Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño e implementación de un bot para la plataforma de mensajería Telegram que permita la gestión remota de una cámara de seguridad

Design and implementation of a bot for the messaging platform Telegram that allows the remote management of a security camera

Autor

Arturo Manuel Dito Iranzo

Director

Jorge Sancho Larraz

Ponente

Álvaro Alesanco Iglesias

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. ARTURO MANUEL DITO IRANZO,

con nº de DNI 72990607-S en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Diseño e implementación de un bot para la plataforma de mensajería Telegram que permita la gestión remota de una cámara de seguridad,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 20 de septiembre de 2017

Fdo:

Agradecimientos

A Jorge y a Álvaro, especialmente, por dirigir, coordinar y realizar conmigo este proyecto, por haber depositado su confianza en mí y por brindarme la oportunidad de aprender junto a ellos.

A mi familia y amigos, por transmitirme siempre su apoyo, su ayuda, su motivación y su cariño.

Diseño e implementación de un bot para la plataforma de mensajería Telegram que permita la gestión remota de una cámara de seguridad

RESUMEN

En los últimos años, las aplicaciones de mensajería instantánea han logrado posicionarse como las más utilizadas dentro del mercado de las aplicaciones móviles, ahora engloban en sí mismas servicios que anteriormente se desarrollaban en aplicaciones diferentes, como el envío de archivos, las videollamadas o las grabaciones de audio, por ejemplo.

Esto nos hace pensar que progresivamente las aplicaciones de mensajería, englobarán tal cantidad de funcionalidades que el usuario podrá cubrir la mayoría de sus necesidades haciendo uso de estas. Y es de aquí donde nace la motivación de este proyecto. Se ha analizado cómo añadir funcionalidades en una aplicación de mensajería. Una forma de añadir funcionalidades es mediante bots conversacionales, programas que simulan el comportamiento humano, comprenden nuestras órdenes y actúan en consecuencia. Se ha elegido el servicio de mensajería con mayor soporte a desarrolladores de bots, Telegram.

Este proyecto ofrece un servicio completo mediante la implementación de un bot estable que permita gestionar una cámara de vigilancia interpretando toda la información que esta proporciona.

A lo largo del documento se presenta la aplicación realizada, una descripción de características y requisitos principales, todas las tecnologías utilizadas en el desarrollo de la herramienta, tanto del lado cliente como del lado del servidor, así como un análisis que recorrerá todas las fases del proyecto desde el diseño hasta las pruebas. Por último, se enumeran posibles líneas futuras para dotar al proyecto de una mayor solidez.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1 Motivación

Actualmente, la tecnología para dispositivos móviles ha sufrido un avance tan importante que muchas acciones cotidianas que tradicionalmente se realizaban de una forma, actualmente se realizan en gran medida utilizando teléfonos inteligentes. Desde la comunicación instantánea entre dos personas, hasta la compra de productos en internet. Cubiertas estas funcionalidades, la tecnología actual tiende hacia el llamado internet de las cosas, un concepto que persigue interconectar digitalmente objetos cotidianos con internet, con el objetivo de poder controlarlos remotamente.

En los últimos años, la mayoría de los servicios se han ofrecido en los teléfonos inteligentes a través de aplicaciones específicas, con interfaces propias y accesibles desde tiendas de aplicaciones. En el futuro, muchos de estos servicios acabarán por simplificarse y ser ofrecidos mediante bots, integrados en aplicaciones de mensajería, por ser las más populares entre los usuarios. Los bots están experimentando una gran aceptación en el mercado y Grandes empresas como Facebook ya han liberado APIs para el desarrollo de estas herramientas. Es cuestión de tiempo que aplicaciones masivas como Whatsapp liberen de forma oficial interfaces de programación para implementarlos. En la actualidad, ya existen bots para leer y enviar e-mails, descargar música, películas, o consultar todo tipo de

información de una manera más natural, rápida y eficaz que accediendo vía web o a través de una aplicación dedicada. Además, las actualizaciones son transparentes a los usuarios, que sólo deben preocuparse de interactuar con los bots como si fueran un contacto más.

Atendiendo a las ventajas citadas, parece apropiada la implementación de un bot basado en el concepto de internet de las cosas, en una aplicación de mensajería. En este proyecto se plantea la creación de un bot que permita la gestión asistida de una cámara de seguridad para el hogar.

1.2 Objetivos

El objetivo de este Trabajo de Fin de Grado es desarrollar un bot para la plataforma Telegram usable en cualquiera de sus aplicaciones cliente (iOS, Android, PC, etc) que sea capaz de gestionar una cámara doméstica con una interfaz fácil de entender por el usuario.

Para considerar el trabajo como exitoso la aplicación deberá cumplir un catálogo de requisitos, que especifique los servicios que ha de ofrecer el sistema y las restricciones de su funcionamiento. Se han dividido en funcionales y no funcionales. A continuación, se muestra un listado de ellos:

1.2.1 Requisitos Funcionales

Describen las interacciones entre los usuarios y el software, muestran el comportamiento general del sistema:

- **RF1: Gestión de usuarios**

  La aplicación debe ser capaz de registrar a los nuevos usuarios y mantener un registro de todos ellos, pudiendo eliminarse si un usuario lo desea. Además, los ficheros deben ser accesibles por el desarrollador.

- **RF2: Registro de interacciones**

  La aplicación debe ser capaz de registrar todas las interacciones y procesarlas según cada usuario y su paso en el sistema.

- **RF3: Menú**

  La aplicación debe ser capaz de mostrar un menú predefinido con acceso a todas las funcionalidades implementadas. Deben ser accesibles también desde el teclado estándar, donde un algoritmo deberá interpretar la petición.

  Las peticiones incluyen mostrar listas de usuarios, con información relativa a su estado en casa, pudiendo gestionar este estado también desde la propia aplicación. Acceder a contenido en directo, como un *streaming* o una petición de fotografía. También se debe poder acceder a un histórico de eventos, descargar la información multimedia relativa a estos y consultar el estado del sistema para detectar errores.

- **RF4: Notificaciones**

  La aplicación debe ser capaz de notificar automáticamente al usuario cuando se produzca cualquier evento relacionado con la cámara; eventos relacionados con personas, con movimientos o derivados del funcionamiento de la cámara (cortes de suministro, cortes en la transmisión, errores en la tarjeta SD, etc.). El usuario debe poder interactuar con las notificaciones y acceder a su contenido. El usuario debe poder gestionar las notificaciones, pudiendo elegir qué tipo de avisos sí quiere recibir, cuáles no. Además, deben poder crearse avisos personalizados con información relativa al usuario y a la fecha del evento. El usuario siempre podrá añadir y borrar cuantos quiera.

1.2.2 Requisitos No Funcionales

Requisitos complementarios o atributos de calidad. Se enfocan en las características del funcionamiento del sistema:

- **RNF1: Seguridad**

  Para poder acceder a los datos de los usuarios de la cámara se utilizará un *framework* de acceso seguro (OAuth2 Framework) que deberá refrescar el token cada cierto tiempo.

- **RNF2: Interfaz**

  A pesar de las limitaciones impuestas por Telegram, se debe desarrollar una interfaz sencilla e intuitiva.

- **NRF3: Rendimiento**

  La aplicación debe funcionar alojada en un servicio de computación en la nube, funcionar 24/7 y tener tiempos de respuesta razonables. Debe ser robusto, estable a largo plazo y manejar a los usuarios de forma eficiente. Llevar a cabo un proyecto software supone un reto y para ello es importante seguir una. metodología adecuada. Será necesario planificar las etapas, desde el estudio y análisis, hasta la codificación y las pruebas necesarias. Se seguirá una planificación detallada del tiempo y los recursos, para poner en práctica las capacidades y conocimientos adquiridos a lo largo de todo el grado.

1.3 Material y herramientas utilizadas

En este apartado se enumeran las herramientas hardware y software utilizadas durante el desarrollo del proyecto:

- **Ordenador:** Ordenador personal con conectividad a internet para el desarrollo del bot y sus algoritmos.

- **Cámara:** Una cámara Netatmo inteligente con sistema de almacenamiento propio y conexión a internet. La información obtenida con este equipo se utilizará para el procesado de los datos del bot.

- **Amazon Web Services:**

  Agrupación de servicios web que en conjunto forman una plataforma de computación en la nube. De los numerosos servicios que se ofrecen, en este proyecto se ha utilizado el servicio EC2 (*Elastic Compute Cloud*) en el que hemos creado una instancia sobre un sistema operativo Ubuntu. **CloudWatch** es una herramienta integrada en AWS que permite mostrar gráficos y visualizar el consumo y las tareas de la máquina

- **Ubuntu**: Sistema operativo sobre el que se ha realizado el desarrollo. Es una distribución de Linux gratuita. La mayoría del software construido para este SO es de código abierto, y en este proyecto se hará uso de él.

- **Python**: Es un lenguaje de programación interpretado, multiparadigma y de código abierto. Su filosofía hace hincapié en una sintaxis que favorezca un código legible. La existencia de todas las APIs y módulos necesarios traducidas a este lenguaje lo hacen perfecto para alcanzar los objetivos de este proyecto. Algunos de los módulos más relevantes utilizados han sido *threading, cherrypy, random, time, request, ffmpy, json* y los que se detallan a continuación:

  - **pyTelegramBot API** Implementación gratuita traducida a Python de TelegramBotAPI. Es una API creada por la comunidad GitHub y contiene todas las herramientas necesarias para desarrollar un bot en la plataforma de Telegram.
  - **LNetatmo-API:** Librería gratuita traducida a Python de Dev Netatmo API. Está creada por Philippe Larduinat (Github) y contiene las herramientas básicas para conectar a los servidores de Netatmo y gestionar la cámara Netatmo Welcome.
  - **Telethon:** Librerías para implementar clientes de Telegram. Se ha utilizado para realizar pruebas y test de esfuerzo al bot. Es muy versátil.

- **Atom:** Es un editor de código fuente multiplataforma, también puede ser utilizado como un entorno de desarrollo integrado (IDE). Al ser compatible con el lenguaje Python, se ha empleado como herramienta principal para desarrollar todo el código del programa.

- **pyCharm EDU:** IDE específico para Python. Permite instalar módulos automáticamente y tiene un potente *debugger*. La facilidad de uso de Atom y el tener que hacer las pruebas en un servidor alojado en la nube, hizo relegar esta herramienta a un segundo plano.

- **Wireshark:** Es un analizador de protocolos con interfaz gráfica utilizado para realizar análisis en redes de comunicaciones y para el desarrollo de software. Se ha utilizado en el proyecto para interpretar la información de los *webhooks* y para verificar el cifrado de todas las etapas.

- **No-IP:** Servidor DNS para direcciones dinámicas. Se ha utilizado para proveer un *hostname* a nuestro servidor alojado en la nube en caso de haber cambios dinámicos en la dirección IP.

- **Apache:** Servidor web http de código abierto utilizado para desarrollar un servidor donde consultar información básica del bot. Se ha utilizado junto a:

- **Bootstrap:** *Framework* de código abierto para el diseño de sitios y aplicaciones web haciendo uso de tecnologías como CSS y HTML5.

- **Visual Paradigm:** Se trata de una herramienta que proporciona un conjunto de ayudas para el desarrollo de diagramas en UML para programas en las fases de planificación, análisis y diseño. Se han realizado los diagramas de la aplicación.

1.4 Organización de la memoria

Se pretende dar una visión global de las etapas que se han seguido para la realización de este proyecto, su función es dotar de una estructura coherente a la memoria:

- Capítulo 1 - Introducción: Capítulo actual que recoge un pequeño avance de lo que va a ser el proyecto, una descripción básica, cuáles son los objetivos que se persiguen y qué herramientas se han utilizado.

- Capítulo 2 - Estado del Arte: Estudio de los recursos disponibles y el estado actual de esta tecnología, un análisis de su viabilidad como producto y algunos ejemplos de aplicaciones similares.

- Capítulo 3 - Arquitectura y desarrollo del sistema: Se procede a diseñar la herramienta, con los diagramas oportunos y la implementación.

- Capítulo 4 - Pruebas: Una vez finalizadas todas las fases anteriores, se hace necesario la realización de una fase de pruebas generales, se ha desarrollado un cliente para poder someter a esfuerzos al programa.

- Capítulo 5 - Conclusiones y Líneas futuras: Se propondrán futuras actualizaciones y mejoras en el servicio como nuevas funcionalidades. Además, se incluye un informe personal sobre las conclusiones de este proyecto.

Capítulo 2

Estado del Arte

En la actualidad, el 81 por ciento de españoles utiliza un smartphone, [*Google Consumer Barometer Report*] [10] una cifra muy considerable, que refleja más aún su tendencia alcista si lo comparamos con este mismo análisis hecho en 2012, donde tan solo el 41 % de la población hacía uso de un smartphone a diario. Si atendemos a estudios realizados sobre el uso de aplicaciones de mensajería instantánea, se observa cómo el número de usuarios que las utilizan aumenta conforme pasan los años, y se prevén más de dos mil millones de usuarios activos en 2019 [1.1] y más del doble que en 2014, esto nos hace pensar la fuerte progresión de este segmento.

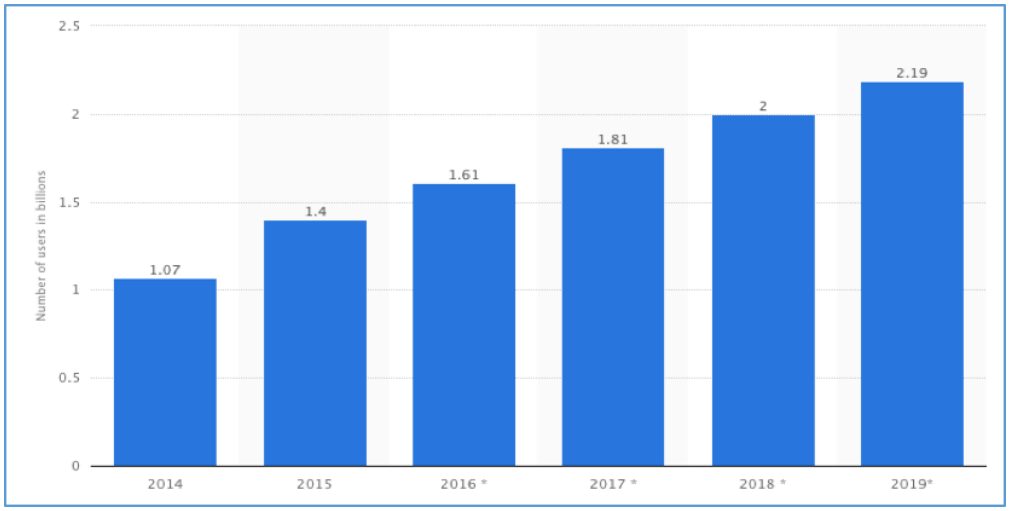


Figura 2.1: Crecimiento de usuarios en aplicaciones de mensajería [comscore© statista.com]

Por otro lado, la figura [2.2] muestra como por primera vez el uso de aplicaciones

de mensajería instantánea ha superado ya al de las redes sociales. También se ha analizado la distribución de tiempo utilizado por el usuario en las aplicaciones según su tipología, el 25 % [2.3] de tiempo, se destina a aplicaciones sociales o de mensajería, el valor más alto de cualquier categoría.

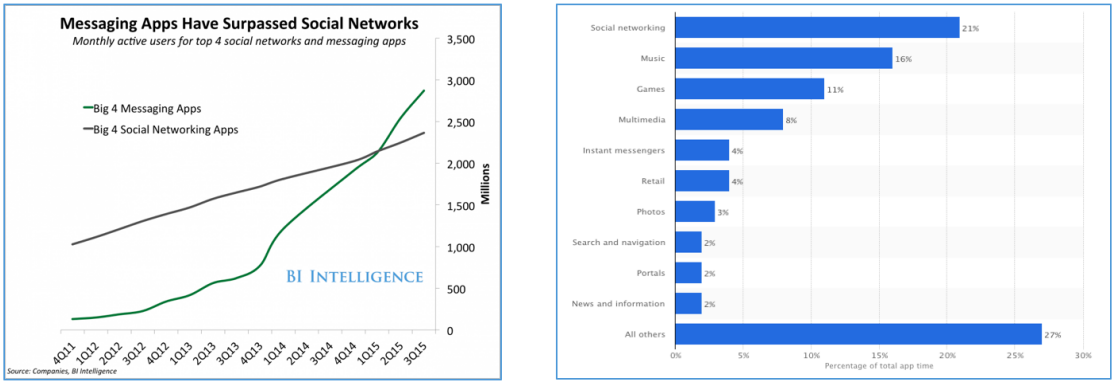


Figura 2.2: 2.3 :Usuarios mensuales y uso de aplicaciones móviles [comscore© statista.com]

Las últimas tendencias en el campo de la oferta de servicios a través de internet están migrando del modelo de utilizar una aplicación específica para proporcionar un servicio, a proporcionar el servicio a través de aplicaciones de mensajería por medio de los llamados bots.

Los bots son software autónomo que puede interactuar con los usuarios a través de aplicaciones de mensajería, ofreciéndoles casi cualquier tipo de servicio. Los bots se integran en las aplicaciones como un contacto más, y la interacción con ellos se hace de la misma forma que se haría con un usuario real. Esto hace que sean una muy buena alternativa frente a la aplicación móvil tradicional, porque la mensajería instantánea ya está presente en nuestras vidas y su uso está muy generalizado.

A su vez, son independientes de las aplicaciones de mensajería que las utilizan, es decir, 'viven' fuera de ellas, pueden estar alojadas desde en entornos locales a entornos de computación en la nube. Esto aporta otra ventaja, al no estar alojados en el equipo del usuario, el desarrollador puede introducir actualizaciones sin su expreso consentimiento. Si lo bots trabajan con datos potencialmente personales (datos médicos, información del hogar), tienen que implementar también políticas

de seguridad y privacidad para lograr un ecosistema completamente seguro.

Los bots a su vez se clasifican de acuerdo a la forma con la que interactúan con los usuarios finales; pueden estar divididos en tres categorías [2]:

1. Chat-bots: La interacción se realiza mediante el intercambio de mensajes impulsados técnicas de procesado del lenguaje natural (PLN).

2. App-bots: Son mínimamente conversacionales y utilizan botones durante el flujo de la conversación para interactuar con los usuarios finales.

3. Mix-bots: Son una combinación de los dos anteriores.

En el siguiente gráfico se muestra una comparativa de las plataformas de mensajería más conocidas, el número de usuarios, y su situación actual respecto al cifrado y con el desarrollo de bots [2.3].

Plataforma	Bot API	Usuarios	Cifrado	Orientación Profesional
Whatsapp	NO	1200M (2017)	e2e	NO
Facebook Messenger	SI	1200M (2017)	e2s	NO
Telegram	SI	100M (2016)	e2s	NO
Slack	SI	4M (2016)	e2s	SÍ
Microsoft Teams	SI	85M (2016)	e2s	SÍ
Viber	SI	920M (2017)	e2s	NO
LINE	SI	169M (2017)	e2s	NO

Figura 2.3: Plataformas de mensajería y su características

En términos generales, podemos distinguir dos tipos de plataformas de mensajería: orientadas al entorno profesional y orientadas al personal. Las plataformas de mensajería orientadas al trabajo serían adecuadas para crear comunidades de profesionales para gestión de datos sensibles, como datos médicos de pacientes, bots de servicios bancarios, o de compra-venta. Deben contar además con regulaciones oficiales en términos de seguridad. Las plataformas de mensajería personales deberán servir para bots de información, por ejemplo, de información meteorológica, prensa o servicios multimedia. Sus restricciones en materia de

seguridad podrán ser más permisivas.

Existen bots para infinidad de escenarios [2.4]. Gestionar nuestros datos médicos, realizar *trading* en el mercado de valores, consultar la previsión meteorológica, aprender inglés, reservar vuelos y hoteles, utilizarlos como cliente de correo electrónico o incluso gestionar los dispositivos conectados de nuestro hogar (IoT), que es lo que este proyecto pretende, poder gestionar remotamente cámaras de vigilancia para tener el control sobre nuestro hogar. La siguiente ilustración muestra tres posibles plataformas de desarrollo de bots:

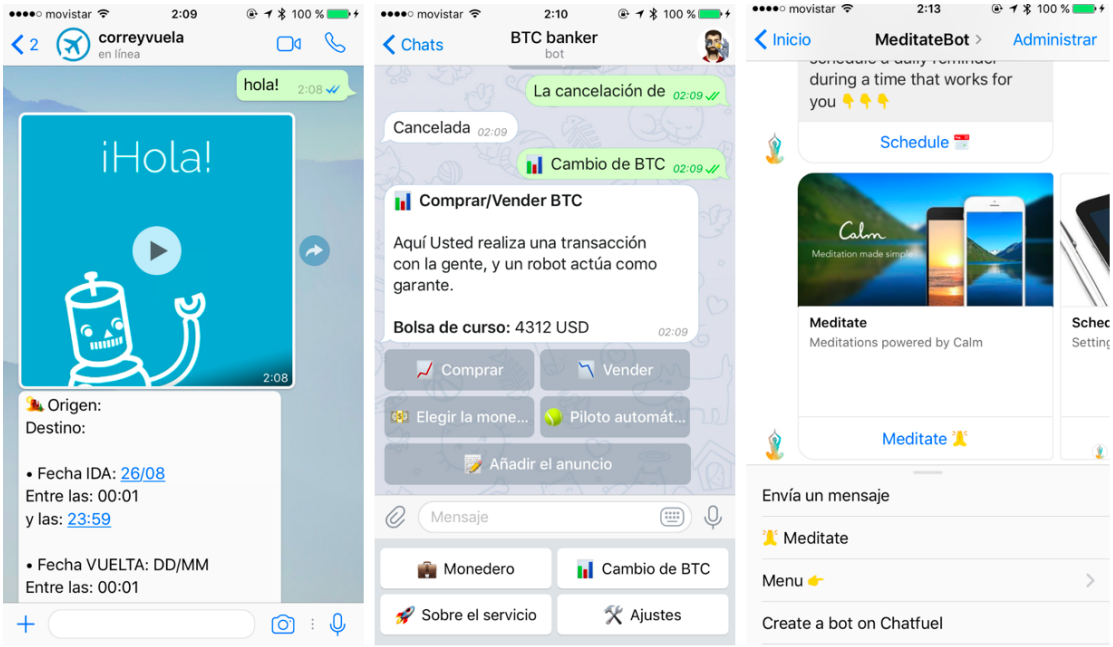


Figura 2.4: Ejemplos de bots actuales en Whatsapp, Telegram y Facebook

Telegram es una de ellas y nace como un proyecto de mensajería instantánea segura sin ánimo de lucro y actualmente ronda los doscientos millones de usuarios registrados. Permite incluso desarrollar aplicaciones cliente y tiene una gran comunidad de desarrolladores. En junio de 2015 es una de las primeras en liberar una API completa para el desarrollo de bots conversacionales y se traduce a numerosos lenguajes de programación. pyTelegramBotAPI es uno de ellos. [1]

Capítulo 3

Arquitectura y desarrollo del sistema

En el presente capítulo se describe la arquitectura y el desarrollo del sistema. Se ha diseñado una estructura de almacenamiento de datos y procesado, basado en computación en la nube.

3.1 Arquitectura del sistema

La arquitectura de la aplicación va a definir la manera en que es diseñado lógicamente nuestro proyecto. La plataforma de desarrollo de bots escogida ha sido Telegram, los motivos que han llevado a ello son su amplia comunidad de desarrollo y la facilidad de uso que ofrece la API. El lenguaje seleccionado ha sido Python, cuenta con el soporte necesario para el uso de todas las tecnologías utilizadas. Hay cinco bloques que componen nuestro sistema; la aplicación cliente Telegram, el servidor de Telegram, el bot alojado en la nube, el servidor de Netatmo y las cámaras que tengamos instaladas. Los módulos desarrollados que luego se explicarán en más detalle son:

- **Cliente:** Elemento encargado de enviar las peticiones al servidor de Telegram. Se ha desarrollado también un cliente propio para verificar la robustez frente al tráfico. Está detallado en el Capítulo 4.

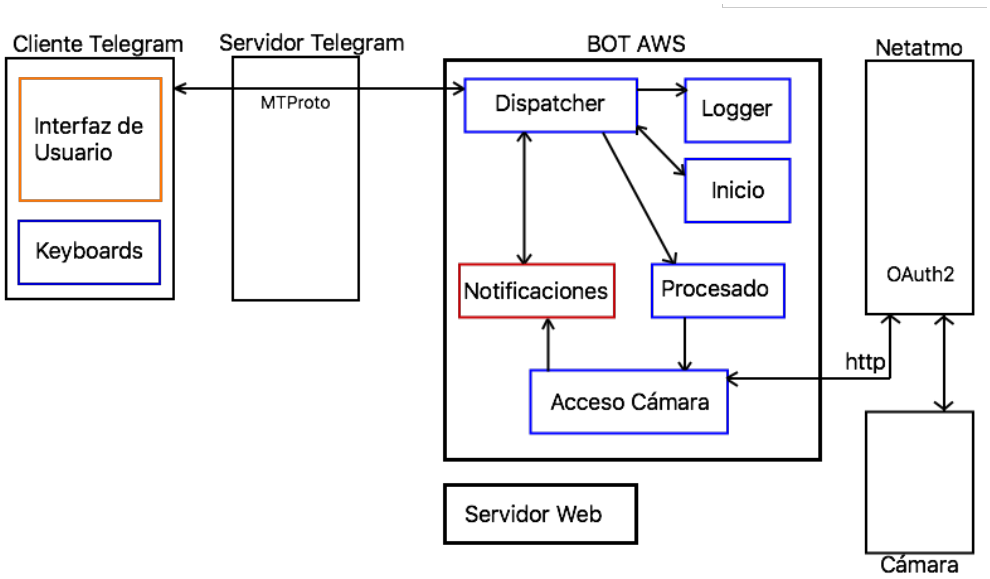


Figura 3.1: Arquitectura del sistema

- **Servidor de Telegram:** Se encarga de recibir las peticiones de la aplicación cliente, funciona con un protocolo propio de Telegram y es el puente entre nuestro programa y el usuario.

- **Dispatcher:** Elemento que permite manejar las peticiones generadas por el cliente. Las actualizaciones llegan a través de *webhooks* y se procesan en los bloques siguientes.

- **Inicio:** Bloque encargado de registrar a los nuevos usuarios comprobando sus credenciales de Netatmo. Para acceder al sistema deben ser validadas.

- **Logger:** Elemento que se encarga de mantener un registro de datos con los usuarios activos en el sistema y su configuración. Registra todos los mensajes de los usuarios y mantiene un registro con información multimedia de todos los eventos

- **Procesado:** Elemento que se encarga de interpretar y evaluar las peticiones del cliente al bot. Una vez interpretadas, se encarga de realizar las operaciones requeridas.

- **Acceso cámara:** Se encarga de traducir las peticiones procesadas en peticiones al servidor de Netatmo. Este bloque incluye el proceso de autentificación, con el objetivo de proveer seguridad al sistema.
- **Notificaciones:** Elemento que se encarga de notificar al usuario mediante mensajes la ocurrencia de eventos en la cámara. Es un proceso que funciona de forma asíncrona y paralela al proceso principal. Permite añadir y eliminar notificaciones personalizas y configurar las predefinidas.
- **Netatmo:** Servidor a través del cual la cámara y el bot se comunican. Provee la estructura de los datos.
- **Cámara:** Almacena en una tarjeta de memoria toda la información multimedia en un formato encriptado.
- **Servidor Web:** Elemento que provee una página web con la información sobre el bot y su funcionamiento.

3.1.1 Cliente

La aplicación cliente utilizada en el proyecto es la versión oficial para iOS y PC, pero cualquier otra oficial hubiera sido válida. Esta aplicación recoge la información procedente de la interacción con el usuario, que se envía al servidor de Telegram, el cual la hará llegar hasta nuestro bot una vez se encuentre correctamente registrado.

Sobre este cliente se ha personalizado el modo de enviar las peticiones mediante varios tipos de teclado que permiten orientar al usuario para facilitar la interacción. En el proyecto se han utilizado las dos variantes de teclado que facilita la API de la Telegram, además del estándar:

1. Reply Keyboard: Es un teclado que sustituye al alfanumérico por defecto. Las opciones que muestra se consideran respuestas escritas por teclado. Se ha utilizado para guiar las funcionalidades del menú principal. [3.4]

2. Inline Keyboard: Es un teclado fruto de la interacción del usuario con un mensaje. Va directamente adherido al texto del mensaje y se activa mediante

parámetros en los mensajes de respuesta Se ha utilizado para proveer información relativa a:

- Botones de acceso para *live streaming*.
- Teclado dinámico para indicar que alguien ya no está en casa.
- Teclado dinámico para mostrar el listado de personas conocidas.
- Mostrar opciones relativas a la descarga o visualización de un vídeo.
- Mostrar calendario para selección de históricos de eventos.
- Teclado dinámico para la creación y gestión de los avisos.

Figura 3.2: Formateos posibles del teclado en la aplicación

3. Teclado estándar: Además de poder interactuar a través de teclados predefinidos mediante botones, las funcionalidades se pueden servir a través del teclado estándar, gracias a la detección de patrones. Consideramos importante dotar al programa de cierta inteligencia a la hora de procesar las peticiones que se reciben. Se ha implementado un intérprete de mensajes escritos por teclado basado en la detección de palabras clave.

Se han creado bloques de diccionarios que representan cada funcionalidad. Existen diccionarios construidos con palabras que signifiquen la petición de fotografías, mostrar listas de usuarios o incluso saludar a la aplicación.

- A modo de ejemplo se muestra una petición concreta:

El usuario escribe por teclado una petición, por ejemplo: 'Arturo se ha marchado'. El programa en este momento analiza mediante bloques condicionales si la frase es literalmente una funcionalidad del teclado de botones. En este caso no sucede, por lo que comienza a analizar toda la cadena a través del módulo *find* implementado en la función *isKeyWord*. Para este ejemplo encontraría dos palabras clave. La primera de ellas es 'Arturo', que está contenida en el diccionario de personas conocidas. Además, ?marchado? está incluida en el diccionario que indica al programa que un usuario ha abandonado el hogar. A través de bloques condicionales se evalúan las dos condiciones y se envía la petición correspondiente para actualizar el estado de Arturo.

Figura 3.3: Ejemplos de peticiones por teclado y sus respuestas iOS

En [3.3] se muestran algunos ejemplos. Además de este cliente oficial se ha desarrollado una aplicación cliente propia para realizar pruebas. En el Capítulo 4

se enuncian sus resultados.

3.1.2 Dispatcher

El dispatcher es el elemento encargado de recibir las peticiones, desempaquetarlas y entregárselas al subsistema correcto. También se encarga de empaquetar y enviar los mensajes de respuesta al cliente. Recibe como información los mensajes que el cliente genera, que compulsa con las credenciales y se envían al subsistema correspondiente. Por otro lado, gestiona la información ya procesada de la cámara, para reenviarla al cliente a través de la API de Telegram incluyendo los parámetros necesarios (mensaje, archivo, botones, etc.).

La implementación de este bloque se podía hacer mediante el método *polling* o mediante *webhooks*. Son dos formas de gestionar las peticiones que llegan al sistema; por un lado, en *polling*, nuestra aplicación estaría continuamente preguntando a Telegram si existe algún mensaje nuevo que pudiera procesar, Con *webhooks*, sin embargo, notificamos a Telegram que tenemos un servidor web escuchando, y que cada nuevo mensaje a procesar nos lo envíe directamente. En términos de eficiencia es mejor, ya que ahorra ancho de banda y tiempo de procesado.
Para recibir las peticiones en nuestra máquina hemos configurado un *webhook* con la dirección de nuestra máquina virtual, y para su implementación configuramos en un principio el módulo BaseHTTPServer, donde aparecieron errores cuando se hacían peticiones simultáneas desde distintos terminales. Se ha configurado con cherryPY, un web *framework* escrito en Python, que permite crear aplicaciones web rápido y con menos código.
Se ha dado de alta un servidor web escuchando por el puerto 8443 los *webhooks* que Telegram genera en http request. Para gestionar las llamadas se hace una comprobación de la longitud y el tipo del contenido recibido; si cumplen con las especificaciones (tipo JSON), se procede a decodificar el mensaje Unicode mandando procesar la respuesta al subsistema de procesado.

```
class WebhookServer(object):
    @cherrypy.expose
    def index(self):
        if 'content-length' in cherrypy.request.headers and \
           'content-type' in cherrypy.request.headers and \
           cherrypy.request.headers['content-type'] == 'application/json':
            length = int(cherrypy.request.headers['content-length'])
            json_string = cherrypy.request.body.read(length).decode("utf-8")
            update = telebot.types.Update.de_json(json_string)
            bot.process_new_updates([update])
            return ''
        else:
            raise cherrypy.HTTPError(403)
```

Figura 3.4: Servicio del dispatcher

Una vez configurado el comportamiento, hace falta añadir seguridad SSL para autenticar las llamadas de Telegram. Se utilizarán una clave privada y un certificado generados manualmente con *openssl* en el que identificaremos nuestro equipo como el receptor final de las actualizaciones que genera la API de Telegram. Así aseguramos que nadie pueda cambiar el destinatario de la comunicación.
Para poner en funcionamiento el servidor, pasamos los parámetros de host y puerto, el módulo SSL a utilizar y la clave junto con el certificado que hemos emitido anteriormente.

3.1.3 Inicio

Este subsistema se encarga de catalogar y evaluar las credenciales de todos los usuarios que utilizan el bot. Recibe como información las actualizaciones de los registros que hace el dispatcher, y envía esta información directamente al bloque logger, que graba las modificaciones.
Cuando un usuario comienza a utilizar la aplicación debe realizar la fase de registro, en la cual el sistema pide sus credenciales: usuario, contraseña, identificador de cliente y secreto de cliente. Si los datos introducidos son correctos este bloque añadirá el usuario al sistema que permite gestionar la cámara asociada. Si en algún caso el usuario quisiera eliminar sus credenciales y dejar de utilizar el bot, existe un comando (/reset) para borrar las credenciales y ser borrado del sistema.

3.1.4 Logger

Este bloque permite registrar la información de los usuarios que utilizan el sistema. Recibe la información de nuevos eventos por el dispatcher y por el módulo de inicio la gestión de nuevos usuarios. Esta información es almacenada y enviada a los subsistemas que acceden a la información (inicio para verificar usuarios, procesado y notificaciones para aplicar las preferencias) Se han implementado cuatro registros:

1. Mediante la función update listener gestionamos cualquier mensaje de un usuario que envía desde la aplicación cliente. Este subsistema se encarga de copiar estas interacciones en un archivo de texto. Todas las interacciones que un usuario tiene con el sistema quedan registradas con una doble finalidad. Se pueden detectar más fácilmente los errores sabiendo qué comandos y mensajes se han introducido y, además, al quedar registradas en un archivo de texto se puede consultar el historial completo de actividad y conocer el uso que los usuarios le dan a la aplicación. Todos los mensajes quedan registrados con los parámetros de id, nombre, mensaje y fecha. Se añade un modelo de información en los anexos.

2. Para administrar a los usuarios que tienen una cámara registrada se ha creado un objeto JSON que asocia las credenciales de los usuarios autenticados con su identificador en Telegram. También incluye la configuración individual de las notificaciones y un listado con los avisos personalizados de cada usuario. También es posible añadir o banear usuarios modificando manualmente los valores de este registro. Se añade un modelo de información completo en los anexos.

3. Para gestionar las notificaciones se ha creado otro objeto JSON que asocia cada cámara Netatmo con los usuarios que tienen acceso a ésta. Aquí un ejemplo real donde una cámara está gestionada por tres usuarios:

u'70:ee:50:1d:54:8f': [u'333408406', u'10663093', u'378251683']

4. El histórico de eventos se almacena en otro JSON que se actualiza cada vez que ocurre un nuevo evento. Incluye toda la información referida al evento y en aquellos que tengan asociados información multimedia como una fotografía, se descarga en local y se etiqueta para que pueda ser accesible siempre. Se detalla en los anexos el modelo de información utilizado.

3.1.5 Procesado

Es el subsistema que describe la lógica del bot. Su función es interpretar todos los mensajes y procesarlos correctamente. Recibe como información las peticiones del cliente traducidas por el dispatcher y la información que provee Netatmo que debe ser procesada. La información que genera se envía a los mismos subsistemas, ya que hace de puente entre las peticiones y la cámara.
La gestión de las interacciones que llegan al bot se van a hacer mediante manejadores de eventos, que a su vez son funciones decoradas, es decir, permiten ejecutar otros métodos dentro de estas. Todas las interacciones del usuario se sirven según el resultado de estas funciones. Para nuestro proyecto se han utilizado funciones lambda, que establecen las condiciones necesarias para acceder a un método en concreto.
Se han implementado diecinueve manejadores de eventos. Cuatro de ellos son para la gestión de los comandos. Uno se encarga de mostrar y gestionar el menú principal para usuarios logeados, cuatro gestionan los pasos o niveles de cada función mostrada en el menú principal, cuatro gestionan todo el proceso de gestión de nuevos avisos y seis la gestión de peticiones de datos históricos. A continuación, se detallan de forma global todos ellos:

1: Procesado de comandos

Este bloque procesa los comandos de la aplicación, se han implementado cuatro. El comando /start es el encargado de procesar la entrada de nuevos usuarios al sistema. Se comprueba si tienen alguna cámara registrada a su nombre para acceder directamente a la aplicación. En el caso de no encontrarse en el registro, ofrece un

menú para iniciar el registro. El comando /reset permite borrar todos los registros de un usuario, tanto su información de login como sus cámaras asociadas (ver Anexo F) El comando /help y /about no procesan datos, son sencillamente un menú con información relativa al bot.

2. Main

Este bloque es el encargado de procesar los mensajes del usuario una vez registrado. Realiza dos comprobaciones iniciales. Por un parte comprueba si la cámara está operativa, y si es el caso, analiza el mensaje introducido.
Este bloque procesa peticiones para conocer el estado del sistema, elabora las listas de usuarios conocidos, ofreciendo al usuario cuantas y cómo desea recibirlas. También procesa las peticiones para el acceso directo a la cámara, interpreta mensajes para conocer el estado de las personas en el hogar.
Asimismo, además de analizar palabras clave relacionadas con las funcionalidades del sistema, el proceso es capaz de interpretar mensajes agradables, malsonantes o de despedida. Para ellos, se han creado varias respuestas predefinidas que mediante módulos de generación de números aleatorios seleccionan una de ellas y se reenvía al usuario.
Además de todo lo descrito inicia los procesos de gestión de los avisos y la muestra de eventos pasados.

3. Datos de personas

Bloque encargado de gestionar un mensaje que indique que un usuario ha dejado el hogar o requiera información adicional sobre este. Se activa mediante un *Inline Keyboard* y pasa a procesar la petición con los datos del registro de personas, una vez recogidos, envía directamente la petición a Netatmo.

4. Procesado de eventos

Recibe información del bloque Main y del registro de eventos. Se encarga de formalizar la petición a Netatmo y preparar el mensaje de reenvío al usuario con los eventos requeridos. En la siguiente ilustración se ve el procesado de una petición

de datos históricos entre dos fechas.

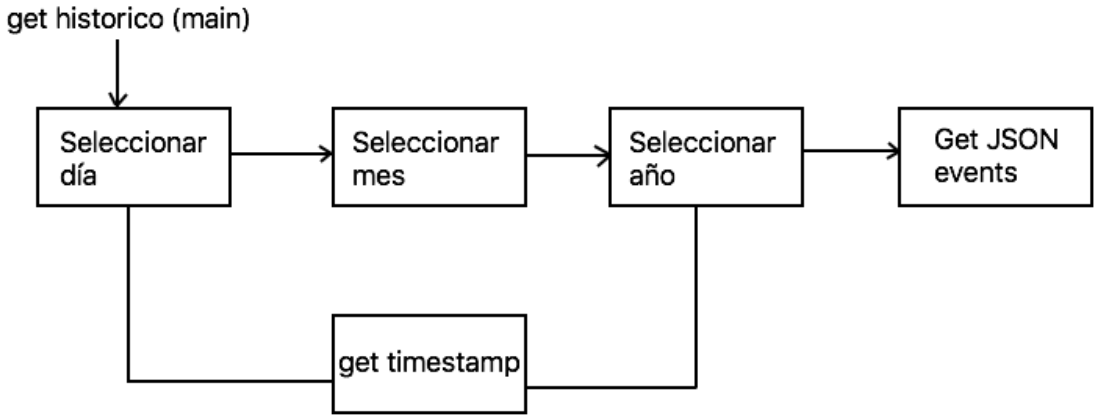


Figura 3.5: Procesado de una petición de datos históricos

5: Procesado y creación de avisos

Para la creación de avisos se hace uso del registro *login*. En este registro se encuentra la información necesaria que necesita este subsistema. Está compuesto por cuatro manejadores. Una vez introducidos los campos necesarios para configurar el aviso, se modifica el registro y se notifica al proceso que gestiona las notificaciones que evalúe este aviso en las próximas iteraciones.
Para administrar peticiones con varios pasos en el proceso como este (persona a monitorizar, hora de inicio y hora de fin), se ha implementado un registro del paso en el que se encuentran, para poder avanzar o retroceder entre los manejadores.

6. Procesado petición de video

Este manejador se encarga de gestionar la conversión de archivos de video y enviarlos al usuario que lo ha demandado. Está petición se inicia desde cualquier botón que lleve asociado un parámetro de URL. Al pulsarse se produce un evento *callback* y el manejador que corresponde evalúa la función.

Mediante el módulo ffmpy, una traducción literal de ffmpeg [7], realiza la conversión del archivo a un formato legible por Telegram. El formato de vídeo que mayor compatibilidad ofrece tanto a clientes en smartphones como en PC ha

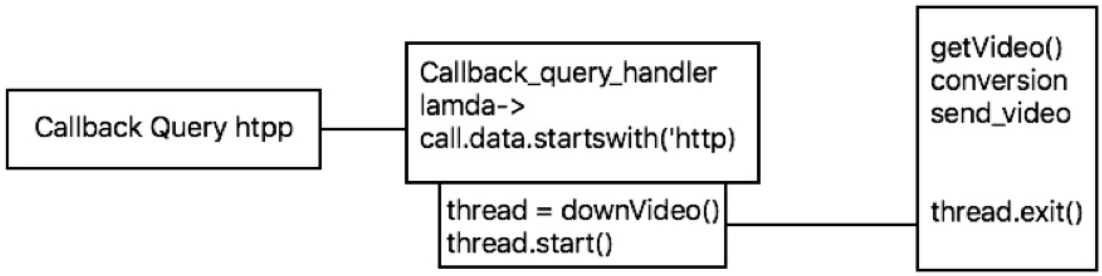


Figura 3.6: Servicio de un callback handler que gestiona peticiones de video

sido *.mp4.

El proceso de conversión de los archivos de vídeo es delicado, hubo que encontrar el módulo adecuado para convertir el sonido codificado en MPEG-2/4 AAC ADTS (contenedores de MPEG-TS) a MPEG-4, por suerte ffmpeg lo incluye como parámetro en su última versión. Otro problema a tratar con el procesado del vídeo ha sido el cómo Netatmo gestiona sus notificaciones. Envía la notificación, pero mantiene la grabación de vídeo por si pasa algo más; eso quiere decir que en la propia notificación no se puede enviar un *link* al vídeo del evento porque no está preparado. Se implementó esperar un tiempo más largo para poder enviar la notificación con el vídeo incluida, pero en algunos casos, dependiendo del evento, ya no parecía una notificación en tiempo real, y se descartó. Se ha considerado por la importante carga de CPU que conlleva este proceso de conversión la utilización de *threads* para distribuir la carga y disminuir el tiempo de conversión. Por lo tanto, es el manejador el que al ser evaluado llama a la creación de un nuevo hilo para servir la petición.

3.1.6 Netatmo y Autenticación

Este bloque se encarga de la gestión y configuración de la cámara para acceder a sus datos e integrarlos en el programa. Recibe como información de entrada las peticiones ya procesadas y su información de salida es directamente la petición con los parámetros oportunos que envía al servidor de Netatmo.

En primer lugar, se ha asociado la cámara en una cuenta de desarrollador en el programa de desarrolladores de Netatmo. Una vez inscritos en el programa se

rellena el formulario para la creación de una aplicación. Se ha solicitado a Netatmo los scopes de acceso y escritura para poder modificar las bases de datos. Una vez realizado este proceso Netatmo devuelve varios parámetros, que junto con el usuario y contraseña son suficientes para gestionar nuestros dispositivos.
Para acceder a la información de la cámara, es necesario incluir un token de acceso en cada petición. El proceso para obtenerlo comienza por enviar un HTTP Post a *https://api.netatmo.com/api/oauth2/token* con nuestras credenciales cómo parámetros. Como resultados se obtienen el token de acceso para las demás peticiones, el tiempo que tiene validez el token y un nuevo token de refresco para pedir uno nuevo una vez expire.

Este token se puede utilizar para llamar a las siguientes funciones de la API, a las cuales se accede enviando otro HTTP Post con los parámetros especificados:

1. AUTH_REQ: Recibe el token de acceso.
2. GETHOMEDATA_REQ: Recibe información actual de un hogar incluyendo parámetros como el estado y localización de la cámara, la información sobre los usuarios y los últimos eventos que se han producido.
3. GETCAMERAPIC_REQ: Descarga de archivos multimedia referidos a eventos de movimiento o personas.
4. GETEVENTSUNTIL_REQ: Actualiza el listado de eventos. Utiliza una cola FIFO para eliminar los más antiguos.
5. SETPERSONAWAY_REQ: Edición manual del estado de una persona en el hogar.
6. PING_REQ: Comprueba el estado de conexión de la cámara y provee una dirección url de la red local si fuera posible. No es el caso por estar alojado en AWS.
7. VOD-LIVE_REQ: Petición que devuelve información en vivo.
8. ADDWEBHOOK - DROPWEBHOOK: Enlazar u olvidar un *webhooks*.

Para implementar los Post Request se han utilizado los módulos urllib y urllib2. Estos nos permiten codificar los parámetros de la petición y poder enviarla a Netatmo. En la respuesta se decodifican en formato Unicode si son peticiones de datos o se extraen los bytes en claro si se trata de imágenes o contenido multimedia.

Asimismo, toda esta información es estructurada para facilitar la implementación. Permite organizar los datos para que puedan ser accesibles y utilizados de manera eficiente. La estructura de de datos de la información recibida se muestra a continuación [3.7].

```
+--ro home_id
|  +--ro id*                  string -> 596d039cac34a532...
|  +--ro name          string -> eina1
|  +--ro cameras
|  |  +--ro id*               MAC  -> 70:ee:50:1d:54:8f
|  |  +--ro is_local          boolean-> True
|  |  +--ro name              string -> cam1
|  |  +--ro sd_status         string -> on
|  |  +--ro use_pin_code      boolean-> false
|  |  +--ro date_setup
|  |  |  +--ro usec           int    -> 956000
|  |  |  +--ro sec            int    -> 1495122050
|  |  +--ro vpn_url           string -> https://v1.netatmo.net...
|  |  +--ro alim_status       boolean -> True
|  |  +--ro type              string -> NACamera
|  |  +--ro status            string -> on
|  |
|  +--ro persons
|  |  +--rw out_of_sight      boolean -> False
|  |  +--ro face
|  |  |  +--ro version        int-> 1
|  |  |  +--ro id*            string -> 59821343ea00a02b..
|  |  |  +--ro key            string -> f8882866d1a9ae7f3...
|  |  +--ro id*               string -> 0f00abf1-6a2b-4eb-...
|  |  +--ro last_seen         int    -> 1500482169
```

```
|  |  +--ro pseudo            string -> Arturo
|  |
|  +--ro place
|  |  +--ro timezone string -> Europe/Madrid
|  |  +--ro city              string -> Zaragoza
|  |  +--ro country           string -> ES
|  |
|  +--ro events
|  |  +--ro category  string -> human, animal
|  |  +--ro video_status      string -> available
|  |  +--ro video_id string -> 9b9b88f7-87e1-...
|  |  +--ro vignette
|  |  |  +--ro version        int     -> 1
|  |  |  +--ro id             string -> 598c2b6c2b2b46...
|  |  |  +--ro key            string -> 07d27b72ebf119e...
|  |  +--ro snapshot
|  |  |  +--ro version        int     -> 1
|  |  |  +--ro id             string -> 598c2b6c2b2b46...
|  |  |  +--ro key            string -> 07d27b30bf119...
|  |  +--ro camera_id         string -> 70:ee:50:1d:54:8f
|  |  +--ro time              int     -> 1502355815
|  |  +--ro message           ASCII -> <b>Arturo</b> visto
|  |  +--ro type              string -> person, sd, home_away
|  |  +--ro sub_typeint       -> 1
|  |  +--ro id*               string -> 598e3190b26ddfe4...
```

Figura 3.7: Descripción del diseño de las respuestas DEV-Netatmo en formato JSON

En el modelo de información mostrado se describen los campos accesibles a través de la API. Se muestra en la tercera columna una posible instancia a modo de ejemplo. En el lenguaje Python va a estar implementado en un tipo de dato diccionario, no hay límite en el número de instancias de cada sección y se accede a los campos a través de sus índices, marcados en el diagrama con un asterisco.

Este modelo de información permite mantener organizado el sistema. Cada hogar está etiquetado con un identificador. A partir de aquí se encuentran las

cámaras disponibles, en el caso que un usuario tuviera dos cámaras instaladas podría recibir información de ambas instalaciones. Otro bloque de información está formado por las personas almacenadas en el sistema, con parámetros como el nombre, un identificador de su fotografía de perfil o la última vez que fue visto. También se encuentra un bloque con la información de ubicación de la instalación, para determinar el huso horario y el posicionamiento del hogar. Por último, el bloque de eventos contiene información de los eventos más recientes con la información personalizada de cada uno.

3.1.7 Notificaciones

La gestión de las notificaciones es una de las partes más importantes que debe soportar la aplicación. Este módulo debe notificar automáticamente al usuario de cualquier evento que le suceda a la cámara o al entorno que graba. Recibe como información de entrada los nuevos eventos ocurridos y la configuración de los avisos de cada usuario a través del dispatcher. La información de salida del bloque es directamente el dispatcher que reenvía la información a los usuarios que lo requieran La cámara notifica cada evento a los servidores de Netatmo, que se almacenan en un registro JSON, identificados por su timestamp correspondiente y su tipología. La aplicación distingue los siguientes tipos de evento:

- **Persona:** Cara detectada, bien puede ser conocida o desconocida.
- **Person_away:** Se activa por geolocalización, asocia personas con smartphones. en casa. También funciona por tiempo expirado.
- **Movimiento:** Se activa al detectar cualquier tipo de movimiento.
- **Conexión:** Se produce cuando la conexión es establecida con los servidores de Netatmo.
- **Desconexión:** Se produce al perder la conexión con los servidores de Netatmo.
- **Monitorización On:** Se produce al activar la señal de monitorización.
- **Monitorización Off:** Se produce al desactivar la señal de monitorización.

- **SD:** Activado si capacidad llena o retirada de la tarjeta de la bahía.
- **Alimentación:** Se activa al perder el contacto con la cámara (conexión perdida).
- **Aviso Personalizado:** Se produce cuando se cumplen los parámetros de los avisos personalizados.

Una vez definidos los objetivos, es necesario que las notificaciones lleguen mientras el bot siga en funcionamiento. El método más fiable que se ha encontrado es la utilización del módulo *threading* (hilos de ejecución). Este módulo nos va a permitir lanzar un nuevo hilo de ejecución paralelo para gestionar las notificaciones mientras el resto del programa no se ve alterado. También se ha activado el *daemon* del *thread* para que no se quedase trabajando intermitentemente si sucede algún fallo en el programa principal.
La implementación de esta función nace de la necesidad de mantener un registro completo de todos los eventos, Netatmo únicamente devuelve los 30 últimos eventos más recientes, y con esta función enlazada al logger de eventos, se consigue mantener un registro ilimitado de todos los eventos de la instalación.
Se ha creado la función *checkNewEvent* que se encarga de hacer una comparación entre los identificadores del último evento actual y el último evento recogido hace 50 segundos. Si resulta que son distintos indica que se ha producido un nuevo evento en ese periodo de tiempo, mandando una señal de activación al método que corre en el *thread*, y este mediante bloques condicionales decide qué tipo de evento ha sucedido y lo reenvía al usuario. Todo este proceso es personalizado para cada usuario con su correspondiente cámara.

Para saber el tipo del evento, existe un campo en el JSON que lo describe. Si es de tipo persona y su identificador no se encuentra en el registro de personas conocidos se notificará un desconocido detectado. Si por el contrario el id de la persona detectada contiene el nombre del usuario, es conocido y se gestiona como tal. Todas las notificaciones quedan registradas con la hora y la descripción del evento, si ha sido un evento con información multimedia de tipo persona o

movimiento, descarga una captura y despliega un menú si tiene más opciones disponibles. Para eventos que no provienen de la grabación de imágenes como cortes de señal, desactivación de la vigilancia o desconexiones, se notifica mediante un mensaje sin archivos multimedia adjuntos.

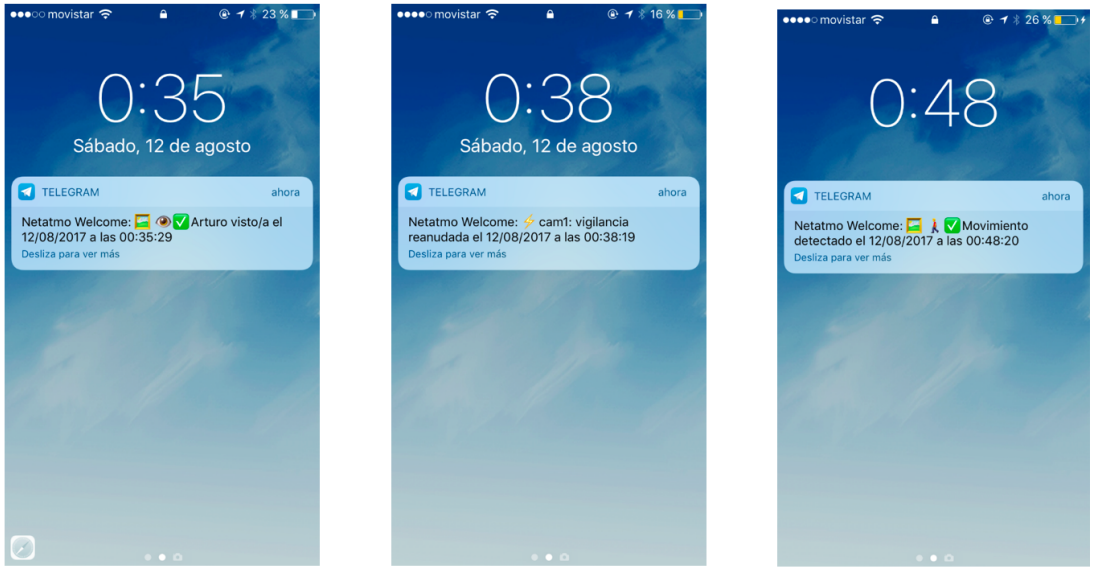


Figura 3.8: Varias notificaciones en aplicación iOS

Existe además la opción de programar avisos personalizados temporales. A través de esta funcionalidad el usuario accede a un menú teclado *inline*:

- **Conocidos:** Permite configurar las notificaciones de los conocidos, desactivarlas, activarlas y crear nuevos avisos personalizados. Cada aviso personalizado necesita los parámetros de nombre, hora de inicio y hora final que se introducen vía teclado inline. Una vez configurado el aviso, la aplicación monitoriza todos los nuevos eventos relacionados con esa persona, cuando expira el tiempo de la notificación verifica si ha sido visto en las horas configuradas y notifica al usuario con el resultado del análisis. La aplicación permite añadir un número de avisos personalizados ilimitados y se dispone de la opción de borrado.

- **Desconocidos:** Bloque que permite configurar los avisos sobre desconocidos. Mediante un *toggle* se configura la activación o no de este tipo de eventos.

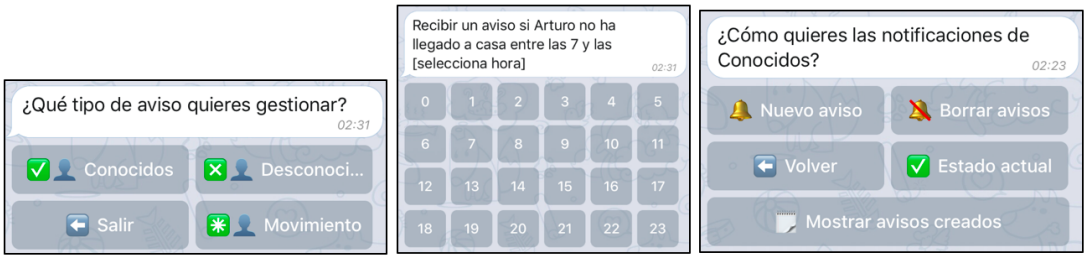


Figura 3.9: Ejemplos de creación y gestión de Avisos personalizados

- **Movimiento:** Bloque que permite configurar los avisos por movimiento. Mediante otro *toggle* se gestiona la activación y desactivación de los eventos.

3.1.8 Servidor Web

Se ha desarrollado un servidor web que ofrezca una página de presentación de la aplicación. De las numerosas implementaciones posibles, en un principio se optó por utilizar el *web framework* para Python denominado Bottle. Con este módulo se ha creado un servidor web que al recibir las peticiones GET mostraba un texto HTML básico con una guía de la aplicación.

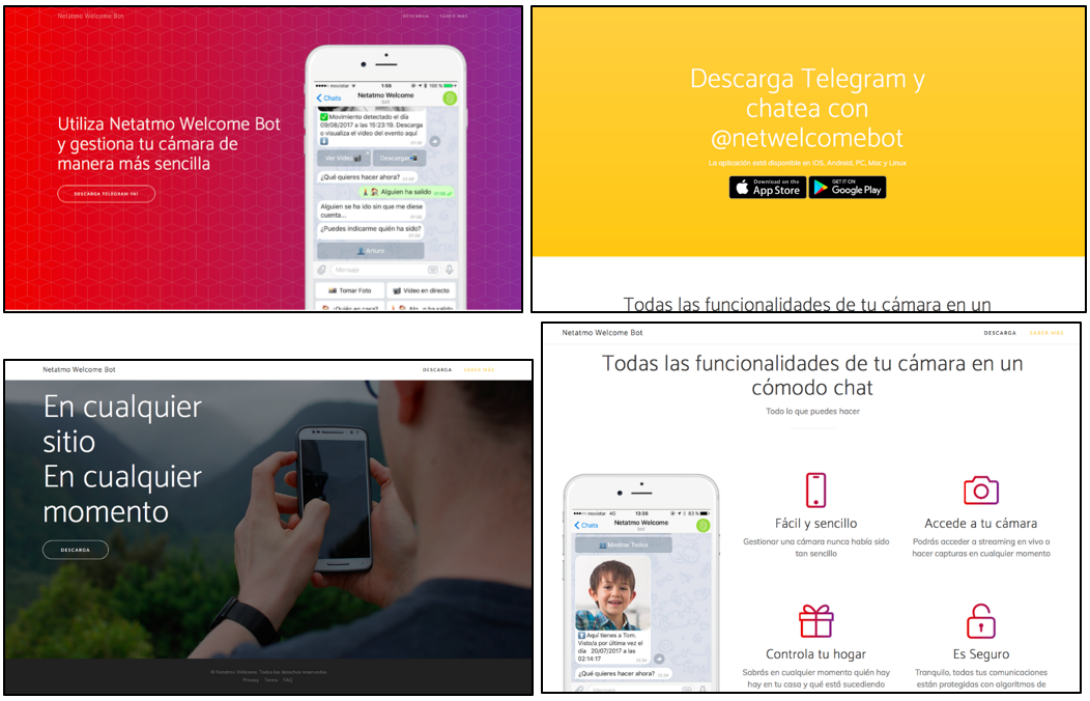


Figura 3.10: Ejemplos de creación y gestión de Avisos personalizados

Se ha decidido optar por una segunda vía más completa, instalar un servidor

HTTP Apache en la máquina y desarrollar una web con Bootstrap; una de las muchas ventajas de usar los elementos de Bootstrap, es que son adaptativos (*responsive*), lo cual nos permite obtener una correcta presentación de la página web en dispositivos de distinto tamaño, cómo smartphones.
Para ello se ha asociado el servidor Apache2 con el nombre del servidor de la máquina y posteriormente se ha configurado el *firewall* para permitir las conexiones entrantes por el puerto 80. Se ha instalado un *template* ya desarrollado por Bootstrap, y a partir de este se han hecho modificaciones en el código para mostrar la información relativa a nuestro proyecto. Se muestra una captura [3.10] con varias secciones de la página.

3.2 Diagrama de actividad

Se ha realizado un diagrama de actividades en notación UML, éste es capaz de mostrar visualmente el flujo entre los pasos de nuestro sistema favoreciendo la compresión del proceso. Se indican los dos eventos que despiertan al proceso:

- Llegada da un mensaje al servidor.
  - Usuario registrado con credenciales activas.
  - Usuario sin registrar
- Nuevo evento producido (notificación)

En el caso de llegada de mensajes al servidor hay que diferenciar dos grandes tipos, si el usuario está registrado en el sistema, saltará toda la fase de registro y permitirá directamente la gestión de la cámara, sin embargo, el no tener credenciales o ser erróneas indicará al usuario la necesidad de estas y sólo permitirá acceso al registro o a los menús de ayuda. Se muestran los nodos de decisión más importantes y los diferentes procesos que tienen lugar durante la ejecución de las tareas. La recepción de nuevos mensajes, que no cumplan los requisitos necesarios serán desechados.

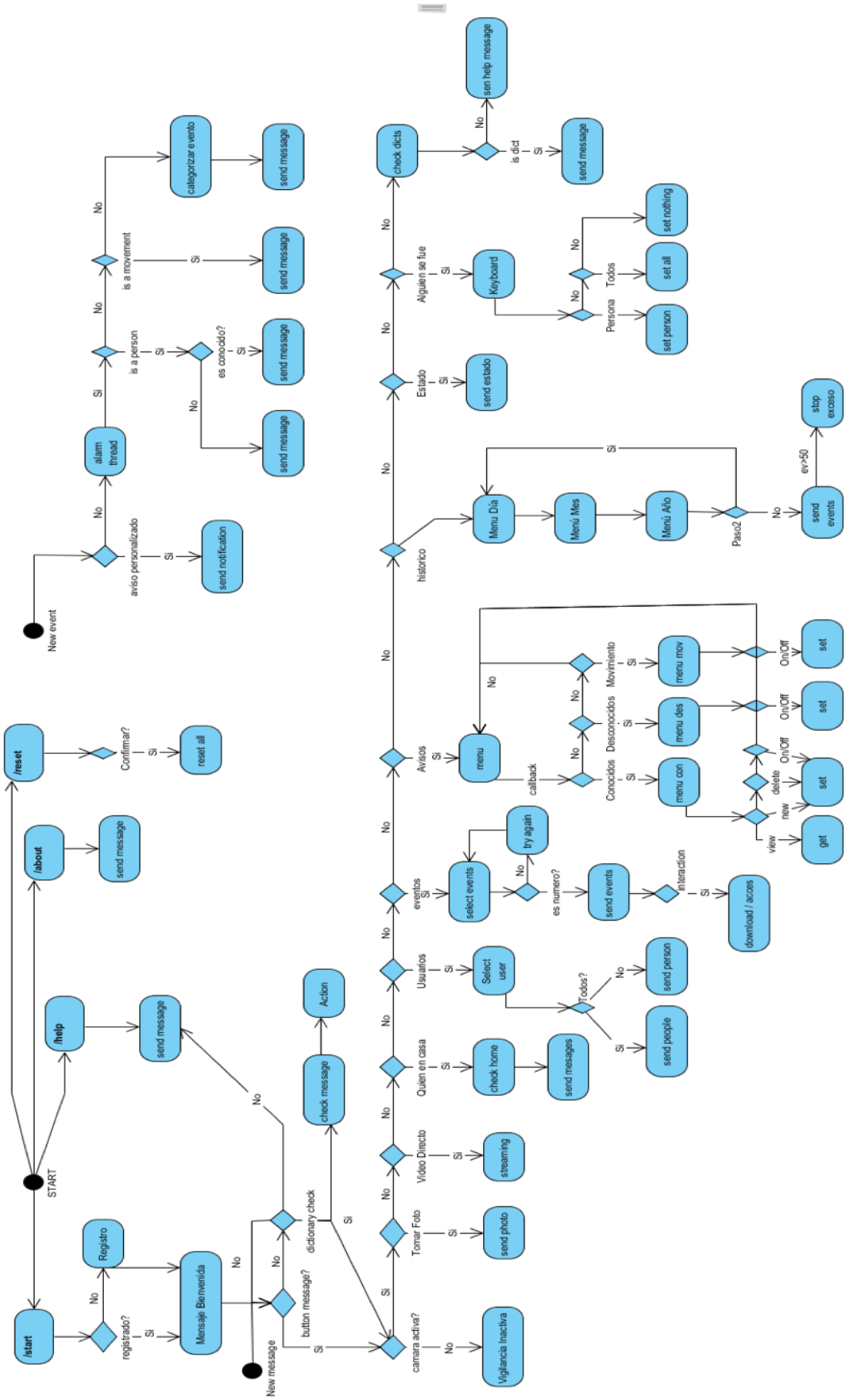


Figura 3.11: Diagrama de actividad

3.3 Implementación

Para poder tener un servidor corriendo indefinidamente, se ha optado por utilizar los servicios de computación en la nube, para ello creamos una máquina Ubuntu 16.04.3 LTS, del tipo EC2 en Amazon Web Services. Se accede a esta máquina sobre el protocolo seguro SSH, Amazon envía en el periodo de creación una clave privada para evitar el uso de contraseñas; que debemos conservar para poder autenticarnos.

```
arturo$ ssh -i "ubuntuAWS.pem" ubuntu@botarturo.hopto.org
```

Para gestionar archivos entre la máquina AWS y nuestro equipo, que incluyan el registro de las interacciones, el número de usuarios diarios, los eventos producidos y el listado de usuarios registrados se ha utilizado el protocolo SCP.

```
scp -i "ubuntuAWS.pem" ubuntu@botarturo.hopto.org:/home/ubuntu/tfg/history.txt
/Users/arturo/Desktop
```

Se ha utilizado el servicio No-IP para proveer a nuestra máquina un DNS público accesible (*botarturo.hopto.org*). La configuración de los puertos de la máquina es necesaria, se utilizan el 8443 que será el que utilicemos para gestionar los *webhooks* y el 22 por el que realizamos la conexión vía SSH. Además, se ha montado un servidor web Apache para mostrar información vía web, accesible por el puerto 80.

La aplicación para el cliente, se distribuye como un único ejecutable. Para el correcto funcionamiento de la aplicación se deberán tener configuradas todas las librerías y módulos necesarios (ver Anexos).

Para conectar nuestro sistema con Telegram es necesaria la creación de un identificador del bot que se expide a través de un chat en la aplicación cliente con

@BotFather, donde Telegram crea la asociación y nos entrega el token de acceso. Una vez expedido el token, desde @BotFather se puede personalizar el nombre del bot, añadir una descripción de sus funcionalidades, configurar privacidad, editar la imagen de perfil e incluye la creación de una guía con los comandos útiles para el usuario.

Capítulo 4

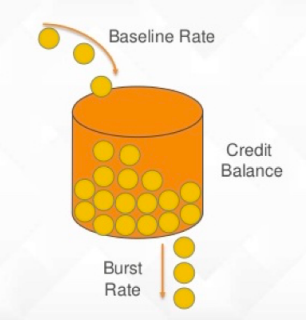
Banco de Pruebas

En este apartado se enumeran distintos test que se han hecho a la aplicación para valorar su rendimiento.
Referido a la tolerancia que tiene el bot a interacciones por parte del usuario, es total, en parte, gracias a la buena arquitectura de la API de Telegram. Los mensajes que llegan y no están recogidos en ningún manejador se desechan (si quisiéramos enviar una nota de voz o una imagen, por ejemplo). Por otra parte, la disponibilidad completa del servicio queda sujeta al funcionamiento correcto de los servidores de Netatmo.

4.1 Rendimiento del sistema

Ahora nos centraremos en la carga que soporta la máquina virtual, el número de usuarios simultáneos que permite y si necesitamos establecer algún límite en el uso.
Por un tema de privacidad no se ha permitido acceso al bot a muchos usuarios, por lo cual no se ha podido probar a nivel real hasta qué punto puede aguantar una cantidad notable de usuarios. Todas las mediciones se han hecho a través de *Cloud Watch Management Console*, una aplicación de AWS que monitoriza automáticamente diversos parámetros de las instancias activas.
Antes de todo, es importante saber los límites y el rendimiento de la máquina en cuestión. La instancia arranca con un balance de créditos de CPU suficiente para

que el rendimiento sea bueno, mientras la instancia esté en modo *idle* (o sin llegar a usar su *baseline performance*) los créditos CPU se acumulan en nuestra t2.micro a un ritmo de 6 créditos por hora, hasta llegar a 24 horas (144 créditos máximo), donde si no se han gastado no se seguirán acumulando (ver [4.1]).



Instance type	Initial (boot) credits	Credits earned per hour	Base performance (CPU utilization)	Maximum CPU credit balance
t2.micro	30	6	10%	144
t2.small	30	12	20%	288
t2.medium	60	24	40%(*)	576

Figura 4.1: Balance de créditos en AWS

En el caso que una instancia requiera de un nivel de CPU superior a su *baseline performance* en momentos puntuales (ráfagas) usará los créditos disponibles acumulados hasta terminarlos y seguirá con el rendimiento marcado en su *baseline performance*.

4.1.1 Tolerancia a peticiones

Se ha simulado con dos dispositivos simultáneos enviando peticiones continuamente durante 10 minutos (sin peticiones de conversión de vídeo) y apenas se ha superado un 1 % [5.1] de utilización de la CPU. Si hacemos la conversión a créditos AWS [4.1] en una hora se consumirán aproximadamente 0,6 créditos.
Si disponemos de 6 créditos por hora, siguiendo una progresión lineal se estima que el bot soportará 20 usuarios simultáneos haciendo peticiones ininterrumpidamente (algo poco probable). De esta forma se conservaría intacto el *credit balance*. Si se quisiera consumir completo y bloquear el sistema harían falta 480 usuarios adicionales haciendo peticiones ininterrumpidas durante una hora.

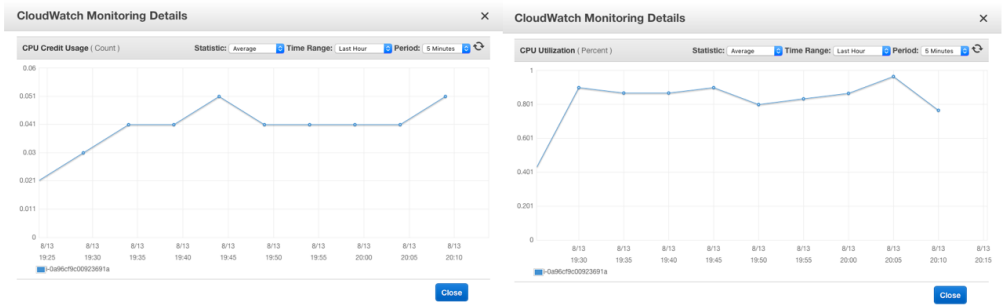


Figura 4.2: Monitorización Dashboard CPU Credit Usage y CPU Utilization

4.1.2 Tolerancia a conversión de video

Cuando se hacen peticiones de descarga de vídeo el análisis debe ser distinto. Cada petición de descarga, incluye una conversión en la codificación del archivo de m3u8 a mp4 que conlleva un alto coste en términos de uso de CPU y provee resultados distintos a los anteriores. Para analizar esto se ha monitorizado el uso de CPU durante la conversión de archivos.

El proceso de conversión de un archivo de 35 segundos (lo más usual) consume 1,25 créditos, y tarda alrededor de 2 minutos en servirse. Se probaron dos posibles implementaciones:

1. Sin utilizar threads las descargas siguen una cola FIFO, es decir, se podrán servir un máximo de 1,25 créditos * 30 operaciones/hora = 37?5 créditos/hora. Si el *credit balance* es de 144 créditos, se podría bloquear el sistema en 4 horas con peticiones continuas de conversión de vídeo.

2. Cada conversión abre un nuevo *thread* para permitir servir todas las peticiones al mismo tiempo. Es preferible porque permite varias peticiones por usuario, y en el caso que el buffer de conversión se ve sobrepasado corta la conversión. En la ilustración [5.3] se ve la utilización de créditos en el modo *threads* y el número de bytes que pasan por las interfaces. Que haya en torno al triple de paquetes de entrada que de salida se debe al hecho de que un archivo m3u8 es un contenedor de archivos de vídeo con distintas resoluciones. De salida únicamente se envía el resultado en mp4.

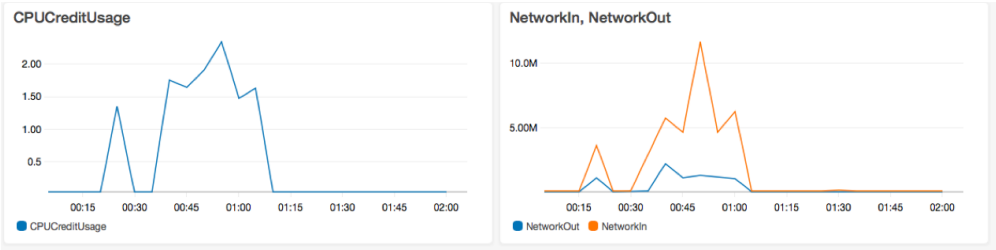


Figura 4.3: Monitorización ancho de banda

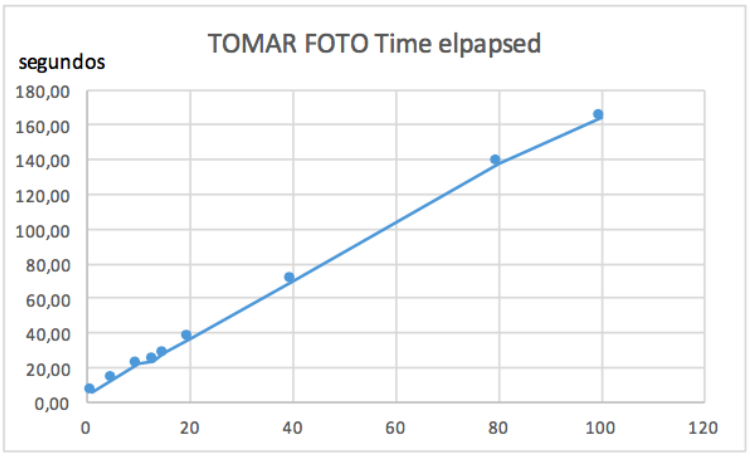
Se ha realizado una gestión eficaz de este contenido. Con el objetivo de optimizar el espacio de la máquina virtual (8gb), todos los archivos que se intercambian con el usuario se sobrescriben con cada nueva petición, de tal forma que solo tenemos que almacenar un archivo de imagen y otro de vídeo.

4.1.3 Cliente Personalizado

También se ha desarrollado una aplicación cliente personalizada. Este módulo está construido a partir de la librería Telethon escrita en Python. Se ha implementado para poder generar peticiones y analizar el rendimiento del sistema, no sin advertir que han surgido problemas, Telegram como método de seguridad banea aquellos clientes que realicen determinado número de peticiones en un corto periodo de tiempo. Se ha desarrollado un cliente sencillo. Telegram nos ha proporcionado un identificador de API y un hash, que junto con un código de verificación son suficientes para arrancar la aplicación.

El script envía peticiones al bot de forma ininterrumpida para someter a esfuerzos al servidor. Tiene como objetivo automatizar las peticiones para hacer un análisis más fiable. Se ha configurado un script que realiza peticiones de forma ininterrumpida. No se han hecho más de 100 peticiones ya que a partir de este número Telegram devuelve una excepción *(420 Flood - The maximum allowed number of attempts to invoke the given method with the given input parameters has been exceded)*. imponiendo 24 horas de baneo del servicio. A partir de los resultados se ha ilustrado [4.4]. Se observa una progresión lineal en las dos funcionalidades analizadas.

TOMAR FOTO	
Peticiones	Time elpapsed
1	5,80
5	13,38
10	22,21
13	24,10
15	27,86
20	36,83
40	70,21
80	138,35
100	164,33



QUIEN EN CASA 3 PERSONAS	
Peticiones	Time elapsed
1	7,17
5	17,38
10	32,76
13	46,02
15	46,02
20	48,30
40	98,44
80	194,32
100	245,31

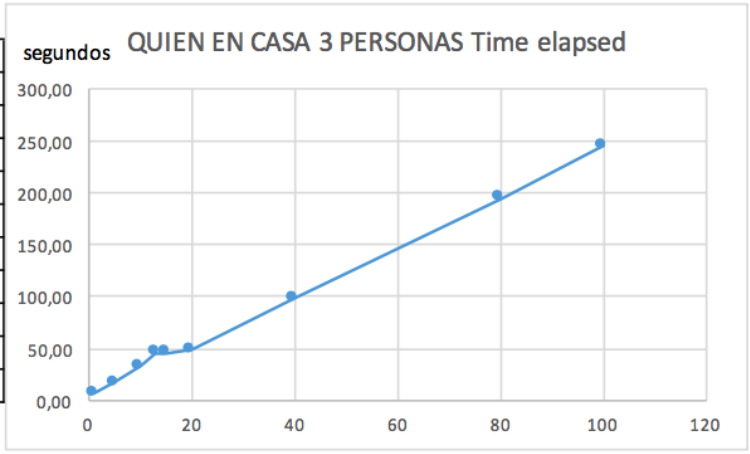


Figura 4.4: Tolerancias en Análisis personalizado

Capítulo 5

Conclusiones y líneas futuras

5.1 Conlusiones

En este trabajo Fin de Grado se han llevado a cabo todas las tareas que componen la creación de un nuevo proyecto de software. En la primera etapa, se investigó acerca del estado actual de esta tecnología, y tras estudiar numerosos gráficos y estadísticas sobre su uso llegamos a la conclusión de que esta tecnología tiene un largo camino por recorrer. Los grandes sistemas operativos ya cuentan con asistentes personales (Siri, Cortana o Bixby) y grandes empresas como Facebook empiezan a liberar APIs para desarrollarlos en sus plataformas. Creemos que uno de los puntos fuertes de estas tecnologías es la similitud que tienen con la verdadera comunicación entre personas.

Tras una primera etapa de estudio e investigación se procedió a analizar todas las tecnologías que teníamos disponibles. Desarrollarlo en Whatsapp hubiera sido ideal, pero desde un principio sabía que el resultado nunca sería tan bueno como el que se podía conseguir con Telegram, su experiencia y herramientas disponibles en el campo de los bots era mucho más sólida y además tenía una comunidad de desarrollo considerable.
El haber elegido un lenguaje como Python ha sido tremendamente acertado, se ha podido comprobar que es un lenguaje sencillo, fácilmente legible y muy potente

al mismo tiempo. Además, he podido contactar con desarrolladores, que en todo momento me han prestado su ayuda y han recibido mi *feedback* de correcciones u optimizaciones de las APIs que se han utilizado.

También se ha conocido el funcionamiento de los servicios de computación en la nube, en concreto Amazon Web Services. Considero que es una herramienta muy recomendable para desarrollar cualquier proyecto de software, permite ahorrar costes de equipos y mantenimiento, y por lo que aquí se ha visto, funcionan de manera excepcional. El periodo de pruebas ha consistido en poner en funcionamiento las distintas versiones de los prototipos e ir utilizando el software día a día. De esta forma se han encontrado multitud de pequeños, y no tan pequeños fallos que se fueron corrigiendo hasta llegar a una versión estable.

En definitiva, a través del protocolo SSH, conseguimos conectar con una máquina Linux Ubuntu, con el objetivo de hacer funcionar en este un programa Python que actúa como servidor del cliente de mensajería Telegram. Este servidor se encarga de gestionar una cámara con detección de movimiento y reconocimiento facial. Consigue extraer la información a través del protocolo http y, tras darle forma y almacenarla correspondientemente, se ofrece de manera visual al cliente. El bot al final es, por tanto, la interfaz gráfica visible finalmente por el usuario.

Por todo esto, puedo decir a título personal que he sentido como todo lo aprendido durante estos años se ha materializado en forma de este proyecto, y sumando mi deseo de conocer en profundidad productos con cierta inteligencia artificial como este, dan como resultado una gran experiencia de trabajo.

5.2 Líneas de futuro

Sobre la temática de este TFG se proponen nuevas líneas de investigación viables:

- **1.** Añadir funcionalidades de otros dispositivos al bot sería una línea muy interesante. Netatmo dispone de otros dispositivos inteligentes como termostatos o estaciones meteorológicas. Aunar la gestión de todas ellas en un mismo bot tendría un gran resultado.

- **2.** Se podría incluso canalizar en el bot la gestión de todos los dispositivos conectados en el hogar, bombillas programables, enchufes inteligentes, persianas automáticas. El resultado sería poder controlar todo el hogar de forma sencilla.

- **3.** Implementación en otras plataformas de mensajería como Whatsapp. El motivo de esta línea de desarrollo es únicamente por la fuerte presencia social que tiene esta plataforma. Sería más fácil llegar a los usuarios.

- **4.** Mejorar la inteligencia en la comprensión del texto introducido. Aumentando la complejidad de los bloques condicionales de los diccionarios con RE o utilizando librerías de reconocimiento de texto como wit.ai.

Bibliografía

[1] Telegram Bot API: https://core.telegram.org/bots/api,

[2] Bots in messaging platforms, a new paradigm in healthcare delivery: application to custom prescription in dermatology. A. Alesanco, J.Snacho, Y.Gilaberte, E.Abarca and J.GarcÃa. ,

[3] Centre Seguretat TIC de la Comunitat Valenciana: https://www.csirtcv.gva.es/es/noticias/as%C3%AD-funciona-la-seguridad-de-telegram.html ,

[4] MTProto Mobile Protocol: https://core.telegram.org/mtproto ,

[5] Python Telegram Bot API: https://github.com/eternnoir/pyTelegramBotAPI ,

[6] Netatmo Connect APIs Reference Guidelines Smart Home API https://dev.netatmo.com/resources/technical/reference ,

[7] cherryPy - A minimalist Python Web Framework: http://docs.cherrypy.org/en/latest/ ,

[8] Ffmpeg Documentation: https://ffmpeg.org/ffmpeg.html ,

[9] Conceptos BÃ¡sicos de OAuth2: http://www.thegameofcode.com/2012/07/conceptos-basicos-de-oauth2.html ,

[10] Entendiendo OAuth https://es.scribd.com/document/91623356/Entendiendo-oAuth ,

[11] Comscore Statista Universal htpp://comscore.com ,

[12] Telegram Bot API: https://core.telegram.org/bots/api,

[13] Google Consumer Barometer Report ,

[14] Stack Overflow Python Q&A https://stackoverflow.com/questions/,