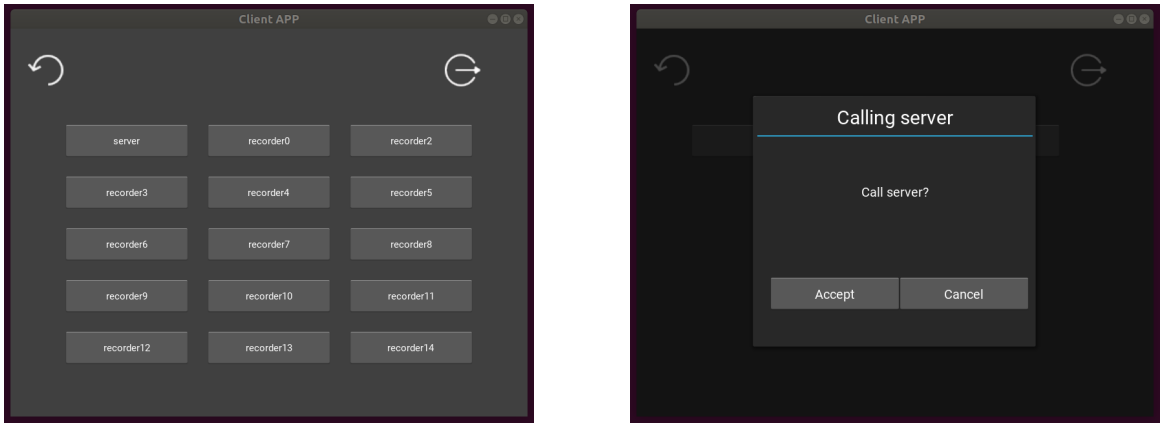


(b) Detalle de un grabador

Figura D.10: Vistas para mostrar el detalle de un grabador

D.5. Vista de conexión

Cuando un usuario independientemente del rol que posea, desee conectarse con algún grabador, se le generará la misma vista que en los casos de eliminar y listar. En el momento que pulse sobre algún botón, el sistema intentará conectarse con el servidor requerido: una vez lo consiga, mostrará un listado de todos los recursos de los que dispone dicho grabador en forma de botón. Tras este paso, el usuario podrá seleccionar cualquier recurso pulsando sobre su botón correspondiente; finalmente el sistema procederá a su descarga en el disco local del equipo.



(a) Listado de grabadores

(b) Listado de recursos

Figura D.11: Vistas para la conexión con un grabador


UAM
UNIVERSIDAD AUTONOMA
DE MADRID