Vulture Security Cam System

Sistema de video vigilancia remoto

Desarrollado por Enrique Sánchez Vicente

Repositorio:

https://github.com/EnriqueSanVic/vulture-security-cam-system

Perfil LinkedIn de Enrique Sánchez Vicente:

https://www.linkedin.com/in/enrique-sánchez-vicente

Índice

Descripción del producto

Vulture Security Cam System es una solución integral de video vigilancia para empresas y particulares, lo que implica un sistema con las siguientes capacidades:

- Retransmisión de vídeo en streaming (vídeo en directo).
- Historial de grabaciones.
- Capacidad para soportar múltiples usuarios.
- Capacidad para soportar múltiples cámaras de vídeo vigilancia para cada usuario.
- Aplicación cliente para dispositivos móviles Android con la capacidad de ofrecer el servicio al consumidor final.

Todo el sistema está enfocado al consumo del servicio de video vigilancia por el usuario final de manera remota, que se llevaría a cabo desde la aplicación móvil con capacidad para acceder al set de cámaras y poder consumir el vídeo en streaming así como el historial de grabaciones de cada cámara instalada.

Esta forma de ofrecer el servicio es idónea debido a que aporta portabilidad para el cliente, pudiendo observar la transmisión en directo de cada cámara desde cualquier parte con la facilidad y la sencillez de una aplicación móvil intuitiva y fácil de utilizar.

La filosofía del producto es abaratar costes en las componentes del sistema, haciendo especial énfasis en el uso de hardware económico en las cámaras de video vigilancia para ser un producto competitivo.

Partes del producto

El sistema Vulture tiene 3 componentes claramente diferenciados que trabajan en conjunto con una arquitectura de red cliente-servidor.

- El **Servidor** es el pilar fundamental que gestionará todo el sistema y al que se conectarán las cámaras para enviar la señal de vídeo streaming pero también también se conectarán los clientes para recibir todos los datos que requieran y la señal de vídeo en streaming para consumirla a tiempo real.

  El servidor también es el encargado de ir comprimiendo el streaming en formatos de vídeo y almacenar el histórico de grabaciones de cada cámara.

- El **Sistema de Cámaras** está compuesto por un número variable de video cámaras desarrolladas a partir de un *single-board computer* Raspberry Pi adaptado para la función de transmisión de vídeo en streaming con un módulo de cámara conectado a la placa base y una carcasa creada para la fácil manipulación y conservación del dispositivo.

  El sistema de cámaras depende de una instalación de internet en el área en el que se quiera instalar, la instalación de internet requiere de una línea y un router con capacidad de conexión de múltiples dispositivos conectados, esta conexión podrá ser inalámbrica para más facilidad de instalación pero de manera idónea para mayor velocidad de conexión y fiabilidad, la instalación debería de ser por cable de red.

- La **Aplicación Android** es el medio por el cual el cliente podrá consumir el servicio de video vigilancia, esta aplicación es intuitiva y sencilla de usar y da acceso al cliente al flujo de vídeo en streaming de cada una de sus cámaras instaladas así como al histórico de grabaciones de cada una de ellas, además podrá apagar o encender la transmisión de cada una de las cámaras.

Requisitos Hardware

El hardware del sistema se ha escogido para abaratar costes sin perder funcionalidades ni rendimiento. Recordemos que lo que se pretende es que el sistema Vulture sea asequible para el gran público, lo que es una limitación a la hora de escoger componentes.

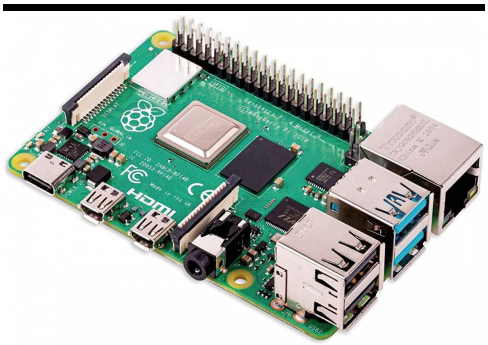
Cámara de video vigilancia

El sistema de cámaras de video vigilancia estará conformado por un micro computador y un módulo de cámara con el que se obtendrán las imágenes que conformarán el vídeo en streaming.

Micro computador Raspberri Pi

La parte más cara del producto a la hora de ofertarlo como solución para un cliente sin duda es la computadora de la cámara de vigilancia, que es un *single-board computer* Raspberry Pi.

Para poder hacer una segmentación en gamas del producto, la cámara de video vigilancia estará disponible en dos versiones, el primer modelo de cámara estará creado a partir del modelo de micro computador Raspberry Pi 4B 1Gb que es uno de los más caros de toda la gama de micro computadoras, el segundo modelo de cámara estará creado a partir del modelo Raspberry Pi Zero W que es uno de los modelo más económicos.



Raspberry Pi 4B

Raspberry Pi Zero W

La diferencia entre ambos modelos son las especificaciones de sus componentes, lo que repercutirá en el rendimiento del vídeo por streaming y la calidad de la imagen.

Esto hará que el software que controla las cámaras tenga dos configuraciones distintas para cada una de las configuraciones hardware.

La idea es que a la hora de ofrecer la solución al cliente, este escoja entre ambos modelos para conformar a su gusto la distribución de cámaras de vídeo vigilancia.

La Raspberry Pi 4B 2Gb es el modelo con el que se llevará a cabo el desarrollo inicial y la fase de pruebas del sistema debido a que es el modelo que se tiene a disposición inmediata para iniciar el desarrollo, aun así se pretende en un futuro realizar las pruebas y la implementación de la configuración del modelo Zero W.

Por esa razón todas las decisiones de diseño y requisitos del sistema se enfocan en la total compatibilidad del hadware y del software con el modelo Zero W para poder tener una fácil y rápida migración del desarrollo llegado el momento de realizar la implementación del sistema en el modelo Zero W.

Link a ambos modelos de micro computador:

Raspberry Pi 4B 1Gb: link a las especificaciones del producto

Raspberry Pi Zero W: link a las especificaciones del producto

Módulo de cámara

El módulo de cámara escogido es para recoger los fotogramas que conformarán el vídeo en streaming es el **Pi Camera Module 2**, que es el módulo de cámara oficial que ofrece la compañía.

Pi Camera Module 2

La elección de este módulo de cámara y no de otro más barato es que este módulo de cámara se programa con una librería oficial de la marca llamada Pi Camera y que tiene un buen soporte y actualizaciones, lo que garantiza la fiabilidad, robustez y seguridad del sistema a parte de ser compatible con una gran variedad de módulos de cámara de

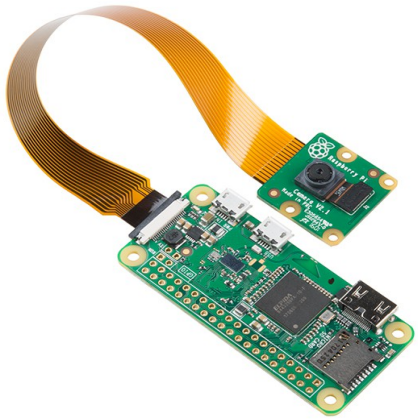
gamas superiores lo que deja abierta la posibilidad de escalar o ampliar el sistema de cámaras en un futuro pudiendo hacer implementaciones interesantes con otros módulos de cámara disponibles en el mercado como cámaras con sensores infrarrojos para implementar un modo de vigilancia en la oscuridad.

Pi Camera Module 2: link a las especificaciones del producto

Bus adaptador

Para realizar la migración del micro computador a la versión más económica del Raspberry Pi Zero W es necesario un adaptador del bus que conecta el módulo de cámara de la Pi Camera con la placa base, dicho adaptador es un bus económico ofrecido también por los distribuidores oficiales de Raspberry Pi, este adaptador se debe a que el conector del bus de cámara de las Raspberry Pi Zero W es ligeramente más pequeño que el del resto de los micro computadores pero la configuración de los pines del bus es la misma.

Bus adaptador: link a las especificaciones del producto



Configuración Raspberry Pi Zero W + Bus adaptador + Pi Camera Module 2

Servidor

El servidor es el ordenador encargado de manejar toda la lógica de negocio y gestionar todas las conexiones, tanto usuarios como cámaras de video vigilancia.

Las especificaciones hardware del servidor no están ligadas a la plataforma hardware debido a que el software encargado de manejar el servidor estará desarrollado para la máquina virtual de Java (JVM), lo que hace que el software no sea dependiente en de la plataforma sobre la que se ejecuta si no que la dependencia radica en si existe o no una implementación de la JVM para la plataforma hardware que se quiera escoger, cabe destacar que la mayoría de plataformas y sistemas operativos tienen implementaciones de la JVM.

Esto permite tener una gran maniobrabilidad a la hora de escoger el servidor que se quiera usar debido a que podemos escoger entre plataformas x86 o arm y entre multitud de sistemas operativos.

Por eficacia y seguridad, la configuración más optima sería un x86 + Linux Debian debido a que es una configuración hardware - software segura y eficiente para la tarea que se requiere.

Per o una vez más por eficacia en el desarrollo, el sistema será desarrollado sobre una plataforma x86 + Windows 10, debido a que es más cómoda a la hora de trabajar en el desarrollo y es la inmediatamente disponible.

Esto no debería de ser un problema a la hora de migrar el software desarrollado a la plataforma Linux debido a que la implementación en Java no debería de tener muchos problemas a la hora de ejecutarse sobre otra JVM diferente, aunque puede llegar a existir alguna pequeña incompatibilidad.

Factores de escalabilidad

La primera versión del servidor está pensada para poder soportar pocos usuarios y pocas cámaras por usuario, esto es debido a una serie de factores limitantes en la escalabilidad del producto y el número de clientes a los que se puede ofrecer el servicio.

La idea inicial es tener un servidor centralizado encargado de manejar la lógica de

negocio de todo el producto, pero existe un gran problema de escalabilidad que sucedería en el momento en el que empezase a aumentar el número de clientes a los que se ofrece el producto.

Los factores limitantes para la escalabilidad del producto son:

- La velocidad de conexión de la red posiblemente no fuese suficiente para satisfacer el número de conexiones y el ancho de banda que demandaría el sistema, probablemente con unas pocas decenas de cámaras de vídeo vigilancia transmitiendo streaming hacia el servidor, ya congestionaría la red.
- La velocidad de procesamiento del servidor es otro gran factor limitantes debido a que por cada cámara conectada al servidor, este tiene un hilo procesando la transmisión del streaming y comprimiendo el vídeo a ficheros de vídeo para almacenarlos, lo que consume cierta capacidad del procesador.
- La capacidad de almacenamiento es un mal inevitable a la hora de querer almacenar gran cantidad de ficheros para cada cliente como es el historial de grabaciones, este por suerte es un factor limitante que tiene soluciones inmediatas como la instalación de más capacidad o alguna solución de memoria distribuida.

Para enfrentarnos a esta serie de problemas podemos hacerlo por diferentes vías, para la velocidad de conexión podemos instalar una red mucha más capacidad y un router capaz de gestionarla, así como instalar en el servidor una tarjeta de red que soporte la potencia instalada como puede ser una tarjeta de 10 GB ethernet.

Para resolver el problema de la capacidad de procesamiento no queda más remedio que ampliar las capacidades de procesamiento del servidor llegado el momento, ya sea instalando un procesador con más capacidad o cambiando todo el equipo a otro que tenga más posibilidades. Esto podría hacerse instalando alguna de las soluciones de servidor en configuración de RAC que se ofrecen en el mercado, que son muy capaces para tareas de este estilo y muchas de ellas tienen capacidad para albergar multitud de procesadores y aumentar la capacidad de procesamiento.

Para el problema del almacenamiento habrá que tomar medidas de eliminación automática del historial de los clientes pasado determinado tiempo, pero aún así podemos necesitar ampliar frecuentemente la capacidad de almacenamiento instalada,

para esto, uno de los siguientes puntos de los requisitos hardware trata de como manejar este problema con un NAS.

En caso de adquirir una gran cartera de clientes y no poder satisfacer los requerimientos de escalabilidad habría que replantear la arquitectura de todo el sistema reformulándola en una arquitectura de micro servicios con multitud de servidores distribuidos para poder mejorar los requerimientos y capacidades del sistema en conjunto.

Configuración opcional NAS

Un NAS es un *Network Attached Storage*, es decir, otro servidor que sirve solamente para gestionar una gran capacidad de almacenamiento en forma de discos duros físicos o discos duros de estado sólido dependiendo de la configuración.

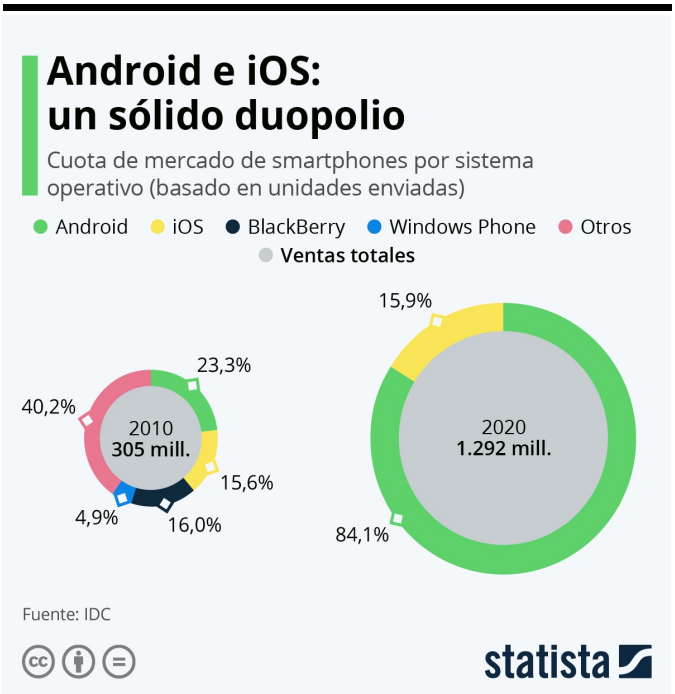
Servidor NAS

Esta configuración es muy popular para aumentar significativamente la capacidad de almacenamiento de un servidor, haciendo que se conecte a otro servidor NAS a modo de micro servicio encargado del almacenamiento debido a que suelen tener una grán capacidad para discos duros y para gestionar el espacio en conjunto en esos discos, ofreciendo también buenas velocidades de transmisión de ficheros y red de 10 GB ethernet y seguridad y persistencia para las grabaciones. Por lo general estos servidores NAS suelen tener capacidad para muchos espacios de discos y suelen soportar configuraciones en RAID para asegurar la persistencia de los datos almacenados.

Smartphone Cliente

El desarrollo de la aplicación cliente se va a realizar la plataforma Android dado que la cuota de mercado de estos dispositivos en el mercado de los smarthphones es la plataforma líder y por lo tanto facilita la recepción del gran público de la aplicación móvil del sistema Vulture debido a que la mayoría de la población tiene un dispositivo Android.

En los últimos años la plataforma ha logrado la supremacía de la cuota de mercado mundial en el sector de los sistemas operativos para dispositivos móviles alcanzando en 2020 un 84,1% de la cuota de mercado.



fuente de referencia

Versión del sistema Android

En único requisito que que debe de tener el smarthphone es que debe de ser capaz de soportar la versión Android 8.0 Oreo debido a que la app se va a desarrollar para la versión de API 26, de esta manera la aplicación sería compatible con esa versión de API y versiones posteriores, lo que se consigue con esto es que la aplicación sea instalable

en 89.1% de los dispositivos Android, dato recopilado a fecha 17/4/2022.

Esta versión de Android ha sido escogida para poder llegar al mayor número de clientes posible sin perder funcionalidades necesarias para el desarrollo.

El uso porcentual de la cuota de uso acumulada por cada versión de sistema operativo y API se puede ver en la siguiente tabla:

Version	SDK / API level	Version code	Codename	Cumulative usage [1]	Year
Android 13 DEV	Level 33	T	Tiramisu [2]	*No data*	*TBA*
Android 12	Level 32 Android 12L BETA	S_V2	Snow Cone [2]		
	Level 31 Android 12	S		9.5%	2021
	▪ `targetSdk` will need to be 31+ for new apps by August 2022 and updates by November 2022. [3] ▪ `targetSdk` will need to be 31+ for *all existing apps* by November 2023. [4]				
Android 11	Level 30	R	Red Velvet Cake [2]	45.1%	2020
	▪ `targetSdk` must be 30+ for new apps and app updates. [3] ▪ `targetSdk` will need to be 30+ for *all existing apps* by November 2022. [4]				
Android 10	Level 29	Q	Quince Tart [2]	68.9%	2019
Android 9	Level 28	P	Pie	80.7%	2018
	▪ `targetSdk` must be 28+ for new Wear OS apps and Wear OS app updates.				
Android 8	Level 27 Android 8.1	O_MR1	Oreo	87.0%	2017
	Level 26 Android 8.0	O		89.7%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	91.3%	2016
	Level 24 Android 7.0	N		94.1%	

fuente de referencia

El hecho de haber elegido esta versión de la API facilita la reutilización de código fuente debido a que esta versión de la API es compatible con el JDK 11 y a la hora de codificar la clases y la lógica relacionada con la comunicación y los protocolos de envío de streaming se podrán reutilizar ciertas clases o partes del código entre la aplicación gestora del servidor y la aplicación de Android.

Diagrama de componentes hardware

Para la comunicación de los componentes hardware del sistema se han creado los protocolos de red de bajo nivel por encima del protocolo TCP/IP que harán posible la comunicación por las conexiones socket entre los componentes del sistema para poder transmitir el streaming, la información y los ficheros necesarios.

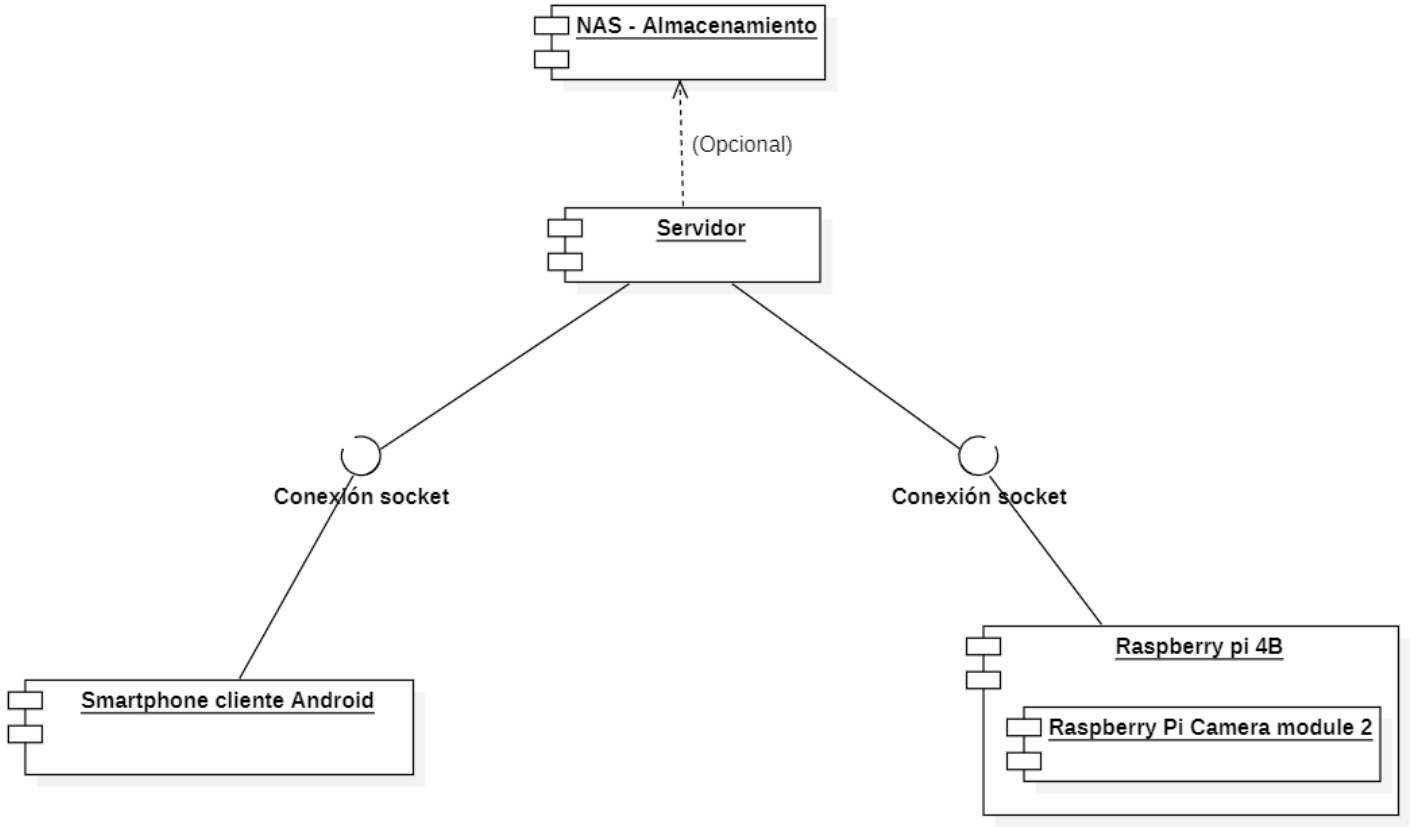


diagrama de componentes hardware

El diagrama muestra como se disponen los componentes y que dependencias tienen entre sí tienen, en este caso se ve como la Raspberry Pi Camera module 2 está dentro del componente de la Raspberry Pi 4B, es decir, es un subcomponente.

También se aprecia el componente opcional del servidor NAS y como el Servidor principal dependería de este para la gestión del almacenamiento en caso de optar por la configuración opcional del sistema.

También se aprecian las interfaces de comunicación entre el servidor y las cámaras y el servidor y los smartphones que son conexiones socket para los que se han desarrollado protocolos propietarios de comunicación de bajo nivel.

Requisitos Software

El apartado de software del producto es muy variado teniendo diferentes softwares que manejaran los distintos componentes hardwares a demás de usar software de terceros para la administración de micro servicios como puede ser el sistema gestor de bases de datos utilizado.

Software propietario

Este apartado pretende dejar claro cuales van a ser los diferentes componentes del software creado expresamente para el producto Vulture Security Cam System © y cuales serán sus responsabilidades y requisitos funcionales así como una pequeña descripción de los requisitos no funcionales de cada componente como la plataforma sobre la que se desarrolla y el lenguaje de programación en el que se codifica.

Vulture Cam

Este programa tiene el cometido del control de la cámara de video vigilancia (El micro computador Raspberry Pi y sus periféricos), es decir, debe de controlar las funcionalidades relacionadas con la captura de imagen, la transmisión del streaming y el control del estado de la cámara de vigilancia (activada o suspendida).

Este software se ejecutará sobre el sistema operativo Linux Raspbian ARM 64 bits (Distribución creada a partir de Linux Debian), que es un sistema operativo especialmente concebido para ejecutarse sobre micro computadoras Raspberry Pi, por encima de este estará el intérprete de Python que es la plataforma software sobre la que se ejecutará directamente el Vulture Cam.

El sistema lenguaje de programación escogido para este componente es Python 3.X debido a que es el lenguaje para que se ha desarrollado el software de terceros (librerías) que permite interactuar con el periférico del modulo de cámara Pi Camera Module 2.

Isotipo del lenguaje Python

Vulture Server

Este software back-end del sistema, es decir, el encargado de gestionar el servidor, por lo tanto tiene la mayor responsabilidad del sistema dado que es el que tienen la lógica de negocio y del que dependen el resto de componentes para poder funcionar.

El servidor tiene diversas responsabilidades:

- **Gestión de conexiones**: El servidor debe de gestionar las conexiones de las cámaras de video vigilancia que están transmitiendo el streaming y ser capaz de controlarlas pero también de satisfacer las peticiones de los smartphones que le soliciten información implementando dos protocolos de comunicación propietarios del sistema, uno para cada tarea respectivamente.
- **Compresión de vídeo y gestión del historial de grabaciones**: El servidor ha de poder comprimir en formatos mp4 las grabaciones de cada cámara de video vigilancia a partir de la tranmisión del streaming y almacenarlas un tiempo determinado para ofrecer un historial de grabaciones de cada una de las cámaras de video vigilancia.
- **Seguridad**: El servidor debe de garantizar la seguridad de los datos que transmite y que le son transmitidos implementando un sistema de logeo para cada cliente, de esta manera solo se le ofrecerán datos a los clientes que hayan sido verificados por el sistema de logeo.
- **Gestión de usuarios**: El servidor debe de ser capaz de asociar cámaras de vigilancia con los usuarios a los que pertenece para tener todas las partes del sistema relacionadas de tal manera que cuando un usuario solicite información de las cámaras de video vigilancia se le ofrezca solamente la de las cámaras asociadas a él.

Isotipo del lenguaje Java

Para poder llevar a cabo estos requisitos el servidor usará software de terceros como puede ser un SGBD (Sistema Gestor de Bases de Datos) o drivers y conectores para realizar la cominicación entre los servicios locales.

Para esta tarea el servidor tendrá una rica estructura de clases diferenciando en distintas entidades los artefactos del software involucrados en el programa, precisamente por esta

y otras razones que ahora serán expuestas se ha escogido Java como lenguaje de programación para codificar Vulture Server.

El desarrollo se ha hecho sobre el JDK 11 debido a que tiene funciones ampliadas en muchas de las componentes de la API respecto al JDK 8 que es la versión más estandarizada.

Java es una plataforma ideal para desarrollar el servidor dado que es una máquina virtual independiente de la plataforma hardware y el sistema operativo sobre el que se ejecuta lo que permite ejecutar el Vulture Server sobre distintas plataformas como Windows, MacOs y Linux y plataformas x86 y ARM para realizar diferentes pruebas de entorno o incluso poder migrar el servidor a otra plataforma si cambian los requisitos que rigen la elección de la plataforma escogida.

Para el desarrollo se usará un entorno del sistema operativo Windows 10 Home x86 de 64 bits debido a que es el entorno de ejecución inmediatamente disponible.

Vulture App

Este software es la parte front-end del sistema, es la aplicación cliente encargada de visualizar los datos del sistema de cámaras compuestos por la transmisión en streaming y el historial de grabaciones de cada cámara de vigilancia de un cliente.

Esta aplicación ha de ser sencilla he intuitiva y ha de ofrecer las siguientes funcionalidades:

- **Vista de logeo** en el sistema.
- **Vista del listado** de las cámaras asociadas al usuario.
- **Vista de cada cámara** donde se podrán visualizar el streaming de cada cámara o se podrán ver los clips del historial de grabaciones así como controlar el estado de la transmisión de la cámara.

Logotipo de Android

El servidor se desarrollará en Android sobre el lenguaje Java con la API 26, se ha escogido este lenguaje de programación en vez de lenguajes hibridos o kotlin debido a que es el lenguaje nativo del Android Runtime (máquina virtual java de Android) y tiene mejor accesibilidad a partes de bajo nivel de la API como a los métodos nativos de conexiones de red, cosa que facilitará el desarrollo de los protocolos de bajo nivel de

comunicación entre el Vulture App y Vulture Server.

Vulture Streaming Protocol

Protocolo de bajo nivel encargado de la transmisión del streaming entre las cámaras de vigilancia y el servidor. Es decir, pretende ser una interfaz de comunicación entre los componentes software Vulture Cam y Vulture Server.

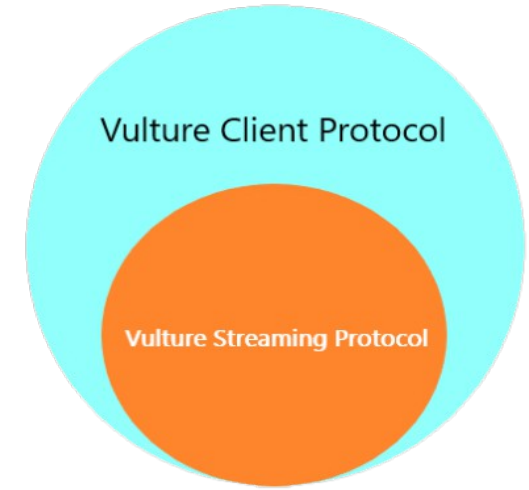
Este protocolo de comunicación es bidireccional trabaja a nivel de lenguaje máquina transmitiendo chorros de bytes y creando un código de comunicación que debe de ser interpretable tanto por el emisor como por el receptor.

El cometido principal de este protocolo es transmitir frame a frame (fotograma por fotograma) cada uno de los fotogramas que componen el video en streaming que conforma la imagen en directo. El cometido secundario del protocolo es controlar el estado de la cámara de video vigilancia, es decir, poder activar y suspender la transmisión del streaming desde el servidor.

Este protocolo usa sockets para realizar la comunicación entre ambas máquinas y está construido sobre el protocolo de internet TCP/IP.

Vulture Client Protocol

Protocolo de medio - bajo nivel encargado de la transmisión de todo tipo de datos entre el servidor y los smartphones Android cliente, es una interfaz de comunicación entre el cliente y el servidor, podría decirse que es una API con una arquitectura propietaria.



Esquema de conjuntos de protocolos

Este protocolo usa el protocolo Vulture Streaming Protocol de manera integrada para la transmisión del streaming entre el cliente y el servidor, es decir, el protocolo Vulture Client Protocol es un conjunto de instrucciones de comunicación para la transferencia de datos que contiene al conjunto Vulture Streaming Protocol.

También usa formatos de medio nivel como el uso de cadenas de caracteres en codificación UTF-8 y el uso de documentos Json para facilitar la lectura y escritura de los datos por medio de estructuras de datos de alto-alto nivel para tener facilidades cuando se requieran en el envío de datos.

El cometido principal de este protocolo es comunicar el software Vulture Server y Vulture App siendo capaz de:

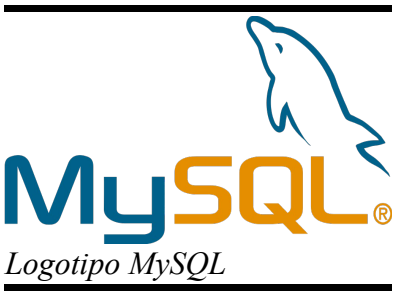
- Transferir los datos del usuario para realizar el sistema de logeo de usuarios.
- Transferir el listado de las cámaras disponibles del usuario.
- Transferir streaming de las cámaras del usuario (Vulture Streaming Protocol).
- Transferir un listado del historial de grabaciones de una cámara.
- Transferir ficheros mp4 para enviar las grabaciones.

Software de terceros

Este apartado pretende dejar claro cuales serán las responsabilidades y requisitos no funcionales de los distintos softwares que se van a utilizar en el proyecto y que no han sido expresamente creados para este proyecto si no que son software *open source* o productos software de empresas y compañías.

MySQL

Este programa será el SGBD (Sistema Gesto de Bases de Datos) del servidor y servirá para organizar la información de manera estructurada y relacionada para satisfacer las necesidades de almacenamiento y persistencia de datos del sistema.



Logotipo MySQL

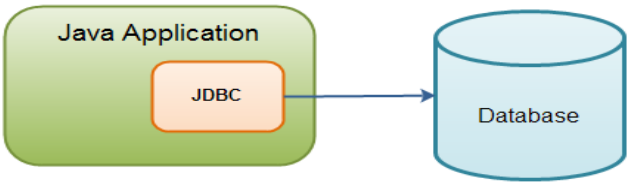
Existen muchos SGBDs pero he escogido este por que es *open source* y por lo tanto de uso libre y por que es uno de los SGBSs más usados y con más recorrido del mundo por

lo tanto tiene un buen mantenimiento y una gran comunidad de desarrolladores, lo que facilita el proceso de desarrollo y el mantenimiento del sistema.

Enlace de MySQL: link al software

JDBC Connector

El JDBC conector es una interfaz de comunicación entre MySQL y cualquier software codificado en Java, en este caso se conectará en con Vulture Server.



El conector es un driver .(jar) codificado en java para complementar al package java.sql del JDK para manejar SGBDs desde cualquier código Java, con esto Vulture Server puede comunicarse con MySQL por medio de sentencias SQL, se pueden realizar todas las operaciones SQL habituales para manejar los datos almacenados en la base de datos y recepcionar y recorrer los sets de datos extraídos de manera sencilla.

Enlace de MySQL Connectors: link al software

Librería PiCamera2

Pi Camera el la librería codificada en Python 3.x para manejar el periférico Pi Camera Module 2 desde un software codificado en Python, esta librería está desarrollada por la compañía Raspberry Pi y es la librería oficial para manejar el periférico por lo que es una librería bien mantenida y fiable.

Link descripción PiCamera2: link a la descripción

Link documentación PiCamera2: link a la documentación

Diagrama de componentes software

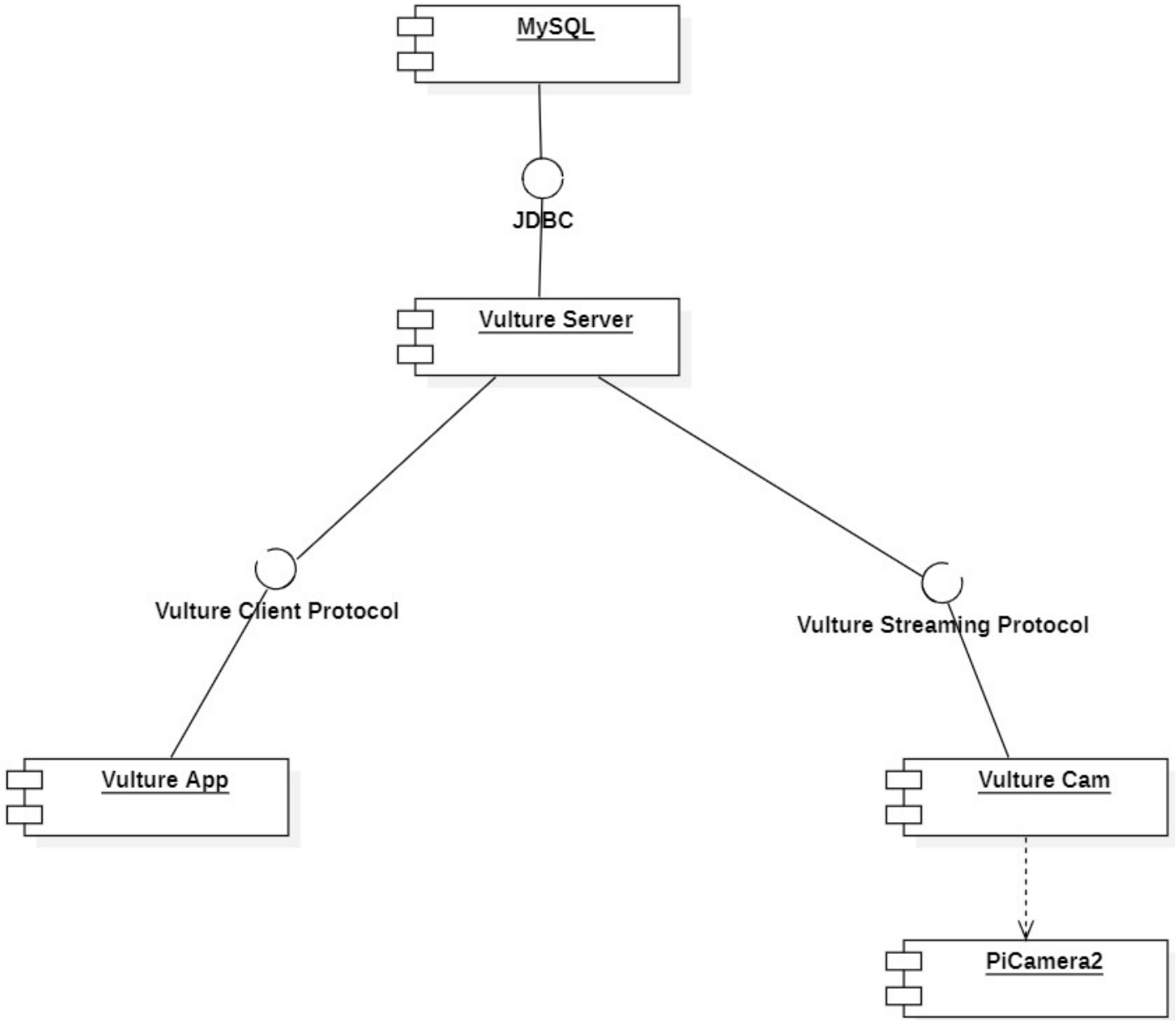


diagrama de componentes software

Se puede observar como los componentes dependen unos de otros y como ciertos componentes actúan como interfaces de comunicación entre los componentes principales, en este caso los componentes propietarios principales se comunican por medio de protocolos de los comunicación.

También podemos observar como Vulture Server se comunica con MySQL por medio del JDBC que hace de interfaz de comunicación a pesar de que ambos softwares residen en la misma máquina.

Todas las relaciones de este diagrama son relaciones de dependencia de componentes, lo que significa que las relaciones señalan las interdependencias que tienen los distintos componentes software para funcionar correctamente.

Existen dos tipos de relaciones:

- Relaciones de dependencia simple.
- Relaciones de dependencia de una interfaz.

Las dependencias simples son como las que hay entre el componente Vulture Cam y PiCamera2, tienen un sentido de dependencia y comunicación directo entre los componentes.

Las relaciones de dependencia de una interfaz son como las que hay entre Vulture Server y Vulture App por medio de la interfaz del protocolo de comunicación Vulture Client Protocol, el significado de estas relaciones es de dependencia directa pero de comunicación indirecta, los componentes se comunican entre sí por medio de otro componente interfaz que media en esa comunicación pero el significado de la relación de dependencia se refiere a que los componentes requieren de esa interfaz de comunicación para funciona.

Datos

Los datos y su gestión son una parte muy importante del sistema debido a que es la forma que tenemos de relacionar todos los clips de grabación con la cámara a la que pertenece y cada cámara de video vigilancia con el usuario al que pertenece.

El sistema tiene un diseño de datos sencillo dado que no es un sistema enfocado en el manejo y movimiento de datos estructurados como los que se almacenan en una base de datos si no que principalmente están enfocado en la transferencia de vídeo en streaming, que es un formato poco convencional y que requiere un tratamiento específico tanto para su transferencia como para su almacenamiento.

Diagrama Entidad – Relación

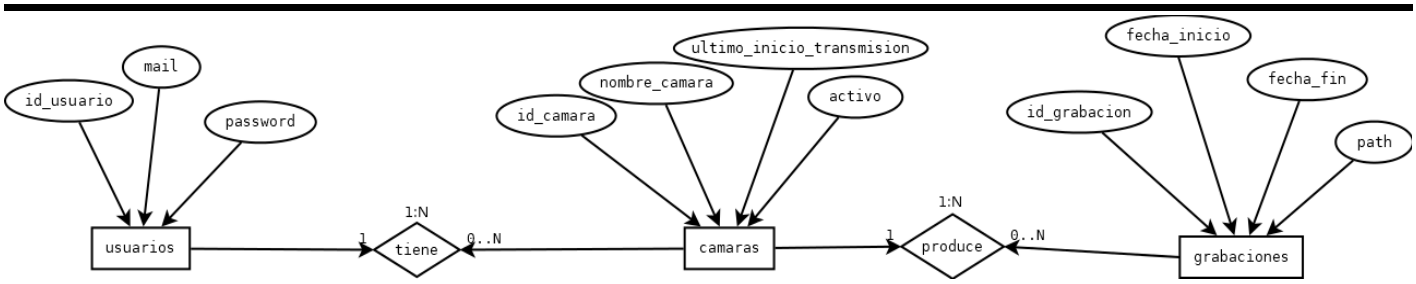


Diagrama E/R

Podemos observar como se relacionan las diferentes entidades del sistema, en este caso la entidad del usuario tiene 0..N cámaras relacionadas con el, es decir podría tener una o ninguna cámaras. De forma parecida cada cámara tiene 0..N grabaciones relacionadas, el conjunto de los registros de grabaciones relacionados con una cámara conforman su historial de grabaciones.

En la tabla de grabaciones se guarda el path (ruta del fichero) al que apunta el registro.

Gestión de ficheros

El sistema tiene que ser capaz de gestionar y almacenar grandes cantidades de ficheros de vídeo mp4 que tendrá que organizar en un sistema de ficheros para poder disponer de esa información en el futuro, este set de ficheros son los historiales de grabación de las cámaras de los usuarios del sistema.

Para organizar los ficheros de forma inequívoca el sistema de ficheros distribuirá a los usuarios por directorios teniendo cada usuario un directorio único identificado de forma inequívoca por la siguiente sintaxis:

user_<id_usuario>

Dentro de cada directorio de usuario habrá subdirectorios que estarán asociados a cada cámara de video vigilancia, habiendo un directorio por cada cámara de vigilancia identificado de forma inequívoca por la siguiente sintaxis:

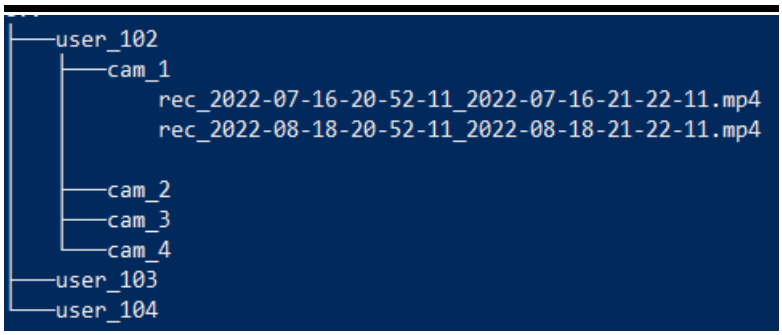
cam_<id_camara>

Dentro de cada directorio de cámara estarán los clips de video (ficheros mp4) de las grabaciones del historial de dicha cámara. Estas grabaciones se identifican de manera inequívoca con la siguiente sintaxis:

rec_<fecha y hora inicio>_<fecha y hora fin>.mp4

Las fechas y horas irán un formato adaptado basado en el formato del tipo de dato DATETIME de MySQL: ***YYYY-MM-DD hh:mm:ss*** → ***YYYY-MM-DD-hh-mm-ss***

De esta manera el árbol de ficheros quedaría con una distribución similar a esta:



Ejemplo de árbol de ficheros

Protocolos de comunicación

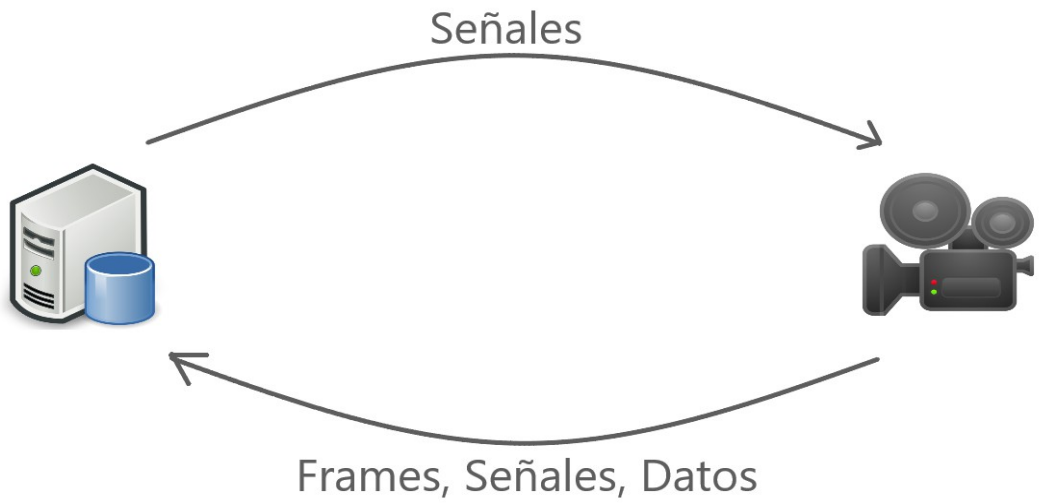
Vulture Streaming Protocol

Es un protocolo de comunicación entre el servidor y cada una de las cámaras de video vigilancia que trabaja a nivel de bytes encargado de la transmisión de vídeo en streaming a través de una conexión mediante un network socket con capacidad para enviar paquetes de datos que contiene información sobre el estado de la conexión, comandos a modo de señales que transmiten sentencias que se deben ejecutar y el contenido de cada frame de la transmisión de vídeo.

Esto permite que ambas partes de la comunicación (servidor y cámara) tengan una comunicación estable y síncrona del vídeo en streaming y se puedan realizar acciones como la suspensión temporal del envío, la reanudación o el apagado total de la cámara.

Flujo de datos

El flujo de datos es bidireccional, es decir, el protocolo puede mantener dos canales de comunicación entre la cámara y el servidor (principalmente pensado para el envío de los frames que componen la transmisión de vídeo) o entre el servidor y la cámara (pensado para la transmisión de señales/comandos de acción que la cámara debe efectuar).



Fases de la comunicación

La comunicación se compone de varias fases, algunas de ellas suceden de forma ordenada y síncrona pero otras fases pueden suceder de manera simultánea en los dos canales de comunicación del socket.

Las máquinas implicadas en la comunicación tiene que tener un comportamiento predeterminadado en cuanto al formato de los paquetes de datos que esperan recibir en la transmisión, lo cual dificulta la tarea de enviar distintos paquetes de datos con distintas longitudes de bytes y distintos tipos de datos.

Por ello en cada paso de la comunicación, ambas máquinas deben de conocer todas las características de los paquetes de datos que se espera que en un determinado comento de la counicación se envíen por el canal de comunicación.

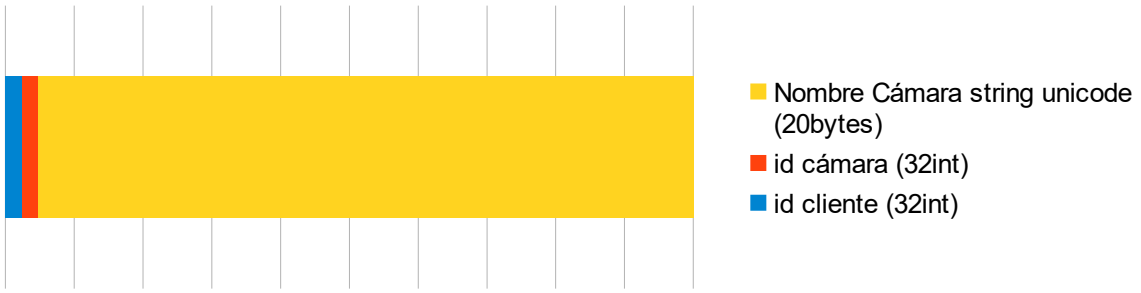
Fase de conexión: Para que la comunicación pueda iniciarse, la cámara debe conocer le dirección IP del servidor, este tiene al software Vulture Server escuchando en el puesto 4548 listo para admitir nuevas conexiones. Cuando una cámara quiera abrir una conexión nueva, el servidor siempre la aceptará en primera instancia. En este proceso de comunicación el protocolo Vulture no manda paquetes de datos de forma explícita, pero si que se envían paquetes de datos de los protocolos de las capas de comunicación inferiores como los paquetes relacionados con el protocolo TCP/IP que indican a la máquina receptora de la petición datos sobre esta como la dirección IP de la máquina que ha solicitado la conexión.

Fase de identificación: Una vez se abierta la conexión cuando el servidor acepta a la cámara, esta se tiene que identificar ante el servidor para validar la conexión entrante, esto se realiza mediante el id del usuario, el id de la cámara y el nombre de la cámara, estos son datos estáticos configurados en la cámara y que se le darán al servidor para que compruebe que tanto la cámara como el usuario al que dice pertenecer están relacionados entre sí y registrados en la base de datos, en ese momento el servidor enviará una señal a la cámara admitiendo la conexión o denegándola en caso de no haber sido posible autentificar la procedencia de la conexión.

En esta fase de la conexión si que se envían los primeros paquetes de datos del protocolo, compuesto (en orden de envío) por:

Significado	Especificación	Tipo de dato
id del cliente al que pertenece la cámara	nº entero de 32 bits con signo	signed int (32bit)
id de la cámara	nº entero de 32 bits con signo	signed int (32bit)
nombre de la cámara	cadena de caracteres de longitud fija de 160 bits	string unicode (20 bytes)

Quedando el primer paquete de la fase de identificación de esta manera:



Fase de transmisión del streaming:

Esta es quizás la fase más relevante de todo el protocolo, en ella se realiza la transmisión frame a frame del vídeo en streaming y se vasa en enviar estos frames en bucle en un tiempo indeterminado hasta que se interrumpa la conexión por una señal/comando de acción como son los de apagado o pausado de la transmisión o por factores externos al protocolo.

Este bucle consta de dos sencillos paquetes de datos:

El **primero es un número entero** con signo de 32 bits que tiene dos funciones en la transmisión.

Estas funciones dependerán del signo del número que se envíe.

Por una parte si el número es positivo, toma el significado de el tamaño en bytes del siguiente paquete (paquete del frame), de esta manera, el servidor sabrá cual es la medida del siguiente paquete que debe de leer para recibir el frame dado que el tamaño de los frames es variable. Entonces cuando el servidor recibe el primer paquete entiende que la transmisión continúa de manera satisfactoria y procede a esperar y procesar el siguiente frame de la transmisión.

Por otra parte si el número es negativo, toma el significado de una señal/comando de acción proveniente de la cámara de la cámara. Estas señales pero suelen ser de confirmación de apagado o de pausa de la transmisión. En el momento en el que el servidor recibe este paquete con un número negativo, procede a analizar la acción y el significado que tiene dicha señal para actuar en consecuencia.

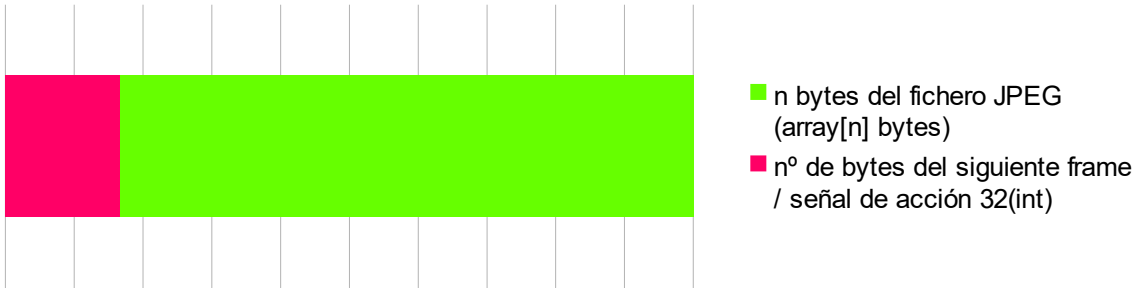
Señales habilitadas:

- ***código: CONFIRM_STOP_STREAMING_FROM_CAMERA_SIGNAL valor:*** `-2341`
  Confirma al servidor que la cámara se ha pausado sin cerrar el socket, para que este se quede a la escucha de reanudaciones.
- ***código: CONFIRM_SHUTDOWN_CAMERA_FROM_CAMERA_SIGNAL valor:*** `-6732`
  Confirma al servidor que la cámara va a la conexión para finalizar la transmisión.

El segundo paquete consta de los n bytes que componen el frame codificado en JPEG (jpg). Cuando el servidor recibe este paquete de datos procede a procesar la imagen para codificarla en MPEG-4 (mp4) y almacenarla en el clip vídeo actual o redireccionar el frame a los escuchadores que lo requieran (La vista de administración o un dispositivo cliente conectado al streaming), <más adelante>.

Significado	Especificación	Tipo de dato
nº de bytes del siguiente frame / señal de acción	nº entero de 32 bits con signo	signed int (32bit)
bytes del frame	n bytes del fichero JPEG	array[n] bytes

Quedando el paquete de la fase de transmisión de esta manera:



Fase de señales/comandos de actividad entre el servidor y la cámara: hasta ahora todas las fases anteriores se realizaban en el sentido cámara → servidor y de manera síncrona, pero esta fase de la comunicación se realiza en el sentido servidor → cámara y de manera asíncrona con el resto de fases, es decir, puede suceder en cualquier momento mientras se realizan el resto de fases en el otro sentido de la comunicación.

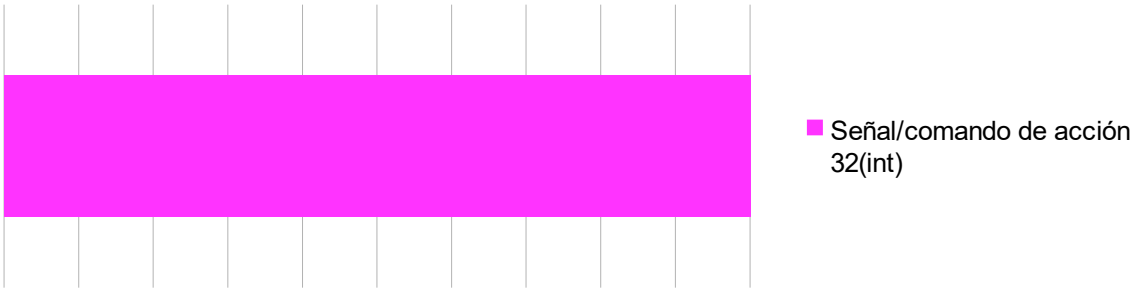
Este sentido de la comunicación solo admite el envío de paquetes de datos individuales de señales de números enteros positivos.

Señales habilitadas:

- *código:* ***START_STREAMING_TO_CAMERA_SIGNAL*** *valor:* 54
- *código:* ***STOP_STREAMING_TO_CAMERA_SIGNAL*** *valor:* 34
  Ordena a la cámara la acción de detener el streaming sin cerrar el socket.
- *código:* ***SHUTDOWN_CAMERA_TO_CAMERA_SIGNAL*** *valor:* 78
  Ordena a la cámara la acción de finalizar la transmisión cerrando el socket.

Significado	Especificación	Tipo de dato
Señal/comando de acción	nº entero de 32 bits con signo	signed int (32bit)

Quedando el paquete de la fase de señales/comandos entre el servidor y la cámara de esta manera:



Vulture Client Protocol

Este protocolo se usa entre el smartphone del cliente y el servidor y hereda grán parte del funcionamiento del protocolo Vulture Streaming Protocol dado que el software Vulture App puede consumir el streaming de una cámara del cliente, en este caso, se heredan los métodos de funcionamiento y la lógica del protocolo de envío de imágenes cámara → servidor pero ahora en el sentido servidor → smartphone modificando parte de la lógica y añadiendo algunos datos y fases de la comunicación extras para satisfacer los requerimientos.

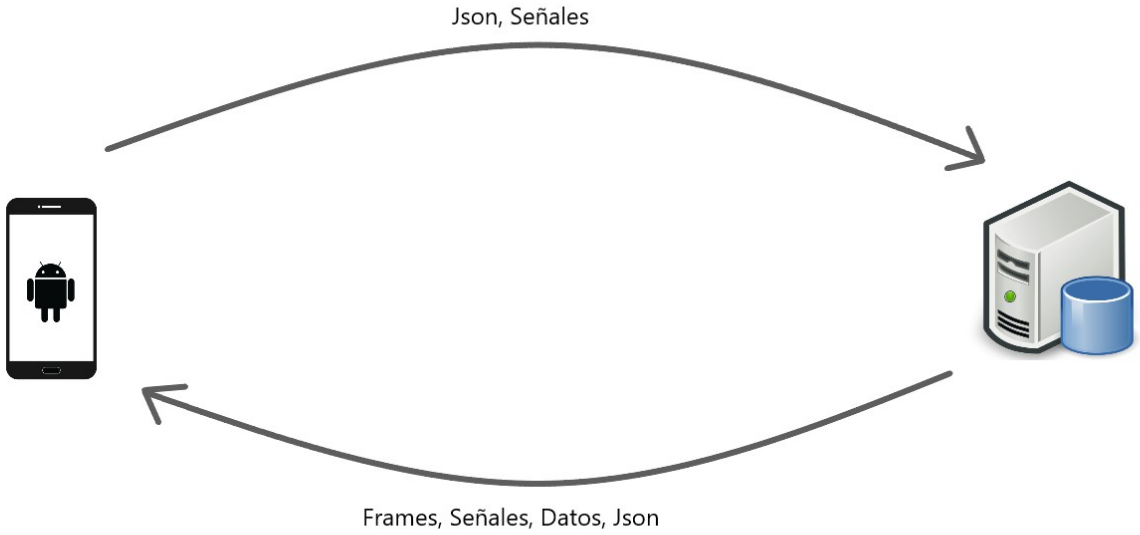
Este protocolo es en grán parte de alto nivel dado que se van a usar estructuras de datos como Json para el envío de información sobre las cámaras y las credenciales de logueo del usuario, pero también tiene el componente de bajo nivel de la lectura de bytes del socket heredado del protocolo de streaming.

En este caso, para los dátos de alto nivel se usará la lectura de socket por medio del

caracter separador del salto de línea \n para poder diferenciar los paquetes enviados por el socket, hasta que se requiera de la lectura de estructuras de bytes como pueden ser las señales o el bucle de streaming.

Flujo de datos

En este caso el flujo de datos es más complejo habiendo mucha más variedad de tipos de datos y estructuras de datos usadas para la comunicación. El flujo de datos, al igual que en el primer protocolo, es bidireccional entre el smarthphone → servidor quedando de esta manera.



Fases de la comunicación

El igual que en el protocolo de streaming, en este existen distintas fases de la comunicación y algunas de ellas son síncronas entre sí y otras son asíncronas, en este caso, el protocolo ofrece más flexibilidad respecto al orden de los pasos que el protocolo debe de seguir dado que en varias ocasiones el usuario puede tomar diversas decisiones que implican una fase de comunicación u otra dependiendo del caso de uso en el que nos encontremos, por eso muchas de las fases de comunicación de datos de alto nivel van a poder ser reemplazables unas por otras para que, dependiendo del momento el smarthphone pueda solicitar diferentes datos sin condición, esto es posible

dado que estas fases de cominicación de alto nivel van empaquetadas en estructuras Json de cadenas de caracteres y son individuales,, es decir, no suponen un flujo constante de paquetes, actúa de manera similar al protocolo http. Esto sí ocurre en las fases de cominicación heredadas del protocolo de streaming, que requieren de envíos constantes para satisfacer el flujo de frames entre una máquina y otra.

Fase de conexión: Esta fase es heredada del protocolo Vulture Streaming Protocol y funciona de la misma manera con la sutil diferencia de que la conexión no se realiza a través de puerto 4548 si no que se realiza a través del puerto 4547 para tener una clara diferenciación de las conexiones de cámaras y de smartphones.

Fase de logueo: En esta fase el smarthphone debe de identificarse enviando un Json con su correo electrónico de clinete y su contraseña, estra contraseña se cifra por medio del algoritmo de hash SHA256 para mayor seguridad del envío. El servidor responderá con otro Json confirmando la autenticación del cliente o denegándola, en ese momento si se deniega cerrará el socket y el smartphone también.

Estructura del Json de petición de smartphone:

```
{
  "request": "login",
  "user": "user@example.com",
  "password": "f90e22675ced9c138e1935a8246a5d47ffcb14763a25eb
    7fb2f3984876741c01"
}
```

Estructura del Json de respuesta del servidor:

```
{
  "status": "true"
}
```

Fase de listado de cámaras: En esta fase el smartphone solicitará al servidor la lista de cámara del client enviando un Json con la siguiente estructura:

```json
{
  "request": "list_of_cams"
}
```

Y el servidor tiene que responder con un Json con la siguiente estructura:

```json
{
  "status": "true",
  "list_cams": [
    {
      "id": 1,
      "name": "Entrada del jardín"
    },
    {
      "id": 2,
      "name": "Puesta delantera"
    },
    {
      "id": 3,
      "name": "Habitación bebé"
    },
    {
      "id": 4,
      "name": "Cocina"
    }
  ]
}
```

Fase de solicitud de clips de una cámara: En esta fase el smartphone solicitará al servidor la lista de clips de una cámara concreta del cliente, la petición del smartphone tendrá la siguiente estructura:

```json
{
  "request": "list_of_clips",
  "cam_id": 12
}
```

Estructura del Json de respuesta del servidor:

```json
{
  "status": "true",
  "cam_id": 12,
  "list_of_clips": [
    {
      "id": 2432,
      "date_time": "2022-5-16 01:40:36 - 2022-5-16 02:10:36"
    },
    {
      "id": 2411,
      "date_time": "2022-5-16 01:40:36 - 2022-5-16 02:10:36"
    },
    {
      "id": 1344,
      "date_time": "2022-5-16 01:40:36 - 2022-5-16 02:10:36"
    }
  ]
}
```

Fase de solicitud de un clip: En esta fase el smartphone solicitará el envío de un clip en concreto, la primera fase es una solicitud Json donde se solicita el clip, la respuesta del servidor es otro Json con el status de la respuesta y un campo que contiene el tamaño en bytes del siguiente envío que se va a realizar del clip. En ese momento tanto el servidor como el smartphone so pondrán en modo de lectura escritura del socket de bajo nivel a nivel de byte para enviar y recibir el binario del fichero MPEG-4 del vídeo del clip.

Estructura del Json de la petición del smartphone:

```json
{
  "request": "clip_download",
  "clip_id": 2411
}
```

Estructura de la respues Json del servidor:

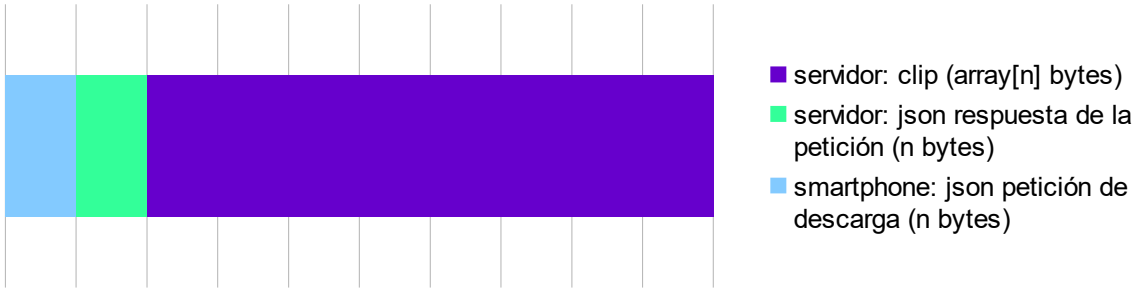
```json
{
  "status": "true",
  "n_bytes_of_clip": 20453
}
```

En este punto el servidor escribirá en el socket los bytes del fichero del clip y el cliente leerá los n bytes que le ha comunicado el servidor que debe de leer del socket.

Los paquetes de este envío quedaría estructurados en este orden:

Significado	Especificación	Tipo de dato
Petición de descarga de un clip por id.	Cadena de caracteres Json de longitud n (hasta encontrar carácter \n)	String unicode (n bytes)
Respuesta de la petición indicando el nº de bytes que se van a enviar a continuación por el socket.	Cadena de caracteres Json de longitud n (hasta encontrar carácter \n)	String unicode (n bytes)
bytes del clip	n bytes del fichero MPEG-4	array[n] bytes

Quedando la estructura de los paquetes de esta manera:



Fase de solicitud de streaming de una cámara: En esta fase el smartphone solicita al servidro iniciar la parte del protocolo de streaming para que este le redirija el streaming Si de la cámara. Esta hase contiene la fase de transmisión del protocolo Vulture Streaming Protocol. En esta fase el smartphone manda un petición Json que contiene el id de la cámara sobre la que solicita abrir la transmisión, con este formato:

```
{
  "request": "streaming_transmision",
  "cam_id": 23
}
```

Y el servidor le responde con esta respuesta Json:

```
{
  "status": "true"
}
```

Si el campo de status de la respuesta del servidor el positiva entonces se comenzará la fase de transmisión del protocolo de streaming, en este caso el servidor hace el rol de productor del streaming y el cliente hace el rol de consumidor del streaming, de forma homóloga a como actuan la cámara con el servidor en el protocolo original.

En la fase de transmisión están todos las señales/comandos que irán redireccionados a la cámara desde el servidor pero se añade una señal más que es la señal de parar la transmisión entre el servidor y el smartphone.

Es decir, las señales originales se conservan de la misma manera pero se añade una nueva señal que interrumpe la transmisión del straming del redireccionamiento del servidor al smartphone.

- ***código: CLOSE_REDIRECTION_STREAMING_FROM_SMARTPHONE_SIGNAL valor: -877***
  Confirma el cierre de la fase de transmisión entre el servidor y el smartphone, no cierra la conexión entre el servidor y la cámara. Solamente afecta a la parte del cliente y este comando se puede mandar en ambas direcciones, es decir, tanto el servidor como el smartphone puede enviarla en cualquier sentido del canal de comunicación para para la transmisión.

Vulture Cam

En este apartado se realizará una descripción técnica del software Vulture Cam que controla las cámas de video vigilancia.

Este software está desarrollado en python 3.x y usa la librería oficial del periférico de la cámara modular PiCamera2 para el manejo del periférico.

El software a grandes rasgos realiza la conexión con el servidor abriendo un socket que manejará por medio de dos hilos, uno para el input del socket y otro para el output del socket.

El hilo encargado del output también es el encargado de manejar el periférico de la cámara y realizar la toma de los frames del streaming, procesarlos y enviarlos al servidor.

El hilo del input es el encargado de escuchar las señales del servidor y realizar cambios

en el sistema, en este sentido podemos decir que es el que lleva el control del estado del sistema.

Descripción del funcionamiento

El software está dividido en varios paquetes, pero los más relevantes son el fichero lanzador del programa llamado vulture_cam.py, este fichero contiene la función main() que inicializa el sistema.

Luego está el paquete de streaming_system.py que contiene las clases del proyecto, están todas agrupadas en un mismo código fuente debido a que tienen que lidiar con el problema de las referencias circulares que es un problema que acarrea Python por ser un lenguaje interpretado, lo que a veces conlleva tener varias clases en el mismo fichero.

También están los ficheros de binary_utils.py que contiene métodos para el manejo de datos de bajo nivel y el fichero de constants.py que contiene las constantes de configuración del proyecto para que estén accesibles todas desde el mismo punto.

El main() crea el socket y abre la conexión y crea dos hilos, uno para la escucha del input del socket y otro para el output y para realizar el manejo del periférico de la cámara, son las clases SocketListenerThread y StreamingThread respectivamente.

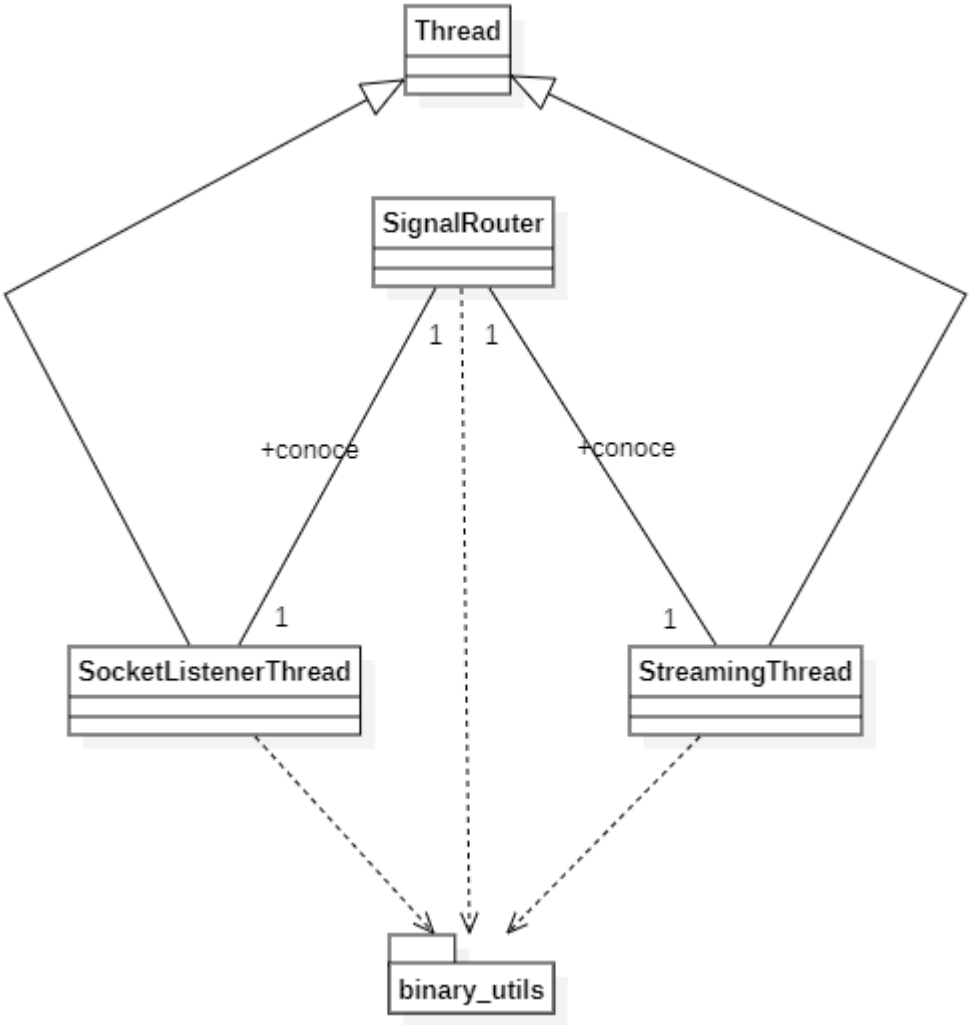
También se crea una clase a modo de controlador del sistema llamada SignalRouter que es un objeto también creado por el main() que conoce a ambos hilos y se encarga de tomar las decisiones de acción cuando se requieren ya sean por problemas en el hilo de streaming o por señales de acción recibidas a través del hilo de input del socket.

La captura del vídeo se realiza a través del método:

```
for iter in self.camera.capture_continuous(output=buffer, format=FRAME_FORMAT, use_video_port=False):
```

Que devuelve una colección de Python que genera un elemento en cada colección que es un frame capturado por la cámara, por lo tanto si se itera (está optimizado para ello) en un bucle for se puede obtener una captura infinita de frames con una función habilitada o bloque de código que procesa el frame. Dentro de este bucle también se envían los bytes del frame por el socket hacia el servidor.

Diagrama de clases



Como podemos ver en el diagrama, las clases StreamingThread y SocketListenerThread heredan de la clase Thread que es una clase del SDK de Python que permite manejar la concurrencia con hilos, muy parecido a como ocurre en Java con su clase homóloga.

También se puede observar como solo existe una instancia de cada clase y que las clases de los hilos solamente conocen a la clase SignalRouter que es el controlador del sistema que básicamente tiene la función de interpretar señales y manejar el sistema de manera coordinada.

Todas las clases dependen del paquete binary_utils por que este contiene los métodos usados para hacer transformaciones de tipos de datos a arrays de bytes (que son los que se reciben por el socket) para poder recibir y enviar datos de bajo nivel.

Vulture Server

{{falta, irá en la documentación final}}

Descripción del funcionamiento

{{falta, irá en la documentación final}}

Administración

{{falta, irá en la documentación final}}

Diagrama de clases

{{falta, irá en la documentación final}}

Vulture App

{{falta, irá en la documentación final}}

Descripción del funcionamiento

{{falta, irá en la documentación final}}

Diagrama de clases

{{falta, irá en la documentación final}}

Diagrama de casos de uso

{{falta, irá en la documentación final}}

Bibliografía y Fuentes

{{falta, irá en la documentación final}}