Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño y desarrollo de un servicio FHIR como PMI

Autor: Daniel García de Sola Alonso
Tutor: Isabel Román Martinez

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño y desarrollo de un servicio FHIR como PMI

Autor:
Daniel García de Sola Alonso

Tutor:
Isabel Román Martínez
Profesor colaborador

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Grado: Diseño y desarrollo de un servicio FHIR como PMI

Autor: Daniel García de Sola Alonso

Tutor: Isabel Román Martínez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2013

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigos

Agradecimientos

Desde que inicié mi vida universitaria solía pensar más en el tiempo que me quedaba para terminar que en disfrutar de lo que posiblemente sea una de las mejores etapas de mi vida. Por el camino he ido cruzándome con diferentes personas, personas que se fueron convirtiendo en mi día a día, personas que en muy poco tiempo se ganaron mi confianza, personas que de alguna manera u otra han dejado una huella irremplazable en mi corazón.

El camino ha estado lleno de obstáculos, periodos difíciles de los que he aprendido que con esfuerzo y constancia todo tiene una solución. El paso del tiempo ha conseguido que poco a poco aprendiera a disfrutar de los pequeños momentos que me ha regalado esta carrera, convirtiendo los últimos años en un agradable paseo.

Con este trabajo se cierra un ciclo de mi vida inolvidable y no me gustaría hacerlo sin agradecer a todas esas personas que me han acompañado en este largo trayecto, haciendo de lo peor lo mejor y de lo mejor lo excepcional.

A mis padres, Natalia y Enrique, a mi hermano Alejandro y en general a mi familia, primos, tíos, abuelos… por apoyarme en las situaciones más complicadas y por darme siempre todo lo que han tenido y más.

A mis otros hermanos Javier, Chequio y Parra que me han acompañado desde siempre, por demostrarme que la amistad verdadera es lo más importante que alguien puede tener en la vida.

A mis amigos de Bachillerato, porque aunque cada persona escoge su camino y el contacto es cada vez menor, por mucho tiempo que pase todo sigue igual.

A la nueva familia que me ha regalado esta estapa universitaria, los Tabernícolas, Baranguá, los Machos y Telelimonar, por hacerme entender que el sentido del humor tiene que ser el principal principio de la vida y que una amistad de pocos años puede convertirse en una pieza imprescindible del día a día.

A mi fiel compañero Velero, por las cientos de historias que ha desencadenado, historias de amistad, amor, alegrías, lágrimas y por supuesto un sinfín de risas.

A los profesores que verdaderamente aman su profesión, por saber motivarnos aún cuando la temática de la asignatura era el mayor obstáculo.

A todos, muchas gracias.

Daniel García de Sola Alonso

Sevilla, 2019

Resumen

En la situación actual, existe una gran diversidad en el conjunto de Tecnologías de la Información y la Comunicación (TICs) empleadas en el entorno sanitario. El uso de múltiples tecnologías y estándares para recopilar información clínica, almacenarla, intercambiarla o gestionarla, dificultan en gran medida la colaboración entre diferentes instituciones.

La interoperabilidad entre distintos organismos es siempre algo complicado, debido a que cada uno de ellos puede trabajar internamente de forma muy diferente. Es precisamente el acceso a datos a través de sistemas de forma estandarizada, facilitando dicha interoperabilidad ente los sistemas de atención médica y aplicaciones de terceros, lo que nos aporta la especificación Fast Healthcare Interoperatibility Resources (FHIR), desarrollado por la organización Health Level 7 (HL7).

Este proyecto tiene como objetivo el diseño y desarrollo de un servicio REST, siguiendo la especificación FHIR, para el acceso a la información demográfica de personas (recurso Person), con funcionalidad de PMI (Patient Master Index), actuando como intermediario entre un cliente y varios sistemas FHIR. El servicio recibe peticiones por parte de los usuarios y realiza una búsqueda de posibles candidatos en múltiples servidores a partir de los parámetros aportados. Devuelve el total de resultados ponderados con un determinado nivel de confianza. Además, es capaz de detectar posibles coincidencias entre los diferentes recursos obtenidos.

Previo a dicho diseño, se ha realizado un análisis profundo del estándar y de su situación actual en el mercado para establecer así un contexto actualizado.

Abstract

In the current situation, there is a big diversity in the set of Information and Communications Technologies (ICTs) used in the healthcare environment. The use of multiple technologies and standards to collect clinical information, store it, exchange it or manage it, complicates the collaboration between different institutions.

The interoperability between different organisms is always complicated, because each of them can work internally in a very different way. It is precisely the access to data through systems in a standardized way, facilitating that interoperability between medical care systems and applications from third parties, which gives us the Fast Healthcare Interoperatibility Resources (FHIR) specification, developed by Health Level 7 (HL7).

The objective of this project is the design and development of a REST service, following the FHIR specification, for access to people's demographic information (Person resource), with PMI functionality (Patient Master Index), acting as an intermediary between a client and some FHIR systems. The service receives requests from users and performs a search of possible candidates in multiple servers based on the parameters provided. Returns the total weighted results with a certain level of confidence. In addition, it is able to detect possible duplicates between the different resources obtained.

Previous to this design, an in-depth analysis of the standard and its current situation in the market has been carried out in order to establish an updated context.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

FHIR Fast Health Interoperability Resources

CDA Clinical Document Architecture

HL7 Health Level Seven

ASF Apache Software Foundation

RIM Reference Information Model

REST Representational State Transfer

PMI Patient Master Index

ASF Apache Software Foundation

SOAP Simple Object Access Protocol

1 INTRODUCCIÓN

El cambio hacia la Sanidad del futuro es algo que ya ha llegado. Cada vez existe más información sanitaria, junto a ella aumentan los medios por los que los usuarios acceden a dicha información y se multiplican las instituciones que aportan datos. Así, el intercambio de estos datos entre sistemas, es un pilar fundamental en los procesos de atención, gestión y generación de documentación sanitaria. Con frecuencia, se desarrollan sistemas de información propietarios que no contemplan dicho intercambio, que tienen una gestión complicada y que no aportan sostenibilidad.

Un servidor demográfico es aquel que únicamente procesa datos demográficos. En este proyecto, el servicio recibe parámetros demográficos a partir de los cuales realiza una búsqueda en una federación de sistemas FHIR, con el objetivo de identificar a una persona en cada uno de los sistemas que lo conforman. Diferentes sistemas FHIR en una federación pueden guardar información sobre una misma persona, pero cada uno de ellos utilizará un identificador distinto para referirse a dicha persona. Un PMI (Master Patient Index) es capaz de resolver ese problema, obteniendo el id que se usa en cada sistema de la federación para referirse a la misma persona. Además, garantiza que cada paciente sea único en casa sistema, sin registros duplicados, mediante diferentes algoritmos de búsqueda.

El acceso a la historia clínica del paciente comienza por la identificación del mismo. Si los datos de la historia están diseminados en varios sistemas será necesario primero identificar al paciente en cada uno de los sistemas de forma unívoca, asegurar que todos los registros hacen referencia a la misma persona, recuperar los datos de cada uno de ellos e integrarlos. Si no es posible la identificación en alguno de los sistemas, puede dar lugar a la generación de información duplicada, por ejemplo introduciendo de nuevo datos ya registrados. La búsqueda e identificación de personas se convierte así, en un pilar fundamental para el acceso a la historia clínica distribuida de los pacientes.

La falta de estándares precisos sobre el formato de los datos demográficos da como resultado una incosistencia en el intercambio de información entre los profesionales de la salud, los mismos pacientes y los repositorios de la información. Esta incosistencia dificulta el problema de la vinculación de registros, cuyo objetivo es identificar aquellas historias clínicas distribuidas en varios sistemas que se refieran a una misma persona en el mundo real. Existen numerosos sofisticados algoritmos de coincidencia que abordan esta compleja problemática.

1.1. Objetivos

FHIR, Fast Healthcare Interoperability Resources, 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. Combina lo mejor de cada uno de los estándares actualmente en uso (fundamentalemente HL7 versión 2, versión 3 y CDA) con estándares web de forma que se mejore en la medida posible la implementación de los estándares de interoperabilidad.

Con la aparición de FHIR, la interoperabilidad entre diversos entornos de atención médica ya no necesita resolverse con extensiones personalizadas, complejas y costosas. Es el estándar de próxima generación que facilita la integración de los sistemas existentes.

El objetivo principal de este proyecto es crear un servicio REST que haga frente a los principales problemas comentados anteriormente. Por un lado, que implemente un estándar que promete una importante revolución

en un mundo tan complejo como el de la interoperabilidad sanitaria, permitiendo la interacción con cualquier cliente que se haya desarrollado conforme a este estándar, y por otro lado que tenga una funcionalidad parecida la de un PMI (Patient Master Index) manejando únicamente recursos tipo *person* y siendo capaz de obtener el id que se usa en cada sistema de la federación para referirse a la misma persona.

Siguiendo la especificación FHIR, el servicio proporciona acceso a la información demográfica de personas en múltiples sistemas externos. Recibe peticiones por parte de los clientes y realiza una búsqueda de posibles candidatos en todos los servidores a partir de los parámetros demográficos aportados. Devuelve el total de resultados ponderados con un determinado nivel de confianza. Para el cálculo de dicho nivel, se realiza una validación determinista y/o probabilista, mediante el uso de diferentes técnicas que serán detalladas más adelante. Una vez resuelta la lista de candidatos finales, realiza un proceso de identificación de coincidencias para informar al usuario sobre registros referidos a una misma persona, devolviendo el id que se usa en cada sistema de la federación para referirse a la misma persona.

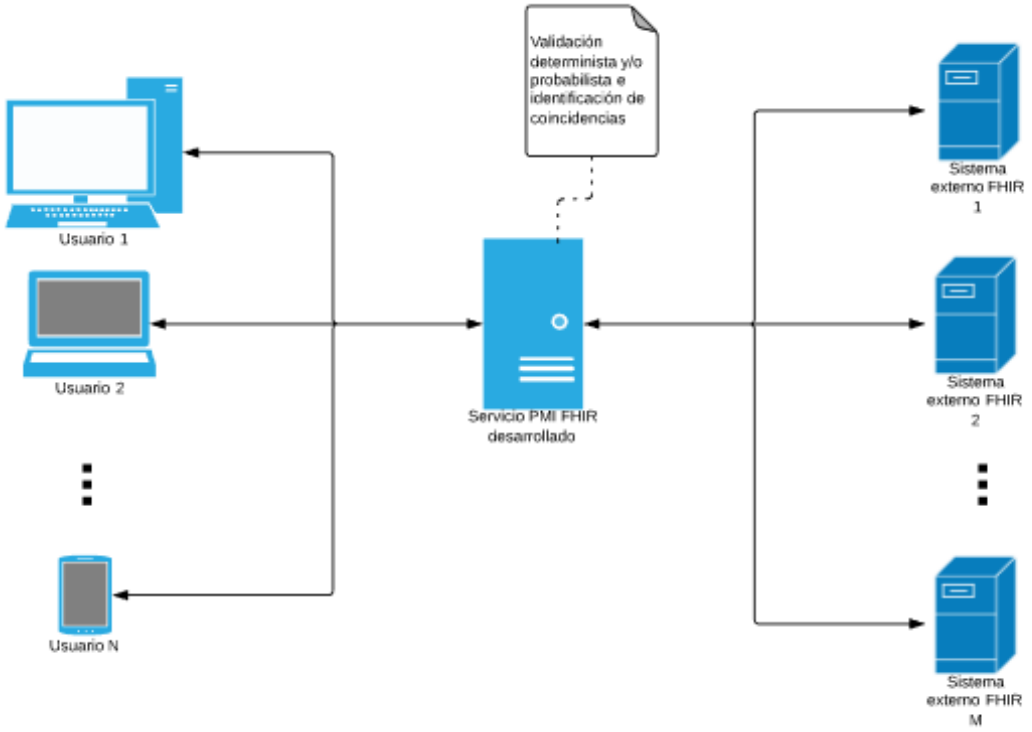


Figura 1. Esquema general del servicio desarrollado

2 ESTADO DEL ARTE

SE hace indispensable presentar el contexto sanitario y técnico para facilitar la comprensión del presente trabajo. Este apartado tiene como objetivo familiarizar al lector con algunos conceptos que se consideran fundamentales a la hora de diseñar e implementar el servicio.

2.1 Interoperabilidad en el ámbito de la salud

La interoperabilidad es la capacidad que tienen los diferentes sistemas de información y aplicaciones software para comunicarse, intercambiar datos y utilizar dicha información. Aplicado al dominio sanitario, puede aumentar la seguridad del paciente, pues un acceso a datos clínicos históricos en tiempo real permite atender de forma más óptima a personas desde cualquier lugar, incrementando la calidad de la asistencia. Es evidente que se requiere un frecuente intercambio de registros entre instituciones para que los profesionales del sector puedan ofrecer una atención de mayor rendimiento. (1)

Cada sistema debe actuar como una caja negra, puede estar implementado con cualquier tecnología, pero debe recibir y suministar la información de forma que pueda ser procesada y entendida fácilmente por el resto. Para que esta comunicación sea posible, es necesario que se comparta un marco de referencia común a través de algún estándar de interoperabilidad.

Los estándares definen la denominación y terminología de los diferentes componentes del sistema y la forma de enviar los datos, para garantizar así un intercambio de información.

2.2 Patient Master Index (PMI)

PMI (Patient Master Index) es un concepto a un sistema que reúne registros de información demográfica de pacientes de multiples sistemas sanitarios, resuelve el identificador de una persona en distintos sistemas a partir de su información demográfica y puede memorizar la relación entre estos identificadores, mediante diferentes algoritmos de búsqueda. Las posibles coincidencias localizadas son medidas y evaluadas para determinar si existe o no duplicación. Es básicamente una funcionalidad parecida la que se quiere implementar en este servicio, manejando únicamente recursos tipo person. Una vez solucionado este problema, se puede proceder a la búsqueda de historiales, pero esto es algo que no se va a desarrollar en este trabajo.

2.3 Identificación de recursos coincidentes

En ocasiones aparecen historias clínicas duplicadas para el mismo paciente o información que no le pertenece debido a una incorrecta identificación, lo que pone en riesgo la seguridad del paciente. Un correcto diseño del servicio de identificación de pacientes puede mitigar estos problemas.

Existen complejos algoritmos que buscan la optimización en la búsqueda e identificación de registros. Este concepto es conocido como “Data Matching” (6) (coincidencia de datos), una tarea muy compleja y muy sensible a la calidad de los datos.

La técnica de “Referential Matching” (7) consiste en comparar los datos demográficos de cada registro con una base de datos de identidades completas y continuamente actualizada, en lugar de comparar directamente los datos demográficos de dos pacientes. Esta herramienta proporciona un gran salto en la tecnología y precisión de comparación. Existe en EEUU una base de datos como la mencionada anteriormente que abarca toda la población. Sin embargo, en España, no tiene sentido su utilización debido a que actualmente no existe una base de datos similar.

Las bases de datos que son objeto de análisis de coincidencia de datos pueden ser muy grandes. En el peor de los casos, cada registro de una base de datos debe compararse con todos los demás registros de la otra. Tareas como estas pueden ser extremadamente costosas en términos de computación y difíciles de realizar en un tiempo factible, ya que este enfoque involucra un algoritmo de complejidad cuadrática. Para lidiar con esta situación y hacer que la coincidencia de datos sea escalable, se puede aplicar algún tipo de técnica de indexación para reducir el número de pares de registros que se compararán, sin que afecte a la calidad de los pares coincidentes identificados, algo que no siempre es sencillo. Por lo tanto, una indexación óptima busca reducir el número de comparaciones sin que dicha disminución empeore el número de registros duplicados detectados.

2.3.1 Coincidencia determinista realizada por el servicio

En la coincidencia determinista realizada por el servicio, se comparan los datos demográficos de los registros utilizando una comparación exacta entre los campos demográficos de los pacientes. Existen por lo tanto dos posibles resultados, positivo o negativo. El porcentaje de resultados positivos indica la probabilidad de coincidencia entre dos registros. Esta comparación no es especialmente segura, ya que en algunos casos los datos pueden incluir faltas de ortografías, diferente formato de presentación, estar vacíos, etc.

2.3.2 Coincidencia probabilista realizada por el servicio

En la comparación probabilista realizada por el servicio, se comparan los datos demográficos de los registros y se asigna a cada campo un peso que indica el nivel de similitud entre ambos. La suma de las ponderaciones de los campos individuales indica la probabilidad de una coincidencia entre dos registros. Esta comparación proporciona una mayor flexibilidad y tolerancia frente a posibles errores ortográficos y distintos formatos de presentación, por ejemplo, al indicar la dirección completa del domicilio de una persona (Calle San Jacinto nº5, C/ San Jacinto número 5, San Jacinto 5, etc).

2.3.3 Concepto Perfil Genérico

Se considera fundamental que el lector entienda el concepto de perfil genérico para facilitar la comprensión de la implementación del proyecto.

Para la búsqueda de un recurso de tipo *Person* (explicado en apartado 2.4.2.2) que cumpla ciertos parámetros de búsqueda, se requiere una primera lista amplia de posibles candidatos, a partir de la cual se realizará a cada candidato una ponderación determinista y/o probabilista del resto de datos demográficos para obtener un nivel de confianza. Los candidatos que no superen un cierto umbral mínimo configurado no serán devueltos como resultado.

Para obtener esta primera lista, se hace uso del concepto que se ha denominado como perfil genérico (9), realizando una búsqueda a partir de un subconjunto de los parámetros de búsqueda solicitados. Para seleccionar este subconjunto es necesario tener en cuenta las siguientes características:

- Validez: ¿Se sabe que este parámetro es correcto? La información demográfica del paciente que consta de valores por defecto o temporales está completa pero no es válida.

- Distinción: ¿Es el parámetro capaz de identificar de manera única a una persona? Por ejemplo, un parámetro como el sexo (es decir, el género administrativo) no está asociado a un solo individuo, mientras que un parámetro como el número de teléfono o el DNI sí es identificativo.

- Comparabilidad: ¿Es el parámetro estructurado, codificado (o numérico), o es texto libre en formato de cadena? Una dirección es un ejemplo de un parámetro relativamente difícil de comparar, mientras

que un número de seguro social (SSN) puede ser más fácil de comparar.

- Estabilidad: ¿Cuánto permanece constante el parámetro a lo largo de la vida del paciente? Aunque existen ejemplos de lo contrario, los parámetros como el género, la fecha de nacimiento y el número de la Seguridad Social tienden a ser relativamente inmutables en el tiempo. Otros parámetros, como la dirección actual, tienden a cambiar con relativa frecuencia.

A continuación, se aportan los datos de un análisis realizado por "The Sequoia Project" (9), donde se evaluaron siete combinaciones de parámetros para determinar su éxito previsto en términos de integridad y singularidad. La tabla se ha simplificado, pero sigue siendo "direccionalmente correcta" para estas siete combinaciones de rasgos del paciente. El valor de integridad (completeness) indica el porcentaje de los parámetros combinados que tenían todos los datos presentes en los registros de los pacientes, mientras que el valor de singularidad (uniqueness) representa el porcentaje de búsquedas que dieron como resultado una única coincidencia, identificando a una sola persona.

Analysis of Patient Trait Combinations

Sequence	Combination of Traits	Completeness	Uniqueness
1	FN+LN+DoB	98.2%	95.7%
2	FN+LN+DoB+Sex	98.2%	95.9%
3	FN+LN+DoB+Sex+ZIP(first 5)	91.1%	99.2%
4	FN+LN+DoB+Sex+Phone	76.2%	99.5%
5	FN+LN+DoB+Sex+MN	59.9%	98.9%
6	FN+LN+DoB+Sex+MN(initial)	60.0%	97.7%
7	FN+LN+DoB+Sex+SSN(last 4)	61.9%	99.7%

Table 7: Analysis of Patient Trait Combinations

Figura 2. Análisis de combinación de parámetros de búsqueda

FN: Nombre (First Name)

LN: Apellido (Last Name)

DoB: Fecha de nacimiento (Date of Birth)

Sex: Género (Administrative Gender)

ZIP: Código Postal (Five-Digit zip)

Phone: Teléfono

MN: Segundo nombre (Middle Name)

SSN: Numero de seguridad social (Social Security Number)

2.4 El estándar HL7 FHIR

Este proyecto se basa en el uso del estándar FHIR (2) de la organización HL7 que define una interfaz REST (apartado 2.4.1). HL7 (Health Level Seven) es una organización internacional nacida en EEUU de desarrollo

de estándares globales para facilitar el intercambio electrónico de información sanitaria. En definitiva, su misión es lograr una interoperabilidad clínica real entre los diferentes sistemas de información presentes en las organizaciones de salud. FHIR (2) es una nueva especificación basada en el enfoque de la industria actual emergente, obtenida a través del aprendizaje y la experiencia en la definición e implementación de estándares anteriores, HL7 v2 (3), HL7 v3 (4), RIM (10) y CDA (5). Este nuevo estándar, se puede usar de forma independiente o asociado con los más utilizados, gracias a mecanismos incorporados para la trazabilidad a RIM (Reference Information Model) y otros modelos de contenido importantes.

FHIR persigue la simplificación en la integración sin poner en riesgo la integridad de la información. Aprovecha los diferentes modelos lógicos para proporcionar un mecanismo consistente, fácil de implementar y riguroso para intercambiar datos entre aplicaciones de atención sanitaria. Se ha desarrollado en base a los siguientes requisitos, que eliminan las debilidades de los estándares anteriores:

- Fácil de implementar con una baja curva de aprendizaje, incluso para programadores con poca experiencia, reduciendo de forma notable los tiempos y costes de los proyectos.
- Semántica firme y robusta.
- Los artefactos tienen sentido a la vista humana y pueden ser validados electrónicamente.
- Integrable con tecnologías modernas de comunicación basadas en web, para facilitar dicha integración con los sistemas más actuales.

Como se ha comentando en el listado de requisitos, FHIR está diseñado específicamente para utilizar HTTP. El intercambio se basa en estructuras XML o JSON que utilizan un protocolo REST basado en HTTP (en contraposición a los servicios basados en SOAP que se pueden encontrar en la mayoría los sistemas de este tipo). La API RESTful (11) describe tanto los recursos como el conjunto de operaciones en dichos recursos donde las instancias individuales se administran en colecciones de su correspondiente tipo.

Es importante recalcar que FHIR no es un protocolo de seguridad, ni define ninguna función relacionada con ella. Sin embargo, sí define los protocolos que deben utilizarse para un intercambio de información seguro (12). El estándar propone una serie de medidas de seguridad que pueden ser tenidas en cuenta por los implementadores.

2.4.1 Rest Web Services

Los servicios web, empleados especialmente en desarrollos de sistemas distribuidos, es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

REST (10) define un conjunto de principios arquitectónicos por los que se pueden diseñar servicios Web que se centran en los recursos de un sistema, lo que incluye la forma en que los estados de los recursos se intercambian a través de HTTP por clientes que pueden estar escritos en diferentes lenguajes. Se caracteriza por no tener estado, es decir, el servidor no es capaz de recordar el estado de la anterior solicitud REST que pudo, o no, hacer un cliente, simplicando realmente la implementación.

En este proyecto se desarrolla un servicio web REST, que responderá a las peticiones de los usuarios, independientemente de la tecnología que utilice cada cliente en su implementación interna (su interfaz tiene que ser REST).

2.4.2 Recursos FHIR

Un concepto importante en REST es la existencia de elementos de información denominados "Recursos" (13). Estos recursos, en la especificación FHIR, se refieren a representaciones de conceptos empleados en el contexto de la salud (paciente, medicación, observación, etc), que funcionan como bloques de construcción que permiten componer estructuras de mensajes y/o documentos. Además, esta especificación define varias formas diferentes de intercambiar los recursos.

Todos los recursos tienen las siguientes características comunes:

- Se define una ruta (URI completa) a través de la cual se puede acceder al correspondiente recurso.
- El contenido está formado por un conjunto de elementos de datos estructurados tal y como se describe en la definición del propio recurso (formato XML o JSON)
- Existe un mecanismo de extensión que permite al implementador añadir nuevas propiedades con gran facilidad.
- Incluye la identificación de la version que cambia si el contenido del recurso es modificado.

La documentación de la especificación FHIR para cada recurso (13), contiene una descripción detallada, definiciones formales de los elementos, ejemplos, mapeos (para HL7 V3 y HL7 V2.x) y perfiles.

Cada recurso contiene un elemento identificativo, *id*, que contiene la identificación lógica del recurso, asignado por el servidor que lo almacena. Esta identificación es única en el espacio de todos los recursos del mismo tipo en dicho servidor y nunca se cambia, una vez que ha sido asignada. No se debe confundir este elemento con el de tipo *Identifier*, que identifica a la entidad en el mundo real de manera consistente en todos los contextos de uso. Este tipo de elemento se conoce como identificador comercial y podría ser para una persona, por ejemplo, el documento nacional de identidad.

Se puede observar en la siguiente captura, una parte de la definición del recurso *Person*, donde se han resaltado los dos elementos identificativos comentandos en el párrafo anterior.

Name	Flags	Card.	Type	Description & Constraints
Person	TU		DomainResource	A generic person record Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier		0..*	Identifier	A human identifier for this person
name	Î£	0..*	HumanName	A name associated with the person
telecom	Î£	0..*	ContactPoint	A contact detail for the person
gender	Î£	0..1	code	male | female | other | unknown AdministrativeGender (Required)
birthDate	Î£	0..1	date	The date on which the person was born
address		0..*	Address	One or more addresses for the person
photo		0..1	Attachment	Image of the person
managingOrganization	Î£	0..1	Reference(Organization)	The organization that is the custodian of the person record
active	?! Î£	0..1	boolean	This person's record is in active use
link		0..*	BackboneElement	Link to a resource that concerns the same actual person
target		1..1	Reference(Patient | Practitioner | RelatedPerson | Person)	The resource to which this actual person is associated
assurance		0..1	code	level1 | level2 | level3 | level4 IdentityAssuranceLevel (Required)

Figura 3. Contenido del recurso Person

En la definición de cada recurso, se pueden observar los campos donde va almacenada la información. Estos campos son de un tipo específico y varían segun el tipo de recurso que se está manejando. En la figura anterior, el recurso *Person* contiene entre otros, los campos *Name* y *telecom,* que son de tipos *HumanName* y

ContactPoint.

La especificación FHIR define un conjunto de tipos de datos (14) que se dividen en cuatro categorías:

- Tipos simples / primitivos, que son elementos individuales con un valor primitivo.

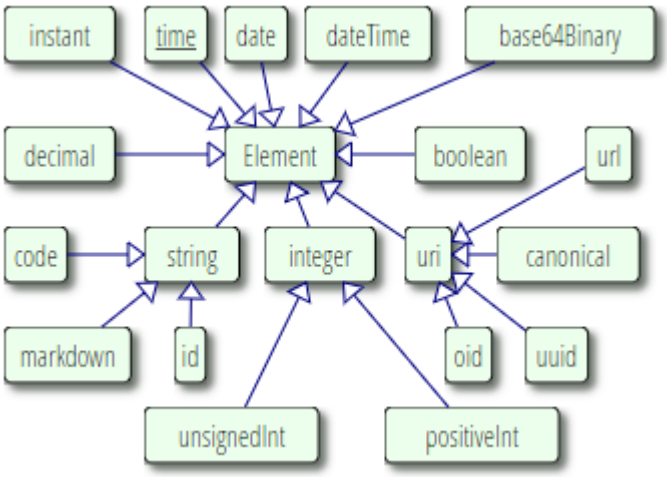


Figura 4.Tipos simples en especificación FHIR

- Tipos complejos de uso general, formado por elementos secundarios.

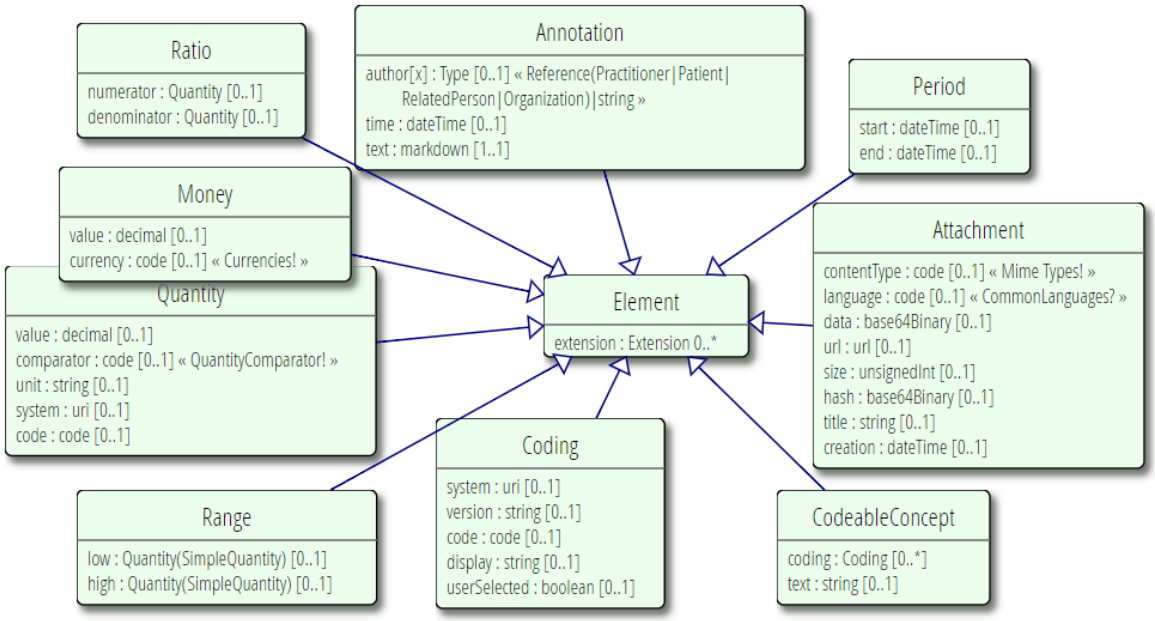


Figura 5. Tipos complejos en especificación FHIR

- Tipos de metadatos, que son un conjunto de tipos utilizados para los grupos de datos que describen el contenido informativo de un recurso..

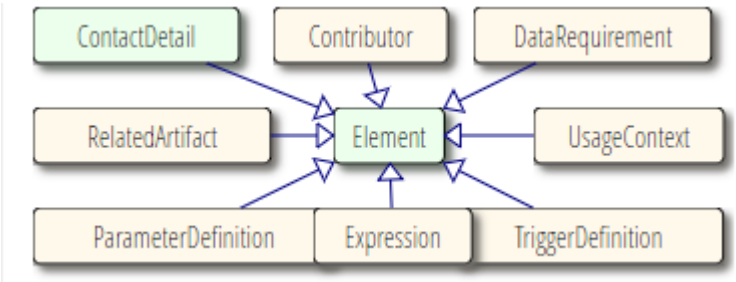


Figura 6. Tipos de metadatos en especificación FHIR

- Tipos de datos de propósito especial, definidos en otra parte en la especificación para usos específicos.

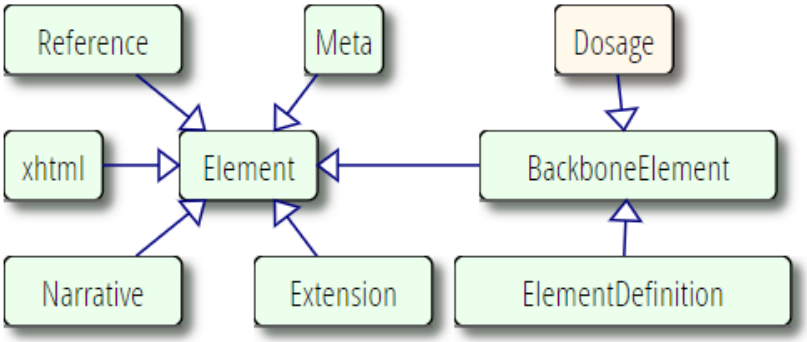


Figura 7. Tipos de datos de propósito especial en especificación FHIR

2.4.2.1 Bundle

Bundle (15) es un contenedor que contiene una colección de recursos. Este tipo de recurso es útil en un conjunto de diferentes contextos. En este proyecto, será utilizado para devolver el conjunto de recursos que cumplan los criterios de búsqueda indicados en una petición (search) al servidor RESTFul.

Un conjunto de resultados de búsqueda consiste en una serie de cero o más entradas (*entry*). Cada elemento de entrada DEBE contener un recurso. Además, *Bundle.total* se puede usar para devolver el número total de recursos que coinciden con la búsqueda realizada.

Para cada entrada, un conjunto de búsqueda también puede contener elementos específicos de información relacionados con la búsqueda:

- *search.mode*: una indicación de si el recurso está en el conjunto de resultados porque coincidió con los criterios de búsqueda o si se incluye porque otro recurso se refiere a él (por ejemplo, mediante el parámetro _include).
- *search.score*: la puntuación de clasificación de búsqueda del servidor para la entrada. No se requiere que los servidores devuelvan una puntuación de clasificación, pero si lo hacen 1 es más relevante y 0 es menos relevante.

Structure | UML | XML | JSON | Turtle | R3 Diff | All

Structure

Name	Flags	Card.	Type	Description & Constraints
Bundle	Σ I N		Resource	Contains a collection of resources + *Rule: total only when a search or history* + *Rule: entry.search only when a search* + *Rule: entry.request mandatory for batch/transaction/history, otherwise prohibited* + *Rule: entry.response mandatory for batch-response/transaction-response/history, otherwise prohibited* + *Rule: FullUrl must be unique in a bundle, or else entries with the same fullUrl must have different meta.versionId (except in history bundles)* + *Rule: A document must have an identifier with a system and a value* + *Rule: A document must have a date* + *Rule: A document must have a Composition as the first resource* + *Rule: A message must have a MessageHeader as the first resource* Elements defined in Ancestors: id, meta, implicitRules, language
identifier	Σ	0..1	Identifier	Persistent identifier for the bundle
type	Σ	1..1	code	document | message | transaction | transaction-response | batch | batch-response | history | searchset | collection BundleType (Required)
timestamp	Σ	0..1	instant	When the bundle was assembled
total	Σ I	0..1	unsignedInt	If search, the total number of matches
link	Σ	0..*	BackboneElement	Links related to this Bundle
relation	Σ	1..1	string	See http://www.iana.org/assignments/link-relations/link-relations.xhtml#link-relations-1
url	Σ	1..1	uri	Reference details for the link
entry	Σ I	0..*	BackboneElement	Entry in the bundle - will have a resource or information + *Rule: must be a resource unless there's a request or response* + *Rule: fullUrl cannot be a version specific reference* This repeating element order: For bundles of type 'document' and 'message', the first resource is special (must be Composition or MessageHeader respectively). For all bundles, the meaning of the order of entries depends on the bundle type
link	Σ	0..*	see link	Links related to this entry
fullUrl	Σ	0..1	uri	URI for resource (Absolute URL server address or URI for UUID/OID)
resource	Σ	0..1	Resource	A resource in the bundle
search	Σ I	0..1	BackboneElement	Search related information
mode	Σ	0..1	code	match | include | outcome - why this is in the result set SearchEntryMode (Required)
score	Σ	0..1	decimal	Search ranking (between 0 and 1)
request	Σ I	0..1	BackboneElement	Additional execution information (transaction/batch/history)
method	Σ	1..1	code	GET | HEAD | POST | PUT | DELETE | PATCH HTTPVerb (Required)
url	Σ	1..1	uri	URL for HTTP equivalent of this entry
ifNoneMatch	Σ	0..1	string	For managing cache currency
ifModifiedSince	Σ	0..1	instant	For managing cache currency
ifMatch	Σ	0..1	string	For managing update contention
ifNoneExist	Σ	0..1	string	For conditional creates
response	Σ I	0..1	BackboneElement	Results of execution (transaction/batch/history)
status	Σ	1..1	string	Status response code (text optional)
location	Σ	0..1	uri	The location (if the operation returns a location)
etag	Σ	0..1	string	The Etag for the resource (if relevant)
lastModified	Σ	0..1	instant	Server's date time modified
outcome	Σ	0..1	Resource	OperationOutcome with hints and warnings (for batch/transaction)
signature	Σ TU	0..1	Signature	Digital Signature

Figura 8. Contenido del recurso Bundle

El elemento *entry,* puede contener en el campo *response.outcome*, un recurso de tipo *OperationOutcome* (explicado en el apartado 2.4.2.3), para indicar posibles errores o advertencias acerca de la búsqueda realizada.

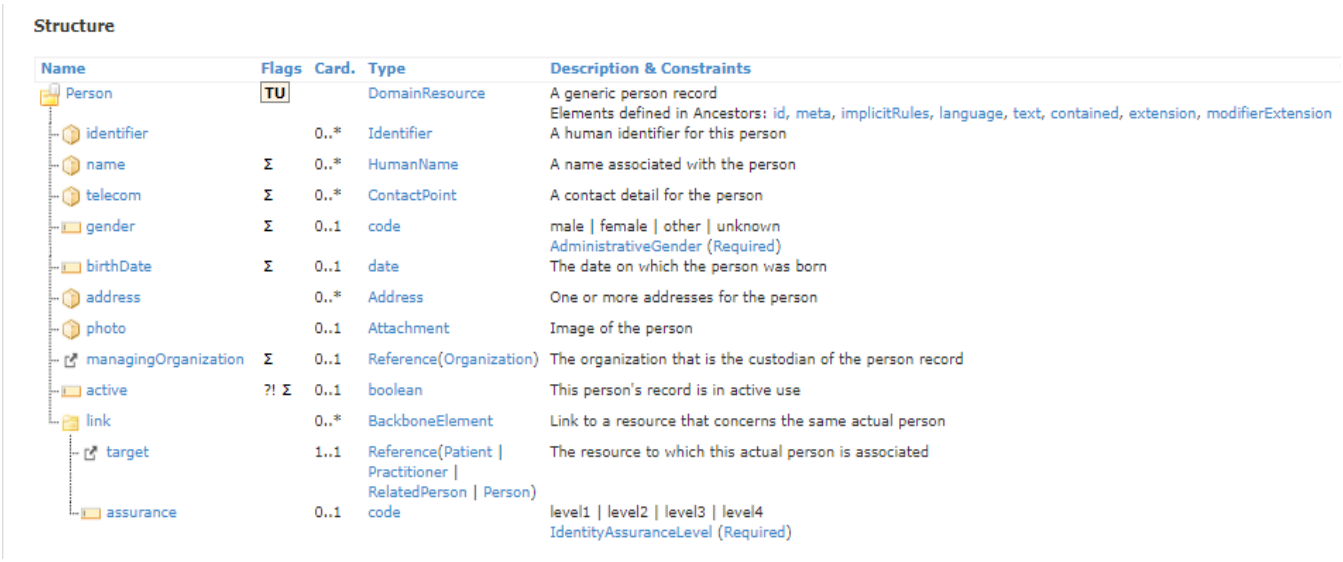
La respuesta devuelta por el servicio es un recurso *Bundle* que cumple con la especificación FHIR y que contiene una lista de las personas que concuerdan con la búsqueda realizada, representadas por un determinado nivel de confianza. Aparte de los metadatos iniciales, el recurso contendrá un conjunto de entradas (campos tipo *entry)* con los diferentes recursos tipo *person* y sus correspondiente campos *link* (donde se detalla tanto la URL completa del candidato en el sistema FHIR como el nivel de confianza). Esta respuesta está detallada más adelante en el apartado 4.4 (Comunicación con el servicio web).

2.4.2.2 Person

Person (16) es el único recurso que se va a tratar en este proyecto, realizando búsquedas en todos los servidores y devolviendo los resultados agrupados en el recurso *Bundle*.

Proporciona un conjunto de referencias de datos demográficos comunes para un individuo que puede tener diferentes roles. Los enlaces contenidos pueden ser directos (ruta completa) a recursos FHIR específicos del rol (paciente, profesional y persona relacionada) que residen en el mismo o posiblemente distintos sistemas.

Se observa en la siguiente figura la definición del recurso Person. El implementador puede consultar en dicha definición los diferentes campos que estructuran la información, el tipo de cada uno de ellos y su cardinalidad.



Structure

Name	Flags	Card.	Type	Description & Constraints
Person	TU		DomainResource	A generic person record Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier		0..*	Identifier	A human identifier for this person
name	Σ	0..*	HumanName	A name associated with the person
telecom	Σ	0..*	ContactPoint	A contact detail for the person
gender	Σ	0..1	code	male | female | other | unknown AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date on which the person was born
address		0..*	Address	One or more addresses for the person
photo		0..1	Attachment	Image of the person
managingOrganization	Σ	0..1	Reference(Organization)	The organization that is the custodian of the person record
active	?! Σ	0..1	boolean	This person's record is in active use
link		0..*	BackboneElement	Link to a resource that concerns the same actual person
target		1..1	Reference(Patient | Practitioner | RelatedPerson | Person)	The resource to which this actual person is associated
assurance		0..1	code	level1 | level2 | level3 | level4 IdentityAssuranceLevel (Required)

Figura 9. Contenido del recurso Person

En la sección 4.4 se detalla la única operación GET permitida por el servicio REST implementado, accediendo únicamente al recurso *Person.* El usuario introducirá en la petición los parámetros demográficos permitidos para el recurso *person* (apartado 2.4.3.1) y el servicio, tras una búsqueda en cada uno de los sistemas FHIR de la federación, devolverá en un recurso *bundle* (2.4.2.1) una lista de candidatos ponderados según el nivel de confianza.

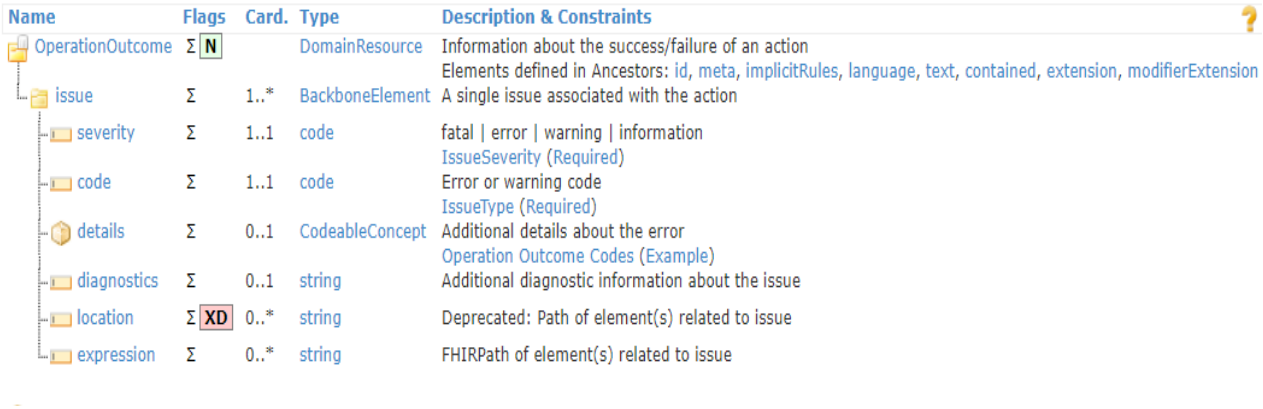
2.4.2.3 OperationOutcome

El recurso OperationOutcome (17) se refiere a un conjunto de mensajes de error, advertencia e información que proporcionan información detallada sobre el resultado de un intento de operación del sistema. Pueden existir uno o varios mensajes de error en el mismo recurso (campo *issue*). Se generan como respuesta directa del sistema, o componente de uno, y proporcionan información sobre el resultado de la operación.

El recurso OperationOutcome se utilizará en este proyecto en las siguientes circunstancias:

- Cuando falla una interacción u operación REST.
- Como parte de una respuesta de mensaje, generalmente cuando el mensaje no se ha procesado correctamente.
- Como parte de la respuesta en un paquete de búsqueda (*Bundle*) que contiene información sobre la búsqueda realizada

.



Name	Flags	Card.	Type	Description & Constraints
OperationOutcome	Σ N		DomainResource	Information about the success/failure of an action Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
issue	Σ	1..*	BackboneElement	A single issue associated with the action
severity	Σ	1..1	code	fatal | error | warning | information IssueSeverity (Required)
code	Σ	1..1	code	Error or warning code IssueType (Required)
details	Σ	0..1	CodeableConcept	Additional details about the error Operation Outcome Codes (Example)
diagnostics	Σ	0..1	string	Additional diagnostic information about the issue
location	Σ XD	0..*	string	Deprecated: Path of element(s) related to issue
expression	Σ	0..*	string	FHIRPath of element(s) related to issue

Figura 10. Contenido del recurso OperationOutcome

Los campos *issue.severity* e *issue.code* indican el tipo de error/advertencia/información aportada por el servidor. Además, mediante *issue.diagnostics,issue.location* e *issue.expression* se puede generar una información más ampliada.

2.4.3 Operación Search

La búsqueda de recursos es fundamental en el estándar FHIR. Las operaciones de búsqueda realizan un filtrado de recursos a partir de los parámetros suministrados en dicha operación. Los implementadores sólo necesitan implementar la complejidad que requieran para su proyecto, pues existe un conjunto amplio de operaciones definidas.

En el caso más simple, una búsqueda se ejecuta realizando una operación GET en el marco RESTful:

GET [base]/[type]?name=value&...{&_format=[mime-type]}}

El servidor determina qué conjunto de recursos cumple los criterios especificados y devuelve los resultados en una respuesta HTTP en un recurso *bundle* que incluye el conjunto obtenido en la búsqueda. Existen parámetros de búsqueda que se aplican a todos los tipos de recursos y otros que son específicos para cada tipo.

Las operaciones de búsqueda se ejecutan en uno de los tres contextos definidos que controlan qué conjunto de recursos se están buscando:

- Un tipo de recurso específico:

  GET [base]/[type]?parameter(s)

- Un compartimento específico, quizás con un tipo de recurso específico en ese compartimiento

  GET [base]/Person/[id]/[type]?parameter(s)

  Un compartimento es una agrupación lógica de recursos que comparte una propiedad común. En la sentencia mostrada como ejemplo, se refiere al conjunto de recursos tipo *person* asociados con una persona particular, estando incluidos en la búsqueda los recursos de tipo *type* a los que dicha persona hace referencia y viceversa.

- Todos los tipos de recursos:

  GET [base]?parameter(s)

Los siguientes parámetros se pueden aplicar a todos los recursos: _contenido, _id, _lastUpdated, _profile, _query, _security, _source, _tag, _text y _filter. Además, cada tipo de recurso define su propio conjunto de parámetros de búsqueda con sus nombres, tipos y significados. Estos parámetros de búsqueda están en la misma página que las definiciones de recursos y también se publican como parte de la declaración de

capacidad estándar. Para *Person,* nuestro recurso de interés, los parámetros de búsquedas son los indicados en el punto 2.4.3.1.

Además de los parámetros mencionados anteriormente, existen prefijos, modificadores, encadenado de parámetros, caracteres de escape, búsqueda por listas, por tipo de recurso, etc, que no se utilizarán en el desarrollo del proyecto y que por lo tanto no se considera necesario comentar en esta sección. Todos estos elementos pueden ser consultados evidentemente en la documentación del estándar (18).

2.4.3.1 Parámetros del recurso Person

En este proyecto se implementarán únicamente los parámetros definidos para el recurso Person, detallados en la siguiente tabla.

Parámetro	Tipo	Descripcion
address	string	Busca coincidencia con cualquiera de los campos de cadena en la dirección, incluida la línea, ciudad, distrito, estado, país, código postal y / o texto.
address-city	string	Busca coincidencia con el campo ciudad de la dirección.
address-country	string	Busca coincidencia con el campo país de la dirección.
address-postalcode	string	Busca coincidencia con el campo código postal de la dirección.
address-state	string	Busca coincidencia con el campo estado de la dirección.
address-use	token	Busca coincidencia con el campo uso de la dirección.
birthdate	date	Busca coincidencia con el campo fecha de nacimiento.
email	token	Busca coincidencia con uno de los campos de contacto email.
gender	token	Busca coincidencia con el campo género.
identifier	token	Busca coincidencia con el campo de identificación.
link	reference	Busca coincidencia con el campo link.
name	string	Busca coincidencia con cualquiera de los campos de cadena en HumanName, incluyendo familia, nombre, prefijo, sufijo, sufijo y / o texto
organization	reference	Busca coincidencia con el campo organización.
patient	reference	Busca coincidencia con el campo paciente.
phone	token	Busca coincidencia con uno de los campos de contacto teléfono.
phonetic	string	Busca coincidencia en alguna parte del nombre utilizando algún tipo de algoritmo de correspondencia fonética.
practitioner	reference	Busca coincidencia con el campo practitioner.

relatedperson	reference	Busca coincidencia con el campo persona relacionada.
telecom	token	Busca coincidencia con cualquiera de los campos de contacto.

Tabla 1. Parámetros de búsqueda para el recurso Person

3 ENTORNO DE TRABAJO Y TECNOLOGÍAS

A Continuación, se listan los programas que se han empleado para el desarrollo del proyecto, además de las diferentes tecnologías elegidas para alcanzar el objetivo.

3.1 C Sharp (C#)

Lenguaje de programación utilizado para programar la mayor parte del servicio. Es un lenguaje orientado a objetos, desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes, entre ellos Delphi (19).

Aunque forma parte de la plataforma.NET, ésta es una interfaz de programación de aplicaciones (API), mientras que C♯ es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco de DotGNU - Mono que genera programas para distintas plataformas como Win32, UNIX y Linux. Para este proyecto, el servicio se ha implementado de forma que sea un código multiplataforma.

3.2 Python

Lenguaje de programación utilizado para desarrollar el script encargado de detectar posibles coincidencias entre los candidatos obtenidos en la búsqueda, debido a la gran variedad de algoritmos de gestión de coincidencias de datos (20) ya implementados que ofrece Python.

Pyhton es un lenguaje interpretado que resalta por una sintaxis especialmente legible. Se trata de un lenguaje multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Además, usa tipado dinámico y es multiplataforma.

3.3 Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows utilizado en este proyecto para el desarrollo del código. Soporta múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET MVC, Django, etc (21).

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así, se pueden crear aplicaciones desplegadas en páginas web, dispositivos móviles, dispositivos embebidos y consolas, entre otros. Además, el depurador del entorno ofrece muchas formas de ver lo que hace el código durante la ejecución. Esto permite revisar el código y fijarse en los valores almacenados en las variables, establecer inspecciones en ellas para ver cuándo cambian esos valores, examinar la ruta de ejecución del código, etc.

Es importante aclarar que NET Core es una nueva versión modular del framework .NET, que permite el uso multiplataforma de .NET. Es un subconjunto del framework .NET por lo que no tiene toda la funcionalidad del framework completo, y puede emplearse para creación de aplicaciones web, de escritorio y móviles.

Este proyecto se ha llevado a cabo mediante ASP.NET Core, un nuevo framework de código abierto y multiplataforma para la creación de aplicaciones conectadas a Internet, como los servicios web. La plantilla generada por este framework ha facilitado el desarrollo del servicio REST.

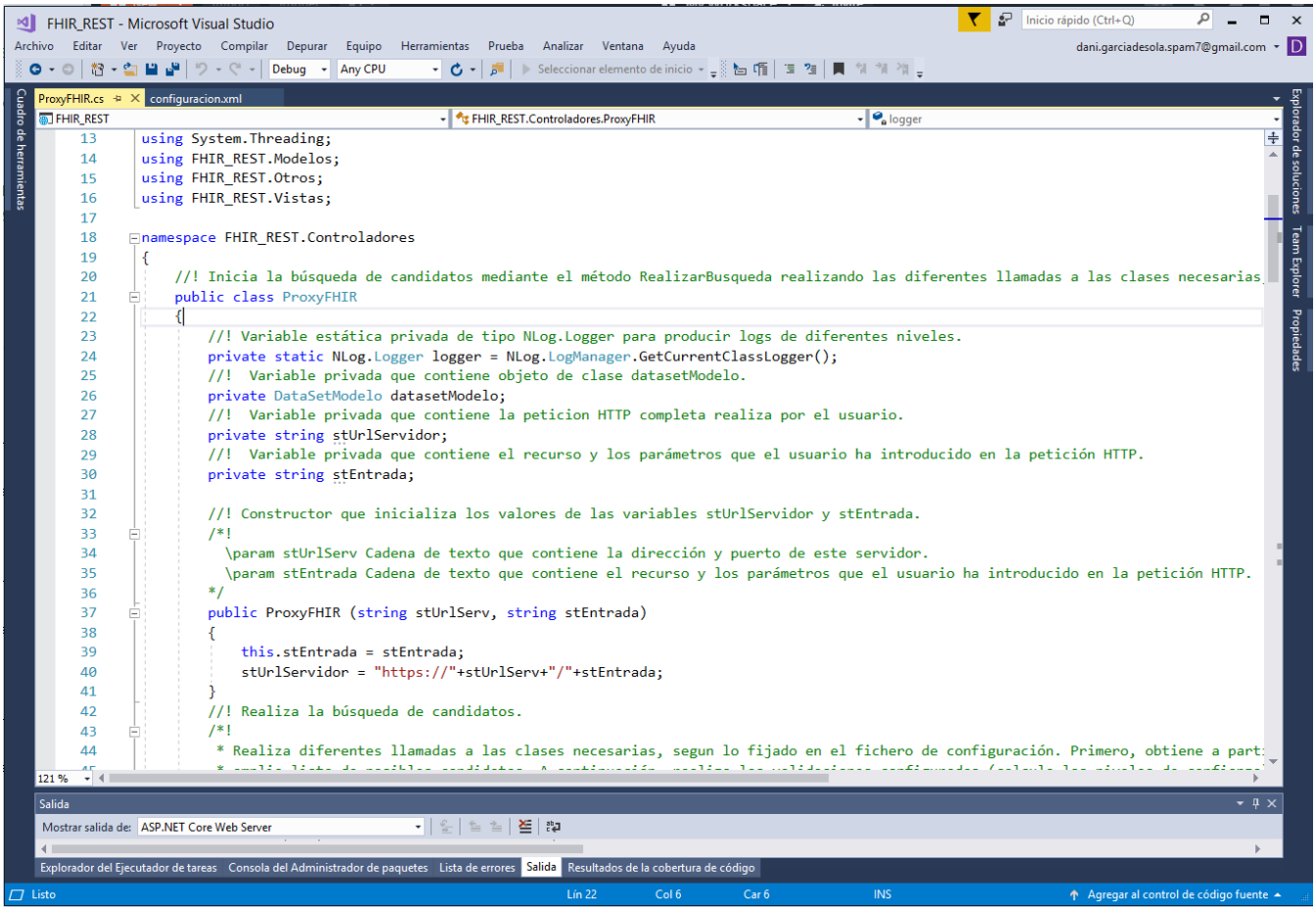


Figura 11. Microsoft Visual Studio

3.4 Servidores FHIR de ejemplo

Para el desarrollo del proyecto, se ha necesitado ejecutar varios servidores FHIR RESTFul (11) en Tomcat (explicado a continuación), con el objetivo de realizar diferentes pruebas del servicio en desarrollo, accediendo dicho servicio a estos servidores durante la operación de búsqueda para la obtención del conjunto de candidatos finales.

Los servidores ejecutados para pruebas, pertenecen a la librería HAPI FHIR (22) (24) y se pueden usar para crear un punto final de servidor FHIR contra una fuente de datos arbitraria, que podría ser una base de datos de su propio diseño, un sistema clínico existente, un conjunto de archivos o cualquier otra cosa. Sin embargo, para el desarrollo del proyecto, se ha utilizado directamente la propia base de datos incoporada en el servidor ejemplo.

Estos ejemplos admiten todas las operaciones estándar (leer, crear, eliminar, etc). Además, incluyen una instancia integrada de la base de datos Java de Apache Derby para que el servidor pueda ejecutarse sin depender de ninguna base de datos externa.

Apache Tomcat (23), a menudo denominado Tomcat Server, es un contenedor de Servlets Java de código abierto desarrollado por Apache Software Foundation (ASF). Tomcat es un software de código abierto que

implementa varias especificaciones de Java EE, incluidas Java Servlet, JavaServer Pages (JSP), Java EL y WebSocket, y proporciona un entorno de servidor web HTTP en el que se puede ejecutar el código Java.

Es importante aclarar que los servidores ejemplo explicados anteriormente están codificados en lenguaje Java, mientras que el servicio que se desarrolla en este proyecto utiliza los lenguajes c# y python.

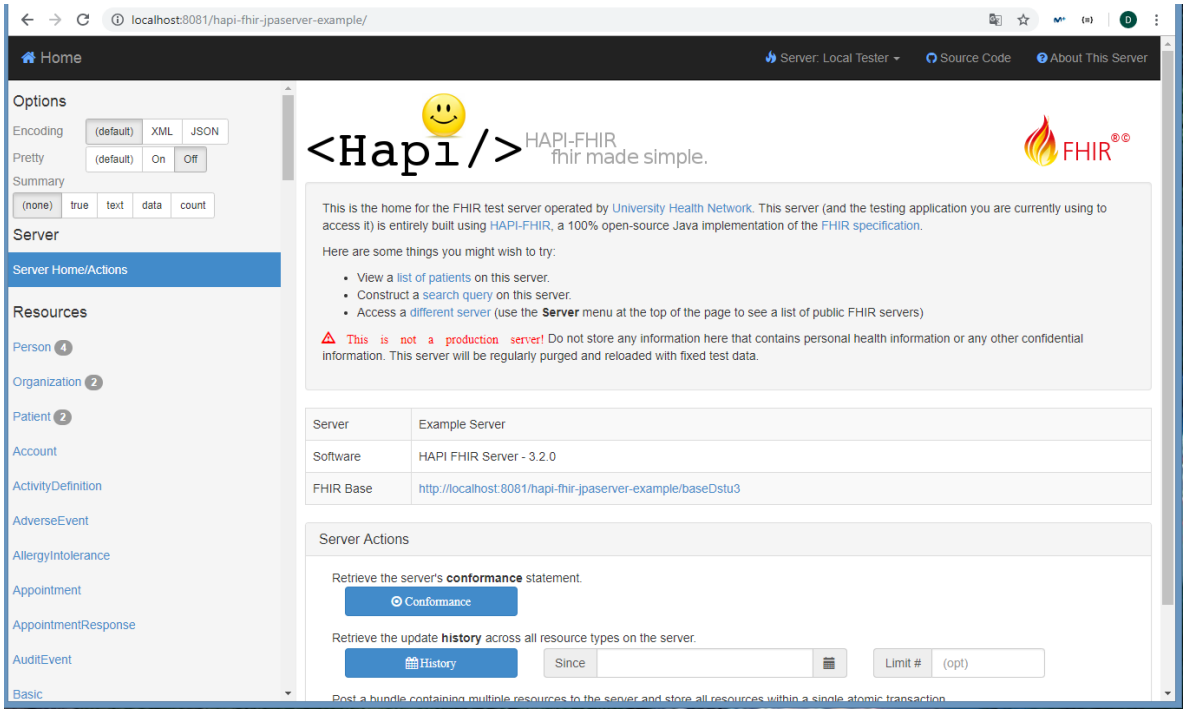


Figura 12. Servidor HAPI FHIR de ejemplo

3.5 Postman

Las peticiones GET para realizar las diferentes búsquedas han sido enviadas al servicio REST desarrollado en este proyecto mediante esta herramienta, facilitando tanto la interacción como la verificación de las respuestas. Postman (24) es un potente cliente HTTP para probar servicios web, una herramienta dirigida a desarrolladores que permite realizar peticiones a cualquier API REST. Es muy útil a la hora de programar y hacer pruebas, puesto que nos ofrece la posibilidad de comprobar el correcto funcionamiento de nuestros desarrollos. Postman facilita la prueba, el desarrollo y la documentaciónde las APIs al permitir a los usuarios agrupar rápidamente solicitudes HTTP simples y complejas para enviarlas al servidor. Ofrece un conjunto de funcionalidades que ayudarán a organizar las peticiones en colecciones, hacer y automatizar pruebas, mantener equipos sincronizados y crear Mocks de APIs. Además, proporciona una interfaz de usuario con la que hacer solicitudes sin la necesidad de escribir excesivo código.

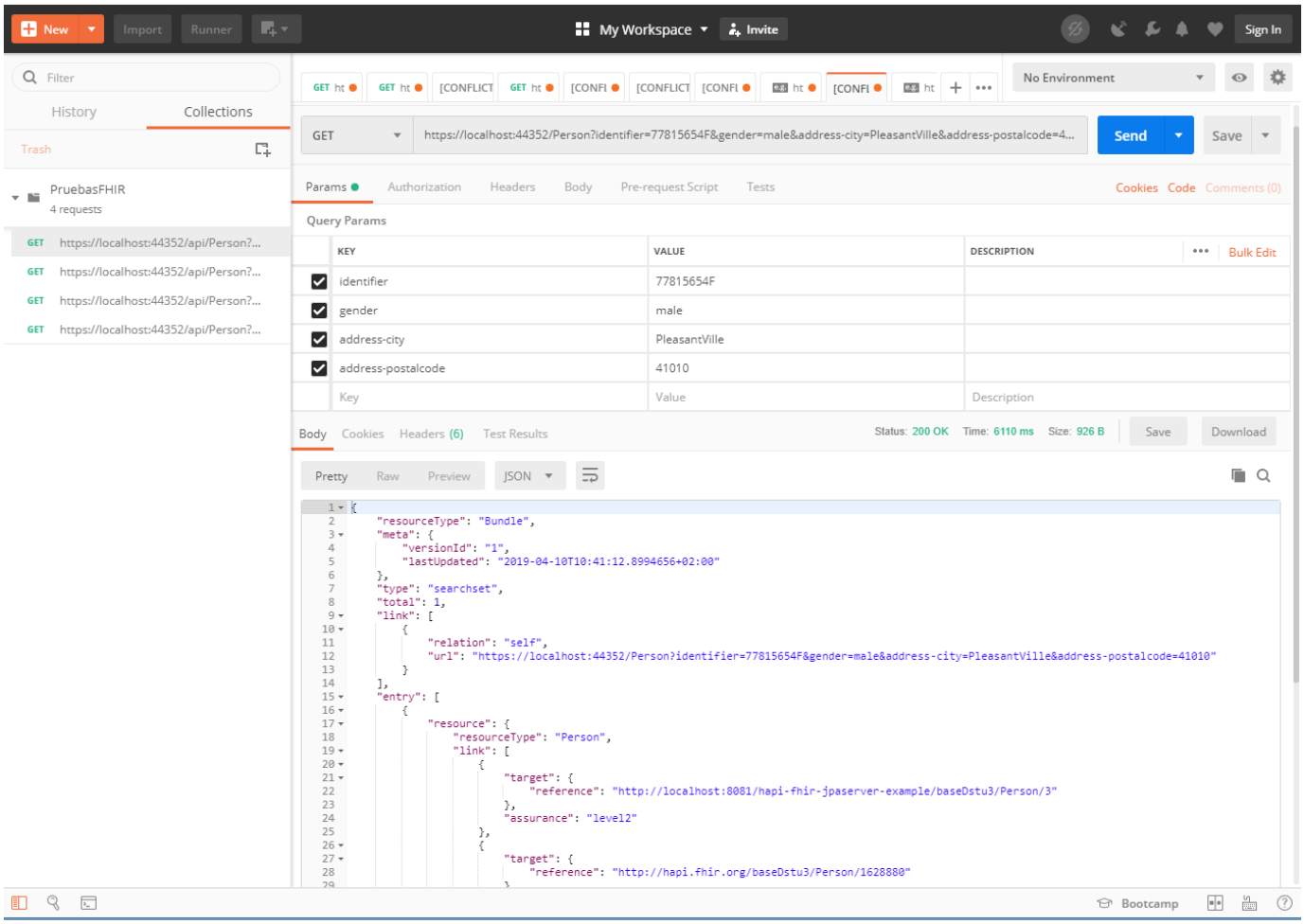


Figura 13. Interfaz Postman

3.6 JSON

Json (26) es un formato de texto sencillo para el intercambio de datos, utilizado para el servicio implementado en este proyecto. Es una forma de almacenar información de manera organizada y de fácil acceso. En pocas palabras, nos brinda una colección de datos legibles por humanos a los que podemos acceder fácilmente. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

Son estos dos formatos (Json y XML) los que soporta el estándar FHIR, siendo Json el elegido para el servicio desarrollado, tal y como se ha mencionado anteriormente. Se puede observar en la siguiente figura un ejemplo de parte de respuesta recibida por el usuario.

```
{
  "resourceType": "Bundle",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2019-03-16T20:02:28.8438156+01:00"
  },
  "type": "searchset",
  "total": 1,
  "link": [
    {
      "relation": "self",
      "url": "https://localhost:44352/Person?identifier=77815654F"
    }
  ],
  "entry": [
    {
      "resource": {
        "resourceType": "Person",
        "link": [
          {
            "target": {
              "reference": "http://hapi.fhir.org/baseDstu3/Person/1628880"
            },
            "assurance": "level4"
          }
        ]
      }
    },
```


Figura 14. Ejemplo respuesta del servicio formato JSON

3.7 Doxygen

Doxygen (27) es una herramienta estándar utilizada en este proyecto para la generación de la documentación a partir de los archivos del código fuente. Soporta lenguajes como C, C++, C#, PHP, Java, y Python entre otros. El usuario puede elegir el formato de la documentación generada, ofreciendo un conjunto de posibilidades como HTML, LaTeX y PDF. La documentación se extrae directamente del código, lo cual hace que sea mucho más fácil mantener la documentación consistente con el código fuente.

En la siguiente figura se muestra un ejemplo generado en formato HTML, perteneciente a este trabajo.

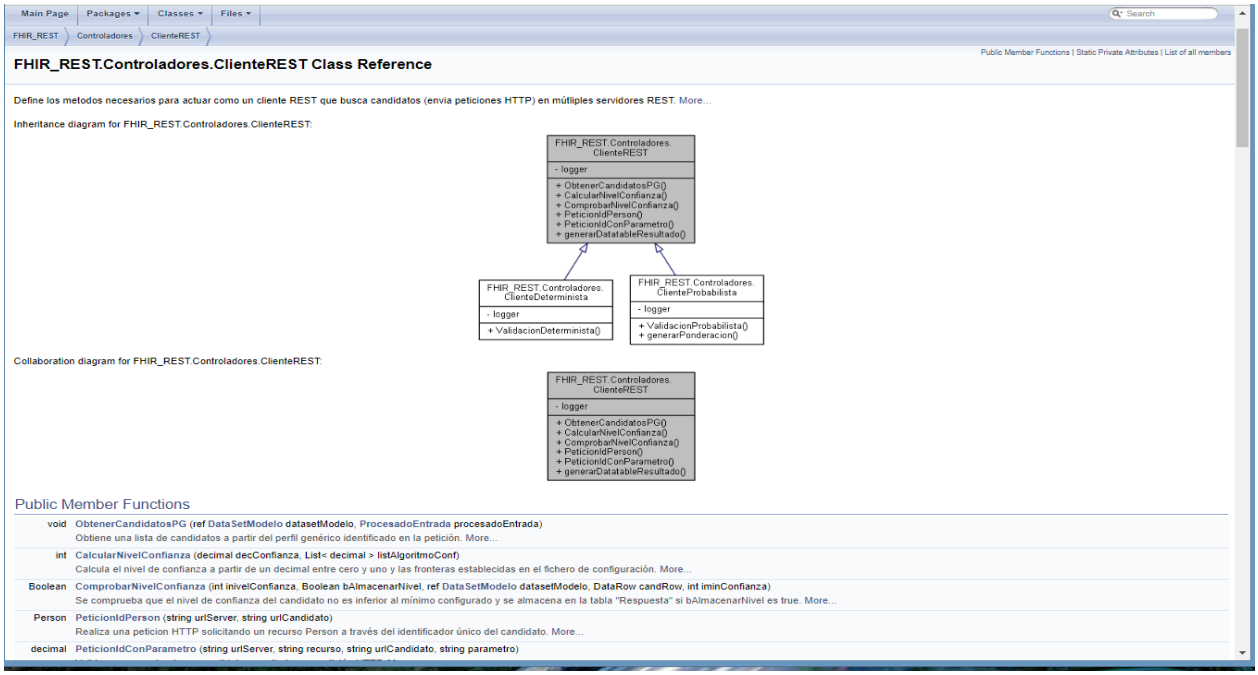


Figura 15. Documentación HTML generada por Doxygen

3.8 Librerias externas utilizadas

Se ha realizado un análisis y una búsqueda profunda de librerías que pudiesen resultar útiles para el desarrollo del proyecto (explicado más adelante en el apartado 4.6) . Finalmente, las diferentes librerías externas seleccionadas para su reutilización, tanto en lenguaje C# como Python, han sido las siguientes:

- BoomTown.FuzzySharp (28): es una biblioteca de código abierto que implementa un sistema de coincidencia de cadenas difusa para C # basada en el algoritmo FuzzyWuzzy Python. El algoritmo usa la distancia de Levenshtein para calcular la similitud entre cadenas. Es utilizada para la generación de la ponderación probabilista (en cada posible candidato) de los parámetros de búsqueda emitidos por el usuario.
- HL7.Fhir.STU3 (29): es una librería del estándar HL7 para C# en su versión STU3. Es utilizada para trabajar con el modelo de datos FHIR, para parsear dicho modelo a formato Json y para trabajar con el cliente REST.
- NLog.Web.AspNetCore (30): es un sistema de trazado de código gratuito y de código abierto para C#. Puede enviar los logs al clásico fichero, con innumerables opciones de configuración para dar nombre al fichero destino, rotar ficheros cuando alcanzan determinado tamaño y todo lo que se nos ocurra. Pero aparte de trabajar con ficheros como toda la vida también puede enviar los logs a una base de datos, al sistema de eventos de Windows, a la consola o usando la red local a un servidor en otra máquina
- Record Linkage Toolkit (31): es una biblioteca Python de código abierto para vincular registros pertenecientes a diferentes fuentes de datos. Proporciona la mayoría de las herramientas necesarias para la vinculación de registros y la deduplicación. El paquete contiene métodos de indexación, funciones para comparar registros y clasificadores. Se utiliza para el proceso final de detección de candidatos duplicados.

3.9 MagicDraw

MagicDraw (32) es una herramienta CASE (Computer Aided Software Engineering), compatible con el estándar UML y ha sido utilizada en las tareas de modelado necesarias para el diseño del servicio REST.

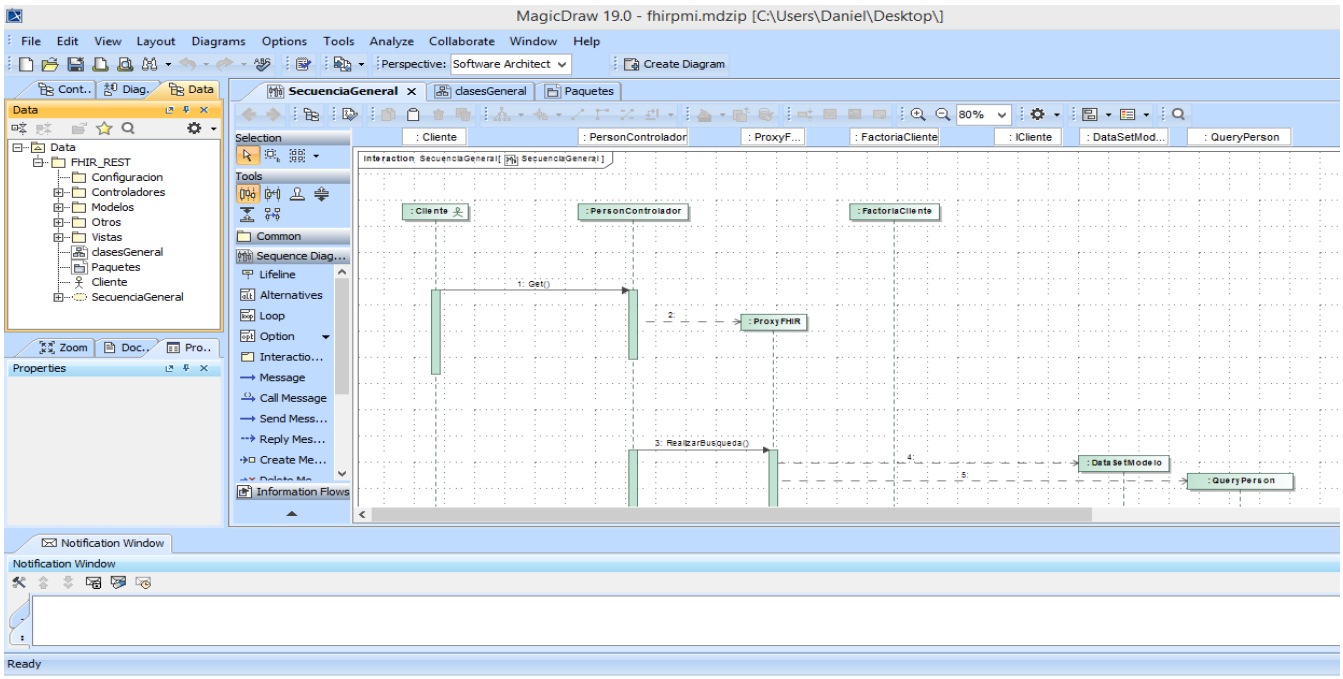


Figura 16. Interfaz MagicDraw con diagrama de secuencia

4 ANÁLISIS

En este capítulo, se detalla el análisis de los requisitos especificados, los objetivos marcados y la estructura del sistema que se va a implementar.

4.1 Introducción

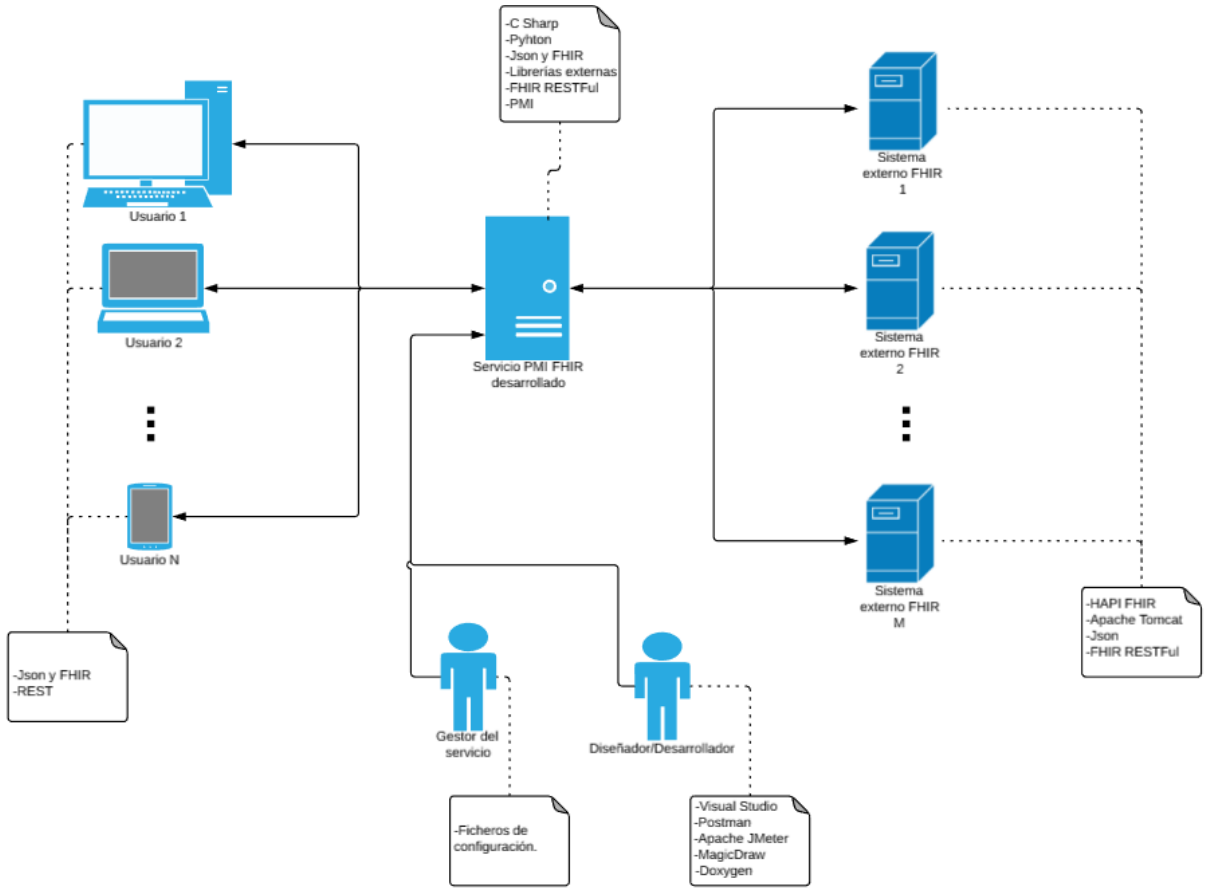


Figura 17. Esquema genérico de tecnologías utilizadas en el desarrollo del servicio

El servicio que se va a implementar en este trabajo es un Servicio Web multiplataforma, desarrollado en lenguaje C# mayoritariamente y en python para el proceso de identificación de coincidencias. A través de este servicio, los usuarios podrán centralizar las búsquedas de recursos *person* a partir de diferentes datos demográficos, obteniendo posibles candidatos de los diferentes sistemas, evaluados según su nivel de confianza y comprobando finalmente en el resultado la existencia de posibles coincidencias. Esta comparación se realizará tanto en el mismo como en diferentes sistemas. Así pues, se quiere conseguir una funcionalidad parecida la de un PMI (ver apartado 2.2) manejando únicamente recursos tipo person y siendo capaz de obtener el id que se usa en cada sistema de la federación para referirse a la misma persona. Es importante destacar que el usuario podrá acceder a esta API independientemente de la tecnología que utilice, siempre y cuando soporte HTTP y utilice la API REST proporcionada.

Este proyecto será presentado como Trabajo Fin de Grado, utilizando parte de los conocimientos adquiridos en la especialidad Telemática del Grado Ingeniería de las Tecnologías de la Telecomunicación. La organización llevada a cabo para el desarrollo se explica en este documento.

4.1.1 Propósito del capítulo

El propósito de la presente sección es ayudar al lector a entender las diferentes partes del servicio implementado, objetivos, actores que interactúan y el conjunto de requisitos que se deben satisfacer, fijando las bases que marcarán el desarrollo del proyecto.

4.1.2 Alcance

Se ha buscado la mayor parametrización posible del servicio, para que una vez en producción en sistemas reales, se estudie estadísticamente y se encuentren los valores del fichero de configuración que consigan obtener una optimización máxima tanto en la búsqueda como en el proceso de identificación de coincidencias. El perfil genérico definido para la opción de búsqueda exhaustiva, los parámetros de ponderación, el algoritmo para los niveles de confianza, la posibilidad de validación determinista, etc, han sido fijados de forma orientativa, sin una garantía de que sea la mejor posible configuración.

El servicio accederá para realizar las búsquedas a todos los sistemas que hayan sido registrados en el fichero de configuración, siempre y cuando todos cumplan con el estándar FHIR en su versión STU3.

4.1.3 Actores

A continuación se describen brevemente los actores que interaccionan con el servicio desarrollado en este proyecto.

4.1.3.1 Usuarios

Personal sanitario encargado de realizar búsquedas de diferentes personas. Introduce los diferentes parámetros demógraficos (en la petición HTTP) definidos en el estándar para el recurso *Person*.

4.1.3.2 Sistemas externos FHIR

Diferentes sistemas FHIR consultados por el servicio, fijados en el fichero *servidores.xml*. El servicio envía peticiones HTTP de operaciones tipo Search a los diferentes sistemas FHIR configurados, esperando recibir las respuestas correspondientes para procesarlas y generar resultados.

4.1.3.3 Gestor del servicio

Encargado del mantenimiento del servicio. Necesario para iniciar y parar el servicio, atender posibles incidencias y por supuesto, fijar los valores de los diferentes archivos de configuración.

4.2 Información del dominio del problema

Como se ha comentado en capítulos anteriores, la descentralización de competencias y servicios sanitarios, y el aumento de movilidad de ciudadanos hace que los diferentes proveedores de sanidad necesiten colaborar entre ellos, más allá de las posibles competencias y límites linguísticos, para poder garantizar una asistencia de calidad y seguridad al paciente. Para conseguir esta colaboración, se necesita una estandarización, y una interoperabilidad semántica, por lo que no solamente es un problema tecnológico.

La presente carencia de esa estándarización trae consigo el gran probema del intercambio de la información y la aportación de una vision parcial e incompleta de la historia clínica por parte del personal médico, lo que pone en riesgo la seguridad del paciente.

En una federación de sistemas FHIR, los sistemas guardan información de una persona determinada, pero cada uno puede utilizar distintos identificadores para referirse a dicha persona. El intercambio de estos datos y la vinculación de registros, cuyo objetivo es recopilar la información distribuida en varios sistemas que se refieran a una misma persona en el mundo real, se convierte en una necesidad muy presente en la actualidad.

Además, la incapacidad de buscar y vincular registros da lugar en muchas ocasiones a la generación de información repetida, al registrar de nuevo datos que no se han podido localizar durante dicho proceso de vinculación.

4.3 Necesidades de negocio

Existe en la actualidad una fuerte demanda de sistemas sanitarios interoperables entre sí. Sin embargo, la dificultad que conlleva la adaptación de los protocolos y sistemas actuales para que sean compatibles, muy dependiente de configuraciones y adaptaciones, hace que incluso dos sistemas que pueden parecer idénticos a simple vista sean incapaces de comunicarse.

Los pacientes reciben atención médica en distintos lugares, muchas veces incluso de diferentes organizaciones. A menudo, los identificadores utilizados para los pacientes varían entre instituciones. Cuando se realiza intercambio de información entre organizaciones sanitarias, la existencia de identificadores múltiples y diferentes sistemas de historias clínicas utilizados suponen un gran problema. Un pequeño error administrativo puede generar historias diferentes e incompletas para el mismo paciente. Para unificar la información de dicho paciente, se necesita un PMI (Patient Master Index) que consiga alinear de forma correcta la información demográfica y de identificación procedente de las diferentes fuentes. Además, la posibilidad de detectar historias clínicas duplicadas y aplicar acciones correctivas, garantiza que todos los datos de pacientes sean correctos y coherentes en toda la organización, independientemente del sistema en el que estén registrados.

FHIR consigue reducir notablemente los tiempos y costes de integración, permitiendo la convivencia e integración con sistemas actuales que emplean versiones antiguas de los protocolos HL7.

El gran apoyo creciente que está recibiendo este estándar por parte de la industria, supone el impulso definitivo al uso de FHIR como estándar para la interoperabilidad. Se espera que en un futuro muy cercano se empiece a incorporar definitivamente como parte de nuestros sistemas de información. Una vez asentado en los diferentes sistemas la existencia de un servicio que centralice todas las peticiones, con funcionalidad parecida a la de un PMI, puede facilitar notablemente el proceso de búsqueda de personas a través de parámetros demográficos. Es precisamente esta centralización de peticiones una de las soluciones que aporta este proyecto, siendo capaz de tolerar posibles pequeños fallos ortográficos introducidos en los campos como mayúsculas, tildes, nombres incompletos, teléfonos sin prefijo, etc (debido a errores administrativos) y posibles duplicaciones de registros clínicos. Además, se considera fundamental que el servicio se base en la tecnología web, cada vez más demandada por los desarrolladores en el día a día.

4.4 Comunicación con el servicio web

El cliente REST utilizado por el usuario realiza la comunicación con el servicio web por medio de peticiones HTTP. Dicha comunicación se realiza a través de RESTful y se usa únicamente un tipo de peticion:

- GET: Para consultar y obtener recursos.

A través de este tipo de petición, el usuario solo tendrá acceso al recurso *Person* tal y como se muestra en la siguiente tabla:

Petición	URI	Parámetros permitidos
GET	/Person	{ Birthdate, identifier, name, address-country, email, link, organization, phone, telecom, address, address-city, address-use, phonetic, address-postalcode, address-state, gender, patient, practitioner, relatedperson}

Tabla 2. Operaciones permitidas del servicio REST

Con esta petición, el usuario inicia el proceso de búsqueda de candidatos en todos los sistemas FHIR de la federación configurados a partir de los diferentes parámetros demográficos aportados.

La respuesta devuelta por el servicio al usuario es un recurso *Bundle* (apartado 2.4.2.1) que cumple con la especificación FHIR y que contiene un conjunto de personas que concuerdan con la búsqueda y que se representan mediante un determinado nivel de confianza. Aparte de los metadatos iniciales y de la petición que genera dicha respuesta (*link.url)*, el recurso contendrá un conjunto de entradas (campos tipo *entry).* Cada entrada, contendrá un recurso tipo *person* (apartado 2.4.2.2) y cada recurso *person* contendrá un campo *link* compuesto por los campos *link.target* (con la URL completa del candidato en el sistema FHIR externo) y *link.assurance* (nivel de confianza).


```
{
    resourceType: "Bundle",
  - meta: {
        versionId: "1",
        lastUpdated: "2019-05-03T17:55:38.3290324+02:00"
    },
    type: "searchset",
    total: 20,
  - link: [
      - {
            relation: "self",
            url: "https://localhost:44352/Person?gender=male"
        }
    ],
  - entry: [
      - {
          - resource: {
                resourceType: "Person",
              - link: [
                  - {
                      - target: {
                            reference: "http://hapi.fhir.org/baseDstu3/Person/230259"
                        },
                        assurance: "level2"
                    }
                ]
            }
        },
      - {
          - resource: {
                resourceType: "Person",
              - link: [
                  - {
                      - target: {
                            reference: "http://hapi.fhir.org/baseDstu3/Person/283862"
                        },
                        assurance: "level2"
                    }
                ]
            }
        },
      - {
          - resource: {
```


Figura 18. Ejemplo respuesta del servicio JSON

El campo *link.assurance* puede contener cuatro posibles valores (level1|level2|level3|level4) que indican una menor o mayor confianza de que el candidato devuelto se refiere realmente a la persona solicitada por parte del usuario, según el número de parámetros demográficos (introducidos por el usuario) que cumpla. En la figura anterior mostrada, se puede observar que el nivel de confianza devuelto es bajo, debido a que se ha realizado una búsqueda únicamente por género.

En el caso de que el servicio tenga que comunicar al usuario alguna incidencia ocurrida durante el proceso de búsqueda, se informará mediante un recurso tipo *OperationOutcome* (apartado 2.4.2.3), incluida dentro de una entrada (campo tipo *entry*).

```
- entry: [
    - {
        - resource: {
              resourceType: "Person",
            - link: [
                - {
                    - target: {
                          reference: "http://hapi.fhir.org/baseDstu3/Person/1628880"
                      },
                      assurance: "level4"
                  }
              ]
          }
      },
    - {
        - response: {
            - outcome: {
                  resourceType: "OperationOutcome",
                - issue: [
                    - {
                          severity: "warning",
                          code: "incomplete",
                          diagnostics: "Resultado incompleto por no poder acceder a los siguientes servidores: http://localhost:8081/hapi-fhir-jpaserver-
                          example/baseDstu3/"
                      }
                  ]
              }
```


Figura 19. Ejemplo respuesta del servicio con recurso OperationOutcome

En la figura anterior, se observa un ejemplo donde el servicio notifica al usuario de que la búsqueda está incompleta mediante un *OperationOutcome*, debido a que no ha sido posible acceder a uno de los servidores configurados.

4.5 Arquitectura de la aplicación

Para un mejor entendimiento y desarrollo del servicio, se ha dividido el proceso de búsqueda en cuatro subprocesos distintos, con el objetivo de aportar al lector una mayor comprensión del trabajo:

- **Lectura de los ficheros de configuración xml:** Este subproceso se ejecuta únicamente una vez al iniciar el servidor. Lee los diferentes ficheros de configuración (apartado 5.2.4) y almacena la información relevante en memoria.
- **Procesado de los parámetros de la petición HTTP:** Analiza la peticion HTTP, validando que los parámetros introducidos por el usuario sean correctos. Además, identifica el perfil genérico según la configuración fijada.
- **Búsqueda inicial de candidatos a partir de perfil genérico**: Se encarga de obtener a partir del perfil genérico detectado una amplia lista de posibles candidatos, realizando peticiones a cada sistema externo y almacenando únicamente la dirección del servidor y la URL completa del recurso *Person* en la tabla Candidatos.
- **Validación determinista**: Se encarga de realizar una petición HTTP por cada parámetro a los servidores donde se encuentra cada candidato. Según si hay o no respuesta, se rellena la tabla "Validaciones" con los valores uno o cero respectivamente por cada parámetro. Además, calcula el nivel de confianza. Se descartan aquellos candidatos que no superan el nivel de confianza mínimo fijado en el fichero de configuración para la validación determinista.
- **Validación probabilista:** Se encarga de realizar una petición HTTP por cada candidato, obteniendo así el recurso Person completo de cada uno de ellos. Pondera cada parámetro solicitado por el usuario, generando un decimal con valores comprendidos entre cero y uno, rellenando la tabla "Validaciones" con dicho nivel de similitud. Además, se calcula el nivel de confianza. Se descartan aquellos candidatos que no superan el nivel de confianza mínimo fijado en el fichero de configuración para la validación probabilista.
- **Proceso de identificación de coincidencias**: Se encarga de detectar posibles coincidencias entre los

candidatos finales obtenidos de los diferentes servidores, agrupando los registros coincidentes sobre el mismo recurso *Person*.

- **Generación respuesta**: Se encarga de generar un recurso *Bundle* (apartado 2.4.2.1) con la información de los candidatos finales obtenidos, agrupando en el mismo recurso *Person* (apartado 2.4.2.2) aquellos registros identificados como coincidentes.

Excepto los tres primeros subprocesos, que deben ejecutarse siempre que se vaya a realizar una consulta por parte del usuario, el resto de subsistemas podrán o no ejecutarse según si están activados/desactivados en el fichero de configuración.

4.6 Búsqueda de librerías

Se ha realizado una investigación y análisis de librerías que se pudiesen reutilizar para el desarrollo de este proyecto, que se ha decidido implementar en lenguaje C Sharp en su mayor parte. Este lenguaje se ha escogido para complementar la carencia de conocimientos sobre este lenguaje y su entorno de desarrollo adquiridos durante el grado, además de las facilidades que aporta a la hora de codificar un servicio REST.

En primer lugar, se ha investigado sobre la existencia de una implementación del estándar FHIR adaptado a este lenguaje, pues únicamente se tenia referencia de la librería HAPI-FHIR, implementado en Java. Se ha escogido finalmente la librería **HL7.Fhir.STU3** (29), adaptada al lenguaje C Sharp, y se han analizado las diferentes funcionalidades que abarca para aprovechar al máximo dichos métodos.

En segundo lugar, para la realización de la validación probabilista (ver apartado 4.7.5), donde es necesario generar un porcentaje de similitud entre el parámetro introducido por el usuario y el perteneciente al candidato que se está analizando, es necesario realizar una comparación de cadenas mediante algún algoritmo, calculando a partir del porcentaje obtenido un determinado nivel de confianza. Python ofrece una gran cantidad de librerías relacionadas con el Data Matching (apartado 2.3), por lo que tras un análisis de las diferentes opciones, se ha encontrado **BoomTown.FuzzySharp** (28), un sistema de coincidencia de cadenas difusa para C Sharp basada en el algoritmo FuzzyWuzzy de Python (el algoritmo se basa en la la distancia de Levenshtein, utilizado por muchas herramientas actuales relacionadas con la coincidencia de datos).

En tercer lugar, para la identificación de coincidencias entre todos los candidatos encontrados (identificados por su URL completa y su nivel de confianza) es necesario realizar una comparativa bastante compleja. Para este proceso, es necesario comparar todos los datos demográficos pertenecientes al recurso *person* (no sólo los parámetros introducidos por el usuario) de todos los candidatos, comparando así las diferentes combinaciones de parejas y estableciendo un nivel de similitud entre cada par de candidatos. La detección de registros coincidentes en una base de datos es algo muy trabajado en python, por lo que tras un rastreo de librerías que pudiesen ser de utilidad para facilitar este proceso, se ha encontrado la librería **Record Linkage Toolkit** (31). Esta librería no está disponible para C Sharp, por lo que se ha decidido incluir un script en python para esta parte del servicio, facilitando muchísimo el emparejamiento de candidatos mediante las funciones de indexación y los métodos para comparar y clasificar los registros.

Por último, en búsca de un sistema de trazado de código que abarcase más funcionalidades que el incluido por defecto en el propio entorno de desarrollo, como el envío de logs a ficheros, consolas, bases de datos o servidores en otras maquinas, se ha decidido utilizar la librería **NLog.Web.AspNetCore** (30).

4.7 Requisitos a desarrollar

En este apartado se definen los requisitos que tienen que cumplir los diferentes subprocesos para alcanzar los objetivos del proyecto. Es importante destacar que en este proyecto no se han implementado políticas de seguridad. Se recuerda que FHIR no es un protocolo de seguridad, ni define ninguna funcionalidad relacionada con ella. Sin embargo, define los protocolos de intercambio y los modelos de contenido que necesitan ser usados con varios protocolos de seguridad definidos. Pueden ser consultados en la especificación.

4.7.1 Lectura de los ficheros de configuración xml

El objetivo principal es la lectura y el procesamiento de los ficheros de configuración al inicio del servidor. Se encarga de almacenar en memoria la información relevante para que no se tengan que volver a leer por cada petición del usuario. Los requisitos que tendrá que cumplir son los siguientes:

- Realizar la lectura una única vez al iniciar el servidor.
- No iniciar el servidor si la lectura de algunos de los parámetros es incorrecta.
- Generar una constante tipo struct que contenga la información.

4.7.2 Procesado de los parámetros de la petición HTTP

El objetivo principal es analizar la peticion HTTP, validando que los parámetros sean correctos. Además, identifica el perfil genérico (ver apartado 2.3.3). según la configuración fijada. En *configuracion.xml* se fija *BusquedaExhaustiva=True/False*, para que se realice o no una búsqueda Exhaustiva. En caso de no estar activa, se selecciona de los parámetros introducidos por el usuario, únicamente el de mayor ponderación y se utiliza como perfil genérico. Por el contrario, si está activa, detecta de los parámetros de entrada los que se han fijado manualmente para dicha búsqueda (establecido en el fichero de configuración mediante el atributo busquedaExhaustiva=true), utilizándolos como perfil genérico.

En la siguiente figura se puede observar un ejemplo de configuración de las ponderaciones de los distintos parámetros de búsqueda para el recurso *person*.

```
<BusquedaExhaustiva> False </BusquedaExhaustiva>

<PonderacionParametrosPerson>
  <address>0,6</address>
  <address-city busquedaExhaustiva="true">0,6</address-city>
  <address-country>0,7</address-country>
  <address-postalcode>0,5</address-postalcode>
  <address-state>0,5</address-state>
  <address-use>0,6</address-use>
  <birthdate busquedaExhaustiva="true"> 0,9</birthdate>
  <email> 0,7</email>
  <gender> 0,5</gender>
  <identifier busquedaExhaustiva="true"> 0,9</identifier>
  <link> 0,7</link>
  <name> 0,8</name>
  <organization> 0,7</organization>
  <patient> 0,5</patient>
  <phone busquedaExhaustiva="true"> 0,7</phone>
  <phonetic> 0,6</phonetic>
  <practitioner> 0,5</practitioner>
  <relatedperson> 0,5</relatedperson>
  <telecom> 0,7</telecom>
</PonderacionParametrosPerson>
```

Figura 20. Definición de ponderaciones de parámetros del recurso Person en configuracion.xml

La opción BusquedaExhaustiva está desactivada, por lo que si el usuario introduce por ejemplo los parámetros *phone*, *email* e *identifier* al realizar la búsqueda, el servicio elegiría únicamente el parámetro *identifier* como perfil genérico para obtener la primera lista amplia de candidatos, por ser el de mayor ponderación (sólo se elige uno).

Por el contrario, si BusquedaExhaustiva estuviera desactivada, suponiendo que el usuario realizase la búsqueda con los mismos parámetros, el servicio elegiría los parámetros *identifier* y *phone* como perfil genérico, ya que tienen fijado el atributo busquedaExhaustiva=true.

Los requisitos que tendrá que cumplir dicho procesado son los siguientes:

- No se procesan peticiones que contengan algún parámetro no permitido
- No se procesan peticiones que no contengan parámetros.

- Generar una variable con la petición descompuesta, con el perfil genérico identificado.

4.7.3 Búsqueda inicial de candidatos a partir de perfil genérico

El objetivo principal es recoger un amplio listado de candidatos desde diferentes sistemas externos, a partir del perfil genérico (ver apartado 2.3.3). Este listado será procesado posteriormente en los subprocesos de validación. Los requisitos que tendrá que cumplir son los siguientes:

- Realizar una búsqueda independiente por cada parámetro perteneciente al perfil genérico en cada sistema externo, en caso de que esté formado por más de uno. Puede darse la posibilidad de que algunos candidatos cumplan con un parámetro del perfil genérico pero no con el resto, por lo que si se hace una búsqueda directamente del conjunto, todos aquellos que no cumplan con alguno de ellos quedarán descartados de la búsqueda.
- Permitir obtener un listado de únicamente las URLs de los diferentes candidatos, que se almacenarán en memoria.
- No almacenar URLs repetidas en memoria.
- Tener en cuenta posibles excepciones al realizar las peticiones

En la tabla Candidatos (ver apartado 5.2.3.1) se almacenará tanto la dirección del servidor al que pertenece como la URL completa para acceder al recurso *Person* correspondiente. A pesar de que la URL ya incluye la dirección, se almacena también de forma explícita para facilitar la navegación por los diferentes datos.

4.7.4 Validación determinista

El objetivo principal es realizar una petición HTTP a cada sistema por cada parámetro [identificador del candidato (unívoco en dicho sistema) + parámetro a validar] mediante una petición *search* de la librería HL7.Fhir.STU3 (29) (implementación del estándar FHIR para .net), realizada en el método *PeticionConParametro*. Según si hay o no respuesta, se rellena la tabla "Validaciones" (ver apartado 5.2.3.1) con los valores uno o cero respectivamente por cada parámetro. Todos aquellos valores distintos de cero, se multiplican por su ponderación correspondiente, fijada en el fichero de configuración (ver *PonderacionParametrosPerson* en apartado 5.2.4.1) y se suman, diviendo dicha suma por el total de parámetros introducidos por el usuario y obteniendo así un porcentaje de parámetros válidos para cada candidato. A partir de estos porcentajes, se calcula el nivel de confianza para cada candidato, ya que se han definido los intervalos para los distintos niveles de confianza en el fichero de configuración (ver *AlgoritmoConfianzaDeterminista* en apartado 5.2.4.1). Se descartan aquellos candidatos que no superan el nivel de confianza mínimo fijado en la configuración (*ConfianzaMinDeterminista* en apartado 5.2.4.1). Los requisitos que tendrá que cumplir son los siguientes:

- Optimizar la peticiones realizadas, pues sólo hay que comprobar que el contenido de la respuesta no sea nulo, no hay que recuperar ningún campo de información.
- Tener en cuenta posibles excepciones al realizar las peticiones
- Generar la tabla *Respuesta* con los respectivos niveles de confianza, únicamente si no se va a realizar la validación probabilista.

La validación determinista se ejecutará o no en función de si en *configuracion.xml* se fija *AnalisisDeterminista=True/False* respectivamente.

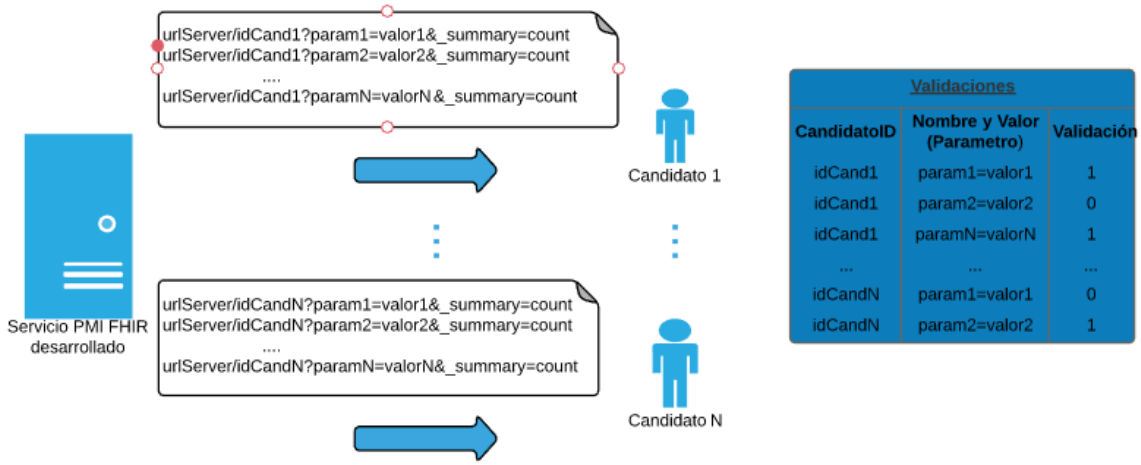


Figura 21. Esquema de validación determinista

4.7.5 Validación probabilista

El objetivo principal es realizar una petición HTTP a cada sistema por cada candidato mediante una petición *search* de la librería HL7.Fhir.STU3 (29) (implementación del estándar FHIR para .net), realizada en el método *PeticionIdPerson* al sistema FHIR correspondiente, obteniendo así el recurso *Person* completo de cada uno de ellos. Mediante el método *generarPonderación*, se pondera cada parámetro solicitado por el usuario, generando un decimal con valores comprendidos entre cero y uno, según el nivel de similitud y rellenando la tabla "Validaciones" (ver apartado 5.2.3.1) con dicho decimal. Todos aquellos valores distintos de cero, se multiplican por su ponderación correspondiente, fijada en el fichero de configuración (ver *PonderacionParametrosPerson* en apartado 5.2.4.1) y se suman, diviendo dicha suma por el total de parámetros introducidos por el usuario y obteniendo así un porcentaje de parámetros válidos para cada candidato. A partir de estos porcentajes, se calcula el nivel de confianza para cada candidato, ya que se han definido los intervalos para los distintos niveles de confianza en el fichero de configuración (ver *AlgoritmoConfianzaProbabilista* en apartado 5.2.4.1). Se descartan aquellos candidatos que no superan el nivel de confianza mínimo fijado en la configuración (*ConfianzaMinProbabilista* en apartado 5.2.4.1). Los requisitos que tendrá que cumplir son los siguientes:

- Tener en cuenta posibles excepciones al realizar las peticiones
- Generar la tabla *Respuesta* con los respectivos niveles de confianza.

La validación probabilista se ejecutará o no en función de si en *configuracion.xml* se fija *AnalisisProbabilista=True/False* respectivamente.

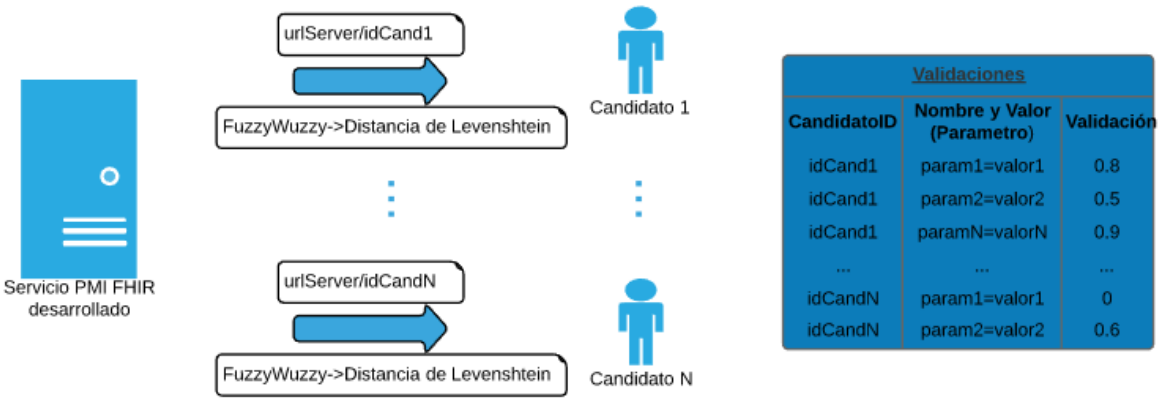


Figura 22. Esquema de validación probabilista

4.7.6 Identificación de coincidencias

El objetivo principal es detectar posibles coincidencias entre los candidatos finales obtenidos de los diferentes servidores, agrupando los registros coincidentes que se refieren a la misma persona en el mismo recurso Person. Se realizará una comparación entre los resultados finales de cada campo, no sólo los introducidos por parte de los usuarios, sino de todos los campos demográficos pertenecientes al recurso *person* (apartado 2.4.2.2) , generando una ponderación de silimitud por cada par de candidatos y marcando como coincidentes todos los que superen un umbral mínimo, que será fijado en el fichero de configuración (apartado 5.2.4.1) mediante el parámetro *ValorUmbralCoincidencia.* Los requisitos que tendrá que cumplir son los siguientes:

- Recibir un fichero csv generado con los candidatos y sus respectivos campos a comparar.
- Generar un fichero de salida con los pares coincidentes detectados.
- Recibir por parámetros la ruta del fichero generado y el umbral mínimo de silimitud para marcar los registros coincidentes.

La identificación de duplicados se ejecutará o no en función de si en *configuracion.xml* se fija *RealizarIdentCoincidencias=True/False* respectivamente.

5 DISEÑO

En este capítulo, se explicarán de forma estructurada los distintos subprocesos identificados en la sección anterior, con ayuda de algunos diagramas UML. Se empezará desde un nivel más general y se irá reduciendo el nivel de abstracción para detallar más detenidamente cada subproceso.

En primer lugar, se resumirá desde un punto de vista más general, la estructuración del código, la división por paquetes y el funcionamiento básico mediante un diagrama de secuencia general. A continuación, se detallarán las clases y métodos que intervienen en cada paquete, con ayuda de algunos diagramas UML.

Es importante resaltar que se han utilizado en el diseño los patrones Modelo Vista Controlador y Singleton, tal y como se muestra en los diagramas. La vista se ocupa de dar forma a las respuestas que son enviadas al usuario, para que cumplan así con el estándar FHIR. Los controladores son todas las clases que gestionan eventos e interacciones tanto con el usuario como con otras clases y sistemas. Por último, el modelo está compuesto por las clases que definen los datos y que trabajan directamente con los archivos de configuración (lectura de ficheros xml).

La clase encargada de leer los ficheros de configuración, implementa el patrón Singleton, para asegurar que sólo exista una única instancia y que sólo se realice la lectura al iniciar el servidor.

5.1 Diseño global

5.1.1 Diagrama de paquetes

En este diagrama se observa cómo se ha divido el sistema mediante la división por diferentes paquetes.

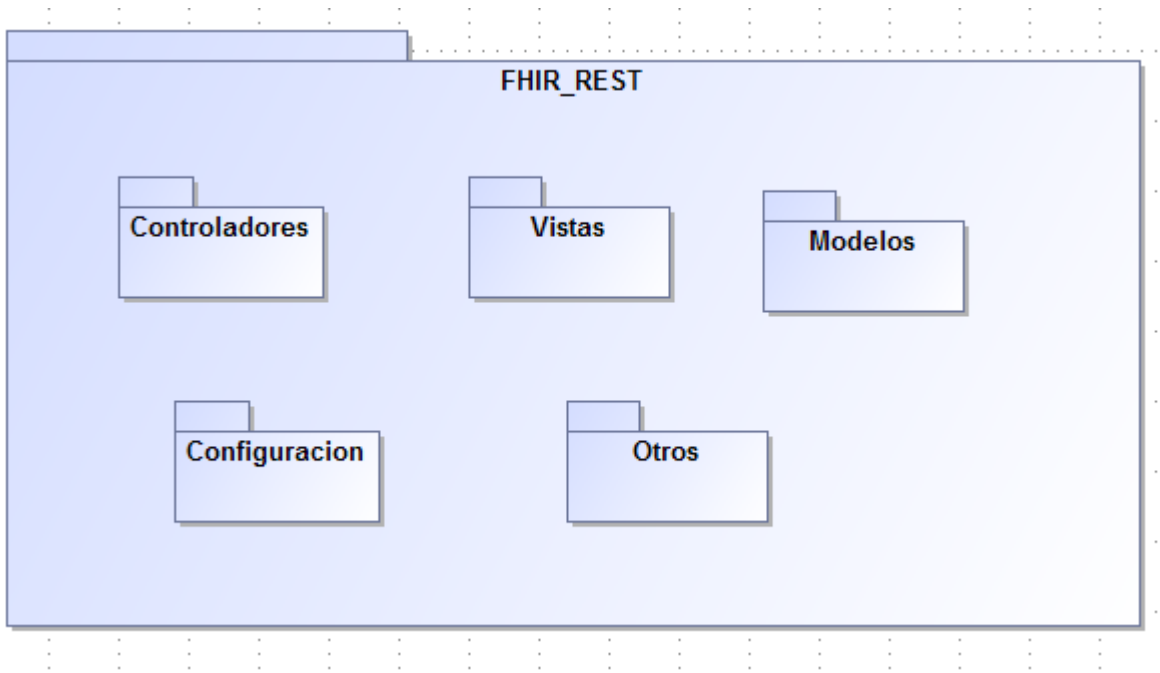


Figura 23. Diagrama de paquetes genérico

5.1.2 Diagrama de clases

En este diagrama se representa el modelo de la aplicación desarrollada: métodos, atributos y relaciones entre clases.

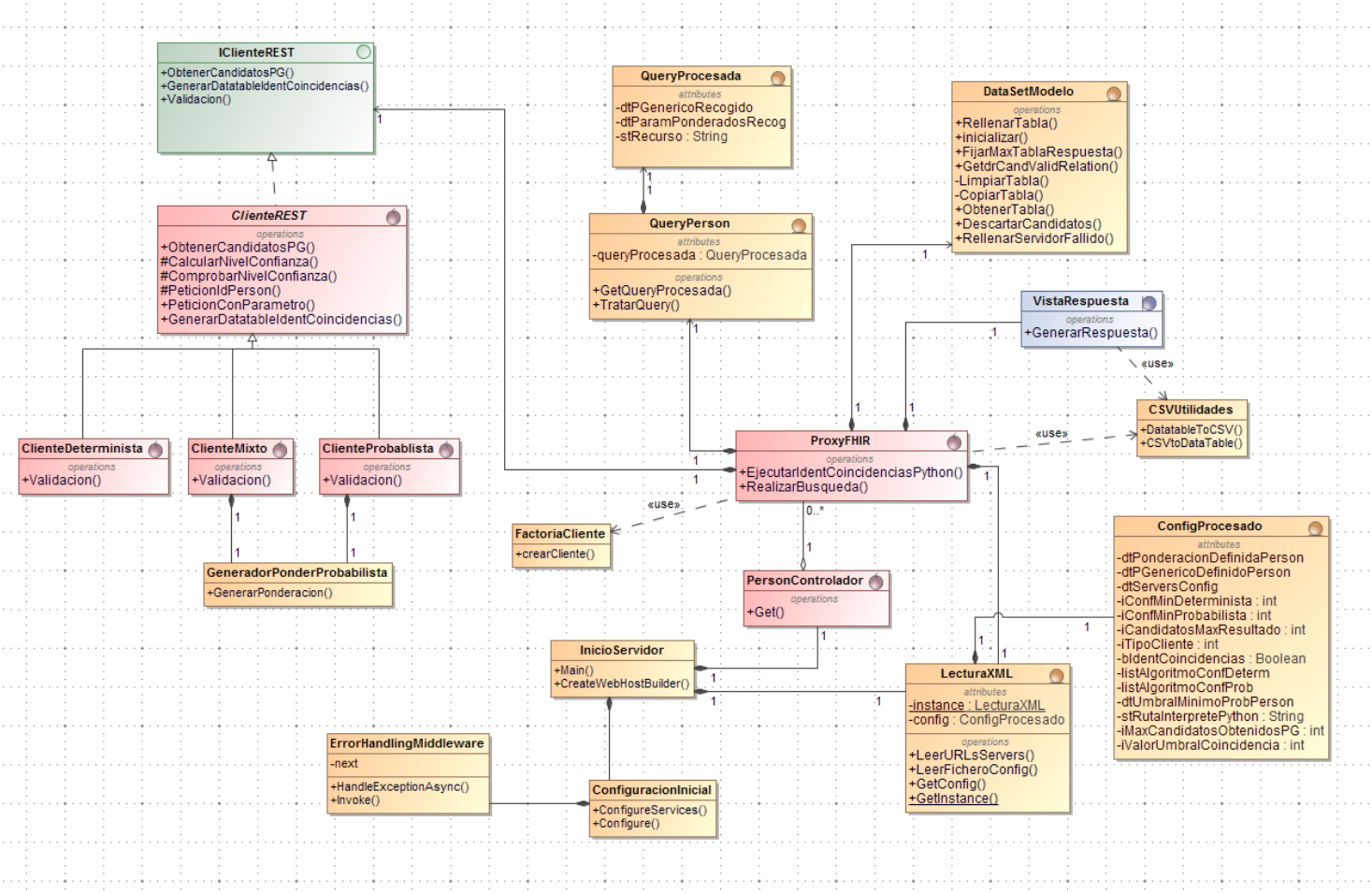


Figura 24. Diagrama de clases genérico

5.1.3 Diagrama de secuencia

En este diagrama se muestra la secuencia de intercambio de mensajes cuando un usuario realiza una determinada búsqueda a partir de una petición HTTP enviada al servicio

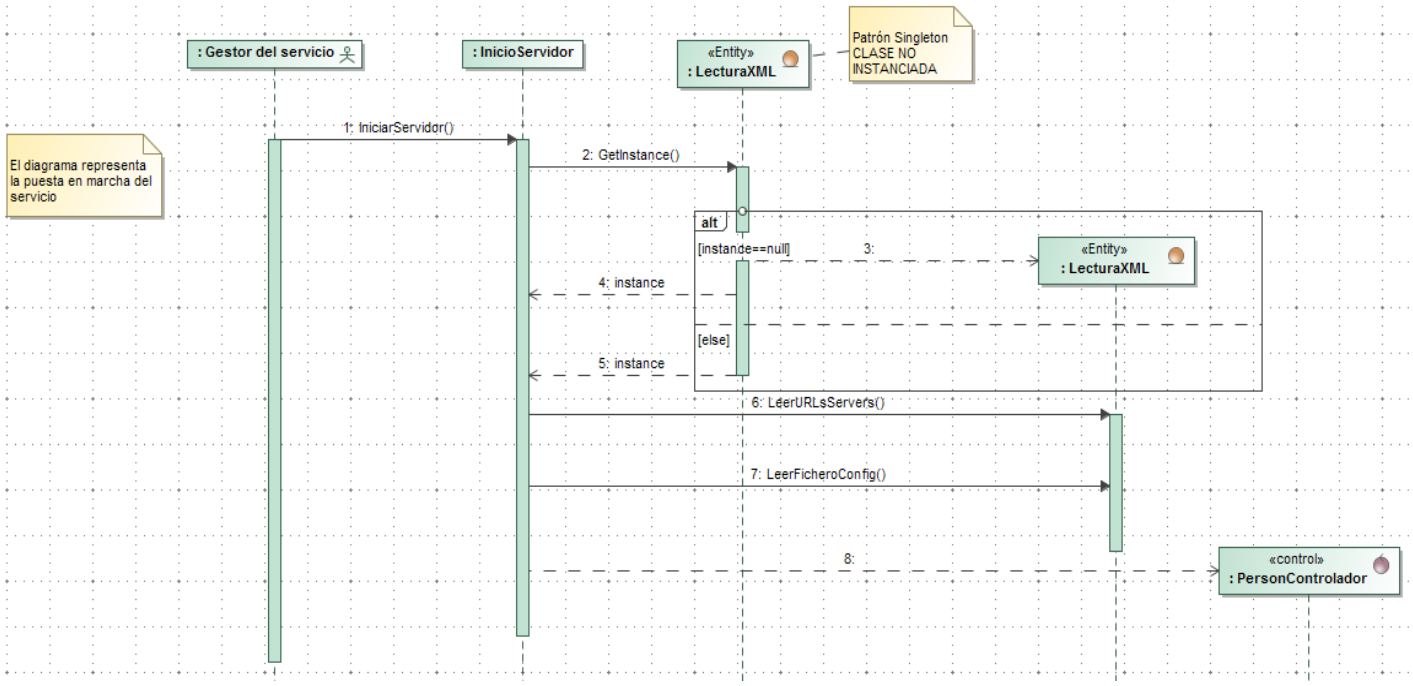


Figura 25. Diagrama de secuencia del inicio del servicio

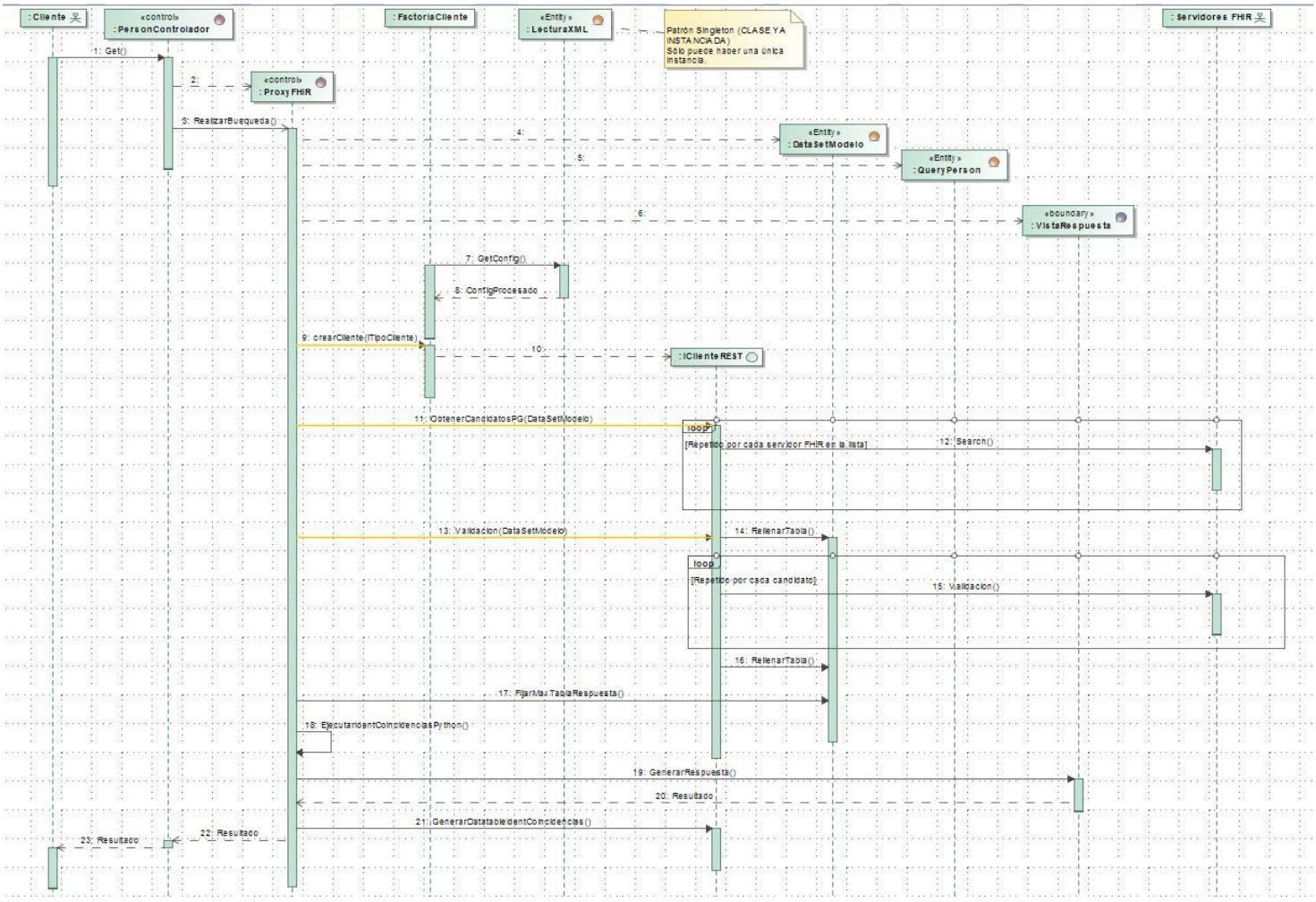


Figura 26. Diagrama de secuencia de la búsqueda de candidatos

5.2 Diseño específico

El proyecto está estructurado en paquetes que contienen diferentes archivos con una determinada relación. Cada archivo, contiene una clase definida con el mismo nombre.

5.2.1 Paquete *Controladores*

Este paquete contiene los archivos encargados de gestionar los eventos e interacciones tanto con el usuario como con otras clases y sistemas.

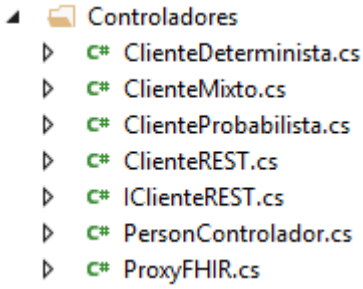


Figura 27. Contenido del paquete Controladores

5.2.1.1 ClienteDeterminista

Hereda de la clase ClienteREST. Se encarga de realizar la validación determinista y añade únicamente el método Validación.

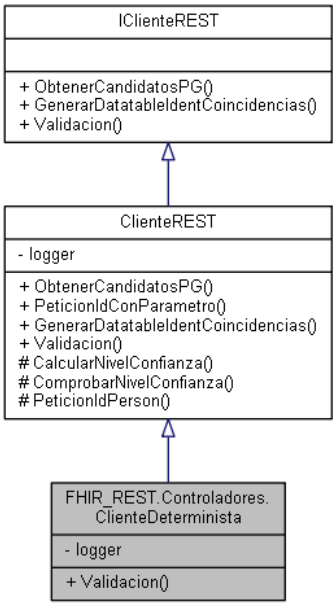


Figura 28. Diagrama de colaboración de ClienteDeterminista

Método	Validacion
Descripción	Valida los parámetros de forma determinista (ver apartado 4.7.4) en todos los

	candidatos y calcula el nivel de confianza, descartando todos aquellos que no superen el umbral mínimo configurado.
Parámetros de entrada	• DatasetModelo (por referencia): Contiene el DataSet que se va a actualizar, rellenando las tablas "Validaciones" y "Respuesta". • QueryPerson: Contiene la query realizada por el usuario descompuesta y validada.
Respuesta	Nada

Tabla 3. Descripción método ClienteDeterminista.Validacion

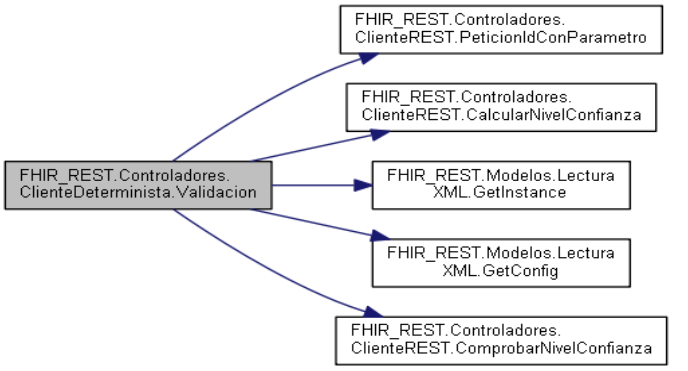


Figura 29. Gráfico de llamadas ClienteDeterminista.Validacion

5.2.1.2 ClienteProbabilista

Hereda de la clase ClienteREST. Se encarga de realizar la validación probabilista y añade únicamente el método Validación.

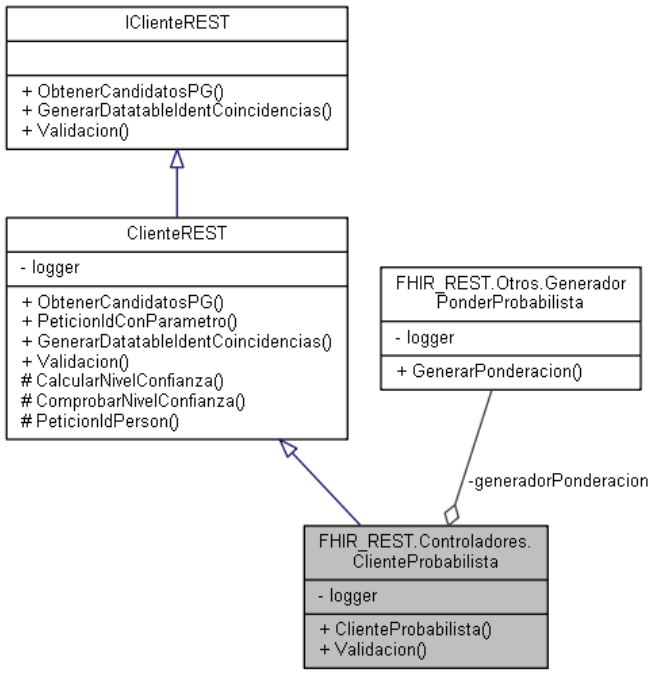


Figura 30. Diagrama de colaboración de ClienteProbabilista

Método	Validacion
Descripción	Valida los parámetros de forma probabilista (ver apartado 4.7.5) en todos los candidatos y calcula el nivel de confianza, descartando todos aquellos que no superen el umbral mínimo configurado.
Parámetros de entrada	• DatasetModelo (por referencia): Contiene el DataSet que se va a actualizar, rellenando las tablas "Validaciones" y "Respuesta". • QueryPerson: Contiene la query realizada por el usuario descompuesta y validada.
Respuesta	Nada

Tabla 4. Descripción método ClenteProbabilista.Validacion

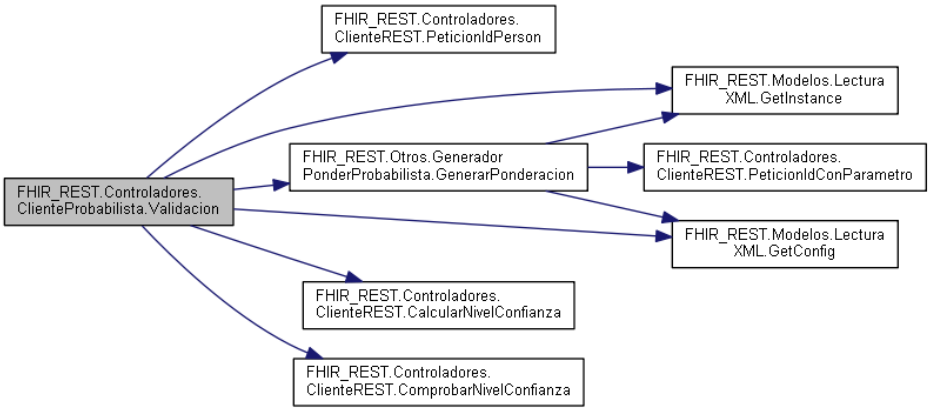


Figura 31. Gráfico de llamadas ClienteProbabilista.Validacion

5.2.1.3 ClienteMixto

Hereda de la clase ClienteREST. Se encarga de realizar una validación mixta (determinista y probabilista) y añade únicamente el método Validación.

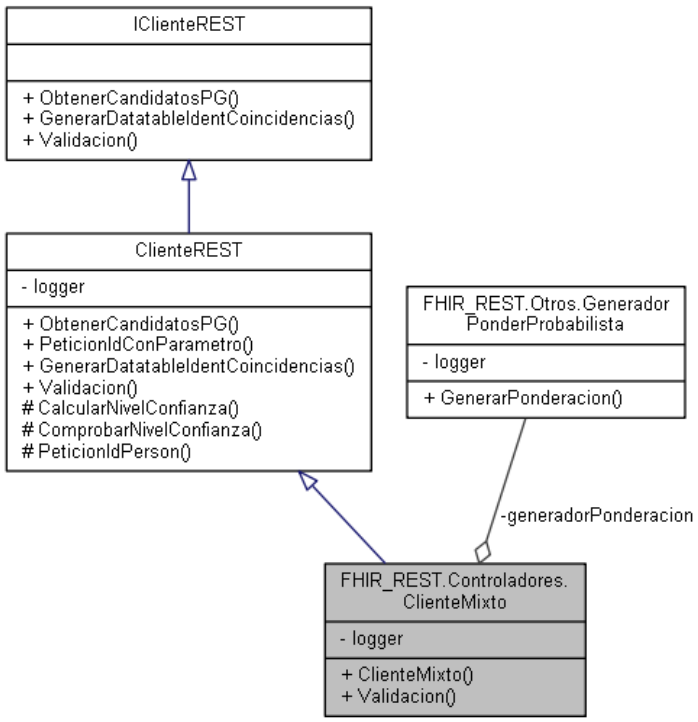


Figura 32. Diagrama de colaboración de ClienteMixto

Método	Validacion
Descripción	Realiza primero una validación determinista (ver apartado 4.7.4), descartando los candidatos que no superan el nivel de confianza mínimo fijado (criba primaria). A continuación, realiza a los candidatos no descartados una validacion probabilista (ver apartado 4.7.5), validando únicamente los parámetros marcados anteriormente con un 0. Por último, vuelve a realizar los descartes necesarios (criba secundaria).
Parámetros de entrada	• DatasetModelo (por referencia): Contiene el DataSet que se va a actualizar, rellenando las tablas "Validaciones" y "Respuesta". • QueryPerson: Contiene la query realizada por el usuario descompuesta y validada.
Respuesta	Nada

Tabla 5. Descripción método ClienteMixto.Validacion

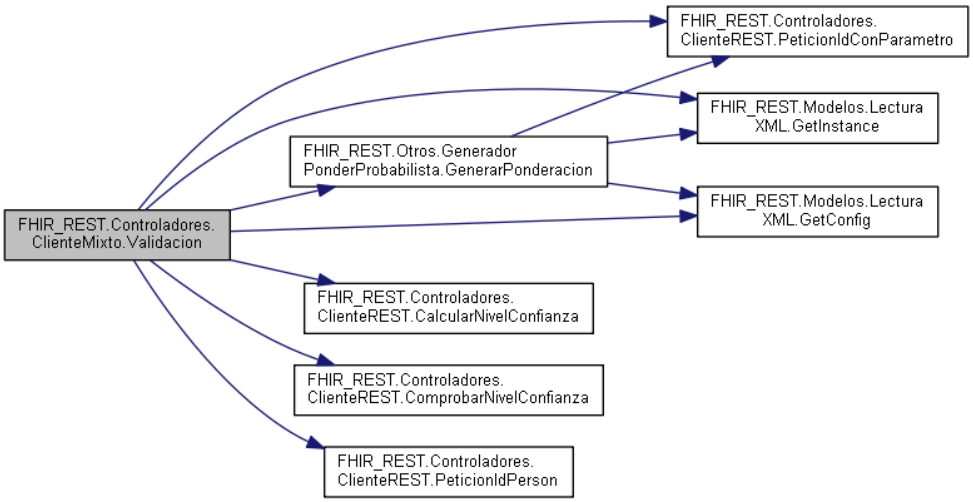


Figura 33. Gráfico de llamadas ClienteMixto.Validacion

5.2.1.4 ClienteREST

Clase padre que define los métodos necesarios para actuar como un cliente REST que busca candidatos (envía peticiones HTTP) en múltiples servidores REST y calcula el nivel de confianza.

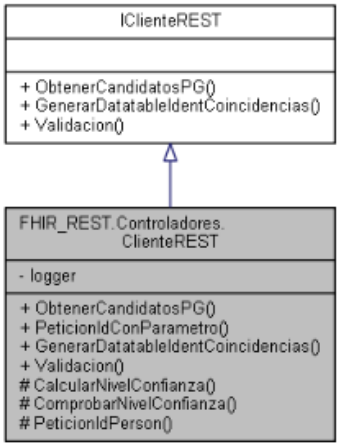


Figura 34. Diagrama de colaboración de ClienteREST

Método	CalcularNivelConfianza
Descripción	Calcula el nivel de confianza a partir de un decimal entre cero y uno y las fronteras (decimales consecutivos entre 0 y 1) establecidas en el fichero *configuracion.xml* (apartado 5.2.4.1) mediante los parámetros *AlgoritmoConfianzaDeterminista* y *AlgoritmoConfianzaProbabilista.*
Parámetros de entrada	• decConfianza: Decimal entre cero y uno que contiene una ponderación del candidato. • listAlgoritmoConf: Lista que contiene las fronteras configuradas para el calculo del nivel de confianza.

Respuesta	Un entero entre uno y cuatro que indica el nivel de confianza.

Tabla 6. Descripción método ClienteREST.CalcularNivelConfianza

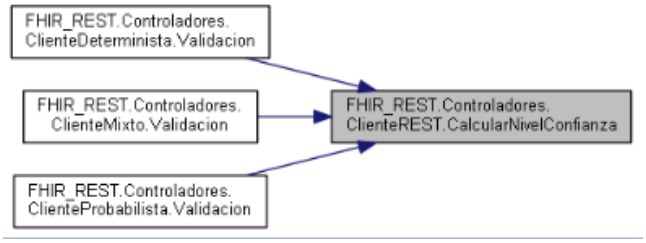


Figura 35. Gráfico de llamadas ClienteREST.CalcularNivelConfianza

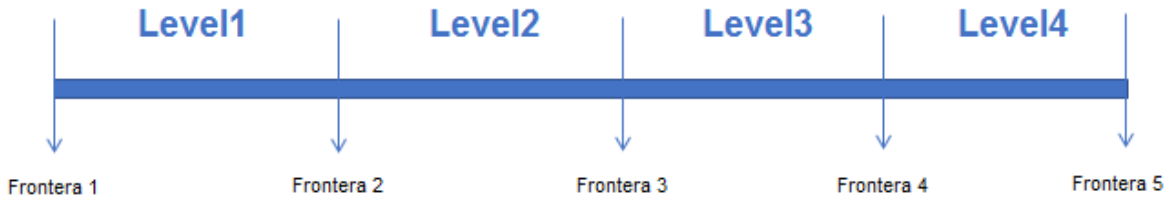


Figura 36. Fronteras para el cálculo del nivel de confianza

Método	ComprobarNivelConfianza
Descripción	Se comprueba que el nivel de confianza del candidato no es inferior al umbral mínimo pasado como parámetro. Los niveles mínimos son fijados en *configuracion.xml* mediante los parámetros *ConfianzaMinDeterminista* y *ConfianzaMinProbabilista*. Se almacena el candidato en la tabla "Respuesta" si bAlmacenarNivel es true.
Parámetros de entrada	• inivelConfianza: Nivel de confianza que se va a comprobar. • bAlmacenarNivel: Indica si se almacena o no en la tabla "Respuesta" en caso de comprobación válida. • datasetModelo (por referencia): Necesario para actualizar tabla "Respuesta" • candRow: Fila de la tabla "Candidatos" que contiene el candidato que se está analizando. • iminConfianza: Umbral mínimo de confianza.
Respuesta	Booleano que indica si el nivel de confianza supera o no el mínimo fijado.

Tabla 7. Descripción método ClienteREST.ComprobarNivelConfianza

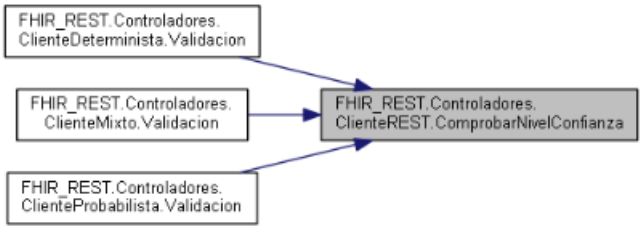


Figura 37. Gráfico de llamadas ClienteREST.ComprobarNivelConfianza

Método	GenerarDatatableIdentCoincidencias
Descripción	Genera una tabla (datatable) donde se almacenan todos los parámetros que se van a comparar de los candidatos pertenecientes a la tabla "Respuesta" (candidatos finales) durante el proceso de identificación de duplicados. Esta tabla será escrita en un fichero .csv para que el proceso pueda analizarlo.
Parámetros de entrada	• datasetModelo: Necesario para acceder a la tabla "Respuesta"
Respuesta	Tabla (datatable) generada con tantas columnas como parámetros se van a comparar y tantas filas como candidatos haya en la tabla "Respuesta".

Tabla 8. Descripción método ClienteREST.GenerarDatatableIdentCoincidencias

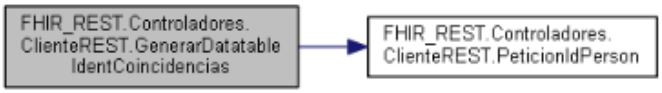


Figura 38. Gráfico de llamadas ClienteREST.GenerarDatatableIdentCoincidencias

Método	ObtenerCandidatosPG
Descripción	Obtiene una lista de candidatos a partir del perfil genérico identificado en la petición. . Las peticiones se realizan con el parámetro _summary=true para reducir datos innecesarios (sólo se requiere el campo fullURL de cada candidato) Se va a rellenar la tabla "Candidatos" con los resultados.
Parámetros de entrada	• datasetModelo (por referencia): Necesario para actualizar tabla "Candidatos". • queryProcesada: Variable (struct) que contiene el perfil genérico identificado en la petición.
Respuesta	Nada

Tabla 9. Descripción método ClienteREST.ObtenerCandidatosPG

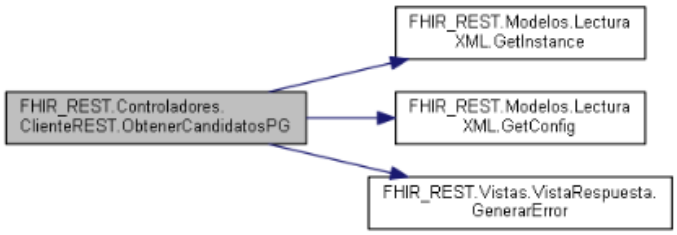


Figura 39. Gráfico de llamadas ClienteREST.ObtenerCandidatosPG

Método	PeticionIdConParametro
Descripción	Valida un parametro de un candidato a partir de una petición HTTP [urlServer + recurso + urlCandidato (identificador único en el sistema) + parametro]. Establece _sumary=count para obtener únicamente el número de resultados, pues sólo es necesario comprobar si devuelve uno o cero coincidencias, evitando obtener información innecesaria.
Parámetros de entrada	• urlServer: URL del servidor al que se realiza la peticion. • recurso: Recurso al que se va a acceder (estándar FHIR). • urlCandidato: Identificador único del candidato en el sistema al que se realiza la petición. • parámetro: Parámetro (nombre=valor) que se quiere validar.
Respuesta	True/False si se obtiene o no respuesta, es decir, si el candidato tiene o no el parámetro con el valor indicado.

Tabla 10. Descripción método ClienteREST.PeticionIdConParametro

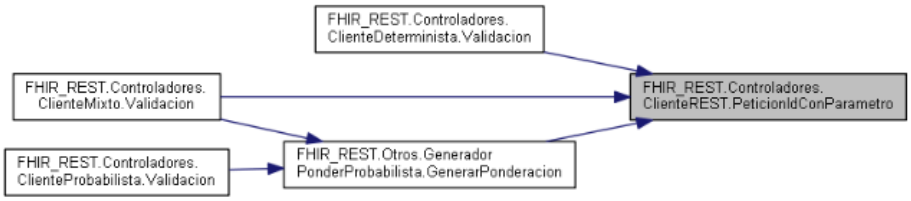


Figura 40. Gráfico de llamadas ClienteREST.PeticionIdConParametro

Método	PeticionIdPerson
Descripción	Realiza una peticion HTTP solicitando un recurso *Person* (apartado 2.4.2.2) a través del identificador único del candidato.
Parámetros de entrada	• urlServer: URL del servidor al que se realiza la petición. • urlCandidato: Identificador único del candidato en el sistema al que se realiza la petición.

Respuesta	Recurso *Person* obtenido.

Tabla 11. Descripción método ClienteREST.PeticionIdPerson

Figura 41. Gráfico de llamadas ClienteREST.PeticionIdPerson

Método	Validacion (abstracto)
Descripción	Función implementada en las clases hijas (CienteDeterminista, ClienteProbabilista y ClienteMixto). Valida los candidatos pertenecientes a la tabla "Candidatos".
Parámetros de entrada	• DatasetModelo (por referencia): Contiene el DataSet que se va a actualizar, rellenando las tablas “Validaciones” y “Respuesta”. • QueryPerson: Contiene la query realizada por el usuario descompuesta y validada.
Respuesta	Nada

Tabla 12. Descripción método ClienteREST.Validacion (abstracto)

5.2.1.5 IClienteREST.cs

Interfaz implementada por la clase ClienteREST. Define las firmas de los siguientes métodos:

- GenerarDatatableIdentCoincidencias
- ObtenerCandidatosPG
- Validacion

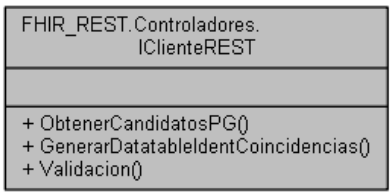


Figura 42. Diagrama de colaboración de IClienteREST

5.2.1.6 PersonControlador.cs

Define los métodos que atienden las posibles peticiones HTTP de los usuarios. En este proyecto sólo se atienden peticiones GET.

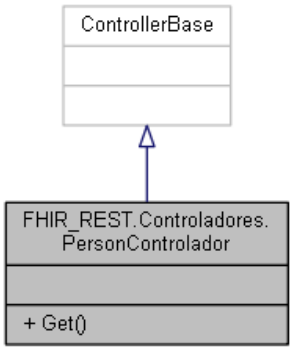


Figura 43. Diagrama de colaboración de PersonControlador

Método	Get
Descripción	Atiende la petición GET para el recurso *Person* (apartado 2.4.2.2) con los parámetros que el usuario considere oportunos, realizando la búsqueda de candidatos.
Parámetros de entrada	Nada.
Respuesta	Json serializado (cadena de texto) que contiene las respuestas obtenidas. Cumple estándar FHIR.

Tabla 13. Descripción método PersonControlador.Get

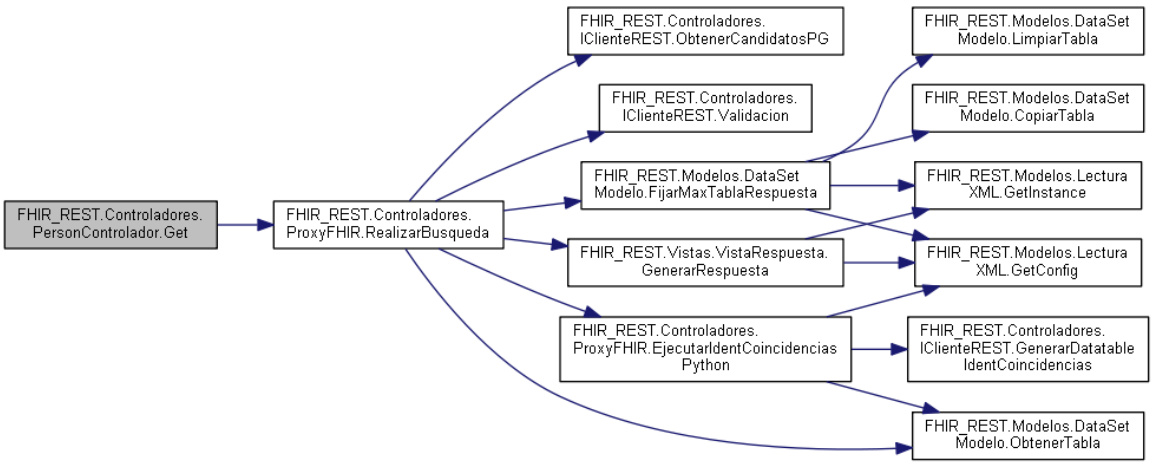


Figura 44. Gráfico de llamadas de PersonControlador.Get

5.2.1.7 ProxyFHIR.cs

Crea un objeto de un tipo de cliente determinado mediante FactoriaCliente, según la funcionalidad definida en el fichero de configuración, que conoce cuando consulta el objeto configProcesado de la clase LecturaXML (patrón Singleton). Inicia el proceso de búsqueda mediante el método RealizarBusqueda.

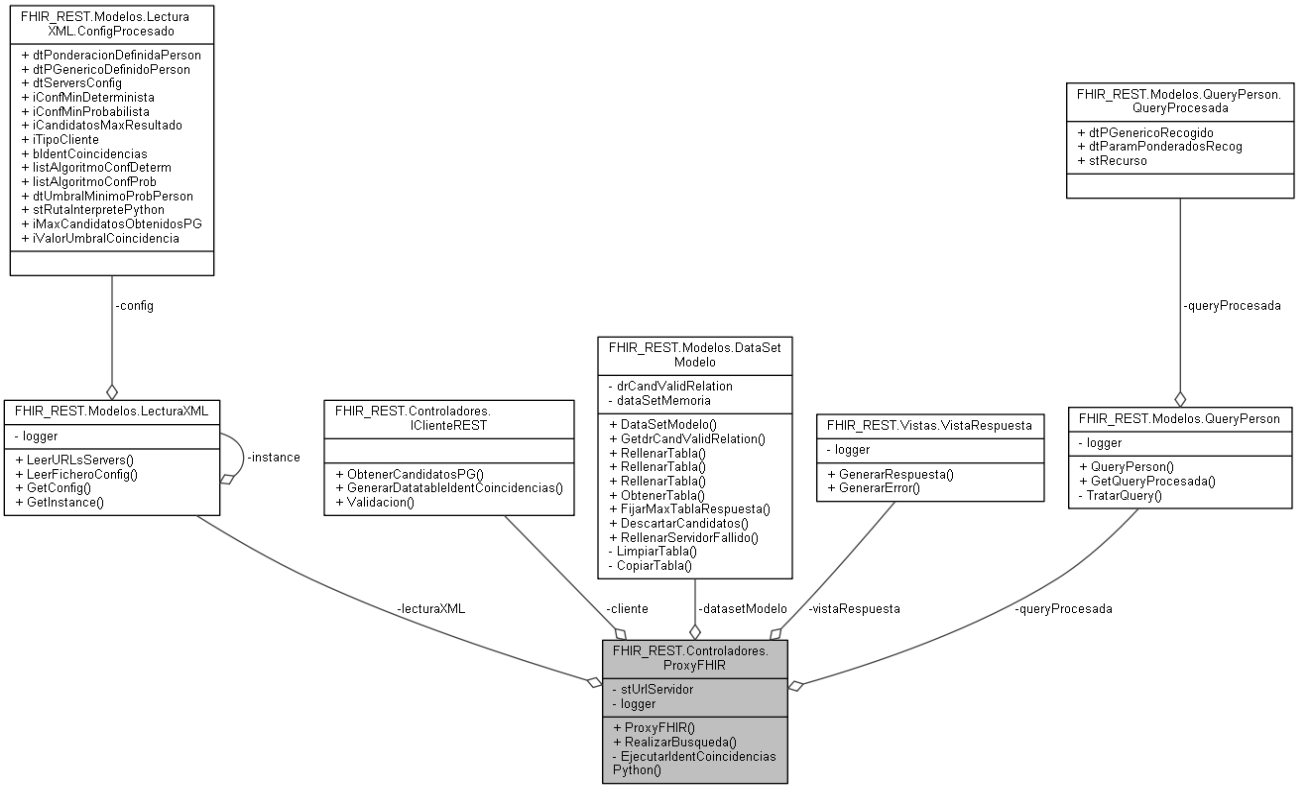


Figura 45. Diagrama de colaboración de ProxyFHIR

Método	RealizarBusqueda
Descripción	Primero, obtiene a partir del perfil genérico (ver apartado 2.3.3) detectado una amplia lista de posibles candidatos. A continuación, realiza las validaciones configuradas (calcula los niveles de confianza) de tipo determinista y/o probabilista, según el tipo de cliente creado mediante FactoriaCliente. Después, ejecuta el proceso de identificación de coincidencias (si así se ha establecido). Por último, genera la respuesta que se va a enviar al usuario para que cumpla con el estándar FHIR.
Parámetros de entrada	Nada.
Respuesta	Json serializado (cadena de texto) que contiene las respuestas obtenidas. Cumple estándar FHIR.

Tabla 14. Descripción método ProxyFHIR.RealizarBusqueda

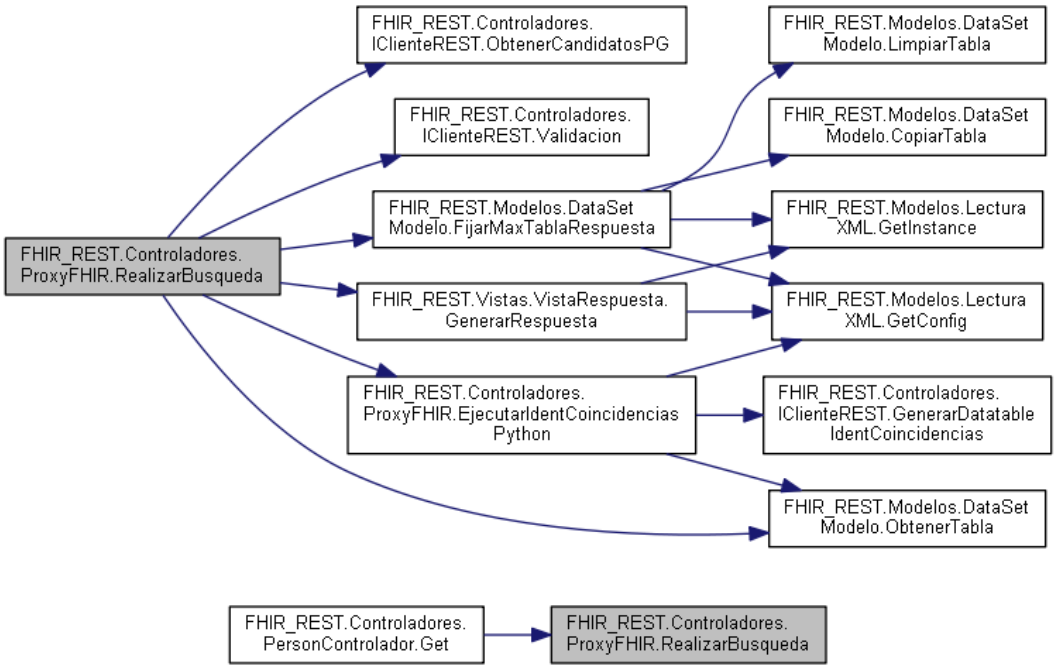


Figura 46. Gráfico de llamadas de ProxyFHIR.RealizarBusqueda

Método	EjecutarIdentCoincidenciasPython
Descripción	Ejecuta el proceso de identificación de coincidencias (script en python) mediante la inicialización de un nuevo proceso. Se encarga de inicializar dicho proceso indicando la ruta del fichero .csv a procesar, loggear la salida y sus posibles errores y de indicar si la ejecución ha sido exitosa o no.
Parámetros de entrada	• stRutaCSV: Ruta del fichero .csv que contiene los candidatos finales con los valores de los parámetros que se van a comparar.
Respuesta	Un booleano que indica si la ejecución se ha realizado o no de forma exitosa.

Tabla 15. Descripción método ProxyFHIR.EjecutarIdentCoincidenciasPython

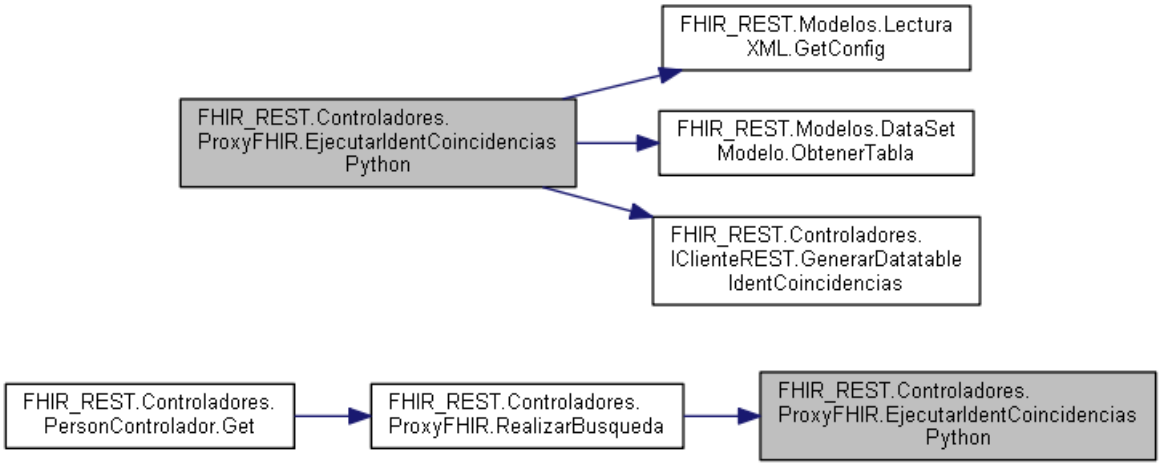
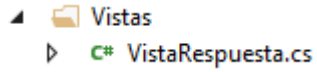


Figura 47. Gráfico de llamadas de ProxyFHIR.EjecutarIdentCoincidenciasPython

5.2.2 Paquete *Vistas*

Este paquete contiene el archivo encargado de dar forma a las respuestas que son enviadas al usuario, para que cumplan así con el estándar FHIR.



5.2.2.1 VistaRespuesta.cs

Se encarga de generar la respuesta que se va a enviar al usuario para que cumpla con el estándar FHIR.

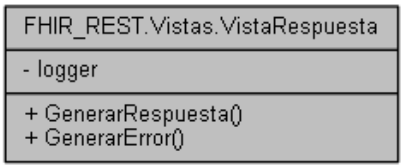


Figura 48. Diagrama de colaboración de VistaRespuesta

Método	GenerarRespuesta
Descripción	Crea un recurso tipo *Bundle* (apartado 2.4.2.1) y se rellena con los valores obtenidos como resultados, almacenados en la tabla "Respuesta". En caso de que se hayan detectado candidatos duplicados, se devuelven en el mismo recurso mediante el campo *Link*. Para serializar el objeto *Bundle* en un string, se utiliza la libreria Hl7.Fhir.Serialization.
Parámetros de entrada	• stRutaCSV: Indica la ruta donde se ecuentra el fichero .csv generado por el proceso de identificación de coincidencias. • datasetModelo: Necesario para acceder a la tabla "Respuesta". • stURLServidor: Petición URL completa realizada por el usuario (contiene la dirección del propio servicio). • bIdentCoincidenciasOK: Indica si la identificación de coincidencias ha tenido algun error (false) y debe ser indicado mediante el recurso *OperationOutcome* (apartado 2.4.2.3).
Respuesta	Json serializado (cadena de texto) que contiene las respuestas obtenidas. Cumple estándar FHIR.

Tabla 16. Descripción método VistaRespuesta.GenerarRespuesta

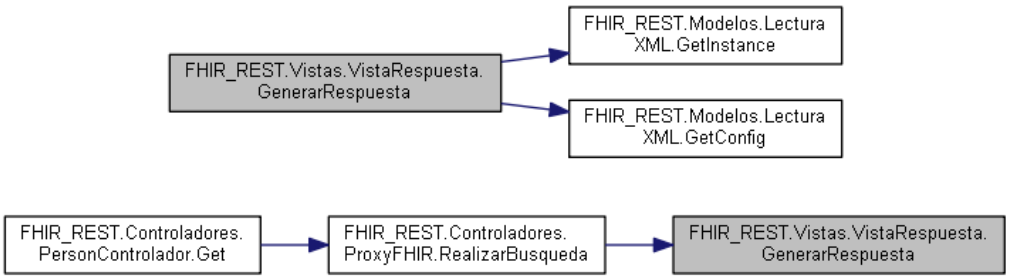


Figura 49. Gráfico de llamadas de VistaRespuesta.GenerarRespuesta

Método	GenerarError
Descripción	Genera la respuesta de error que se envía al usuario para que cumpla con el estándar FHIR. Crea el recurso de tipo *OperationOutcome* (apartado 2.4.2.3) y se rellena con los valores correspondientes. Para serializar el recurso *OperationOutcome* en un string, se utiliza la libreria Hl7.Fhir.Serialization.
Parámetros de entrada	• diagnostic: Valor que se fijará en el campo *Diagnostic* del recurso *OperationOutcome.* • issueType: Valor que se fijará en el campo *issue.Code* del objeto *OperationOutcome.*
Respuesta	Json serializado (cadena de texto) que contiene el error. Cumple estándar FHIR.

Tabla 17. Descripción método VistaRespuesta.GenerarError

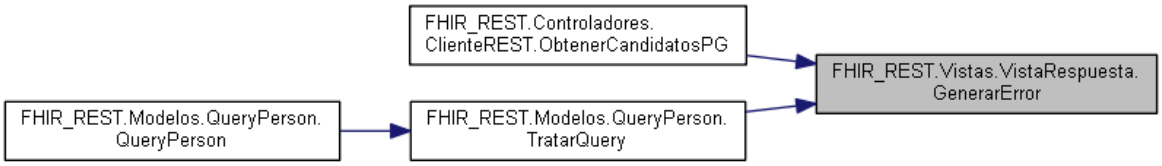


Figura 50. Gráfico de llamadas de VistaRespuesta.GenerarError

5.2.3 Paquete *Modelos*

Este paquete contiene los archivos que definen los modelos de datos y que trabajan directamente con los ficheros xml de configuración.

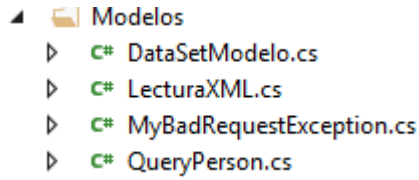


Figura 51. Contenido paquete Modelos

5.2.3.1 DataSetModelo

Contiene un DataSet y algunos métodos para facilitar la actualización y acceso a los datos almacenados en las diferentes tablas definidas.

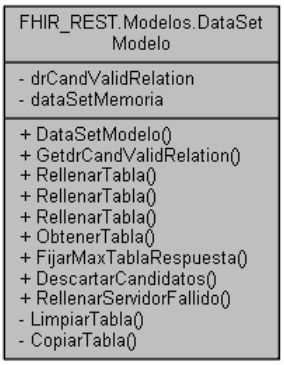


Figura 52. Diagrama de colaboración de DataSetModelo

Un DataSet (33) representa una caché de datos en memoria y está compuesta por una colección de tablas que pueden relacionarse entre sí mediante objetos DataRelation (34). Se crea una instancia de esta clase para cada usuario que accede a este servicio. En cada instancia, se almacenan los datos relativos a la búsqueda (candidatos, validaciones, respuesta, etc). Una vez devuelta la respuesta al usuario, se elimina la caché en memoria.

Mediante la siguiente figura se representa un esquema de las tablas que componen el DataSet y de las relaciones existentes entre ellas.

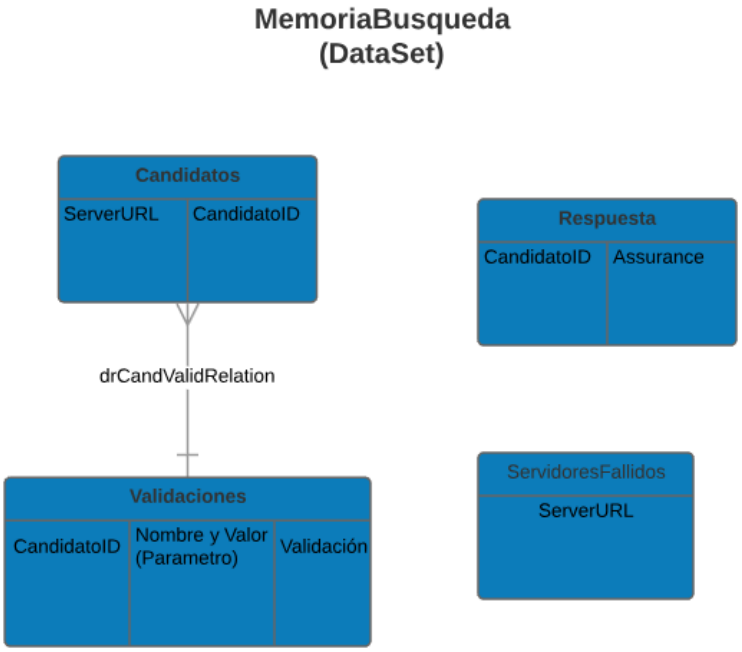


Figura 53. Esquema relacional DataSet MemoriaBusqueda

Método	CopiarTabla
Descripción	Copia la tabla pasada como parámetro en una tabla perteneciente al DataSet.

Parámetros de entrada	• nomTablaDestino: Nombre de la tabla perteneciente al DataSet. • tablaOrigen: Tabla (datatable) que se va a copiar.
Respuesta	Nada

Tabla 18. Descripción método DataSetModelo.CopiarTabla

Figura 54. Gráfico de llamadas DataSetModelo.CopiarTabla

Método	DescartarCandidatos
Descripción	Elimina de la tabla "Candidatos" todas las filas pertenecientes a la tabla (datatable) pasada como parámetro.
Parámetros de entrada	• dtCandidatosDescartados: Tabla que contiene los candidatos para descartar.
Respuesta	Nada.

Tabla 19. Descripción método DataSetModelo.DescartarCandidatos

Método	FijarMaxTablaRespuesta
Descripción	Ordena la tabla "Respuesta" por nivel de confianza descendente, eliminado las últimas filas (menor nivel de confianza) en caso de que el número de candidatos sea mayor que el máximo fijado en el fichero de configuración (*CandidatosMaxDevueltos)*
Parámetros de entrada	Nada.
Respuesta	Nada.

Tabla 20. Descripción método DataSetModelo.FijarMaxTablaRespuesta

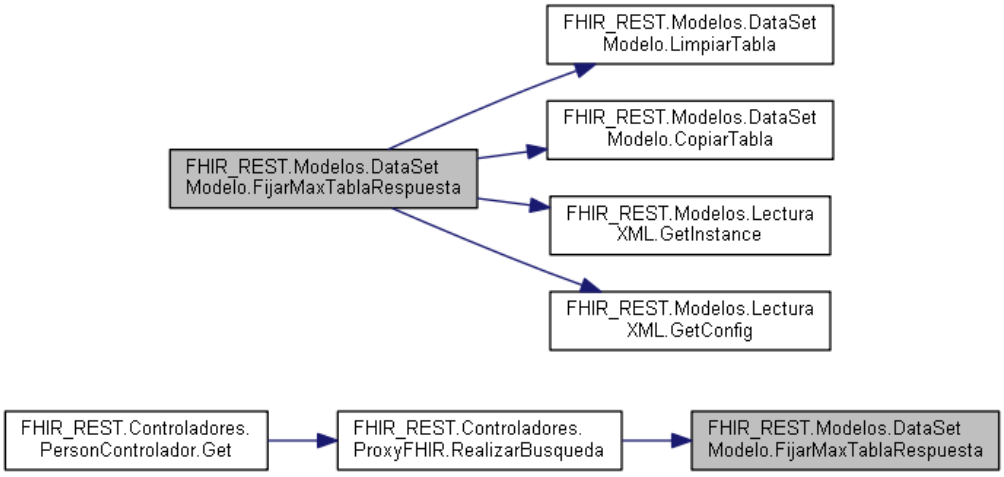


Figura 55. Gráfico de llamadas DataSetModelo.FijarMaxTablaRespuesta

Método	GetdrCandValidRelation
Descripción	Método Get que retorna la relación entre "Candidatos y "Validaciones".
Parámetros de entrada	Nada
Respuesta	Variable de tipo DataRelation.

Tabla 21. Descripción método DataSetModelo.GetdrCandValidRelation

Método	LimpiarTabla
Descripción	Limpia (vacía) la tabla indicada perteneciente al DataSet
Parámetros de entrada	• nomTabla: Tabla perteneciente al DataSet que se va a vaciar.
Respuesta	Nada

Tabla 22. Descripción método DataSetModelo.LimpiarTabla

Figura 56. Gráfico de llamadas DataSetModelo.LimpiarTabla

Método	ObtenerTabla
Descripción	Retorna la tabla indicada como parámetro.
Parámetros de entrada	• nomTabla: Nombre de la tabla que se va a devolver.

Respuesta	Variable tipo datatable.

Tabla 23. Descripción método DataSetModelo.ObtenerTabla

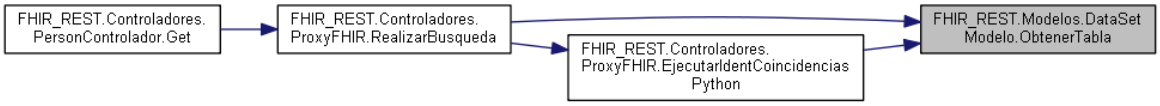


Figura 57. Gráfico de llamadas DataSetModelo.ObtenerTabla

Método	RellenarServidorFallido
Descripción	Añade a la tabla "ServidoresFallidos" la ULR pasada como parámetro, siempre y cuando no exista ya en la tabla. Estos servidores son aquellos que han dado error durante alguna de las peticiones realizadas. Al devolver la respuesta al usuario, se indica si se ha producido algun error durante las diferentes consultas a los servidores externos.
Parámetros de entrada	• URLServerFallido: URL que se agrega a la tabla “ServidoresFallidos”.
Respuesta	Nada.

Tabla 24. Descripción método DataSetModelo.RellenarServidorFallido

Método	RellenarTabla
Descripción	Método que utiliza la sobrecarga y que rellena la tabla indicada con los valores pasados como parámetros.
Parámetros de entrada	• nomTabla: Nombre de la tabla que se va a rellenar. • ValorX: Valores que se van a introducir en la tabla, generando una nueva fila.
Respuesta	Nada.

Tabla 25. Descripción método DataSetModelo.RellenarTabla

5.2.3.2 MyBadRequestException

Gestiona las excepciones HTTP de tipo MyBadRequestException, que se enviarán al usuario cuando el error esté provocado por una incorrecta petición.

El atributo *text* contiene el json serializado (cadena de texto) que representa el recurso *OperationOutcome,* devuelto por el método Vistas.GenerarError, mientras que el atributo *log*, contiene únicamente el mensaje producido en el error, que se imprimirá en el trazado de código.

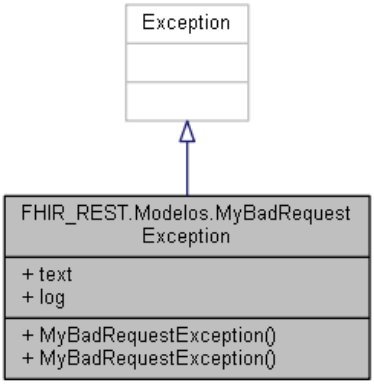


Figura 58. Diagrama de colaboración de MyBadRequestException

5.2.3.3 LecturaXML

Realiza la lectura de los diferentes ficheros de configuración, generando una variable (struct) con toda la información obtenida.

Implementa el patrón Singleton, que garantiza que esta clase sólo tenga una instancia, proporcionando un punto de acceso global a ella. La lectura se realiza por lo tanto una única vez al iniciar el servidor.

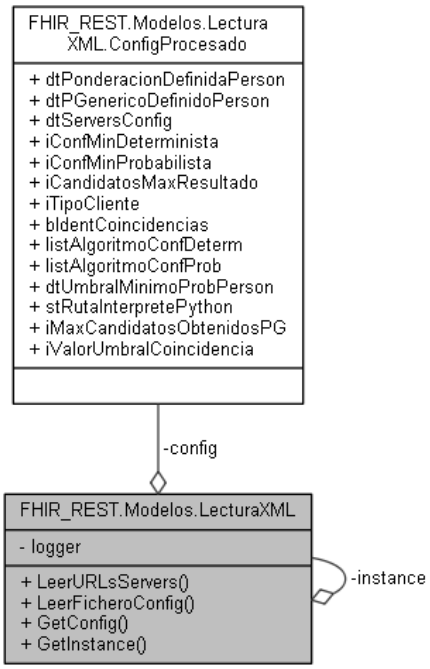


Figura 59. Diagrama de colaboración de LecturaXML

Método	GetConfig
Descripción	Método Get que retorna la variable tipo ConfigProcesado (struct), donde se almacena la información obtenida de los ficheros de configuración
Parámetros de entrada	Nada

Respuesta	Variable tipo ConfigProcesado (struct)

Tabla 26. Descripción método LecturaXML.GetConfig

Método	GetInstance
Descripción	Se crea una instancia de la clase si no existía, retornando dicho valor. Sólo es posible una instancia (Patrón Singleton). En el caso de que ya existiera, se devuelve un puntero al objeto.
Parámetros de entrada	Nada
Respuesta	Variable tipo LecturaXML

Tabla 27. Descripción método LecturaXML.GetInstance

Método	LeerFicheroConfig
Descripción	Lee el fichero *configuración.xml* (apartado 5.2.4.1) que contiene diferentes parámetros de configuración que definen la funcionalidad del servicio y actualiza la variable tipo ConfigProcesado (struct).
Parámetros de entrada	• rutaXMLConfig: Ruta del fichero *configuracion.xml*
Respuesta	Nada

Tabla 28. Descripción método LecturaXML.LeerFicheroConfig

Figura 60. Gráfico de llamadas LecturaXML.LeerFicheroConfig

Método	LeerURLsServers
Descripción	Lee el fichero *servidores.xml* (apartado 5.2.4.3) que contiene las URLs de los servidores externos a los que se va a acceder y actualiza la tabla "dtServersConfig" de la variable tipo ConfigProcesado (struct).
Parámetros de entrada	• rutaXMLServer: Ruta del fichero *servidores.xml*
Respuesta	Nada

Tabla 29. Descripción método LecturaXML.LeerURLsServer

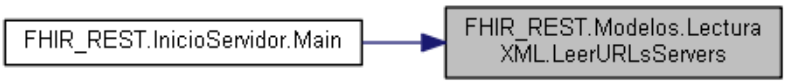


Figura 61. Gráfico de llamadas LecturaXML.LeerURLsServer

5.2.3.4 QueryPerson

Contiene la estructura y los métodos necesarios para procesar la peticion HTTP realizada por el usuario. Valida y descompone la petición, almacenando la información en una variable tipo QueryProcesada (struct).

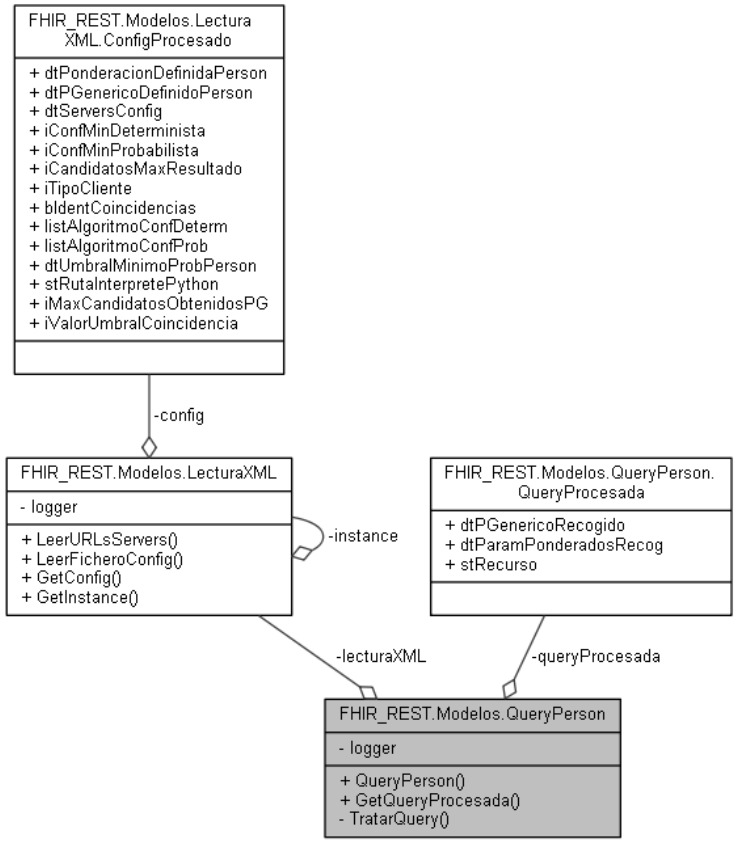


Figura 62. Diagrama de colaboración de QueryPerson

Método	GetQueryProcesada
Descripción	Método Get que retorna la variable tipo QueryProcesada que contiene la información relevante de la petición.
Parámetros de entrada	Nada
Respuesta	Variable tipo QueryProcesada (struct).

Tabla 30. Descripción método QueryPerson.GetQueryProcesada

Método	TratarQuery
Descripción	Se encarga de validar que cada parámetro en la query esté definido para el recurso *person*, asignar la ponderación correspondiente definida en *configuracion.xml* para cada parámetro recogido (se almacenan en la tabla dtParamPonderadosRecog de la variable queryProcesada en la clase LecturaXML) y de detectar el perfil genérico (se almacena en la tabla dtPGenericoRecogido). Se registran los parámetros marcados como PG para la búsqueda Exhaustiva si BusquedaExhaustiva=True (*configuracion.xml*). Si no se detecta ningún parámetro marcado como PG o BusquedaExhaustiva=False, se utiliza el parámetro de mayor ponderación.
Parámetros de entrada	• stQuery: Contiene la petición HTTP enviada por el usuario (recurso + parámetros y valores).
Respuesta	Variable de tipo QueryProcesada (struct) donde se almacena la petición HTTP descompuesta.

Tabla 31. Descripción método QueryPerson.TratarQuery

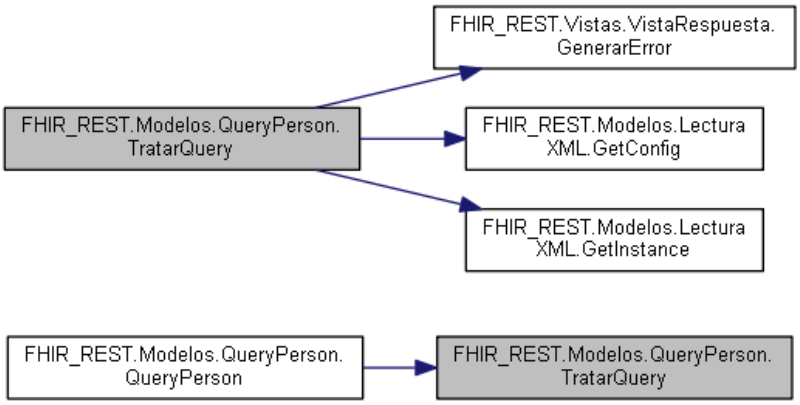


Figura 63. Gráfico de llamadas QueryPerson.TratarQuery

5.2.4 Paquete *Configuracion*

Este paquete incluye los diferentes archivos de configuración de la aplicación. Mediante los parámetros incluidos en estos ficheros, el gestor del servicio puede definir la funcionalidad deseada del servidor. Esta información sólo sera procesada una única vez al iniciar dicho servidor, por lo que si se desea realizar alguna modificación, será necesario un reinicio del sistema.

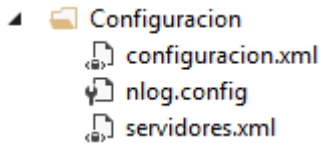


Figura 64. Contenido del paquete Configuracion

5.2.4.1 configuracion.xml

Este fichero consta de diferentes parámetros de configuración para la funcionalidad del servicio, detallados en la tabla siguiente.

Parámetro	Valores posibles	Resumen
BusquedaExhaustiva	True/False	Realiza o no una búsqueda exhaustiva. En caso de no estar activa, se selecciona de los parámetros introducidos por el usuario, únicamente el de mayor ponderación y se utiliza como perfil genérico. De lo contrario, si está activa, detecta de los parámetros de entrada los que se han configurado manualmente para dicha búsqueda exhaustiva (indicado en *PonderacionParametros*), utilizando todos los detectados como perfil genérico.
MaxCandidatosObtenidosPG	Número entero positivo	Número máximo de candidatos que permite obtener durante la primera consulta a partir del perfil genérico (antes de realizar validaciones).
AnalisisDeterminista	True/False	Realiza o no la validación determinista (ver apartado 4.7.4)
ConfianzaMinDeterminista	Número entero positivo entre 1 y 4	Indica el nivel mínimo de confianza que debe cumplir cada candidato durante la validación determinista. En caso de no cumplirse, se descarta.
AnalisisProbabilista	True/False	Realiza o no la validación determinista (ver apartado 4.7.5). Si ambos tipos de análisis estan desactivados, el servicio no se inicia indicando dicha causa.
ConfianzaMinProbabilista	Número entero positivo entre 1 y 4	Indica el nivel mínimo de confianza que debe cumplir cada candidato durante la validación probabilista. En caso de no cumplirse, se descarta.
RealizarIdentCoincidencias	True/False	Realiza o no el proceso de identificación de coincidencias.
ValorUmbralCoincidencia	Número entero positivo entre 1 y 15	Umbral mínimo de la suma de ponderaciones realizadas en la comparación de campos entre pares de candidatos para que se consideren duplicados (proceso de identificación de coincidencias). Se realiza la comparación de 15 campos (pertenecientes al recurso *person*) por cada par de candidatos, sumando el total de ponderaciones y verificando si es superior o no al umbral mínimo fijado.
PonderacionParametrosPerson	Decimales entre 0 y 1 (compuesto por varios campos)	Se fijan los valores de las diferentes ponderaciones para los parámetros de búsqueda permitidos en el recurso *Person*. Para marcar los parámetros deseados en las búsquedas exhaustivas, se introduce el atributo *busquedaExhaustiva=true* en dichos

		parámetros.
CandidatosMaxDevuelto	Número entero positivo	Número máximo de candidatos devueltos al usuario. Si el número de candidatos finales recuperados por el servicio es mayor que este parámetro, se descartan los sobrantes de menor nivel de confianza.
AlgoritmoConfianzaDeterminista	Decimales ascendentes entre 0 y 1 (compuesto por varios campos)	Con las cinco fronteras (decimales ascendentes) se marcan las regiones correspondientes a los cuatro niveles de confianza que se van a generar durante la validación determinista.
AlgoritmoConfianzaProbabilista	Decimales ascendentes entre 0 y 1 (compuesto por varios campos)	Con las cinco fronteras (decimales ascendentes) se marcan las regiones correspondientes a los cuatro niveles de confianza que se van a generar durante la validación probabilista.
PonderacionProbabilistaPerson-UmbralMinimo	Números enteros positivos entre 0 y 100 (compuesto por varios campos)	Se fijan los valores de umbrales mínimos en la ponderación probabilista de los parámetros solicitados en la búsqueda. Si algún parámetro es ponderado por debajo del umbral mínimo, se fija a cero.
RutaInterpretePyhton	Ruta accesible	Ruta donde se encuentra el intérprete de python, necesario para la ejecución del script encargado de la identificación de duplicados.

Tabla 32. Descripción parámetros del fichero configuracion.xml

configuracion.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuracion >
  <BusquedaExhaustiva> False </BusquedaExhaustiva>

  <MaxCandidatosObtenidosPG>120</MaxCandidatosObtenidosPG>

  <AnalisisDeterminista>True</AnalisisDeterminista>
  <ConfianzaMinDeterminista> 2 </ConfianzaMinDeterminista>

  <AnalisisProbabilista>True</AnalisisProbabilista>
  <ConfianzaMinProbabilista> 3 </ConfianzaMinProbabilista>

  <RealizarDeduplicacion>True</RealizarDeduplicacion>
  <ValorCoincidenciaDeduplicacion>4</ValorCoincidenciaDeduplicacion>

  <PonderacionParametrosPerson>
    <address>0,6</address>
    <address-city busquedaExhaustiva="true">0,6</address-city>
    <address-country>0,7</address-country>
    <address-postalcode>0,5</address-postalcode>
    <address-state>0,5</address-state>
    <address-use>0,6</address-use>
    <birthdate busquedaExhaustiva="true"> 0,9</birthdate>
    <email> 0,7</email>
    <gender> 0,5</gender>
    <identifier busquedaExhaustiva="true"> 0,9</identifier>
    <link> 0,7</link>
    <name> 0,8</name>
    <organization> 0,7</organization>
    <patient> 0,5</patient>
    <phone busquedaExhaustiva="true"> 0,7</phone>
    <phonetic> 0,6</phonetic>
    <practitioner> 0,5</practitioner>
    <relatedperson> 0,5</relatedperson>
    <telecom> 0.7</telecom>
```

Figura 65. Parte del fichero configuracion.xml

5.2.4.2 nlog.config

En este fichero se configura el sistema de trazado NLog mediante diferentes parámetros que permiten una gran variedad de reglas y que pueden ser consultados en la web (33).

nlog.config

```xml
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      autoReload="true"
      internalLogLevel="Info"
      internalLogFile="c:\temp\internal-nlog.txt">

  <!-- enable asp.net core layout renderers -->
  <extensions>
    <add assembly="NLog.Web.AspNetCore"/>
  </extensions>

  <!-- the targets to write to -->
  <targets>
    <!-- write logs to file  -->
    <target xsi:type="File" name="allfile" fileName="c:\temp\nlog-all-${shortdate}.log"
            layout="${longdate}|${event-properties:item=EventId_Id}|${uppercase:${level}}|${logger}|${message} ${exception:format=tostring}"

    <!-- another file log, only own logs. Uses some ASP.NET core renderers -->
    <target xsi:type="File" name="ownFile-web" fileName="c:\temp\nlog-own-${shortdate}.log"
            layout="${longdate}|${event-properties:item=EventId_Id}|${uppercase:${level}}|${logger}|${message} ${exception:format=tostring}|

    <target name="logconsole" xsi:type="Console" />

  </targets>

  <!-- rules to map from logger name to target -->
  <rules>
    <!--All logs, including from Microsoft-->
    <logger name="*" minlevel="Trace" writeTo="allfile" />

    <!--Skip non-critical Microsoft logs and so log only own logs-->
    <logger name="Microsoft.*" maxlevel="Info" final="true" /> <!-- BlackHole without writeTo -->
    <logger name="*" minlevel="Trace" writeTo="ownFile-web" />
```

Figura 66. Parte del fichero nlog.config

5.2.4.3 servidores.xml

En este fichero se introducen únicamente la dirección de los servidores externos FHIR a los que el servicio accederá para realizar las diferentes consultas oportunas en cada búsqueda.

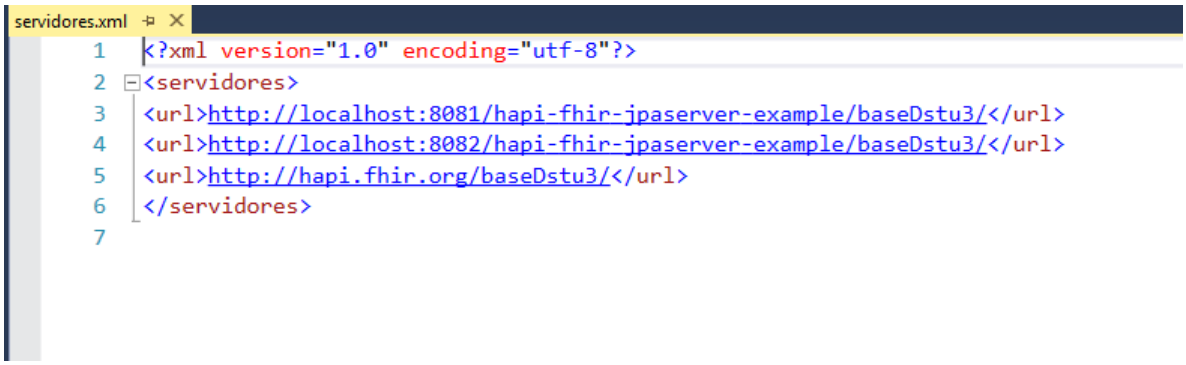


Figura 67. Fichero servidores.xml

5.2.5 Paquete *Otros*

Este paquete contiene archivos de diferente utilidad que serán necesarios para el proceso de búsqueda.

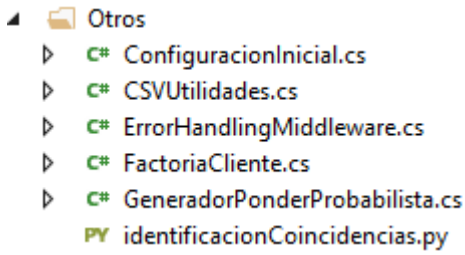


Figura 68. Contenido del paquete Otros

5.2.5.1 ConfiguracionInicial

La clase ConfiguracionInicial configura los servicios y el control de las solicitudes y respuestas. Esta configuración no tiene relación con los diferentes ficheros incluidos en la carpeta *configuracion* (apartado 5.2.4).

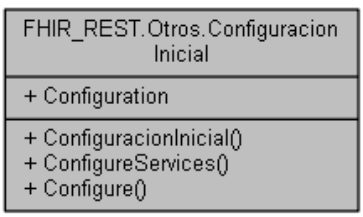


Figura 69. Diagrama de colaboración de ConfiguracionInicial

Método	Configure

Descripción	Especifica la forma en que la aplicación responde a las solicitudes HTTP. El control de solicitudes se configura mediante la adición de componentes de middleware a una instancia de tipo IApplicationBuilder. El método de extensión UseMvc agrega el middleware de enrutamiento al control de solicitudes y establece MVC como controlador predeterminado. Se especifica el servicio adicional IHostingEnvironment en los parámetros del método para configurar servicios según el entorno fijado (desarrollo, producción, etc). Además, se establece la clase ErrorHandlingMiddleware (apartado 5.2.5.4) como manejador de errores.
Parámetros de entrada	• app: Instancia IApplicationBuilder a la que se añaden los components de middleware. • env: Servicio adicional IHostingEnvironment.
Respuesta	Nada

Tabla 33. Descripción método ConfiguracionInicial.Configure

Método	ConfigureServices
Descripción	Se llama antes que el método Configure para configurar los servicios de la aplicación. Añade servicios al contenedor para que estén disponibles cuando se ejecute el método Configure.
Parámetros de entrada	• services: Colección de servicios que se van a añadir al contenedor.
Respuesta	Nada

Tabla 34. Descripción método ConfiguracionInicial.ConfigureServices

5.2.5.2 CSVUtilidades

Contiene funciones útiles para pasar de tabla (Datatable) a ficheros CSV (comma-separated values) y viceversa. Es una clase estática y no se instancia durante el funcionamiento del servicio.

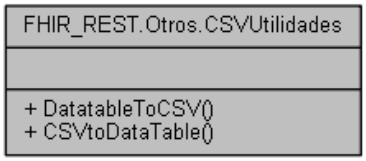


Figura 70. Diagrama de colaboración de CSVUtilidades

Método	DatatableToCSV
Descripción	Se crea un fichero .csv donde aparecen las diferentes columnas del datatable separadas por comas.
Parámetros de entrada	• dtDatatable: Datatable que se va a escribir en un fichero .csv • strFilePath: Ruta donde se va a crear el fichero

Respuesta	Nada

Tabla 35. Descripción método CSVUtilidades.DatatableToCSV

Figura 71. Gráfico de llamadas CSVUtilidades.DatatableToCSV

Método	CSVtoDataTable
Descripción	Lee un fichero con formato .csv (campos separados por comas) y devuelve una tabla (Datatable) con la información.
Parámetros de entrada	• strFilePath: Ruta donde se encuentra el fichero a leer.
Respuesta	Tabla (Datatable) con la información obtenida.

Tabla 36. Descripción método CSVUtilidades.CSVtoDataTable

Figura 72. Gráfico de llamadas CSVUtilidades.CSVtoDataTable

5.2.5.3 IdentificacionCoincidencias.py

Script en python encargado de realizar el proceso de identificación de coincidencias, mediante el procesamiento del fichero .csv, que contiene los candidatos con los valores de los diferentes campos a comparar.

La clase ProxyFHIR (apartado 5.2.1.7) se encarga de ejecutar este script mediante la inicialización de un proceso nuevo en el método EjecutarIdentCoincidenciasPython. El intérprete python recibe como parámetros de entrada la ruta del fichero y el valor mínimo en la comparación de cada par de candidatos para que sea identificado como coincidencia (*ValorUmbralCoincidencia),* fijado en *configuracion.xml (*apartado 5.2.4.1).

En primer lugar, el script procesa el fichero .csv y realiza una indexación para formar pares de registros a comparar. Existen diferentes algoritmos de indexación disponibles (36). Una indexación completa es un índice con todas las posibles combinaciones de pares, lo que en ocasiones puede resultar un trabajo demasiado costoso. La indexación óptima es aquella que busca reducir el número de comparaciones sin que dicha disminución empeore el número de registros duplicados detectados. Para este proyecto, se utiliza el método de bloqueo, recogiendo todos los pares de registros que concuerdan en alguno de los campos que se hayan establecido. Se agrupan todos aquellos pares que coinciden en el campo identificador, teléfono, correo, nombre o fecha de nacimiento.

En segundo lugar, se procesa la agrupación del paso anterior, realizando una comparación de los diferentes campos, ponderando el nivel de similitud por cada par y calculando el total de puntuación obtenida para cada pareja de candidatos. Se utiliza el algoritmo de comparación más eficaz, según el tipo de dato que se esté procesando (37). Se puede observar en la siguiente figura la parte de código encargada de estas

comparaciones.

```
compare_cl = rl.Compare()
compare_cl.exact('Use', 'Use', label='Use')
compare_cl.exact('Gender', 'Gender', label='Gender')
compare_cl.exact('PostalCode', 'PostalCode', label='PostalCode')
compare_cl.exact('Birthdate', 'Birthdate', label='Birthdate')
compare_cl.exact('Identifier', 'Identifier', label='Identifier')
compare_cl.string('City', 'City', method='jarowinkler', threshold=0.85, label='City')
compare_cl.string('Country', 'Country', method='jarowinkler', threshold=0.85, label='Country')
compare_cl.string('State', 'State', method='jarowinkler', threshold=0.85, label='State')
compare_cl.string('Email', 'Email', method='jarowinkler', threshold=0.80, label='Email')
compare_cl.string('Family', 'Family', method='jarowinkler', threshold=0.80, label='Family')
compare_cl.string('Given', 'Given', method='jarowinkler', threshold=0.80, label='Given')
compare_cl.string('Prefix', 'Prefix', method='jarowinkler', threshold=0.80, label='Prefix')
compare_cl.string('Suffix', 'Suffix', method='jarowinkler', threshold=0.80, label='Suffix')
compare_cl.exact('Phone', 'Phone', label='Phone')
```

Figura 73. Parte de código del script IdentificacionCoincidencias.py

Las comparaciones exactas son aquellas que detectan únicamente una coincidencia total, lo que podría identificarse como una comparación determinista. Se utiliza para los campos código postal, fecha de nacimiento, teléfono, etc, donde no suelen producirse errores de formato, y para aquellos que sólo permiten un conjunto de valores, como el campo género (masculino o femenino).

Las comparaciones que realizan una ponderación en el nivel de similitud (probabilistas), contienen los parámetros *method* y *threshold,* para definer el método a utilizar y el umbral mínimo. Se ha considerado oportuno utilizar el método Jaro-Winkler (38), algoritmo que mide la distancia de edición entre dos cadenas, otorgando mayor importancia a las cadenas que coinciden desde el principio. Aquellos campos que tengan una coincidencia total, serán calificados con 1, mientras que los que hayan sido ponderados con un decimal inferior al umbral mínimo, serán calificados con 0.

En tercer lugar, se realiza la clasificación, marcando como coincidencias todas aquellas parejas cuyo nivel de similitud total (suma de todos los campos ponderados) sea superior o igual al valor pasado como parámetro de entrada (*ValorUmbralCoincidencia).*

Por último, se sobreescribe el fichero procesado de entrada con otro fichero que contiene los pares de registros marcados como coincidencias.

5.2.5.4 ErrorHandlingMiddleware.cs

Maneja los errores producidos durante la ejecución del servidor.

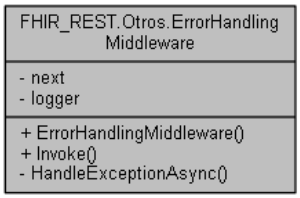


Figura 74. Diagrama de colaboración de ErrorHandlingMiddleware

Método	Invoke
Descripción	Hace que el manejo de errores se realice de forma asíncrona.

Parámetros de entrada	• context: Contexto HTTP
Respuesta	Nada

Tabla 37. Descripción método ErrorHandlingMiddleware.Invoke

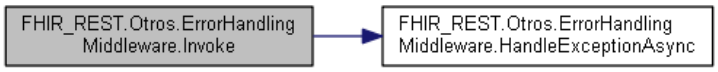


Figura 75. Gráfico de llamadas ErrorHandlingMiddleware.Invoke

Método	HandleExceptionAsync
Descripción	Maneja las excepciones producidas, indentificando el tipo de excepción y convirtiendo en Json la respuesta (recurso *OperationOutcome* en la clase MyBadRequestException). Se encarga de establecer el código de error correspondiente.
Parámetros de entrada	• context: Contexto HTTP • exception: Excepción producida
Respuesta	Respuesta Json que se envía al usuario

Tabla 38. Descripción método ErrorHandlingMiddleware.HandleExceptionAsync

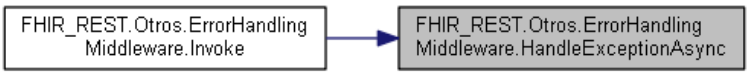


Figura 76. Gráfico de llamadas ErrorHandlingMiddleware.HandleExceptionAsync

5.2.5.5 FactoriaCliente

Se encarga de crear la instancia del tipo de cliente necesario para las validaciones (determinista, probabilista o mixto).

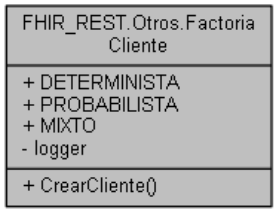


Figura 77. Diagrama de colaboración de FactoriaCliente

Método	CrearCliente
Descripción	Genera una instancia de un tipo de cliente determinado, necesario para realizar las validaciones de los candidatos. Dicho cliente hereda de ClienteREST e implementa la interfaz ICliente.

Parámetros de entrada	• iTipo: Entero que indica el tipo de cliente que se va a instanciar.
Respuesta	Objeto tipo ICliente (interfaz implementada por los tres tipos de clientes definidos)

Tabla 39. Descripción método FactoriaCliente.CrearCliente

Figura 78. Gráfico de llamadas FactoriaCliente.CrearCliente

5.2.5.6 GeneradorPonderProbabilista

Realiza la ponderación de parámetros de los candidatos durante el análisis probabilista.

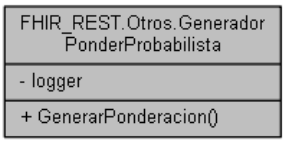


Figura 79. Diagrama de colaboración de GeneradorPonderProbabilista

Método	GenerarPonderacion
Descripción	Recupera del recurso *Person* el valor del parámetro que se va a comparar y genera una ponderacion mediante el algoritmo de la Distancia de Levenshtein (39). Dicha distancia es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Además, se ha definido en *configuracion.xml* el umbral mínimo para cada parámetro (*PonderacionProbabilistaPerson-UmbralMinimo*). En el caso de que la ponderación sea inferior a dicho umbral, se retorna el valor cero.
Parámetros de entrada	• parametro: Nombre y valor del parámetro que el usuario ha introducido en la petición HTTP. • persona: Recurso Person del candidato que se está ponderando. • stURLcandidato: Identificador único del candidato que se está ponderando. • Cliente: Instancia de ClienteRest que se utilizará en el caso de que haya que generar ponderación para el parámetro phonetic.
Respuesta	Decimal entre cero y uno que indica el nivel de similitud del parámetro.

Tabla 40. Descripción método GeneradorPonderProbabilista.GenerarPonderacion

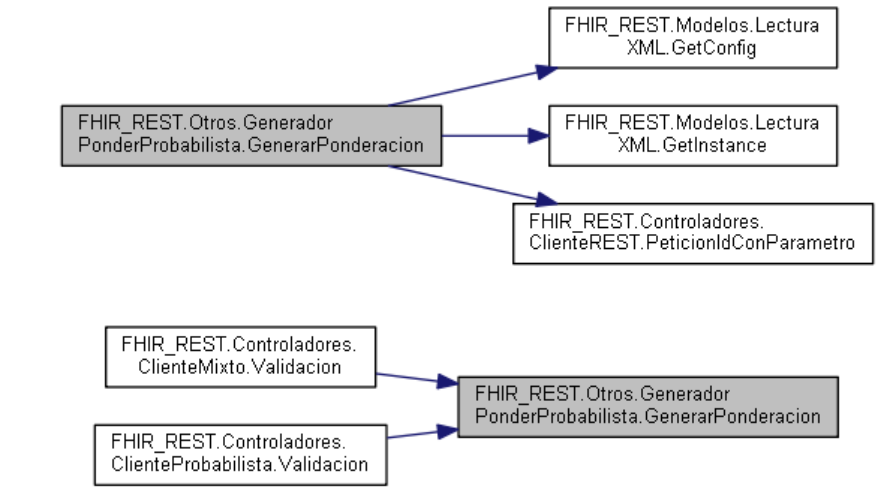


Figura 80. Gráfico de llamadas de GeneradorPonderProbabilista.GenerarPonderacion

6 CONCLUSIONES

En este capítulo se tratan las principales dificultades encontradas a lo largo del desarrollo del proyecto y cómo estas han sido solucionadas. Además, se analizan las posibles líneas futuras de desarrollo, para ampliar y mejorar el servicio implementado. Por último, se finaliza con una conclusión personal.

6.1.1 Esfuerzos necesarios

- Estándar HL7 FHIR: En este proyecto se ha estudiado este estándar para el diseño y desarrollo del servicio. Este primer contacto con un sistema sanitario ha implicado un análisis profundo de FHIR. Se han realizado numerosas lecturas y consultas de la especificación a lo largo del trabajo para la asimilación de los conceptos que maneja este estándar. Además de la especificación, se han estudiado las diferentes funcionalidades y opciones que permite la librería FHIR para la realización de búsquedas en los diferentes servidores.
- Búsqueda y análisis de librerías: Se ha llevado a cabo una investigación previa con el objetivo de reutilizar librerías externas para reducir y optimizar el código desarrollado.
- Desarrollo en lenguaje C Sharp: El proyecto ha sido codificado mayoritamente en lenguaje C Sharp mediante Visual Studio. Este lenguaje y entorno eran prácticamente desconocidos para mí. Se ha afrontado el desarrollo mediante la consulta de tutoriales encontrados en internet y de la documentación oficial, donde he conseguido adquirir conocimientos básicos para la implementación.
- Desarrollo en lenguaje Python: El proceso de identificación de coincidencias del servicio se ha codificado mediante un script en lenguaje Python, debido a las diferentes librerías útiles que nos aporta este lenguaje. Python era también completamente desconocido para mí, por lo que ha sido necesario la consulta de diferentes tutoriales y documentación oficial para la utilización de la librería escogida (apartado 3.8, Record Linkage).
- Análisis probabilista: Para que el servicio realice una ponderación probabilista de los diferentes parámetros de búsqueda, es necesario acceder a un campo determinado de un recurso, funcionalidad no implementada por la librería FHIR. Además, hay que tener en cuenta la posibilidad de que un parámetro puede incluir más de un campo, por lo que es necesario ponderar cada uno de ellos y devolver el de mayor puntuación (el de mayor similitud). Para afrontar este problema, se ha desarrollado una función que puede acceder a cada uno de los parámetros permitidos para el recurso *person*, ponderando todos los campos en el caso de que tenga una cardinalidad superior a uno y retornando el valor de mayor ponderación.

6.2 Línea futura de desarrollo

Este proyecto es el primer paso para el desarrollo con funcionalidad más completa de un PMI (Patient Master Index) , sistema que reúne información demográfica e identificadores de pacientes de múltiples sistemas pertenecientes a una federación y que asegura que los datos están libres de registros duplicados. En el caso de implementar el servicio web en un entorno real, existen diferentes tareas y mejoras que habría que desarrollar para que se convierta realmente en un PMI.

- Incluir más recursos FHIR en el análisis: Tal y como se ha detallado en la memoria, el servicio implementado únicamente procesa recursos de tipo *person* a partir de datos demográficos. Podría extenderse el estudio de recursos para un análisis más profundo, abarcando por ejemplo historiales clínicos.
- Mecanismos de seguridad: Se hace fundamental en un uso real del servicio donde se intercambia información sanitaria, la implementación de mecanismos de seguridad, los cuales no se han tenido en cuenta en este proyecto. FHIR no define ninguna funcionalidad relacionada con la seguridad, pero sí define los protocolos que deben utilizarse (12):

1. Sincronización de relojes mediante NTP/SNTP.
2. Intercambio de datos mediante TLS/SSL.
3. Sistemas de autenticación de usuarios/clientes.
4. Sistemas de autorización y control de acceso.
5. Recursos de procedencia y auditoría de eventos.
6. Uso de firmas digitales
7. Precacución en los archivos adjuntos.
8. Uso de etiquetas relacionadas con la seguridad
9. Políticas de gestión de datos
10. Precaución a la hora de mostrar la narrativa de los recursos FHIR.
11. Validación de los datos de entrada

- Aumento de funcionalidades: En este proyecto únicamente se ha implementado la operación GET para la realización de búsquedas por parte del usuario. Se podría aumentar esta funcionalidad si se incluyen otras operaciones como CREATE o DELETE.
- Base de datos local: Podría ser interesante el uso de una base de datos local donde se almacene un histórico de los resultados de búsquedas realizadas por los usuarios. Además, podrían registrarse los recursos identificados como coincidentes para optimizar posibles búsquedas futuras repetidas.
- Tratamiento de los registros coincidentes: El servicio podría notificar automáticamente a aquellos sistemas que contengan algún registro duplicado o podría mostrar en la respuesta al usuario directamente una fusión de los recursos coincidentes.

6.3 Conclusión

El desarrollo del proyecto me ha servido para profundizar los conocimientos adquiridos a lo largo de la carrera, especialmente en el concepto de servicio REST, incluido en la rama de telemática. Estos tipos de servicios son una de las tecnologías más importantes para las aplicaciones web. Su importancia va a seguir creciendo a medida que las empresas busquen proporcionar interfaces abiertas bien definidas para los servicios ofrecidos, por lo que se considera fundamental que los desarrolladores actuales tengan una clara comprensión de esta tecnología.

La codificación del sistema me ha brindado la oportunidad de adquirir nuevos conocimientos sobre los lenguajes de programación C Sharp y Python, que no se habían estudiado a lo largo del grado. Estos dos lenguajes se encuentran actualmente entre los más demandados por las empresas, por lo que esta primera experiencia me hace más competente de cara al futuro próximo.

Además de todo lo aprendido, entender la importancia de las TICs en el sector de la salud y tener una primera toma de contacto con un estándar sanitario, me ha creado un gran interés en este ámbito, añadiendo otra posibilidad a las diferentes líneas de trabajo para orientar mi vida laboral.

7 BIBLIOGRAFÍA

1. *eHealth system interoperability.* **Weber-Jahnke, Jens , Peyton, Liam y Topaloglou, Thodoros.** March de 2012, Information Systems Frontiers, Vol. 14, págs. 1-3.

2. HL7.org. *FHIR Overview.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.hl7.org/fhir/overview.html.

3. HL7.org. *HL7 Version 2 Product Suite.* [En línea] [Citado el: 10 de Abril de 2019.] http://www.hl7.org/implement/standards/product_brief.cfm?product_id=185.

4. HL7.org. *HL7 Version 3 Product Suite.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.hl7.org/implement/standards/product_brief.cfm?product_id=186.

5. HL7.org. *CDA Release 2.* [En línea] [Citado el: 10 de Abril de 2019.] http://www.hl7.org/implement/standards/product_brief.cfm?product_id=7.

6. **Christen, Peter.** *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* s.l. : Springer Publishing, 2012.

7. Verato. *Referential Matching: The Silver Bullet for Patient Identity and Patient Matching.* [En línea] [Citado el: 10 de Abril de 2019.] https://verato.com/what-is-referential-matching-a-primer-patient-matching-technology-interoperability/.

8. Healthitanalytics. *Data Integrity Strategies for Patient Matching, Identification.* [En línea] [Citado el: 10 de Abril de 2019.] https://healthitanalytics.com/features/data-integrity-strategies-for-patient-matching-identification.

9. The Sequoia project. [En línea] 2018. [Citado el: 10 de Abril de 2019.] https://sequoiaproject.org/wp-content/uploads/2018/06/The-Sequoia-Project-Framework-for-Patient-Identity-Management-v31.pdf.

10. **Kurtz, Jamie y Wortman, Brian.** ASP.NET Web API 2: Building a REST Service from Start to Finish. s.l. : Apress, Berkeley, CA, 2014, 2, págs. 9-19.

11. HL7.org. *Reference Information Model.* [En línea] [Citado el: 10 de Abril de 2018.] http://www.hl7.org/implement/standards/rim.cfm.

12. HL7.org. *RESTful API.* [En línea] [Citado el: Abril de 10 de 2019.] https://www.hl7.org/fhir/http.html.

13. HL7.org. *FHIR Security.* [En línea] [Citado el: 10 de Abril de 2010.] https://www.hl7.org/fhir/security.html.

14. HL7.org. *Resource Index.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.hl7.org/fhir/resourcelist.html.

15. HL7.org. *Data Types.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.hl7.org/fhir/datatypes.html.

16. HL7.org. *Resource Boundle.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.hl7.org/fhir/bundle.html.

17. HL7.org. *Resource Person.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.hl7.org/fhir/person.html.

18. HL7.org. *Resource OperationOutcome.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.hl7.org/fhir/operationoutcome.html.

19. HL7.org. *Search.* [En línea] [Citado el: 10 de Abril de 2010.] https://www.hl7.org/fhir/search.html.

20. C Sharp. *Documentation.* [En línea] [Citado el: 10 de Abril de 2019.] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/.

21. Python. *Documentation.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.python.org/doc/.

22. Visual Studio. [En línea] [Citado el: 10 de Abril de 2019.] https://visualstudio.microsoft.com/es/.

23. HAPI. *HAPI FHIR.* [En línea] [Citado el: 10 de Abril de 2019.] http://hapifhir.io/.

24. *Learning HL7 FHIR Using the HAPI FHIR Server and Its Use in Medical Imaging with the SIIM Dataset.* **A. Hussain, Mohannad , G.Langer, Steve y Kohli, Marc.** June de 2018, Journal of Digital Imaging, Vol. 31, págs. 334-340.

25. Apache Tomcat. *Tomcat 8.0.* [En línea] [Citado el: 10 de Abril de 2010.] https://tomcat.apache.org/download-80.cgi.

26. Postman. [En línea] [Citado el: 10 de Abril de 2019.] https://www.getpostman.com/.

27. Apache JMeter. [En línea] [Citado el: 10 de Abril de 2019.] https://jmeter.apache.org/.

28. Json. *Documentation.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.json.org/.

29. Doxygen. *Documentation.* [En línea] [Citado el: 10 de Abril de 2019.] http://www.doxygen.nl/.

30. FuzzySharp. *C# (DotNet) Implementation.* [En línea] [Citado el: 10 de Abril de 2019.] https://github.com/BoomTownROI/BoomTown.FuzzySharp.

31. HL7 FHIR .NET. *GitHub.* [En línea] [Citado el: 10 de Abril de 2019.] https://github.com/FirelyTeam/fhir-net-api.

32. NLog. *NLog.Web.AspNetCore.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.nuget.org/packages/NLog.Web.AspNetCore/.

33. Python Record Linkage Toolkit. *Documentation.* [En línea] [Citado el: 10 de Abril de 2010.] https://recordlinkage.readthedocs.io/en/latest/index.html.

34. No Magic. *Incd. MagicDraw.* [En línea] [Citado el: 10 de Abril de 2019.] https://www.nomagic.com/products/magicdraw.

35. docs.microsoft. *DataSet.* [En línea] [Citado el: 10 de Abril de 2019.] https://docs.microsoft.com/es-es/dotnet/api/system.data.dataset?view=netframework-4.7.2.

36. docs.microsoft. *DataRelation.* [En línea] [Citado el: 10 de Abril de 2019.] https://docs.microsoft.com/es-es/dotnet/api/system.data.datarelation?view=netframework-4.7.2.

37. NLog. *GitHub - Configuration File.* [En línea] [Citado el: 10 de Abril de 2019.] https://github.com/NLog/NLog/wiki/Configuration-file.

38. Record Linkage Toolkit. *Documentation-Indexing.* [En línea] [Citado el: 10 de Abril de 2019.] https://recordlinkage.readthedocs.io/en/latest/ref-index.html.

39. Python Record Linkage Tolkit. *Documentation- Comparing.* [En línea] [Citado el: 10 de Abril de 2019.] https://recordlinkage.readthedocs.io/en/latest/ref-compare.html.

40. Jaro-Winkler distance. *Wikipedia.* [En línea] [Citado el: 10 de Abril de 2019.] https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance.

41. Levenshtein distance. *Wikipedia.* [En línea] [Citado el: 10 de Abril de 2019.] https://en.wikipedia.org/wiki/Levenshtein_distance.