Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Trabajo Fin de Máster

Desarrollo de un bot en la plataforma de mensajería Telegram para el seguimiento de actividad mediante pulseras fitness

Development of a bot on Telegram messaging platform for activity tracking using fitness wristbands

Autor:

Fernando Alegre Ojer

Director:

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. Fernando Alegre Ojer,

con nº de DNI 73009635E 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) Máster, (Título del Trabajo) Desarrollo de un bot en la plataforma de mensajería Telegram para el seguimiento de actividad mediante pulseras fitness,

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

Zaragoza, 5 de junio de 2017

Fdo: Fernando Alegre Ojer

Agradecimientos

Este trabajo fin de máster supone la culminación de una etapa importante de mi vida. Durante este tiempo mucha gente ha estado a mi lado y este es un pequeño agradecimiento a su constante apoyo.

Quisiera dar las gracias, en primer lugar, a ***Álvaro*** *por haberme guiado tanto en el presente trabajo como en el anterior trabajo fin de grado y por todo lo que me ha enseñado durante mi etapa universitaria. Me gustaría agradecerle también la confianza que ha depositado en mi y la ayuda que me ha brindado en todo momento.*

A mis padres y mi hermana, quienes verdaderamente han hecho posible que haya llegado hasta aquí. Gracias por aguantarme cada día, por alentarme en los malos momentos y proporcionarme toda la ayuda que necesitaba.

A mis primos, ***Guillermo*** *y* ***Nacho****, por ofrecerse como testers de la aplicación y, en definitiva, al resto de mi familia por estar a mi lado cada fin de semana.*

A todos mis amigos que he conocido durante mi etapa universitaria por haber llenado de risas las interminables horas que hemos pasado juntos. El camino se ha hecho más fácil gracias a vosotros.

A mis amigos de ***Luesia****, por todos los momentos que hemos vivido juntos desde que tengo uso de razón.*

A mis amigos de ***Zaragoza****, porque seguimos juntos desde hace 23 años y espero que durante el resto de nuestras vidas.*

Al resto de amigos que se han tomado un ratito de su tiempo para preguntarme cómo avanzaba con el trabajo y ver qué tal me iba la vida.

En definitiva, a todas las personas importantes para mí, simplemente gracias.

Desarrollo de un bot en la plataforma de mensajería Telegram para el seguimiento de actividad mediante pulseras fitness

RESUMEN

Actualmente se está produciendo un boom en la utilización de aplicaciones automáticas, llamadas comúnmente bots, en la plataforma de mensajería instantánea Telegram. Estos bots permiten automatizar muchas funcionalidades como pueden ser obtener imágenes, vídeos o cualquier otro contenido multimedia, obtener una previsión meteorológica o programar recordatorios, entre muchas otras.

Por otro lado, las pulseras fitness también están ganando auge entre la población debido a la innovación de mezclar la tecnología con la realización de ejercicio físico y la cantidad de funciones que ofrecen estos dispositivos frente a los costosos Smartwatch, salvando las diferencias entre ambos dispositivos.

Por lo tanto, el objetivo de este proyecto es combinar los datos de actividad obtenidos con estas novedosas pulseras fitness con un bot de Telegram de modo que ofrezca a los usuarios un seguimiento de su actividad diaria, semanal, etc.

Para ello se ha diseñado una arquitectura de almacenamiento y procesado basada en computación en la nube (Amazon Web Service) que permite al usuario tanto almacenar sus datos como poder obtener gráficas de sus registros de actividad (pasos, calorías consumidas, actividad diaria) mediante la utilización de un bot desarrollado en Telegram. Para el diseño de esta arquitectura se han integrado módulos existentes (el servidor de autenticación OAuth 2.0 y la Fitness Store de Google, el stack ELK) con implementaciones propias (bot, servidor HTTP) creando una estructura de microservicios. Las arquitecturas basadas en microservicios constituyen un enfoque para desarrollar aplicaciones sofware como pequeños servicios que se ejecutan de forma autónoma y se comunican entre sí, a través de peticiones HTTP. Para utilizar los servicios provistos por la arquitectura se han definido las interacciones que deben seguirse y se han implementado los módulos antes mencionados, utilizando diferentes tecnologías como Python, JavaScript y JQuery, capaces de recoger los datos procedentes de la pulsera inteligente y transmitirlos por los diferentes elementos de la arquitectura hasta que

finalmente son almacenados en una base de datos no relacional llamada ElasticSearch. Posteriormente estos datos se monitorizan y se le devuelven al usuario en forma de imagen gráfica ya que facilitan la interpretación de dichos registros.

Índice general

Índice de figuras

Capítulo 1

Introducción y Objetivos

1.1 Los bots y las pulseras fitness

Hoy en día las "tradicionales" aplicaciones de mensajería como WhatsApp, Facebook Messenger o Telegram intentan implementar nuevas características que doten a sus aplicaciones de un mayor atractivo y usabilidad e innovar en las tradicionales formas de comunicación entre usuarios.

Algunos ejemplos de estas nuevas funcionalidades podrían ser los nuevos estados del WhatsApp o los programas informáticos que intentan imitar el comportamiento humano, denominados comúnmente, bots. Los chat bots son aquellos chats que imitan una conversación de mensajería instantánea y responden y reciben órdenes del usuario. Estos bots permiten automatizar muchas funcionalidades como pueden ser obtener imágenes, vídeos o cualquier otro contenido multimedia, obtener una previsión meteorológica o programar recordatorios, entre muchas otras. Telegram fue una de las primeras plataformas de mensajería en implementar estos nuevos tipos de chats.

Por otra parte, la aparición de pulseras fitness ha aumentado el interés de la gente en realizar ejercicio y controlar sus estadísticas diarias como pueden ser la distancia recorrida, las pulsaciones o el control del sueño.

Teniendo en cuenta todo esto sería interesante el diseño de una plataforma que aproveche los bots proporcionados por Telegram para almacenar y presentar

al usuario las estadísticas de su rutina diaria de forma que le motiven aún más a realizar ejercicio en un afán de auto-superación o, simplemente, permita a las personas mayores a realizar un seguimiento de su actividad en un dispositivo tan común hoy en día como es su Smartphone.

1.2 Objetivos

El objetivo principal de este proyecto es el desarrollo de una arquitectura que, utilizando los bots proporcionados por la aplicación Telegram y la pulsera inteligente de Xiaomi (Mi Band 2), permita realizar un seguimiento de la actividad de los usuarios. Se plantean además una serie de objetivos específicos para el correcto funcionamiento del sistema:

- Análisis de los métodos de obtención de la información de las pulseras inteligentes y cómo almacenarlos en la aplicación.
- Implementación de un bot que procese las peticiones de los usuarios y les presente posteriormente los resultados requeridos de forma visual.
- Diseño y desarrollo de la arquitectura basada en microservicios que permita realizar el procesado de las peticiones en la nube.
- Adaptación de los datos recopilados por las pulseras a un recurso del estándar de comunicación médico FHIR pensando que en un futuro pudiese tener una aplicación e-Health.

1.3 Materiales y herramientas utilizadas

Para la realización de este proyecto se han utilizado los siguientes recursos hardware y software.

- Harware:
  - **Xiaomi Mi band 2:** Pulseras inteligentes de bajo coste con conexión Bluetooth que permiten recopilar la información de la actividad de los usuarios.
  - **Smartphone:** Con conectividad Bluetooth para obtener los datos de actividad antes mencionados y conectividad de internet (WiFi/Datos móviles) para comunicarse con el bot mediante la aplicación Telegram.
  - **Servidor en la nube:** Una instancia en Amazon Web Service con sistema operativo Amazon Linux AMI (CentOS) que permita tanto ejecutar el bot como el almacenamiento y procesado de los datos obtenidos por las pulseras.
- Software:
  - Apps móviles que el usuario deberá instalar (desde Google Play) para obtener los datos de su actividad:
    - **MiFit:** Aplicación de Xiaomi que permite la sincronización de la pulsera inteligente con el Smartphone para transferir los datos recopilados. Como tiene una base de datos propietaria será necesario exportar estos datos a la aplicación de Google "Google Fit" de modo que sean accesibles.
    - **Google Fit:** Aplicación de Google que utilizaremos como "puente" para poder obtener la información en el servidor.
  - **Amazon Linux Ami 2016.03.3:** Es una distribución gratuita de Linux sobre el que se ha realizado el desarrollo de la parte del servidor.
  - **Python:** Lenguaje de programación interpretado, multiparadigma y de código abierto. Se ha utilizado como entorno de desarrollo, IDLE, la

aplicación PyCharm ya que facilita la programación al contar con una interfaz gráfica bastante atractiva.

- **pyTelegramBot** implementación en Python para la API de Telegram Bot.
- **Google-api-python-client 1.5.3** librería que permite el acceso a las APIs de Google.
- **oauthclient 3.0.0** librería empleada para la autenticación OAuth.
- **rsa 3.4.2** librería que permite realizar tareas de encriptado y desencriptado, firmado y verificación de firmas digitales y generación de claves.
- **simple-json 3.8.2:** librería para codificar y decodificar objetos JSON.
- **BaseHTTPServer:** librería que implementa un servidor HTTP.
- **httplib2 0.9.2** librería utilizada en la gestión de peticiones HTTP.
- **Requests:** librería para la generación de peticiones HTTP.
- **PhantomJS:** librería que permite lanzar un navegador web sin interfaz visual.

– **JavaScript y JQuery:** JavaScript es un lenguaje de programación interpretado, orientado a objetos, débilmente tipado y dinámico. JQuery es una biblioteca multiplataforma de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, modificar el árbol DOM (Document Object Model), etc. Ambos son software libre y de código abierto. Se utilizan principalmente en su forma del lado del cliente permitiendo mejoras en la interfaz de usuario y páginas web dinámicas (las gráficas de Kibana están basadas en código JavaScript). Como IDLE de desarrollo se ha utilizado la aplicación XCode que es el entorno de desarrollo en sistemas con sistema operativo MAC OS X.

- **Stack ELK:**
  - **ElasticSearch:** base de datos no relacional (NoSQL) que almacena sus datos como documentos en formato JSON. Estas bases de datos están enfocados a su uso en Big Data.
  - **Logstash:** filtro de logs que permite formatear los datos de entrada y prepararlos con una estructura JSON determinada para su posterior almacenamiento en ElasticSearch.
  - **Kibana:** frontal web que permite la representación gráfica para realizar una monitorización de los documentos almacenados en la base de datos ElasticSearch.
- **VirtualBox:** Entorno de virtualización.
  - **CentOS 7:** Como la versión gratuita de AWS sólo ofrecía una máquina virtual con 500 Mb de memoria RAM se utilizó para poder ejecutar toda la arquitectura una máquina virtualizada en mi equipo local.
- **Vagrant:** Se trata de una herramienta open source para la creación y configuración de entornos de desarrollo virtualizados. Aunque originalmente se desarrolló en Ruby, se puede utilizar en otro tipos de proyectos escritos en otros lenguajes de programación como Python, PHP, Java, etc.
- **Ansible:** Se trata de una herramienta de open source para configurar y administrar equipos. Permite manejar nodos a través de SSH y no requiere ningún software remoto adicional (excepto Python 2.4 o superior para instalarlo). El lenguaje nativamente utilizado para describir configuraciones reusables de los sistemas es YAML.

1.4 Organización de la memoria

La memoria está estructurada de la siguiente manera:

- **Capítulo 1. Introducción:** Contiene una breve descripción del trabajo realizado; así como de sus principales objetivos y las herramientas utilizadas para su ejecución.
- **Capítulo 2. Estado del arte:** Se describe el estado actual de los diferentes elementos que se han utilizado para desarrollar la arquitectura.
- **Capítulo 3. Arquitectura del sistema:** Presenta la arquitectura desarrollada, los diferentes módulos que la componen y su interacción entre ellos.
- **Capítulo 4. Resultados:** Presenta los resultados obtenidos y el modo en que se le presentan los datos al usuario.
- **Capítulo 5. Conclusiones y líneas futuras:** Contiene las conclusiones que se han sacado de este proyecto, sus limitaciones y las posibles líneas futuras que se podrían seguir para mejorarlo.

Adicionalmente se presentan los siguientes anexos:

- **Anexo A:** Acrónimos.
- **Anexo B:** DataSources.
- **Anexo C:** DataSets.
- **Anexo D:** FHIR.
- **Anexo E:** WebScraping.
- **Anexo F:** Configuración Vagrant.
- **Anexo G:** Configuración Ansible.

En el anexo A se tiene una lista de los acrónimos utilizados en esta memoria.

En el anexo B se presenta la estructura de DataSources enviada por la Fitness Store en formato JSON.

En el anexo C se presenta la estructura de DataSets enviada por la Fitness Store en formato JSON.

En el anexo D se muestra el recurso FHIR utilizado, *Observation*, donde se resaltan los campos que se han utilizado.

En el anexo E se explica cómo se aplica el método de WebScraping para obtener la gráfica deseada y se muestra el script que lo permite.

En el anexo F se presenta el archivo de configuración de Vagrant que permite configurar rápidamente una máquina virtual.

En el anexo G se presenta el archivo de configuración de Ansible que permite configurar en unos pocos minutos la arquitectura desarrollada.

Capítulo 2

Estado del Arte

2.1 Las pulseras inteligentes

Las pulseras inteligentes, también llamadas pulseras fitness, son unos dispositivos de bajo costo capaces de cuantificar la actividad física de los usuarios. Estas pulseras no sólo permiten obtener valores del usuario sino que permiten avisarle mediante las notificaciones que recibe en su Smartphone, utilizarlas como despertador y obtener la frecuencia cardíaca entre otras funciones.

La pulsera Mi Band 2, de la conocida marca de móviles Xiaomi, nos proporciona una pulsera con una autonomía de unos 20 días. Además de contar con una pantalla donde poder visualizar los datos de las diferentes funcionalidades que ofrece, esta pulsera cuenta con una conexión Bluetooth con la que conectarse a un Smartphone y trasmitir los valores registrados por el usuario. Entre estos valores se encuentran los pasos diarios, las calorías consumidas, el ritmo cardíaco instantáneo o los registros de sueño, por ejemplo. Una vez en la aplicación del dispositivo móvil, llamada Mi Fit, se almacenan los valores en la base de datos de Xiaomi. Esta aplicación permite conectarse a otras aplicaciones como por ejemplo a la aplicación de fitness de Google, llamada Google Fit, y transmitirle los valores capturados por la pulsera.

2.2 Los bots de Telegram

Desde mediados del 2015, Telegram cuenta con bots en su aplicación. Se tratan de pequeños programas que se encuentran ejecutándose en un servidor independiente de la plataforma Telegram y podemos interactuar con ellos en cualquier momento para iniciar la funcionalidad para los que están programados.

Telegram tiene una extensa documentación que nos explica cómo podemos crear este tipo de aplicaciones para ser utilizadas en la plataforma de mensajería instantánea. Para utilizarlos tenemos que abrir una ventana de chat en Telegram con ellos o incluirlos en el grupo o conversación que deseemos utilizarlos. Esto es tan sencillo como añadir un usuario más a dicha conversación, lo llamaremos con su nombre añadiéndole una arroba al principio.

2.3 Microservicios

Una Arquitectura de microservicios es un distintivo sistema de desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP). En la arquitectura, cada servicio se encarga de implementar una funcionalidad completa del negocio. Cada servicio es desplegado de forma independiente y puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos.

Una definición más precisa de microservicios serían pequeños servicios autónomos que trabajan juntos. Son pequeños y se concentran en una tarea para realizarla correctamente (dar una funcionalidad). Con esto se consigue una mayor facilidad a la hora de realizar modificaciones frente a tener que realizarlas en un fragmento de código muy extenso. Son autónomos porque se tratan de entidades separadas que se pueden desplegadar como un servicio aislado o como un proceso del sistema operativo. Las comunicaciones entre microservicios se realizan mediante llamadas de red, para cumplir la separación entre los servicios y evitar los peligros de acoplamiento.

El principio de racionamiento de los microservicios radica en que los servicios tienen que poder cambiarse independientemente unos de otros y desplegarse por sí solos sin necesidad de que el resto cambie.

Utilizar una arquitectura de microservicios proporciona los siguientes beneficios:

1. Usar una tecnología diferente en cada servicio (diferente lenguaje de programación) dependiendo de nuestras necesidades.

2. Ante un fallo de red se puede aislar el servicio problemático mientras el resto continúa funcionando.

3. Se pueden escalar los servicios permitiéndonos asignar diferentes cantidades de recursos dependiendo de las necesidades de cada uno.

4. Facilita el despliegue en caso de pequeñas modificaciones.

5. Permiten reemplazar o eliminar implementaciones de una forma más sencilla.

2.4 El stack ELK

El stack ELK es un paquete de tres herramientas open source de la empresa Elastic. Estas herramientas son ElasticSearch, Logstash y Kibana. Se trata de herramientas que pueden funcionar por separado pero es cuando se utilizan conjuntamente cuando explotan su máximo potencial. Estas herramientas permiten leer y almacenar la información necesaria para posteriormente consultarla o monitorizarla. Su principal aplicación es el tratamiento de logs de todo tipo.

ElasticSearch se trata de un servidor de búsqueda basado en Lucene. Provee un motor de búsqueda de texto completo a través de una interfaz web RESTful, es decir, provee un servicio de búsquedas a través de una API RESTful. Mediante peticiones HTTP podemos almacenar información de forma estructurada en Elasticsearch para que éste la indexe, y posteriormente poder realizar búsquedas sobre ella.

Logstash es una herramienta para la administración de logs. Se encarga de recolectar, parsear y filtrar los logs para posteriormente darles alguna salida como por ejemplo, guardarlos en ElasticSearch. Estos logs le pueden llegar a Logstash desde el mismo servidor o desde un servidor externo. La aplicación se encuentra basada en jRuby y requiere de Java Virtual Machine para ejecutarse.

Kibana es una herramienta analítica open source (licencia Apache) que permite interactuar con la información almacenada en ElasticSearch y monitorizarla. Para la creación de gráficas utiliza JavaScript lo que permite mostrar gráficas dinámicas que se van modificando en tiempo real.

El flujo de información es el siguiente:

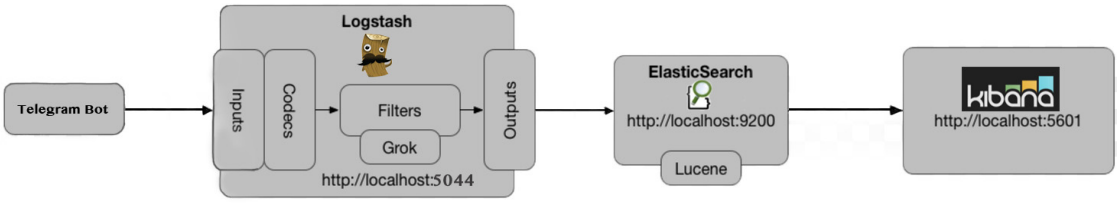

Figura 2.1: Diagrama del stack ELK.

2.5 Recursos FHIR

FHIR responde a las siglas de Fast Healthcare Interoperability Resources y se trata del último estándar desarrollado y promovido por la organización internacional HL7 (Health Level Seven), responsable de algunos de los protocolos de comunicaciones más utilizados hoy en día en el ámbito sanitario. Este estandar trata de combinar lo mejor de cada uno de los estándares actualmente en uso (HL7 versión 2, versión 3 y CDA) con estándares web modernos de forma que se mejore en la medida posible la implementación de los estándares de interoperabilidad.

FHIR parte del concepto fundamental de *Recursos*, donde un recurso es la unidad básica de interoperabilidad, la unidad más pequeña que tiene sentido intercambiar. Los recursos son representaciones de conceptos del mundo sanitario:

paciente, médico, problema de salud, observación, etc. En este trabajo se empleará el recurso *Observation* cuya estructura se muestra en detalle en el anexo D.

FHIR está diseñado específicamente para la web y sus recursos se basan en estructuras XML o JSON que utilizan un protocolo REST basado en HTTP. Esto permite realizar una analogía entre el recurso Observation y los documentos (JSONs) utilizados por ElasticSearch. En el capítulo 3 se explicará cómo se realiza la adaptación de los datos obtenidos en el bot a un recurso *Observation* de modo que a la hora de almacenarlos como documentos en ElasticSearch tengan estructura de recurso FHIR.

Capítulo 3

Arquitectura del sistema

Se ha diseñado e implementado una arquitectura de almacenamiento y procesado de los datos de actividad de los usuarios, basada en la computación en la nube (mediante un servidor de Amazon, llamado Amazon Web Service), que permite una aplicación en tiempo real que presenta al usuario gráficas de los datos que ha recogido durante su actividad diaria.

En la Fig. 3.1 se muestra el esquema de la arquitectura desarrollada.

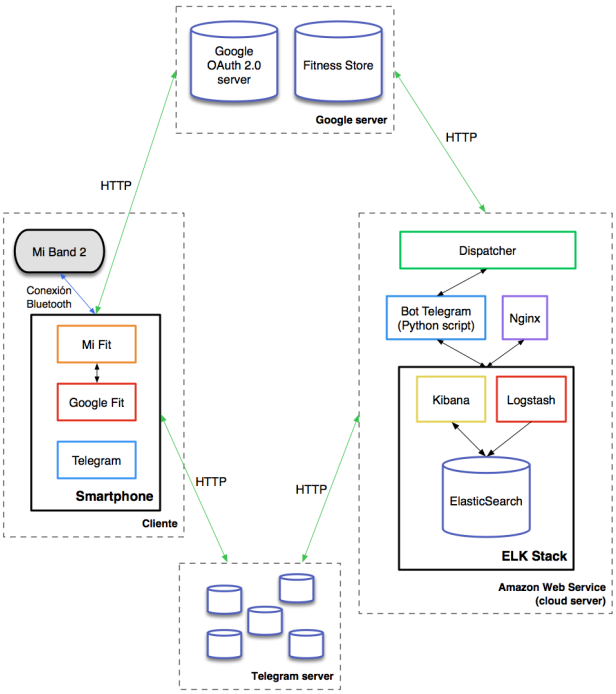

Figura 3.1: Diagrama de la arquitectura planteada.

Los elementos que forman dicha arquitectura se explicarán a continuación:

3.1 El cliente

Lo constituyen la pulsera inteligente Mi Band 2 y el Smartphone del usuario. A su vez el teléfono móvil está compuesto por varios componentes: la aplicación propietaria de Xiaomi, Mi Fit, la aplicación de Google, Google Fit y la aplicación de mensajería Telegram.

Elementos hardware del cliente:

- **Mi Band 2:** Se trata de la pulsera inteligente con la que se registrarán los diferentes datos de actividad del usuario en su día a día. La conexión Bluetooth permite enviar la información registrada al Smartphone en el momento en que se activa la aplicación que la controla, Mi Fit.

- **Smartphone:** Donde se ejecutarán las aplicaciones que se describen a continuación. Debido a limitaciones de la aplicación Mi Fit, actualmente sólo es posible su utilización en dispositivos con sistema operativo Android.

Elementos software del cliente:

- **Mi Fit:** La aplicación para smartphones de Xiaomi con la que se sincroniza la Mi Band 2, se deberá instalar desde el "market" de su dispositivo (Google Play). Una vez instalada se deberá permitir el envío de datos a la aplicación Google Fit.

- **Google Fit:** La aplicación de Google para smartphones que centraliza los datos de actividad de los usuarios que también se deberá instalar en el dispositivo. Esta aplicación utiliza como sensores el propio dispositivo o datos recibidos de otras aplicaciones, como en el caso del presente trabajo, Mi Fit. Esta aplicación está compuesta a su vez por varios elementos:

  - **Google Fitness Store:** La aplicación, una vez recibe los datos, los almacena en una base de datos llamada Google Fitness Store.

Este repositorio central reside en la nube y almacena los datos obtenidos de diferentes dispositivos y aplicaciones en la infraestructura de Google. Las aplicaciones en diferentes plataformas y dispositivos pueden almacenar y acceder a los datos creados por otras aplicaciones, por lo que se utilizarán las APIs que proporciona Google para obtener los datos de actividad en el bot.

- **Sensor Framework:** Se tratan de un conjunto de representaciones de alto nivel que facilitan trabajar con la Fitness Store. Estas representaciones se utilizan en las Google Fit APIs. Entre ellas destacan:
  - **Data Sources:** Representan sensores y están compuestos por el nombre, el tipo de datos obtenidos y otros detalles del sensor. Pueden ser sensores hardware (pulseras) o software (la propia aplicación Google Fit contabiliza los pasos, por ejemplo). Los ids de los DataSources obtenidos desde Mi Fit son:
    - **raw:com.google.step_count.delta:com.xiaomi.hm.health:**
    - **raw:com.google.calories.expended:com.xiaomi.hm.health:**
    - **raw:com.google.activity.segment:com.xiaomi.hm.health:**

    El anexo B se puede ver un ejemplo de la estructura de DataSources obtenidos.
  - **Data Types:** Representan diferentes tipos de datos de fitness (como pasos o pulsaciones). Un data type consiste en un nombre y una lista ordenada de campos donde cada campo representa una dimensión. Los data types utilizados en este proyecto son:
    - **com.google.step_count.delta** Google Fit define los pasos realizados durante un intervalo temporal mediante un entero.
    - **com.google.calories.expended** Google Fit también define las calorías consumidas durante un intervalo temporal mediante un entero.
    - **com.google.activity.segment** Google Fit define enteros constantes para cada tipo de actividad. Las contempladas en el

desarrollo son: *andando (7)*, *sueño ligero (109)*, *sueño profundo (110)* y *despierto (112)*.

- **Data Points:** Consiste en un vector de valores con timestamp para un data type, leído de un DataSource. Se utilizan para leer datos raw de un DataSource. Los data point que contienen un tiempo de inicio representan un rango de tiempo en vez de una medida instantánea.
- **DataSets:** Representan un conjunto de data points del mismo tipo de un data source particular correspondiente a un intervalo de tiempo. Se utilizan para insertar datos en el fitness store. Las peticiones para leer datos del fitness store también devuelven DataSets.

  El anexo C se puede ver un ejemplo de la estructura de DataSets obtenidos.
- **Sesiones:** Representan un intervalo de tiempo durante el cual los usuarios desarrollan una actividad como andar o correr. Las sesiones ayudan a organizar los datos y a representar consultas detalladas o agregadas al Fitness Store para una actividad.

– **Permisos y controles del usuario:** Google Fit requiere el consentimiento de los usuarios antes de que las aplicaciones puedan almacenar o acceder a datos de actividad. Por ello se definen los ámbitos de OAuth a los que se tendrá permiso de acceso. Para la aplicación de Fitness se definen tres ámbitos que se corresponden con tres grupos de permisos con distintos privilegios de lectura y escritura: la actividad, la ubicación y el cuerpo. En este proyecto sólo nos interesarán el primer y último grupo ya que cada grupo de permisos concede el acceso a un conjunto de tipos de datos. Por lo tanto en el bot se definirán los ámbitos para obtener los DataSets correspondientes a los 3 DataSources arriba mencionados:

  · https://www.googleapis.com/auth/fitness.**activity.read**

· https://www.googleapis.com/auth/fitness.**body.read**

– **Google Fit APIs:** Se utilizarán REST APIs (interfaces de programación de aplicaciones que obtienen datos o generan operaciones sobre esos datos en formatos JSON por medio de peticiones HTTP) para autenticarse con el servidor OAuth de Google, para elegir los ámbitos, para obtener información de la Fitness Store...

La conexión entre ambas aplicaciones se realiza por medio de la red de datos móviles o WiFi.

- **Telegram:** Es la aplicación de mensajería utilizada ya que permite la implementación de bots. Constituye tanto el punto de partida de los datos hacia el servidor como el punto de destino de los resultados obtenidos. A través de esta aplicación se podrán pedir los datos deseados al bot y éste devolverá la información tanto en forma gráfica como en formato texto para una mejor interpretación de los resultados. Se ha implementado un teclado de botones (adicional a los típicos teclados QWERTY) para facilitar el manejo del bot.

3.2 El servidor de Google

Hoy en día, una funcionalidad que se realiza en los primeros usos de misma, pero no por ello menos importante, es la autenticación de los usuarios; es decir, comprobar que el usuario es quien realmente dice ser. Con ello se consigue no mostrar información sensible a otra persona que no sea la que realmente debe recibir la información.

Por lo tanto, cuando queremos realizar una acción tan simple como solicitar datos de un usuario a la base de datos de Google a través de una API, las aplicaciones se tienen que autenticar. Cuando una API accede a datos privados del usuario, la aplicación también debe de ser autorizada por el usuario para acceder a los datos del mismo.

Estas APIs normalmente utilizan el protocolo OAuth 2.0 para la autorización y autenticación. Esto implica que antes de poder ejecutarlas, el usuario que tiene acceso a los datos privados debe permitir el acceso de la aplicación. Por lo tanto, la aplicación se debe autenticar, el usuario debe permitir la aplicación y el usuario debe ser autenticado para permitir el acceso. Todo esto se consigue con OAuth 2.0.

Este método de autenticación define varios aspectos importantes:

- **El ámbito:** Cada API define uno o varios ámbitos que declaran un conjunto de operaciones permitidas (lectura, escritura, lectura-escritura). Cuando la aplicación solicita acceso a los datos del usuario, la solicitud debe incluir uno o más ámbitos. Para ello, previamente el usuario necesita aprobar el ámbito de acceso que la aplicación solicita; es decir, aceptar que se vayan a leer o modificar sus datos privados.

- **Refresh y Access tokens:** Cuando un usuario permite el acceso de la aplicación, el servidor OAuth 2.0 de autorización provee a la aplicación de access token y refresh tokens que son sólo válidos para el ámbito seleccionado. La aplicación utiliza los access tokens para autorizar las llamadas de las APIs. Los access token tienen un tiempo de vida limitado, por lo que, transcurrido un tiempo determinado, el token expira dejando de ser válido. Por el contrario, los refresh tokens no expiran por lo que la aplicación puede utilizar un refresh token para adquirir un nuevo access token y volver a utilizar la aplicación. Estos tokens se deben de mantener privados ya que si alguien los obtiene podría utilizarlos para acceder a los datos privados del usuario.

- **Client ID y client secret:** Se tratan de credenciales que las aplicaciones deben utilizar para autenticarse. Estos strings identifican unívocamente la aplicación y son utilizados para adquirir los tokens antes mencionados.

Como se ha explicado, para implementar en el bot un proceso de autenticación OAuth 2.0 es necesario obtener las credenciales de autenticación

que lo identifiquen unívocamente. Estas credenciales se deben crear desde la consola de Google que se encuentra accesible a través de internet (https://console.developers.google.com/apis/credentials/oauthclient/) En el caso de esta aplicación se deberá crear un ID de cliente de Web. En esta pantalla se rellenan algunos datos como son la Redirect URI, es decir, un endpoint donde el servidor OAuth 2.0 de Google puede enviar respuestas (en este caso, el servidor al que se mandará el access token que es necesario usar en las peticiones a la Fitness Store). Una vez rellenados los datos se podrá descargar un fichero llamado "client_secrets.json" cuyo contenido se muestra en la Fig. 3.2.


```
{
  "web":{
    "client_id":"279516540071-1jgts2nbckb0dh40ti6dp4lcgtml6drl.apps.googleusercontent.com",
    "project_id":"dulcet-name-144919",
    "auth_uri":"https://accounts.google.com/o/oauth2/auth",
    "token_uri":"https://accounts.google.com/o/oauth2/token",
    "auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/v1/certs",
    "client_secret":"g4qM4sXVy437ukLYW9AlCWd3",
    "redirect_uris":[
      "http://mibandbot.myddns.me:8088/"
    ]
  }
}
```


Figura 3.2: Contenido del archivo client_secrets_web.json.

En él se muestran los dos códigos antes mencionados (client ID y client secret) así como otros datos de interés como la auth_uri (dirección en la que se realiza la autenticación), token_uri (dirección a la que se solicitan los tokens de acceso y refresco) o redirect_uri (direcciones a las que se mandará el código para poder utilizar la aplicación). Este fichero se deberá utilizar en el bot cuando se hagan uso de las APIs de autenticación OAuth 2.0, por lo que es importante que esté guardado en un lugar de acceso exclusivo del bot.

El método de autenticación OAuth 2.0 explicado se trata de una autenticación entre 3 entidades que en este caso son el usuario, la aplicación y la Fitness Store. Este tipo de autenticación se utiliza para permitir a la aplicación acceder a los datos del usuario sin conocer sus credenciales de Google. La autenticación sigue el siguiente esquema:

El servidor OAuth de Google responde a la petición de acceso del bot usando la

URL definida como REDIRECT_URI (que en este caso coincide con el endpoint donde se ejecuta el bot). Si el usuario ha aceptado la petición de permisos, la respuesta contiene el código de autorización, en caso contrario contiene un mensaje de error.

Los ejemplos de las dos posibles respuestas son:

- Una respuesta de error:
  - https://mibandbot.myddns.me/auth?error=access_denied
- Una respuesta con código de autorización:
  - https://mibandbot.myddns.me/auth?code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7

Una vez se complete la operación de autenticación OAuth 2.0, el usuario podrá ver la aplicación aceptada en su cuenta de Google como se muestra en la Fig. 3.3.

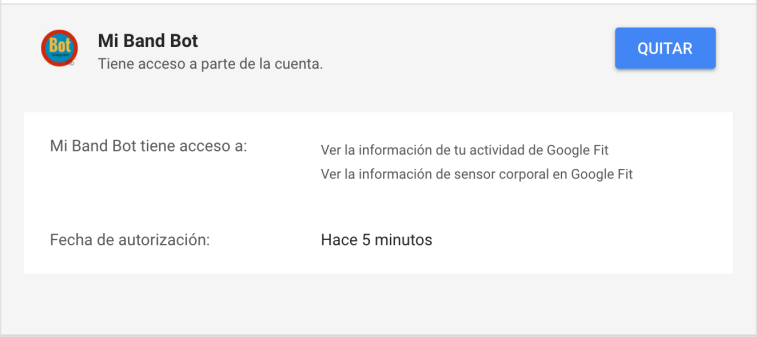

Figura 3.3: Imagen de los permisos de la cuenta de Google.

A partir de este momento el bot podrá obtener datos de la Fitness Store del usuario utilizando el access token proporcionado por el servidor OAuth 2.0 de Google.

3.3 Servidor en la nube

En el presente proyecto se ha utilizado una máquina virtual alojada en el servicio de cloud computing de Amazon, Amazon Web Service (AWS). El

sistema operativo utilizado es una instancia de Linux llamada Amazon Linux Ami 2016.03.8. De las diferentes opciones ofrecidas por el servicio se optó por la más básica al tratarse de una micro instancia gratuita. De esta elección se derivarán algunos problemas de rendimiento debido a los escasos recursos que proporciona esta instancia gratuita que se comentarán posteriormente.

La administración de la máquina virtual se realiza por medio del protocolo de acceso a máquinas remotas a través de la red, SSH (Secure Shell), y del protocolo de transferencia de ficheros, FTP. Para ambos protocolos es necesario utilizar un certificado del cliente para realizar la autenticación por lo que se deberá generar previamente (mediante la aplicación de Shell openssl por ejemplo).

Al tratarse de una arquitectura basada en microservicios, en el servidor conviven los siguientes elementos: un servidor HTTP, el stack ELK, un proxy reverso (Nginx) y el bot de Telegram si bien estos elementos se podrían distribuir en diferentes máquinas (ya que las comunicaciones entre los distintos módulos se realizan por medio de la red) obteniendo uno de los beneficios de la arquitectura basada en microservicios al eliminar el punto de fallo único.

3.3.1 Dispatcher (ServerHandler.py)

Se trata de un servidor HTTP programado también en Python. Este servidor se encuentra permanentemente esperando peticiones HTTP con el código de acceso procedentes de la Fitness Store, es decir, se ha de alojar en el servidor que se configure como REDIRECT_URI. Una vez recibido, reenvía dicho código al bot de Telegram y así el bot podrá finalizar el proceso de autenticación OAuth.

3.3.2 Stack ELK

Como ya se ha comentado en capítulos anteriores, está formado los las aplicaciones ElasticSearch, Logstash y Kibana y permite tanto almacenar la información procedente del usuario así como su posterior monitorización. En este capítulo se va explicar cómo se han aplicado estas herramientas a la arquitectura diseñada:

3.3.2.1 Logstash

Se trata de un formateador de logs. Los datos que el bot transmite en forma de cadena de texto son procesados por 3 filtros que producen un JSON que se envía a ElasticSearch. Los archivos de configuración de estos tres filtros son:

- **02-tcp-input.conf**: filtro de entrada. Recibe los datos mediante un filtro TCP en el puerto 5044.

  · /etc/logstash/conf.d/02-tcp-input.conf

```
input {
  tcp {
    port => 5044
    type => "MiBBot"
  }
}
```

Contenido del filtro de entrada de Logstash.

- **10-filter.conf:** filtro de procesamiento del texto. Este filtro se encarga de procesar la cadena de texto que envía el bot de Telegram y lo formatea en un JSON con el nombre del campo y tipo de archivo correcto. Este es el filtro que realiza la adaptación de los datos obtenidos por la pulsera al recurso *Observation*. Se realiza un mapeo de los diferentes datos enviados por el bot (startTimestamp, endTimestamp, duration, value, chat_id, fitness_type) a campos del recurso estándar FIHR para observaciones. La asociación de los datos, basado en el recurso "Observation", es la siguiente:

  - **status:** indica el estado de la observación (Registered, Preliminary, Final, Amended +). En todos los casos se mapea el valor *FINAL* ya que se trata de actividades finalizadas.

- **code:** identifica el tipo de observación. Este campo se mepea con el tipo de dato recogido (pasos, calorías o actividad) en el campo *fitness_service* por el bot.
- **subject:** identifica a quién o a qué corresponde la observación. Este campo se mapea con el *chat_id* del usuario de telegram enviado por el bot. Así se establece una relación de paciente-id de Telegram.
- **efectivePeriodStart:** indica periodo de inicio de una observación. Este campo se mapea con el timestamp de inicio del intervalo del DataSet, el *startTimestamp*.
- **efectivePeriodEnd:** indica periodo de finalización de una observación. Este campo se mapea con el timestamp de finalización del intervalo del DataSet, el *endTimestamp*.
- **valueQuantity:** indica un valor de cantidad de una observación. Este campo se mapea con la cantidad medida de cada tipo de datos recogidos (número de pasos, número de calorías, tipo de actividad), es decir, el *value*.
- **device:** indica el sensor que se ha utilizado para realizar la medida de la observación. En todos los casos se registra el valor *xiaomi_mi_band*.
- **duration:** Se trata de un campo extra (que no aparece en el recurso FHIR) que contiene la duración del segmento del DataSet en minutos.

  · /etc/logstash/conf.d/10-filter.conf

```
filter {
  if [type] == "MiBBot" {
    grok {
      match => { "message" =>
      "%{TIMESTAMP_ISO8601:efectivePeriodStart}
      %{ TIMESTAMP_ISO8601:efectivePeriodEnd}
      %{ NUMBER:duration:int}
      %{ NUMBER:valueQuantity:int}
```

```
        %{DATA:status}
        %{NOTSPACE:subject}
        %{NOTSPACE:device}
        %{NOTSPACE:code} "
        }
      }
      date {
        match => ["efectivePeriodStart",
                  "yyyy-MM-dd HH:mm:ss"]
        target => "efectivePeriodStart"
        timezone => "UTC"
      }
      date {
        match => ["efectivePeriodEnd",
                  "yyyy-MM-dd HH:mm:ss"]
        target => "efectivePeriodEnd"
        timezone => "UTC"
      }
      mutate {
        replace => ["@timestamp",
                    "%{efectivePeriodStart}"]
      }
    }
}
```

Contenido del filtro de formateo de datos de Logstash.

- **30-elasticsearch-output.conf:** filtro de salida. Si el parseo de datos se realiza correctamente (no se produce un _grokparsefailure) envía los datos a ElasticSearch al índice *tbot_test* rellenado en el campo "document_id" la cadena de texto recibida del bot para evitar duplicados.

· /etc/logstash/conf.d/30-elasticsearch-output.conf

```
output {
  file { path => "/usr/lib64/python2.7/site-packages/
  mibandbot/logstash_miband.log"
  }
  if "_grokparsefailure" not in [tags] {
    elasticsearch {
      hosts => ["127.0.0.1:9200"]
      index => "tbot_test"
      document_id => "%{message}"
    }
  }
}
```

Contenido del filtro de salida de Logstash.

3.3.2.2 ElasticSearch

Se trata una base de datos que indexa los datos en forma de documentos JSON. Estos JSON son los proporcionados por el filtro de salida de Logstash.

Este servicio se encuentra ejecutándose en la máquina local (localhost), esperando peticiones en el puerto 9200. La configuración actual constituye un único nodo activo (una única instancia de ElasticSearch), con un único índice (un contenedor ligero para los datos) llamado *tbot_test* que permite la búsqueda de los documentos.

3.3.2.3 Kibana

Se trata de un front end web utilizado para monitorizar los documentos existentes en ElasticSearch. Realiza búsquedas por timestamp para representar los datos en las gráficas por lo que es necesario en el filtro de Logstash (10-filter.conf) rellenar el campo timestamp con el valor de efectivePeriodStart.

Se ha configurado un DashBoard (una interfaz gráfica que permite administrar al usuario varias gráficas al mismo tiempo) con las tres gráficas que la aplicación permite obtener: pasos, calorías y actividad. En el frontal web estas gráficas se forman dinámicamente y se van actualizando periódicamente por lo que es posible elegir distintos periodos de tiempo para realizar las representaciones y posteriormente hacer "zoom" a los datos para ver periodos de tiempo más reducidos.

Este servicio se encuentra corriendo en la máquina local sobre el puerto 5601.

3.3.3 Proxy inverso (Nginx)

Se trata de un servidor web/proxy inverso. Esta herramienta open source es multiplataforma, por lo que corre en sistemas tipo Unix (GNU/Linux, BSD, Solaris, Mac OS X...) y Windows. El funcionamiento de este proxy inverso es recibir todo el tráfico procedente de Internet con destino al servidor y realizar una función NAT mapeando los puertos del exterior del dominio con los puertos internos del mismo. Esto permite hacer un mapeo del puerto 80 del dominio al puerto 5601 de modo que Kibana sea accesible desde el exterior del servidor. Para realizar esta tarea habrá que crear un archivo de configuración (kibana.conf) en la ruta de configuración de nginx con el siguiente contenido:

· /etc/nginx/conf.d/kibana.conf

```
server {
  listen 80;
  server_name 0.0.0.0;
  location / {
    proxy_pass http://localhost:5601;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
```

```
    proxy_cache_bypass $http_upgrade;
  }
}
```

Contenido del archivo de configuración de Nginx para Kibana.

3.3.4 Bot telegram (bot.py)

Se trata del bot de Telegram programado en el lenguaje de programación Python. Es el elemento principal de la arquitectura, el que contiene la lógica de funcionamiento del bot.

El primer paso para crear un bot es obtener un API_TOKEN que identifica unívocamente al bot. Este código se puede obtener fácilmente desde la aplicación de Telegram. Para ello hay iniciar una conversación con @BotFather, que es un Bot creado por Telegram que sirve para crear y administrar nuestros bots. Una vez se le envía el nombre del futuro bot y algunos parámetros más, responderá con el API_TOKEN que se deberá incluir en el script de Python.

Este bot se implementa por medio de un WebHook (también llamados Web Callback o HTTP push API), es decir, una URL HTTPS (con su correspondiente certificado que también habrá que generar) habilitada para recibir notificaciones de eventos por medio de peticiones GET y POST. Cuando se recibe una petición a dicha URL, el WebHook notificará al bot para que ejecute la función requerida. Esto permite evitar la utilización de polling (la otra forma de implementar bots de Telegram) en la que el bot se encuentra permanentemente conectándose a los servidores de Telegram para comprobar si hay nuevas actualizaciones. La utilización de un WebHook configura un bot mucho más eficiente que en los casos de los bots gestionados por polling. Hay una serie de requisitos para recibir actualizaciones vía WebHook:

- Una dirección IP pública o dominio.
- Un certificado SSL (ya que todas las comunicaciones con los servidores de Telegram deben estar encriptadas con HTTPS usando SSL. Con polling los

servidores de Telegram se encargan de ello pero con WebHook el desarrollador ha de encargarse de ello). Hay dos tipos de certificados admitidos: un certificado verificado emitido por una autoridad de certificación de confianza (CA) o un certificado autofirmado. En este proyecto se ha utilizado la segunda opción al tratarse de la opción gratuita.

- Un servidor HTTP que escuche conexiones WebHook. La librería python-telegram-bot incluye un servidor HTTP integrado por lo que sólo es necesario preocuparse de su configuración. Este servidor tiene una limitación ya que sólo soporta 4 puertos para WebHook (80, 88, 443 y 8443) por lo que sólo se podrían lanzar 4 bots por dominio. En el caso del presente proyecto, al necesitar sólo un bot, no hay ningún problema en utilizar este servidor integrado.

En la Fig. 3.4 se muestra el diagrama UML del sofware desarrollado.

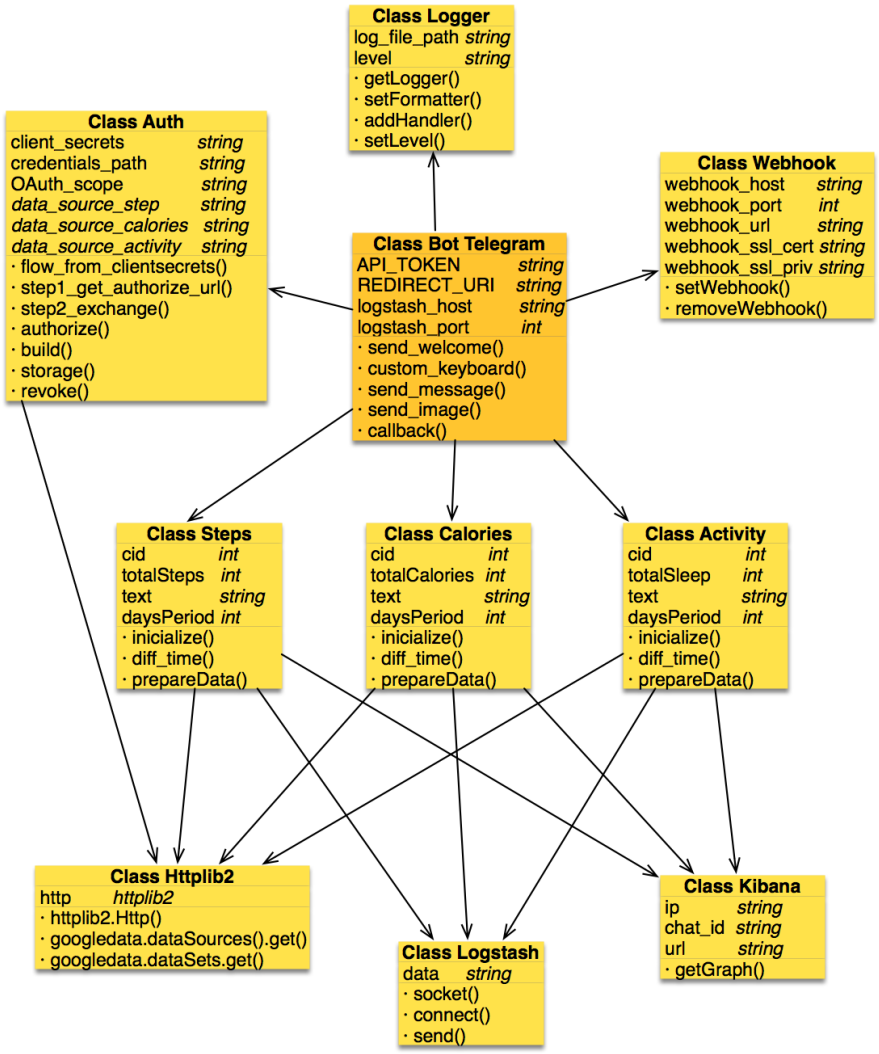

Figura 3.4: Diagrama UML del bot de Telegram.

En el script que simula el comportamiento del bot se pueden encontrar las siguientes funciones:

- **Una función para la autenticación OAuth:** En ella se utilizan las APIs de la librería oauthclient que facilitan el realizar el proceso. A estas APIs es necesario proporcionarles el access token que se recibe a través del servidor HTTP. En esta función también se crea, con la ayuda de las librerías Google-Api-Python-client, un objeto de transporte necesario para realizar las peticiones a la Fitness Store y obtener los DataSets con la infomación del usuario.

- **Una función para establecer el Webhook:** Para establecer el Webhook es necesario establecer la dirección IP, puerto, clave privada, certificado. Sólo puede haber un Webhook corriendo en un determinado puerto al mismo tiempo por lo que hay que asegurarse que no haya uno anterior corriendo a la hora de establecer el Webhook (para ello se establece la función de remover Webhook).

- **Una función que establece un Logger en la aplicación:** En el bot también se ha implementado un sistema de logging que registre todos los acontecimientos de la ejecución del mismo (tanto las interacciones de los usuarios como los posibles errores que se pudieran generar durante su utilización). Esto permite tener constancia del comportamiento del sistema. El fichero en el que se almacena dicha información se encuentra en la siguiente ruta:

  · /var/tmp/MiBand.log

- **Funciones para obtener los pasos, calorías y actividad:** Consisten en 3 funciones que interactúan con la Fitness Store y obtienen los DataSets (en formato JSON) con la información del usuario. Esta información se formatea en texto legible y se le envía al usuario por medio de la aplicación de Telegram.

- **Una función que establece la comunicación con Logstash:** Esta función prepara la cadena de texto que se enviará al filtro de entrada de Logstash. En ella se crea también un socket para enviar la información. Esta información se transmite por medio de paquetes TCP.

- **Una función que genera las imágenes desde Kibana:** Esta función se encarga de obtener las gráficas que se enviarán al usuario. Para ello se ayuda de 3 scripts desarrollados en JavaScript que realizan una tarea de WebScraping (técnica en la que, por medio de programas software, se extrae información de sitios web, en este caso Kibana). Una vez se ha

identificado la gráfica requerida (pasos, calorías o actividad) se ejecuta el JavaScript correspondiente que elimina las otras 2 de las 3 disponibles en el DashBoard. Entonces, ayudándose de la librería phantomJS se realiza un screenshot de la gráfica objetivo y se almacena en el servidor en formato PNG. Posteriormente, esta imagen generada se le envía al usuario junto con valores numéricos que complementan la gráfica.

El directorio que contiene el bot (script Python) tiene la siguiente estructura:

- /usr/lib64/python2.7/site-packages/mibandbot
  - El bot:
    - **bot.py**
  - El servidor HTTP utilizado en la autenticación OAuth:
    - **ServerHandler.py**
  - Tres archivos JavaScript que obtienen las distintas gráficas de Kibana:
    - **PasosPNG.js**
    - **CaloriasPNG.js**
    - **ActividadPNG.js**
  - El archivo con las credenciales de autorización del bot:
    - **client_secrets_web.json**
  - Un archivo de logs que muestra la salida del filtro de Logstash:
    - **logstash_miband.log**
  - Un directorio para almacenar las credenciales de los usuarios:
    - **credentials**
  - Un directorio para almacenar el certificado y clave necesarios para configurar el WebHook:
    - **webhookutiles**

3.4 Servidor local

Como ya se ha comentado, al utilizar una instancia gratuita de AWS, la memoria RAM disponible es limitada por lo que supone un gran hándicap al utilizar el stack ELK. Tanto es así que en las pruebas que se realizaron no fue posible tener activos al mismo tiempo todos los módulos en el servidor. Hubo que tener elementos aislados (el bot, Logstash y Elasticsearch para almacenar los datos en la base de datos y, posteriormente, dejar de ejecutar el bot y Logstash para activar Kibana y comprobar el correcto funcionamiento de la monitorización) por lo que no se pudieron realizar pruebas de su funcionamiento en tiempo real. Por lo tanto, se buscó una alternativa que permitiese ejecutar la arquitectura en tiempo real aunque eso implicase no realizar un procesado en la nube. Se planteó entonces crear un servidor local por medio de una máquina virtual.

Al proponer este nuevo despliegue se consideró interesante que la máquina virtual se auto-configurase con todo lo necesario para ejecutar la arquitectura de modo que con activar la máquina virtual y esperar unos minutos un usuario pudiera interactuar con el bot sin necesidad de estar configurando toda la arquitectura. Esta autoconfiguración se consiguió con la utilización de Vagrant y Ansible.

3.4.1 Vagrant

Vagrant es una herramienta para desarrolladores que facilita la creación de entornos virtuales para desarrollo. En Vagrant se puede instalar y configurar software en una máquina virtual para poder simular el servidor en el que se alojará la arquitectura planteada.

Antes de empezar a utilizar Vagrant es necesario instalar VirtualBox ya que es el entorno de virtualización elegido. Una vez instalado se instalará Vagrant por línea de comandos.

3.4.2 Ansible

Ansible es una herramienta desarrollada en Python que permite una configuración de equipos remotos por medio de un solo fichero llamado playbook

que contiene todas las tareas necesarias que se han de realizar sobre el host.

La instalación de las librerías y dependencias necesarias para el funcionamiento de la arquitectura se realiza mediante un playbook.yml que automatiza este proceso. Este archivo contiene todas las acciones (instalar librerías y dependencias, instalar aplicaciones, modificar archivos de configuración de los programas) que se realizarían en el servidor la primera vez que hubiera que preparar la arquitectura de modo que se podrán desplegar la máquina virtual perfectamente configurada en los equipos que queramos en cuestión de minutos.

Para que esta servidor local sea accesible públicamente hay que realizar una serie de configuraciones que se comentan a continuación:

- En primer lugar hay que establecer el servidor en un dominio fijo ya que las configuraciones que hay que realizar no pueden estar variando continuamente. Normalmente, en los domicilios particulares, no se dispone de una dirección IP fija (ya que supone un coste extra) por lo que hay que realizar una asociación entre esa dirección IP dinámica y un nombre de dominio. La página web https://www.noip.com/ permite establecer un DNS dinámico gratuito que vaya actualizando dinámicamente la asociación entre la dirección IP dinámica del router y un nombre de dominio que nos proporciona dicha página. Así, se podrá utilizar dicho nombre de dominio en las configuraciones necesarias (Redirect_URI, Webhook) y no habrá que preocuparse de modificar la configuración. En la Fig. 3.5 se muestra el nombre de dominio obtenido, http://www.mibandbot.myddns.me.

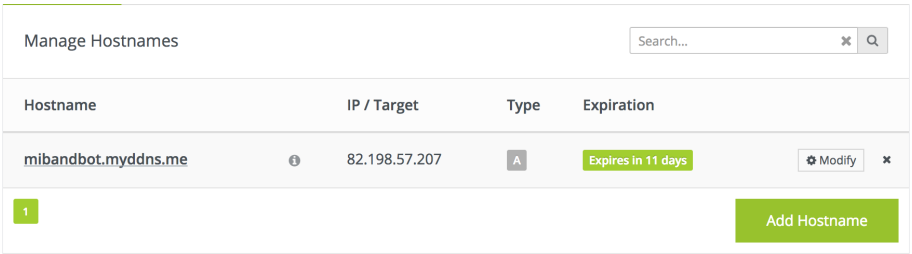

Figura 3.5: Configuración de NoIp.

- Después hay que permitir que las peticiones que lleguen a dicho dominio acaben finalmente en el servidor donde se aloja el núcleo de la arquitectura.

La máquina virtual tiene una dirección IP privada dentro de la red particular por lo que es necesario que esta máquina sea accesible desde el exterior. Para ello hay que configurar la pasarela (router) para que realice una función NAT. Se debe, por lo tanto, realizar un reenvío de puertos para realizar un mapeo entre los puertos externos de la red con los puertos de la IP privada que tiene la máquina virtual. La configuración realizada se muestra en la Fig. 3.6.

Local			Externo				
Dirección IP	Puerto inicial	Puerto final	Puerto inicial	Puerto final	Protocolo	Descripción	Activado
192.168.1.21	80	80	80	80	TCP	HTTP	sí
192.168.1.21	8088	8088	8088	8088	TCP	Servidor HTTP	sí
192.168.1.21	8443	8443	8443	8443	Ambos	Webhook	sí

Figura 3.6: Configuración del router.

- Por último hay que configurar el firewall de la máquina CentOS. Por defecto el firewall está configurado para bloquear cualquier acceso desde el exterior por lo que es necesario permitir los distintos protocolos (HTTP, HTTPS) y puertos (*Kibana(80)*, *ServidorHTTP(8088)* y *Webhook(8443)*) sean accesibles. Esta configuración se realiza mediante el Playbook de Ansible con las siguientes tareas:

```
- name: Start firewalld
  service: name=firewalld
           state=started
- name: Add bridged interface to public zone
  command: firewall-cmd --permanent
           --zone=public --add-interface=eth1
- name: Open http service in firewall
  command: firewall-cmd --permanent
           --zone=public --add-service=http
- name: Open https service in firewall
  command: firewall-cmd --permanent
           --zone=public --add-service=https
```

```
- name: Open 80 port in the firewall
  command: firewall-cmd --permanent
           --zone=public --add-port=80/tcp
- name: Open 8088 port in the firewall
  command: firewall-cmd --permanent
           --zone=public --add-port=8088/tcp
- name: Open 8443 port in the firewall
  command: firewall-cmd --permanent
           --zone=public --add-port=8443/tcp
- name: Load the newest firewall configuration
  command: firewall-cmd --reload
```

Una vez se han realizado todas las configuraciones anteriores, sólo queda levantar el servidor local correctamente configurado mediante Vagrant y Ansible (previamente instalados). Para iniciar la máquina virtual sólo es necesario ejecutar los siguientes comandos por línea de comandos:

- vagrant init hemangajmera/centos7-gnome
- vagrant up

3.5 Flujo de la información

Se han diseñado dos diagramas de flujo. El primero corresponde a la fase de autenticación y autorización mediante la cual el usuario se registra en el sistema y se autoriza a la aplicación para obtener datos de este y, el segundo, muestra el flujo del funcionamiento de la aplicación

3.5.1 Flujo de la fase de autenticación

Esta fase comprende desde que el usuario accede por primera vez al chat bot de la aplicación hasta que se registra correctamente en la arquitectura. Cuando se completa este proceso, el usuario es capaz de solicitar gráficas de su actividad. Este diagrama se puede ver en la Fig. 3.7.

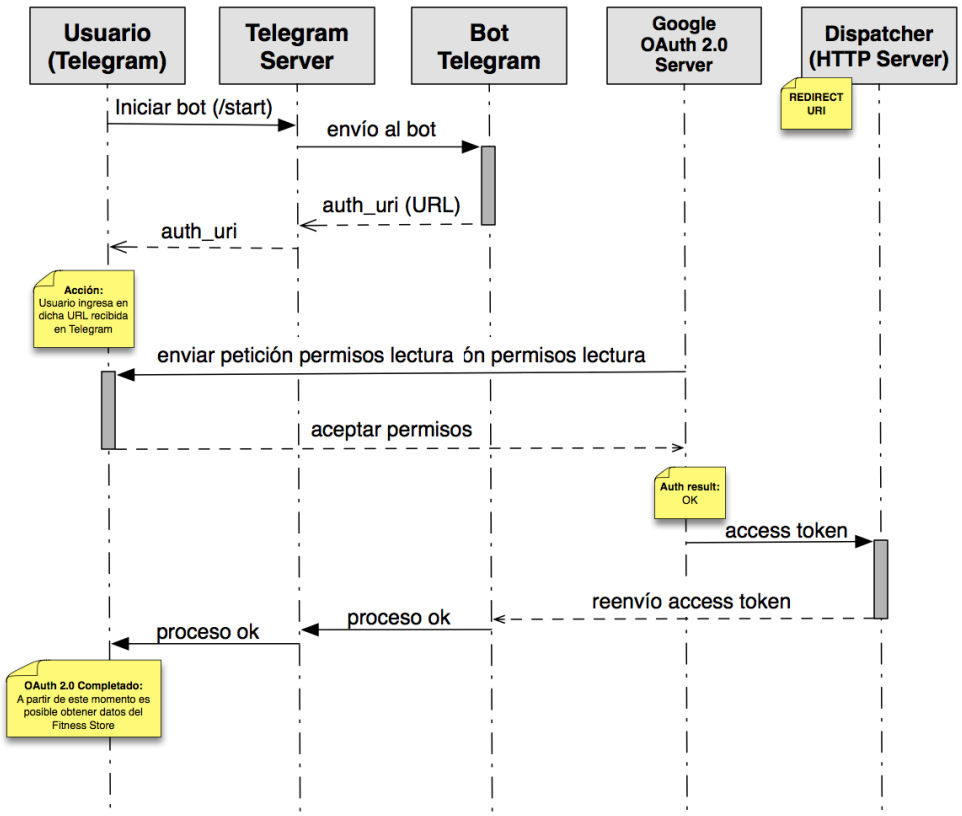

Figura 3.7: Diagrama de flujo de la fase de autenticación y autorización.

3.5.2 Flujo del funcionamiento de la aplicación

El funcionamiento de la aplicación comienza cuando el usuario pulsa sobre uno de los botones del teclado personalizado de la aplicación. El bot recibe la orden y desencadena las acciones necesarias para obtener la gráfica requerida. Una vez la obtiene, se la reenvía al usuario.

El diagrama de flujo de dicho funcionamiento se muestra en la Fig. 3.8.

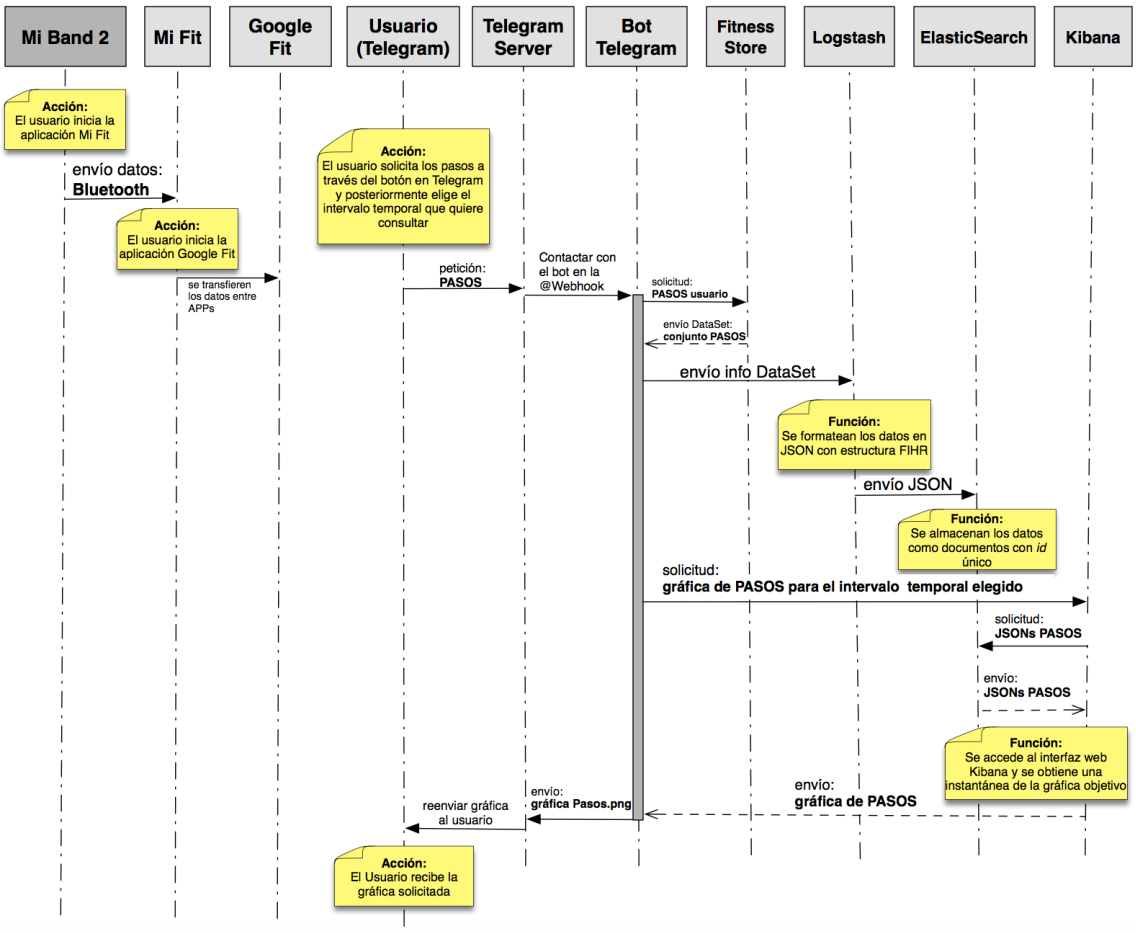

Figura 3.8: Diagrama de flujo del funcionamiento de la aplicación.

Capítulo 4

Resultados obtenidos

En este capítulo se va a explicar el funcionamiento de la aplicación y los resultados obtenidos con el desarrollo del trabajo.

En la Fig. 4.1 se muestra una captura de pantalla de la aplicación Telegram.

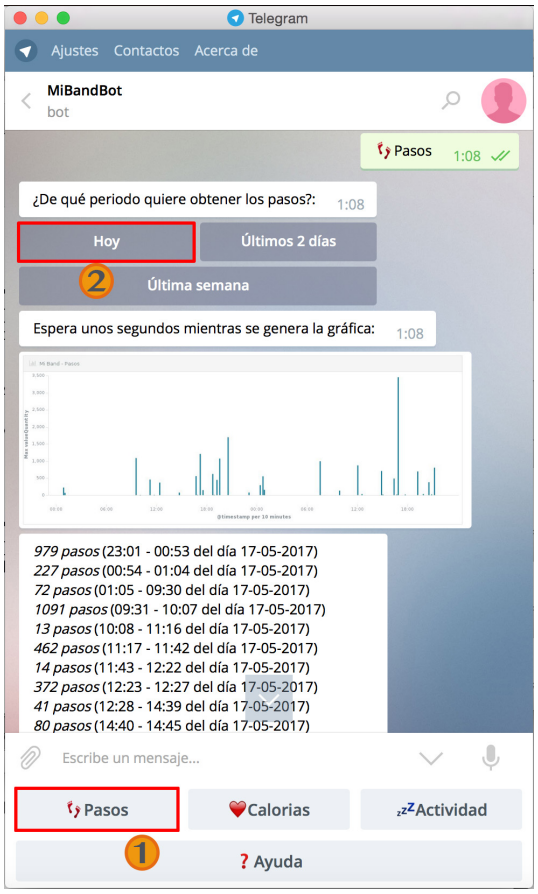

Figura 4.1: Funcionamiento de la aplicación al pedir los pasos del día.

Como ya se ha comentado, la aplicación consiste en un chat bot de Telegram que envía gráficas al usuario de sus registros obtenidos mediante la pulsera fitness. El funcionamiento es sencillo. Una vez el usuario pulsa uno de los botones del teclado personalizado (pasos, por ejemplo) recibe un teclado inline que le muestra los periodos en los que quiere recibir los datos. Tras pulsar uno de estos nuevos botones recibidos (*Hoy*, *Últimos 2 días* o *Última semana*), la aplicación realizará los procedimientos necesarios para devolverle al usuario la gráfica pedida. Esta gráfica se envía en formato PNG y ocupa unos 20 kb por lo que, supone un mínimo gasto en caso de utilizarse mediante una tarifa de datos móviles.

4.1 Pasos

En este tipo de gráfica se indica el cantidad de pasos realizados por el usuario a lo largo del tiempo. El eje de abscisas indica el intervalo temporal en el que se muestran los pasos mientras que el eje de ordenadas indica el número de pasos realizados.

La Fig. 4.2 corresponde a los pasos realizados en el día que se solicitan los pasos.

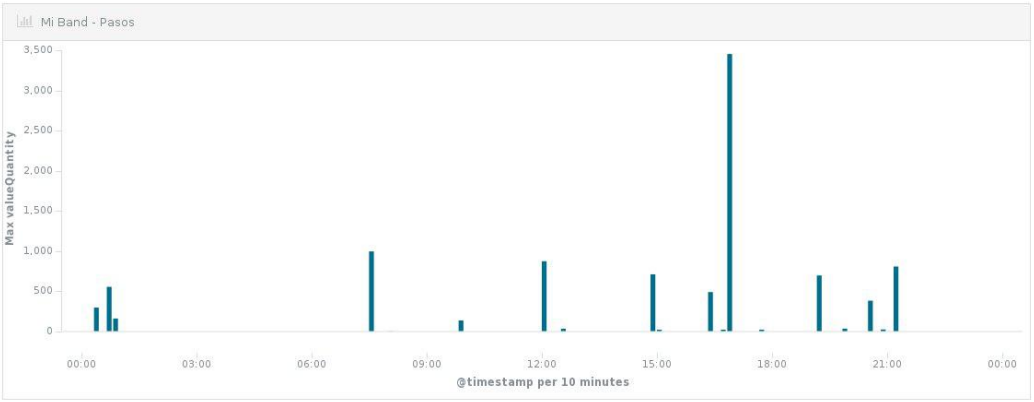

Figura 4.2: Gráfica de los pasos diarios.

La Fig. 4.3 a los pasos realizados en el día que se solicitan los pasos y el día anterior.

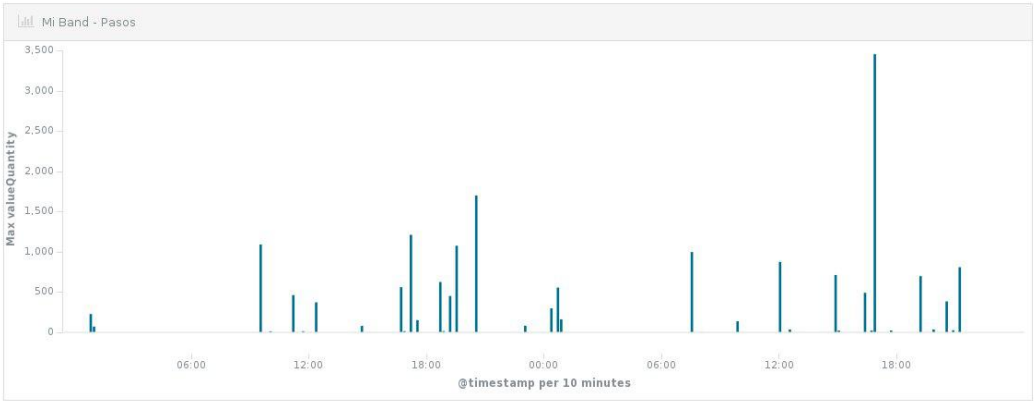

Figura 4.3: Gráfica de los pasos de dos días.

Por último, la Fig. 4.4 corresponde a los pasos realizados durante la última semana desde el momento en el que se solicitan los pasos.

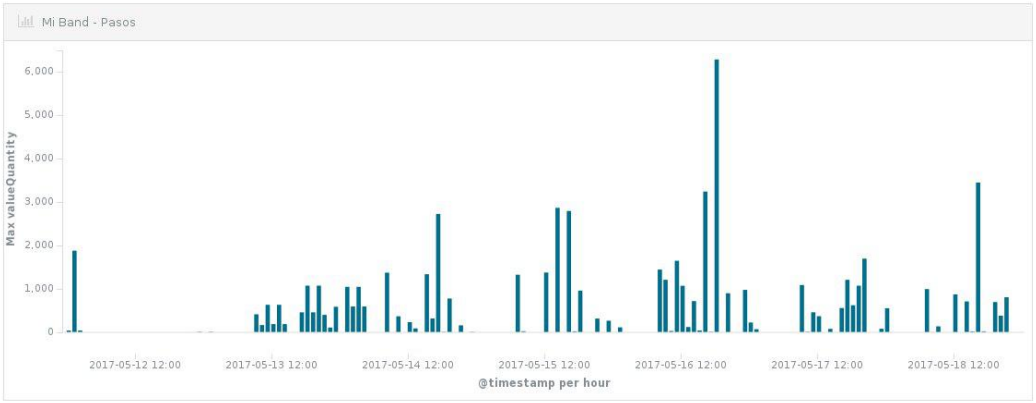

Figura 4.4: Gráfica de los pasos de una semana.

4.2 Calorías

En este tipo de gráfica se indica el cantidad de calorías consumidas por el usuario a lo largo del tiempo. El eje de abscisas indica el intervalo temporal en el que se muestran los pasos mientras que el eje de ordenadas indica el número de calorías consumidas. Se aprecia claramente como estas gráficas tienen gran

similitud con las obtenidas para los pasos ya que las calorías se empiezan a consumir cuando se realiza un número determinado de pasos.

La Fig. 4.5 corresponde a las calorías consumidas durante el día que se solicitan los datos.

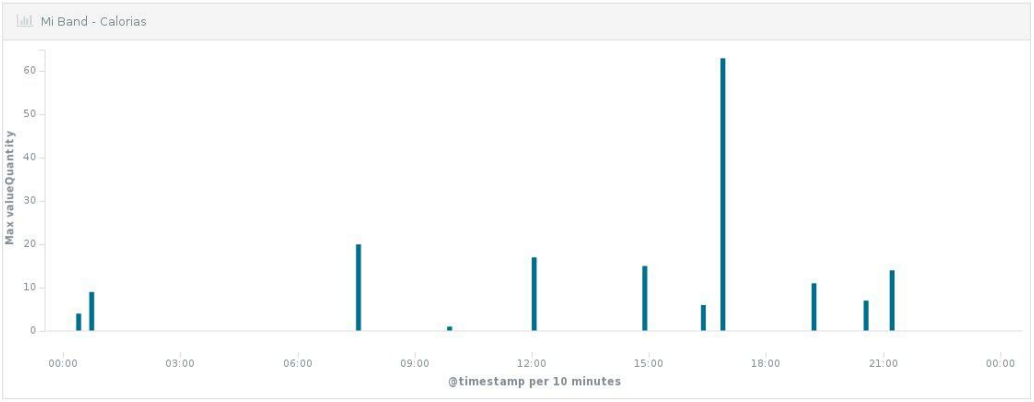

Figura 4.5: Gráfica de las calorías diarias consumidas.

La Fig. 4.6 corresponde a las calorías consumidas en el día que se solicitan los datos y el día anterior.

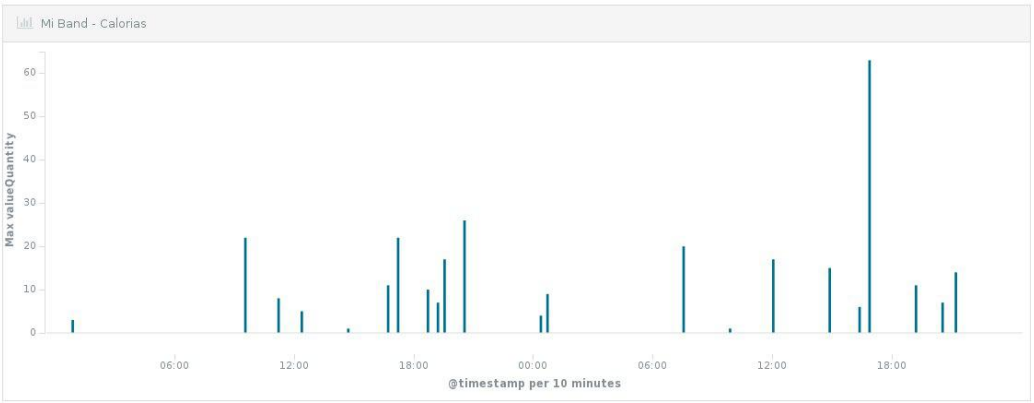

Figura 4.6: Gráfica de las calorías consumidas durante los dos últimos días.

Por último, la Fig. 4.7 corresponde a las calorías consumidas durante la última semana desde el momento en el que se solicitan los datos:

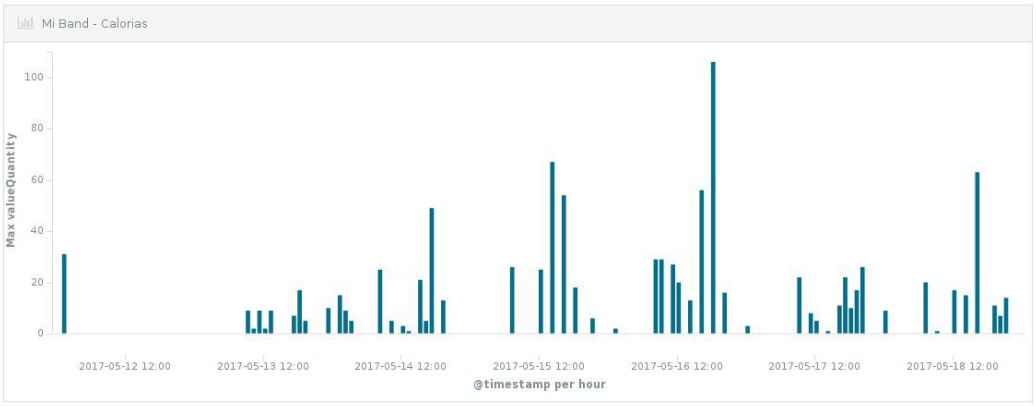

Figura 4.7: Gráfica de las calorías consumidas durante una semana.

4.3 Actividad

Este tipo de gráfica muestra la actividad registrada por el usuario. El eje de abscisas indica el intervalo temporal en el que se muestran la actividad mientras que el eje de ordenadas indica el tipo de actividad realizada correspondiéndo cada actividad con un valor determinado:

- **Despierto:** corresponde a un valor 10 en la gráfica.
- **Andando:** corresponde a un valor 30 en la gráfica.
- **Sueño ligero:** corresponde a un valor 90 en la gráfica.
- **Sueño profundo:** corresponde a un valor 100 en la gráfica.

Estos valores se han elegido para diferenciar los intervalos de *descanso(90,100)* de los intervalos de *reposo(10)* o *paseos(30)*.

Cabe destacar que Kibana, al ser una aplicación destinada al tratamiento de logs (que son eventos puntuales en el tiempo), impone una restricción ya que no se puede prolongar cada muestra un intervalo temporal diferente. El caso más claro se muestra en los intervalos de descanso. Es decir, los intervalos de sueño ligero y

sueño profundo deberían abarcar desde que empieza un intervalo hasta que empieza el siguiente, pero como se aprecia en la gráfica todas las barras verticales tienen la misma duración temporal.

La Fig. 4.8 corresponde a la actividad realizada durante el día que se solicitan los datos:

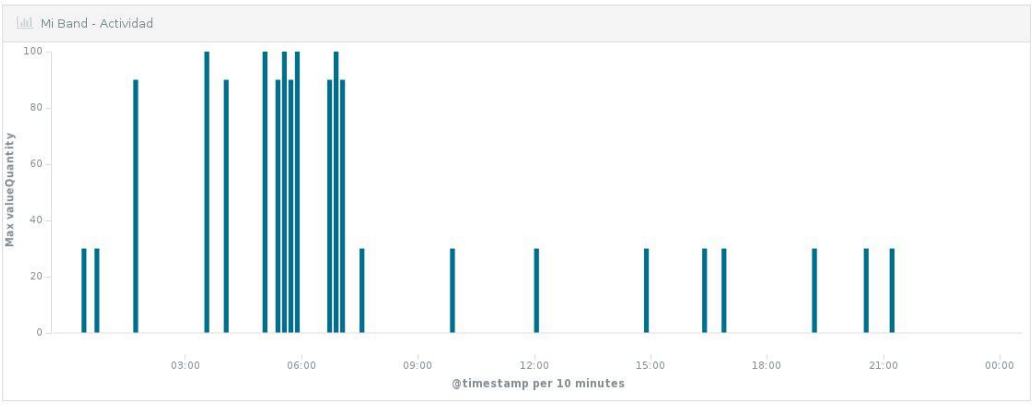

Figura 4.8: Gráfica de las calorías diarias consumidas.

La Fig. 4.9 corresponde a la actividad realizada durante el día que se solicitan los datos y el día anterior:

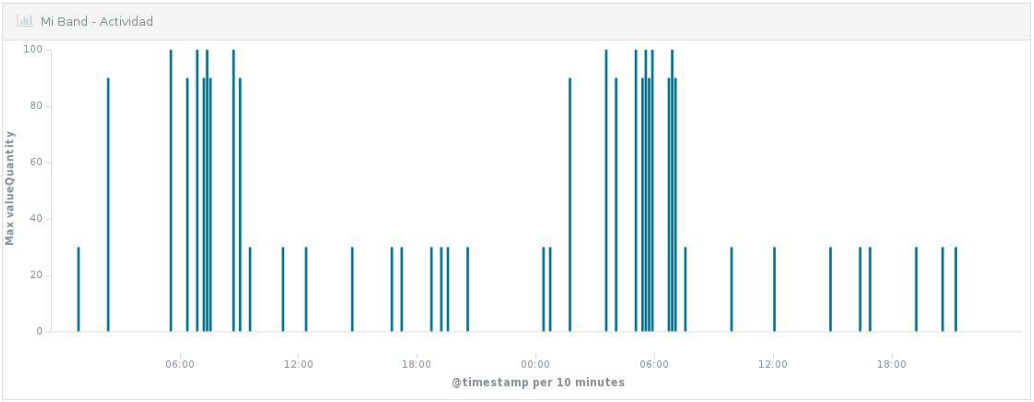

Figura 4.9: Gráfica de las calorías consumidas durante los dos últimos días.

Por último, La Fig. 4.10 corresponde a la actividad realizada durante la última semana desde el momento en el que se solicitan los datos:

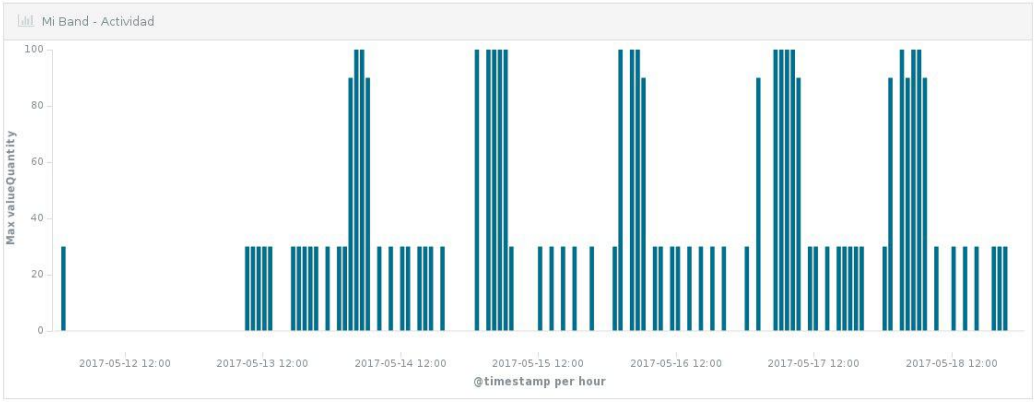

Figura 4.10: Gráfica de las calorías consumidas durante una semana.

4.4 Ayuda

Finalmente, se encuentra el botón *Ayuda*, que envía un mensaje al usuario de información que le permita interpretar las gráficas recibidas.

En la Fig. 4.11 se muestra el funcionamiento de dicho botón.

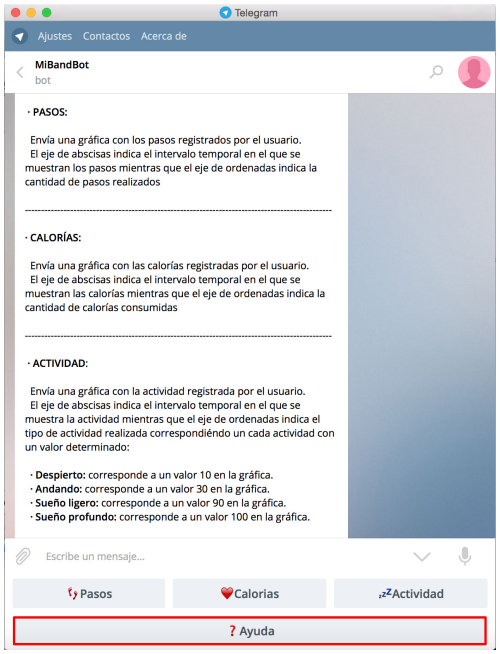

Figura 4.11: Funcionamiento de la aplicación al pulsar el botón ayuda.

Capítulo 5

Conclusiones y líneas futuras

5.1 Conlusiones

En este proyecto se ha desarrollado una arquitectura basada en pulseras inteligentes y bots capaz de mostrar la actividad de los usuarios mediante representaciones gráficas en la aplicación de mensajería instantánea Telegram.

En primer lugar se realizó un estudio de cómo transferir la información desde las pulseras fitness hasta el servidor donde se aloja el bot de Telegram que incluían etapas como la autorización y autenticación, la obtención de datos de la base de datos de Google, el almacenamiento de dicha información.

Adicionalmente se realizó un estudio de los diferentes elementos de la arquitectura que debían intervenir con el objetivo de realizar las implementaciones de elementos que, en principio, no tienen nexo en común. Como resultado se consiguió que la información cuantificada por el sensor del usuario (la pulsera inteligente) llegara al servidor, donde se realiza un procesado de la misma mediante diferentes elementos del servidor y, finalmente, retornara al usuario en forma de representación estática de una gráfica realizada con la aplicación de monitorización Kibana.

Finalmente se creó un sistema de auto-configuración para desplegar el servidor en la máquina local, que se conecta al repositorio Github para obtener algunos de los archivos necesarios, de modo que la puesta a punto de la arquitectura fuera un

sistema automático que no requiriera la participación del usuario (a excepción de la introducción de algunos secretos que identifican unívocamente al bot, como por ejemplo el API_TOKEN). Otros archivos de configuración, al tratarse de archivos de credenciales de usuario, no se alojaron en dicho repositorio online por motivos de seguridad sino que se proporcionan junto a los archivos que configuran el servidor.

Como resultado se ha obtenido una plataforma funcional que permite a los usuarios realizar un seguimiento de su actividad diaria. El algoritmo permite una gestión en tiempo real varios usuarios de modo que les permita obtener dicha información en pocos segundos. El sistema desarrollado representa una herramienta útil que complementa a las aplicaciones móviles ya existentes y fácilmente ampliable gracias a las continuas funcionalidades de las que Telegram va dotando a los bots.

5.2 Líneas de futuro

El sistema desarrollado se enmarca tanto en el área de la configuración de arquitecturas de red como en el área de la seguridad informática por los métodos empleados para securizarla (sistema autenticación y autorización, utilización de certificados digitales, configuración de firewalls...).

Como primera línea de futuro se podría investigar si es posible realizar un método alternativo de envío de información de las pulseras inteligentes hasta la arquitectura ya que, a día de hoy, es un proceso que el usuario debe realizar manualmente (conectándose tanto a MiFit como a Google Fit para que los datos obtenidos por la pulsera se registren en la base de datos de Fitness Store). A día de hoy esto supone una gran limitación ya que para solicitar datos actualizados es necesario conectarse a las dos aplicaciones móviles y esperar a que estén disponibles en la Fitness Store. Sin embargo, con los valores históricos (última semana) no hay problema.

Basándose en la primera línea, también se podría plantear el desarrollo de una aplicación móvil que realizara la obtención automática de datos de la pulsara y enviara los datos directamente a la Fitness Store, de modo que no fueran necesarias

las aplicaciones utilizadas (Mi Fit y Google Fit). Esto permitiría evitar al usuario la tarea de enviar los datos al servidor para centrarse únicamente en obtenerlos e interpretarlos.

También sería interesante ampliar la funcionalidad del bot añadiendo algunas características, como por ejemplo, la inclusión de un sistema de alarmas en función de unos umbrales predefinidos que avisaran al usuario (de que no ha andado lo suficiente durante el día, de que no ha descansado lo suficiente) o un sistema que permitiera compartir esta información con sus familiares o amigos creando una especia de micro red-social.

Una alarma sería que no ha andado lo suficiente durante el día o mandarle a sus familiares la actividad que ha tenido durante el día, que ha sido alta/baja etc.

Por otro lado se podría implementar cada módulo de la arquitectura en un equipo diferente para obtener todos los beneficios de la arquitectura basada en microservicios.

Por último, para complementar la idea anterior, se podría integrar en la arquitectura algún módulo orientado a e-Health de modo que estos datos de los usuarios pudieran acabar siendo gestionados por un médico que pudiera, de esta forma, realizar un seguimiento personalizado de sus pacientes.

Bibliografía

[1] Newman, S., *Building microservices.* 1ª ed. Sebastopol: O' Reilly Media Inc., Febrero 2015

[2] *Google Fit.* Actualizada: 2 Junio 2016. [Fecha de consulta: Septiembre 2016]. Disponible en: https://developers.google.com/fit/overview

[3] Wang, F., *Telegram Bot API.* Actualizada: 2 Junio 2017. [Fecha de consulta: Octubre 2016].
Disponible en: https://github.com/eternnoir/pyTelegramBotAPI

[4] HL7, Affiliates & FIHR Foundation., *FIHR.* Actualizada: 19 Abril 2017. [Fecha de consulta: Enero 2017].
Disponible en: https://www.hl7.org/fhir/observation.html

[5] *ElasticSearch.* Actualizada: 2017. [Fecha de consulta: Noviembre 2016]. Disponible en: https://www.elastic.co/

[6] Hashi Corp., *Vagrant.* Actualizada: 2017. [Fecha de consulta: Mayo 2017]. Disponible en: https://www.vagrantup.com/

[7] Red Hat Inc., *Ansible.* Actualizada: 2017. [Fecha de consulta: Mayo 2017]. Disponible en: http://docs.ansible.com/ansible/index.html

[8] Hidayat, A., *PhantomJS.* Actualizada: 2017. [Fecha de consulta: Abril 2017].
Disponible en: http://phantomjs.org/documentation/