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

Servicio de detección de conflictos en consentimientos de pacientes

Autor: Pablo López Vílchez
Tutor: Jorge Calvillo Arbizu

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

Sevilla, 2024

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

Servicio de detección de conflictos en consentimientos de pacientes

Autor:
Pablo López Vílchez

Tutor:
Jorge Calvillo Arbizu
Profesor Permanente Laboral

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Servicio de detección de conflictos en consentimientos de pacientes

Autor: Pablo López Vílchez

Tutor: Jorge Calvillo Arbizu

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

A lo largo de toda mi vida académica, he aprendido que los logros más significativos no se alcanzan en solitario, sino gracias al apoyo de los que nos rodean. Por eso, quiero tomarme un momento para agradecer a quienes han sido fundamentales en este camino.

En primer lugar, quiero agradecer a mi tutor, Jorge Calvillo, por su ayuda, orientación y paciencia a largo de todo el proyecto.

En segundo lugar, a Gonzalo y Óscar, más que amigos me llevo una familia de la carrera. Gracias por esos momentos de risa en las máquinas de los laboratorios, por la charlitas y apoyos en los momentos más difíciles tanto académicos como personales. Vuestra amistad ha sido un pilar fundamental en este camino, y no puedo imaginar haber llegado hasta aquí sin vosotros a mi lado.

Por último, y no menos importantes, a toda mi familia, la que siempre ha estado detrás de mí dándome el apoyo y los ánimos necesarios para seguir adelante, incluso cuando las cosas se ponían cuesta arriba. Cada logro en este camino también es suyo, porque sin su constante apoyo, nada de esto habría sido posible.

En resumen, este proyecto es el resultado del esfuerzo conjunto de todos los que me han apoyado y acompañado en este proceso. Os estaré siempre agradecido.

Pablo López Vílchez

Sevilla, 2024

Resumen

En la actualidad, el sector sanitario se encuentra en un proceso de transformación profunda, impulsado por los avances tanto técnicos como metodológicos en la gestión de datos y la toma de decisiones clínicas. Esta evolución ha permitido una mejora considerable en la recopilación y el intercambio de información sanitaria. Sin embargo, estos avances traen consigo nuevos desafíos, particularmente en lo que respecta a la coherencia y la validez de los consentimientos proporcionados por los pacientes para el tratamiento de su información clínica.

La correcta gestión de los consentimientos de los pacientes es fundamental para garantizar la confianza y la seguridad entre el paciente y el personal médico. En un entorno donde la información se comparte entre múltiples sistemas y actores, es crucial que los consentimientos proporcionados por los pacientes sean consistentes, estén actualizados y no presenten conflictos que puedan comprometer su validez o la privacidad de los datos. Este aspecto resulta de especial relevancia en el marco del Reglamento General de Protección de Datos (GDPR), que exige un manejo riguroso y transparente de la información personal, garantizando que los datos clínicos se utilicen de manera ética y conforme a la legalidad.

El principal objetivo de este trabajo es analizar, desarrollar e implementar un servicio especializado en la detección de conflictos en los consentimientos proporcionados por los pacientes. Este servicio tiene como principal finalidad la identificación de posibles inconsistencias en los consentimientos registrados, alertando sobre situaciones que podrían requerir revisión para asegurar que el uso de la información sanitaria cumpla con las preferencias establecidas por los pacientes

Para alcanzar este objetivo, se ha desarrollado un microservicio que, dado un nuevo consentimiento por un paciente, analiza los consentimientos existentes para dicho paciente, utilizando algoritmos de detección de conflictos. El diseño modular del microservicio permite su fácil integración en infraestructuras sanitarias ya existentes, ofreciendo una herramienta ágil y eficaz para la verificación continua de los consentimientos otorgados por los pacientes.

El servicio ha sido implementado con tecnologías que garantizan su compatibilidad e interoperabilidad con entornos que utilizan el estándar FHIR (*Fast Healthcare Interoperability Resources*), creado por la organización *Health Level Seven International* (HL7), garantizando una integración sencilla en diversas plataformas de gestión sanitarias.

Abstract

The healthcare sector is currently undergoing a profound change, driven by both technical and methodological advances in data management and clinical decision-making. These developments have led to considerable improvements in the collection and exchange of health information. However, these developments bring with them new challenges, particularly about the consistency and validity of consents provided by patients for the processing of their clinical information.

Proper management of patient consents is essential to ensure trust and confidence between patient and medical staff. In an environment where information is shared among multiple systems and actors, it is crucial that the consents provided by patients are consistent, up-to-date and free of conflicts that could compromise their validity or data privacy. This is particularly relevant in the context of General Data Protection Regulation (GDPR), which requires rigorous and transparent handling of personal information, ensuring that clinical data are used in an ethical and legally compliant manner.

The main objective of this work is to analyze, develop and implement a service specialized in the detection of conflicts in the consents provided by patients. The main purpose of this service is to identify possible inconsistencies in the recorded consents, alerting about situations that may require revision to ensure that the use of health information complies with the preferences established by the patients.

To achieve this goal, a microservice has been established developed that, given a new consent by a patient, analyses the existing consents for that patient, using conflict detection algorithms. The modular design of the microservice allows it to be easily integrated into existing healthcare infrastructures, offering an agile and efficient tool for the continuous verification of the consents given by patients.

The service has been implemented with technologies that guarantee its compatibility and interoperability with environments that use the FHIR (Fast Healthcare Interoperability Resources) standard, created by the Health Level Seven International (HL7) organization, guaranteeing easy integration into various healthcare management platforms.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

FHIR	Fast Healthcare Interoperabilty Resources
GDPR	Reglamento General de Protección de Datos (General Data Protection Regulation)
HL7	Health Level Seven International
REST	Transferencia de Estado Representacional (Representational State Transfer)
API	Interfaz de Programación de Aplicaciones (Application Programming Interface)
HTTP	Protocolo de transferencia de hipertexto (Hypertext Transfer Protocol)
POM	Project Object Model
JSON	Notación de Objeto de JavaScript (JavaScript Object Notation)
XML	Lenguaje de Marcado Extensible (Extensible Markup Language)
RDF	Marco de Descripción de Recursos (Resource Description Framework)
GUI	Interfaz Gráfica de Usuario (Graphical User Interface)

1 INTRODUCCIÓN

Da tu primer paso con fe, no es necesario que veas toda la escalera completa, sólo da tu primer paso.

- Martin Luther King Jr. -

En este primer capítulo se abordarán la motivación que ha llevado a la realización de este trabajo, los objetivos que se desean alcanzar con su desarrollo, y el propósito que orienta la investigación. También se mostrarán el ámbito, alcance y limitaciones de este estudio. Además, se expondrá la metodología empleada para el desarrollo del trabajo y se mostrará el plan de trabajo seguido.

1.1 Motivación y objetivos

En el contexto actual del sector sanitario, la gestión de los datos personales está sometida a una creciente complejidad y rigurosidad. La implementación del Reglamento General de Protección de Datos (GDPR) de la Unión Europea ha establecido un marco regulatorio más estricto para la protección de la información personal, donde destaca el papel esencial de los consentimientos informados como base para la legitimidad en el tratamiento de los datos [1].

El GDPR pone un especial interés en la protección de los datos sensibles relacionados con la salud, exigiendo un manejo cuidadoso y transparente de esta información. El correcto tratamiento de estos datos es crucial no solo para proteger la privacidad de la información de los pacientes, sino también para garantizar el cumplimiento de la estricta normativa europea.

Sin embargo, muchas organizaciones se enfrentan a retos significativos en la correcta aplicación de estas normas. La complejidad en el manejo de los datos personales, especialmente aquellos relacionados con al ámbito sanitario, puede dar lugar a problemas considerables. Estos desafíos incluyen no solo la gestión de los permisos de los pacientes, sino también errores humanos en la entrada de datos. Estos errores pueden aparecer en múltiples puntos de interacción, desde la obtención inicial de la información del paciente hasta su actualización y mantenimiento en los sistemas de almacenamiento o registro.

Dichos errores pueden conducir a inconsistencias en los registros, donde la información del paciente puede reflejar permisos desactualizados o incorrectos. La falta de precisión en la entrada de datos puede resultar en una mala gestión de los consentimientos, lo cual afecta a la privacidad del paciente y puede tener implicaciones legales para la organización encargada.

Esta situación destaca la necesidad de una solución integral que no solo aborde los problemas en los sistemas de gestión de los datos personales, sino que también mitigue el impacto del factor humano. Un sistema que facilite la detección e información de inconsistencias en los consentimientos de los pacientes es fundamental. Este enfoque ayudará a las organizaciones sanitarias a manejar los datos sensibles con mayor eficacia, garantizando la exactitud y consistencia en los registros.

Tras esto, se define el objetivo principal del proyecto: diseñar, desarrollar e implementar un sistema que funcione como un servicio de detección de conflictos en los consentimientos otorgados por los pacientes bajo el GDPR, utilizando el estándar FHIR como base para la interoperabilidad de la información sanitaria. Este sistema podrá integrarse con otros ya existentes de gestión de consentimientos [2]. El objetivo lo podemos desglosar en los siguientes puntos:

- Estudio y comprensión del estándar FHIR: para el almacenamiento, intercambio y comunicación de información sanitaria.
- Diseño de un modelo de detección de conflictos que identifique inconsistencias en los consentimientos de los pacientes, basándonos en el marco regulatorio del GDPR y lo especificado por el estándar FHIR.
- Desarrollo e implementación de un microservicio que realice la verificación continua de los consentimientos, capaz de integrarse en las infraestructuras sanitarias existentes y accediendo a las bases de datos FHIR para analizar los datos en tiempo real.
- Pruebas y validación del microservicio para asegurar su correcto funcionamiento y precisión en la detección de conflictos.

En la siguiente figura, se muestra el esquema del sistema que vamos a diseñar, desarrollar e implementar. Se trata de un servicio web que, mediante llamadas a distintas API, recibirá el nuevo consentimiento introducido por el paciente y en función de la versión FHIR que se trate (R4 o R5) se obtendrán los consentimientos existentes para ese mismo paciente y se procederá a la búsqueda de posibles conflictos. Una vez terminado se informará al sistema de gestión o al usuario final de los conflictos detectados.

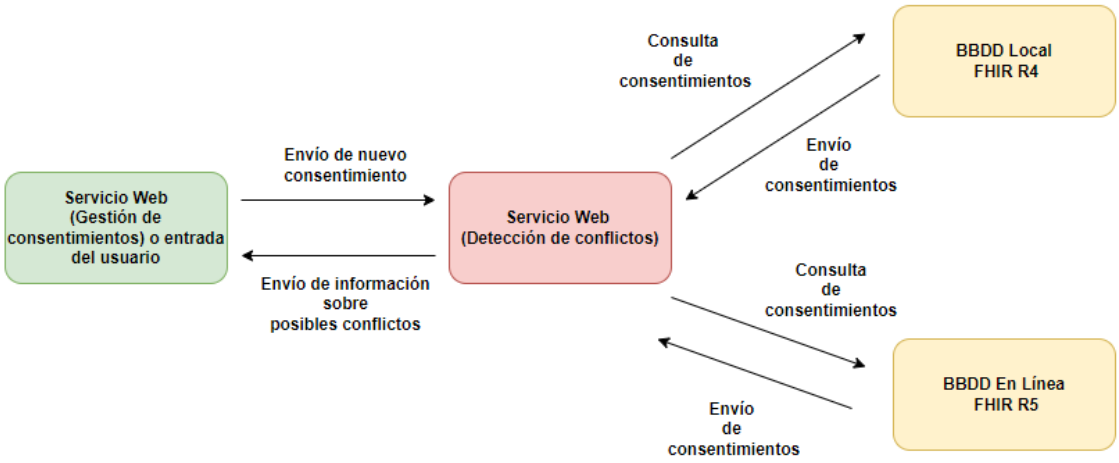


Figura 1-1. Funcionamiento del sistema.

1.2 Metodología empleada

Para la realización de este proyecto, se ha propuesto una metodología incremental. Esta metodología destaca por su facilidad para adaptarse a los cambios y su flexibilidad ante los requisitos. Siguiendo esta estrategia, se desarrollarán versiones sucesivas del trabajo, cada una introduciendo mejoras sobre la anterior. De este modo, se conseguirá un avance progresivo y ordenado, asegurando que cada etapa o incremento aporte valor y que los

posibles errores se detecten y solucionen a tiempo.

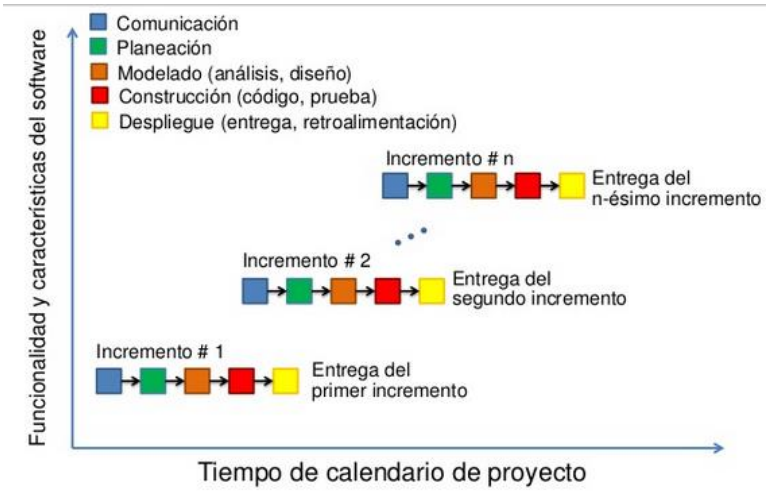


Figura 1-2. Modelo Incremental.

En cada etapa, se buscará optimizar el código y mejorar la calidad de este, garantizando un progreso constante y correctamente ordenado. Este enfoque no solo facilitará la detección y corrección temprana de errores, sino que también contribuirá a mantener un proyecto limpio y de fácil manejo.

1.3 Plan de trabajo

Se adjunta la siguiente tabla, en la que se pueden observar todas las tareas que se han realizado a lo largo del proyecto.

Tabla 1–1. Planificación temporal

Tareas	Horas Estimadas	Horas Reales
Estudio y comprensión del estándar FHIR (R4 y R5)	24	18
Estudio, despliegue y prueba de la BD local FHIR R4	5	5
Estudio y prueba de la BD en línea FHIR R5	7	6
Estudio de los recursos FHIR a implementar	6	8
Estudio de la librería FHIR	15	20
Diseño de la lógica de detección de conflictos	15	20
Diseño de la arquitectura del proyecto	8	10
Implementación del código	40	50
Pruebas y evaluación de los resultados obtenidos	60	50
Corrección y refactorización del código	80	100
Total de horas	260	287

2 ESTADO DEL ARTE

En este capítulo vamos a describir desde un punto de vista teórico el punto de partida del proyecto. Se tratarán las principales cuestiones sobre el GDPR y su importancia en la gestión de consentimientos en el ámbito de la salud, así como el uso del estándar FHIR para garantizar la interoperabilidad de la información sanitaria. Junto al estándar se presenta la implementación que hemos usado en este proyecto, así como los recursos que hemos utilizado. Por último, se describirán las herramientas y plataformas utilizadas a lo largo de todo el proyecto.

2.1 Marco teórico

2.1.1 Reglamento General de Protección de Datos (GDPR)

El Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, comúnmente conocido como Reglamento General de Protección de Datos (GDPR), establece un marco legal para la protección de los datos personales de los ciudadanos de la Unión Europea.

Dentro de todos los campos que abarca esta normativa, los datos personales relativos a la salud se consideran especialmente sensibles y por tanto reciben un nivel de protección superior [3], lo cual implica un manejo riguroso y cuidadoso de los mismos. Esta clasificación es de gran importancia en el sector sanitario, ya que garantiza que la información de los pacientes sea gestionada de la manera más segura posible para preservar su privacidad.

Este reglamento define una serie de derechos que los ciudadanos de la UE pueden ejercer sobre el tratamiento de su información personal [4]. En el contexto de este proyecto, la detección de conflictos en consentimientos de pacientes, algunos de estos derechos son especialmente importantes. A continuación, se presentan dichos derechos:

- **Transparencia en la información**: Este derecho garantiza que los pacientes estén completamente informados de forma clara y concisa sobre cómo se gestionan sus datos personales. Si la comunicación es insuficiente o confusa podrían darse conflictos en la interpretación y la aplicación de los consentimientos.
- **Derecho de acceso del interesado**: Los pacientes tienen derecho a acceder a sus datos personales y a la información sobre cómo están siendo utilizados estos datos. Este derecho permite al paciente comprobar que sus consentimientos se han aplicado correctamente.
- **Derecho de rectificación**: Este derecho permite a los pacientes corregir datos personales incompletos o inexactos. Este derecho es crucial en la gestión de consentimientos, ya que la rectificación evita inconsistencias que podrían generar conflictos en el tratamiento de la información del paciente.
- **Derecho de supresión (<<el derecho al olvido>>)**: Este derecho permite a los pacientes solicitar la eliminación de sus datos personales cuando dejan de ser necesarios para el fin que fueron recogidos. Si no se aplica correctamente podrían darse conflictos al eliminarse datos todavía en uso para algunos consentimientos.
- **Derecho a la limitación del tratamiento**: Este derecho permite a los pacientes limitar el tratamiento de sus datos personales en ciertas situaciones. La limitación del tratamiento podría dar lugar a conflictos si no se gestiona correctamente en relación con consentimientos anteriores o activos actualmente.

- **Derecho a la portabilidad de los datos**: Este derecho facilita que los pacientes puedan transferir su información personal entre distintos responsables del tratamiento. La portabilidad debe manejarse con cuidado para evitar que el traspaso de datos genere inconsistencias o conflictos.
- **Derecho de oposición**: Este derecho permite a los pacientes oponerse al tratamiento de sus datos personales en determinadas circunstancias. Está oposición podría entrar en conflicto con consentimientos previos, por lo que resulta de especial importancia que este derecho sea aplicado correctamente para mantener la coherencia con los consentimientos registrados.

2.1.2 Fast Healthcare Interoperability Resources (FHIR)

El estándar FHIR (*Fast Healthcare Interoperability Resources*) es un marco que facilita la interoperabilidad en el ámbito sanitario, desarrollado por HL7 (*Health Level Seven International*) [5]. FHIR está diseñado para que el intercambio de información clínica entre diferentes sistemas sea de manera eficiente, aprovechando las tecnologías web modernas y enfocándose en facilitar su implementación. FHIR incluye un protocolo RESTful basado en HTTP y una representación de datos que usa JSON, XML y RDF.

FHIR se organiza en módulos en distintos niveles, cada uno de ellos abordando aspectos específicos en el intercambio de la información. Estos niveles permiten una implementación gradual y modular, adaptándose a las necesidades y contextos dentro de todo el ámbito sanitario. A continuación, se muestra en la Figura 2-1 los niveles existentes y se detalla brevemente cada uno de ellos:

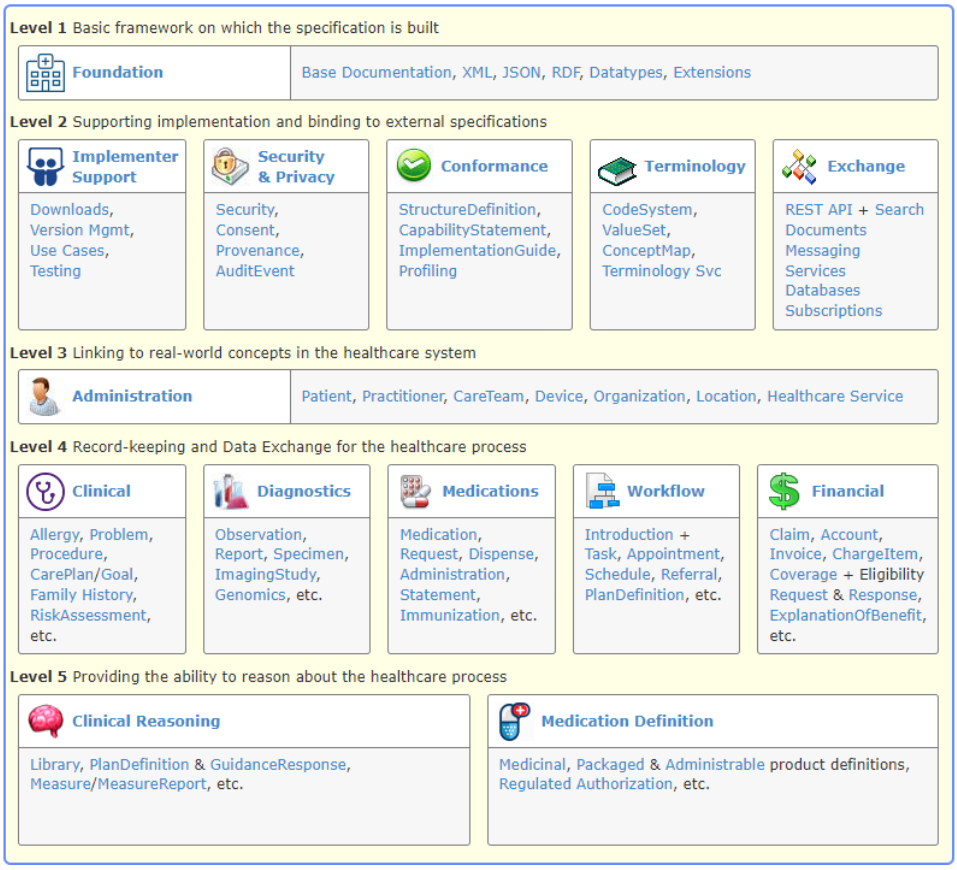


Figura 2-1. Niveles Módulos FHIR.

- **Level 1**: Marco de referencia sobre el que se ha construido la especificación. Proporciona los recursos esenciales como la documentación, formatos de datos (XML, JSON, etc.), los tipos de datos y las extensiones necesarias para adaptar los recursos a las necesidades específicas.

- **Level 2**: Soporte para la implementación y la vinculación a especificaciones externas. Proporciona herramientas y recursos para facilitar su implementación práctica.
- **Level 3**: Relación de conceptos del mundo real con los del sistema sanitario. Este nivel abarca recursos básicos en el sistema sanitario como pueden ser los pacientes, profesionales u organizaciones, entre otros.
- **Level 4**: Objetos involucrados en el registro y el intercambio de datos en el proceso sanitario. Este nivel proporciona información más compleja, como pueden ser módulos relacionados con procesos clínicos, diagnósticos, medicación, flujos de trabajo o finanzas.
- **Level 5**: Recursos para el razonamiento clínico. Proporciona recursos para la comprensión de los procesos clínicos e información detallada sobre medicamentos.

FHIR ha evolucionado a lo largo del tiempo, con distintas versiones que se han ido publicando para ir mejorando y expandiendo sus capacidades. Entre todas las versiones destacan dos, que además han sido las empleadas en este proyecto:

- **FHIR R4** [6]: Fue lanzada en 2018 y es la versión estable más usada en la actualidad. Entre sus características destaca el soporte de documentación clínica y la gestión de consentimientos.
- **FHIR R5** [7]: Fue lanzada en 2023 y progresivamente los sistemas están migrando a esta versión. Mejora y amplía la versión anterior con nuevas funcionalidades y optimización de las existentes.

2.1.2.1 Recursos FHIR

FHIR trabaja con lo que se denominan recursos [8]. Los recursos son componentes básicos que representan tipos de datos y conceptos dentro del ámbito sanitario. En tiempo de ejecución, se crean instancias de recursos que son las que modelan las entidades reales de un sistema. En base a la especificación, una instancia de un recurso es una entidad que posee las siguientes características:

1. Tiene una identidad conocida por la que se puede localizar.
2. Se identifica como uno de los tipos de recursos definidos dentro del estándar.
3. Contiene un conjunto de elementos estructurados tal como se describe en la definición del tipo de recurso.
4. Tiene una versión identificada que varía si cambia el contenido del recurso.

En la Figura 2-2 podemos observar el contenido del recurso básico "*Resource*" junto con la descripción de sus componentes.

Name	Flags	Card.	Type	Description & Constraints
Resource	«A» N		Base	Base Resource Elements defined in Ancestors:
id	Σ	0..1	id	Logical id of this artifact
meta	Σ	0..1	Meta	Metadata about the resource
implicitRules	?! Σ	0..1	uri	A set of rules under which this content was created
language		0..1	code	Language of the resource content Binding: All Languages (Required) **Additional Bindings** Purpose Common Languages Starter Set

Figura 2-2. Recurso básico FHIR.

Además de estos campos básicos, cada recurso tendrá unos componentes específicos de la información que almacenen. Por ejemplo, tenemos, los recursos "*Organization*" y "*Practitioner*" que, contienen respectivamente, información sobre organizaciones y personal sanitario.

Dentro de la especificación, existe gran variedad de recursos ya definidos y otros que se están todavía desarrollando. Estos recursos tienen lo que se denomina "Grados de madurez" [9] que van desde "0", cuando se acaba de incorporar un nuevo recurso, hasta "N" cuando el recurso ya no va a sufrir más cambios y se estandariza.

Los campos que conforman un recurso FHIR pueden ser de alguna de las siguientes categorías que define el estándar [10]:

- **Primitivos o simples**: Son elementos individuales con único valor y no tienen elementos hijos, como pueden ser *string*, *boolean* o *integer*.
- **De propósito general o complejos**: Campos que pueden contener múltiples valores o elementos hijos, formando estructuras más complejas, como pueden ser *Period* o *Address*.
- **Metadatos**: Campos que proporcionan información adicional sobre los datos o cómo deben usarse. Ejemplos son *ContactDetail* o *DataRequirement*.
- **De propósito especial**: Campos con funciones específicas dentro del estándar como pueden ser la referencia (*Reference*) o extensión (*Extension*) de otros recursos.

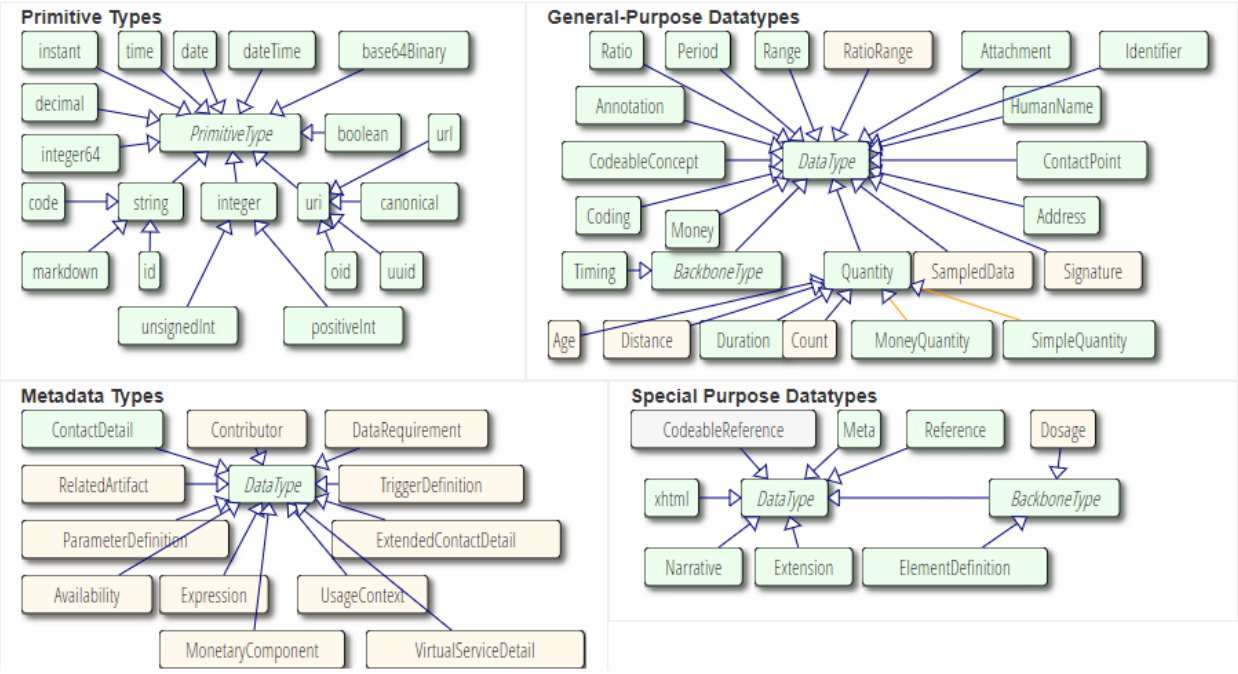


Figura 2-3. Tipos de datos en FHIR.

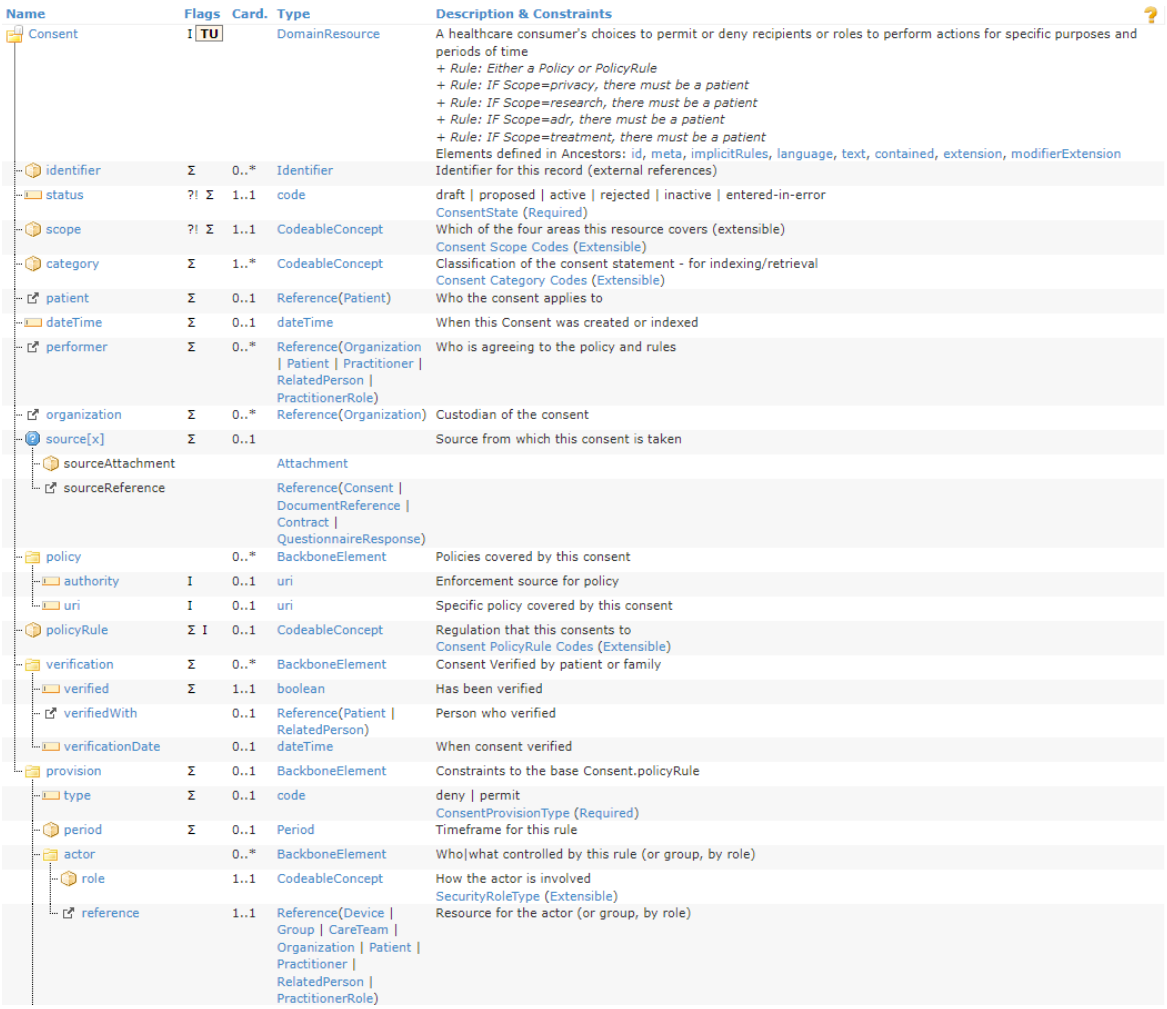
En este proyecto, de todos los recursos existentes, se han utilizado en mayor medida los recursos: ***Consent***, ***Patient***, ***Practitioner***, ***Organization*** y ***Bundle***. A continuación, se presentan cada uno de ellos:

A. Recurso Consent

Este recurso es el pilar fundamental de este proyecto, ya que es el más importante para la gestión de consentimientos en el ámbito sanitario y analizando sus campos podremos detectar posibles conflictos con otros consentimientos. Se trata de un registro de las elecciones de un consumidor de asistencia

sanitaria o de las elecciones realizadas en su nombre por un tercero, que permite o deniega al receptor/es identificado o rol/es de receptor realizar una o más acciones dentro de un contexto de política determinado, para propósitos y periodos de tiempo específicos [11]. El recurso contiene información como el alcance del consentimiento, el periodo de validez, el propósito o los actores involucrados (a quien aplica el consentimiento, por ejemplo, un profesional médico o una organización).

Este recurso varía en su contenido dependiendo de la versión de FHIR (R4 o R5) que estemos utilizando. R5 mejora algunos aspectos de R4, como la posibilidad de especificar con más detalle las acciones permitidas o prohibidas o el uso de codificaciones más avanzadas.



Name	Flags	Card.	Type	Description & Constraints
Consent	I TU		DomainResource	A healthcare consumer's choices to permit or deny recipients or roles to perform actions for specific purposes and periods of time *+ Rule: Either a Policy or PolicyRule* *+ Rule: IF Scope=privacy, there must be a patient* *+ Rule: IF Scope=research, there must be a patient* *+ Rule: IF Scope=adr, there must be a patient* *+ Rule: IF Scope=treatment, there must be a patient* Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	Identifier for this record (external references)
status	?! Σ	1..1	code	draft | proposed | active | rejected | inactive | entered-in-error ConsentState (Required)
scope	?! Σ	1..1	CodeableConcept	Which of the four areas this resource covers (extensible) Consent Scope Codes (Extensible)
category	Σ	1..*	CodeableConcept	Classification of the consent statement - for indexing/retrieval Consent Category Codes (Extensible)
patient	Σ	0..1	Reference(Patient)	Who the consent applies to
dateTime	Σ	0..1	dateTime	When this Consent was created or indexed
performer	Σ	0..*	Reference(Organization | Patient | Practitioner | RelatedPerson | PractitionerRole)	Who is agreeing to the policy and rules
organization	Σ	0..*	Reference(Organization)	Custodian of the consent
source[x]	Σ	0..1		Source from which this consent is taken
sourceAttachment			Attachment	
sourceReference			Reference(Consent | DocumentReference | Contract | QuestionnaireResponse)	
policy		0..*	BackboneElement	Policies covered by this consent
authority	I	0..1	uri	Enforcement source for policy
uri	I	0..1	uri	Specific policy covered by this consent
policyRule	Σ I	0..1	CodeableConcept	Regulation that this consents to Consent PolicyRule Codes (Extensible)
verification	Σ	0..*	BackboneElement	Consent Verified by patient or family
verified	Σ	1..1	boolean	Has been verified
verifiedWith		0..1	Reference(Patient | RelatedPerson)	Person who verified
verificationDate		0..1	dateTime	When consent verified
provision	Σ	0..1	BackboneElement	Constraints to the base Consent.policyRule
type	Σ	0..1	code	deny | permit ConsentProvisionType (Required)
period	Σ	0..1	Period	Timeframe for this rule
actor		0..*	BackboneElement	Who|what controlled by this rule (or group, by role)
role		1..1	CodeableConcept	How the actor is involved SecurityRoleType (Extensible)
reference		1..1	Reference(Device | Group | CareTeam | Organization | Patient | Practitioner | RelatedPerson | PractitionerRole)	Resource for the actor (or group, by role)

Figura 2-4. Recurso *Consent* FHIR R4 1.

action	Σ	0..*	CodeableConcept	Actions controlled by this rule Consent Action Codes (Example)
securityLabel	Σ	0..*	Coding	Security Labels that define affected resources SecurityLabels (Extensible)
purpose	Σ	0..*	Coding	Context of activities covered by this rule V3 Value SetPurposeOfUse (Extensible)
class	Σ	0..*	Coding	e.g. Resource Type, Profile, CDA, etc. Consent Content Class (Extensible)
code	Σ	0..*	CodeableConcept	e.g. LOINC or SNOMED CT code, etc. in the content Consent Content Codes (Example)
dataPeriod	Σ	0..1	Period	Timeframe for data controlled by this rule
data	Σ	0..*	BackboneElement	Data controlled by this rule
meaning	Σ	1..1	code	instance | related | dependents | authoredby ConsentDataMeaning (Required)
reference	Σ	1..1	Reference(Any)	The actual data reference
provision		0..*	see provision	Nested Exception Rules

Documentation for this format

Figura 2-5. Recurso *Consent* FHIR R4 2.

Name	Flags	Card.	Type	Description & Constraints
Consent	TU		DomainResource	A healthcare consumer's or third party's choices to permit or deny recipients or roles to perform actions for specific purposes and periods of time Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	Identifier for this record (external references)
status	?! Σ	1..1	code	draft | active | inactive | not-done | entered-in-error | unknown Binding: Consent State (Required)
category	Σ	0..*	CodeableConcept	Classification of the consent statement - for indexing/retrieval Binding: Consent Category Codes (Example)
subject	Σ	0..1	Reference(Patient | Practitioner | Group)	Who the consent applies to
date	Σ	0..1	date	Fully executed date of the consent
period	Σ	0..1	Period	Effective period for this Consent
grantor	Σ	0..*	Reference(CareTeam | HealthcareService | Organization | Patient | Practitioner | RelatedPerson | PractitionerRole)	Who is granting rights according to the policy and rules
grantee	Σ	0..*	Reference(CareTeam | HealthcareService | Organization | Patient | Practitioner | RelatedPerson | PractitionerRole)	Who is agreeing to the policy and rules
manager		0..*	Reference(HealthcareService | Organization | Patient | Practitioner)	Consent workflow management
controller		0..*	Reference(HealthcareService | Organization | Patient | Practitioner)	Consent Enforcer
sourceAttachment		0..*	Attachment	Source from which this consent is taken
sourceReference		0..*	Reference(Consent | DocumentReference | Contract | QuestionnaireResponse)	Source from which this consent is taken
regulatoryBasis		0..*	CodeableConcept	Regulations establishing base Consent Binding: Consent PolicyRule Codes (Example)
policyBasis		0..1	BackboneElement	Computable version of the backing policy
reference		0..1	Reference(Any)	Reference backing policy resource
url		0..1	url	URL to a computable backing policy
policyText		0..*	Reference(DocumentReference)	Human Readable Policy
verification	Σ	0..*	BackboneElement	Consent Verified by patient or family
verified	Σ	1..1	boolean	Has been verified
verificationType		0..1	CodeableConcept	Business case of verification Binding: Consent Vefication Codes (Example)
verifiedBy		0..1	Reference(Organization | Practitioner | PractitionerRole)	Person conducting verification
verifiedWith		0..1	Reference(Patient | RelatedPerson)	Person who verified
verificationDate		0..*	dateTime	When consent verified

Figura 2-6. Recurso *Consent* FHIR R5 1.

decision	?! Σ	0..1	code	deny | permit Binding: Consent Provision Type (Required)
provision	Σ	0..*	BackboneElement	Constraints to the base Consent.policyRule/Consent.policy
period	Σ	0..1	Period	Timeframe for this provision
actor		0..*	BackboneElement	Who|what controlled by this provision (or group, by role)
role		0..1	CodeableConcept	How the actor is involved Binding: Participation Role Type (Extensible)
reference		0..1	Reference(Device | Group | CareTeam | Organization | Patient | Practitioner | RelatedPerson | PractitionerRole)	Resource for the actor (or group, by role)
action	Σ	0..*	CodeableConcept	Actions controlled by this provision Binding: Consent Action Codes (Example)
securityLabel	Σ	0..*	Coding	Security Labels that define affected resources Binding: Example set of Security Labels (Example)
purpose	Σ	0..*	Coding	Context of activities covered by this provision Binding: PurposeOfUse (Extensible)
documentType	Σ	0..*	Coding	e.g. Resource Type, Profile, CDA, etc Binding: Consent Content Class (Preferred)
resourceType	Σ	0..*	Coding	e.g. Resource Type, Profile, etc Binding: Resource Types (Extensible)
code	Σ	0..*	CodeableConcept	e.g. LOINC or SNOMED CT code, etc. in the content Binding: Consent Content Codes (Example)
dataPeriod	Σ	0..1	Period	Timeframe for data controlled by this provision
data	Σ	0..*	BackboneElement	Data controlled by this provision
meaning	Σ	1..1	code	instance | related | dependents | authoredby Binding: Consent Data Meaning (Required)
reference	Σ	1..1	Reference(Any)	The actual data reference
expression		0..1	Expression	A computable expression of the consent
provision		0..*	see provision	Nested Exception Provisions

Figura 2-7. Recurso *Consent* FHIR R5 2.

```
"resource": {
  "resourceType": "Consent",
  "id": "1412373",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2020-07-20T10:30:24.200+00:00",
    "source": "#26dxoOPnIViP1x2b"
  },
  "status": "active",
  "scope": {
    "coding": [ {
      "system": "http://terminology.hl7.org/CodeSystem/consentscope",
      "code": "patient-privacy"
    } ]
  },
  "category": [ {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "59284-0",
      "display": "Patient Consent"
    } ]
  } ],
  "patient": {
    "reference": "Patient/37"
  },
  "dateTime": "2016-06-16",
  "policyRule": {
    "coding": [ {
      "system": "http://terminology.hl7.org/CodeSystem/v3-ActCode",
      "code": "OPTOUT"
    } ]
  },
  "provision": {
    "type": "permit",
    "actor": [ {
      "role": {
        "coding": [ {
          "system": "http://http://cct.eng.it/FHIR/schema/PractitionerRole",
          "code": "AAS"
        } ]
      },
      "reference": {
        "reference": "Practitioner/1141"
      }
    } ],
    "action": [ {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/consentaction",
        "code": "access"
      } ]
    } ],
    "securityLabel": [ {
      "system": "http://terminology.hl7.org/CodeSystem/v3-Confidentiality",
      "code": "N"
    } ],
    "code": [ {
      "coding": [ {
        "system": "2.16.840.1.113883.6.1",
        "code": "59258-4",
        "display": "Verbale di pronto soccorso"
      } ]
    } ],
    "data": [ {
      "meaning": "related",
      "reference": {
        "reference": "DocumentReference/26a9df9b-b345-4e03-9b50-131df18c0c12"
      }
    } ]
  }
},
```


Figura 2-8. Ejemplo JSON *Consent* FHIR R4.

```
"resource": {
  "resourceType": "Consent",
  "id": "ReceiveSMSMessagesExample",
  "meta": {
    "versionId": "2",
    "lastUpdated": "2023-08-15T18:05:44.269+00:00",
    "source": "#d9GMFzVG4btOvG8H",
    "profile": [ "http://openhie.org/fhir/rwanda-hiv/StructureDefinition/receive-sms-messages" ]
  },
  "status": "active",
  "category": [ {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "89057-4",
      "display": "Permission to receive text messages"
    } ],
    "text": "Patient consent for SMS messages"
  } ],
  "subject": {
    "reference": "Patient/487a6866-0f9d-450e-9b0c-1615f21d175e"
  },
  "decision": "permit"
},
```

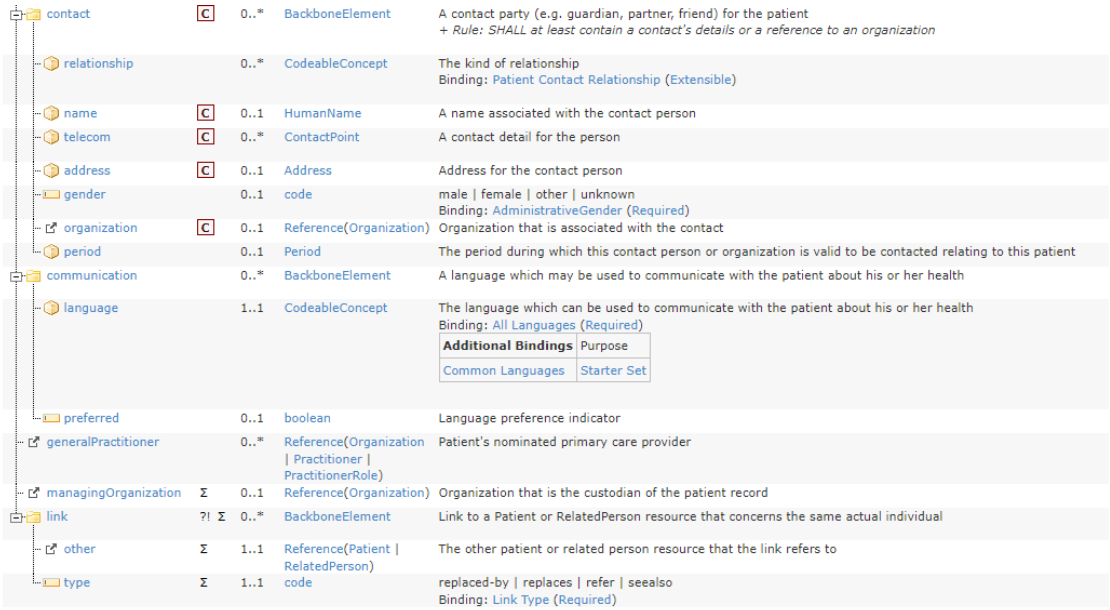

Figura 2-9. Ejemplo JSON *Consent* FHIR R5.

B. Recurso Patient

Este recurso almacena datos demográficos y otra información administrativa sobre un individuo o animal que recibe atención u otros servicios relacionados con la salud [12]. Entre estos servicios se encuentran: actividades curativas, atención psiquiátrica, servicios sociales, cuidados durante el embarazo, enfermería y vida asistida, servicios dietéticos, seguimientos de datos personales de salud y ejercicio y de servicios financieros. Este recurso alcanzó el nivel de madurez "N" en la versión R4.

Name	Flags	Card.	Type	Description & Constraints
Patient	N		DomainResource	Information about an individual or animal receiving health care services Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	An identifier for this patient
active	?! Σ	0..1	boolean	Whether this patient's record is in active use
name	Σ	0..*	HumanName	A name associated with the patient
telecom	Σ	0..*	ContactPoint	A contact detail for the individual
gender	Σ	0..1	code	male | female | other | unknown Binding: AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date of birth for the individual
deceased[x]	?! Σ	0..1		Indicates if the individual is deceased or not
deceasedBoolean			boolean	
deceasedDateTime			dateTime	
address	Σ	0..*	Address	An address for the individual
maritalStatus		0..1	CodeableConcept	Marital (civil) status of a patient Binding: Marital Status Codes (Extensible)
multipleBirth[x]		0..1		Whether patient is part of a multiple birth
multipleBirthBoolean			boolean	
multipleBirthInteger			integer	
photo		0..*	Attachment	Image of the patient

Figura 2-10. Recurso *Patient* FHIR 1.



contact	C	0..*	BackboneElement	A contact party (e.g. guardian, partner, friend) for the patient *+ Rule: SHALL at least contain a contact's details or a reference to an organization*
relationship		0..*	CodeableConcept	The kind of relationship Binding: Patient Contact Relationship (Extensible)
name	C	0..1	HumanName	A name associated with the contact person
telecom	C	0..*	ContactPoint	A contact detail for the person
address	C	0..1	Address	Address for the contact person
gender		0..1	code	male | female | other | unknown Binding: AdministrativeGender (Required)
organization	C	0..1	Reference(Organization)	Organization that is associated with the contact
period		0..1	Period	The period during which this contact person or organization is valid to be contacted relating to this patient
communication		0..*	BackboneElement	A language which may be used to communicate with the patient about his or her health
language		1..1	CodeableConcept	The language which can be used to communicate with the patient about his or her health Binding: All Languages (Required) **Additional Bindings** | Purpose Common Languages | Starter Set
preferred		0..1	boolean	Language preference indicator
generalPractitioner		0..*	Reference(Organization | Practitioner | PractitionerRole)	Patient's nominated primary care provider
managingOrganization	Σ	0..1	Reference(Organization)	Organization that is the custodian of the patient record
link	?! Σ	0..*	BackboneElement	Link to a Patient or RelatedPerson resource that concerns the same actual individual
other	Σ	1..1	Reference(Patient | RelatedPerson)	The other patient or related person resource that the link refers to
type	Σ	1..1	code	replaced-by | replaces | refer | seealso Binding: Link Type (Required)

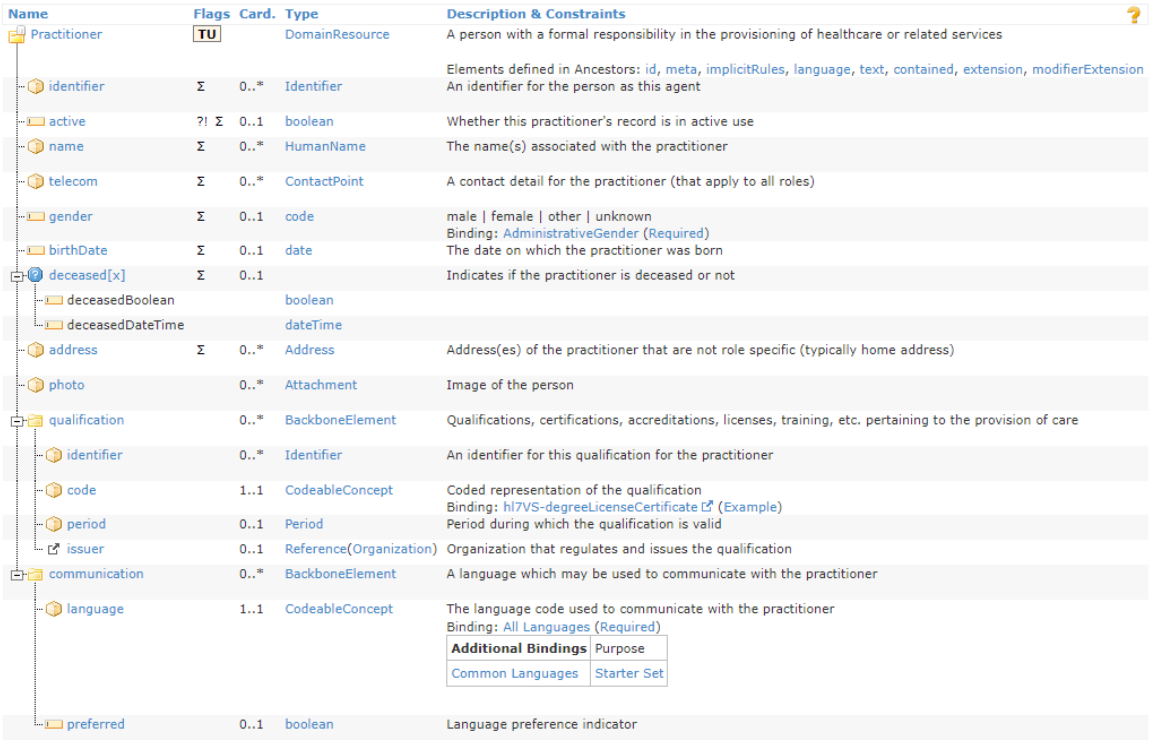
Figura 2-11. Recurso *Patient* FHIR 2.


```
"resource": {
  "resourceType": "Patient",
  "id": "1417",
  "meta": {
    "versionId": "2",
    "lastUpdated": "2020-09-28T04:13:13.236+00:00",
    "source": "#fw5Ox0CzvZK65gWe"
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">name:李<br/>ID:10501006620<br/>Gendar:female<br/>HospID:HHX<br/>Birthdate:1969-05-
8</div>"
  },
  "identifier": [ {
    "use": "usual",
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
        "code": "MR"
      } ]
    },
    "system": "urn:oid:1.2.36.146.595.217.0.1",
    "value": "10501006620",
    "period": {
      "start": "2020-09-26"
    },
    "assigner": {
      "display": "HHX"
    }
  } ],
  "active": true,
  "name": [ {
    "text": "李",
    "family": "李"
  } ],
  "gender": "female",
  "birthDate": "1969-05-20",
  "managingOrganization": {
    "display": "HHX"
  }
},
```


Figura 2-12. Ejemplo JSON *Patient* FHIR.

C. Recurso Practitioner

Este recurso representa a cualquier persona que esté involucrada directa o indirectamente en la prestación de servicios de atención médica o servicios relacionados [13]. Comprende desde médicos hasta recepcionistas o personal de IT que manejan los registros de los pacientes.



Name	Flags	Card.	Type	Description & Constraints
Practitioner	TU		DomainResource	A person with a formal responsibility in the provisioning of healthcare or related services Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	An identifier for the person as this agent
active	?! Σ	0..1	boolean	Whether this practitioner's record is in active use
name	Σ	0..*	HumanName	The name(s) associated with the practitioner
telecom	Σ	0..*	ContactPoint	A contact detail for the practitioner (that apply to all roles)
gender	Σ	0..1	code	male | female | other | unknown Binding: AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date on which the practitioner was born
deceased[x]	Σ	0..1		Indicates if the practitioner is deceased or not
deceasedBoolean			boolean	
deceasedDateTime			dateTime	
address	Σ	0..*	Address	Address(es) of the practitioner that are not role specific (typically home address)
photo		0..*	Attachment	Image of the person
qualification		0..*	BackboneElement	Qualifications, certifications, accreditations, licenses, training, etc. pertaining to the provision of care
identifier		0..*	Identifier	An identifier for this qualification for the practitioner
code		1..1	CodeableConcept	Coded representation of the qualification Binding: hl7VS-degreeLicenseCertificate (Example)
period		0..1	Period	Period during which the qualification is valid
issuer		0..1	Reference(Organization)	Organization that regulates and issues the qualification
communication		0..*	BackboneElement	A language which may be used to communicate with the practitioner
language		1..1	CodeableConcept	The language code used to communicate with the practitioner Binding: All Languages (Required) Additional Bindings: Purpose — Common Languages: Starter Set
preferred		0..1	boolean	Language preference indicator

Figura 2-13. Recurso *Practitioner* FHIR R5.

Las principales diferencias respecto a FHIR R4 son:

Changes from both R4 and R4B

Practitioner	
Practitioner.active	• Now marked as Modifier
Practitioner.deceased[x]	• Added Element
Practitioner.communication	• Type changed from CodeableConcept to BackboneElement • Remove Binding `http://hl7.org/fhir/ValueSet/languages` (preferred), max =`http://hl7.org/fhir/ValueSet/all-languages`
Practitioner.communication.language	• **Added Mandatory Element**
Practitioner.communication.preferred	• Added Element

Figura 2-14. Diferencias Recurso *Practitioner* FHIR R4-R5.

```
"resource": {
  "resourceType": "Practitioner",
  "id": "853",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2020-03-28T10:02:58.369+00:00",
    "source": "#jiNYX6WyyUDcinPZ"
  },
  "active": false,
  "telecom": [ {
    "system": "fax",
    "value": "4153799045",
    "use": "work"
  } ],
  "gender": "male",
  "address": [ {
    "use": "home",
    "line": [ "77 VAN NESS AVENUE", "SUITE 302" ],
    "city": "SAN FRANCISCO",
    "state": "CA",
    "postalCode": "94102"
  }, {
    "use": "work",
    "line": [ "77 VAN NESS AVENUE", "SUITE 302" ],
    "city": "SAN FRANCISCO",
    "state": "CA",
    "postalCode": "94102"
  } ]
},
```


Figura 2-15. Ejemplo JSON *Practitioner* FHIR.

D. Recurso Organization

Este recurso representa una agrupación de personas u organizaciones reconocida formal o informalmente, formada para lograr algún tipo de acción colectiva [14]. Incluye empresas, instituciones, corporaciones, departamentos, grupos comunitarios, grupos de profesionales sanitarios aseguradores, etc.

Name	Flags	Card.	Type	Description & Constraints
Organization	TU		DomainResource	A grouping of people or organizations with a common purpose *+ Rule: The organization SHALL at least have a name or an identifier, and possibly more than one* Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ C	0..*	Identifier	Identifies this organization across multiple systems
active	?! Σ	0..1	boolean	Whether the organization's record is still in active use
type	Σ	0..*	CodeableConcept	Kind of organization Binding: Organization Type (Example)
name	Σ C	0..1	string	Name used for the organization
alias		0..*	string	A list of alternate names that the organization is known as, or was known as in the past
description	Σ	0..1	markdown	Additional details about the Organization that could be displayed as further information to identify the Organization beyond its name
contact	C	0..*	ExtendedContactDetail	Official contact details for the Organization *+ Rule: The telecom of an organization can never be of use 'home'* *+ Rule: The address of an organization can never be of use 'home'*
partOf	Σ	0..1	Reference(Organization)	The organization of which this organization forms a part
endpoint		0..*	Reference(Endpoint)	Technical endpoints providing access to services operated for the organization
qualification		0..*	BackboneElement	Qualifications, certifications, accreditations, licenses, training, etc. pertaining to the provision of care
identifier		0..*	Identifier	An identifier for this qualification for the organization
code		1..1	CodeableConcept	Coded representation of the qualification Binding: Qualification (Example)
period		0..1	Period	Period during which the qualification is valid
issuer		0..1	Reference(Organization)	Organization that regulates and issues the qualification

Figura 2-16. Recurso *Organization* FHIR R5.

Las principales diferencias respecto a FHIR R4 son:

Changes from both R4 and R4B

Organization	
Organization.description	• Added Element
Organization.contact	• Type changed from BackboneElement to ExtendedContactDetail
Organization.qualification	• Added Element
Organization.qualification.identifier	• Added Element
Organization.qualification.code	• **Added Mandatory Element**
Organization.qualification.period	• Added Element
Organization.qualification.issuer	• Added Element
Organization.telecom	• Deleted (-> Use contact.telecom to provide context of use)
Organization.address	• Deleted (-> Use contact.address to provide context of use)
Organization.contact.purpose	• Deleted (-> Organization.contact.purpose)
Organization.contact.name	• Deleted (-> Organization.contact.name)
Organization.contact.telecom	• Deleted (-> Organization.contact.telecom)
Organization.contact.address	• Deleted (-> Organization.contact.address)

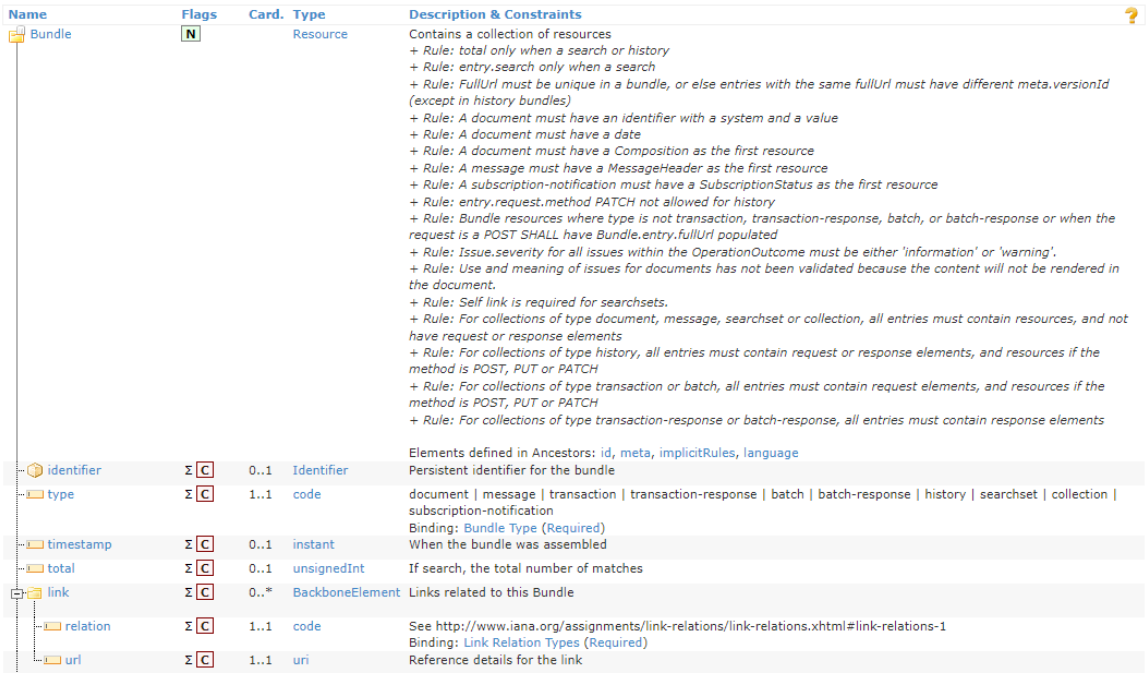
Figura 2-17. Diferencias Recurso *Organization* FHIR R4-R5.


```
"resource": {
  "resourceType": "Organization",
  "id": "543",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2020-03-05T06:58:41.752+00:00",
    "source": "#zExZROrsVp4OmjRI"
  },
  "type": [ {
    "coding": [ {
      "system": "Jackson",
      "code": "prov",
      "display": "Healthcare Provider"
    } ],
    "text": "Healthcare Provider"
  } ],
  "name": "Capital"
},
```


Figura 2-18. Ejemplo JSON *Organization* FHIR.

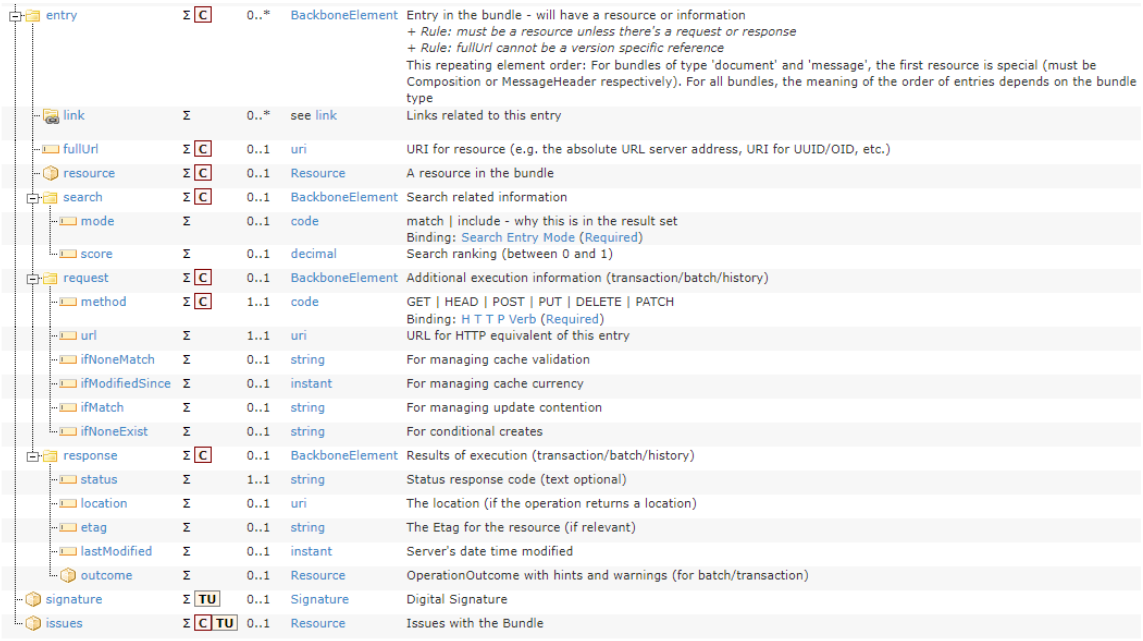
E. Recurso Bundle

Este recurso representa un contenedor para una colección de recursos [15]. Un *Bundle* puede contener recursos que representan una colección de información sanitaria de un paciente, como pueden ser su historial médico o una lista de consentimientos otorgados. Este recurso alcanzó el nivel de madurez "N" en la versión R4.



Name	Flags	Card.	Type	Description & Constraints
Bundle	N		Resource	Contains a collection of resources *+ Rule: total only when a search or history* *+ Rule: entry.search only when a search* *+ 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* *+ Rule: A subscription-notification must have a SubscriptionStatus as the first resource* *+ Rule: entry.request.method PATCH not allowed for history* *+ Rule: Bundle resources where type is not transaction, transaction-response, batch, or batch-response or when the request is a POST SHALL have Bundle.entry.fullUrl populated* *+ Rule: Issue.severity for all issues within the OperationOutcome must be either 'information' or 'warning'.* *+ Rule: Use and meaning of issues for documents has not been validated because the content will not be rendered in the document.* *+ Rule: Self link is required for searchsets.* *+ Rule: For collections of type document, message, searchset or collection, all entries must contain resources, and not have request or response elements* *+ Rule: For collections of type history, all entries must contain request or response elements, and resources if the method is POST, PUT or PATCH* *+ Rule: For collections of type transaction or batch, all entries must contain request elements, and resources if the method is POST, PUT or PATCH* *+ Rule: For collections of type transaction-response or batch-response, all entries must contain response elements* Elements defined in Ancestors: id, meta, implicitRules, language
identifier	Σ C	0..1	Identifier	Persistent identifier for the bundle
type	Σ C	1..1	code	document | message | transaction | transaction-response | batch | batch-response | history | searchset | collection | subscription-notification Binding: Bundle Type (Required)
timestamp	Σ C	0..1	instant	When the bundle was assembled
total	Σ C	0..1	unsignedInt	If search, the total number of matches
link	Σ C	0..*	BackboneElement	Links related to this Bundle
relation	Σ C	1..1	code	See http://www.iana.org/assignments/link-relations/link-relations.xhtml#link-relations-1 Binding: Link Relation Types (Required)
url	Σ C	1..1	uri	Reference details for the link

Figura 2-19. Recurso *Bundle* FHIR 1.



entry	Σ C	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 C	0..1	uri	URI for resource (e.g. the absolute URL server address, URI for UUID/OID, etc.)
resource	Σ C	0..1	Resource	A resource in the bundle
search	Σ C	0..1	BackboneElement	Search related information
mode	Σ	0..1	code	match | include - why this is in the result set Binding: Search Entry Mode (Required)
score	Σ	0..1	decimal	Search ranking (between 0 and 1)
request	Σ C	0..1	BackboneElement	Additional execution information (transaction/batch/history)
method	Σ C	1..1	code	GET | HEAD | POST | PUT | DELETE | PATCH Binding: H T T P Verb (Required)
url	Σ	1..1	uri	URL for HTTP equivalent of this entry
ifNoneMatch	Σ	0..1	string	For managing cache validation
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 C	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
issues	Σ C TU	0..1	Resource	Issues with the Bundle

Figura 2-20. Recurso *Bundle* FHIR 2.

```
{
  "resourceType": "Bundle",
  "id": "af774c4b-19dd-4ac5-b419-fa1d04f56f74",
  "meta": {
    "lastUpdated": "2024-08-28T19:32:52.946+00:00"
  },
  "type": "searchset",
  "link": [ {
    "relation": "self",
    "url": "https://hapi.fhir.org/baseR5/Bundle?_pretty=true"
  }, {
    "relation": "next",
    "url": "https://hapi.fhir.org/baseR5?_getpages=af774c4b-19dd-4ac5-b419-
fa1d04f56f74&_getpagesoffset=20&_count=20&_pretty=true&_bundletype=searchset"
  } ],
  "entry": [ {
    "fullUrl": "https://hapi.fhir.org/baseR5/Bundle/4e701cb4-46a2-49df-becd-686e51fb99c4",
    "resource": {
      "resourceType": "Bundle",
      "id": "4e701cb4-46a2-49df-becd-686e51fb99c4",
      "meta": {
        "versionId": "2",
        "lastUpdated": "2019-09-09T16:16:06.703+00:00",
        "source": "#399e1e44fefdade3"
      },
      "type": "collection",
      "total": 1,
      "link": [ {
        "relation": "self",
        "url": "http://hapi:8080/hapi-fhir-jpaserver/fhir/Bundle?_format=xml&_pretty=true"
      } ],
      "entry": [ {
        "extension": [ {
          "url": "https://kds.com/extension-notification-message",
          "valueString": "test message"
        } ],
        "fullUrl": "http://hapi:8080/hapi-fhir-jpaserver/fhir/Bundle/154",
        "resource": {
          "resourceType": "Bundle",
          "id": "154",
          "type": "collection",
          "total": 19,
          "link": [ {
            "relation": "self",
            "url": "https://example.com/base/MedicationRequest?"
          }, {
            "relation": "next",
            "url": "https://example.com/base/MedicationRequest?"
          } ],
          "entry": [ {
            "fullUrl": "https://example.com/base/MedicationRequest/3123",
            "resource": {
              "resourceType": "MedicationRequest",
              "id": "3123",
              "text": {
                "status": "generated",
                "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><p><b>Generated Narrative with Details</b></p><p><b>id</b>: 3123\r\n
</p><p><b>status</b>: unknown\r\n                    </p><p><b>intent</b>: order\r\n                    </p><p><b>medication</b>: \r\n
<a>Medication/example</a></p><p><b>subject</b>: \r\n                    <a>Patient/347</a> Some other test 8\r\n
</p></div>"
              },
              "status": "unknown",
              "intent": "order"
            }
          }, {
            "fullUrl": "https://example.com/base/Medication/example",
            "resource": {
              "resourceType": "Medication",
              "id": "example",
```


Figura 2-21. Ejemplo JSON Bundle FHIR.

2.2. Herramientas software, lenguajes y tecnologías empleadas

En este apartado se describirán las herramientas de software, lenguajes de programación y tecnologías que se han usado a lo largo del desarrollo del proyecto.

2.2.1. IntelliJ IDEA

Es un entorno de desarrollo integrado (IDE) ampliamente utilizado para la programación en diversos lenguajes, sobre todo Java y Kotlin [16]. Desarrollado por la compañía JetBrains s.r.o., este IDE ofrece una gran cantidad de herramientas y características, entre las que destacan la integración con sistemas de control de versiones y soporte para los frameworks más utilizados del mercado. Se encuentra disponible en los sistemas operativos Windows, macOS y Linux, y dispone de una versión gratuita (*Community Edition*) y una de pago con características avanzadas (*Ultimate Edition*).

En este IDE se ha desarrollado el grueso del proyecto utilizando el framework Spring con Java, y se han utilizado las siguientes extensiones:

- **Lombok**: Librería Java que simplifica la escritura de código al reducir la necesidad de código repetitivo, como *getters*, *setters* o constructores [17].
- **SonarLint**: Herramienta de análisis estático de código que permite detectar errores y malas prácticas en tiempo real [18].
- **Spring Boot**: No es una extensión como tal sino una herramienta de código abierto, que acelera y simplifica el desarrollo de microservicios y aplicaciones web Java con Spring Framework [19].

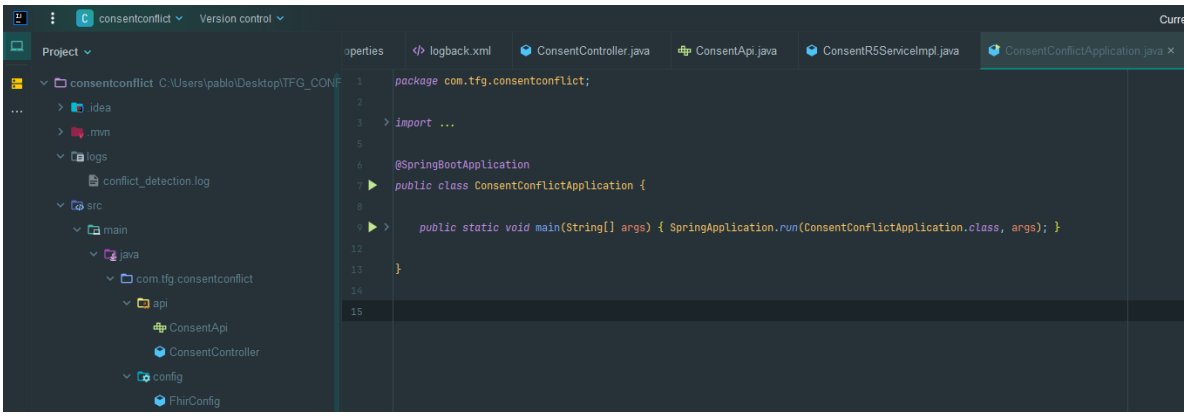


Figura 2-22. IntelliJ IDEA

Figura 2-23. Logo IntelliJ IDEA

2.2.2. Apache Maven

Es una herramienta de gestión y comprensión de proyectos software. Basado en el concepto de POM (*Project Object Model*), Maven puede gestionar la compilación, los informes y la documentación de un proyecto a partir de una pieza central de información [20]. Se ha utilizado en este proyecto para la gestión de dependencias y construcción del proyecto.

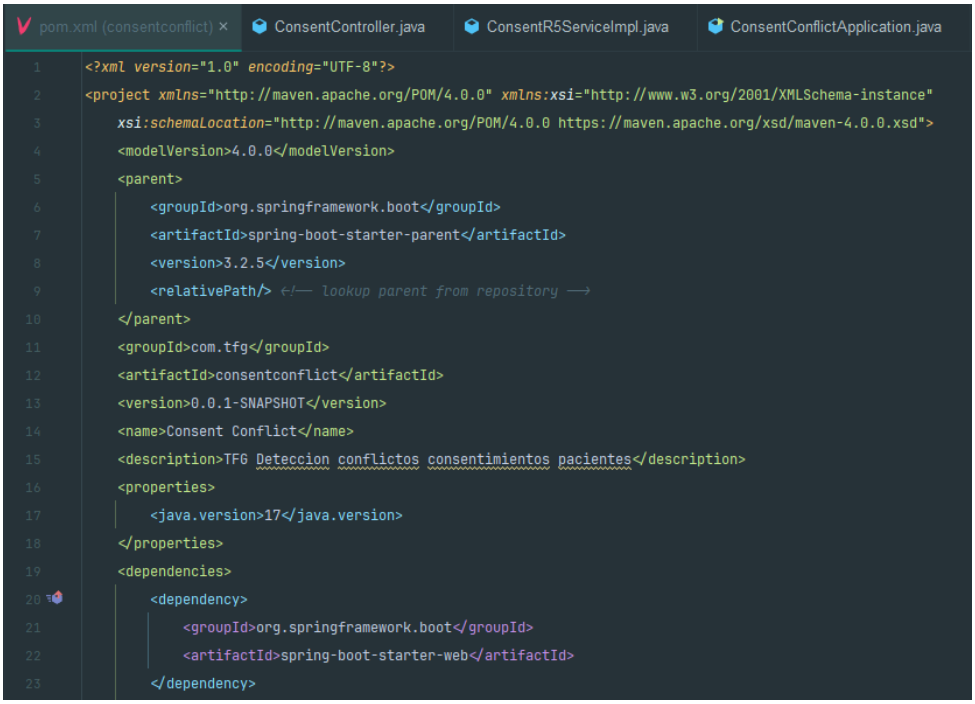


Figura 2-24. Ejemplo fichero POM.xml

Figura 2-25. Logo Apache Maven

2.2.3. Postman

Es una herramienta de colaboración y desarrollo que permite interactuar y probar el funcionamiento de servicios web y aplicaciones [21]. Postman nos permite el desarrollo, prueba y documentación de APIs, y está disponible en todos los sistemas operativos. Esta herramienta ofrece una GUI que facilita a los desarrolladores lanzar pruebas a sus aplicaciones a través de peticiones RESTful [22]. Entre sus principales características destacan:

- Envío de peticiones REST: GET, POST, PUT, DELETE y otros métodos HTTP a una API especificando encabezado, cuerpo y parámetros de la solicitud.
- Fácil configuración para diferentes entornos (desarrollo, preproducción, producción, …).
- Peticiones relacionadas se pueden agrupar en colecciones, para facilitar la organización y ejecución de pruebas.
- Ejecución de pruebas automatizadas para verificar el comportamiento de las APIs.
- Generación automatizada de documentación de las APIs a partir de las peticiones y las respuestas.

Esta herramienta se ha usado en este proyecto, para probar los endpoints del servicio de detección de conflictos, así como para insertar consentimientos en alguna de las bases de datos FHIR.

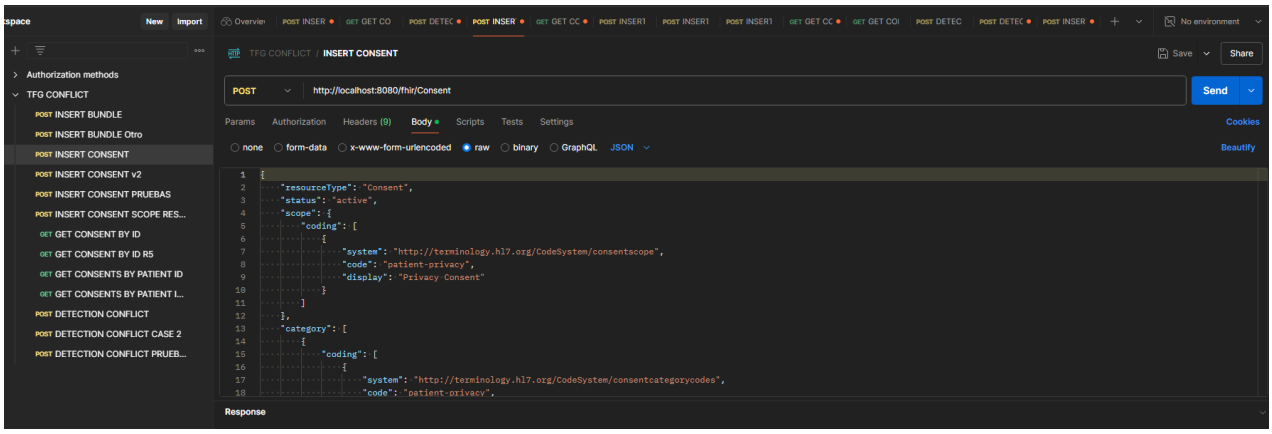


Figura 2-26. Postman GUI

Figura 2-27. Logo Postman

2.2.4. Swagger

Swagger es un conjunto de herramientas software de código abierto que facilita a los desarrolladores la construcción, el diseño, la documentación y el consumo de servicios web RESTful [23]. Esta desarrollado por Smartbear Software y se puede utilizar en todos los sistemas operativos al ser consumido como un servicio web. Permite a los desarrolladores describir la estructura y el funcionamiento de sus APIs siguiendo el formato estándar de OpenAPI.

En este proyecto se ha utilizado Swagger UI para documentar las APIs desarrolladas con el framework Spring. Como se encuentra integrado con este framework, se ha generado la documentación de los endpoints del proyecto para ser visualizada y probada directamente desde un navegador web. Esta interfaz puede resultar más intuitiva de utilizar para usuarios no familiarizados con entornos como Postman.

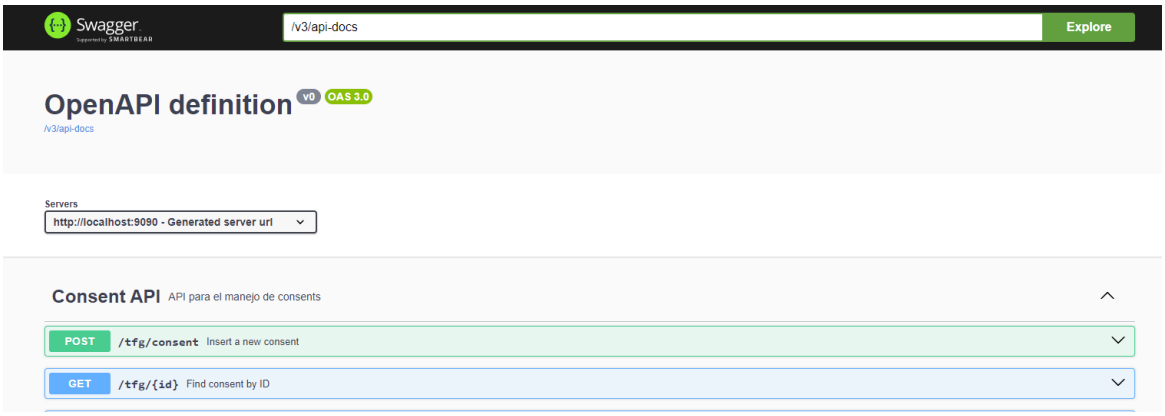


Figura 2-28. Swagger UI

Figura 2-29. Logo Swagger

2.2.5. Docker

Es una plataforma software que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones en lo que se conoce como contenedores [24]. Estos contenedores son unidades ligeras y portables que incluyen todo lo

necesario (código, runtime, dependencias, configuraciones, etc.) para que una aplicación funcione.

En este proyecto se ha utilizado la versión de escritorio Docker Desktop, para desplegar un contenedor con la imagen de una base de datos HAPI FHIR R4.

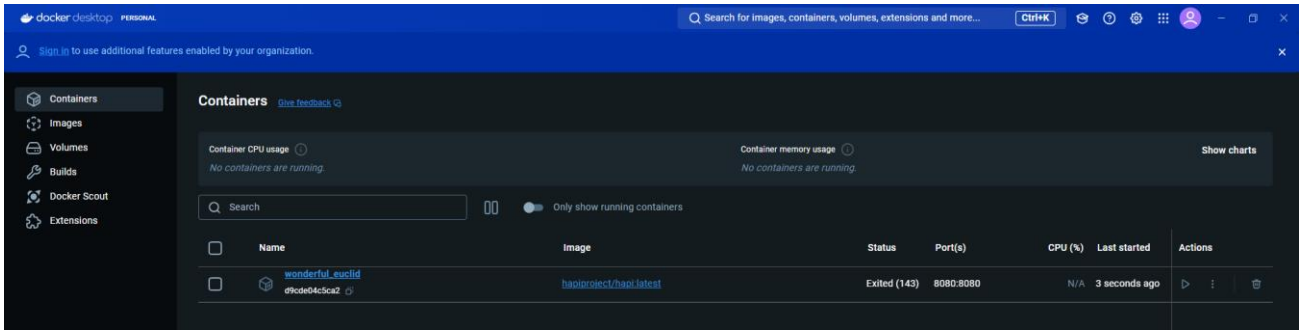


Figura 2-30. Docker Desktop GUI

Figura 2-31. Logo Docker

2.2.6. HAPI FHIR

Es una implementación completa del estándar HL7 FHIR para la interoperabilidad sanitaria en lenguaje Java [25]. Proporciona a los desarrolladores una gran variedad de herramientas y bibliotecas para crear, gestionar y validar recursos FHIR, facilitando la interoperabilidad entre sistemas sanitarios.

En este proyecto se ha desplegado una base de datos local HAPI FHIR R4 contra la que se han realizado pruebas de detección de conflictos para consentimientos de esta versión de FHIR. Asimismo, se han utilizado las librerías Java y Spring para crear los recursos necesarios en el sistema o enviándolos a través de peticiones de Postman. Para la versión de FHIR R5 se ha usado la base de datos pública que proporciona el equipo de HAPI FHIR [26].

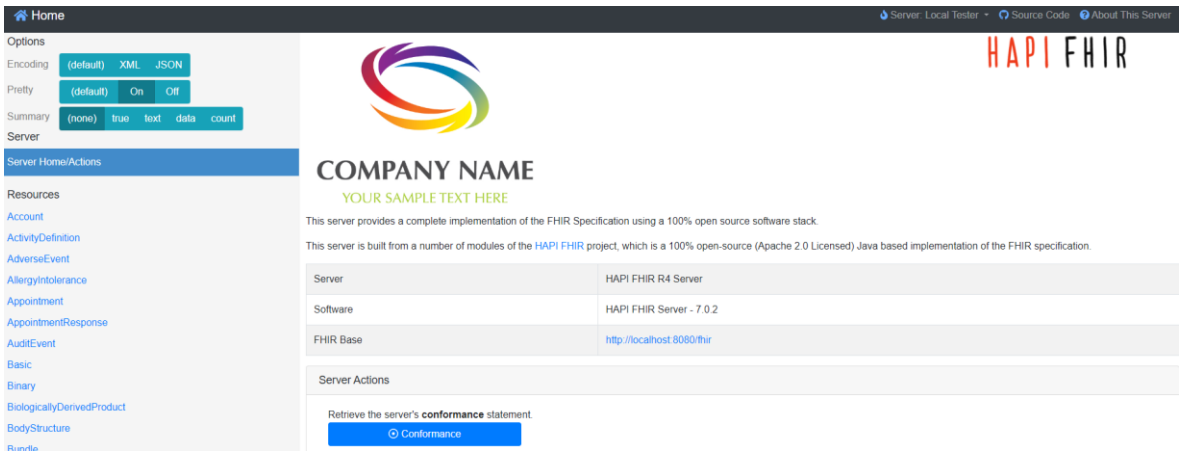


Figura 2-32. Base de datos local HAPI FHIR R4

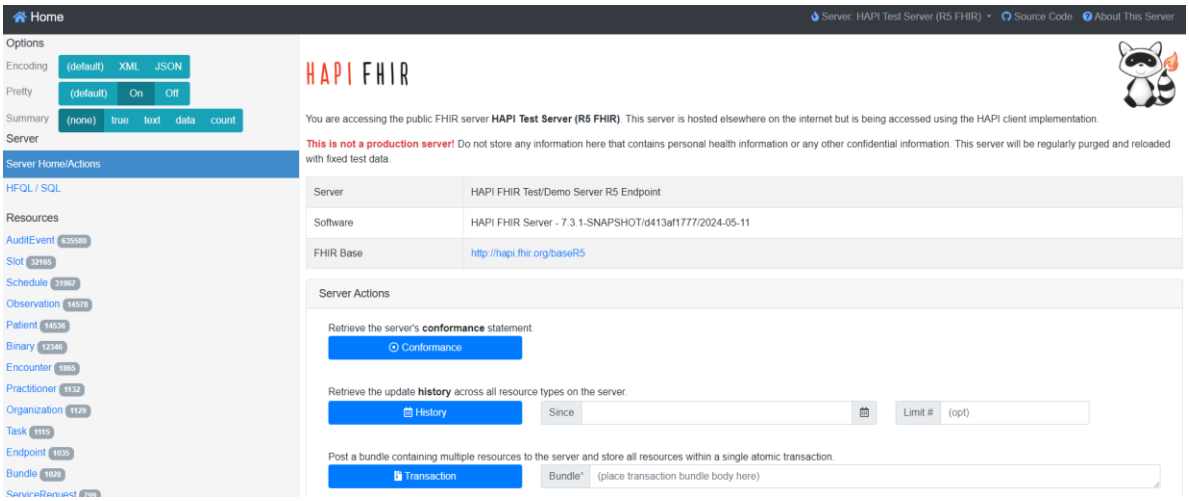


Figura 2-33. Base de datos pública HAPI FHIR R5

Figura 2-34. Logo HAPI FHIR

2.2.7. Batch Script

Este tipo de scripts propios del sistema operativo Windows permiten automatizar tareas. Al escribir secuencias de comandos en archivos de texto sin formato con extensión .bat o .cmd, se pueden ejecutar varios comandos automáticamente, evitando riesgos y ahorrando tiempo [27].

En este proyecto se ha utilizado un batch script para automatizar el proceso de arranque y carga de datos iniciales en el contenedor con la imagen de la base de datos HAPI FHIR R4.

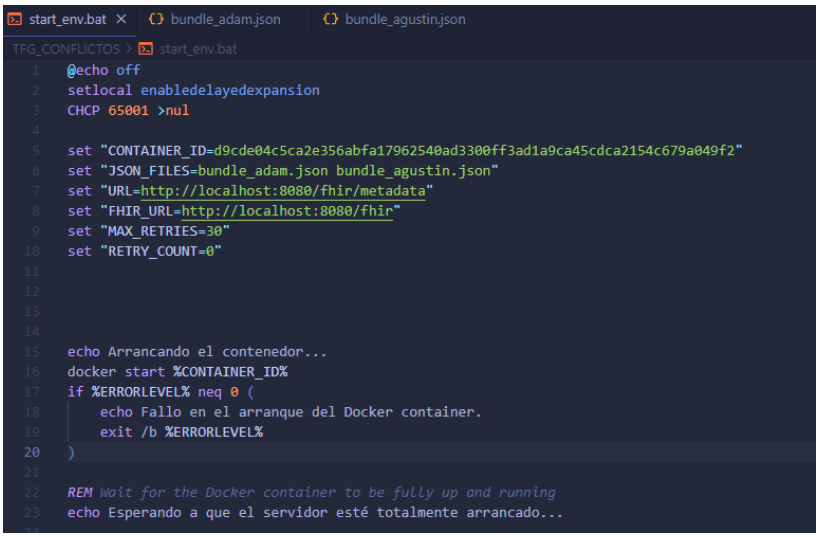


Figura 2-35. Ejemplo Batch Script

3 RESULTADOS

En este capítulo se van a presentar los resultados obtenidos durante la realización del proyecto. Se detallará primero cómo se ha diseñado el sistema de detección de conflictos en consentimientos de pacientes, después seguirá su implementación y por último veremos las pruebas que se han realizado.

3.1. Diseño del sistema

El sistema de detección de conflictos en consentimientos de pacientes se ha diseñado siguiendo un esquema general de funcionamiento, que tiene como objetivo la identificación eficiente de inconsistencias o conflictos entre los consentimientos nuevos y los ya existentes en los registros del paciente. Cabe destacar que, además de la funcionalidad principal de detección de conflictos, el sistema también es capaz de recibir y procesar peticiones para devolver consentimientos específicos (búsqueda por identificador), o todos los otorgados por un paciente (búsqueda por identificador del paciente).

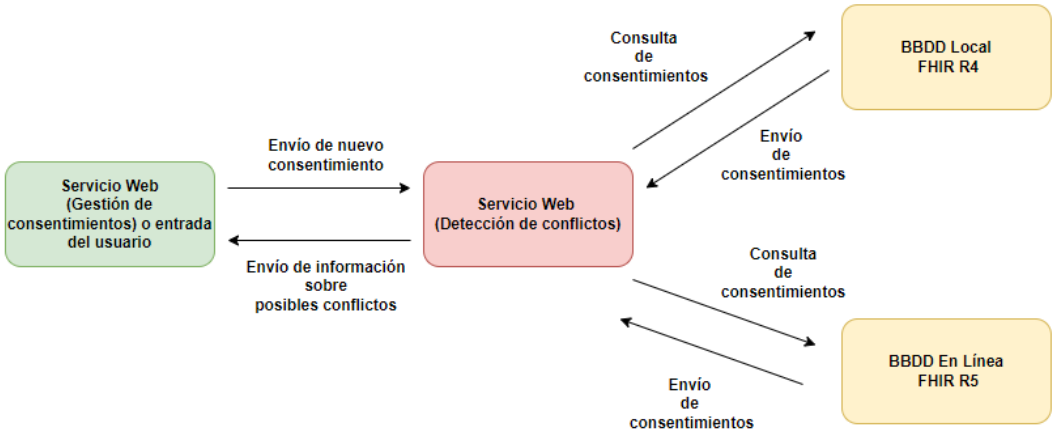


Figura 3-1. Funcionalidad general del sistema

El proceso comienza con la recepción de consentimientos a través de una API REST, que permite la entrada de nuevos consentimientos de pacientes en formato FHIR. Este consentimiento puede ser enviado bien desde un sistema de gestión de consentimientos o desde un usuario final (paciente) directamente. Estos consentimientos pueden ser de las versiones R4 o R5 del estándar FHIR, lo que garantiza que el sistema se puede adaptar a cualquier sistema sanitario actual.

Una vez recibido el consentimiento por el servicio web central, este realiza una consulta a la base de datos FHIR correspondiente (R4 o R5) para obtener los consentimientos previamente registrados otorgados por el mismo paciente. Dependiendo de la versión que estemos tratando el sistema lanzará la consulta a una base de datos local (FHIR R4) o a una base de datos pública (FHIR R5).

Una vez recibido el *Bundle* con todos los consentimientos previos del paciente, llegamos al eje central del proyecto, el **algoritmo de detección de conflictos**, que compara el nuevo consentimiento con los existentes. Esta comparación se basa en reglas que identifican inconsistencias o conflictos potenciales como pueden ser solapamiento de fechas o acciones contradictorias, entre otros ejemplos.

Finalmente, tras la ejecución del algoritmo, se genera una respuesta, enviada a través de la API REST, indicando si se han encontrado conflictos y proporcionando datos relevantes. Asimismo, se podrá avisar por correo electrónico al paciente para que revise el consentimiento en caso de que se hayan encontrado conflictos. Por otro lado, el equipo de mantenimiento del sistema dispondrá de un log para ver todas las peticiones juntos con los posibles conflictos.

3.1.1. Algoritmo de detección de conflictos

El algoritmo de detección de conflictos es el componente central del sistema y se encarga de analizar los consentimientos nuevos en comparación con los ya registrados en búsqueda de posibles inconsistencias que puedan generar conflictos. En función de la versión de FHIR con la que estemos trabajando el algoritmo variará en algunos puntos. A continuación, se presentan los flujos de actividad tanto para la versión R4 como R5.

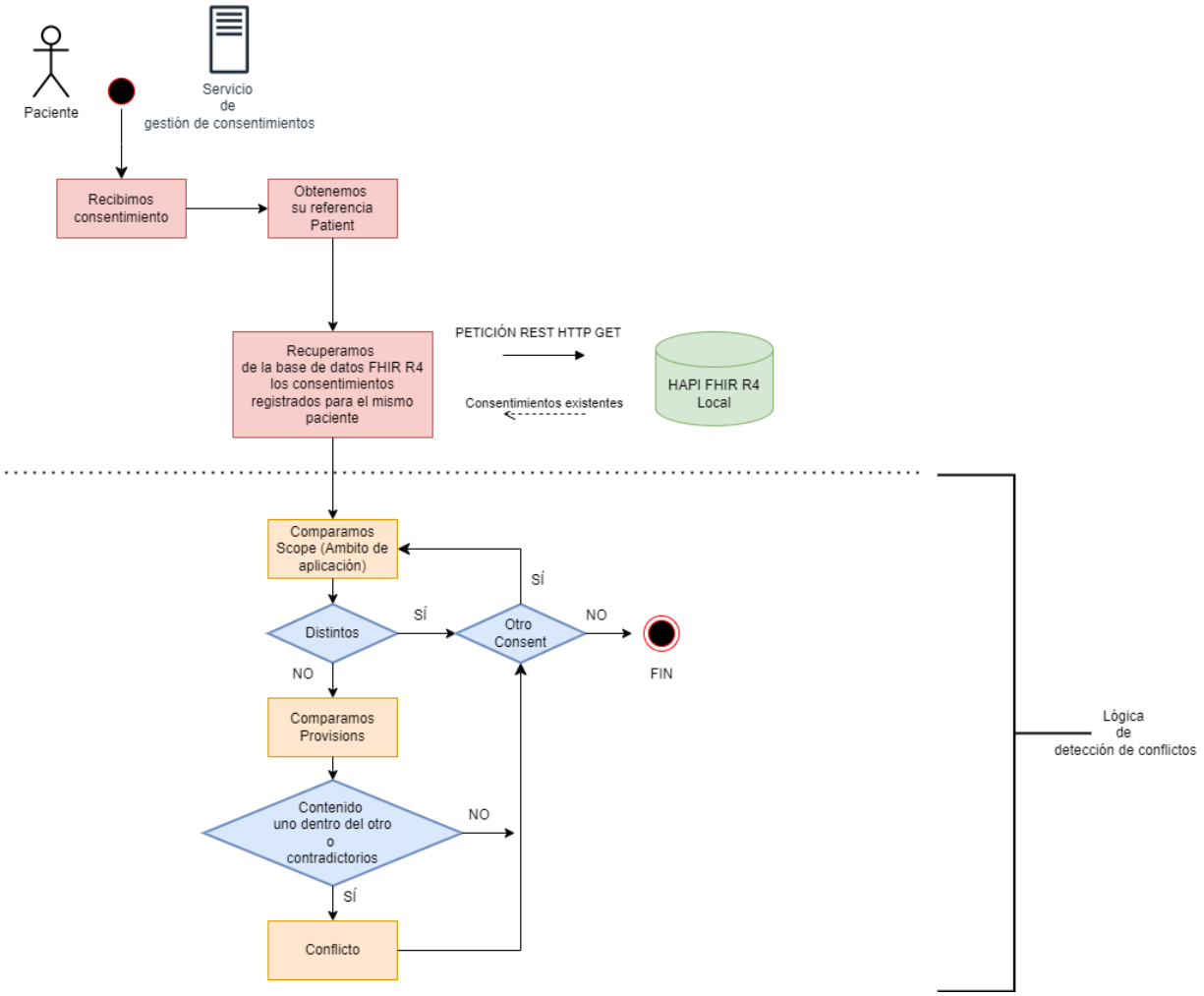


Figura 3-2. Flujo algoritmo de detección de conflictos FHIR R4

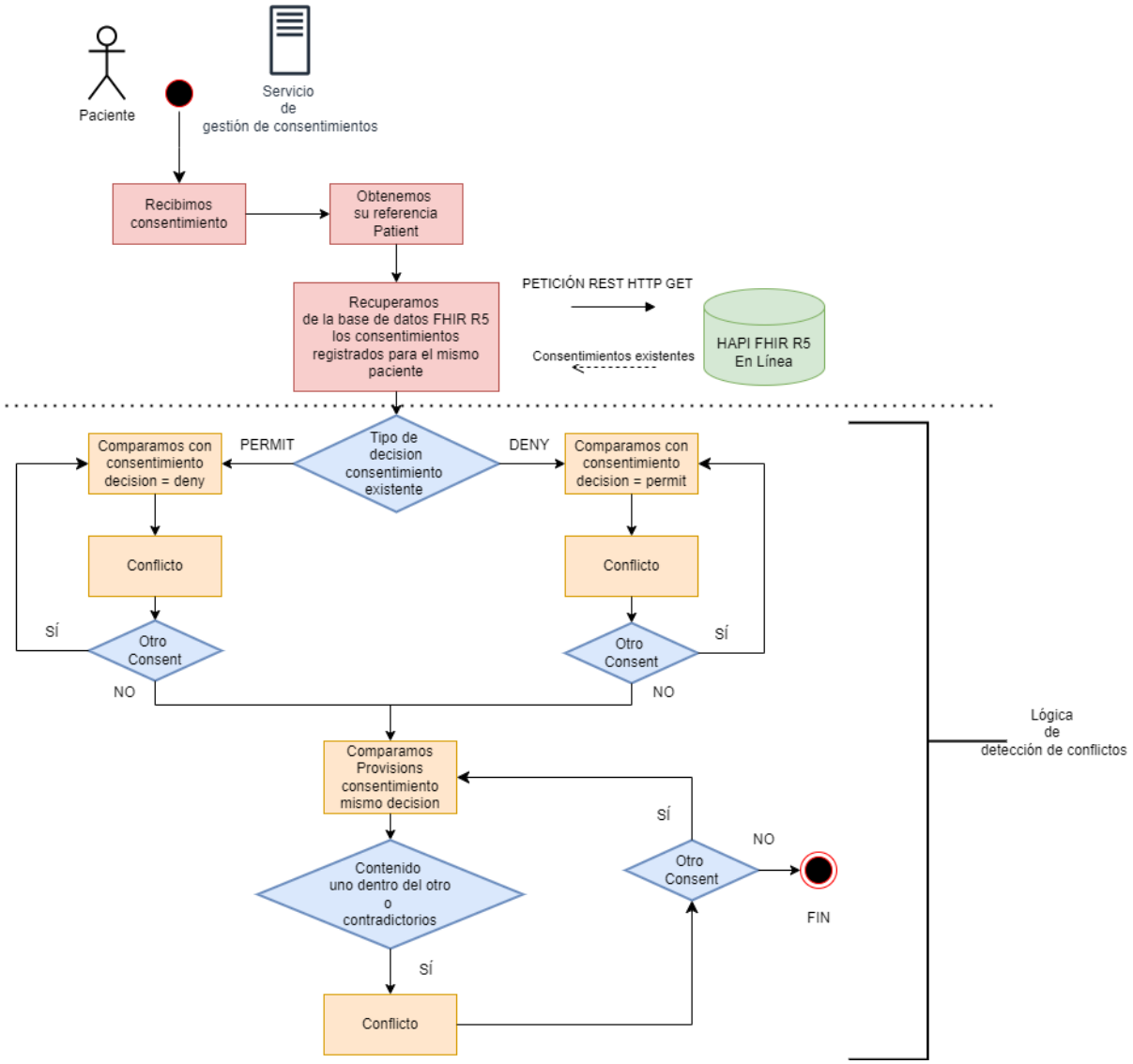


Figura 3-3. Flujo algoritmo de detección de conflictos FHIR R5

Ahora detallamos los aspectos relevantes del algoritmo de detección de conflictos:

1. **Análisis de campos relevantes**

   Una vez se han recibido los consentimientos existentes dentro de un recurso *Bundle*, el algoritmo analiza campos clave de cada consentimiento (variarán en función de la versión de FHIR con la que estemos trabajando), como pueden ser los ámbitos de aplicación, periodo de validez, recursos afectados, actores involucrados, si se permite o deniega la acción, entre otros.

2. **Reglas de detección de conflictos**

   El algoritmo aplica una serie de reglas que hemos definido para la identificación de posibles conflictos entre el nuevo consentimiento y los existentes. Entre estas reglas se incluyen:

   - **Ámbito de aplicación (*Scope*)**: Exclusivo de FHIR R4. Si tienen ámbitos de aplicación distintos, no se considerará que hay conflicto.
   - ***Decision***: Exclusivo de FHIR R5. Si tienen valores del campo decisión contrarios se considerará a priori que existe un conflicto y se evaluarán más reglas. Si son iguales, se

evaluarán directamente más reglas.

- **Solapamiento de fechas**: Si las fechas de ambos consentimientos se superponen se indicará que a priori existe un conflicto y se evaluarán más reglas.
- **Actores involucrados**: Si los actores (personal médico, organizaciones, etc.) a los que afecta el consentimiento son iguales se considerará que a priori existe un conflicto y se evaluarán más reglas.
- **Acción que realizar**: Si las acciones (acceder a datos, recogerlos, compartirlos, etc.) que recogen los consentimientos coinciden se considerará que a priori existe un conflicto y se evaluarán más reglas.
- **Recursos afectados**: Si los recursos (documentos clínicos) a los que afecta el consentimiento son iguales se considerará que a priori existe un conflicto y se evaluarán más reglas.

Existen más reglas que se han implementado en el microservicio, pero no tienen tanto peso en la consideración de conflictos.

3. **Valoración y clasificación de conflictos**

   Una vez se han aplicado todas las reglas, el algoritmo en función de las combinaciones de conflictos que haya ido registrando, dará una conclusión indicando si el nuevo consentimiento es compatible o no con el que se ha comparado. Por tanto, tendremos una conclusión por cada comparación que haya realizado el algoritmo.

4. **Generación de respuestas**

   El algoritmo genera una respuesta detallada indicando:

   - Existencia de conflictos
   - Lista de comparaciones que contiene:
     - Referencia del consentimiento con el que se compara
     - Mensajes por cada posible conflicto que se haya detectado
     - Conclusión

   Esta respuesta se envía por la API REST para que el sistema de gestión de conflictos o el paciente pueda revisar lo que ha ocurrido con el nuevo consentimiento.

5. **Notificación al paciente y registro para el sistema**

   En caso de que se detecten conflictos, se puede enviar un correo electrónico al paciente para informarle de que ha habido un problema con el nuevo consentimiento que ha generado. Además, los conflictos y errores que ocurren se registran en logs, para que puedan ser consultados en posibles auditorías o mantenimiento del sistema.

3.2. Implementación del Sistema

3.2.1. Bases de datos FHIR

Para el desarrollo de este proyecto, se han necesitado dos bases de datos que almacenen los datos de los pacientes. Se ha optado por las implementaciones que proporciona HAPI FHIR, ya preparadas para trabajar con los datos del estándar FHIR.

3.2.1.1. Base de datos FHIR R4

Se ha desplegado una base de datos FHIR R4 local, accesible a través de la dirección http://localhost:8080. Esta base de datos se ha creado utilizando la imagen "hapiproject/hapi:latest". Esta imagen contiene una instancia del servidor HAPI FHIR con la configuración por defecto para trabajar en la versión FHIR R4. Para facilitar su montaje y la carga de datos iniciales, se ha desarrollado un script en batch que automatiza tanto el arranque de la base de datos como la carga inicial de datos.

Este script como hemos mencionado no solo ejecuta el servidor, sino que también se encarga de la precarga de datos de prueba necesarios para el correcto funcionamiento del sistema. Estos datos son recursos *Bundle* que contienen:

- ***Patient***: Paciente que otorga el consentimiento.
- ***Organization***: Institución a la que puede afectar un consentimiento.
- ***Practitioner***: Profesional médico al que puede afectar un consentimiento.

Los *Bundle* contienen más recursos, pero los nombrados son los mínimos necesarios para que podamos crear en el sistema consentimientos asociados a recursos que ya existan. Si tratásemos de introducir un nuevo consentimiento que referencia a recursos inexistentes, el sistema devolvería un error indicando que no conoce alguno de los recursos a los que se ha hecho referencia dentro del consentimiento.

La configuración del despliegue de esta base de datos y la automatización mediante el batch script se detallan en el **Anexo 1**.

3.2.1.2. Base de datos FHIR R5

Para la versión de FHIR R5 hemos utilizado la base de datos pública que proporciona HAPI FHIR, disponible a través de la dirección http://hapi.fhir.org/baseR5. Esta base de datos es mantenida por toda la comunidad de HAPI FHIR y posee una gran cantidad de recursos listos para ser consumidos.

En este caso al existir ya los recursos mínimos necesarios para funcionar correctamente, solo se han tenido que crear nuevos consentimientos o consultar los ya existentes.

En ambos casos, se ha utilizado Postman para la ingestión de consentimientos a través de los endpoints que exponen las dos bases de datos a través de sus respectivas API REST. Estos endpoints también son accesibles a través de sus interfaces Swagger-UI [28], observables en las Figuras 3-4 y 3-5 respectivamente, pero por agilizar las operaciones se ha optado por la primera herramienta.

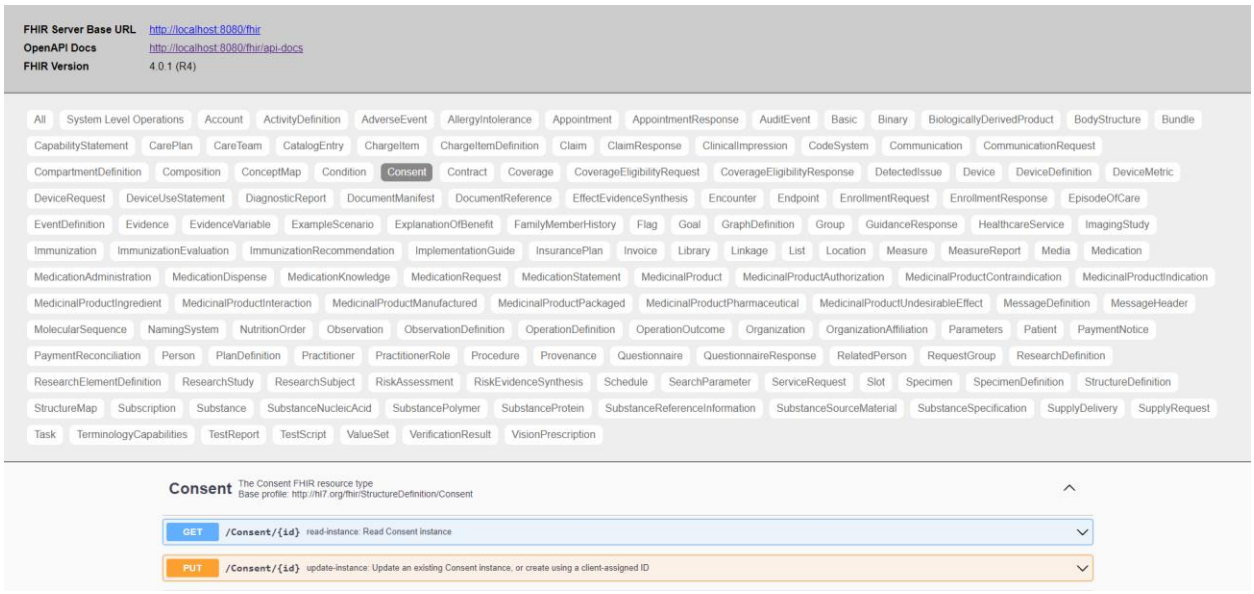


Figura 3-4. Swagger UI - FHIR R4

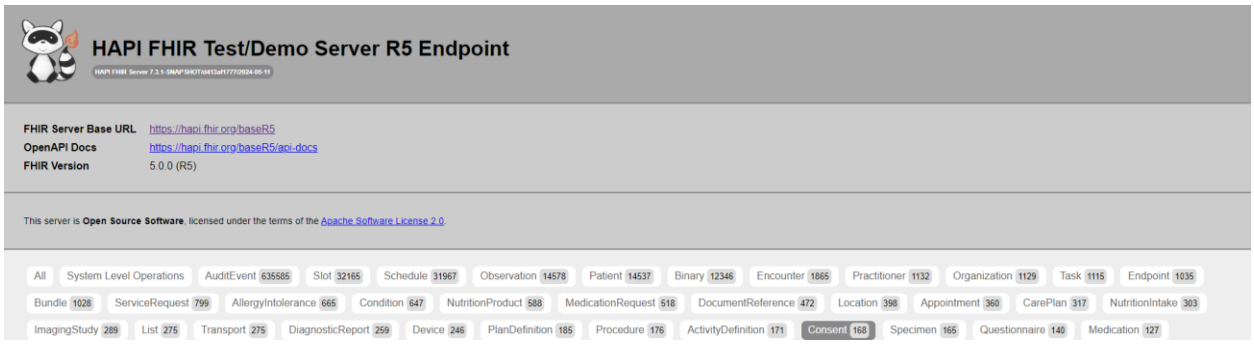


Figura 3-5. Swagger UI - FHIR R5

Dentro de Postman, se han realizado peticiones POST a los endpoints de ambas bases de datos para almacenar los consentimientos. Estos consentimientos viajan en el cuerpo de la petición en formato JSON como se puede ver en las Figuras 3-6 y 3-7.

```
"resourceType": "Consent",
"status": "active",
"scope": {
    "coding": [
        {
            "system": "http://terminology.hl7.org/CodeSystem/consentscope",
            "code": "patient-privacy",
            "display": "Privacy Consent"
        }
    ]
},
"category": [
    {
        "coding": [
            {
                "system": "http://terminology.hl7.org/CodeSystem/consentcategorycodes",
                "code": "patient-privacy",
                "display": "Privacy Consent"
            }
        ]
    }
],
"patient": {
    "reference": "Patient/1"
},
"dateTime": "2024-04-23T13:00:00Z",
"performer": [
    {
        "reference": "Patient/1"
    }
],
"organization": [
    {
        "reference": "Organization/2"
    }
],
"provision": {
    "type": "permit",
```


Figura 3-6. Extracto *Consent* FHIR R4 formato JSON


```
{
  "resourceType": "Consent",
  "status": "active",
  "category": [
    {
      "coding": [
        {
          "system": "http://loinc.org",
          "code": "89057-4",
          "display": "Patient Consent"
        }
      ]
    }
  ],
  "subject": {
    "reference": "Patient/86aa6c8c-f050-4602-8591-e77ee8e0e27c"
  },
  "date": "2024-01-01",
  "period": {
    "start": "2024-01-01",
    "end": "2024-06-30"
  },
  "grantor": [
    {
      "reference": "Practitioner/866"
    }
  ],
  "grantee": [
    {
      "reference": "Organization/80037"
    }
  ],
  "decision": "permit",
  "provision": [
    {
      "period": {
        "start": "2024-01-01",
        "end": "2024-06-30"
      }
```


Figura 3-7. Extracto *Consent* FHIR R5 formato JSON

Como hemos comentado anteriormente, se necesitan algunos recursos previos a la creación de un consentimiento. Entonces cuando lanzamos las peticiones desde Postman, podemos obtener las siguientes respuestas:

- **201 Created**: Se ha creado correctamente el consentimiento.

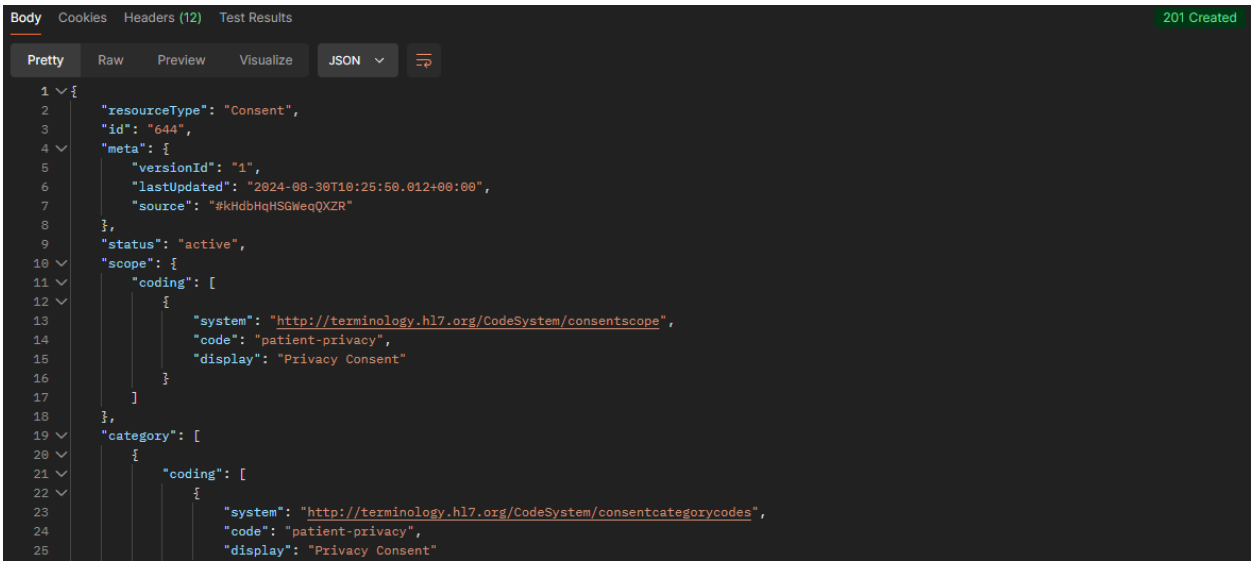


Figura 3-8. Respuesta correcta HTTP 201: Creación consentimiento.

- **422 Unprocessable Entity (WebDAV) (RFC 4918)** [29]: Para el servidor FHIR R4 local. El cuerpo de mensaje es sintácticamente correcto, pero no ha podido encontrar algunas de las referencias.

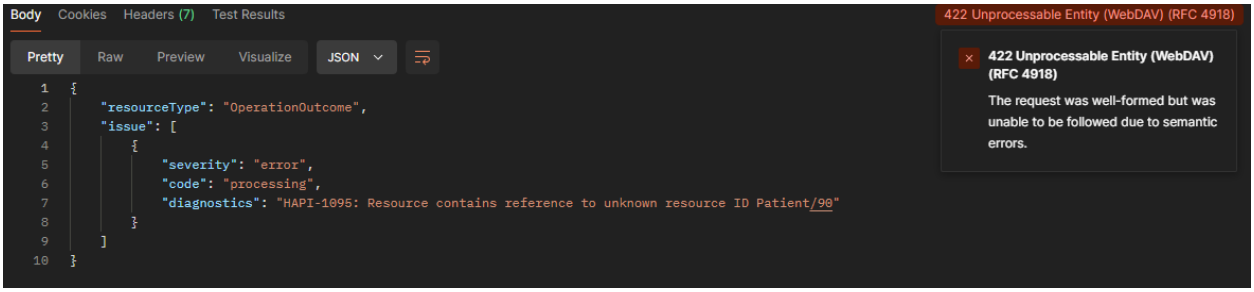


Figura 3-9. Respuesta de error HTTP 422: Referencia no encontrada FHIR R4.

- **400 Bad Request**: Para el servidor público FHIR R5. El cuerpo de mensaje es sintácticamente correcto, pero no ha podido encontrar algunas de las referencias.

Figura 3-10. Respuesta de error HTTP 400: Referencia no encontrada FHIR R5.

3.2.2. Microservicio Spring Boot

En este apartado detallaremos la implementación del núcleo del sistema de detección de conflictos en consentimientos de pacientes, un microservicio usando el framework Spring Boot Java. Este microservicio se ha diseñado para recibir, procesar y responder a solicitudes que afectan al análisis para la detección de posibles conflictos en los consentimientos.

3.2.2.1. Arquitectura del Proyecto

El microservicio que se ha desarrollado sigue la arquitectura de microservicios [30], donde cada componente funciona de manera autónoma y tiene una función específica. Esto permitirá que este microservicio pueda integrarse con facilidad en entornos ya funcionales que necesiten de su funcionalidad. La estructura del proyecto se puede ver en la Figura 3-11.

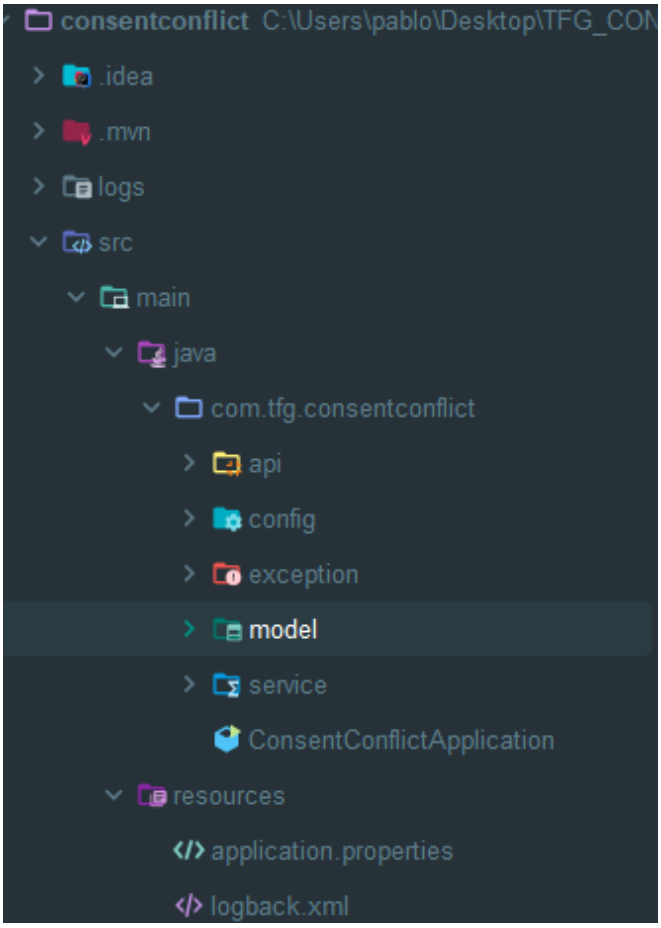


Figura 3-11. Estructura del proyecto.

Se ha seguido un modelo en capas para la estructuración de este proyecto. Este modelo de diseño de software se caracteriza por separar las distintas funcionalidades del sistema en distintas capas o niveles, donde cada capa se encarga de un conjunto de tareas específicas y la comunicación entre capas se realiza a través de interfaces bien definidas [31]. A continuación, se detalla cada una de las capas.

3.2.2.1.1. Capa de Presentación (Capa de Controllers)

Esta capa es la encargada de manejar las solicitudes HTTP que se reciban y en ella se definen los endpoints que el microservicio expondrá a través la API REST. Cada endpoint definido tiene un propósito específico como son la recepción de un nuevo consentimiento para detectar posibles conflictos o la consulta de consentimientos ya sea a través del identificador del consentimiento o el del paciente. El controlador será el encargado de desencadenar la lógica de negocio y de devolver las respuestas al sistema o paciente solicitante. Es decir, hace de intermediario entre la entidad que hace la solicitud y el núcleo del sistema encargado de la detección de

conflictos o la consulta de consentimientos.

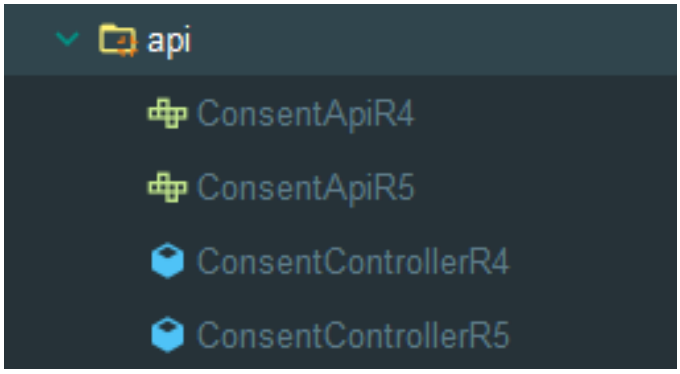


Figura 3-12. Capa de presentación del proyecto.

En esta capa nos encontramos dos interfaces, una para cada versión de FHIR:

- ***ConsentApiR4***

  Interfaz que define la API REST para las operaciones relacionadas con la versión de FHIR R4. También contiene la especificación OpenAPI para documentar los endpoints en Swagger.

- ***ConsentApiR5***

  Interfaz que define la API REST para las operaciones relacionadas con la versión de FHIR R5. También contiene la especificación OpenAPI para documentar los endpoints en Swagger.

Como ambas interfaces poseen la misma estructura, en la Figura 3-13 se detalla el contenido de una de ellas. En particular, se ha usado la anotación de Spring *@RequestMapping* para definir la ruta raíz a la que responderán las API REST, tanto para la versión R4 como R5. Además, se han utilizado las anotaciones *@GetMapping* y *@PostMapping* para especificar a qué métodos HTTP (GET y POST en este caso) y a qué ruta responderán los métodos que hemos declarado. Para las peticiones GET y POST se han usado dos anotaciones que recogen datos de la petición, estas son:

1. *@RequestBody*: para indicar a la función que el parámetro de entrada que tiene es el cuerpo de la petición.
2. *@PathVariable*: para indicar que el parámetro de entrada forma parte de la ruta de la petición.

```java
@Tag(name = "Consent API - FHIR R4", description = "API para el manejo de consents en la versión FHIR R4")  1 usage  1 implementation
@RequestMapping("/tfg/R4")
public interface ConsentApiR4 {

    @Operation(summary = "Buscar consent por ID", description = "Recupera un registro de consent por su ID en la versión FHIR R4.")  no usages  1 implementation
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Consent encontrado",
                    content = @Content(mediaType = MediaType.APPLICATION_JSON_VALUE,
                            schema = @Schema(implementation = String.class))),
            @ApiResponse(responseCode = "404", description = "Consent no encontrado")
    })
    @GetMapping(value = "/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    String findConsentById(
            @Parameter(description = "ID del consent", required = true)
            @PathVariable String id);

    @Operation(summary = "Buscar consents por ID de paciente", description = "Recupera todos los registros de consents asociados a un paciente especifico en la versión FHIR R4.")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Consents encontrados",
                    content = @Content(mediaType = MediaType.APPLICATION_JSON_VALUE,
                            schema = @Schema(implementation = String.class))),
            @ApiResponse(responseCode = "404", description = "No se encontraron consents")
    })
    @GetMapping(value = "/patient/{patientId}", produces = MediaType.APPLICATION_JSON_VALUE)
    String findConsentByPatientId(
            @Parameter(description = "ID del paciente", required = true)
            @PathVariable String patientId);

    @Operation(summary = "Insertar un nuevo consent", description = "Recibe un nuevo registro de consent y detecta posibles conflictos en la versión FHIR R4.")  no usages  1 implementat
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Conflictos detectados",
                    content = @Content(mediaType = MediaType.APPLICATION_JSON_VALUE,
                            schema = @Schema(implementation = ConflictDetail.class)))
    })
    @PostMapping(value = "", produces = MediaType.APPLICATION_JSON_VALUE)
    ResponseEntity<ConflictDetail> insertConsent(
            @Parameter(description = "Datos del consent en formato JSON", required = true)
            @RequestBody String data);
}
```

Figura 3-13. Interfaz *ConsentApiR4*.

Ambas interfaces declaran los mismos métodos:

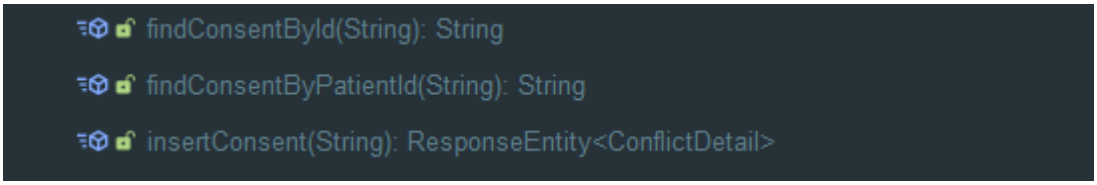


Figura 3-14. Funciones declaradas en las interfaces

En esta capa también tenemos los dos controladores que implementan las interfaces anteriormente mencionadas:

- ***ConsentControllerR4***

```java
@RestController  1 usage
public class ConsentControllerR4 implements ConsentApiR4{
    @Autowired  3 usages
    private ConsentR4Service consentR4Service;
    @Autowired  2 usages
    private IParser fhirParserR4;
    @Autowired  2 usages
    private MailService mailService;
    private static final Logger logger = LoggerFactory.getLogger(ConsentControllerR4.class);  no usages

    @Override  no usages
    public String findConsentById(String id) {
        return consentR4Service.getConsentById(id);
    }

    @Override  1 usage
    public String findConsentByPatientId(String patientId) {
        return consentR4Service.findConsentsByPatientId(patientId);
    }

    @Override  no usages
    public ResponseEntity<ConflictDetail> insertConsent(String data) {
        Consent consent = (Consent) fhirParserR4.parseResource(data);
        Bundle existingConsents = (Bundle) fhirParserR4.parseResource(findConsentByPatientId(consent.getPatient().getReference().split( regex: "/")[1]));
        ConflictDetail conflictDetail = consentR4Service.detectConflicts(consent, existingConsents);

        if(conflictDetail.isConflict()){
            mailService.sendEmail( to: "gestorpalovi@gmail.com",  subject: "CONFLICTOS R4",  body: "Se han detectado conflictos entre el nuevo consentimiento proporcionado
        }

        return ResponseEntity.ok(conflictDetail);
    }
}
```

Figura 3-15. Clase *ConsentControllerR4*.

- ***ConsentControllerR5***

```java
@RestController  1 usage
public class ConsentControllerR5 implements ConsentApiR5{
    @Autowired  3 usages
    private ConsentR5Service consentR5Service;
    @Autowired  2 usages
    private IParser fhirParserR5;
    @Autowired  2 usages
    private MailService mailService;
    private static final Logger logger = LoggerFactory.getLogger(ConsentControllerR5.class);  no usages

    @Override  no usages
    public String findConsentById(String id) {
        return consentR5Service.getConsentById(id);
    }

    @Override  1 usage
    public String findConsentByPatientId(String patientId) {
        return consentR5Service.findConsentsByPatientId(patientId);
    }

    @Override  no usages
    public ResponseEntity<ConflictDetail> insertConsent(String data) {
        Consent consent = (Consent) fhirParserR5.parseResource(data);
        Bundle existingConsents = (Bundle) fhirParserR5.parseResource(findConsentByPatientId(consent.getSubject().getReference().split( regex: "/")[1]));
        ConflictDetail conflictDetail = consentR5Service.detectConflicts(consent, existingConsents);

        if(conflictDetail.isConflict()){
            mailService.sendEmail( to: "gestorpalovi@gmail.com",  subject: "CONFLICTOS R5",  body: "Se han detectado conflictos entre el nuevo consentimiento proporcionado
        }

        return ResponseEntity.ok(conflictDetail);
    }
}
```

Figura 3-16. Clase *ConsentControllerR5*.

Ambos controladores siguen una estructura similar, lo cual asegura que las solicitudes recibidas se manejen de igual manera, sin importar si se trata de la versión de FHIR R4 o R5. A continuación se presenta una tabla con el resumen de los métodos que contienen ambos controladores:

Tabla 3–1. Métodos de los controladores

Método	Descripción	Parámetros de entrada	Salida
insertConsent	Recibe un nuevo consentimiento, detecta conflictos con consentimientos existentes para el mismo paciente y si hay conflictos envía un correo al paciente. Por último, devuelve el mismo objeto en el cuerpo de la respuesta	El nuevo consentimiento en formato String JSON	Un objeto en formato JSON que contiene un marcador booleano indicando si se han detectado conflictos y un listado con los conflictos detectados
findConsent ById	Busca consentimientos por su identificador	El identificador del consentimiento en formato String	El consentimiento, en caso de encontrarse en la base de datos, o un mensaje de error en caso contrario
findConsent ByPatientId	Busca consentimientos por el identificador del paciente	El identificador del paciente en formato String	Todos los consentimientos pertenecientes al paciente indicado, en caso de encontrarse en la base de datos, o un mensaje de error en caso contrario

3.2.2.1.2. Capa de Servicio

Esta capa es el núcleo donde reside la lógica principal del microservicio de detección de conflictos en los consentimientos. La capa de servicio se encarga de dirigir las operaciones necesarias para procesar las peticiones recibidas de la capa superior de presentación, y también de realizar todas las tareas complejas relacionadas con la detección de conflictos en los consentimientos.

Entre las funciones que se realizan en esta capa destacan:

- **Interacción con las bases de datos FHIR**

  La capa de servicio interactúa con las bases de datos FHIR R4 y R5 para recuperar consentimientos que ya se encuentren registrados. Esta operación es fundamental para que el sistema pueda comparar el nuevo consentimiento proporcionado con los existentes y detectar posibles conflictos.

```
Consent cons = fhirClientR4.read() IRead
        .resource(Consent.class) IReadTyped<Consent>
        .withId(id) IReadExecutable<Consent>
        .prettyPrint()
        .encodedJson()
        .execute();
```

Figura 3-17. Ejemplo de interacción con la base de datos FHIR R4.

- **Algoritmo de detección de conflictos**

  Esta es la operación más importante de la que se encarga la capa de servicio. Este algoritmo compara el nuevo consentimiento con los existentes para identificar posibles conflictos. Los conflictos pueden deberse entre otros a acciones contradictorias, superposiciones de fechas o actores involucrados. Dentro de este algoritmo se aplican una serie de reglas que van añadiendo, a un listado inicialmente vacío, mensajes con los posibles conflictos detectados y una vez completadas se evalúan en conjunto para establecer una conclusión respecto a la posibilidad de aceptación del nuevo consentimiento.

```
if (scopeConflict(newConsent, existingConsent)) {
    conflictMessages.add("Conflicto en el alcance: " + existingConsent.getScope().getCodingFirstRep().getCode());
} else {
    return new ConflictBlock(existingConsent.getId(), conflictMessages,  conclusion: null);
}
```

Figura 3-18. Extracto del algoritmo de detección de conflictos en su versión FHIR R4.

- **Envío de correos electrónicos**

  Esta capa es la responsable de enviar el correo electrónico al paciente, encargada por la capa de presentación. Construye el mensaje y lo envía al destinatario indicado.

Tras la ejecución del algoritmo de detección de conflictos o alguna de las consultas a las bases de datos, la capa de servicio envía una respuesta a la capa de presentación, que luego es enviada al cliente. La respuesta puede incluir los consentimientos encontrados, en caso de tratarse una consulta a la base de datos, o el detalle sobre los conflictos detectados, si se ha introducido un nuevo consentimiento.

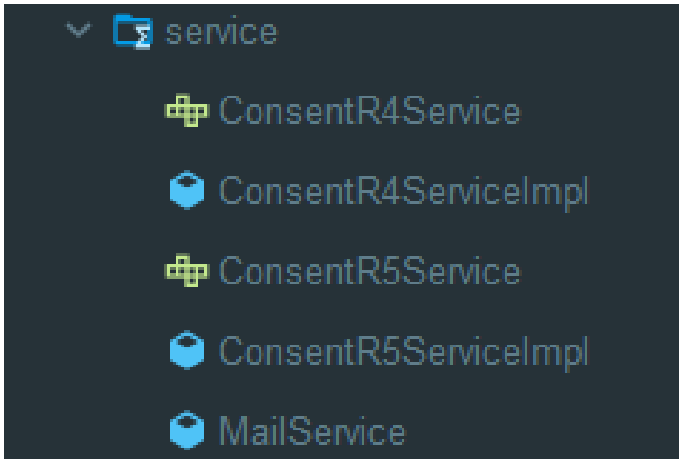


Figura 3-19. Capa de servicio del proyecto.

En esta capa nos encontramos dos interfaces, una para cada versión de FHIR:

- ***ConsentR4Service***

  Interfaz que define las operaciones relacionadas con la consulta de consentimientos y la detección de conflictos para la versión de FHIR R4.

```
public interface ConsentR4Service {  3 usages  1 implementation

    String getConsentById(String consentId);  1 usage  1 implementation

    String findConsentsByPatientId(String patientId);  1 usage  1 implementation

    ConflictDetail detectConflicts(Consent newConsent, Bundle existingConsents);  1 usage  1 implementation

    ConflictBlock isConflict(Consent newConsent, Consent existingConsent);  1 usage  1 implementation

}
```

Figura 3-20. Interfaz *ConsentR4Service*.

- ***ConsentR5Service***

  Interfaz que define las operaciones relacionadas con la consulta de consentimientos y la detección de conflictos para la versión de FHIR R5.

```
public interface ConsentR5Service {  3 usages  1 implementation

    String getConsentById(String consentId);  1 usage  1 implementation

    String findConsentsByPatientId(String patientId);  1 usage  1 implementation

    ConflictDetail detectConflicts(Consent newConsent, Bundle existingConsents);  1 usage  1 implementation

    ConflictBlock isConflictWithOppositeDecision(Consent newConsent, Consent existingConsent);  1 usage  1 implementation

    ConflictBlock isConflictWithSameDecision(Consent newConsent, Consent existingConsent);  1 usage  1 implementation

}
```

Figura 3-21. Interfaz *ConsentR5Service*.

Podemos observar cómo ambas interfaces poseen los mismos métodos para la consulta de consentimientos y para el lanzamiento del proceso de detección de conflictos. Sin embargo, ambas versiones se diferencian ya que para FHIR R4 solo tenemos el método ***isConflict()***, en el que se ejecuta toda la lógica de detección de conflictos en esta versión, y por otro lado para FHIR R5 tenemos los métodos ***isConflictWithOppositeDecision()*** y ***isConflictWithSameDecision()***, dado que en esta versión tenemos el campo *decision* que hace que varíe un poco la forma en la que se aplica la lógica de detección.

Como resumen de los métodos que contienen los servicios se presenta la siguiente tabla.

Tabla 3–2. Métodos de los servicios

Método	Descripción	Parámetros de entrada	Salida
detectConflicts	Recibe un nuevo consentimiento y el listado de consentimientos del mismo paciente y va detectando conflictos conforme va llamando a su método correspondiente según la versión de FHIR para la lógica de detección	El nuevo consentimiento como objeto *Consent* y un *Bundle* con todos los consentimientos pertenecientes al mismo paciente	Un objeto *ConflictDetail* que contiene un marcador booleano indicando si se han detectado conflictos y un listado de objetos *ConflictBlock* con los conflictos detectados
getConsentById	Busca consentimientos por su identificador	El identificador del consentimiento en formato String	El consentimiento en caso de encontrarse en la base de datos, o un mensaje de error en caso contrario
findConsents ByPatientId	Busca consentimientos por el identificador del paciente	El identificador del paciente en formato String	Todos los consentimientos pertenecientes al paciente indicado en caso de encontrarse en la base de datos, o un mensaje de error en caso contrario
isConflict (FHIR R4)	Comprueba si existen conflictos entre un nuevo consentimiento y uno existente	Dos objetos *Consent*: los consentimientos nuevo y existente	Un objeto *ConflictBlock* con los detalles de los conflictos detectados entre ambos consentimientos
isConflictWith OppositeDecision (FHIR R5)	Comprueba si existen conflictos entre un nuevo consentimiento y uno existente dado que tienen valores para el campo *decision* opuestos	Dos objetos *Consent*: los consentimientos nuevo y existente	Un objeto *ConflictBlock* con los detalles de los conflictos detectados entre ambos consentimientos
isConflictWith SameDecision (FHIR R5)	Comprueba si existen conflictos entre un nuevo consentimiento y uno existente dado que tienen valores para el campo *decision* iguales	Dos objetos *Consent*: los consentimientos nuevo y existente	Un objeto *ConflictBlock* con los detalles de los conflictos detectados entre ambos consentimientos

Como implementaciones de las dos interfaces mencionadas antes, tenemos los servicios ***ConsentR4ServiceImpl*** y ***ConsentR5ServiceImpl***. Estas implementaciones poseen métodos similares para las consultas a las respectivas bases de datos. Por tanto, en la Figura 3-22 se muestran estos métodos comunes para una de las implementaciones.

```java
@Override  1 usage
public String getConsentById(String consentId) {
    logger.info("Sirviendo petición de consultar consentimiento FHIR R5 por ID: {}", consentId);
    Consent cons = fhirClientR5.read() IRead
            .resource(Consent.class) IReadTyped<Consent>
            .withId(consentId) IReadExecutable<Consent>
            .prettyPrint()
            .encodedJson()
            .execute();
    return fhirParserR5.encodeResourceToString(cons);
}

@Override  1 usage
public String findConsentsByPatientId(String patientId){
    logger.info("Sirviendo petición de consultar consentimiento FHIR R5 por ID de paciente: {}", patientId);
    Bundle allConsents = fhirClientR5.search() IUntypedQuery<IBaseBundle>
            .forResource(Consent.class) IQuery<IBaseBundle>
            .where(new ReferenceClientParam( theName: "patient").hasId(patientId))
            .returnBundle(Bundle.class) IQuery<Bundle>
            .execute();

    return fhirParserR5.encodeResourceToString(allConsents);
}
```

Figura 3-22. Implementación métodos de consulta a la BD en *ConsentR5ServiceImpl*.

Sin embargo, en lo que respecta a la gestión y detección de conflictos general, cada implementación tiene métodos ligeramente distintos ya que cada uno sigue la lógica de detección que definimos en el apartado de diseño del sistema. A continuación, se presentan dos figuras con respectivas implementaciones de dicho método.

```java
public ConflictDetail detectConflicts(Consent newConsent, Bundle existingConsents) {
    logger.info("Sirviendo petición de detectar conflictos en consentimientos FHIR R4");
    List<ConflictBlock> conflictList = new ArrayList<>();
    logger.info("Detección de conflictos para el consent introducido por el Patient: {}", newConsent.getPatient().getReference().split( regex: "/")[1]);
    for (Bundle.BundleEntryComponent entry : existingConsents.getEntry()) {

        if (entry.getResource() instanceof Consent existingconsent) {
            ConflictBlock conflictBlock = isConflict(newConsent, existingconsent);
            if(!conflictBlock.getConflictMessages().isEmpty()){
                conflictList.add(conflictBlock);
            }
        }
    }
    return new ConflictDetail(!conflictList.isEmpty(), conflictList);
}
```

Figura 3-23. Implementación métodos *detectConflicts* en *ConsentR4ServiceImpl*.

```java
public ConflictDetail detectConflicts(Consent newConsent, Bundle existingConsents) {
    logger.info("Sirviendo petición de detectar conflictos en consentimientos FHIR R5");
    List<ConflictBlock> conflictList = new ArrayList<>();
    logger.info("Detección de conflictos para el consent introducido por el patient {}", newConsent.getSubject().getReference().split( regex: "/")[1]);
    List<Consent> oppositeDecisionConsents = new ArrayList<>();
    List<Consent> sameDecisionConsents = new ArrayList<>();

    String newConsentDecision = newConsent.getDecision().getDisplay().toLowerCase();
    String oppositeDecision = "permit".equals(newConsentDecision) ? "deny" : "permit";

    for (Bundle.BundleEntryComponent entry : existingConsents.getEntry()) {
        if (entry.getResource() instanceof Consent existingConsent) {
            String existingDecision = existingConsent.getDecision().getDisplay().toLowerCase();
            if (oppositeDecision.equals(existingDecision)) {
                oppositeDecisionConsents.add(existingConsent);
            } else if (newConsentDecision.equals(existingDecision)) {
                sameDecisionConsents.add(existingConsent);
            }
        }
    }

    for (Consent existingConsent : oppositeDecisionConsents) {
        ConflictBlock conflictBlock = isConflictWithOppositeDecision(newConsent, existingConsent);
        if (!conflictBlock.getConflictMessages().isEmpty()) {
            conflictList.add(conflictBlock);
        }
    }

    for (Consent existingConsent : sameDecisionConsents) {
        ConflictBlock conflictBlock = isConflictWithSameDecision(newConsent, existingConsent);
        if (!conflictBlock.getConflictMessages().isEmpty()) {
            conflictList.add(conflictBlock);
        }
    }

    return new ConflictDetail(!conflictList.isEmpty(), conflictList);
}
```

Figura 3-24. Implementación métodos *detectConflicts* en *ConsentR5ServiceImpl*.

Como se puede observar son similares, la principal diferencia es que en el caso de FHIR R5 tenemos dos métodos con lógicas distintas de detección de conflictos en función del valor del campo *decision*. Mientras que en el caso de FHIR R4, al no existir este campo, solo hay un método con lógica de detección de conflictos. Ambos construyen finalmente las respuestas del mismo modo.

En lo que refiere a los métodos que contienen las lógicas principales de detección, dada su elevada extensión, y que cada uno tiene campos específicos en los que detectar conflictos, se mostrarán extractos de cada uno de los métodos. Asimismo, se detallarán algunas de las reglas para emitir las conclusiones respecto a los conflictos detectados.

```java
@Override  1 usage
public ConflictBlock isConflict(Consent newConsent, Consent existingConsent) {
    List<String> conflictMessages = new ArrayList<>();
    String conclusion = "";

    if (newConsent == null || existingConsent == null) {
        return new ConflictBlock(existingConsent != null ? existingConsent.getId() : "unknown",
                List.of( e1: "Uno de los consentimientos es nulo."),
                 conclusion: "No se puede determinar el conflicto debido a datos incompletos.");
    }
    if (scopeConflict(newConsent, existingConsent)) {
        conflictMessages.add("Conflicto en el alcance: " + existingConsent.getScope().getCodingFirstRep().getCode());
    } else {
        return new ConflictBlock(existingConsent.getId(), conflictMessages,  conclusion: null);
    }
    boolean datesOverlap = checkDatesOverlap(newConsent, existingConsent);
    boolean actorsConflict = actorsMatch(newConsent, existingConsent);
    boolean resourcesConflict = resourcesMatch(newConsent, existingConsent);
    boolean actionConflict = actionConflict(newConsent, existingConsent);
    boolean typeConflict = provisionTypeConflict(newConsent, existingConsent);

    if (datesOverlap) {
        conflictMessages.add("Las fechas se superponen con el consentimiento existente: " + existingConsent.getId());

        if (actorsConflict) {
            conflictMessages.add("Los actores involucrados en ambos consentimientos son los mismos: " +
                    existingConsent.getProvision().getActorFirstRep().getReference().getReference());

            if (resourcesConflict) {
                conflictMessages.add("El nuevo consentimiento y el existente afectan al mismo recurso: " +
                        newConsent.getProvision().getDataFirstRep().getReference().getReference());

                if (actionConflict && typeConflict) {
                    conflictMessages.add("El nuevo consentimiento realiza el mismo tipo de acción que el existente: " +
                            newConsent.getProvision().getActionFirstRep().getCodingFirstRep().getCode());
```

Figura 3-25. Extracto Implementación método is*Conflict* en *ConsentR4ServiceImpl*.

```java
//REGLAS DE DETECCION FHIR R4
private boolean scopeConflict(Consent newConsent, Consent existingConsent){  1 usage
    if (newConsent.getScope().getCodingFirstRep().getCode() == null || existingConsent.getScope().getCodingFirstRep().getCode() == null) {
        return false;
    }
    return existingConsent.getScope().equalsDeep(newConsent.getScope());
}

private boolean checkDatesOverlap(Consent newConsent, Consent existingConsent){  1 usage
    if (newConsent.getProvision().getPeriod().isEmpty() || existingConsent.getProvision().getPeriod().isEmpty()) {
        return false;
    }
    return newConsent.getProvision().getPeriod().getStart().before(existingConsent.getProvision().getPeriod().getEnd()) &&
            newConsent.getProvision().getPeriod().getEnd().after(existingConsent.getProvision().getPeriod().getStart());
}

private boolean provisionTypeConflict(Consent newConsent, Consent existingConsent){  1 usage
    if (newConsent.getProvision().getType() == null || existingConsent.getProvision().getType() == null) {
        return false;
    }
    return !existingConsent.getProvision().getType().equals(newConsent.getProvision().getType());
}
```

Figura 3-26. Extracto reglas de detección de conflictos en *ConsentR4ServiceImpl*.

Para el caso de FHIR R4, la lógica que se sigue es si el ámbito de aplicación (*scope*) de los consentimientos no es coincidente, no se considera conflicto. En caso contrario se van aplicando más reglas para detectar solapamiento de fechas y campos relevantes dentro de *provision*. Una vez se han aplicado las reglas se procede a su evaluación conjunta para dar una conclusión respecto a los conflictos. Así tenemos las siguientes conclusiones:

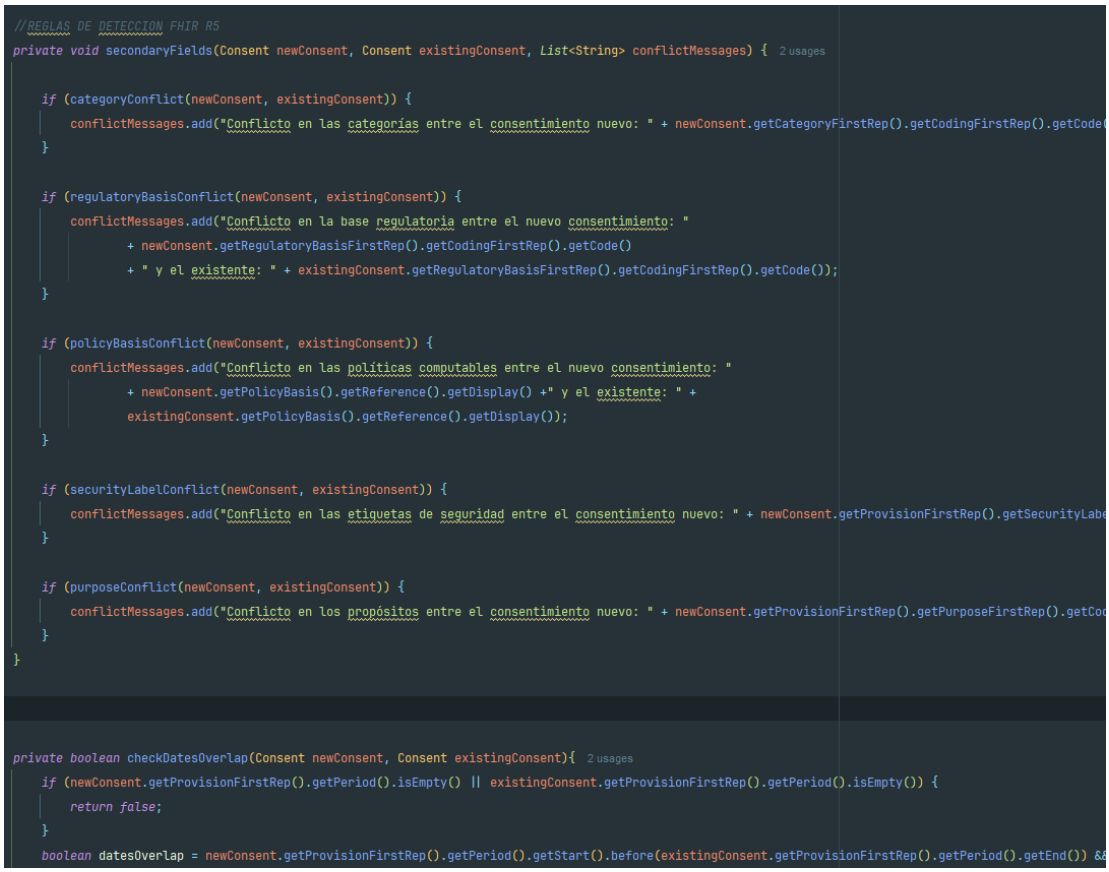
- **Conflicto Total**: Si se realizan las mismas acciones, pero con indicación contraria (*permit o deny*), ya sea sobre el mismo recurso o sobre un ámbito general.
- **Conflicto Parcial**: Si los consentimientos coinciden en algunos campos, pero no en todos y entonces existe la posibilidad de que puedan coexistir o que el paciente elija cuál quiere mantener.
- **Sin conflicto**: No hay suficientes conflictos relevantes como para considerar que ambos consentimientos son incompatibles.

```
public ConflictBlock isConflictWithOppositeDecision(Consent newConsent, Consent existingConsent) {
    List<String> conflictMessages = new ArrayList<>();
    String conclusion = "";

    if (newConsent == null || existingConsent == null) {
        return new ConflictBlock(existingConsent != null ? existingConsent.getId() : "unknown",
                List.of( e1: "Uno de los consentimientos es nulo."),
                 conclusion: "No se puede determinar el conflicto debido a datos incompletos.");
    }

    conflictMessages.add("Potencial Conflicto por decision opuesta: " + existingConsent.getDecision().getDisplay());

    boolean datesOverlap = checkDatesOverlap(newConsent, existingConsent);
    boolean actorsConflict = actorsMatch(newConsent, existingConsent);
    boolean resourcesConflict = resourcesMatch(newConsent, existingConsent);
    boolean actionsConflict = actionConflict(newConsent, existingConsent);

    //En este caso somos más estrictos ya que la regla general es contraria
    if (datesOverlap && actorsConflict) {
        conflictMessages.add("Las fechas se superponen con el consentimiento existente: " + existingConsent.getId());
        conflictMessages.add("Los actores involucrados en ambos consentimientos son los mismos: " +
                existingConsent.getProvisionFirstRep().getActorFirstRep().getReference().getReference());
        if (resourcesConflict && actionsConflict) {
            conflictMessages.add("El nuevo consentimiento y el existente afectan al mismo recurso: " +
                    newConsent.getProvisionFirstRep().getDataFirstRep().getReference().getReference());
            conflictMessages.add("El nuevo consentimiento realiza el mismo tipo de acción que el existente: " +
                    newConsent.getProvisionFirstRep().getActionFirstRep().getCodingFirstRep().getCode());
            conclusion = "Conflicto total: Consentimientos con decisiones opuestas afecta al mismo actor y recursos con la m
        }
        else if(actionsConflict){
            conflictMessages.add("El nuevo consentimiento realiza el mismo tipo de acción que el existente: " +
                    newConsent.getProvisionFirstRep().getActionFirstRep().getCodingFirstRep().getCode());
            conclusion = "Conflicto total: Consentimientos con decisiones opuestas afecta al mismo actor con la misma accion
        }
```

Figura 3-27. Extracto Implementación método is*ConflictWithOppositeDecision* en *ConsentR5ServiceImpl*.

```java
public ConflictBlock isConflictWithSameDecision(Consent newConsent, Consent existingConsent) {
    List<String> conflictMessages = new ArrayList<>();
    String conclusion = "";

    if (newConsent == null || existingConsent == null) {
        return new ConflictBlock(existingConsent != null ? existingConsent.getId() : "unknown",
                List.of( e1: "Uno de los consentimientos es nulo."),
                 conclusion: "No se puede determinar el conflicto debido a datos incompletos.");
    }

    boolean datesOverlap = checkDatesOverlap(newConsent, existingConsent);
    boolean actorsConflict = actorsMatch(newConsent, existingConsent);
    boolean resourcesConflict = resourcesMatch(newConsent, existingConsent);
    boolean actionConflict = actionConflict(newConsent, existingConsent);

    //Logica parecida al caso de FHIR R4
    if (datesOverlap) {
        conflictMessages.add("Las fechas se superponen con el consentimiento existente: " + existingConsent.getId());

        if (actorsConflict) {
            conflictMessages.add("Los actores involucrados en ambos consentimientos son los mismos: " +
                    existingConsent.getProvisionFirstRep().getActorFirstRep().getReference().getReference());

            if (resourcesConflict) {
                conflictMessages.add("El nuevo consentimiento y el existente afectan al mismo recurso: " +
                        newConsent.getProvisionFirstRep().getDataFirstRep().getReference().getReference());

                if (actionConflict) {
                    conflictMessages.add("El nuevo consentimiento realiza el mismo tipo de acción que el existente: " +
                            newConsent.getProvisionFirstRep().getActionFirstRep().getCodingFirstRep().getCode());
                    conclusion = "Conflicto parcial: los consentimientos afectan al mismo recurso, pero las acciones no son completamente
                }

            } else {
                if (actionConflict) {
```

Figura 3-28. Extracto Implementación método is*ConflictWithSameDecision* en *ConsentR5ServiceImpl*.



```
//REGLAS DE DETECCION FHIR R5
private void secondaryFields(Consent newConsent, Consent existingConsent, List<String> conflictMessages) {  2 usages

    if (categoryConflict(newConsent, existingConsent)) {
        conflictMessages.add("Conflicto en las categorías entre el consentimiento nuevo: " + newConsent.getCategoryFirstRep().getCodingFirstRep().getCode(
    }

    if (regulatoryBasisConflict(newConsent, existingConsent)) {
        conflictMessages.add("Conflicto en la base regulatoria entre el nuevo consentimiento: "
                + newConsent.getRegulatoryBasisFirstRep().getCodingFirstRep().getCode()
                + " y el existente: " + existingConsent.getRegulatoryBasisFirstRep().getCodingFirstRep().getCode());
    }

    if (policyBasisConflict(newConsent, existingConsent)) {
        conflictMessages.add("Conflicto en las políticas computables entre el nuevo consentimiento: "
                + newConsent.getPolicyBasis().getReference().getDisplay() +" y el existente: " +
                existingConsent.getPolicyBasis().getReference().getDisplay());
    }

    if (securityLabelConflict(newConsent, existingConsent)) {
        conflictMessages.add("Conflicto en las etiquetas de seguridad entre el consentimiento nuevo: " + newConsent.getProvisionFirstRep().getSecurityLabe
    }

    if (purposeConflict(newConsent, existingConsent)) {
        conflictMessages.add("Conflicto en los propósitos entre el consentimiento nuevo: " + newConsent.getProvisionFirstRep().getPurposeFirstRep().getCod
    }
}


private boolean checkDatesOverlap(Consent newConsent, Consent existingConsent){  2 usages
    if (newConsent.getProvisionFirstRep().getPeriod().isEmpty() || existingConsent.getProvisionFirstRep().getPeriod().isEmpty()) {
        return false;
    }
    boolean datesOverlap = newConsent.getProvisionFirstRep().getPeriod().getStart().before(existingConsent.getProvisionFirstRep().getPeriod().getEnd()) &&
```

Figura 3-29. Extracto reglas de detección de conflictos en *ConsentR5ServiceImpl*.

En el caso de FHIR R5, la lógica que se sigue es que en función del valor que tome el campo *decision* se aplicará un método para la detección de conflictos u otro. En caso de que el valor de los campos *decision* sea opuesto, se aplicarán las reglas del método ***isConflictWithOppositeDecision*** que son más estrictas. Por otro lado, si el valor de los campos *decision* son iguales, se aplicarán las reglas del método ***isConflictWithSameDecision*** que son similares a las aplicadas para FHIR R4. Una vez se han aplicado las reglas, en cada método, se procede a su evaluación conjunta para dar una conclusión respecto a los conflictos. Así tenemos las siguientes conclusiones:

- **Conflicto Total (Solo *decision* opuestos)**: Si se realizan las mismas acciones, ya sea sobre el mismo recurso o sobre un ámbito general, pero en este caso marcadas por valores de *decision* opuestos (*permit o deny*),
- **Conflicto Parcial (Solo *decision* iguales)**: Si los consentimientos coinciden en algunos campos, pero no en todos y entonces existe la posibilidad de que puedan coexistir o que el paciente elija cual quiere mantener.
- **Sin conflicto (Ambos valores de *decision*)**: No hay suficientes conflictos relevantes como para considerar que ambos consentimientos son incompatibles.

Por último, en esta capa nos encontramos el servicio ***MailService*** cuya función principal es el envío de correos electrónicos. Este servicio tiene dos métodos:

1. ***sendEmail***: Envío del correo electrónico al destinatario indicado, junto con el asunto el cuerpo del mensaje.
2. ***generateEmailBody***: Construcción del cuerpo del mensaje. Se mapea un objeto *ConflictDetail* a un JSON String que pueda enviarse como cuerpo del mensaje correctamente.

```java
@Service  4 usages
public class MailService {

    @Autowired  1 usage
    private JavaMailSender mailSender;

    public void sendEmail(String to, String subject, String body) {  2 usages
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject(subject);
        message.setText(body);

        mailSender.send(message);
    }

    public String generateEmailBody(ConflictDetail conflictDetail) {  2 usages
        try {
            ObjectMapper objectMapper = new ObjectMapper();
            return objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(conflictDetail);
        } catch (Exception e) {
            e.printStackTrace();
            return "{}";
        }
    }
}
```

Figura 3-30. Clase *MailService*.

3.2.2.1.3. Capa de Dominio (Capa de Modelos)

Esta capa es el lugar donde se definen los objetos que representan datos importantes para el microservicio. En este proyecto esta capa es donde residen las clases que contendrán toda la información importante relativa a la detección de conflictos.

Figura 3-31. Capa de dominio del proyecto.

Dentro de esta capa tenemos las siguientes clases:

- ***ConflictBlock***

  Esta clase representa un bloque de conflictos específico detectado entre consentimientos. Como atributos tiene una referencia del consentimiento con el que se ha comparado, una lista a la que se van añadiendo mensajes por cada posible conflicto que se detecte, y por último una conclusión sobre todos los conflictos detectados en esa comparación.

```java
@Getter  24 usages
@AllArgsConstructor
public class ConflictBlock {
    private String reference;
    private List<String> conflictMessages;
    private String conclusion;
}
```

Figura 3-32. Clase *ConflictBlock*.

- ***ConflictDetail***

  Esta clase representa el resultado final de la operación de detección de conflictos entre consentimientos. Como atributos tiene un indicador booleano para indicar si, en general, ha habido o no conflictos, y por último contiene un listado de objetos *ConflictBlock* anteriormente detallado.

```java
@AllArgsConstructor  25 usages
@Getter
public class ConflictDetail {
    private boolean conflict;
    private List<ConflictBlock> conflictList;
}
```

Figura 3-33. Clase *ConflictDetail*.

3.2.2.1.4. Configuración y arranque del proyecto

Una vez hemos descrito todas las capas del proyecto, es necesario detallar algunos puntos importantes del microservicio que aseguran su correcto funcionamiento:

- **Inicio de la aplicación**

  El proyecto dispone de la clase ***ConsentConflictApplication***, la cual contiene el método *main()* para arrancar la aplicación.

```java
@SpringBootApplication
public class ConsentConflictApplication {

    public static void main(String[] args) { SpringApplication.run(ConsentConflictApplication.class, args); }

}
```

Figura 3-34. Clase *ConsentConflictApplication*.

En esta clase se usa la anotación *@SpringBootApplication* para configurar que la aplicación use las características automáticas de Spring Boot [32]. Dentro del método *main()*, se llama al método *run()* para lanzar la aplicación configurando todos los recursos necesarios y arrancando el servidor Tomcat integrado que Spring provee por defecto [33].

- **Gestión del proyecto**

  El proyecto dispone del fichero ***pom.xml***, que contiene toda la información del proyecto. Entre esta información destacan la gestión de dependencias, datos para su identificación, y todo lo relativo a la compilación y empaquetado del proyecto.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version>
        <relativePath/>
    </parent>
    <groupId>com.tfg</groupId>
    <artifactId>consentconflict</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Consent Conflict</name>
    <description>TFG Deteccion conflictos consentimientos pacientes</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

Figura 3-35. Extracto fichero *pom.xml* del proyecto.

En este proyecto se han necesitado las siguientes dependencias:

1. **Spring**: Todas las herramientas del framework para montar la base del servicio, documentar las APIs en Swagger y para enviar correos electrónicos.
2. **Lombok**: Para simplificar y eliminar código repetitivo.
3. **HAPI FHIR**: Todas las herramientas necesarias para manejar los recursos FHIR en el microservicio.

- **Propiedades del proyecto**

  El proyecto dispone del fichero ***application.properties***, que contiene las propiedades necesarias para la configuración principal del microservicio. Dentro de este fichero, se encuentran las propiedades del servidor, las direcciones de conexión con las bases de datos HAPI FHIR tanto R4 como R5, así como la configuración necesaria para el correcto envío de correos electrónicos.

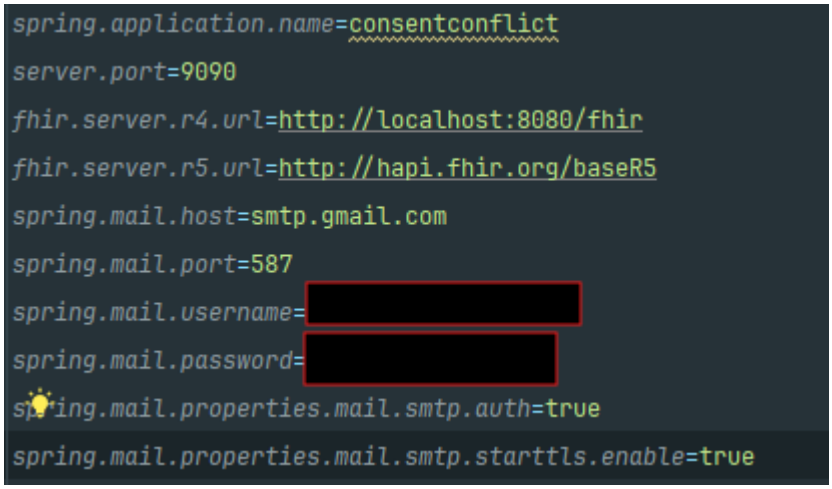


Figura 3-36. Extracto fichero *application.properties* del proyecto.

- **Configuración del servicio**

  El proyecto dispone de la clase de configuración ***FhirConfig*** que se utiliza para definir los "beans" de FHIR que se usan en el microservicio. Esta clase asegura que el microservicio pueda interactuar correctamente con las bases de datos FHIR R4 y R5, a través de los objetos *IGenericClient*, así como la correcta serialización y deserialización de objetos FHIR, gracias a los objetos *IParser*.

```java
@Configuration  no usages
public class FhirConfig {
    @Value("${fhir.server.r4.url}")  1 usage
    private String fhirR4ServerUrl;
    @Value("${fhir.server.r5.url}")  1 usage
    private String fhirR5ServerUrl;
    //Configuracion  HAPI FHIR R4
    @Bean  no usages
    public IGenericClient fhirClientR4() {
        FhirContext context = FhirContext.forR4();
        return context.newRestfulGenericClient(fhirR4ServerUrl);
    }

    @Bean  no usages
    public IParser fhirParserR4(){
        FhirContext context = FhirContext.forR4();
        return context.newJsonParser();
    }

    //Configuracion  HAPI FHIR R5
    @Bean  no usages
    public IGenericClient fhirClientR5() {
        FhirContext context = FhirContext.forR5();
        return context.newRestfulGenericClient(fhirR5ServerUrl);
    }

    @Bean  no usages
    public IParser fhirParserR5(){
        FhirContext context = FhirContext.forR5();
        return context.newJsonParser();
    }
}
```

Figura 3-37. Clase *FhirConfig*.

- **Registro de logs en el proyecto**

  El proyecto dispone del fichero de configuración XML de logs ***logback.xml***. En este fichero se recoge qué y cómo se guardarán y mostrarán los logs de este servicio. En este fichero destaca la configuración para mostrar los logs por consola y los que se redirigirán a un fichero de registro. Los logs que salgan por consola solo serán accesibles por el equipo de infraestructura que se haga cargo del sistema, ya que contienen información más detallada de errores de todo el sistema. El fichero de log ***conflict_detection.log*** solo almacenará detalles del sistema relacionados con la lógica principal del microservicio.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>



    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>



    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>logs/conflict_detection.log</file>


        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} - %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <logger name="com.tfg.consentconflict" level="*">
        <appender-ref ref="FILE"/>
    </logger>



    <root level="info">
        <appender-ref ref="CONSOLE"/>
    </root>

</configuration>
```

Figura 3-38. Extracto fichero *logback.xml* del proyecto.

```
c.t.c.ConsentConflictApplication - Running with Spring Boot v3.2.5, Spring v6.1.6
c.t.c.ConsentConflictApplication - No active profile set, falling back to 1 default profile: "default"
c.t.c.ConsentConflictApplication - Started ConsentConflictApplication in 2.401 seconds (process running for 2.852)
c.t.c.api.ConsentControllerR4 - Petición de detectar conflictos para nuevo consentimiento FHIR R4: POST /consent recibida
c.t.c.api.ConsentControllerR4 - Petición de consultar consentimiento FHIR R4 por ID de paciente: GET /patient/1 recibida
c.t.c.service.ConsentR4ServiceImpl - Sirviendo petición de consultar consentimiento FHIR R4 por ID de paciente: 1
c.t.c.service.ConsentR4ServiceImpl - Sirviendo petición de detectar conflictos en consentimientos FHIR R4
```

Figura 3-39. Extracto contenido fichero *conflicto_detection.log*.

- **Manejo de excepciones en el proyecto**

  El proyecto dispone de la clase ***ConsentExceptionHandler***, la cual se encarga de manejar excepciones no controladas y enviar un mensaje de error para la petición que desencadenó la excepción.

```java
@ControllerAdvice  1 usage
public class ConsentExceptionHandler {

    private static final Logger logger = LoggerFactory.getLogger(ConsentExceptionHandler.class);

    @ExceptionHandler(JsonParseException.class)  no usages
    public ResponseEntity<Object> BadJsonException(JsonParseException ex) {
        Map<String, Object> body = new HashMap<>();
        body.put("message", "Hubo un problema en el JSON. Reviselo por favor");
        body.put("status", HttpStatus.BAD_REQUEST.value());
        logger.error("Error al procesar el JSON recibido {}", ex.getMessage());
        return new ResponseEntity<>(body, HttpStatus.BAD_REQUEST);
    }
}
```

Figura 3-40. Clase *ConsentExceptionHandler*.

3.3. Pruebas realizadas

En este apartado se van a mostrar algunas de las pruebas que se han realizado para comprobar el funcionamiento del servicio. Se hará distinción entre pruebas realizas para las versiones de FHIR R4 y R5 respectivamente.

3.3.1. Configuración previa

Antes de la realización de las pruebas, es necesario arrancar el servidor local HAPI FHIR R4 y cargarle unos datos iniciales. Entre estos datos se encuentran pacientes, organizaciones, personal médico y consentimientos. Para el caso de HAPI FHIR R5, aunque la base de datos pública ya tenga suficientes datos, cargaremos también algunos consentimientos extra.

Una vez cargados los datos necesarios, procedemos al arranque del servicio, el cual se encuentra disponible por defecto a través del puerto **9090** de la máquina local. Una vez se ha iniciado por completo, accedemos al **Swagger UI** del servicio desde donde se realizarán las pruebas, en la dirección http://localhost:9090/swagger-ui/index.html.

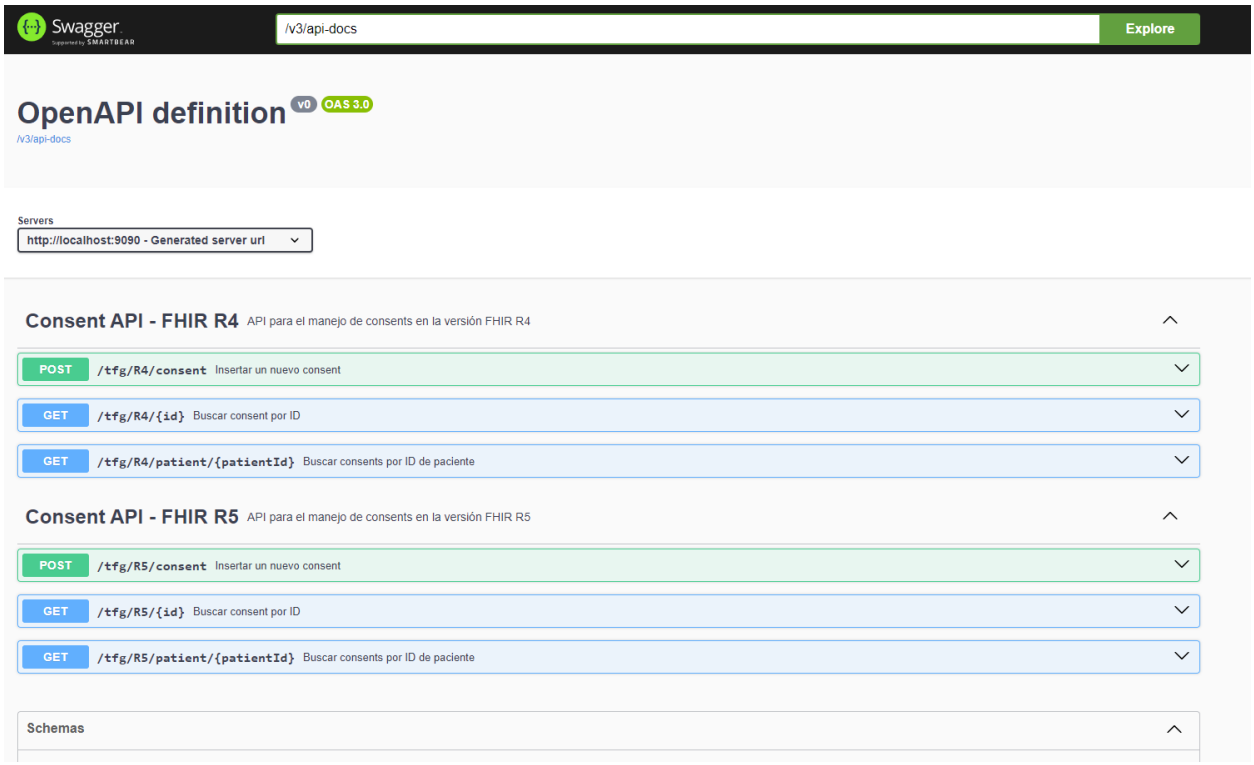


Figura 3-41. Swagger UI del servicio de detección de conflictos.

Se puede observar cómo está dividido el Swagger en dos bloques. Por un lado, tenemos las peticiones para FHIR R4 y por otro las de FHIR R5. En este apartado solo probaremos los endpoints relacionados con la detección de conflictos, pero también están disponibles los otros dos para la consulta de consentimientos en caso de que resulten necesarios para realizar alguna operación.

Para más detalles sobre todo el procedimiento de configuración y despliegue del entorno ir al **Anexo 1**.

3.3.2. Pruebas FHIR R4

En este punto se muestran las pruebas realizadas para la detección de conflictos en consentimientos que siguen la versión de FHIR R4. Las pruebas realizadas tienen como objetivo mostrar el correcto funcionamiento del servicio para distintos casos de conflictos:

1. **Prueba de conflicto total**

   En esta prueba se trata de verificar que el servicio detecta correctamente un conflicto total entre dos consentimientos, cuando ambos realizan acciones contradictorias sobre el mismo recurso. Partimos del *Consent* mostrado en la Figura 3-42 y almacenado en la base de datos FHIR.

   Este consentimiento, permite que un investigador o profesional médico recoja datos específicos de un paciente para el propósito de investigación. Este consentimiento es válido desde el 10 de julio de 2024 hasta el 10 de julio de 2026. Si ahora el paciente trata de otorgar un nuevo consentimiento que deniega la misma acción que el consentimiento existente y en el mismo periodo de tiempo, el sistema detectará un conflicto total.

Figura 3-42. *Consent* almacenado FHIR R4.

Para comprobar este comportamiento, enviamos desde el Swagger la solicitud POST al endpoint **/tfg/R4/consent** con el nuevo consentimiento descrito en el cuerpo del mensaje**.**

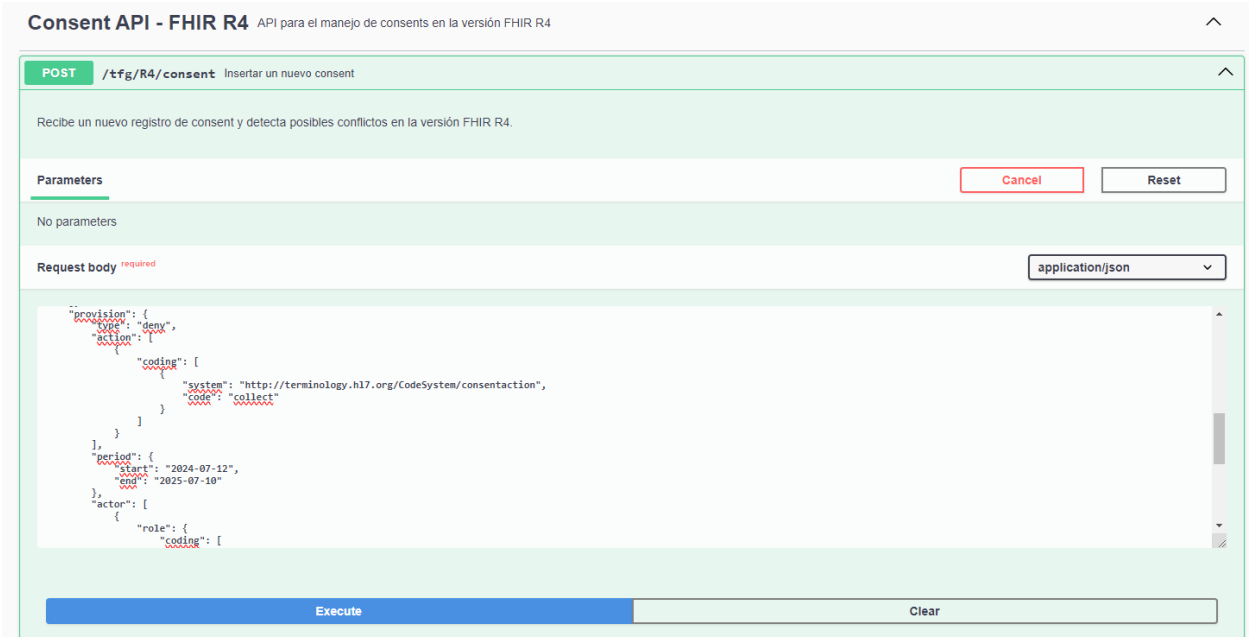


Figura 3-43. Envío *Consent* conflicto total FHIR R4.

Una vez procesada la petición, recibimos la respuesta en la que se verifica el comportamiento esperado. Además, vemos el listado con mensajes relacionados con todos los campos en los que las reglas han detectado conflictos.

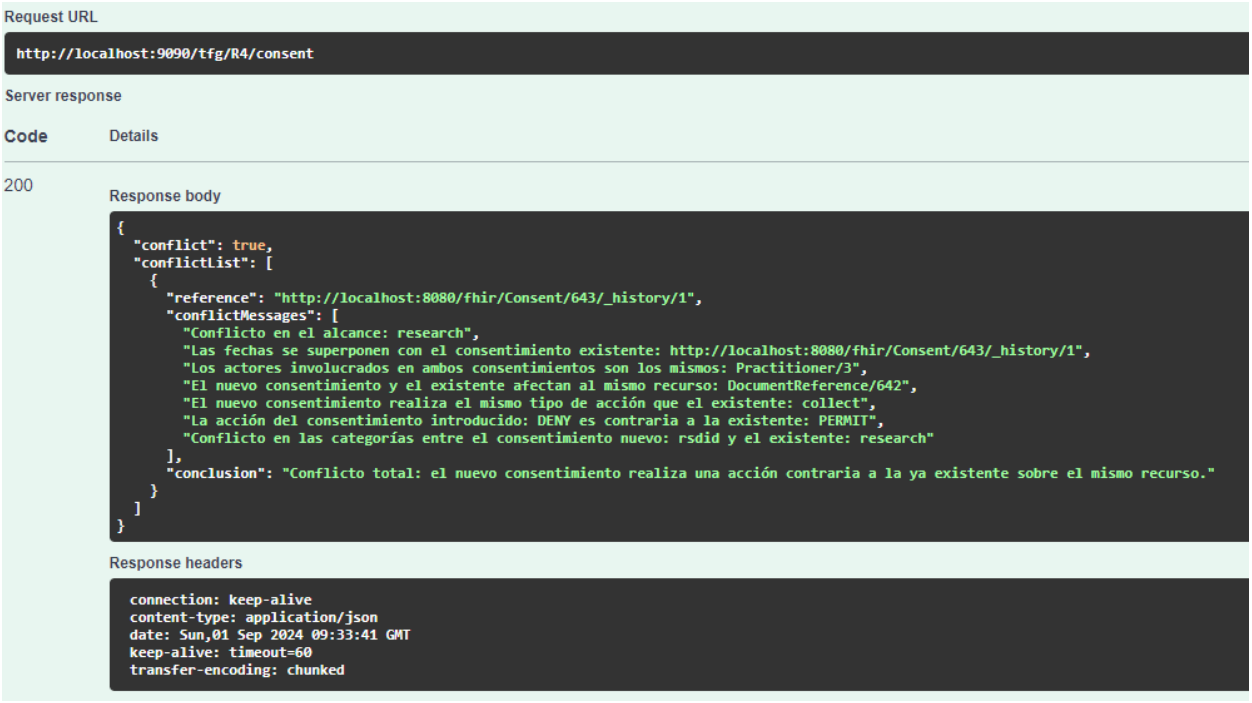


Figura 3-44. Respuesta *Consent* conflicto total FHIR R4.

2. **Prueba de conflicto parcial**

En esta prueba se trata de verificar que el servicio detecta correctamente un conflicto parcial entre dos consentimientos, cuando ambos coinciden en algunos campos, pero pueden coexistir. Partimos del mismo *Consent* existente, utilizado para la anterior prueba. En este caso el paciente trata de otorgar un nuevo consentimiento, pero esta vez se deniega al mismo profesional médico para recoger datos específicos del paciente con el mismo propósito de investigación, siendo los datos afectados un documento distinto al recogido en el otro consentimiento.

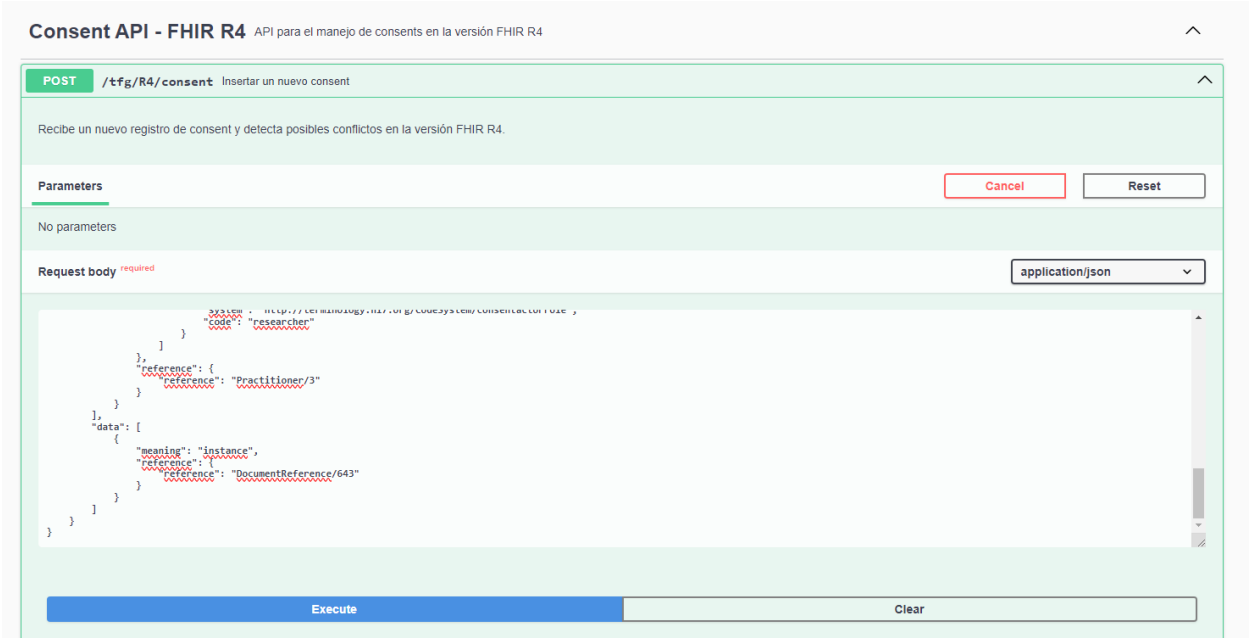


Figura 3-45. Envío *Consent* conflicto parcial FHIR R4.

Si enviamos la misma solicitud con POST descrita en la anterior prueba, pero con el nuevo consentimiento, tras haber sido procesada podemos comprobar que la respuesta verifica el comportamiento esperado. Igualmente vemos el listado con mensajes sobre los conflictos detectados.

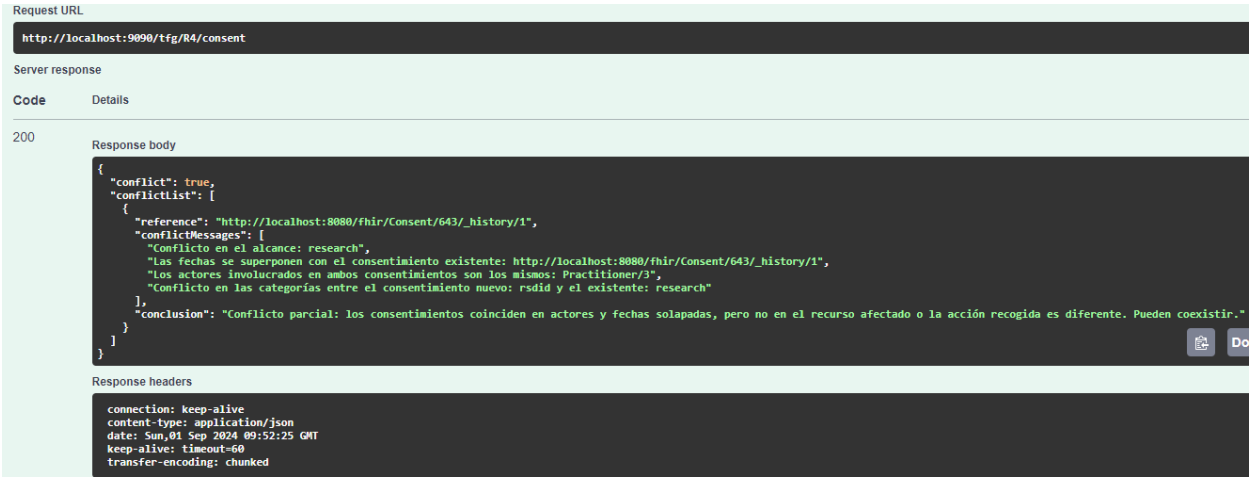


Figura 3-46. Respuesta *Consent* conflicto parcial FHIR R4.

3. **Prueba de no conflictos**

En esta breve prueba se puede comprobar cómo en caso de que los consentimientos tengan distintos ámbitos de aplicación no habrá conflictos. Para esta prueba partimos igualmente del mismo consentimiento de las otras pruebas, cuyo ámbito de aplicación es la investigación. Ahora el paciente trata de otorgar un nuevo consentimiento cuyo ámbito sea la privacidad de sus datos, es decir, que se pueda acceder, recopilar, usar o divulgar sus datos.

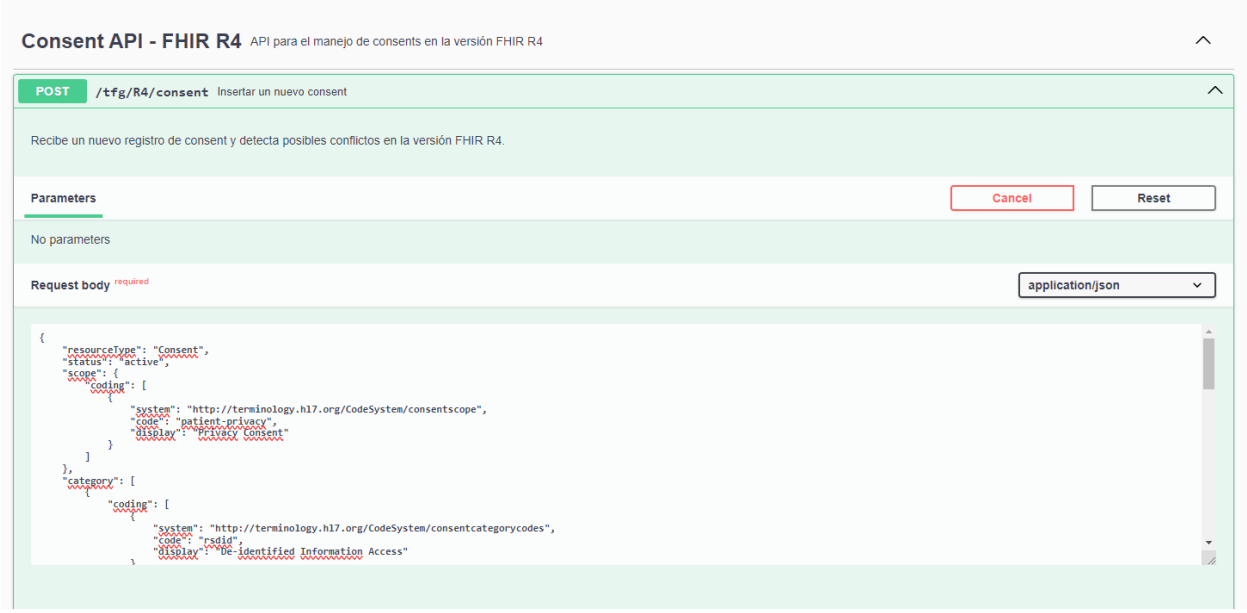


Figura 3-47. Envío *Consent* sin conflictos FHIR R4.

Si enviamos la misma solicitud con POST descrita en las anteriores pruebas, pero con el nuevo consentimiento, tras haber sido procesada podemos comprobar que la respuesta verifica el comportamiento esperado. En este caso no habrá mensajes puesto que no hay conflictos.

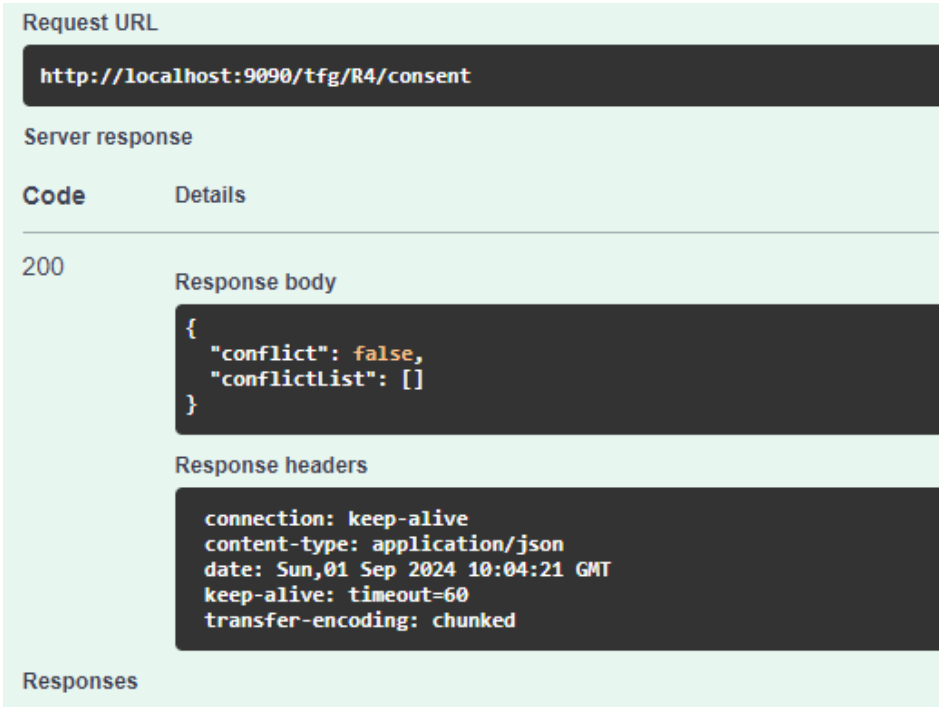


Figura 3-48. Respuesta *Consent* sin conflictos FHIR R4.

3.3.3. Pruebas FHIR R5

En este apartado seguiremos el mismo esquema seguido para las pruebas de FHIR R4, pero teniendo en cuenta en este caso los cambios que introduce la inclusión del campo *decision*. Así tenemos las siguientes pruebas:

1. **Prueba de conflicto total con decisiones opuestas**

   En esta prueba se trata de verificar que el servicio detecta correctamente un conflicto total entre dos consentimientos, cuando ambos tienen decisiones opuestas sobre el mismo recurso. Partimos del siguiente *Consent* almacenado en la base de datos FHIR.


```
{
  "resourceType": "Consent",
  "id": "765980",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2024-09-01T10:16:34.393+00:00",
    "source": "#wa9alGoL1WBLbSgo"
  },
  "status": "active",
  "category": [ {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "89057-4",
      "display": "Patient Consent"
    } ]
  } ],
  "subject": {
    "reference": "Patient/86aa6c8c-f050-4602-8591-e77ee8e0e27c"
  },
  "date": "2024-01-01",
  "period": {
    "start": "2024-01-01",
    "end": "2024-06-30"
  },
  "grantor": [ {
    "reference": "Practitioner/866"
  } ],
  "grantee": [ {
    "reference": "Organization/80037"
  } ],
  "decision": "deny",
  "provision": [ {
    "period": {
      "start": "2024-01-01",
      "end": "2024-06-30"
    },
    "actor": [ {
      "role": {
        "coding": [ {
          "system": "http://hl7.org/fhir/consentactor",
          "code": "patient"
        } ]
      },
      "reference": {
        "reference": "Patient/86aa6c8c-f050-4602-8591-e77ee8e0e27c"
      }
    } ],
    "action": [ {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/consentaction",
        "code": "access"
      } ]
    } ],
    "data": [ {
      "meaning": "instance",
      "reference": {
        "reference": "DocumentReference/956"
      }
    } ]
  } ]
}
```


Figura 3-49. *Consent* almacenado FHIR R5.

Con este consentimiento el paciente deniega el acceso a un documento específico en el periodo comprendido entre el 1 de enero de 2024 y el 30 de junio de 2024. Esta denegación afecta a profesionales pertenecientes a una organización concreta. Si ahora el paciente, trata de otorgar un nuevo consentimiento que afecte al mismo recurso y actores en el mismo periodo, pero con decisiones opuestas, el sistema detectará un conflicto total.

Para comprobar este comportamiento, enviamos desde el Swagger la solicitud POST al endpoint **/tfg/R5/consent** con el nuevo consentimiento descrito en el cuerpo del mensaje**.**

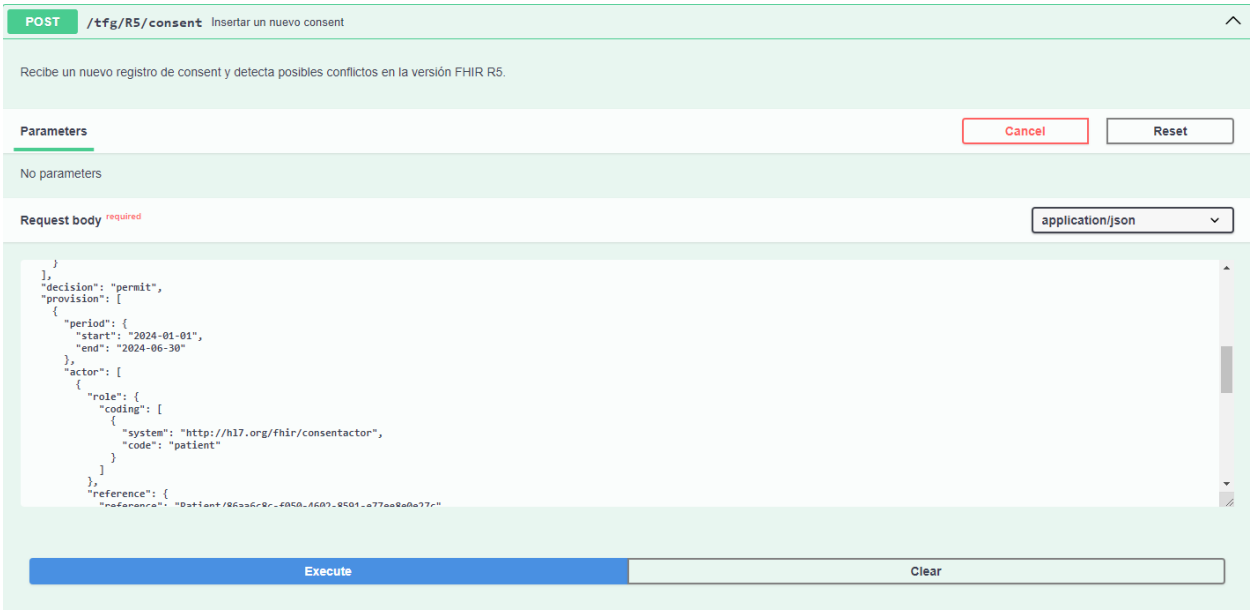


Figura 3-50. Envío *Consent* conflicto total FHIR R5.

Una vez procesada la petición, recibimos la respuesta en la que se verifica el comportamiento esperado. Además, vemos el listado con mensajes relacionados con todos los campos en los que las reglas han detectado conflictos.

Figura 3-51. Respuesta *Consent* conflicto total FHIR R5.

2. **Prueba de conflicto parcial con mismas decisiones**

En esta prueba se trata de verificar que el servicio detecta correctamente un conflicto parcial entre dos consentimientos, cuando ambos toman la misma decisión y coinciden en algunos campos, pero pueden coexistir. Partimos del mismo *Consent* existente utilizado para la anterior prueba. En este caso el paciente trata de otorgar un nuevo consentimiento, pero esta vez se deniega a otro profesional médico

para recoger datos específicos del paciente, siendo los datos afectados el mismo documento recogido en el otro consentimiento.

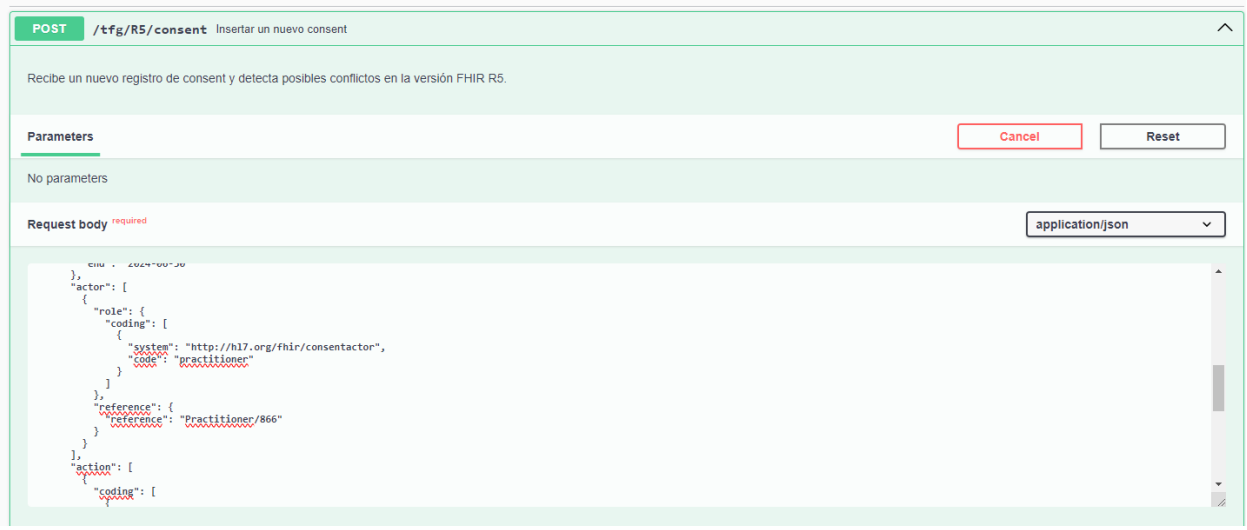


Figura 3-52. Envío *Consent* conflicto parcial FHIR R5.

Si enviamos la misma solicitud con POST descrita en la anterior prueba, pero con el nuevo consentimiento, tras haber sido procesada podemos comprobar que la respuesta verifica el comportamiento esperado. Igualmente vemos el listado con mensajes sobre los conflictos detectados.

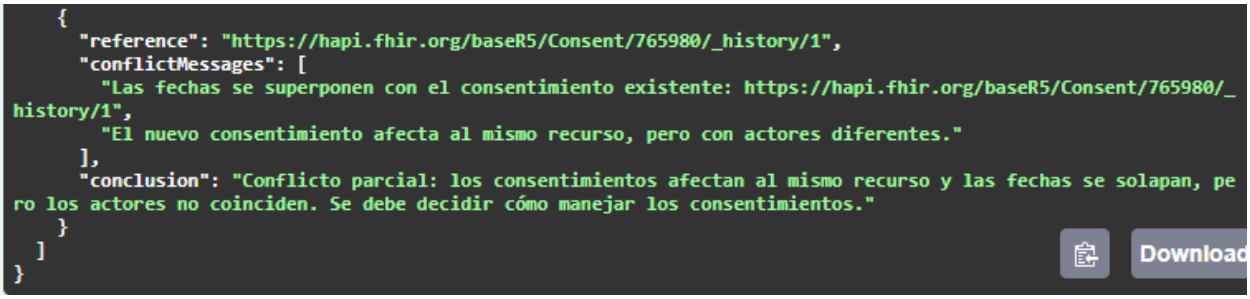


Figura 3-53. Respuesta *Consent* conflicto parcial FHIR R5.

3. **Prueba de no conflictos bajo la misma decisión**

En esta breve prueba se puede comprobar cómo en caso de que los consentimientos tengan distintos valores para sus campos en caso de misma decisión, no habrá conflictos. Para esta prueba partimos igualmente del mismo consentimiento de las otras pruebas. Si ahora el paciente trata de otorgar un nuevo consentimiento con la misma decisión en otro periodo de validez, para otro recurso, no se detectarán conflictos.

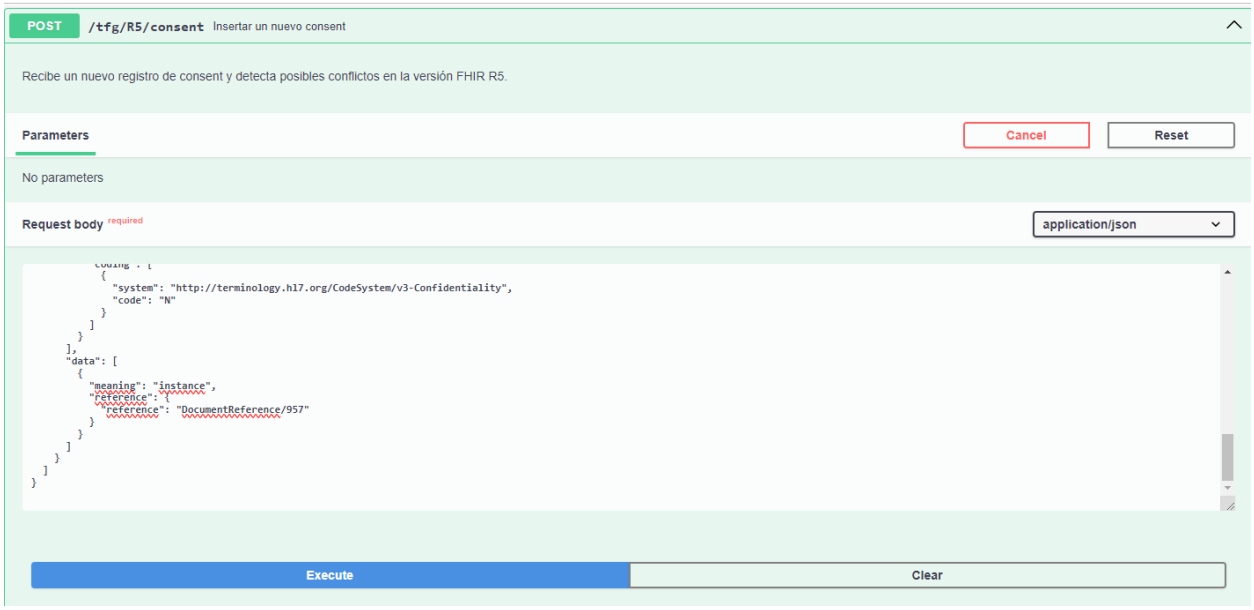


Figura 3-54. Envío *Consent* sin conflictos FHIR R5.

Si enviamos la misma solicitud con POST descrita en las anteriores pruebas, pero con el nuevo consentimiento, tras haber sido procesada podemos comprobar que la respuesta verifica el comportamiento esperado. En este caso se indicará que no se han detectado conflictos aparentes en relación con el consentimiento existente.

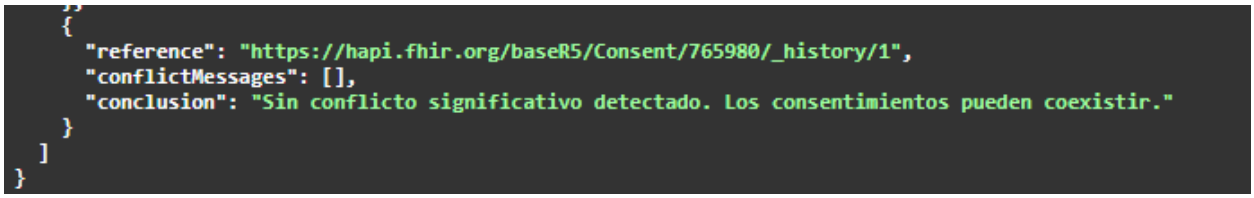


Figura 3-55. Respuesta *Consent* sin conflictos FHIR R5.

3.3.4. Pruebas extra

En este apartado, mostramos algunos resultados extra como son la recepción del correo electrónico y los logs recogidos del microservicio.

1. **Recepción de correo electrónico**

   En la Figura 3-56 podemos ver un ejemplo de correo electrónico enviado como respuesta a la solicitud de detección de conflictos para un nuevo consentimiento. En el correo podemos ver cómo se le indica al paciente los conflictos que se han detectado en relación con el consentimiento que ha introducido, detallando esos conflictos detectados.

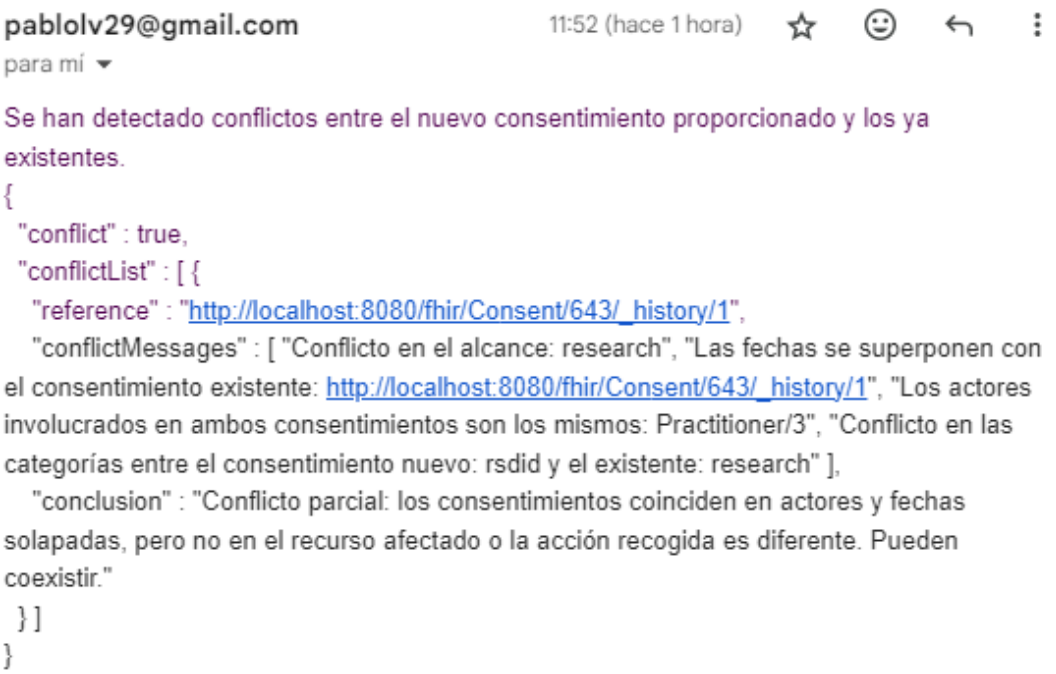


Figura 3-56. Correo electrónico recibido con conflictos.

2. Trazabilidad del sistema

En la Figura 3-57 podemos ver un extracto del log generado tras la realización de las pruebas. Este log es de especial interés para analizar el comportamiento del sistema. Podemos observar las peticiones que se han recibido y cómo se han ido tratando hasta el envío de la respuesta.

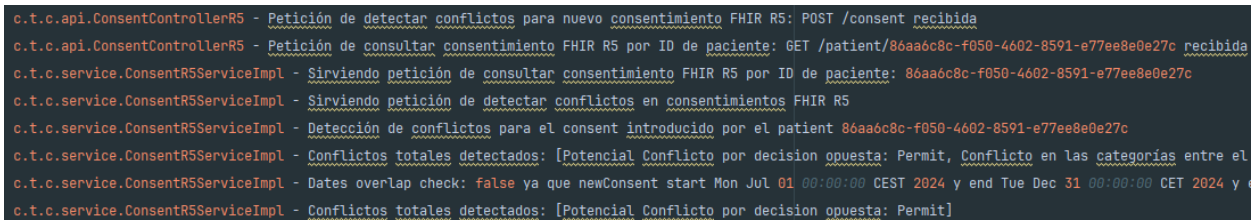


Figura 3-57. Extracto del fichero de log tras las pruebas.

4 CONCLUSIONES Y LÍNEAS FUTURAS

Llegados a este punto, después de haber ido presentando detalladamente todas las fases que ha tenido el desarrollo de este proyecto, es importante echar la vista atrás y reflexionar sobre los logros conseguidos y las lecciones que se han aprendido, así como de las posibles mejoras sistema que se ha implementado.

El objetivo principal de este proyecto ha sido el diseño e implementación de un servicio capaz de detectar conflictos en consentimientos de pacientes, que se rigen por el estándar FHIR y que deben estar alineados con lo enmarcado en la normativa europea de protección de datos GDPR. Por ello, en primer lugar, se ha realizado un estudio detallado del estándar, para comprender los recursos implicados en este proyecto antes de usarlos. Esto ha permitido que el sistema no solo cumpla con las especificaciones técnicas de los recursos del estándar, sino que también está en concordancia con lo establecido por la normativa de protección de datos europea.

Dentro de la consecución de este objetivo, destaca el trabajo realizado en la identificación, desarrollo e implementación de reglas para la detección de conflictos en los consentimientos de los pacientes. Este trabajo ha requerido estudiar gran cantidad de variables que podían conducir a distintos casos de conflictos, desde situaciones muy simples a otras más complejas. La lógica de detección ha sido implementada tanto para la versión R4 como la R5 del estándar, asegurando que el servicio sea flexible y pueda manejar las distintas posibilidades que se le planteen.

Con el desarrollo del microservicio, he conseguido ampliar conocimientos en la construcción de software, aplicando patrones arquitectónicos y tecnologías que no había usado con anterioridad. Además, este proyecto me ha permitido entender cómo se integran normativas legales y estándares dentro de soluciones técnicas, algo que es fundamental en el sector sanitario.

4.1. Debilidades y fortalezas del proyecto

Al finalizar el proyecto se ha logrado el objetivo que se planteaba al inicio, a pesar de ello se puede hacer un análisis de los puntos débiles y fuertes del servicio, lo cual nos permitirá detectar aspectos que podrían mejorarse en el futuro.

Como fortalezas, se pueden destacar la flexibilidad y adaptabilidad del servicio, siendo capaz de trabajar tanto con la versión R4 como la R5 del estándar FHIR. Esto lo convierte en un servicio robusto y preparado para los cambios que pueda sufrir el estándar con el paso del tiempo. Además, la funcionalidad principal del servicio para detectar conflictos supone un avance en los sistemas de gestión de información clínica, ya que permite identificar en una fase temprana de recepción de un nuevo consentimiento, posibles inconsistencias con los ya existentes. Por último, se puede destacar la modularidad e interoperabilidad del servicio, lo que le permite integrarse con otros servicios o sistemas ya existentes en el mercado con el objetivo de conseguir una solución completa en el ámbito de la gestión de consentimientos en los entornos sanitarios.

Por otro lado, como debilidades destacaría la escalabilidad del servicio. Si bien en un entorno controlado con un volumen de datos no muy grande se desenvuelve sin problema, en un entorno sanitario con un volumen de datos muy elevado podría saturarse y habría que explorar posibles cambios para mejorar este punto. También se debe destacar la cantidad de variables que existen a la hora de determinar si un consentimiento entra en conflicto o no con otro, lo cual ha podido conducir a que no se hayan recogido en las reglas de detección de conflictos todos los casos posibles.

4.2. Líneas de desarrollo futuras

Aunque se han conseguido los objetivos propuestos al inicio del proyecto, este podría representar el comienzo de un proyecto más grande y con más recorrido. Es por ello, que podemos destacar los puntos en los que se podría mejorar este proyecto:

1. **Eficiencia y escalabilidad**: Como hemos comentado para un volumen de datos masivo el sistema podría ser insuficiente y hacer consultas a la base de datos directamente, para recoger todos los consentimientos de un paciente, cada vez que se reciba un nuevo consentimiento puede no ser la mejor solución. Es por ello por lo que sería interesante la utilización de un base de datos en caché tipo *REDIS*.
2. **Adaptación a nuevas versiones de FHIR**: Si bien el servicio está preparado para trabajar con las versiones R4 y R5, que son las más usadas actualmente en entornos productivos, ya se está trabajando en el lanzamiento de la versión R6 del estándar. Por lo que sería interesante adaptar el servicio a los futuros cambios que pueda traer consigo esta nueva versión.
3. **Optimización de las reglas de detección**: Actualmente el servicio aplica el algoritmo de detección de conflictos con una alta efectividad para encontrar estas inconsistencias. Pero como se ha comentado, dada la alta cantidad de posibles casos que conduzcan a conflictos, es muy probable que no se hayan abordado todos los casos posibles. Asimismo, podría mejorarse la implementación en código de estas reglas para no hacer el algoritmo tan complejo y mejorar la eficiencia de este.
4. **Cambio de arquitectura**: Actualmente el servicio está pensado para ser consumido de manera síncrona mediante llamadas a alguna de sus API REST. Sin embargo, si se adoptara una arquitectura orientada a eventos, se conseguirían agilizar en gran medida los procesos en la gestión de consentimientos. El sistema con esta nueva arquitectura reaccionaría a eventos en lugar de depender de comunicaciones directas. Por ejemplo, en la creación de un nuevo consentimiento desde otro servicio, el sistema podría usar la solución de Apache Kafka para escuchar ese evento en tiempo real y agilizar la validación de ese nuevo consentimiento antes de que sea registrado completamente en el sistema de gestión de consentimientos.

REFERENCIAS

[1] Unión Europea, «Reglamento general de protección de datos,» 2022. [En línea]. Available: https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_es.html.

[2] F. J. Ruiz, «Gestor de consentimientos para el procesamiento de información sanitaria en el marco de la GDPR y conforme al estándar HL7 FHIR,» 2021. [En línea]. Available: https://idus.us.es/bitstream/handle/11441/126804/TFG-3662-RUIZ%20GOMEZ.pdf?sequence=1&isAllowed=y.

[3] Unión Europea, «Artículo 9 GDPR,» 2023. [En línea]. Available: https://www.privacy-regulation.eu/es/9.htm.

[4] Unión Europea, «Capítulo III - Derechos del interesado,» 2023. [En línea]. Available: https://www.privacy-regulation.eu/es/index.htm.

[5] HL7, «Estándar FHIR,» 2023. [En línea]. Available: https://www.hl7.org/fhir/.

[6] HL7, «Estándar FHIR R4,» 2019. [En línea]. Available: https://hl7.org/fhir/R4/index.html.

[7] HL7, «Estándar FHIR R5,» 2023. [En línea]. Available: https://hl7.org/fhir/R5/index.html.

[8] HL7, «Resource,» 2023. [En línea]. Available: https://www.hl7.org/fhir/resource.html.

[9] HL7, «Niveles de Madurez FHIR,» 2023. [En línea]. Available: https://hl7.org/fhir/versions.html#maturity.

[10] HL7, «Tipos de datos FHIR,» 2023. [En línea]. Available: https://www.hl7.org/fhir/datatypes.html.

[11] HL7, «Recurso Consent,» 2023. [En línea]. Available: https://hl7.org/fhir/consent.html.

[12] HL7, «Recurso Patient,» 2023. [En línea]. Available: https://hl7.org/fhir/patient.html.

[13] HL7, «Recurso Practitioner,» 2023. [En línea]. Available: https://hl7.org/fhir/practitioner.html.

[14] HL7, «Recurso Organization,» 2023. [En línea]. Available: https://hl7.org/fhir/organization.html.

[15] HL7, «Recurso Bundle,» 2023. [En línea]. Available: https://hl7.org/fhir/bundle.html.

[16] JetBrains, «IntelliJ IDEA,» 2024. [En línea]. Available: https://www.jetbrains.com/es-es/idea/.

[17] «Lombok,» 2024. [En línea]. Available: https://projectlombok.org/.

[18] Sonar, «SonarLint,» 2024. [En línea]. Available: https://www.sonarsource.com/products/sonarlint/.

[19] IBM, «Spring Boot,» 2024. [En línea]. Available: https://www.ibm.com/es-es/topics/java-spring-boot.

[20] The Apache Software Foundation, «Apache Maven,» 2024. [En línea]. Available: https://maven.apache.org/.

[21] Postman, «Postman,» 2024. [En línea]. Available: https://www.postman.com/.

[22] Formadores IT, «Características Postman,» 2024. [En línea]. Available: https://formadoresit.es/que-es-postman-cuales-son-sus-principales-ventajas/.

[23] Smartbear, «Swagger,» 2024. [En línea]. Available: https://swagger.io/.

[24] Docker, «Docker,» 2024. [En línea]. Available: https://www.docker.com/.

[25] HAPI FHIR, «HAPI FHIR,» 2024. [En línea]. Available: https://hapifhir.io/.

[26] HAPI FHIR, «HAPI FHIR R5 Test Server,» 2024. [En línea]. Available: https://hapi.fhir.org/home?serverId=home_r5&pretty=false&_summary=.

[27] Tutorialspoint, «Batch Script,» 2024. [En línea]. Available: https://www.tutorialspoint.com/batch_script/index.htm.

[28] HAPI FHIR, «Especificación OpenAPI FHIR R5,» 2024. [En línea]. Available: https://hapi.fhir.org/baseR5/api-docs.

[29] IETF, «RFC 4918 Sección 11.2,» 2007. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc4918#section-11.2.

[30] Atlassian, «Arquitectura de Microservicios,» 2024. [En línea]. Available: https://www.atlassian.com/es/microservices/microservices-architecture.

[31] M. Duran, «Arquitectura en capas,» 2023. [En línea]. Available: https://blog.hubspot.es/website/que-es-arquitectura-en-capas.

[32] Spring, «Anotación Spring Boot Application,» 2024. [En línea]. Available: https://docs.spring.io/spring-boot/reference/using/using-the-springbootapplication-annotation.html.

[33] Spring, «Embedded Tomcat Spring Boot,» 2024. [En línea]. Available: https://spring.io/blog/2014/03/07/deploying-spring-boot-applications.

ANEXO 1 INSTALACIÓN Y DESPLIEGUE DEL ENTORNO

En este anexo se detalla el proceso de despliegue tanto del microservicio como de la base de datos FHIR R4, así como la carga de datos en las dos bases de datos utilizadas en este proyecto.

A.1.1. Despliegue del microservicio

En primer lugar, para el despliegue del microservicio debemos crear el archivo ejecutable de extensión *.jar*. Para ello ejecutamos el comando `mvn clean install` desde el directorio padre donde se encuentra el fichero *pom.xml* del proyecto. Este comando compila el proyecto, ejecuta pruebas (en caso de haberlas), y empaqueta el código en un archivo ejecutable. Si todo ha ido correctamente, tendremos en la carpeta *target* del proyecto el fichero ejecutable.

Figura A-1. Compilación del proyecto.

Ahora, para arrancar la aplicación basta con ejecutarla con el comando `java -jar [archivo ejecutable]`

Figura A-2. Arranque de la aplicación.

En caso de despliegue en entorno productivos reales (por ejemplo, Azure Spring Apps), bastaría con aprovisionar el entorno necesario (recursos, seguridad, endpoints públicos, gestión de secretos…) y el archivo ejecutable para desplegar la aplicación.

A.1.2. Despliegue de base de datos local FHIR R4

Como se comentó en el apartado 3.2 del proyecto, se ha utilizado una imagen Docker que contiene una base de datos HAPI FHIR R4.

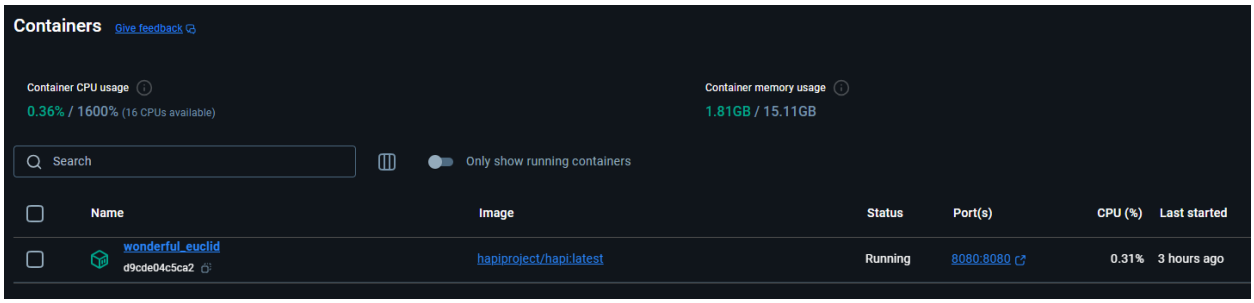


Figura A-3. Contenedor Docker con la imagen de HAPI FHIR R4.

Para automatizar el despliegue y la carga de datos iniciales en esta base de datos, se ha realizado un batch script llamado ***start_env.bat***. Este script primero arranca el contenedor y espera a que se haya iniciado completamente, después tras verificar el inicio envía los datos a la base de datos por medio de peticiones *curl* a la API REST de la base de datos.

```
setlocal enabledelayedexpansion
CHCP 65001 >nul
set "CONTAINER_ID=d9cde04c5ca2e356abfa17962540ad3300ff3ad1a9ca45cdca2154c679a049f2"
set "JSON_FILES=bundle_adam.json bundle_agustin.json"
set "URL=http://localhost:8080/fhir/metadata"
set "FHIR_URL=http://localhost:8080/fhir"
set "MAX_RETRIES=30"
set "RETRY_COUNT=0"

echo Arrancando el contenedor...
docker start %CONTAINER_ID%
if %ERRORLEVEL% neq 0 (
    echo Fallo en el arranque del Docker container.
    exit /b %ERRORLEVEL%
)

echo Esperando a que el servidor esté totalmente arrancado...

:check_server
timeout /t 5 >nul
curl --silent --fail --output nul %URL%
if %ERRORLEVEL% neq 0 (
    set /a RETRY_COUNT+=1
    if !RETRY_COUNT! geq %MAX_RETRIES% (
        echo El servidor no estuvo preparado a tiempo.
        exit /b 1
    )
    goto :check_server
)

echo Server arrancado, cargando datos...
for %%F in (%JSON_FILES%) do (
    if not exist "%%F" (
        echo El archivo JSON %%F no se encontró.
        exit /b 1
    )

    curl --location "%FHIR_URL%" ^
    --header "Content-Type: application/json" ^
    --data "@%%F"

    if %ERRORLEVEL% neq 0 (
        echo Fallo al cargar el archivo %%F.
        exit /b %ERRORLEVEL%
    )
)

echo Datos cargados correctamente.
exit /b 0
```

Figura A-4. Script *start_env.bat*.

A.1.3. Carga de consentimientos en las bases de datos FHIR R4 y R5

Para la carga de consentimientos en las bases de datos, se ha utilizado la herramienta Postman. Se han ido enviando los consentimientos, como cuerpo de peticiones POST, a los endpoint habilitados por cada una de las bases de datos para el registro de nuevos recursos.

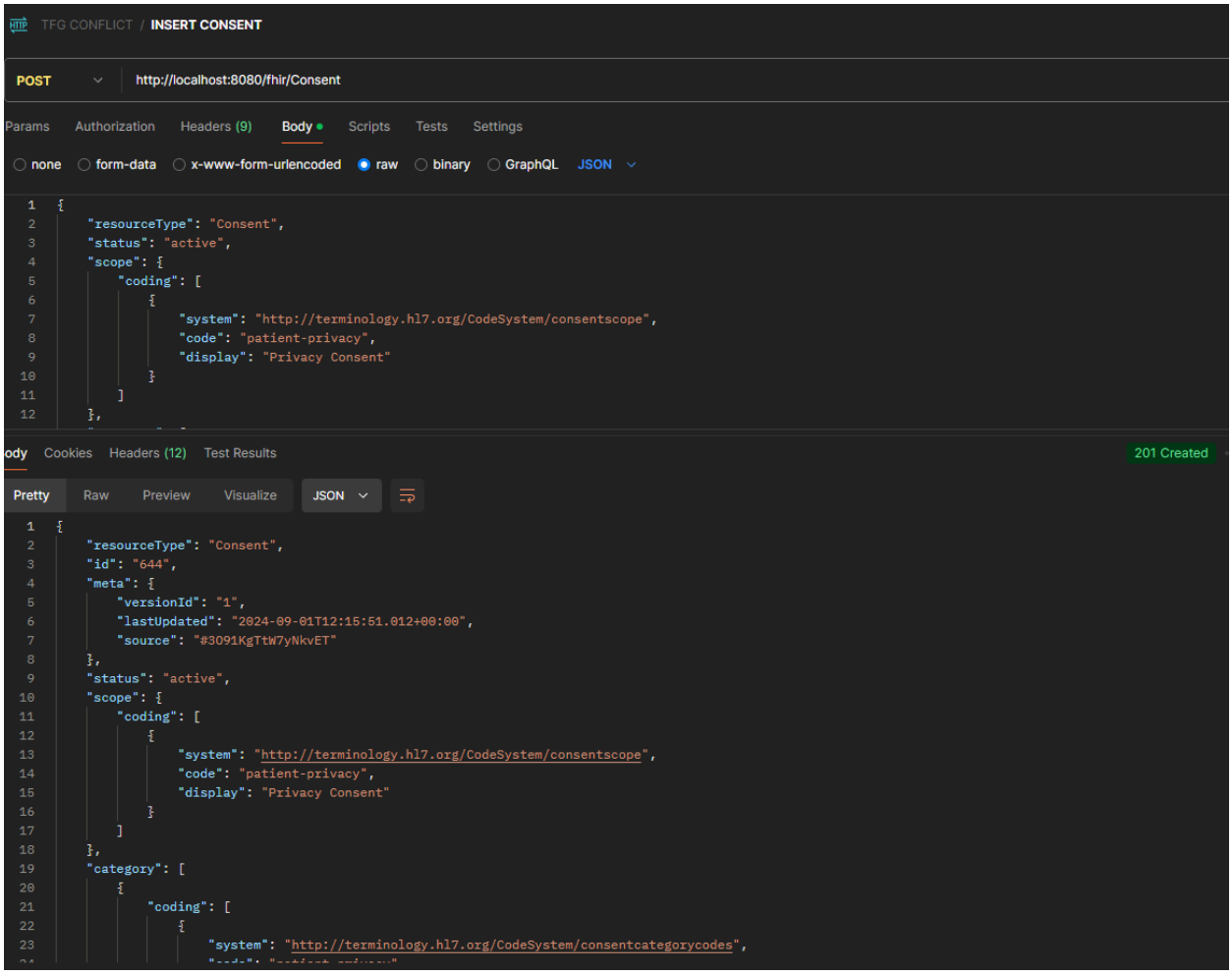


Figura A-5. Inserción de consentimiento en HAPI FHIR R4.

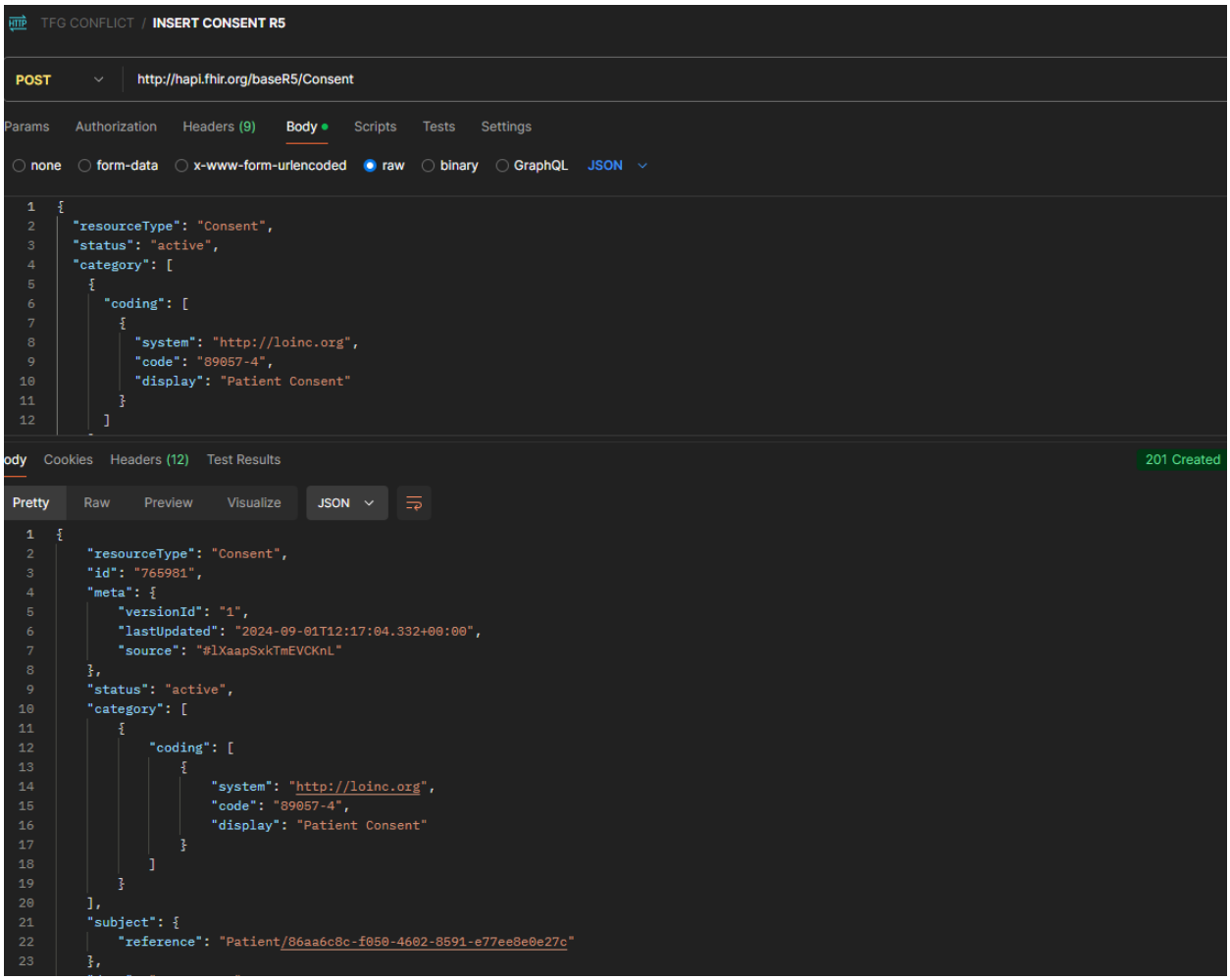


Figura A-6. Inserción de consentimiento en HAPI FHIR R5.