Universidad Zaragoza
1542

Trabajo Fin de Máster

Diseño e implementación de una interfaz gráfica para la gestión de un PHR basado en FHIR utilizando ELK

Design and implementation of a graphic interface to the management of a PHR based on FHIR data using ELK

Autor

Manuel Hernández Rodrigo

Directores

Álvaro Alesanco Iglesias
Jorge Sancho Larraz

Escuela de ingeniería y arquitectura

2018

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. MANUEL HERNÁNDEZ RODRIGO,

con nº de DNI 72892454A 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) Diseño e implementación de una interfaz gráfica para la gestión de un PHR basado en FHIR utilizando ELK,

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

Zaragoza, 20/06/2018

Fdo: MANUEL HERNÁNDEZ RODRIGO

Agradecimientos

En primer lugar me gustaría agradecer el apoyo de mis directores **Jorge** *y* **Álvaro** *por sus explicaciones, consejos y correcciones, que han hecho posible la redacción de este trabajo.*

A mis padres y mi hermana, quienes me han apoyado en todo momento y han hecho posible que haya llegado hasta aquí.

A todos mis compañeros de universidad con los que he compartido esta experiencia durante estos seis años.

A mis compañeros de laboratorio por los buenos momentos juntos y ayudarme a finalizar este trabajo.

A todos, muchas gracias.

Diseño e implementación de una interfaz gráfica para la gestión de un PHR basado en FHIR utilizando ELK

RESUMEN

Actualmente se está produciendo un cambio en el paradigma de la prestación de los servicios sanitarios en todo el mundo debido al desarrollo de las Tecnologías de la Información y Comunicaciones (TIC) en este ámbito. El abaratamiento en los costes de los sensores médicos y el uso de wearables ha incrementado la cantidad de información médica disponible en cada paciente.

En este caso, desde un punto de vista de red, cada uno de los pacientes podría considerarse un nodo de red generando información y el médico sería el administrador de la misma. Dada esta simetría y el gran número de herramientas existentes para la gestión, manipulación y representación de la información recolectada de los nodos de red, sería interesante analizar su funcionamiento en un escenario médico.

Para ello se parte de una aplicación basada en plataformas de mensajería para la recolección de datos médicos de un grupo de pacientes. Esta aplicación se ha extendido para recoger los datos de actividad de Google Fit.

Se ha diseñado una arquitectura que permita integrar las arquitecturas de gestión de red y los pacientes utilizando para ello una herramienta de gestión de red, ELK, para el análisis y la representación de la información y un estándar médico, FHIR, para el intercambio de la información de salud. Esta arquitectura tiene como objetivo crear una carpeta personal de salud (PHR), realizando la visualización de los datos mediante la herramienta Kibana de ELK, en la cual se han definido los dashboards que representan la información médica.

Dada la sensibilidad de la información médica recolectada y que ELK no proporciona un control de acceso, se ha implementado una arquitectura que mediante el protocolo OAuth garantice el acceso autenticado a los datos. Esto se ha desarrollado implementando un control de acceso a la aplicación utilizando Python y la librería Flask. Este control de acceso garantiza la autenticación del usuario, el acceso únicamente a los recursos a los que tiene permiso y define las operaciones que puede realizar.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1 Visión general

Las Tecnologías de la Información y Comunicaciones (TIC) están impulsando un gran desarrollo en el entorno de la prestación de servicios sanitarios, produciendo un incremento en la calidad de atención al paciente. Esto se debe tanto al desarrollo de aplicaciones y dispositivos móviles dentro del campo de la salud, como al desarrollo de infraestructuras de redes de comunicaciones que permiten el intercambio de información sanitaria. Estas aplicaciones buscan mejorar la calidad de los servicios, facilitando una gestión más eficiente y cómoda para el paciente. Destacan aplicaciones relacionadas con la historia clínica del paciente [1] o con recordatorios de medicamentos y citas [2], ya que facilitan el acceso a información de enfermedades y dan la posibilidad de realizar consultas a médicos y profesionales. Sin embargo, aparecen retos para su desarrollo como la interoperabilidad entre los distintos agentes implicados y la estandarización, la usabilidad de los sistemas o la garantía de seguridad y privacidad de la información [3].

1.2 Objetivos

Debido a el aumento de la información médica de la que se dispone causado por abaratamiento en los costes de los sensores médicos y el uso de *wearables*, se

plantea un escenario en el que podría considerarse a cada uno de los pacientes, desde un punto de vista de red, como un nodo de red generando información y al médico como el administrador de la misma, utilizando para ello herramientas de gestión de red ya existentes para manejar la información recolectada. El uso de estas herramientas existentes permitirá un desarrollo rápido a la par que dotar de un valor añadido a la información médica.

Mediante el uso de estas herramientas se plantea el desarrollo de una interfaz gráfica para la gestión de una carpeta personal de salud (PHR). Un PHR es una herramienta de gestión y archivo de información de salud que es mantenida por el paciente. En especial se centrará en el caso de enfermedades crónicas para permitir al usuario una mayor comodidad y facilidad en el acceso a su información médica, así como en permitir el acceso a médicos y profesionales cuando el usuario lo autorice, garantizando la seguridad y la privacidad de la información. Para ello, se parte de una aplicación basada en plataformas de mensajería para la recolección de datos médicos de un grupo de pacientes.

Se plantean además una serie de objetivos específicos para el funcionamiento del sistema:

- Análisis de los diferentes estándares para el intercambio de información médica y como se adaptan a nuestras necesidades.
- Análisis de las diferentes plataformas existentes de gestión de red y como se adaptan a nuestras a nuestras necesidades.
- Diseño de una arquitectura que nos permita la recogida de información de los pacientes y su inclusión en el sistema.
- Inclusión de datos de actividad en el sistema.
- Elaboración de las visualizaciones necesarias para representar toda la información relevante del PHR.
- Adecuación de las herramientas de gestión de red seleccionadas para cumplir con los requisitos de seguridad y privacidad requeridos por la información médica.

1.3 Materiales y herramientas utilizadas

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

Hardware:

- **Ordenador:** Ordenador personal con capacidad para utilizar los programas de la pila ELK y donde se ha desarrollado el código necesario.

Software:

- **Debian:** Sistema operativo sobre el que se ha realizado el desarrollo. Es una distribución de Linux gratuita.
- **Python:** Lenguaje de programación interpretado y de código abierto. Utilizado como lenguaje de programación en los servidores Flask.
  - **Flask:** Es una *microframework* para Python.
  - **Requests:** Librería para generar peticiones HTTP.
  - **JSON:** Librería para codificar y decodificar objetos JSON (JavaScript Object Notation).
- **Elastic Stack:** Compuesto por varios proyectos de código libre:
  - **Elasticsearch:** Motor de búsqueda y análisis.
  - **Logstash:** Servidor de procesado de datos.
  - **Kibana:** Herramienta para visualizar los datos almacenados en Elasticsearch mediante gráficos.
  - **Beats:** Permite enviar datos a ELK.
- **VirtualBox:** Software para la creación de entornos de virtualización.
- **Synthea Patient Generator:** Utilizado para generar pacientes ficticios FHIR.

- **Ruby:** Lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. Se utilizará para la configuración de Logstash.

- **Servidor FHIR:** Servidor donde se almacena la información médica siguiendo el estándar FHIR.

- **Servidor de Autenticación:** Servidor utilizado para autenticar a los usuarios.

- **eHealthzApp:** *Bot* donde se generan los registros FHIR y de uso de los usuarios.

1.4 Organización de la memoria

La memoria consta de 5 capítulos distribuidos de la siguiente manera:

- **Capítulo 1 - Introducción:** En este capítulo se realiza una breve descripción del trabajo realizado.

- **Capítulo 2 - Estado del arte:** En este capítulo se incluye un estudio de las herramientas de gestión de red y los formatos y estándares para el intercambio de información médica.

- **Capítulo 3 - Arquitectura y desarrollo del sistema:** En este capítulo se realiza una descripción de los bloques del sistema y su implementación.

- **Capítulo 4 - Dashboards:** Este capítulo contiene los *dashboards* generados en Kibana.

- **Capítulo 5 - Conclusiones y líneas futuras:** Este capítulo contiene las conclusiones extraídas tras la realización de este trabajo y describe brevemente las líneas futuras a seguir.

También se han añadido los siguientes anexos:

- **Anexo A. Acrónimos**
- **Anexo B. Interfaz Restful FHIR**
- **Anexo C. Recursos FHIR utilizados**
- **Anexo D. Interfaz Restful Elasticsearch**
- **Anexo E. Dispatcher log**

Capítulo 2

Estado del arte

2.1 Estándares para el intercambio de información médica

Uno de los objetivos fundamentales de la informática médica es mejorar la comunicación entre diversas áreas involucradas en la atención de la salud. Para ello propone el desarrollo y la aplicación de estándares que provean de un lenguaje común a todos los sistemas y que permita el intercambio de información entre sistemas de historial clínico electrónico (EHR).

Existen dos grupos de estándares para el intercambio de información sanitaria, los estándares sintácticos, encargados de garantizar que la información pueda ser enviada y recibida y los estándares semánticos para garantizar el significado de los términos empleados.

Estándares sintácticos

- **HL7:** Comprende un conjunto de estándares para el intercambio, integración, compartición y recuperación de información clínica. Los más importantes son la versión dos y tres. La versión dos del estándar permite el envío de una serie de mensajes para intercambiar solicitudes e informes relativos a pruebas clínicas y tratamientos. Con las nuevas actualizaciones fue introduciendo elementos adicionales que le otorgaron una gran flexibilidad. HL7 v3 utiliza

ya una metodología bien definida basada en modelos de información de referencia (RIM). Este modelo facilita la definición de los objetos y especifica la gramática de los mensajes [4].

- **CDA:** Es la aplicación más utilizada de HL7 v3 y ofrece un formato XML estándar para la representación de datos clínicos. Define tres niveles de documentos; el nivel 1 incluye los metadatos básicos y un cuerpo con texto o imágenes, el nivel 2 en el que el cuerpo puede ser un elemento sin estructura o un número cualquiera de secciones con un bloque de texto y el nivel 3 que permite incluir una estructura de datos en las secciones anteriores [5].

- **FHIR:** Este estándar combina las mejores características de HL7 v2, HL7 v3 y CDA para el intercambio de información clínica , utilizando estándares WEB modernos (XML, JSON, RESTful, OAuth, etc.) centrándose en una implementación y desarrollo sencillo.

- **DICOM:** Es un estándar reconocido para el intercambio de estudios imagenológicos, mediante un formato de ficheros y un protocolo TCP/IP para la comunicación entre sistemas [6].

Estándares semánticos

- **SNOMED-CT.** Es una colección semánticamente organizada de términos médicos que proporciona los códigos, los términos y definiciones utilizadas en la documentación clínica. Se considera la terminología de salud clínica más completa [4].

- **LOINC:** Es una base de datos y el estándar universal para la identificación de los exámenes de laboratorio [7].

Para el desarrollo del proyecto se ha decidido utilizar el estándar FHIR puesto que permite el intercambio de información clínica, necesario para realizar una carpeta personal de salud, por su diseño, ya que permite ser fácilmente implementado y por utilizar documentos tipo JSON con un protocolo REST

con el cual se puede trabajar con las herramientas de gestión de red. Toda la información sobre FHIR puede consultarse de forma gratuita en el siguiente directorio: https://www.hl7.org/fhir/ [8].

Se basa en un conjunto de componentes modulares llamadas "Recursos". Estos recursos pueden ser fácilmente introducidos en sistemas de trabajo para permitir resolver problemas clínicos y administrativos reales. Actúan como representaciones de conceptos del mundo sanitario: paciente, médico, observación... Por ejemplo, el recurso "Patient" contiene la información personal de cada paciente, el recurso "Condition", información sobre las enfermedades que padece un paciente y el recurso "Observation" incluye valores de mediciones tomadas por el paciente. Los Recursos FHIR utilizados se encuentran explicados en el Anexo B con mayor detalle.

FHIR proporciona una API REST para manipular los recursos con un conjunto de interacciones, para este proyecto se utilizan las interacciones Read y Search que permiten consultar los recursos disponibles y realizar búsquedas en ellos. El estilo de las interacciones es el siguiente:

VERB [base]/[type]/[id]

Donde "VERB" indica el verbo HTTP (GET, POST, PUT o DELETE), "base" la URL base del servicio, "type" el nombre del tipo de recurso FHIR e "id" el identificador lógico del recurso. El protocolo REST se encuentra explicado con mayor detalle en el Anexo C.

2.2 Herramientas de gestión de red

Existe una alta variedad de herramientas enfocadas a análisis de logs como Splunk, Grafana, Graphite o Elastic Stack, cada una ofreciendo una serie de características diferentes. Este tipo de software provee de herramientas para incrementar el análisis de *logs* y de bases de datos centralizadas donde almacenar

los datos. Estas características de análisis permiten ayudar a detectar, predecir y prevenir anomalías.

- **Splunk:** Es una herramienta que cuenta con una presencia líder en el mercado. Ha creado una gran experiencia de usuario y una gran cantidad de plugins" para mejorar su plataforma. Tiene una poderosa interfaz de búsqueda para visualizar, explorar y analizar datos de diferentes fuentes que permite realizar líneas de regresión temporales de los datos y fijar umbrales para alertas. Splunk no es código abierto y para su uso dispone de un sistema de pago basado en la cantidad de datos indexados [9].

- **Grafana:** Es una herramienta de código abierto para realizar dashboards que puede trabajar con datos de una gran cantidad de fuentes como Graphite o Elasticsearch. Es una herramienta pensada para representar series temporales basadas en medidas como el uso de la CPU o de los puertos. Ofrece acceso a los dashboards basado en roles de usuario que permiten otorgar permisos diferentes a cada grupo de usuarios. No ofrece ninguna plataforma para el almacenamiento de los datos [10]].

- **Graphite:** Fue lanzado en 2006, ofrece funciones de almacenamiento de datos, almacenando series temporales de datos y capacidad para representar cualquier gráfico. Tiene una herramienta, Graphite-web para la representación de visualizaciones. Es de código libre y tiene gran integración con otro software para la representación de datos como Grafana [11].

- **ELK:** Elastic Stack es un software de código abierto que dispone de las herramientas para el envío, parseado, almacenamiento y representación de datos de múltiples fuentes y formatos. Posee una interfaz para realizar búsquedas sobre los datos y una gran comunidad de usuarios y desarrolladores [12].

Se ha optado por utilizar la solución de Elastic Stack puesto que ofrece una solución gratuita de código abierto a diferencia de Splunk, con las herramientas necesarias para tratar los datos desde su creación hasta su visualización final. Por

su facilidad para trabajar con datos tipo JSON como los relativos a FHIR se ha escogido Elastic frente a Grafana.

Elastic comprende cuatro proyectos de código libre: Elasticsearch, Logstash, Beats y Kibana. Cada uno de ellos cumple con una función diferente dentro de Elastic.

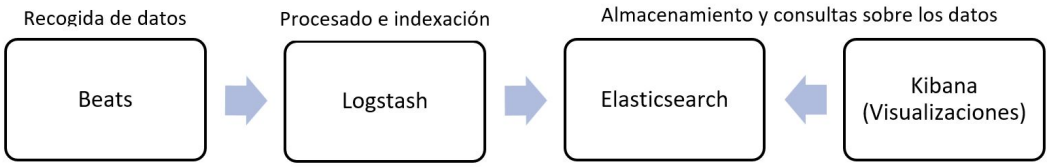

Figura 2.1: Estructura de Elastic Stack.

Beats tiene como único objetivo enviar datos a Logstash o Elasticsearch. Es un programa de fácil instalación y configuración, ligero y utiliza pocos recursos. La información sobre Beats se encuentra disponible gratuitamente en su documentación [13]. Esta plataforma se instala en el equipo donde se encuentran los datos y se encarga de su envío a Elastic. Con sus diferentes productos puede recoger múltiples tipos de datos y estadísticas de los sistemas como: tráfico de red, archivos de log, estadísticas de los sistemas y servicios como uso de la CPU y memoria, eventos generados en Windows o disponibilidad de servicios.

Logstash es un motor para procesado de datos de código libre con capacidad para trabajar en paralelo con múltiples fuentes de entrada. Logstash tiene capacidad para procesar cualquier tipo de evento y transformarlo con sus diferentes extensiones. La figura 2.2 muestra la arquitectura de Logstash que comprende tres etapas, recepción de datos en el *plugin* de entrada, filtrado intermedio y envío de los datos a Elasticsearch en el *plugin* de salida. Se basa en el lenguaje de programación Ruby y soporta una serie de códecs y filtros. Los códecs convierten un formato de entrada en un formato aceptado a la salida. Los filtros son acciones que se utilizan para procesar los eventos y permiten modificarlos o eliminarlos. Algunos filtros son: "grok" que permite parsear cualquier dato sin estructurar, "ruby" para ejecutar código ruby sobre los eventos, "mutate" que permite realizar modificaciones en

los campos como renombrar o eliminar y "date" utilizado para parsear fechas y utilizarlas como la fecha del evento. La información sobre Logstash se encuentra disponible gratuitamente en su documentación [14].

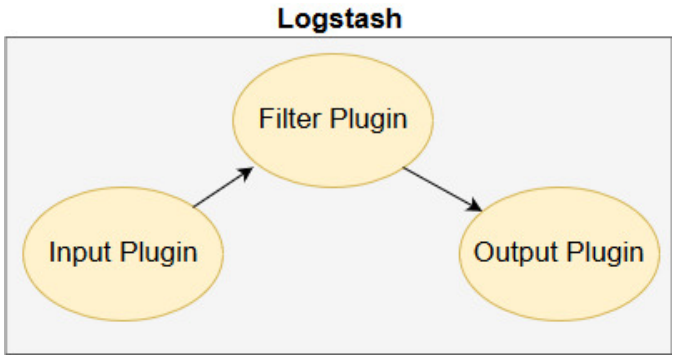

Figura 2.2: Arquitectura de Logstash.

Elasticsearch es un motor de búsqueda de alta escalabilidad de código libre. Permite almacenar, buscar y analizar grandes volúmenes de datos rápidamente y en tiempo casi real (NRT). La unidad básica de información en Elasticsearch es el documento, este documento se expresa en lenguaje JSON. Se basa en una API RESTful para interactuar con los datos y un sistema de analítica para acompañar la evolución de los datos. Para la transferencia de información Elasticsearch utiliza un protocolo RESTful basado en peticiones HTTP y documentos JSON en el que tiene definidas una serie de APIs con las operaciones que se pueden realizar. Este protocolo está explicado con mayor profundidad en el Anexo D. Elasticsearch utiliza un lenguaje Query DSL (Domain Specific Language) para realizar las consultas a los documentos. La información sobre Elasticsearch se encuentra disponible gratuitamente en su documentación [15].

Kibana es una plataforma de visualización y análisis de código abierto diseñada para trabajar con Elasticsearch. Se puede utilizar para buscar, ver e interactuar con los datos almacenados en índices de Elasticsearch. Ofrece análisis avanzado de datos y visualización en una variedad de gráficos, tablas y mapas. Estos gráficos son interactivos e intuitivos y permiten extraer información de manera sencilla para usuarios sin formación en el sector de las TIC. Ha sido

creado para trabajar con grandes volúmenes de datos y permitir de forma rápida crear *dashboards* dinámicos capaces de representar los cambios en Elasticsearch a tiempo real. Kibana ofrece una interfaz para navegadores web con un panel de navegación simple de utilizar. Permite interactivamente explorar todos los campos de cada documento con la posibilidad de introducir filtrado o realizar búsquedas. La información sobre Kibana se encuentra disponible gratuitamente en su documentación [16].

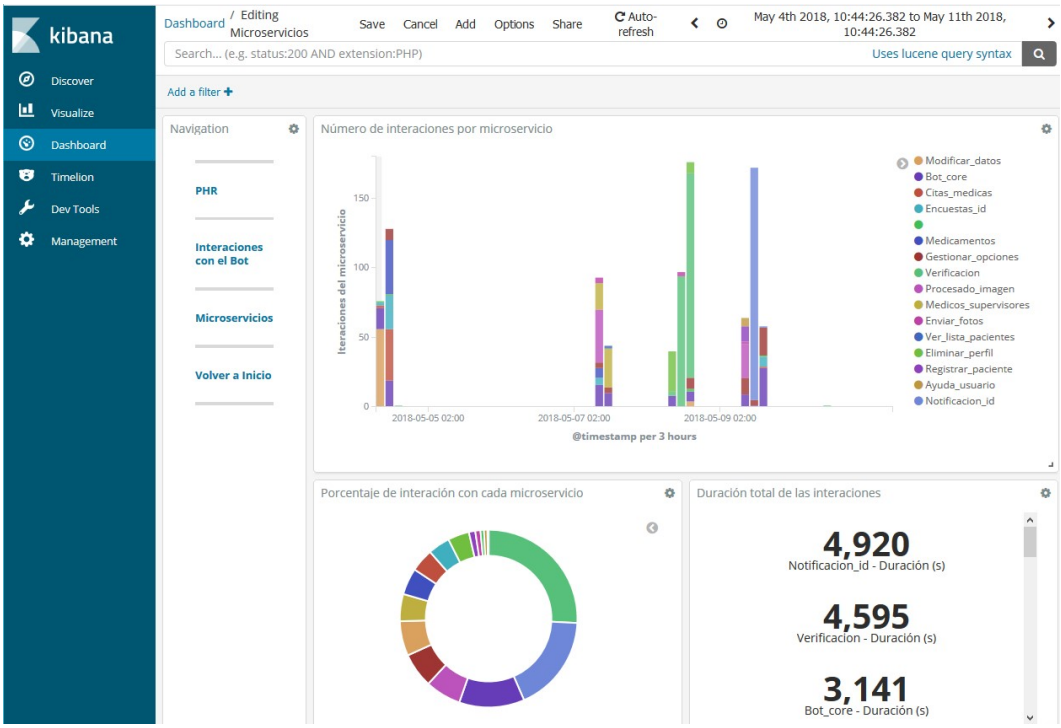

Figura 2.3: Ejemplo de *dashboard* en Kibana.

Capítulo 3

Arquitectura y desarrollo del sistema

Para conseguir ajustar el sistema a los objetivos deseados se ha desarrollado una arquitectura propia, la cual se muestra en la figura 3.1. En ella se pueden ver todos los módulos que serán explicados con mayor grado de detalle.

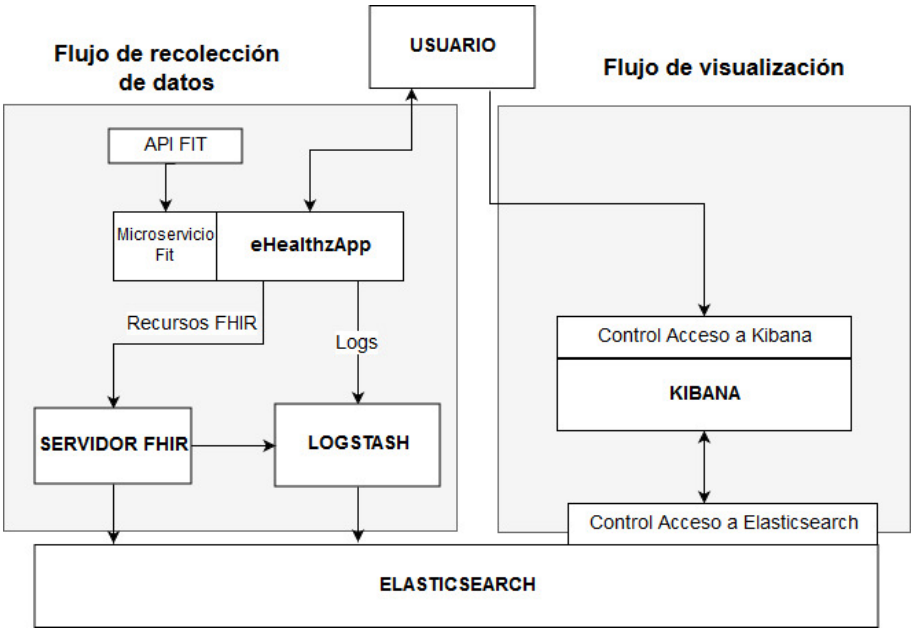

Figura 3.1: Arquitectura del sistema.

- **eHealthzApp:** Aplicación formada por un conjunto de microservicios ya existentes con los que interactúa el usuario mediante un *Bot*, recoge su

información y crea recursos FHIR con ella. Durante su funcionamiento, genera *logs* sobre sus estadísticas de uso .

- **Microservicio Fit:** Microservicio desarrollado en este TFM encargado de recibir los datos de los servidores de Fit tras la autorización del usuario y realiza la adecuación de estos al estándar FHIR para almacenarlos en su servidor.

- **Servidor FHIR:** Servidor encargado de recibir y almacenar los recursos FHIR garantizando su correcto formato.

- **Logstash:** Componente encargado de preprocesar los datos antes de introducirlos a Elasticsearch.

- **Elasticsearch:** Elemento encargado de almacenar e indexar los datos y realizar las búsquedas y peticiones a estos.

- **Control de acceso a Kibana:** Módulo desarrollado en este proyecto al que los usuarios realizan una conexión cuando quieren acceder al sistema. Se encarga de garantizar la autenticidad del usuario y permitir o no su acceso.

- **Kibana**: Aplicación encargada de proporcionar la interfaz gráfica de los datos mediante *dashboards*.

- **Control de acceso a Elasticsearch:** Módulo desarrollado en este proyecto situado entre las peticiones de Kibana a Elasticsearch. Se utiliza para autorizar el acceso a los recursos.

3.1 Flujo de recolección de datos

3.1.1 eHealthzApp

Aplicación formada por un conjunto de microservicios ya existentes cada uno encargado de realizar un proceso. En ella el usuario interacciona con los microservicios enviando información para generar recursos FHIR y como

consecuencia se crean *logs* del uso. La figura 3.2 muestra la arquitectura de la aplicación.

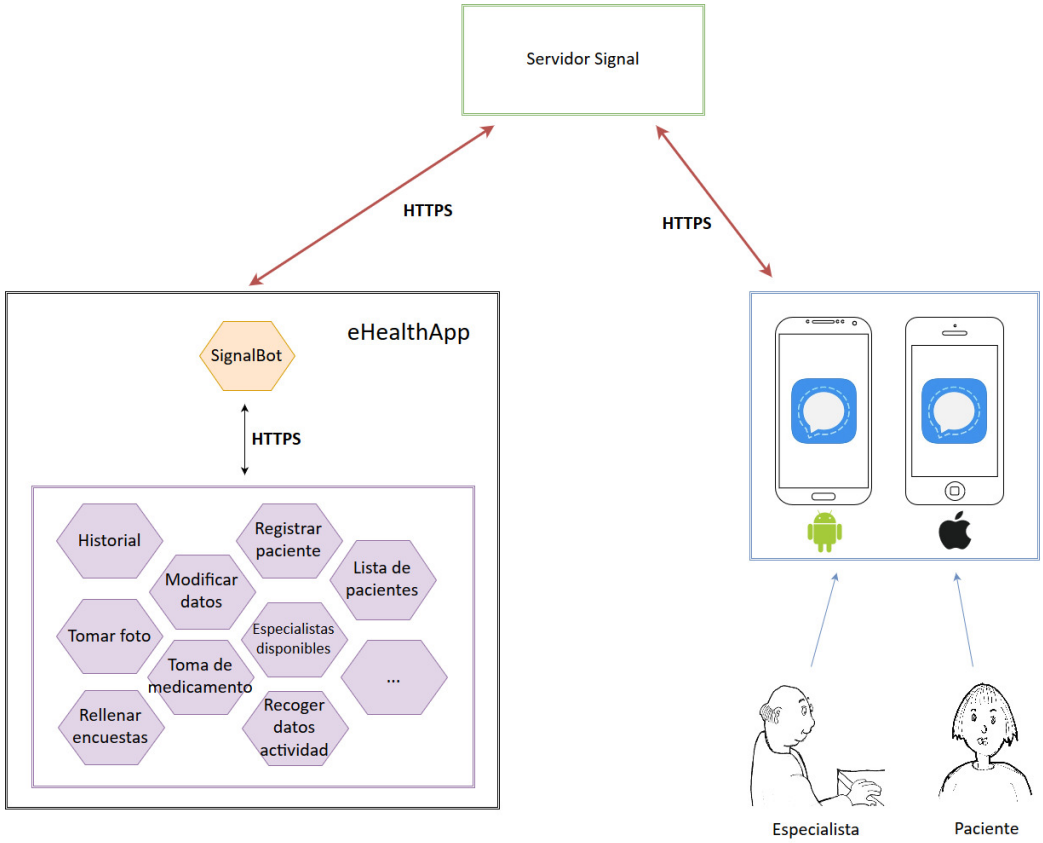

Figura 3.2: Arquitectura de la aplicación eHealthz.

El funcionamiento de la aplicación se basa en la interacción de los pacientes y especialistas con un *Bot* mediante la plataforma de mensajería Signal instalada en sus móviles. Los usuarios se comunican con un *Bot* de Signal que les permite realizar una serie de operaciones definidas mediante los microservicios, como registrar a un paciente o rellenar una encuesta. Mediante estos mensajes el usuario introduce su información médica en la aplicación, que se encarga de recogerla y a partir ella generar recursos médicos del estándar FHIR. Estos recursos médicos se envían al servidor FHIR para su almacenamiento.

Como consecuencia de la interación del usuario con el *bot* se crean *logs* del uso de la aplicación. Estos *logs* se envían a Logstash para introducirlos en la arquitectura de Elastic y trabajar con ellos.

3.1.2 Microservicio Fit

Este módulo se ha desarrollado como un microservicio más de la aplicación eHealthzApp. Obtiene el consentimiento del usuario para acceder a sus datos de actividad de Google Fit, los descarga y formatea en el formato médico FHIR y los envía al servidor FHIR para su almacenamiento.

Para poder recopilar los datos del usuario en la plataforma externa necesita el consentimiento del usuario para acceder a estos. Para ello utiliza Open Authorization (OAuth) mediante el cual un usuario es redirigido a un servidor externo, en este caso de Google, donde se identifica y concede acceso a la aplicación para acceder a sus datos de *fitness*. Una vez se ha autenticado en el servidor externo el usuario es redirigido a nuestra aplicación con un código de autenticación. El microservicio lo comprueba con el servidor externo generando unas credenciales y un *token* de autenticación que es utilizado para conectarse al servidor externo y descargar los datos.

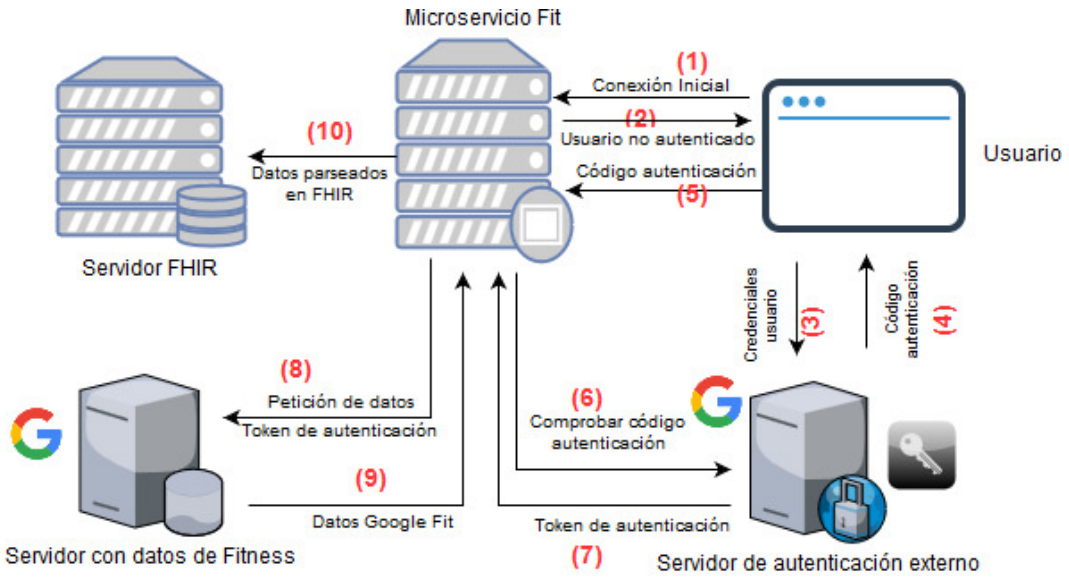

Figura 3.3: Proceso de autenticación OAuth.

Cuando se ha conseguido la autorización del usuario, el microservicio procede a descargar los datos del servidor y parsearlos a un formato FHIR. Los datos extraídos se agrupan temporalmente por franjas de quince minutos. Se recopila información sobre el tiempo de actividad e inactividad, distinguiendo entre el tipo

de actividad, los pasos andados y la distancia recorrida, la posición geográfica en que se encuentra con los valores de latitud y longitud, la frecuencia cardiaca y las calorías consumidas. En caso de no disponer de alguno de estos campos, el valor se encuentra vacío.

Con esta información se genera un documento JSON que cumpla el estándar FHIR utilizando para ello el recurso "Observation". En el que se incluyen los valores indicados anteriormente dentro del campo "component" indicando los códigos relativos a cada una de las medidas anteriores según los estándares SNOMED y LOINC. También se incluye la información relativa al paciente en el campo "subject" y la fecha en el campo "effectiveDateTime". Los recursos generados se envían al servidor FHIR como el resto de recursos con información médica generados en los microservicios.

```
{
  "resourceType": "Observation",
  "id": "30686",
  "code": {
    "coding": [
      {
        "system": "http://snomed.info",
        "code": "719333006",
        "display": "Fitness for activity"
      }
    ],
    "text": "Registro de Google Fit"
  },
  "subject": {
    "reference": "Patient/416"
  },
  "effectiveDateTime": "2018-06-11T09:30:35+01:00",
  "component": [
    {
      "code": {
        "coding": [
          {
            "system": "http://snomed.info",
            "code": "165263003",
            "display": "Walking distance"
          }
        ]
      },
      "valueQuantity": {
        "value": 3412.95930052,
        "unit": "m"
      }
    },
    {
      "code": {
        "coding": [
          {
            "system": "http://snomed.info",
            "code": "68130003",
            "display": "Physical activity"
                    .
                    .
                    .
```


Figura 3.4: Ejemplo de recurso de *Fitness* parseado a FHIR

3.1.3 Servidor FHIR

Este servidor desplegado bajo la librería HAPI-FHIR consiste en una implementación de código abierto de la especificación FHIR [17]. Recibe los recursos generados por los microservicios y los almacena cumpliendo el estándar FHIR. Proporciona una interfaz Rest que cumple con el estándar FHIR (recursos y operaciones) y utiliza Elasticsearch como *backend* para el almacenamiento. Todos los recursos FHIR son almacenados en Elasticsearch y dependiendo del tipo de recurso se envía previamente a Logstash para generar recursos adicionales que faciliten la representación de la información o se trabaja directamente con el fichero JSON en Elasticsearch.

3.1.4 Logstash

El elemento Logstash realiza un preprocesado de los datos antes de introducirlos a Elasticsearch. Recibe datos provenientes de los *logs* de los microservicios o médicos de FHIR y los envía a Elasticsearch. Este preprocesado varía en función del tipo de datos que se utilice como parámetro, diferenciando entre datos FHIR y los *logs* generados por los microservicios.

- **Procesado Log de los microservicios:** Logstash procesa todos los mensajes generados a partir del log de los microservicios de la misma manera, ya que todos siguen el mismo formato. Cada vez que se escribe una nueva entrada en el log se genera un mensaje enviado a Logstash mediante la herramienta de ELK Filebeats. El formato detallado de los campos se encuentra en el anexo E. Para el procesado de este tipo de *logs* se utiliza el filtro "Grok" que permite parsear ficheros con un formato propio a partir de expresiones regulares. Estos ficheros se almacenan bajo el índice de Elasticsearch "dispatcher-YYYY.MM.dd" dónde YYYY.MM.dd indica la fecha en la que se generó.

```
180245796 [Grizzly(11)] 2017-11-30 13:12:05,772 INFO
main.java.resource.Resources - Origen: +34619173338 Destino: Chatbot
Funcionalidad: Ayuda Message: Ayuda Attachment: false
```

Figura 3.5: Ejemplo de registro del "Dispatcher".

Una vez se han obtenido todos los campos del mensaje se realiza una agregación del tráfico proveniente del mismo usuario y generado por el mismo microservicio para poder analizar el tiempo de uso real de cada microservicio por flujo de datos. Se considera que el tráfico corresponde al mismo flujo cuando un nuevo mensaje es generado con una diferencia inferior a cinco minutos respecto al anterior mensaje. Al superarse este tiempo de espera se genera un nuevo fichero con el número de interacciones de ese flujo de datos y la duración entre el último y el primer mensaje, así como el usuario que lo generó, el microservicio y su fecha. Estos ficheros se almacenan bajo el índice de Elasticsearch "flow-YYYY.MM.dd" dónde YYYY.MM.dd indica la fecha en la que se generó.

```
filter {
# Case Microservicies logs
  if ([fields][log_type] == "dispatcher") {
    grok {
      patterns_dir => ["/etc/logstash/patterns/*"]
        match => { "message" => "^%{NUMBER:running_time}
                \[%{CADENA:process}\] %{TIMESTAMP_ISO8601:timestamp}
                %{USERNAME:type} +%{CADENA:service} \- \Origen[:| ]
                +%{ORIGEN:origen} \Destino[:| ]+%{DESTINO:destino}
                Funcionalidad[:| ]+%{FUNCIONALIDAD:functionality}
                Message[:| ]+%{MESSAGE:mensaje} Attachment[:| ]
                +%{ATTACHMENT:attachment}"}
    }
```

Figura 3.6: Filtro "Grok" del los logs de Microservicios .

- **Procesado Recursos FHIR:** Los datos pertenecientes a FHIR se reciben directamente en formato JSON, por lo que su introducción a Elasticsearch es directa, pero se han modificado algunos recursos para ajustarlos a las

representaciones deseadas, introduciendo campos y modificándolos cuando ha sido preciso. Cada tipo de recurso viene marcado en el campo "ResourceType", este valor se ha utilizado para realizar distintos procesados en función del tipo de datos y para la indexación de los ficheros utilizando el valor de este campo y la fecha relativa al fichero: "resourceType-YYYY.MM.dd". Este índice se utiliza en Elasticsearch para diferenciar entre los distintos tipos de documentos. El recurso "Observation" recibe un procesado diferente para poder representar en Kibana observaciones que contengan múltiples valores de muestras de datos como puede ser una señal de electrocardiograma. Para su posterior representación en Kibana se decide crear un nuevo recurso con cada medida disponible dentro de la observación que no se podría representar en Kibana si pertenece al mismo recurso. Para ello se divide el recurso en tantos documentos como datos existan y se les añade una marca temporal, variando esta si hay datos muestreados dentro del fichero, incrementando su valor por el tiempo de muestreo. Estos datos también mantendrán una referencia al paciente y al recurso "Observation" del que proceden, así como el tipo de medida que representan y se almacenan bajo el índice "data".

3.2 Flujo de visualización de datos

3.2.1 Control acceso a Kibana

Este elemento se introduce para controlar el acceso a Kibana utilizando el protocolo OAuth. Recibe los mensajes del usuario cuando intenta conectar a Kibana y le autoriza el acceso. Se ha desarrollado utilizando el lenguaje de programación Python, la librería flask un framework utilizado para el desarrollo de aplicaciones web [18] y un servidor de autenticación de la aplicación donde se almacenan los credenciales y atributos de cada usuario de la aplicación.

Cuando un usuario intenta acceder a Kibana se conecta primero a este control de acceso. Aquí se comprueba si este usuario está autenticado en el sistema. Esto

se hace mediante una *Cookie* de autorización que debe aparecer en la conexión. En una primera conexión al sistema el usuario no dispondrá de esta *Cookie* que se crea en este servidor y no estará identificado, por ello, utilizando el protocolo OAuth, el usuario es redirigido a el servidor de autenticación de la aplicación. En este servidor el usuario deberá introducir sus credenciales de acceso correctamente para obtener permiso para acceder al sistema. Una vez se ha identificado el usuario, será redirigido al control de acceso de Kibana añadiendo un código en su URL otorgado por el servidor de autenticación que se utiliza para validar al usuario. Este código es intercambiado entre el control de acceso y el servidor de Autenticación y en caso de ser correcto, el servidor de autenticación lo cambiará por un *token* de identidad en formato JWT que contendrá el rol y la identificación del usuario. Si todos los pasos anteriores son correctos el control de acceso validará al usuario creando una *Cookie* de autenticación para diferenciar cada usuario en la que se incluye el rol y el identificador del usuario. Esta *Cookie* se firma con la clave privada del servidor para evitar que pueda ser modificada y se reenvía al usuario que la añadirá a sus siguientes conexiones al sistema. Cuando el usuario cuenta con una *Cookie* válida se le concede el acceso a Kibana.

En el caso de que el código de acceso enviado por el usuario sea incorrecto la autenticación fallará y el usuario será reenviado al servidor de autenticación sin ninguna *Cookie* asignada. Las *cookies* se borran frente a un tiempo de quince minutos de inactividad y únicamente son válidas si no se modifica ningún parámetro de la sesión que las inició para otorgar mayor seguridad al sistema.

Este elemento también evitará que un usuario con rol de paciente o médico pueda realizar cambios en los *dashboards* y visualizaciones de Kibana al negarle los permisos para hacerlo comprobando el contenido de los mensajes HTTP *post, put* y *delete*.

3.2.2 Kibana

En esta arquitectura, Kibana se utiliza para crear una interfaz gráfica de la aplicación de salud y de los *logs* de los microservicios. Para ello se han definido

una serie de *dashboards* con la información seleccionada. Kibana está definido por defecto para trabajar con *logs*, como los creados en los microservicios, por lo que ha sido necesario ajustar su funcionamiento para representar datos FHIR.

La aplicación se ha dividido en dos apartados diferentes, la aplicación de salud y los logs de interacción con los microservicios, a partir de ello se han definido los roles de usuario, siendo "patient" y "practitioner" pertenecientes al apartado de salud y sin acceso a los *logs* y el rol "admin" con acceso únicamente a los *logs*.

Un usuario con el rol "**patient**" se corresponde con el de un paciente de nuestra aplicación de salud. Para este tipo de usuarios se define un *dashboard* de inicio al acceder a Kibana llamado "Información personal". Este tipo de usuarios no tiene permisos para modificar los *dashboards*, visualizaciones ni los índices de Elasticsearch, únicamente puede ver su información médica.

Para los usuarios de rol "**practitioner**", correspondientes con los médicos, se define un *dashboard* diferente de inicio, con la información del médico. Este tipo de usuarios tampoco tiene permisos de modificación. Puede acceder a su información personal y únicamente tiene acceso a los datos de aquellos pacientes que se lo otorguen mediante el campo "generalPractitioner" del recurso "Patient".

Los usuarios de rol "**admin**" son los únicos con permisos para modificar los *dashboards*, las visualizaciones y los índices. Este tipo de usuarios corresponde con los desarrolladores de la aplicación y se ha definido como *dashboard* de inicio un panel con información sobre los *logs* de interacciones y uso de los microservicios.

3.2.3 Control de acceso a Elasticssearch

Dado que Elasticsearch no soporta autorización se ha introducido este elemento para realizar esa tarea. Este elemento recibe los mensajes que envía Kibana a Elasticsearch y garantiza que usuario únicamente pueda acceder a aquella información a la que tiene permiso. Se ha desarrollado utilizando el lenguaje de programación Python y la librería flask.

Cuando Kibana realiza una petición de datos a Elasticsearch manda también la *Cookie* de autenticación que recibe del usuario que quiere acceder a esos datos

creada en el control de acceso a Kibana. El control de acceso extrae el valor del rol del usuario de la *Cookie* y tiene definida una lista con los índices a los que puede acceder cada rol. Si el usuario no puede acceder a esos datos se devuelve un mensaje HTTP con código 403 indicando que no tiene permisos para acceder a ese índice. En caso de que el usuario tenga permisos se extrae el valor del identificador de usuario de la *Cookie*. Para un usuario de rol "patient" todas sus peticiones se deben filtrar utilizando el valor de su identificador, por tanto, los resultados tienen que contener este valor para cumplir la petición. Si el rol corresponde a un médico, el filtrado es más complejo. Se debe asegurar que solo tiene acceso a su lista de pacientes filtrando mediante el campo "generalPractitioner" del recurso "Patient", el cual debe corresponder con el valor del identificador del médico. Para acceder a sus datos personales se filtra utilizando su identificador de referencia, esto será así siempre que acceda al recurso "Practitioner". También debe garantizar que en el caso de acceder a los datos de uno de sus pacientes se filtra por el identificador de ese paciente. Esto se realiza mediante un campo de la *Cookie* que contiene el identificador del paciente al que quiere acceder. Para el rol "admin" el filtrado no es necesario puesto que puede acceder a todos los datos de los índices de uso de la aplicación eHealthz, solo es necesario verificar los índices a los que accede.

El filtrado se hace añadiendo el campo "match phrase" al apartado de la petición "bool.must" indicando el campo del índice por el que se quiere filtrar y el valor que debe cumplir. La figura 3.7 resalta las modificaciones a la petición realizadas en Flask para el filtrado de los índices "patient" cuando la petición la realiza un usuario con un identificador "0cd4a310-5fc0-4b80-83af-26467d6b3abd".

Este filtrado se puede realizar para todos los recursos relacionados con un paciente ya que todos ellos cuentan con el campo "subject.reference" que se corresponde con el identificador del recurso paciente y su valor se encuentra almacenado en la *Cookie* de autenticación.

```
GET _msearch/patient*
{
  "index": ["patient*"],
  "ignore_unavailable": true,
  "preference": 1527756689307
}
{
  "sort": [{"_score": {"order": "desc"}}],
                    .
                    .
                    .
  "docvalue_fields": ["@timestamp", "address.period.start", "birthDat
e", "contact.address.period.start", "deceasedDateTime", "identifier
.period.start"],
  "version": true,
  "query": {
    "bool": {
      "filter": [],
      "should": [],
      "must_not": [],
      "must": [
        {
          "match_all": {}
        },
        {
          "match_phrase":{
            "id":{
              "query":"0cd4a310-5fc0-4b80-83af-26467d6b3abd"
            }
          }
        }
      ]
    }
  },
  "size": 500
}
```

Figura 3.7: *Query* del recurso “Patient” con filtro añadido por campo “id”.

Capítulo 4

Dashboards

En este capítulo se muestran los *dashboards* creados en la aplicación Kibana donde se representarán los datos. La aplicación se divide en dos partes según los datos representados. Quedan definidos los *dashboards* de la aplicación de salud con información médica extraída de los ficheros FHIR y los *dashboards* de los *logs* de uso con información sobre las interacciones de los usuarios y los microservicios.

Los *dashboards* se han definido a consecuencia de la información recopilada mediante la aplicación eHealthz, por tanto, se ha representado la información personal del paciente o especialista, la toma de medicamentos, las encuestas que han rellenado, la actividad realizada... Estos *dashboards* se han centrado especialmente en el caso de las enfermedades crónicas con las que se trabaja en la aplicación, diabetes y psoriasis, creando un *dashboard* con información específica de cada una.

4.1 Dashboards de la aplicación de Salud

Para la aplicación de salud se definen una serie de *dashboards* a los cuales se accede mediante la barra de navegación.

- **Información personal Médico:** En este *dashboard* se incluye la información personal del médico, una fotografía y una lista con los pacientes a los que tiene acceso. Esta lista con los pacientes incluye un enlace al *dashboard* de información personal de cada uno de sus pacientes, cuando un médico pincha en ese enlace se le añade a su *Cookie* de autorización la información de referencia de ese paciente para poder tener acceso a sus datos. Esa información se elimina de la *Cookie* cuando, mediante la barra de navegación, el médico selecciona "Volver a Inicio".

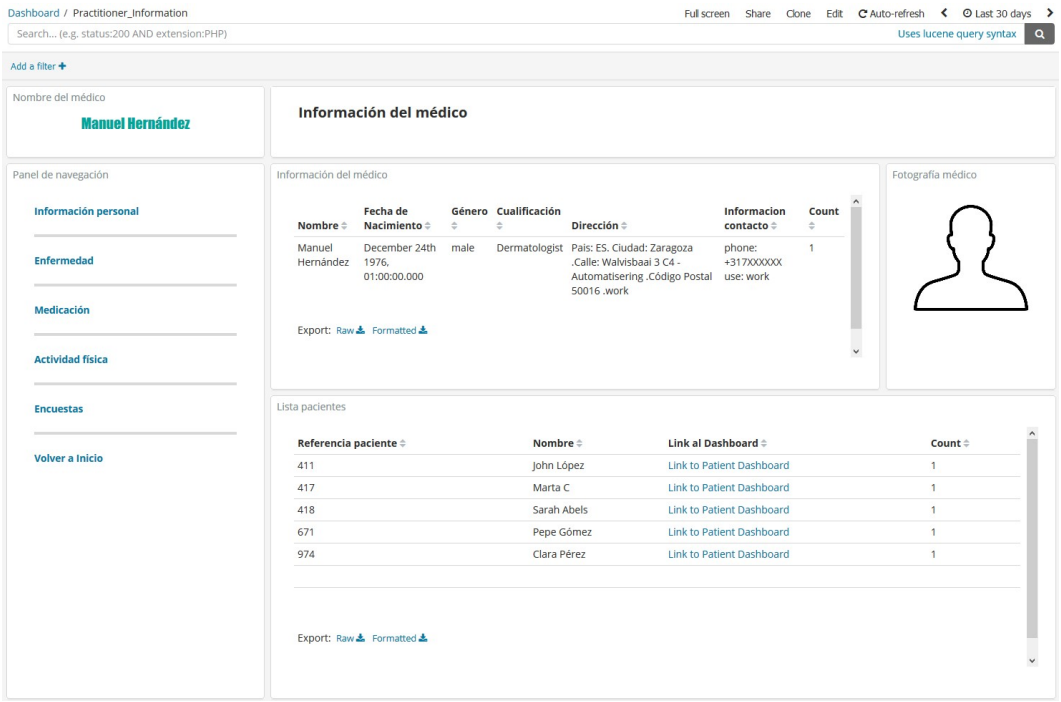

Figura 4.1: *Dashboard* Información personal médico.

-**Información personal Paciente:** Contiene una tabla con información personal del paciente, una fotografía, las alergias registradas e información sobre las próximas citas pendientes. También se añade una barra de navegación que permite moverse entre los *dashboards* creados. Es el *dashboard* de inicio de los pacientes.

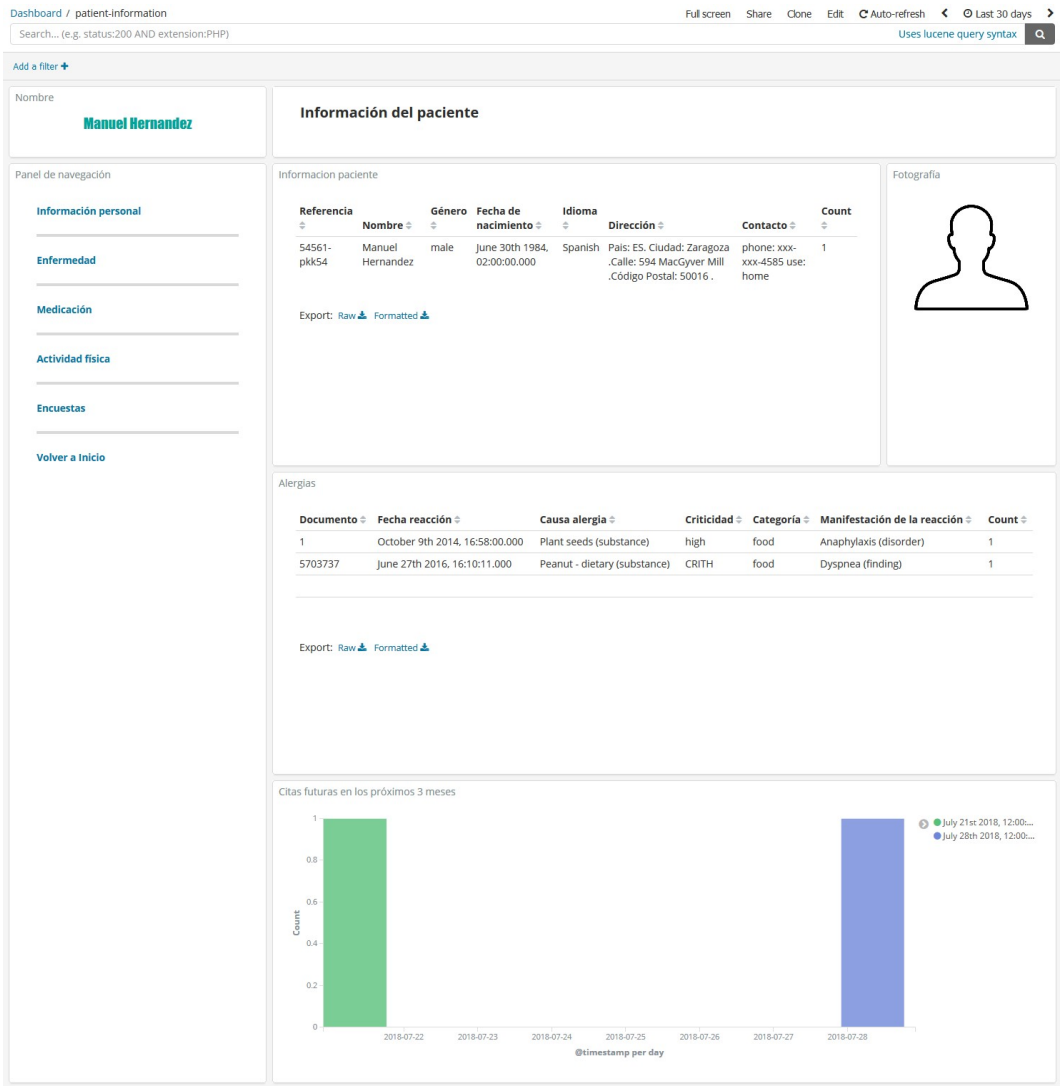

Figura 4.2: *Dashboard* Información personal del paciente.

- **Enfermedades:** Contiene información sacada de los recursos "condition" de FHIR. El enlace lleva a un *dashboard* diferente dependiendo del tipo de enfermedad que padezca el paciente. Se crean tres *dashboards*, uno para enfermos de diabetes, el segundo para enfermos de psoriasis y para el resto se crea un *dashboard* general. En el caso de paciente de diabetes se extrae de los recursos "observation" información relativa al peso, altura, pulso cardiaco, presión sanguínea y glucosa, se han escogido estos parámetros ya que son los valores que se monitorizan para estos pacientes [19]. Para los pacientes de psoriasis se representa la última imagen que se ha incluido en el recurso "media" sobre el desarrollo de la enfermedad junto con la información extraída de ella y una representación temporal indicando las imágenes anteriores disponibles. El *dashboard* general para el resto de pacientes muestra la enfermedad que padecen obteniéndola del recurso "conditions" y los valores de sus últimas observaciones recogidas.

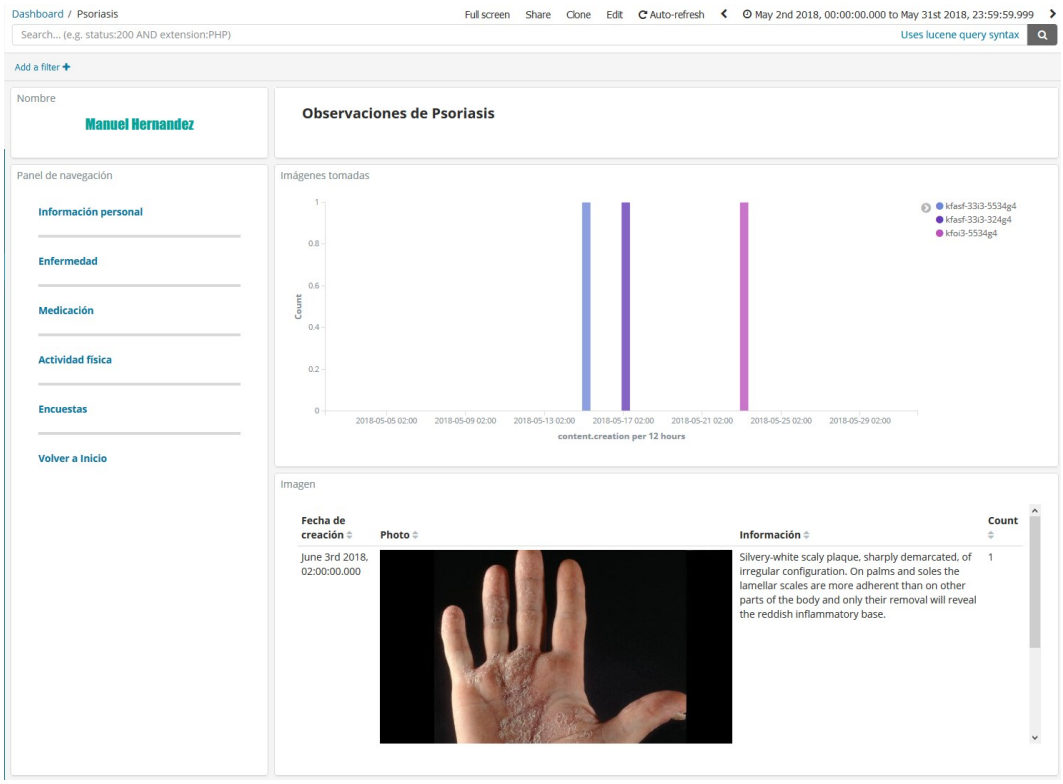

Figura 4.3: *Dashboard* Enfermedad para paciente de psoriasis.

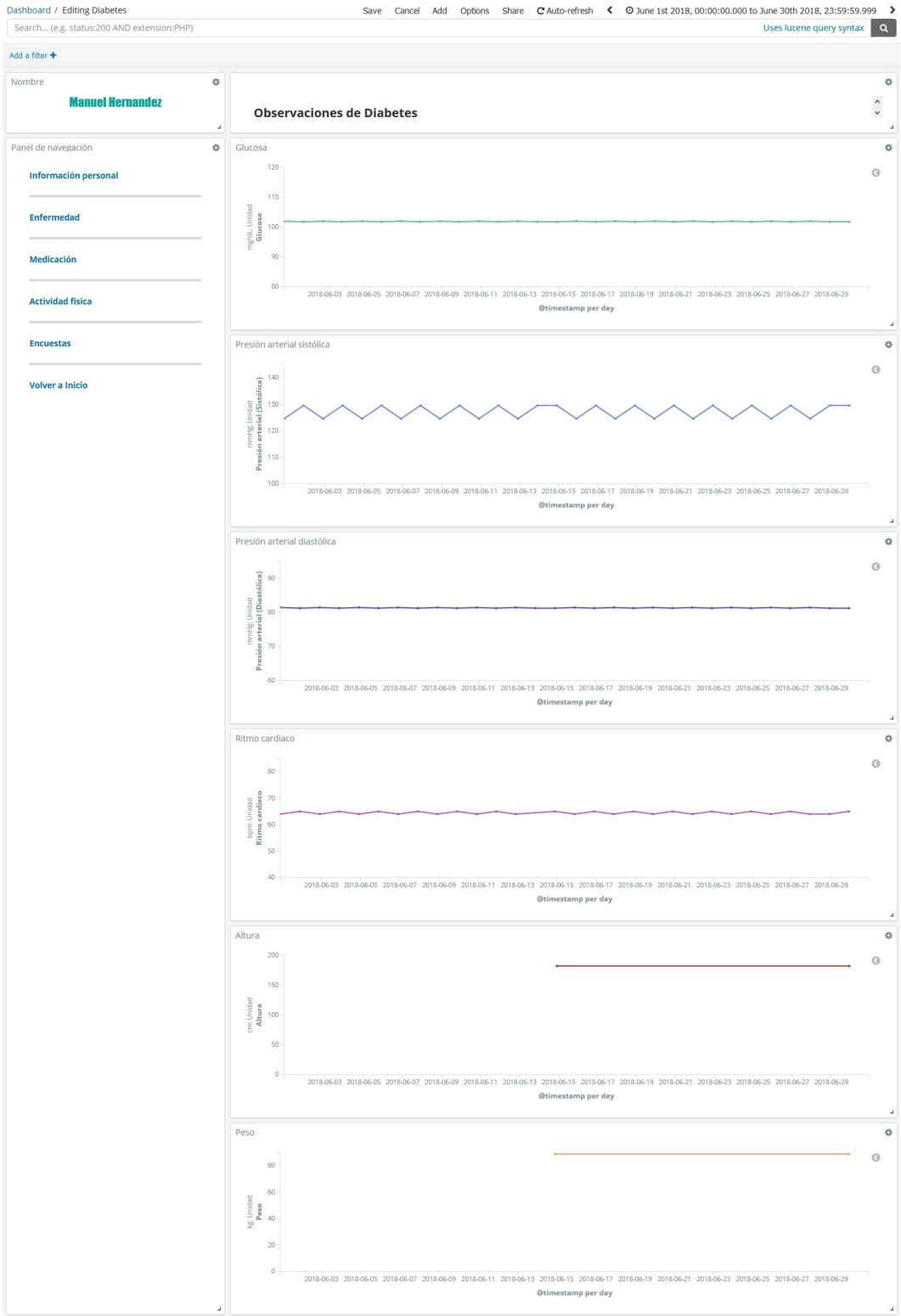

Figura 4.4: *Dahboard* Enfermedad para paciente de diabetes

- **Medicación:** Este *dashboard* se crea para mostrar la medicación que debe tomar un paciente. Esta información se incluye en el recurso "MedicationStatement", donde se indica la medicación y el número de tomas que debe realizar. También se incluye un gráfico mostrando las tomas del medicamento por paciente.

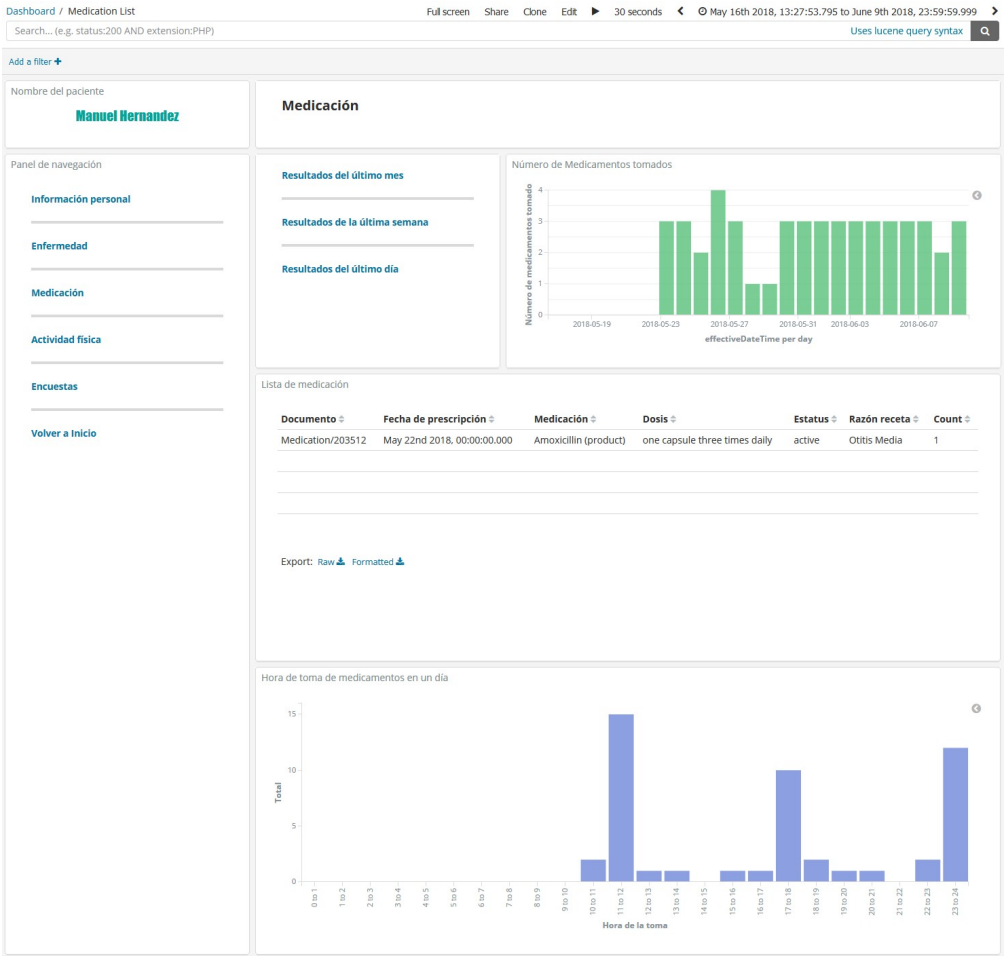

Figura 4.5: *Dashboard* Medicación.

- **Actividad:** En este *dashboard* se recoge la información recogida por Google Fit a la que el paciente permite su acceso. En él se incluyen visualizaciones mostrando su tiempo de actividad e inactividad, los pasos andados y la distancia recorrida, un mapa con las ubicaciones recogidas y los valores de ritmo cardiaco que registra.

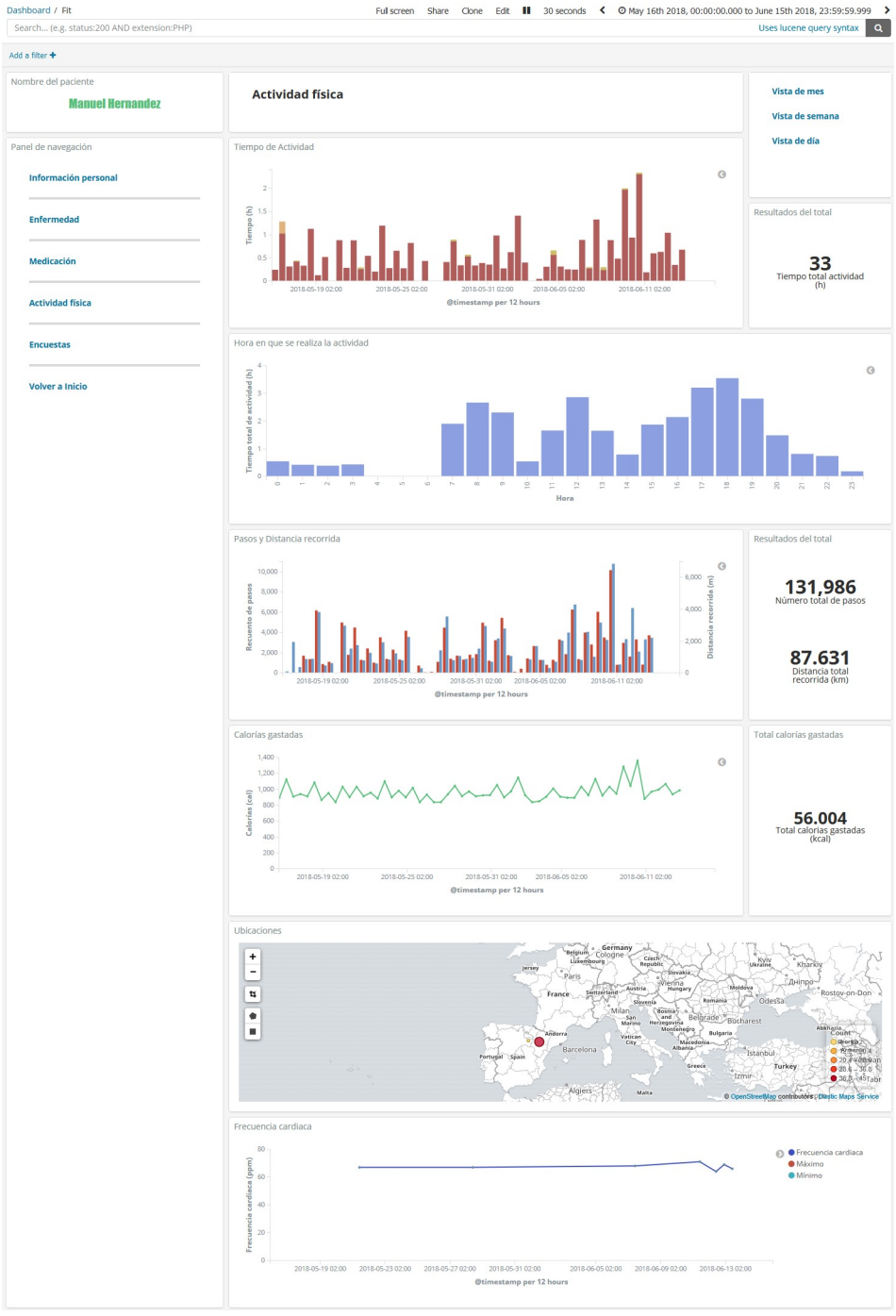

Figura 4.6: *Dashboard* Registro de actividad.

- **Encuestas:** Incluye la información de las encuestas realizadas por el paciente. Estos datos se extraen del recurso FHIR "QuestionnaireResponse", y se representa en una tabla la pregunta general incluida en el recurso "Questionnaire" y la respuesta que ha dado el paciente. También se incluye un gráfico que muestra las encuestas realizadas en un eje temporal y que permite seleccionar la encuesta de la que se desee obtener las respuestas.

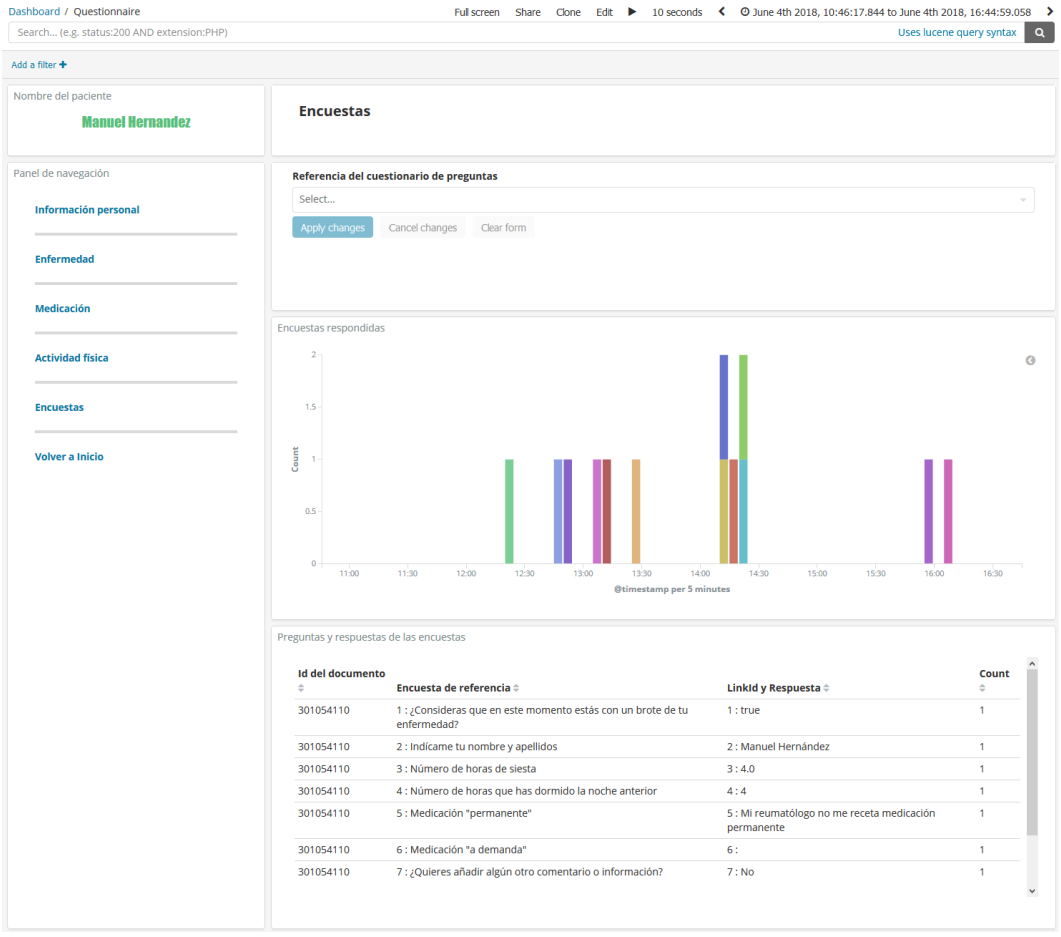

Figura 4.7: *Dashboard* Encuestas.

4.2 Dashboards de logs de interacciones con los microservicios

-**Interacciones con el Bot:** Contiene información sobre las interacciones de cada usuario con los microservicios, los mensajes intercambiados y la duración total de los flujos de comunicación de cada usuario.

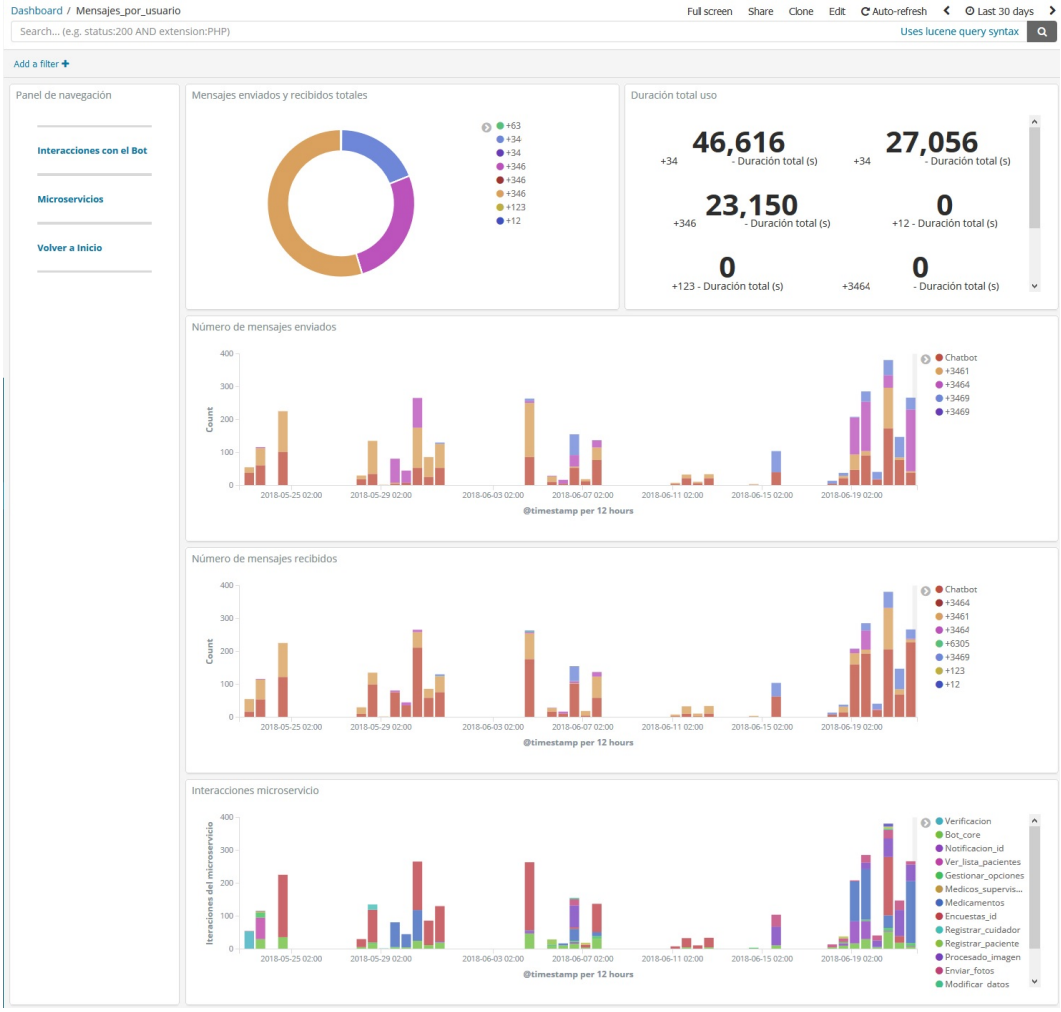

Figura 4.8: *Dashboard* Interacciones con el *Bot*.

-**Microservicios:** Este *dashboard* muestra la información relativa a cada microservicio y los flujos de uso que ha generado. En el *dashboard* se incluye un panel de navegación con enlaces a los dos paneles de la aplicación de *logs*: "Interacciones con el bot" y "Microservicios". Es el dashboard utilizado de inicio para los usuarios de rol "admin".

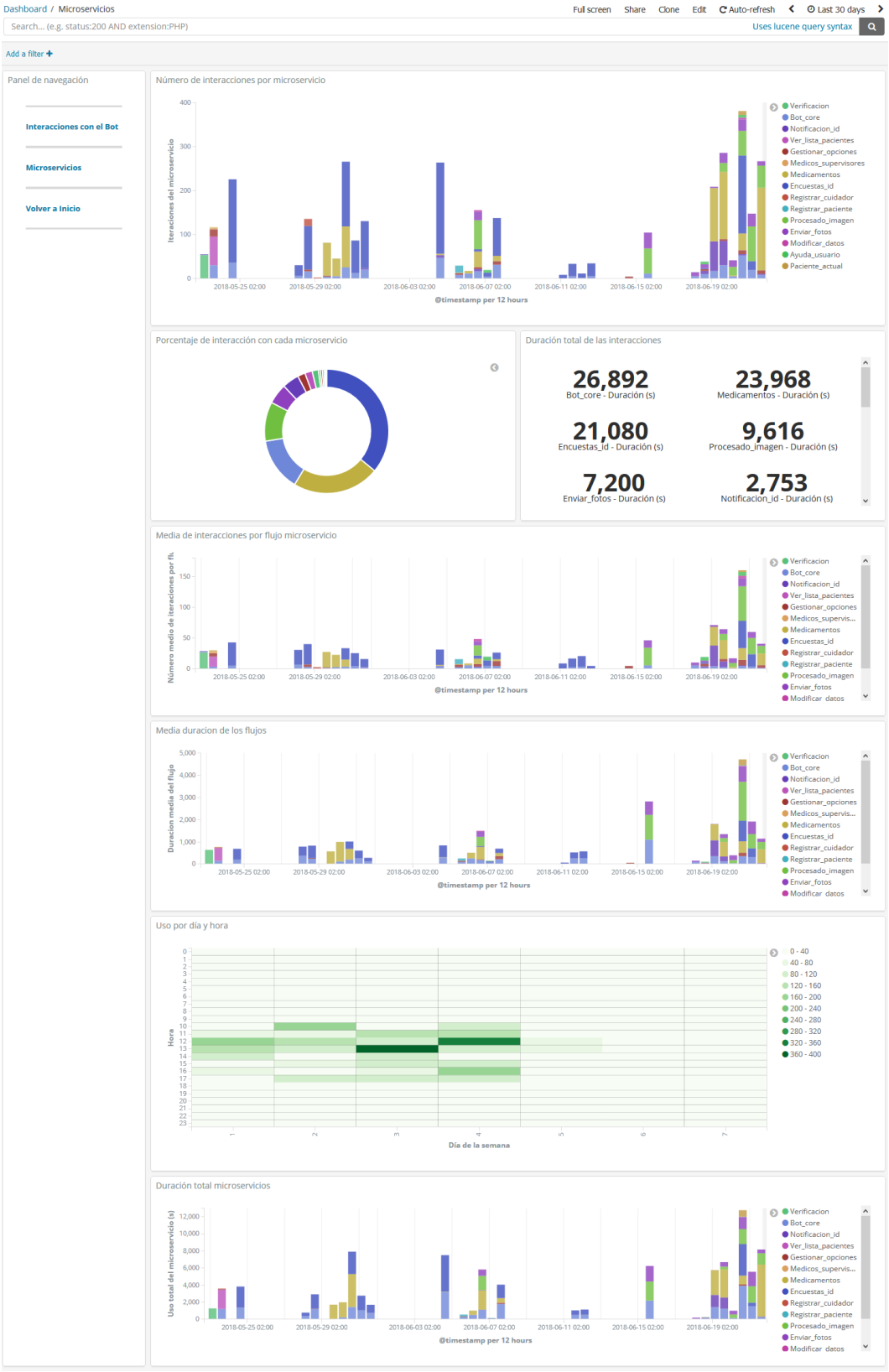

Figura 4.9: *Dashboard* Microservicios.

Capítulo 5

Conclusiones y líneas futuras

5.1 Conclusiones

En este proyecto se han utilizado tecnologías de gestión de red para realizar una carpeta personal de salud utilizando estándares médicos. Dada la importancia de la información se ha implementado una arquitectura de recolección y visualización de la información médica que garantice la identidad del usuario que accede a los datos.

Se han analizado las principales herramientas para realizar cada una de las tareas del proyecto, seleccionando en cada caso la mejor opción posible, buscando herramientas que permitan optimizar el rendimiento para poder crear un resultado funcional en un escenario con un número elevado de usuarios. Se ha utilizado herramientas de gestión de red ya existentes y que se podrían utilizar para este tipo de información pero se han encontrado limitaciones puesto que estas herramientas están preparadas para trabajar con valores numéricos y datos temporales, recibiendo *logs* periódicamente. Esto ha limitado los resultados obtenidos puesto que se trabaja con datos médicos cuyos valores pueden no tener una referencia temporal, como la información personal o enfermedades de un paciente y con valores no numéricos que no permiten realizar todas las agregaciones definidas en Elasticsearch. Estas limitaciones se han evitado modificando los recursos mediante Logstash, representando los valores no numéricos en tablas y

definiendo algunos índices sin referencia temporal para evitar el filtrado que Kibana impone por defecto. Elasticsearch tampoco permite representar las imágenes almacenadas dentro de sus datos, por lo que se han utilizado los servidores de control de acceso para modificar las respuestas y poder representarlas. Aunque estas herramientas tienen limitaciones para trabajar con datos médicos, son especialmente útiles cuando se utilizan para trabajar con *logs*, permitiendo obtener estadísticas y nueva información sobre el uso que puede ser utilizada para mejorar la aplicación.

También se ha abordado el control de acceso en Elastic ya que no queda definido por defecto. Para esta aplicación es necesario garantizar la privacidad de los datos, por lo que se ha modificado la arquitectura del sistema para garantizar un control de acceso. Esto se ha implementado utilizando el protocolo OAuth y una *Cookie* que identifique a cada usuario que acceda a la aplicación y determine a que recursos puede acceder.

Se ha conseguido implementar una carpeta personal de salud funcional, que represente la información de cada paciente garantizando la privacidad y el acceso a la información por parte de médicos y pacientes. Por lo tanto, el objetivo que se pretendía alcanzar con la realización de este trabajo de fin de máster ha sido plenamente satisfecho.

5.2 Líneas futuras

Aunque todos los objetivos se han cumplido en este TFM, se plantean unas posibles mejoras que aportarían funcionalidades extra al sistema.

- **Integración de más aplicaciones de datos de actividad,** como la aplicación Health de Apple o la aplicación Polar Flow de Polar. Esto permitirá introducir nuevos datos al sistema provenientes de distintas fuentes.

- **Añadir nuevas funcionalidades al sistema,** como podría ser la generación de avisos cuando algunos valores médicos superen un umbral.

- **Incluir nuevas enfermedades en los *dashboards*,** para este proyecto se han creado paneles con información específica sobre las enfermedades crónicas psoriasis y diabetes, pero se podría aumentar los *dashboards* con información relacionada con nuevas enfermedades.

- **Incluir más recursos FHIR¸** este estándar incluye numerosos recursos con información médica de los cuales se podría trabajar con un número mayor en los *dashboards* para obtener más información médica.

Bibliografía

[1] Joshua C Mandel, David A Kreda, Kenneth D Mandl, Isaac S Kohane, and Rachel B Ramoni. Smart on fhir: a standards-based, interoperable apps platform for electronic health records. *Journal of the American Medical Informatics Association*, 23(5):899–908, 2016.

[2] iHealth Ventures LLC. Doctors appointment reminder, mobile app platform, 2018. Disponible en, http://www.ihealthventures.com/.

[3] Olaronke Iroju, Abimbola Soriyan, Ishaya Gambo, and Janet Olaleke. Interoperability in healthcare: benefits, challenges and resolutions. *International Journal of Innovation and Applied Studies*, 3(1):262–270, 2013.

[4] Tim Benson. *Principles of health interoperability HL7 and SNOMED*. Springer London:, 2010.

[5] Robert H Dolin, Liora Alschuler, Sandy Boyer, Calvin Beebe, Fred M Behlen, Paul V Biron, and Amnon Shabo. Hl7 clinical document architecture, release 2. *Journal of the American Medical Informatics Association*, 13(1):30–39, 2006.

[6] Medical Imaging and Technology Alliance. Dicom documentation, 2018. Disponible en, https://www.dicomstandard.org/current/.

[7] Clement J McDonald, Stanley M Huff, Jeffrey G Suico, Gilbert Hill, Dennis Leavelle, Raymond Aller, Arden Forrey, Kathy Mercer, Georges DeMoor,

John Hook, et al. Loinc, a universal standard for identifying laboratory observations: a 5-year update. *Clinical chemistry*, 49(4):624–633, 2003.

[8] HL7. Fhir documentation, 2018.

[9] David Carasso. Exploring splunk. *published by CITO Research, New York, USA, ISBN*, pages 978–0, 2012.

[10] GrafanaLabs. Grafana documentation, 2018. Disponible en, http://docs.grafana.org/.

[11] Graphite. Graphite documentation, 2018. Disponible en, http://graphite.readthedocs.io/en/latest/.

[12] Elastic. Elastic stack documentation, 2018. Disponible en, https://www.elastic.co/guide/index.html.

[13] Elastic. Beats documentation, 2018. Disponible en, https://www.elastic.co/guide/en/beats/libbeat/current/index.html.

[14] Elastic. Logstash documentation, 2018. Disponible en, https://www.elastic.co/guide/en/logstash/current/index.html.

[15] Elastic. Elasticsearch documentation, 2018. Disponible en, https://www.elastic.co/guide/en/elasticsearch/current/index.html.

[16] Elastic. Kibana documentation, 2018. Disponible en, https://www.elastic.co/guide/en/kibana/current/index.html.

[17] HAPI FHIR. Hapi fhir documentation, 2018. Disponible en, http://hapifhir.io/docindex.html.

[18] Flask. Flask documentation, 2018. Disponible en, http://flask.pocoo.org/docs/1.0/.

[19] Nelia Lasierra Beamonte, Álvaro Alesanco Iglesias, and José García Moros. An ontology-driven architecture for data integration and management in home-based telemonitoring scenarios. *Zaguan Unizar*, 2012. Presentado: 11 12 2012.