Anexo A

Listado de acrónimos

- **API:** *Application Programming Interface.*
- **CDA:** *Clinical Document Architecture.*
- **CPU:** *Central Processing Unit.*
- **CRUD:** *Create, Read, Update and Delete.*
- **DICOM:** *Digital Imaging and Communication in Medicine.*
- **DSL:** *Domain Specific Language.*
- **EHR:** *Electronic Health Record.*
- **ELK:** *ElasticSearch, Logstash and Kibana.*
- **FHIR:** *Fast Healthcare Interoperability Resources.*
- **HL7:** *Health Level Seven.*
- **HTTP:** *Hypertext Transfer Protocol.*
- **IP:** *Internet Protocol.*
- **JSON:** *JavaScript Object Notation.*
- **JWT:** *JSON Web Token.*

- **LOINC:** *Logical Observation Identifiers Names and Codes.*
- **NRT:** *Near Real Time.*
- **OAuth:** *Open Authorization.*
- **PHR:** *Personal Health Record.*
- **REST:** *Representational State Transfer.*
- **RIM:** *Reference Information Model.*
- **TCP:** *Transport Control Protocol.*
- **TFM:** Trabajo Fin de Máster.
- **URL:** *Uniform Resource Locator.*

Anexo B

Recursos FHIR utilizados

En este anexo se muestra la estructura de los recursos FHIR utilizados. Toda esta información ha sido extraída del propio estándar, el cual se encuentra disponible en el siguiente directorio: https://www.hl7.org/fhir/ [8]

Para el desarrollo del proyecto se han utilizado los siguientes recursos:

- **AllergyIntolerance:** El recurso contiene una valoración clínica de una alergia o intolerancia.
- **Appointment:** Se utiliza para proveer información sobre un encuentro planeado que puede ser futuro o pasado. Cada recurso indica un único encuentro.
- **Condition:** El recurso indica un estado clínico, problema, diagnostico u otro evento, situación o problema que se ha elevado a un nivel de riesgo.
- **Immunization:** Describe el evento de la administración de una vacuna a un paciente o el registro de una vacunación indicado por el paciente, un médico u otra asociación.
- **Media:** Contiene una foto, vídeo o audio que sea utilizado como parte del proceso de cuidado sanitario.
- **MedicationStatement:** Indica un registro de la medicación que está tomando o ha tomado un paciente.

- **Observation:** El recurso contiene resultados sobre pruebas médicas de un paciente.
- **Patient:** Cubre la información personal de cada paciente.
- **Practitioner:** Cubre la información personal de cada doctor.
- **Questionnaire:** En este recurso se almacenan modelos de encuestas que pueden ser realizadas por el paciente, incluyendo las preguntas y el formato de respuesta esperado.
- **QuestionnaireResponse:** Este recurso contiene las respuestas dadas por un paciente a una encuesta.

Se desglosan con mayor detalle los recursos "Patient", "Condition" y "Observation".

B.1 Patient

Este recurso contiene información personal del paciente. El campo "identifier" indica el identificador lógico que permite identificar al paciente al cual pertenece en cualquier recurso FHIR en los campos "patient.reference" o "subject.reference". Otros campos importantes son "name" que contiene el nombre del paciente, "telecom" con la información de contacto o "birthdate" con su fecha de nacimiento.

B.2 Observation

Este recurso contiene resultados sobre pruebas médicas realizadas por un paciente. El campo "subject.reference" permite identificar al paciente al cual pertenece el resultado. "EffectiveDateTime" indica la fecha en la que se realizó la prueba y "code" permite identificar el tipo de prueba realizada. El valor de los resultados aparece en el campo "value" o "component".

B.3 Condition

El recurso indica un estado clínico, problema, diagnostico u otro evento, situación o problema que se ha elevado a un nivel de riesgo. Campos importantes de este recurso son: “severity” indica la gravedad de esta enfermedad, “code” para determinar el tipo de enfermedad y “bodySite” para saber en qué parte del cuerpo se ha dado la enfermedad.

Anexo C

Interfaz RESTful FHIR

En este anexo se explica la interfaz RESTful de comunicaciones utilizada por el estándar FHIR. La información se puede consultar de manera gratuita en la documentación del propio estándar [8].

REST indica cualquier interfaz entre sistemas que utilice HTTP para obtener datos o generar operaciones sobre esos datos en cualquier formato posible, como JSON o XML.

FHIR proporciona una API REST para manipular los recursos con un conjunto de interacciones, las cuales pueden realizarse a nivel de instancia, de tipo de recurso y de sistema. Las operaciones definidas son las siguientes:

- Interacciones definidas a **nivel de instancia:**

  - **Read:** Devuelve el estado actual de un recurso.
  - **Vread:** Devuelve el estado de una versión específica de un recurso.
  - **Update:** Actualiza un recurso existente o lo crea por la id especificada.
  - **Patch:** Actualiza un recurso existente cambiando los campos especificados.
  - **Delete:** Elimina un recurso.
  - **History:** Devuelve el historial de cambios de un recurso concreto.

- Interacciones a **nivel de tipo de recurso:**

- **Create:** Crea un nuevo recurso con una id asignada por el servidor.
- **Search:** Busca el tipo de recurso basado en un criterio de búsqueda.
- **History:** Devuelve el historial de cambios de un tipo de recurso.

- Interacciones a **nivel de sistema:**

- **Capabilities:** Solicita la declaración de capacidad del sistema.
- **Batch/transaction:** Actualiza, crea o elimina un conjunto de recursos en una única interacción.
- **History:** Devuelve el historial de cambios de todos los recursos
- **Search:** Busca sobre todos los tipos de recurso basado en un criterio de búsqueda.

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.

Read

Permite acceder al estado actual de un recurso mediante una interacción de tipo HTTP GET:

GET [base]/[type]/[id]

La respuesta es una única instancia con el contenido específico para el tipo de recurso. La petición se puede anidar con el parámetro "elements" que permite devolver únicamente los campos especificados y no el documento completo.

Update

Esta interacción crea una nueva versión de un recurso o crea un nuevo recurso si no existe ninguno con la id aportada, utiliza HTTP PUT:

PUT [base]/[type]/[id]

Si el recurso existía se devolverá un código 200 OK y en caso contrario 201 Created.

Delete

Esta interacción elimina un recurso existente

DELETE [base]/[type]/[id]

El servidor devolverá un código 204 No Content.

Search

GET [*base*]/[*type*] {?[*parameters*]}

POST [*base*]/[*type*]/*_search* {?[*parameters*]}

Permite realizar una búsqueda de un conjunto de recursos en función de ciertos parámetros de filtrado.

Soporta búsquedas con peticiones del tipo GET y POST. En el campo "parameters" se introduce una tupla "key-value" para filtrar por valores concretos de determinados campos. Este campo puede tener múltiples tuplas "key-value" concatenadas por el carácter "&" .

Anexo D

Interfaz RESTful ElasticSearch

En este anexo se explica la interfaz de comunicaciones utilizada por ElasticSearch. Toda esta información ha sido obtenida de la documentación del propio producto, la cual se encuentra disponible en el siguiente directorio: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html [15]

REST indica cualquier interfaz entre sistemas que utilice HTTP para obtener datos o generar operaciones sobre esos datos en cualquier formato posible, como JSON o XML.

Las API de ElasticSearch utilizan el lenguaje JSON sobre HTTP. ElasticSearch tiene definidas una serie de APIs para trabajar con los datos donde se encuentran definidas las operaciones que se pueden realizar.

API sobre un **único documento:**

- Index: Añade o actualiza un documento con un determinado índice e identificado por su campo "id" a partir de un documento JSON incluido en el mensaje.

- Get: Permite devolver un determinado documento JSON a partir de su índice e identificado por el valor de su campo "id".

- Delete: Permite eliminar un determinado documento JSON a partir de su índice e identificado por el valor de su campo "id".

- Update: Permite actualizar un documento mediante un script que contiene el mensaje. El documento se identifica mediante su campo “id” y su índice.

El formato de las peticiones es el siguiente:

Donde “VERB” indica el verbo HTTP usado, “index” el índice utilizado, “type” el tipo de documento y “id” el identificador lógico del documento. El parámetro “options” es opcional y puede incluir campos auxiliares u operaciones de cada API. Los campos entre “[]” son obligatorios y los campos entre?” son opcionales.

API para **múltiples documentos:**

- Multi Get: Permite obtener múltiples documentos a partir de su índice y parámetros opcionales como su campo “type” o “id”.
- Bulk: Permite realizar varias operaciones de tipo “index” y “delete” en una única llamada a la API.
- Delete By Query: Permite borrar cada documento que cumpla una determinada consulta.
- Update By Query: Realiza una actualización en cada documento de un determinado índice sin necesidad de modificar el origen
- Reindex: Permite copiar documentos de un índice a otro.

Todas estas peticiones incluyen un documento JSON en el cuerpo del mensaje para diferenciar entre varias consultas dentro de la misma petición. El estilo es el siguiente:

Donde “index” y “type” pueden ir dentro de la URL o dentro del contenido del mensaje dependiendo de si la petición múltiple es para el mismo índice o tipo o si es diferente para cada documento. “API” indica la API utilizada para realizar la petición.

API para **búsquedas:**

- Search: Permite ejecutar una petición de búsqueda y devolver los resultados que cumplan la condición.

- Multi Search: Permite ejecutar múltiples peticiones de búsqueda dentro de la misma API.

API para **índices:** Permiten realizar la gestión de índices como su creación o borrado, cambiar su configuración y mapeado o modificar su alias.

Kibana utiliza las API para múltiples documentos "Multi Get", "Search" y "Multi Search" para obtener los documentos necesarios para realizar las representaciones.

Anexo E

Dispatcher log

En este capítulo se muestra la estructura del archivo utilizado para analizar el uso del *Bot*. Estos ficheros se generan en los microservicios por cada interacción que realiza un usuario con el *Bot*, generando una nueva entrada con cada mensaje. Un ejemplo de registro es el mostrado en la figura E.1.

```
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 E.1: Ejemplo de registro del "Dispatcher".

El registro contiene unos campos prefijados que mantienen siempre la misma estructura en cada entrada. Estos campos se encuentran separados por espacios y son los indicados a continuación en el siguiente orden:

- **Running time:** Indica los milisegundos desde que se lanzó el *Bot*.
- **Proceso:** Indica el proceso que genera el log, se encuentra siempre entre los caracteres '[ ]'.
- **Fecha:** Contiene la fecha en que se realizó el mensaje. En formato año-mes-día hora:minutos:segundos,milisegundos.
- **Tipo:** Indica el carácter del mensaje.

- **Servicio:** Indica el servicio que se encuentra activo.
- **Origen:** Usuario origen del mensaje. Se indica precedido del texto "-Origen:".
- **Destino:** Usuario al que va dirigido el mensaje. Se indica precedido del texto "Destino:".
- **Microservicio:** Indica la funcionalidad del microservicio que ha originado el mensaje. Se indica precedido del texto "Funcionalidad:".
- **Mensaje:** Contenido del mensaje. Se indica precedido del texto "Message".
- **Adjunto:** Indica si el mensaje contiene contenido adjunto y en el caso afirmativo se incluye el contenido. Se indica precedido del texto "Attachment:". Este campo es el último que contiene cada entrada.

El contenido de estos mensajes se parsea mediante Logstash con la herramienta Grok que permite obtener el valor de cada campo y generar un nuevo archivo JSON.