UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Grado en Ingeniería Informática

Trabajo de Fin de Grado (TFG)

Sistema de gestión de registros en audio de consultas médicas procesados con técnicas de inteligencia artificial

Proyecto creado por:

Isac Añor Santana

Supervisado por:

Miguel Alemán Flores

Rafael Nebot Medina

Agradecimientos

A mi familia, por su apoyo incondicional durante mis estudios y elaboración del trabajo de fin de grado, siempre han estado conmigo y solventado mis momentos duros para llegar hasta aquí.

A mis tutores, sin los cuales no hubiera sido posible la resolución de los problemas encontrados en la implementación de los objetivos planteados.

A la ULPGC, por darme la posibilidad de formarme y aprender a resolver cualquier problema que me encuentre en mi proyección profesional.

Mención especial a mi madre, que sin su ayuda no hubiese sido posible el estar aquí y ahora.

Resumen

Este proyecto consiste en el análisis, diseño e implementación de un prototipo de sistema de gestión de registros médicos. El objetivo es la creación de un prototipo de prueba de concepto que permita agilizar consultas médicas, observar resultados de dicha consulta y representar la interacción en un formato estandarizado.

El trabajo se enmarca en el proyecto OpenDx28, coordinado por el Servicio Canario de la Salud (SCS), donde el Instituto Tecnológico de Canarias (ITC) se encarga de preparar y desplegar una plataforma de servicios para el ámbito sanitario siguiendo las pautas de la ULPGC.

Se parte de un prototipo de herramienta, un núcleo, que, dado un diagnóstico verbalizado, realiza una transcripción de este. Esta transcripción es posteriormente analizada utilizando técnicas de NLP (*Natural Language Processing*) y de ella se persigue extraer y destacar información relevante, como, por ejemplo, síntomas o medicamentos. A partir de esta información extraída, se crea una historia clínica electrónica en un formato estandarizado, como, por ejemplo, HL7 ([1], [2], [3]).

El prototipo de sistema de gestión de registros será el encargado de emplear la lógica provista por la herramienta mencionada. Dado el carácter sensible de las historias clínicas electrónicas, en este prototipo se trata con ejemplos ficticios y no con información real, dando al conjunto de registros un carácter de *dataset* sobre el que mejorar la tecnología.

Abstract

This project consists in the analysis, design, and development of a medical record management system. The objective is the creation of a proof-of-concept prototype that allows speeding up medical consultations, observing results of those consultations, and representing the interaction in a standardized format.

This work is part of the OpenDx28 project, coordinated by the Canary Islands Health Service (SCS), where the Canary Islands Technological Institute (ITC) manages the preparation and deployment of a service platform for the health field following the guidelines of the ULPGC.

From a prototype tool which takes a verbalized diagnosis as the input, first, a transcription is performed. This transcript is analyzed using NLP (Natural Language Processing) techniques and we aim at extracting and highlighting relevant information, such as symptoms or medications. From this extracted information, an *electronic health record* (EHR) is created in a standardized format, such as HL7 ([1], [2], [3]).

The prototype of the record management system will use the logic provided by the tool. Given the sensitive nature of electronic health records, this prototype will use examples and not with truthful information. The set of records produced can serve as a data set, which could be used to improve the technology.

Índice general

1 Introducción 1
1.1 Marco del proyecto 1
1.2 Motivación 1
1.3 Objetivos 2
1.4 Recursos utilizados 3
1.4.1 Recursos hardware 3
1.4.2 Recursos software 3
1.5 Competencias específicas cubiertas 4
1.5.1 Competencias específicas: formación básica 4
1.5.2 Competencias específicas: comunes a la rama de informática 4
1.5.3 Competencias específicas: tecnología específica – tecnologías de la información 4
1.6 Aportaciones 5
1.7 Estructura del documento 5
2 Análisis 6
2.1 Estado del Arte 6
2.1.1 Estándares 6
2.1.2 Normativas 6
2.1.3 Herramientas de procesamiento de lenguaje natural 7
2.1.4 Empresas con servicios similares 7
2.2 Metodologías de diseño y desarrollo 8
2.2.1 SCRUM 8
2.2.2 Adaptación de SCRUM al proyecto 10
2.3 Objetivos del desarrollo y contextualización detallada 11
2.4 Análisis de requisitos 12
2.4.1 Usuarios de la aplicación 12
2.4.2 Requisitos funcionales 12
2.4.3 Requisitos no funcionales 13
2.5 Historias de usuario 13
2.6 Casos de uso 14
2.7 Tecnologías estudiadas y finalmente seleccionadas/utilizadas 15
2.8 Identificación de riesgos y mitigaciones 16
2.9 Aspectos económicos 17
2.9.1 Plan de negocio 17

2.9.2 Evaluación de costes de alojamiento en la nube de Amazon 17
3 Diseño 19
3.1 Arquitectura del sistema 19
3.2 Aplicativo cliente 20
3.3 Aplicativo servidor 20
3.3.1 Servidor web 20
3.3.2 Backend 20
3.4 Diagramas 21
3.4.1 Arquitectura 21
3.4.2 Backend 21
3.5 Persistencia de las entidades del sistema 23
3.6 Diseño de las baterías de test 23
4 Implementación 24
4.1 Configuración del entorno de desarrollo usando contenedores 24
4.1.1 Descripción de los contenedores usados 24
4.1.2 Gestión de los contenedores y despliegue 25
4.2 Desarrollo del aplicativo Frontend 27
4.2.1 Configuración y puesta en marcha 27
4.2.2 Componentes 30
4.2.3 Servicios 33
4.3 Desarrollo del aplicativo Backend 35
4.3.1 Sobre Python y entornos virtuales 35
4.3.2 Configuración, Flask y puesta en marcha 37
4.3.3 Creación de estrategias 45
4.3.4 Creación de registros 49
4.4 Conexión Frontend-Backend 55
4.4.1 Sesiones y cookies 57
4.5 Baterías de test 58
4.5.1 Uso de pytest 58
4.5.2 Fixtures 59
4.5.3 Ejecución de los test 60
5 Resultados 61
5.1 Ejemplos del sistema trabajando 61
5.1.1 Creación de pipelines 61
5.1.2 Vista y modificación de pipelines 64
5.1.3 Vista y modificación de registros 68

5.1.4 Vista y modificación de estrategias. 70
5.2 Documentación generada (Funcional y Técnica) 72
6 Conclusiones, trabajos futuros y perspectiva 73
6.1 Conclusiones 73
6.2 Trabajos futuros 74
6.2.1 Aspectos relacionados con la seguridad 74
6.2.2 Paso a producción 75
Glosario de acrónimos 76
7 Bibliografía 77
Anexo 82

Índice de figuras

Ilustración 2.6-1 - Diagrama de casos de uso. La sigla CRUD hace referencia a Create, Read, Update and Delete. 14
Ilustración 3.4-1 - Representación simplificada del modelo cliente-servidor empleado. 21
Ilustración 3.4-2 - Patrón de diseño strategy. 21
Ilustración 3.4-3 - Patrón de diseño strategy adaptado. 22
Ilustración 3.5-1 - Entidades del sistema. 23
Ilustración 4.1-1 - Organización de contenedores en Docker Desktop. 26
Ilustración 4.2-1 - Vista del IDE Visual Studio Code abierto en el directorio raíz del contenedor "tfg-frontend" 27
Ilustración 4.2-2 - Puesta en marcha del servidor de desarrollo de Angular. 29
Ilustración 4.2-3 - Página de vista de registros. 30
Ilustración 4.3-1 – Contenido de un entorno virtual de Python. 35
Ilustración 4.3-2 - Activación de un entorno virtual a través de la línea de comandos. 36
Ilustración 4.3-3 - Estructura de directorios de la aplicación Flask. 40
Ilustración 4.3-4 - Puesta en marcha del aplicativo en modo desarrollo. 44
Ilustración 4.3-5 - Formulario de relleno de información relacionada con una nueva estrategia. 45
Ilustración 4.3-6 - Subida de los ficheros que definen la estrategia. 45
Ilustración 4.3-7 - Ambos ficheros subidos. 46
Ilustración 4.3-8 - Resumen de datos introducidos en la creación de estrategias. 46
Ilustración 4.3-9 - Espera durante el proceso de creación de estrategias. 47
Ilustración 4.3-10 - Éxito en la creación de estrategias. 47
Ilustración 4.3-11 - Pantalla de creación de registros a partir de audio. 49
Ilustración 4.3-12 - Solicitud de permisos del micrófono por parte del navegador. 49
Ilustración 4.3-13 - Finalización de la grabación. 50
Ilustración 4.3-14 - Selección del pipeline. 50
Ilustración 4.3-15 – Resumen de lo seleccionado/introducido y comienzo de procesamiento. 50
Ilustración 4.3-16 - Resultado exitoso en la creación del registro. 51
Ilustración 4.3-17 - Vista del registro creado presentada bajo el resultado. 51
Ilustración 4.3-18 - Selección del registro padre en la creación a partir de otro registro. 52
Ilustración 4.3-19 - Selección del pipeline en la creación a partir de otro registro. 52
Ilustración 4.3-20 - Selección de la salida de procesamiento que se utilizará como entrada y etapa del pipeline a partir de la que realizar el procesamiento. 53
Ilustración 4.3-21 - Resumen de la información seleccionada. 53
Ilustración 4.4-1 - Cookie de sesión almacenada en el navegador. 57
Ilustración 4.4-2 - Expiración de la cookie de sesión al cerrar sesión. 57
Ilustración 4.5-1 - Estructura del directorio de pruebas. 58
Ilustración 4.5-2 - Resultado exitoso de la ejecución de pruebas. 60
Ilustración 5.1-1 - Etapas implicadas en la creación de un pipeline. 61
Ilustración 5.1-2 - Relleno del formulario de información acerca de un pipeline. 61
Ilustración 5.1-3 - Selección de una estrategia de tipo "Voz a texto". 62
Ilustración 5.1-4 - Selección de una estrategia de tipo "Intermedia" o "Final". 62
Ilustración 5.1-5 - Finalización de la selección de estrategias que conforman el pipeline. 63
Ilustración 5.1-6 - Resumen de lo creado. 63
Ilustración 5.1-7 - Creación correcta del pipeline. 64
Ilustración 5.1-8 - Tabla de pipelines. 64
Ilustración 5.1-9 - Tabla de pipelines, mostrando un máximo de 10. 65

Ilustración 5.1-10 - Vista de la información detallada de un pipeline. 65
Ilustración 5.1-11 - Vista de un pipeline con un mensaje de advertencia. 66
Ilustración 5.1-12 - Pestaña de modificación de pipeline. 66
Ilustración 5.1-13 - Modificación exitosa de un pipeline. 67
Ilustración 5.1-14 - Cambios reflejados tras la modificación del pipeline 67
Ilustración 5.1-15 - Tabla de registros médicos. 68
Ilustración 5.1-16 - Vista de la historia, transcripción y audio de un registro. 68
Ilustración 5.1-17 - Vista del reconocimiento de la entidad nombrada realizada sobre un registro. .. 69
Ilustración 5.1-18 - Modificación de registros exitosa. 69
Ilustración 5.1-19 - Mensaje de modificación de un registro. 70
Ilustración 5.1-20 - Modificación de la información relativa a una estrategia. 70
Ilustración 5.1-21 - Pantalla de subida del fichero ".py". 71
Ilustración 5.1-22 - Subida del fichero. 71
Ilustración 5.1-23 - Resultado de la actualización del fichero en el backend. 71

Índice de cuadros

Cuadro 1 - Roles de Usuario. 12
Cuadro 2 - HU 01. 82
Cuadro 3 - HU 02. 82
Cuadro 4 - HU 03. 82
Cuadro 5 - HU 04. 83
Cuadro 6 - HU 05. 83
Cuadro 7 - HU 06. 83
Cuadro 8 - HU 07. 84
Cuadro 9 - HU 08. 84
Cuadro 10 - HU 09. 84
Cuadro 11 - HU 10. 85
Cuadro 12 - HU 11. 85
Cuadro 13 - HU 12. 85
Cuadro 14 - HU 13. 86
Cuadro 15 - HU 14. 86
Cuadro 16 - HU 15. 86
Cuadro 17 - HU 16. 87
Cuadro 18 - HU 17. 87
Cuadro 19 - HU 18. 87
Cuadro 20 - HU 19. 88
Cuadro 21 - HU 20. 88
Cuadro 22 - HU 21. 88
Cuadro 23 - HU 22. 89
Cuadro 24 - HU 23. 89
Cuadro 25 - HU 24. 89

Índice de algoritmos y extractos de código

Extracto 4.1-1 - Contenido del fichero "docker-compose.yml". 25
Extracto 4.2-1 - Componente HTML de la página de vista de registros. 31
Extracto 4.2-2 - Decoradores de entrada y salida en el componente de modificación de registros. 32
Extracto 4.2-3 - Parte de la colección de rutas definidas. 32
Extracto 4.2-4 - Servicio de variables globales. 33
Extracto 4.2-5 - Suscripción al valor de una variable perteneciente al servicio de variables globales. 34
Extracto 4.2-6 - Actualización del valor de una variable perteneciente al servicio de variables globales. 34
Extracto 4.3-1 - Fichero de configuración de la aplicación Flask. 38
Extracto 4.3-2 - Inicialización y configuración de la aplicación Flask. 39
Extracto 4.3-3 - Contenido del fichero “/opt/40991-TFG-Backend/.env”. 39
Extracto 4.3-4 - Parte del fichero "views.py" de la entidad pipeline. 40
Extracto 4.3-5 - Ejemplo de registro de Blueprints en la aplicación Flask. 41
Extracto 4.3-6 - Ruta y lógica implicada en la eliminación de un pipeline. 41
Extracto 4.3-7 - Contenido del fichero "models.py" relativo a la entidad Pipeline. 42
Extracto 4.3-8 - Contenido del fichero "/bootstrap.bash" 43
Extracto 4.3-9 - Creación e instalación de dependencias en un entorno virtual de una nueva estrategia. 48
Extracto 4.3-10 - Ejecución de una estrategia de manera individual. 54
Extracto 4.4-1 - Variable de entorno en el frontend con la URL del backend. 55
Extracto 4.4-2 - Uso de un servicio por parte de un componente. 55
Extracto 4.4-3 - Código relativo a un servicio de comunicación con el backend. 56
Extracto 4.5-1 - Declaración de un fixture a nivel de módulo y ejemplo de prueba. 59
Extracto 4.5-2 - Ejecución del fixture a nivel de módulo. 60

Capítulo 1

1 Introducción

1.1 Marco del proyecto

El presente se enmarca en el proyecto OpenDx28, coordinado por el Servicio Canario de la Salud (SCS), donde el Instituto Tecnológico de Canarias (ITC) se encarga de preparar y desplegar una plataforma de servicios para el ámbito sanitario siguiendo las pautas de la ULPGC.

Este trabajo de fin de grado (TFG) consiste en la realización del análisis, diseño y desarrollo de un prototipo de sistema de gestión de registros médicos procesados con inteligencia artificial. Se resalta el hecho de tratarse de una prueba de concepto, es decir, una implementación de una idea con el propósito de verificar que el concepto es susceptible de ser explotado de manera útil [4].

El sistema será una herramienta dirigida a personal sanitario y científicos de datos. El personal sanitario podrá, entre otras cosas, crear registros a través de una grabación. El usuario científico de datos, además de lo que puede hacer el personal sanitario, puede agregar nuevas formas de procesar la información relativa a los registros, así como gestionar las rutinas ya existentes.

1.2 Motivación

La gestión de registros médicos es un proceso costoso con muchas restricciones, entre otras cosas, por el carácter sensible de la información con la que se trata y la estricta normativa que la protege. En consecuencia, surgen problemas al plantear decisiones de uso y almacenamiento de datos personales cuyo conocimiento o desconocimiento puede tener un impacto considerable en la vida de un individuo.

A nivel práctico, muchas organizaciones sanitarias, tanto privadas como públicas, poseen su forma única de gestionar registros y fichas de pacientes. En muchas ocasiones estas entidades pueden ser vistas como islas de información, donde las comunicaciones no son abundantes y están restringidas. Fallos a la hora de solicitar información por usar registros con estructuras, distintas, dando lugar a retrasos en la comprensión y empleo de la información pueden ser fatales.

Con la finalidad de proveer de alternativas y soluciones surge la necesidad de una herramienta capaz de agilizar la toma de registros, su procesamiento, almacenamiento y portabilidad. La motivación principal del trabajo es proponer un prototipo que demuestre que una solución puede ser alcanzada con las capacidades tecnológicas actuales y puede ser llevada más allá con el desarrollo agresivo de nuevas técnicas en los ámbitos de inteligencia artificial y procesamiento del lenguaje natural, dando acceso gratuito a herramientas muy poderosas y eficaces.

1.3 Objetivos

Los objetivos del trabajo son los siguientes:

- Diseño y desarrollo de un prototipo de sistema de gestión de la información correspondiente a la grabación, su transcripción y los datos extraídos de ésta.
- Facilitar y agilizar la interacción con computadores por parte del personal a la hora de crear registros.
- Procesar las transcripciones para crear historias clínicas electrónicas en formatos estandarizados.
- Diseño de una interfaz de usuario que permita interactuar con el sistema desde el punto de vista de varios roles de usuario.
- Diseño de una serie de vistas incluidas en la interfaz de usuario para visualizar el procesamiento paso por paso realizado a un registro.
- Evaluación y estudio de una posible integración en la nube del sistema.

1.4 Recursos utilizados

1.4.1 Recursos hardware

El único recurso hardware empleado es un ordenador personal sobremesa.

1.4.2 Recursos software

1.4.2.1 Frontend

La herramienta software empleada en el desarrollo del Frontend es Angular, un framework para aplicaciones web, desarrollado en TypeScript, de código abierto, mantenido por Google [5], [6].

Las razones principales detrás de la elección de este framework se centran en una experiencia previa de trabajo con éste y la fácil implementación y empleo de la librería Angular Material [7], [6]. Angular Material es una librería de componentes de interfaz de usuario que implementa Material Design en Angular [8]. Angular Material simplifica en gran medida la creación de la interfaz de usuario a través de componentes ya hechos y soluciones a problemas comunes.

1.4.2.2 Backend

El lenguaje de programación empleado en el Backend es Python. Más concretamente, se ha hecho uso del microframework Flask [9], [10]. Es denominado microframework o framework minimalista porque no necesita herramientas o librerías adicionales y, a diferencia de un marco de aplicación completo, carece de la mayor parte de la funcionalidad de abstracción de amplio espectro, enfocándose exclusivamente en el manejo de solicitudes HTTP. No incorpora un sistema de usuarios, abstracción del manejo de bases de datos, por ejemplo, a través de un sistema de mapeo relacional de objetos; funciones para la validación y saneamiento de datos o la abstracción de la vista mediante un motor de plantillas web [11], [12]. Estas funcionalidades son provistas por extensiones adicionales, dando más control acerca de lo que se usa y lo que hay instalado, con la finalidad de mantener la aplicación lo más limpia y minimalista posible.

Adicionalmente, en el Backend se hace uso de distintas herramientas software en las distintas etapas de procesamiento de los registros. Para realizar las transcripciones, se emplea Whisper [13] como *estrategia* por defecto, aunque se podrían emplear otras herramientas S2T (*Speech To Text*), registradas e implementadas como otras *estrategias*. El uso de *estrategias* en las distintas etapas, así como los recursos software de terceros empleados se cubren con más detenimiento en el tercer y cuarto capítulos del documento.

1.4.2.3 Plataforma de ejecución

La aplicación se ejecuta en contenedores de Docker [14], [15]. La arquitectura del sistema y contenedores utilizados se cubre en capítulos posteriores del documento. La principal motivación para emplear Docker es la alta portabilidad provista, aislamiento entre contenedores y eficiencia de trabajo en distintos entornos.

1.4.2.4 Integrated Development Environment (IDE)

El entorno de desarrollo empleado es Visual Studio Code [16], [17]. Es con diferencia uno de los entornos de desarrollo más utilizados por desarrolladores y cuenta con todo tipo de extensiones para distintos casos de uso. De las extensiones empleadas, destaca DevContainers [18], una extensión que permite “adherir” el entorno de desarrollo al contenedor, simulando la ejecución de éste dentro del contenedor, permitiendo inspeccionar los contenidos del contenedor, modificar ficheros, abrir un terminal, e instalar extensiones específicas para el contendor. Esto es especialmente útil porque permite el uso del contenedor como plataforma de desarrollo, sin necesidad de tener el código fuente en un directorio del computador anfitrión y montarlo como volumen cada vez que arranca el contenedor.

1.5 Competencias específicas cubiertas

En este apartado se indica cómo se han cubierto las competencias específicas relacionadas con el TFG.

1.5.1 Competencias específicas: formación básica

1.5.1.1 Competencias

FB4: *Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con la aplicación de la ingeniería*

FB5: *Conocimiento de la estructura, organización, funcionamiento e interconexión de los sistemas informáticos, los fundamentos de la programación y su aplicación para la resolución de problemas propios de la ingeniería*.

1.5.1.2 Justificación

Sobre la competencia FB4, quedan reflejados todos los conocimientos básicos sobre el uso y programación a través del desarrollo de un sistema de gestión que involucra el uso de diversas tecnologías informáticas como plataformas de desarrollo y distintos frameworks y la implementación de funcionalidades informáticas para tareas de procesamiento específico como puede ser la transcripción de archivos de audio o procesamiento del lenguaje natural.

Con respecto a la competencia FB5, la interconexión de sistemas informáticos se ve reflejada en el empleo de múltiples contenedores virtuales que se comunican entre sí, la programación de estos y su diseño y estructura para resolver diversos problemas relacionados con el proyecto, como puede ser la portabilidad de este.

1.5.2 Competencias específicas: comunes a la rama de informática

1.5.2.1 Competencias

CI12: *Conocimiento y aplicación de las características, funcionalidades y estructura de las bases de datos, que permitan su adecuado uso y el diseño y el análisis e implementación de aplicaciones basadas en ellos*.

CI13: *Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los sistemas de información, incluidos los basados en web*.

1.5.2.2 Justificación

La cobertura de ambas competencias se ve reflejado en el empleo de un contenedor virtual que contiene únicamente una base de datos con la finalidad de dar persistencia al sistema. La base de datos concreta y su configuración ha sido seleccionada para cumplir con los requisitos del sistema de gestión a desarrollar. La aplicación de esta herramienta de almacenamiento ha sido puesta en marcha en un sistema de información basado en web.

1.5.3 Competencias específicas: tecnología específica – tecnologías de la información

1.5.3.1 Competencia

TI6: *Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil*.

1.5.3.2 Justificación

El sistema concebido es una aplicación web, donde se hace uso de archivos multimedia (los diagnósticos verbalizados) y la forma de interacción con el aplicativo es a través de páginas web servidas a un navegador.

1.6 Aportaciones

El trabajo de fin de grado aporta una prueba de concepto, es decir, una implementación resumida de una idea con el propósito de verificar que ésta se susceptible de ser explotada de manera útil. En otras palabras, aporta un prototipo operativo que puede tener un impacto muy positivo en el ámbito sanitario donde, a nivel técnico, se plantea un sistema de ejecución de etapas de procesamiento intercambiables y extensible.

1.7 Estructura del documento

En el segundo capítulo se realiza un análisis del sistema a desarrollar. Sirve como preludio al diseño de este, donde se exploran herramientas similares y punteras en la sección de estado del arte; se realiza un análisis de requisitos, se exploran riesgos y se tratan aspectos económicos y temporales.

En el tercer capítulo se realiza el diseño de la aplicación, indispensable para posteriormente desarrollar el proyecto.

En el cuarto capítulo se trata la implementación, subdividida en las partes más importantes desde un punto de vista de abstracción superior, como puede ser la implementación del Frontend, Backend, conexión entre el Frontend y Backend, uso de contenedores…

En el quinto capítulo se exponen los resultados de la implementación, con ejemplos del sistema trabajando y documentación generada.

Por último, en el sexto capítulo se disertan las conclusiones del trabajo, así como posibles ampliaciones.

Capítulo 2

2 Análisis

2.1 Estado del Arte

2.1.1 Estándares

Entre los estándares se destaca y se empleará HL7 (*Health Level 7*). Es un estándar internacional de interoperabilidad de sistemas de información sanitaria utilizado para el intercambio electrónico de datos clínicos y administrativos entre diferentes aplicaciones y sistemas de salud, como sistemas de información hospitalaria, sistemas de información de laboratorio y sistemas de registros médicos electrónicos.

El estándar HL7 define una serie de normas y especificaciones técnicas para la estructura, formato y contenido de los mensajes. Utiliza un enfoque basado en mensajes, donde cada mensaje está compuesto por segmentos y cada segmento de campos con posibles subcampos. Es un estándar configurable que permite adaptarse a diferentes contextos y requisitos locales. Además, el estándar se actualiza periódicamente para mantenerse al día con los avances tecnológicos y las necesidades cambiantes en el campo de la salud.

Adicionalmente, existen otros estándares más especializados, como DICOM (*Digital Imaging and Communications in Medicine*), utilizado principalmente para el intercambio de imágenes médicas como radiografías, tomografías y resonancias magnéticas entre otros.

2.1.2 Normativas

Con respecto a las normativas, es de vital importancia mencionar al Reglamento General de Protección de Datos (RGPD). El RGPD es una normativa de la Unión Europea (UR) que establece las reglas y principios para la protección de los datos personales de los ciudadanos europeos.

Tiene como objetivo principal garantizar la protección de la privacidad y los derechos de las personas en relación con el procesamiento de sus datos personales por parte de organizaciones y empresas.

Otorga una serie de derechos a los titulares de los datos, como el derecho de acceso, rectificación, supresión, portabilidad, limitación del tratamiento y oposición al tratamiento de los datos personales

Las organizaciones y empresas que procesan datos personales deben cumplir con las obligaciones establecidas por el RGPD, incluyendo la obtención del consentimiento del titular de los datos cuando sea necesario. El incumplimiento del RPD puede resultar en sanciones económicas significativas, que pueden alcanzar hasta el 4% de la facturación anual de una empresa o 20 millones de euros, según la cuantía que sea mayor. [19] [20]

Existe otra normativa vigente, HIPAA (*Health Insurance Portability and Accountability Act* o Ley de Portabilidad y Responsabilidad del Seguro Médico), que viene a ser la contraparte estadounidense. HIPAA establece normas y regulaciones para la protección de la información sanitaria, que se aplica a los proveedores de servicios de atención médica, planes de seguro médico, procesadores de datos y otras entidades que manejan información médica protegida.

La ley busca asegurar que la información médica de los pacientes sea utilizada y divulgada de manera adecuada y que mantenga su confidencialidad e integridad. El incumplimiento de la ley puede resultar en sanciones civiles y penales.

2.1.3 Herramientas de procesamiento de lenguaje natural

Las librerías más comunes que implementan técnicas de procesamiento de lenguaje natural son Spark NLP y Spacy. Ambas ofrecen funcionalidades como análisis de texto, clasificación de texto, reconocimiento de la entidad nombrada y relación entre entidades.

Spark NLP se integra con Apache Spark, permitiendo trabajar con grandes volúmenes de datos textuales en entornos distribuidos aprovechando las rutinas que explotan paralelismo ya implementadas por Apache Spark. Es desarrollada y mantenida por John Snow Labs y tiene funcionalidad gratuita y de pago. [21] [22]

Spacy es una librería popular escrita en Python conocida por su facilidad de uso, rendimiento e integración con otras librerías y herramientas de Python. [23] [24]

Además de librerías, distintas empresas que ofrecen servicios en la nube, como Microsoft o Google, también ofrecen acceso de pago a unas API (*Application Programming Interface*) propietarias que realizan funciones de procesamiento del lenguaje natural. Proveen herramientas que realizan procesamiento a nivel general (cualquier texto), como específico, es decir, procesamiento del lenguaje natural orientado a un ámbito sanitario. Para usar estas herramientas es necesario darse de alta en estos servicios y pagar una subscripción. [25] [26].

Otra alternativa puntera consiste en emplear modelos ya entrenados disponibles en Hugging Face [27] [28]. Hugging Face es una compañía estadounidense que desarrolla herramientas para construir aplicaciones empleando aprendizaje automático. Es conocida por librería de transformadores, construida para aplicaciones que realicen procesamiento del lenguaje natural. Esto ha dado lugar a la creación de una plataforma donde usuarios comparten conjuntos de datos y modelos ya entrenados que hagan funciones específicas de procesamiento del lenguaje natural, como, por ejemplo, detección de entidades relevantes en un entorno sanitario [29].

2.1.4 Empresas con servicios similares

Dos empresas con servicios relacionados con la gestión de registros médicos electrónicos son Meditech [30] y Athenahealth [31]. Ambas son empresas con sede en Estados Unidos que proporcionan soluciones de software para mejorar la eficiencia en el sector de la atención médica, tanto a través de herramientas de administración de citas y registros, como de facturación, codificación y cumplimiento normativo.

Es importante destacar el que el contexto sanitario de operación de estas empresas es el estadounidense, donde hospitales y clínicas son a menudo privadas. La interacción de hospitales y clínicas con las empresas mencionadas no es distinta de un caso habitual de subcontratación de servicios, por ejemplo, El Corte Inglés subcontratando su infraestructura de seguridad a una empresa especializada.

2.2 Metodologías de diseño y desarrollo

La palabra metodología es definida por la RAE como el "*Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal*" [32]. Adaptada a este contexto, puede verse como el conjunto de métodos técnicas y herramientas utilizadas para realizar una tarea o proyecto.

Aplicado al desarrollo de software, se puede entender como una pauta a seguir a través de todas las fases del proyecto (análisis, diseño, desarrollo, pruebas y mantenimiento) con el fin de satisfacer las necesidades del cliente y entregar el mejor producto en el menor tiempo posible. La ausencia total de una metodología puede dar lugar a proyectos desorganizados difíciles de continuar, finalizar, probar y mantener. Por norma general, es altamente recomendable e intuitivo proceder siguiendo una seria de pasos estandarizados, probados y documentados para mantener orden, estructura y ritmo en el desarrollo de un proyecto.

La metodología empleada es SCRUM. Una metodología ágil muy popular, pulida a lo largo del grado.

2.2.1 SCRUM

SCRUM es una metodología de desarrollo incremental, que basa la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados y solapa las distintas fases del desarrollo, en lugar de llevarlas a cabo de manera secuencial. [33] [34] [35].

El marco técnico de scrum está formado por [36]:

- Roles
    - El equipo de desarrollo
    - El dueño del producto
    - El Scrum Master
- Artefactos
    - Pila del producto
    - Pila del sprint
    - Incremento
- Eventos
    - Sprint
    - Reunión de planificación del sprint
    - Scrum diario
    - Revisión del sprint
    - Retrospectiva del sprint

Donde la pieza clave es el sprint. Se denomina sprint a cada ciclo o iteración de trabajo que produce una parte del producto terminada y funcionalmente operativa (incremento). Al emplear el marco técnico de scrum, el sprint está basado en pulsos de tiempo prefijado, empleando pues, un incremento iterativo para mantener un ritmo de avance constante.

A continuación, se definen brevemente los componentes mencionados.

2.2.1.1 Artefactos

- **Pila del producto** (*product backlog*): lista de requisitos de usuario, que a partir de la visión inicial de producto crece y evoluciona durante el desarrollo.
- **Pila del sprint** (*sprint backlog*): lista de los trabajos que debe realizar el equipo de desarrollo durante el sprint para generar el incremento previsto.
- **Incremento**: resultado de cada sprint.

2.2.1.2 Eventos

- **Sprint**: nombre que recibe cada iteración de desarrollo. Es el núcleo central que genera el pulso de avance a ritmo de "tiempos prefijados" (*time boxing*)
- **Reunión de Planificación del sprint**: reunión de trabajo que marca el inicio de cada sprint en la que se determina cuál es el objetivo del sprint y las tareas necesarias para conseguirlo.
- **Scrum diario**: breve reunión diaria del equipo, en la que cada miembro responde a tres cuestiones:
    - El trabajo realizado el día anterior.
    - El que tiene previsto realizar.
    - Cosas que puede necesitar, o impedimentos que deben eliminarse para poder realizar el trabajo.
- **Revisión del sprint**: análisis e inspección del incremento generado y adaptación de la pila del producto si resulta necesario.
- **Retrospectiva del sprint**: revisión de lo sucedido durante el Sprint. Reunión en la que el equipo analiza aspectos operativos de la forma de trabajo y crea un plan de mejoras para aplicar al próximo sprint.

2.2.1.3 Roles

- **Propietario del producto**: es la persona responsable de lograr el mayor valor de producto para los clientes, usuarios y resto de implicados.
- **Equipo de desarrollo**: grupo o grupos de trabajo que desarrollan el producto.
- **Scrum Master**: es el responsable del cumplimiento de las reglas de un marco de scrum técnico, asegurando que se entienden en la organización y se trabaja conforme a ellas.

2.2.2 Adaptación de SCRUM al proyecto

Al ser un trabajo tutorizado por personal del ITC, se definen los roles de la siguiente manera: el estudiante, Isac Añor Santana como equipo de desarrollo; el tutor académico, Miguel Alemán Flores, como Scrum Master; y los cotutores de empresa, Rafael Nebot Medina y Paula Moreno Fajardo, como dueños del producto.

Se define una duración del sprint de tres semanas. Con esta periodicidad a lo largo del desarrollo del proyecto se lleva a cabo una reunión que sirve revisión del sprint y reunión de planificación del sprint. En otras palabras, cada tres semanas se muestra el incremento generado y se comunican y acuerdan las funcionalidades a desarrollar para el sprint siguiente.

Al tratarse de un equipo de desarrollo formado por un individuo, se omite el evento de scrum diario.

2.3 Objetivos del desarrollo y contextualización detallada

En este apartado se detalla una serie de objetivos concretos de implementación del trabajo, que parten de los objetivos globales mencionados anteriormente. Los objetivos consisten principalmente en:

- Diseño de un sitio web intuitivo que permita hacer uso de las funcionalidades previstas.
- Desarrollo de un Backend que permita gestionar las solicitudes obtenidas desde el sitio web y gestionar e implementar las funcionalidades.
- Diseño de una arquitectura de la aplicación fácilmente replicable, estándar y portable.
- Implementar un algoritmo que permita usar herramientas o librerías adicionales de forma encapsulada y controlada.
- Implementación de una solución que permita dar persistencia al aplicativo de forma eficiente.

El proyecto se realiza en un Trabajo de Fin de Grado, con restricción temporal de trescientas horas y económica, donde se prefiere el uso de tecnologías gratuitas de código abierto. Los recursos empleados en el desarrollo del proyecto consisten únicamente de un computador personal sobremesa.

2.4 Análisis de requisitos

2.4.1 Usuarios de la aplicación

Una de las primeras cuestiones a la hora de realizar el análisis es la pregunta de ¿quién va a usar la aplicación? En consecuencia, se distinguen tres roles de usuario: personal sanitario, personal con un perfil de científico de datos o desarrollador y un rol de administrador. La pertenencia a un rol limita la cantidad de funcionalidades disponibles.

Cuadro 1 - Roles de Usuario.

Personal sanitario	**Los usuarios con este perfil sólo acceder a las funcionalidades directamente relacionadas con las historias clínicas electrónicas**
Científico de datos/desarrollador	**Además de que puede hacer un usuario con rol de personal sanitario, puede acceder a las funcionalidades relacionadas con el procesamiento de la información.**
Administrador	**El administrador puede hacer lo que el resto de usuarios y además tiene acceso a funcionalidades de gestión de usuarios.**

A continuación, se realiza un análisis de requisitos, donde se detallan las funcionalidades desde el punto de vista de los roles. Es importante comprender el contenido de la tabla, puesto que a continuación se presentará una lista de requisitos funcionales, donde aquellos en los que se explicita el rol de "personal sanitario" debe entenderse que tanto "científicos de datos" y "administrador" también podrán acceder o hacer uso de la funcionalidad asociada al requisito.

2.4.2 Requisitos funcionales

2.4.2.1 Personal sanitario

RF 1. Un profesional sanitario podrá grabar un audio relativo a una consulta.
RF 2. Un profesional sanitario podrá seleccionar las etapas que se emplearán en el procesamiento del audio.
RF 3. Un profesional sanitario podrá crear un registro a partir del audio grabado y el conjunto de etapas seleccionado.
RF 4. Un profesional sanitario podrá ver una lista de los registros existentes.
RF 5. Un profesional sanitario podrá ver un registro concreto en detalle.
RF 6. Un profesional sanitario podrá modificar un registro concreto.
RF 7. Un profesional sanitario podrá eliminar un registro.
RF 8. Un profesional sanitario podrá seleccionar la salida de alguna etapa de procesamiento empleada en la creación de un registro.
RF 9. Un profesional sanitario podrá seleccionar otra etapa de procesamiento de un conjunto de etapas de procesamiento registrado.
RF 10. Un profesional sanitario podrá a crear un registro nuevo a partir de la salida y la etapa previamente seleccionadas, es decir, crear un registro nuevo a partir de uno ya existente.

2.4.2.2 Científico de datos/desarrollador

RF 11. Un científico de datos podrá crear conjuntos de etapas de procesamiento (*pipelines*).
RF 12. Un científico de datos podrá eliminar conjuntos de etapas de procesamiento.
RF 13. Un científico de datos podrá modificar conjuntos de etapas de procesamiento.
RF 14. Un científico de datos podrá consultar todos los conjuntos de etapas de procesamiento.
RF 15. Un científico de datos podrá consultar en detalle un conjunto de etapas de procesamiento.
RF 16. Un científico de datos podrá crear conjuntos de etapas de procesamiento (*estrategias*).
RF 17. Un científico de datos podrá eliminar etapas de procesamiento.
RF 18. Un científico de datos podrá modificar etapas de procesamiento.
RF 19. Un científico de datos podrá consultar todas las etapas de procesamiento.
RF 20. Un científico de datos podrá consultar en detalle una etapa de procesamiento.

2.4.2.3 Administrador

RF 21. Un administrador podrá ver una lista de los usuarios registrados.
RF 22. Un administrador podrá ver en detalle la información relativa a un usuario.
RF 23. Un administrador podrá modificar la información relativa a un usuario.
RF 24. Un administrador podrá eliminar un usuario.

2.4.3 Requisitos no funcionales

2.4.3.1 Rendimiento

RNF 1. Cualquier operación de la aplicación no debe tardar más de cinco segundos en completarse, a excepción de la creación de registros y estrategias.

2.4.3.2 Usabilidad

RNF 2. Debe tener una interfaz intuitiva.
RNF 3. Se debe poder iniciar una grabación en menos de 5 clics.

2.5 Historias de usuario

En este apartado se presentan los requisitos funcionales identificados como historias de usuario. Se incluye una descripción de cada historia, que requisitos funcionales cubre y los criterios de validación que determinar que la historia se ha completado correctamente.

El compendio de historias de usuario se encuentra en el Anexo.

2.6 Casos de uso

Se define caso de uso como una lista de acciones o pasos donde se definen las interacciones entre un rol (conocido en el Lenguaje de Modelado Unificado (UML) como un actor) y un sistema, para conseguir un objetivo. Los casos de uso son una técnica empleada para capturar, modelar y especificar los requisitos de un sistema. [37] [38]

A continuación, se muestra una representación del resultado del análisis de requisitos en un diagrama de casos de uso.

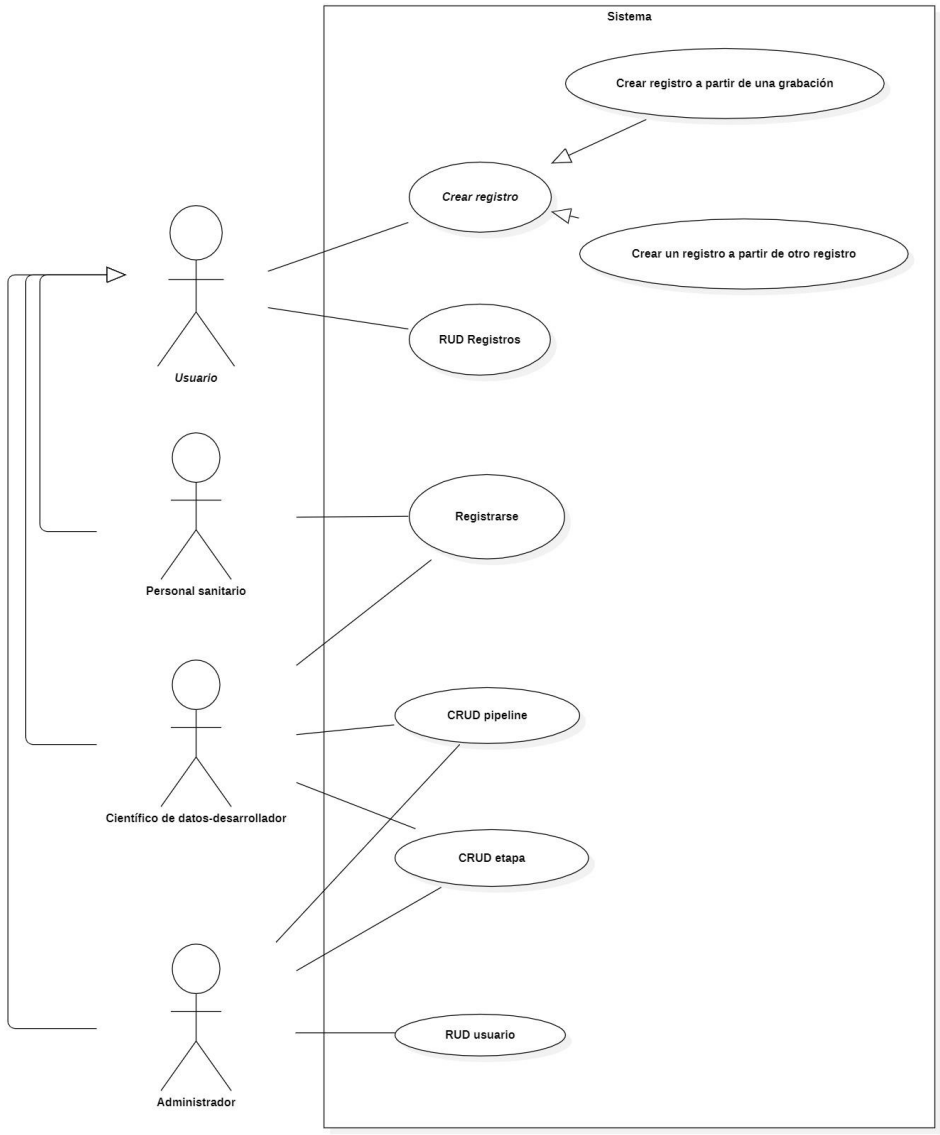

Ilustración 2.6-1 - Diagrama de casos de uso. Las siglas CRUD y RUD hacen referencia a Create, Read, Update and Delete.

2.7 Tecnologías estudiadas y finalmente seleccionadas/utilizadas

Las tecnologías estudiadas y candidatas de uso fuero Django [39] [40]y MariaDB [41] [42]. Sin embargo, Flask y PostgreSQL fueron las tecnologías finalmente utilizadas.

Django es un framework web basado en Python. Django ofrece múltiples componentes en su núcleo, como un sistema de internacionalización, que incluye traducciones de los componentes de Django a varios lenguajes, o un sistema de comunicación interno entre componentes. Adicionalmente ya trae de serie una interfaz dinámica de administración y herramientas avanzadas como un framework para crear Sistemas de Información Geográfica. En consecuencia, dada la magnitud del framework y la configuración de numerosas funcionalidades que probablemente no fueran a ser usadas, se optó por usar Flask, que como ya se ha mencionado, es un microframework, cuya versión base se encarga casi exclusivamente de la gestión de solicitudes HTTP, donde el resto de las funcionalidades son adquiridas en paquetes adicionales opcionales.

Con respecto a la persistencia, en un principio se había estudiado el empleo de MariaDB como base de datos. Tras evaluar la viabilidad de la herramienta y comparando con alternativas se decidió emplear PostgreSQL. Una de las razones es el alto número de vulnerabilidades encontradas [43] y como, de manera consistente, PostgreSQL tiene mejores puntuaciones y es más popular [44].

2.8 Identificación de riesgos y mitigaciones

Uno de los riesgos más destacables es la temporalidad. Se trata de un trabajo con una implementación muy extensa, llena de nuevas tecnologías y paradigmas de programación. Si el tiempo no es administrado correctamente, se puede correr el riesgo de no completar el proyecto a tiempo. Una forma de mitigar el riesgo es haciendo uso de la metodología SCRUM descrita anteriormente. La metodología permite establecer unas reuniones periódicas que ayudan a guiar la cantidad de funcionalidades a completar, así como la realización de un seguimiento del esfuerzo.

Otro riesgo puede surgir de limitaciones tecnológicas. Dado que la herramienta de trabajo empleada es un ordenador personal sobremesa, es posible que las prestaciones no sean suficientes para desplegar el aplicativo. En caso de que este riesgo se materialice, o bien se puede identificar qué prestaciones son las necesarias y resolver la situación a través de una compra concreta. Dado el carácter intensivo de Docker en memoria principal (en parte por sus malas optimizaciones en Windows), se han experimentado escenarios donde la memoria se llena y el sistema procede a paginar. La ralentización como consecuencia de los accesos al disco puede ser resuelta a través de la compra e instalación de memoria principal adicional. Otra alternativa consiste en hacer uso de instancias en la nube y aprovechar su capacidad computacional para las tareas más costosas.

A nivel práctico, se pueden dar fallos en la transcripción y en el procesamiento del lenguaje natural. Sin embargo, habría que determinar el periodo de ocurrencia de estos errores y si son o no deterministas. El problema estaría entonces en las tecnologías empleadas y la solución pasaría por una búsqueda e implementación de alternativas.

Un riesgo importante es la materialización de problemas de seguridad informática. Al tratarse de datos sensibles de potenciales pacientes, en caso de una puesta en marcha en producción, es fundamental garantizar la confidencialidad y privacidad para cumplir con la normativa del RGPD. Es importante considerar las vulnerabilidades a nivel programático e implementar soluciones. Existen numerosas maneras de mitigar este riesgo, como, por ejemplo, haciendo un análisis de riesgos (enfocado a la seguridad) del sistema. Sin embargo, la implementación de medidas básicas de seguridad queda fuera del alcance del proyecto, principalmente, por motivos temporales y se discute con mayor profundidad en la sección de “Trabajos futuros”, “Aspectos relacionados con la seguridad”.

2.9 Aspectos económicos

2.9.1 Plan de negocio

Dado el carácter de prueba de concepto del trabajo y el contexto de desarrollo dentro del proyecto OpenDx28, el aplicativo no será comercializable y en consecuencia no cuenta con ningún plan de negocio.

2.9.2 Evaluación de costes de alojamiento en la nube de Amazon

Amazon Web Services (AWS) es una plataforma de servicios en la nube con más de 200 servicios disponibles a nivel global [45].

Existen numerosas formas de utilizar estos servicios para alojar el aplicativo objeto del proyecto en la nube. Se destacan dos servicios en particular: *Amazon Elastic Container Service* (ECS) [46] y *Amazon Elastic Compute Cloud* (EC2) [47].

2.9.2.1 Descripción y alojamiento usando EC2

EC2 es un servicio de computación, que ofrece "instancias" o máquinas virtuales que se ejecutan con unas prestaciones seleccionadas. Tras seleccionar los requisitos de memoria, almacenamiento y capacidad de computación del procesador, arranca la instancia, a la que se puede acceder de manera remota con un par de claves generados.

El coste de ejecución de una instancia de tipo "t2.micro" es de 0,0116$ (0,11€) por hora [48]. Esta instancia tiene unas prestaciones de 1vCPU (1 hilo de la CPU del servidor) y 1GiB de memoria principal. El coste de otros tipos de instancias con mayores prestaciones asciende rápidamente. Dejar la instancia en ejecución durante un día tiene un coste de 0.264€, lo que implica un coste anual de 96.36€. La selección del tipo de instancia dependerá del uso esperado. Se ha seleccionado la instancia de tipo "t2.micro" como referencia, por ser la "recomendada por defecto", puesto que, si se selecciona la prueba gratuita, se puede ejecutar este tipo de instancia sin coste durante 750 horas al mes [49].

En caso de realizar un alojamiento usando una única instancia de tipo "t2.micro", es necesario configurar el sistema operativo provisto, instalar Docker y usar los contenedores desarrollados. Existe un coste adicional y es el relativo al despliegue de una infraestructura de seguridad. Amazon sólo provee la instancia, las cuestiones de seguridad son responsabilidad del cliente.

En caso de querer una instancia por cada elemento, es decir, una instancia para el Frontend, otra para el Backend y otra para la base de datos, el coste ascendería a los 289.08€ anuales. Amazon además ofrece posibilidades de agrupar instancias y usar herramientas como balanceadores de carga o herramientas de escalado, que, bajo una demanda elevada, crea instancias adicionales a partir de una plantilla y las integra bajo el distribuidor de carga. El coste de empleo de un balanceador de carga es de 0,028$ (0,025€) por hora, sumando al coste anual un total de 219€ [50]. En el caso de usar una única instancia, el coste asciende a los 315.36€ y con tres instancias, a 508.08€ anuales.

2.9.2.2 Descripción y alojamiento usando ECS

El servicio ECS, como su nombre indica, está especializado en desplegar, gestionar y escalar aplicaciones contenerizadas. Está integrado con herramientas de terceros como Docker, o, mejor dicho, principalmente con Docker. Para usar el servicio es esencial dar de alta las imágenes de los contenedores en un registro de imágenes provisto en la propia plataforma.

El servicio permite el despliegue de dos maneras principales: usando instancias de EC2 como las descritas anteriormente, donde el coste es el ya descrito, o usando "Fargate", una herramienta que se encarga de gestionar toda la infraestructura. Fargate abstrae la gestión de servidores o clústeres de instancias en tareas para cada imagen. El precio de uso de Fargate depende del tiempo de uso y las prestaciones asignadas (vCPU, memoria principal, sistema operativo, arquitectura de la CPU y almacenamiento) [51]. Adicionalmente, se puede configurar de tal manera que ciertas tareas se ejecuten durante un tiempo concreto de manera periódica, por ejemplo, durante una hora al día.

El coste de una tarea que se ejecuta 24 horas al día con 2 GB de memoria, 1 vCPU y 20 GB de almacenamiento es de 41,45$ (37,56€) al mes, 450,72€ anuales. Sin embargo, si se estima, por ejemplo, que uno de los contenedores necesita menos prestaciones, se pueden elegir un plan distinto. Una tarea que se ejecuta de manera continua, con 0,25 vCPU, 0,5GB de memoria principal y 20 GB de almacenamiento tiene un coste mensual de 10,37$ (9,40€), es decir, de 112,8€ anuales.

2.9.2.3 Valoración del alojamiento en la nube

De los dos servicios comentados, la opción de menos coste, el empleo de una única instancia de tipo "t2.micro" implica configurar de manera manual muchos aspectos no relacionados con el proyecto, que pueden implicar costes adicionales no previstos. La elección de una solución u otra debe ser discutida en profundidad y evaluada de cara al uso previsto y mantenimiento del proyecto. Amazon provee de planes de ahorro en caso de reservar la instancia unos años por adelantado. Si se tiene claro que se va a dar el servicio durante un tiempo fijo, el uso de uno de estos planes podría ser una opción viable. La otra opción es más cara, pero abstrae y libera de ese coste adicional no previsto. Como ya se ha comentado, la elección de la solución dependerá de las perspectivas de implementación concretas puesto que no hay un servicio claramente superior a otro.

Capítulo 3

3 Diseño

3.1 Arquitectura del sistema

La arquitectura hace referencia al diseño de más alto nivel de la estructura de un sistema. Cada estructura comprende elementos de software, relaciones entre ellos y propiedades tanto de los elementos como las relaciones [52].

La arquitectura de un sistema software es una metáfora, análoga a la arquitectura de un edificio. Funciona como los planos del sistema y el proyecto de desarrollo, que posteriormente serán empleados para extrapolar las tareas necesarias que serán ejecutadas por los equipos y personas involucradas.

Como se ha ido adelantando en las secciones donde se hace referencia a las herramientas empleadas, la arquitectura del sistema está basada en el modelo cliente-servidor [53]. Este modelo es una estructura de aplicación distribuida que divide tareas o cargas de trabajo entre los proveedores de un recurso o servicio, denominados servidores y los clientes. A menudo, los clientes y los servidores se comunican a través de redes informáticas en hardware independiente.

Se denomina servidor a cualquier programa que ofrece un servicio que puede ser accedido a través de la red [54]. Un servidor acepta una solicitud entrante, ejecuta su servicio y devuelve el resultado al solicitante. Un programa actúa como cliente cuando envía una solicitud a un servidor y espera una respuesta. Si el propósito principal de un sistema computador es ofrecer un servicio, el término servidor es usualmente aplicado a la máquina. Con respecto a la escalabilidad, bajo altos niveles de demanda, múltiples servidores pueden ofrecer el mismo servicio, tanto si se hace referencia al programa en ejecución en múltiples procesos de una misma computadora como en varios sistemas computadores.

Se ha elegido seguir este modelo porque es el que mejor se ajusta a los requisitos identificados y funcionalidades solicitadas. De esta manera, la base principal de usuarios, el personal sanitario, podrá hacer uso del programa cliente provisto desde cualquier sistema computador que pueda ejecutar un navegador. Por ejemplo, dispositivos móviles. El aplicativo se conectará a un servidor, con mayores prestaciones, que se encargará de realizar las cargas de trabajo más intensivas y gestionar la información.

Este enfoque da lugar a consideraciones de seguridad adicionales. ¿Debe permitirse el uso del aplicativo en dispositivos personales? ¿Debería desplegarse una infraestructura de red exclusiva para el uso del aplicativo? ¿Debería restringirse el uso en función de la localización del dispositivo? Estas cuestiones se discuten en profundidad en “Trabajos futuros”, “Aspectos relacionados con la seguridad”.

En conclusión, la arquitectura identifica dos tipos de programas principales por desarrollar: los clientes y los servidores.

3.2 Aplicativo cliente

El aplicativo cliente se ejecutará en los navegadores. La razón principal de esta decisión de diseño es la facilidad y los altos niveles de abstracción provistos por *frameworks* y herramientas de desarrollo modernas, que eliminan la necesidad de la configuración manual de las comunicaciones. En otras palabras, proveen de lógica necesaria para desarrollar únicamente en el nivel de aplicación, sin tener que intervenir en los niveles de transporte, red, enlace y físico.

El aplicativo cliente tendrá la responsabilidad de ser la capa de presentación e interacción de la lógica y funcionalidades provistas por el aplicativo servidor.

3.3 Aplicativo servidor

El aplicativo servidor estará constituido por dos servidores, un servidor web y un servidor que provee de lógica de gestión del sistema.

3.3.1 Servidor web

El servidor web es el encargado de servir las páginas web que constituyen el aplicativo cliente descrito en el apartado anterior.

3.3.2 Backend

El otro servidor sirve como la capa de acceso a datos (Backend), donde además se realizan las labores de procesamiento de la información. Es aquí donde se debe analizar y diseñar la forma en la que se procesa un registro.

3.3.2.1 Procesamiento de registros

Partiendo del objetivo de generar un mensaje HL7 a partir de una grabación almacenada en un fichero de audio, se distinguen tres tipos de etapas fundamentales en el procesamiento:

1. Una primera etapa, donde se realiza la transcripción del audio.
2. Una serie de etapas intermedias, que pueden realizar procesamientos adicionales a la transcripción.
3. Una etapa final, que tiene como salida el mensaje HL7.

Para cumplir con los requisitos identificados y lograr la versatilidad deseada, el aplicativo debe permitir usar múltiples implementaciones de cada etapa, o de un conjunto de etapas. Por lo tanto, aparecen los conceptos de:

- **Estrategia:** implementación concreta de una de las etapas.
- **Pipeline:** colección ordenada de estrategias.

La denominación “estrategia” es empleada como consecuencia de adaptar el patrón de diseño “*strategy*” [55] en la resolución del problema. El empleo de este patrón de diseño permite la ejecución estandarizada de múltiples implementaciones concretas distintas, que, concatenadas, forman un *pipeline*.

3.3.2.2 Capa de gestión de datos

El resto de las funcionalidades provistas por el Backend estarán relacionadas con el almacenamiento, modificación, consulta y eliminación de información, sirviendo como interfaz de comunicación con la base de datos.

3.4 Diagramas

En este apartado se adjunta una representación de la arquitectura y los aplicativos descritos en los apartados anteriores de este capítulo.

3.4.1 Arquitectura

La Ilustración 3.4-1 muestra una representación simplificada de la arquitectura del sistema.

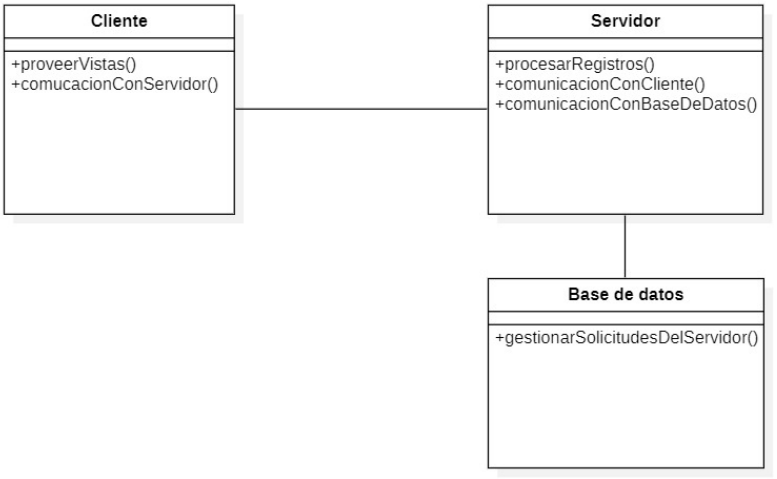

Ilustración 3.4-1 - Representación simplificada del modelo cliente-servidor empleado.

3.4.2 Backend

La Ilustración 3.4-2 contiene una representación canónica del patrón de diseño *strategy*. La clase contexto no implementa un algoritmo de manera directa, sino que delega la implementación concreta del algoritmo a las estrategias concretas.

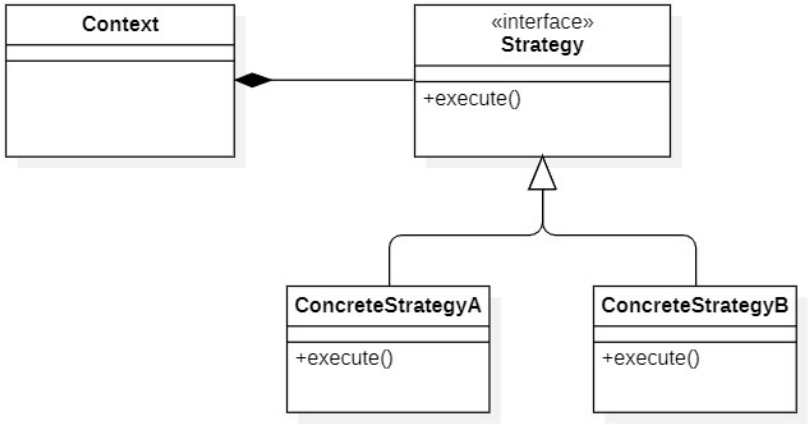

Ilustración 3.4-2 - Patrón de diseño strategy.

Por último, en la Ilustración 3.4-3 se presenta una representación de la adaptación realizada a este patrón. La adaptación es necesaria, puesto que el *pipeline* descrito es el que actúa como contexto de ejecución.

Se modifica la relación entre la interfaz estrategia y el *pipeline* a una agregación puesto que no hay propiedad, es decir, si el *pipeline* es destruido, las estrategias que lo conformaban no son destruidas.

Adicionalmente, es necesario indicar con *"*{sequence}"* la existencia de orden y repetición en la asociación. En otras palabras, el *pipeline* es una secuencia ordenada de estrategias, donde la misma estrategia puede aparecer más de una vez.

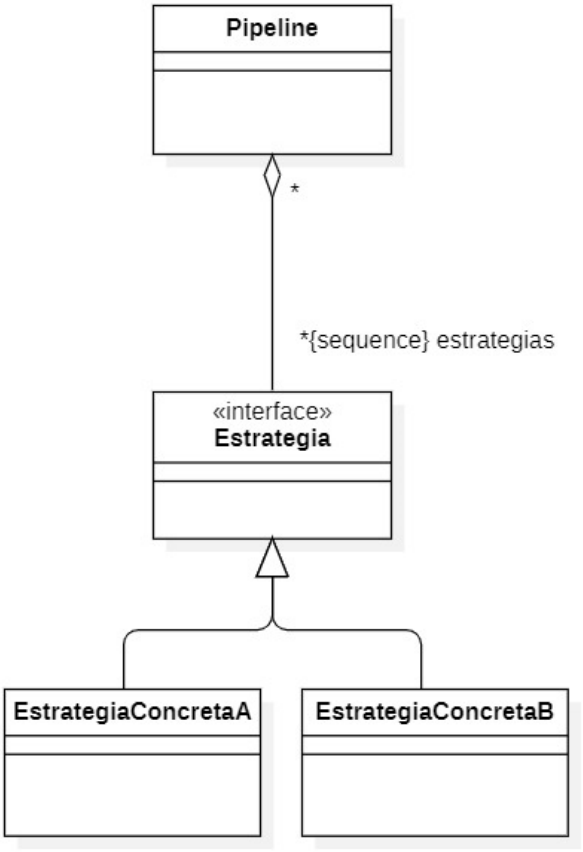

Ilustración 3.4-3 - Patrón de diseño strategy adaptado.

3.5 Persistencia de las entidades del sistema

El sistema cuenta con un total de cuatro entidades: usuarios, historias clínicas electrónicas, *pipelines* y estrategias. Las entidades serán almacenadas en la base de datos PostgreSQL. Se destaca el uso del tipo de dato "*JSONType*" para aquellas estructuras que son representadas con la notación de tipo de objeto de JavaScript, es decir, conjuntos de elementos de tipo clave-valor o listas.

La entidad usuario estará compuesta por el mínimo número de atributos necesarios para identificar un usuario y determinar su rol.

Los registros médicos estarán compuestos por la ruta a la grabación de la que parte, la transcripción realizada en la primera etapa, la historia clínica electrónica, salida de la última etapa, y el conjunto de salidas de todas las etapas de procesamiento implicadas en la creación del registro.

El pipeline sólo contiene un nombre, una descripción y el conjunto de estrategias que lo conforman.

Las estrategias están compuestas de información adicional que será discutida en el apartado de implementación.

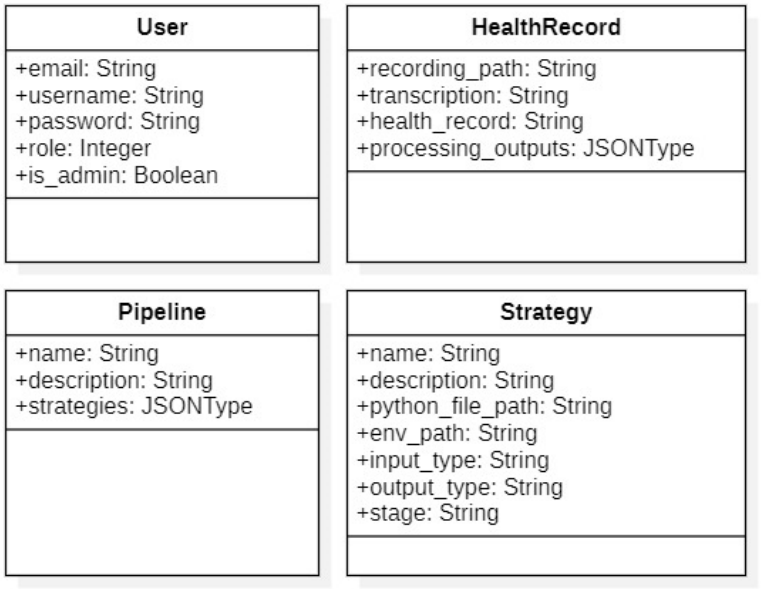

Ilustración 3.5-1 - Entidades del sistema.

3.6 Diseño de las baterías de test

Las baterías de test verificarán las distintas funcionalidades provistas por el aplicativo servidor. El diseño consiste en la prueba de las rutas disponibles por el aplicativo, actuando como cliente. Se realiza una serie de solicitudes al servidor y se comprueba si las respuestas recibidas son satisfactorias. La creación y conceptualización detrás del diseño e implementación de las baterías de test se explican en detalle en "Baterías de test".

Capítulo 4

4 Implementación

En esta sección se explicarán en detalle los pasos, tecnologías y extractos de código destacados que conforman la implementación del prototipo.

4.1 Configuración del entorno de desarrollo usando contenedores

4.1.1 Descripción de los contenedores usados

El primer paso consiste en la configuración del entorno de desarrollo y ejecución. Al tratarse de un sistema con dos servidores con dependencias y funcionalidades distintas, se identifica la necesidad de configurar por separado un entorno para cada servidor.

Por una parte, está el servidor web, encargado de servir el conjunto de páginas de la SPA (*Single Page Application*) desarrollada en Angular. Angular es un framework de JavaScript cuya instalación y uso requiere de Node.js (una plataforma de ejecución de código JavaScript multiplataforma) y su instalador de paquetes, NPM (*Node Package Manager*). Para hospedar el servidor web, se parte de una imagen preconfigurada en Docker Hub [56] de Node JS [57]. Más concretamente, se selecciona la imagen "**node:19-alpine3.16**". Al descargar la imagen, se puede crear un contenedor con el comando "**docker run *imagen***". En este caso, el contenedor puede verse como una máquina virtual con el sistema operativo Alpine Linux con una versión de Node JS ya instalada.

De manera análoga, para el servidor de Backend, se parte de la imagen "**Python:3.10-slim**". Este contenedor está basado en Debian y provee de un entorno estable de ejecución [58].

El último contenedor usado es el relativo a la base de datos PostgreSQL [59]. PostgreSQL, a menudo simplemente "Postgres", es un sistema de administración de base de datos relacional de objetos (ORDBMS) con énfasis en la extensibilidad y el cumplimiento de estándares. Como servidor de base de datos, su función principal es almacenar datos de forma segura y compatible con las mejores prácticas y recuperarlos más tarde, según lo soliciten otras aplicaciones de software, ya sea en la misma computadora o en otra computadora a través de una red. La imagen oficial de Postgres provista ya trae configurado el servidor de la base de datos, de tal manera que la puesta en marcha sólo requiere la determinación de los valores de unas variables de entorno, como el nombre de la base de datos o la contraseña.

4.1.2 Gestión de los contenedores y despliegue

Para facilitar la gestión de los tres contenedores se hace uso de Docker Compose [60], una herramienta diseñada específicamente para definir y compartir aplicaciones con varios contenedores. El modo de empleo es a partir de un archivo de tipo YAML (*Yet Another Markup Language)* [61], usado para definir los distintos servicios que componen la aplicación. Adicionalmente, por cada servicio se define información adicional como nombre del contenedor que se creará, imagen de la que parte, puertos o variables de entorno. La puesta en marcha de un grupo de contenedores a partir de un fichero "**docker-compose.yml**" se puede hacer a través de la consola, ejecutando el comando "**docker compose up**".

Para evitar un despliegue en el que se crearían los contenedores a partir de las imágenes de Node y Python para, posteriormente, de manera manual, configurar individualmente cada uno de los contenedores, se continúa creando un repositorio en Docker Hub [62], donde, tras modificar un contenedor, se puede crear una imagen de este y subirla al repositorio. De esta manera, y cambiando el contenido del fichero "**docker-compose.yml**" para que apunte a estos nuevos repositorios, el despliegue se haría a partir de la última versión de las imágenes, sin necesidad de configurar los contenedores manualmente. El fichero docker compose creado es el siguiente:

```yaml
version: '3.8'

services:
  frontend:
    image: isacas/tfg-frontend
    ports:
      - 6000:6000
    container_name: tfg-frontend
    command: "tail -f /dev/null"

  backend:
    image: isacas/tfg-backend
    ports:
     - 9000:5000
    container_name: tfg-backend
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: tfg-db
    command: "tail -f /dev/null"

  postgres_database:
    image: postgres
    ports:
      - 5432:5432
    container_name: tfg-postgres-db
    environment:
      POSTGRES_DB: tfg-db
      POSTGRES_PASSWORD: pass
```

Extracto 4.1-1 - Contenido del fichero "docker-compose.yml".

Véase como se han creado dos repositorios: "*tfg-frontend*" y "*tfg-backend*", que contendrán las imágenes de los contenedores para los aplicativos Frontend y Backend, respectivamente. En el fichero se definen los tres servicios descritos en el apartado anterior, especificando imágenes, puertos, nombres y variables de entorno. Con respecto a los puertos, la sintaxis funciona de la siguiente manera: se define qué puerto del anfitrión se corresponde con el puerto del contenedor. Por ejemplo, con respecto al servicio del backend, al arrancar el contenedor, Docker capturará el puerto 9000 del anfitrión de manera que las conexiones que se hagan a ese puerto se corresponderán con solicitudes realizadas al puerto 5000 del contenedor. Esta propiedad se discutirá en detalle posteriormente, al comentar cómo se realiza la comunicación entre el Frontend y el Backend.

Tras haber ejecutado el comando "**docker compose up**", en caso de usar Docker Desktop, aparecen los contenedores organizados en un conjunto:

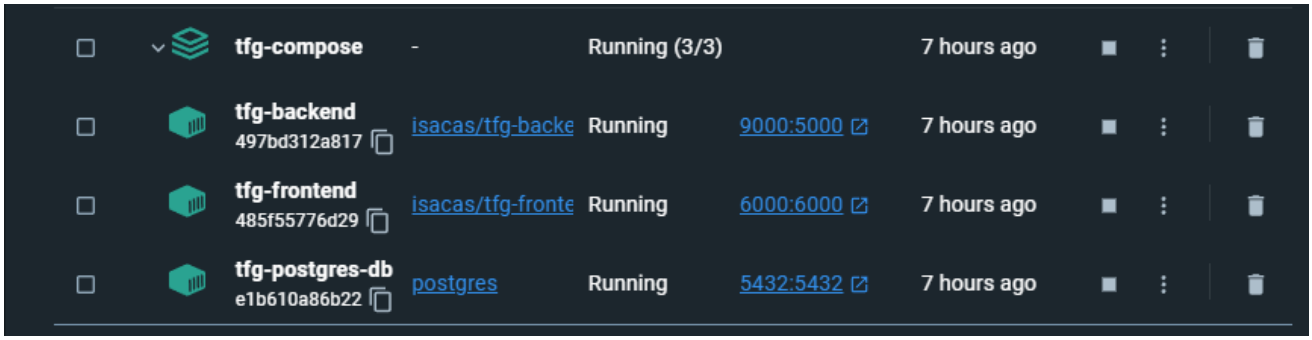

Ilustración 4.1-1 - Organización de contenedores en Docker Desktop.

De esta forma se pueden realizar operaciones (como el arranque, parada o eliminación) sobre el conjunto del aplicativo y no de manera individual para cada contenedor.

4.2 Desarrollo del aplicativo Frontend

4.2.1 Configuración y puesta en marcha

4.2.1.1 Dev Containers

Desde el IDE Visual Studio Code con la extensión "Dev Containers" instalada se procede a ejecutar la funcionalidad de adherencia a un contenedor. El IDE se recarga y se "abre" en el contenedor.

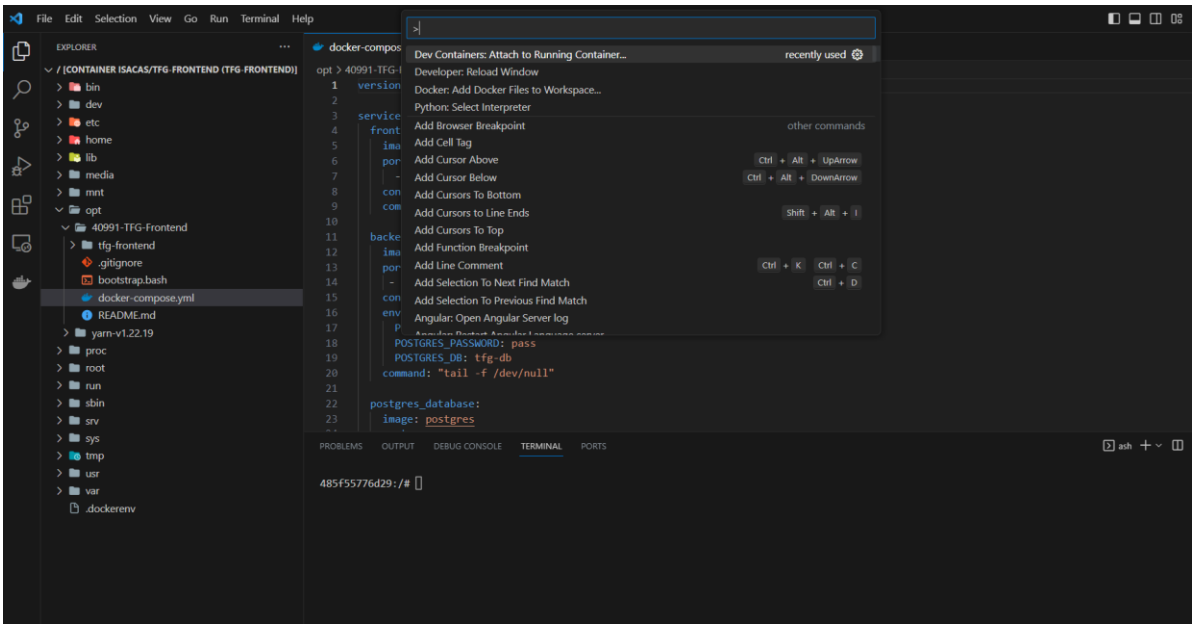

Ilustración 4.2-1 - Vista del IDE Visual Studio Code abierto en el directorio raíz del contenedor "tfg-frontend".

En la Ilustración 4.2-1 se observa cómo se pueden abrir y gestionar ficheros y directorios internos al contenedor de manera transparente, con todas las funcionalidades del entorno de desarrollo. En la imagen se observa cómo se ha creado el aplicativo bajo el directorio /opt/, donde se inicializa el repositorio git con la orden "**git init**".

Se ha decidido realizar el desarrollo bajo este directorio dado que el motivo principal de su existencia es servir como un espacio donde alojar aplicaciones de terceros o paquetes "*add-ons*" que no son parte de la instalación por defecto del sistema operativo. Al instalar, o en este caso, desarrollar una aplicación independiente bajo este directorio, se mantiene separada del resto del software del sistema operativo, evitando dependencias e interferencias innecesarias [63].

4.2.1.2 Instalación de la aplicación de angular

Una vez preparado el entorno, se procede a instalar Angular con el siguiente comando:

```
npm install -g @angular/cli
```

Y se crea un nuevo proyecto con:

```
ng new tfg-frontend
```

Ahora ya quedaría listo el entorno de partida para agregar componentes y desarrollar el aplicativo.

4.2.1.3 Librerías adicionales instaladas

Se instala un total de tres librerías: Angular Material [7], MomentJS [64] y RecordRTC [65].

Angular Material es un conjunto de componentes y funcionalidades provista por el propio equipo de Angular (Google) que permite abstraer la creación de interfaces vistosas para componentes comunes. Se instala haciendo uso de la interfaz de línea de comandos (CLI) previamente instalada:

```
ng add @angular/material
```

Las librerías MomentJS y RecordRTC son empleadas para gestionar la grabación de audio. Se instalan de la siguiente manera:

```
npm i moment
npm i recordrtc
```

4.2.1.4 Puesta en marcha

Finalmente, para poner en marcha el servidor web, es necesario dirigirse al directorio donde se sitúa la aplicación Angular y ejecutar el comando:

```
ng serve
```

Para facilitar la ejecución y hacerla en un paso, se provee de un script, "b**ootstrap.bash**", que cambia de directorio y ejecuta el comando.

En la Ilustración 4.2-2 se muestra una imagen cuyo objetivo es resaltar tres elementos:

A la izquierda se observa la estructura de directorios creada bajo "tfg-frontend", resultado de la ejecución del comando de creación de proyecto "*ng new*". En el siguiente apartado, Componentes, la estructura de directorios, así como el concepto de componente de Angular será descrito en detalle.

En el fichero abierto "*package.json*" en el editor de texto aparecen las dependencias del proyecto, incluyendo las librerías mencionadas en el apartado anterior y las propias dependencias de Angular, como puede ser su compilador o núcleo.

Por último, en la consola se puede ver el resultado de la ejecución de las órdenes relativas a la puesta en marcha del servidor de desarrollo. Como aparece en la última línea, se puede usar el aplicativo a través de un navegador, introduciendo o bien *localhost* o la dirección de *loopback* 127.0.0.1, al puerto 4200. Al hacer esto, el navegador establecerá una conexión con el servidor de desarrollo de Angular, que le transferirá todas las páginas y componentes creados.

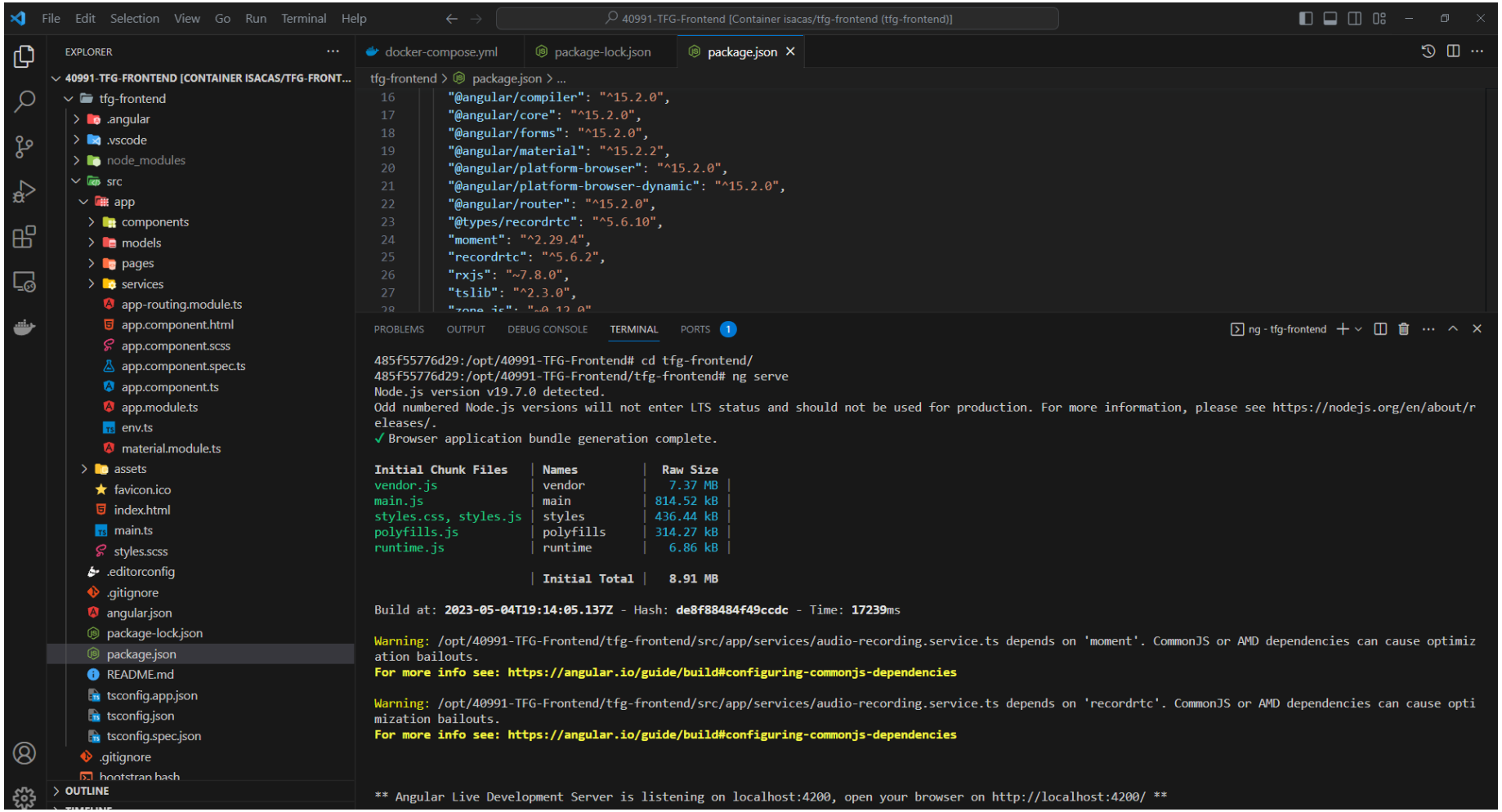

Ilustración 4.2-2 - Puesta en marcha del servidor de desarrollo de Angular.

4.2.2 Componentes

Los componentes son una de las partes fundamentales de la arquitectura de la aplicación. Cada componente tiene su propia plantilla, cuyo núcleo está compuesto por tres ficheros: un fichero HTML (*HyperText Markup Language*), que define el contenido a renderizar, una hoja de estilos CSS (o SCSS) y un fichero con extensión *".ts",* TypeScript, que define una clase para el componente y contiene la lógica de programación requerida para el funcionamiento de éste.

El enfoque basado en componentes permite modularizar el código, haciéndolo más fácil de mantener y reutilizar. Aunque todos los componentes desde un punto de vista abstracto son iguales, es decir, la implementación específica de una plantilla, con el objetivo de simplificar el código, se hace una distinción entre páginas y componentes. Las páginas serían componentes enrutables que usualmente harán uso de otros componentes y los *componentes* serían la implementación concreta de parte de una página.

Un ejemplo que ilustra este concepto es el relativo a la página dedicada a la vista de registros. La página está dividida en tres pestañas, una de vista general de registros, donde aparece una tabla, otra donde aparece la vista de un registro concreto seleccionado de la tabla y una última que permite modificar un registro.

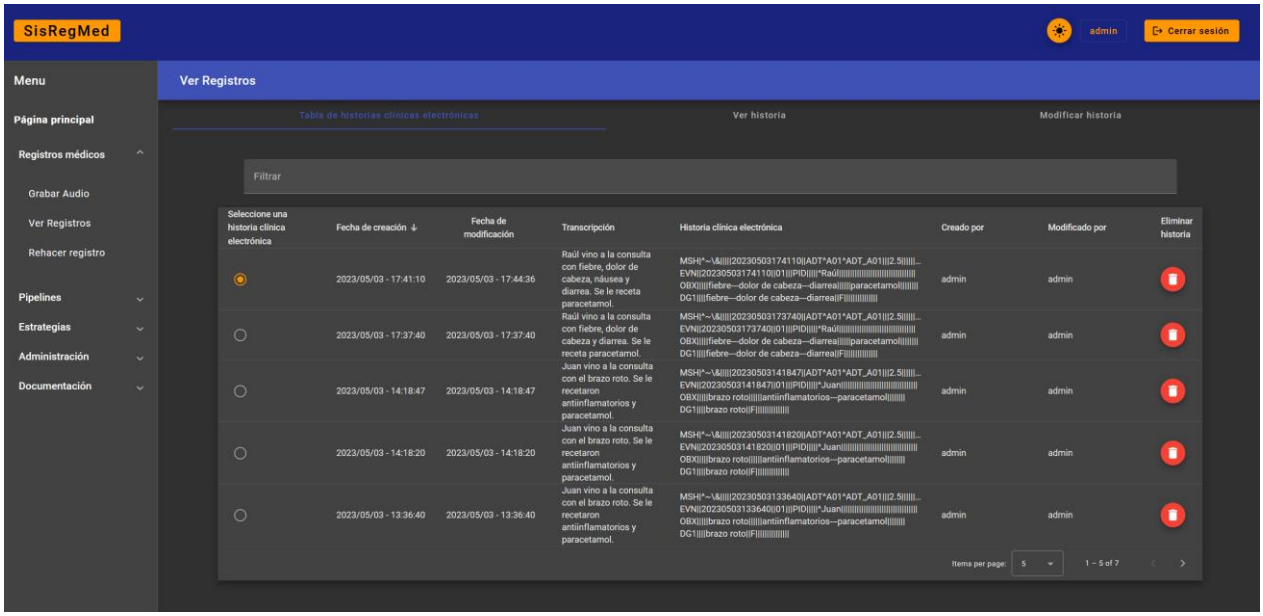

Ilustración 4.2-3 - Página de vista de registros.

El componente considerado como página es bastante ligero. En su HTML sólo es necesario declarar el grupo de pestañas y llamar a los componentes "hijos" que se encargarán de la lógica y las vistas de cada una de las pestañas. Adicionalmente, como ocurre con la vista de un registro concreto, es posible que algunos de los componentes hijos sean reutilizados en otras páginas.

Este enfoque basado en componentes, usado adecuadamente, permite, por diseño, cumplir con los principios SOLID, especialmente con los dos primeros principios: el principio de responsabilidad única, separando las funcionalidades por componentes, y el principio de abierto-cerrado, permitiendo agregar nuevas funcionalidades a través de la agregación de nuevos componentes. La distinción entre páginas y el resto de los componentes tiene como objetivo reducir el riesgo de creación de complejos grafos de dependencias entre componentes, maximizar la cohesión y controlar el acoplamiento. En el Extracto 4.1-1 se muestra en ejemplo de comunicación entre componentes. La *página* incluye *componentes* (etiquetas que empiezan con *app-*) con los que se comunica empleando directivas de Angular.

```
<mat-tab-group>
  <!-- EHR Table -->
  <mat-tab label="Tabla de historias clínicas electrónicas">
    <app-ehr-table
(selectedRecordEmitter)="updateSelectedRecord($event)"></app-ehr-table>
  </mat-tab>

  <!-- EHR View -->
  <mat-tab label="Ver historia">
    <mat-card class="record-card">
      <mat-card-header>
        <mat-card-title>
          Información detallada sobre la historia clínica electrónica
        </mat-card-title>
      </mat-card-header>
      <mat-card-content>
        <app-view-ehr [selectedRecord]="selectedRecord"></app-view-ehr>
      </mat-card-content>
    </mat-card>
  </mat-tab>

  <!-- EHR Modification -->
  <mat-tab label="Modificar historia">
    <mat-card class="record-card">
      <mat-card-header>
        <mat-card-title>
          Modificar historia
        </mat-card-title>
      </mat-card-header>
      <mat-card-content style="margin-top: 1em;">
        <app-modify-ehr [selectedRecord]="selectedRecord"
(updatedRecordEmmiter)="updateSelectedRecord($event)"></app-modify-ehr>
      </mat-card-content>
    </mat-card>
  </mat-tab>
</mat-tab-group>
```

Extracto 4.2-1 - Componente HTML de la página de vista de registros.

4.2.2.1 Comunicación entre componentes

La comunicación de componentes por parte de un componente padre se lleva a cabo a partir de las directivas que aparecen en la imagen.

La comunicación **hacia** los hijos se realiza con la directiva vinculante. Entre corchetes aparece el nombre de la variable esperada como entrada en el hijo y entre comillas el nombre de la variable a compartir por el padre.

La comunicación **desde** los hijos se lleva a cabo a través de un emisor, que, al ser llamado desde el hijo, notifica a los padres, que tendrán implementada una función para gestionar la salida emitida.

En este caso "*selectedRecord*" es una variable empleada por el componente padre para comunicar a los hijos. La selección de un registro en el componente encargado de la tabla implica la emisión de su valor hacia el padre, actualizando su variable almacenada. Al ser esta variable la entrada de los componentes de vista y modificación, el contenido de las pestañas cambiará y se adaptará al registro seleccionado.

Desde un componente hijo, basta con definir las variables que se usarán como entrada o emisor de salida usando los decoradores "*@Input()*" y "*@Output()*".

```
@Component({
  selector: 'app-modify-ehr',
  templateUrl: './modify-ehr.component.html',
  styleUrls: ['./modify-ehr.component.scss']
})
export class ModifyEhrComponent implements OnChanges {

  debug: boolean = false;

  @Input() selectedRecord!: HealthRecord;
  @Output() updatedRecordEmmiter: any = new EventEmitter<any>()
```

Extracto 4.2-2 - Decoradores de entrada y salida en el componente de modificación de registros.

4.2.2.2 Enrutamiento

Mencionado anteriormente, el enrutamiento es el mecanismo empleado para gestionar la visualización de distintas páginas. Es un módulo en el que destaca una colección de elementos con dos propiedades, la "ruta" y el componente. La ruta es el identificador empleado para acceder a un componente y el componente, una referencia al componente a cargar cuando se acceda a la ruta. En este módulo sólo se podrán incluir componentes que se consideren páginas.

```
const routes: Routes = [
  {
    path: "",
    component: HomePageComponent
  },
  {
    path: "login",
    component: LoginPageComponent
  },
  {
    path: "register",
    component: RegisterPageComponent
  },
```

Extracto 4.2-3 - Parte de la colección de rutas definidas.

4.2.3 Servicios

Los servicios son ficheros idealmente terminados con "*.service.ts*" independientes, no pertenecen a ningún componente. Exportan una clase, bajo el decorador "*@Injectable*", cuyos métodos o variables podrán ser accedidos por componentes. Aunque posteriormente en "Conexión Frontend-Backend" se detalla el uso de servicios para realizar la comunicación con el backend, en este apartado se presenta un servicio empleado para otro propósito, variables globales.

El servicio de variables globales tiene como objetivo mantener cuatro variables de tipo "*BehaviorSubject*". Este tipo de dato concreto está pensado para ser suscrito, de tal manera que cuando cambie su valor, todos aquellos componentes inicializados serán notificados del cambio. En este caso las cuatro variables son las siguientes:

- "*pageName*" se usan para cambiar el nombre que aparece en la cabecera inferior. Cada página será encargada de decidir qué texto aparece en esta cabecera, dado que no es parte del componente, sino común a todas las páginas, un cambio de la información requiere de este mecanismo.
- "*loggedInfo*" mantiene parte de la información del usuario, registrado o no, que esté usando el aplicativo. Esta información es empleada para cambiar la apariencia de algunos componentes. Por ejemplo, aquellos usuarios registrados como personal sanitario no verán los enlaces a las páginas que implementan las funcionalidades de gestión de Pipelines y estrategias.
- "*darkThemeActive*" se encarga de alternar entre el modo claro y oscuro.
- "*debug*" decide o no si se muestra información por consola.

```
import { Injectable } from "@angular/core";
import { BehaviorSubject } from "rxjs";

@Injectable({
    providedIn: 'root'
})
export class GlobalService {
    public pageName = new BehaviorSubject<any>({
        currentPageName: 'Página principal'
    });

    public loggedInfo = new BehaviorSubject<any>({
        isLoggedIn: false,
        username: '',
        role: '0',
        is_admin: false,
    });

    public darkThemeActive = new BehaviorSubject<any>({
        isDarkThemeActive: true,
    });

    public debug = new BehaviorSubject<boolean>(false);
}
```

Extracto 4.2-4 - Servicio de variables globales.

A continuación, se muestra el caso de uso del cambio del nombre de la página. Desde el componente que contiene a la cabecera común al resto, se realiza una subscripción a la variable "*pageName*".

```
    public globalService: GlobalService,
    private userAPIService: UserApiService
  ) {
    // Subscription to pageName. All pages must change the page name in their
constructor
    this.globalService.pageName.subscribe({
      next: newValue => {
        this.currentPageName = newValue.currentPageName;
      }
    })
```

Extracto 4.2-5 - Suscripción al valor de una variable perteneciente al servicio de variables globales.

Para cualquiera de las páginas, al ser accedidas desde su ruta y dar comienzo a su ciclo de vida, en el constructor de la clase que las define, se toma la referencia a la variable y se establece el siguiente valor. Esta acción notifica a todos los componentes suscritos, repercutiendo los cambios.

```
constructor(
  public globalService: GlobalService,
) {
  this.globalService.pageName.next({
    currentPageName: 'Ver Registros'
  })
}
```

Extracto 4.2-6 - Actualización del valor de una variable perteneciente al servicio de variables globales.

4.3 Desarrollo del aplicativo Backend

4.3.1 Sobre Python y entornos virtuales

Python es un lenguaje interpretado y de alto nivel. Es un lenguaje de propósito general, empleado para resolver diferentes tipos de problemas, relacionados con la ciencia de datos, inteligencia artificial, desarrollo web y automatización de tareas.

Para poder ejecutar scripts en Python es necesario tener instalado un intérprete. En función del sistema operativo, la instalación del intérprete puede variar. En sistemas operativos Linux y macOS, Python suele venir preinstalado. En el caso concreto del contenedor utilizado, al usar la imagen "**Python:3.10-slim**", se parte de un Debian con la versión 3.10 de Python instalada a nivel global. En el caso de Windows, existen múltiples formas de instalarlo: descargando un instalador a través de la página oficial de Python [66] o usando un gestor de paquetes como "*winget*" [67] o "*chocolatey*" [68]. Una vez instalado, suele quedar agregado en ciertas variables de entorno del sistema operativo, permitiendo su ejecución a través de la línea de comandos.

Python ofrece la posibilidad de crear entornos virtuales. Un entorno virtual es una instalación de Python aislada del sistema operativo y del resto de entornos virtuales. Esto permite tener distintas versiones de Python y librerías en diferentes proyectos sin que se produzcan conflictos entre ellas.

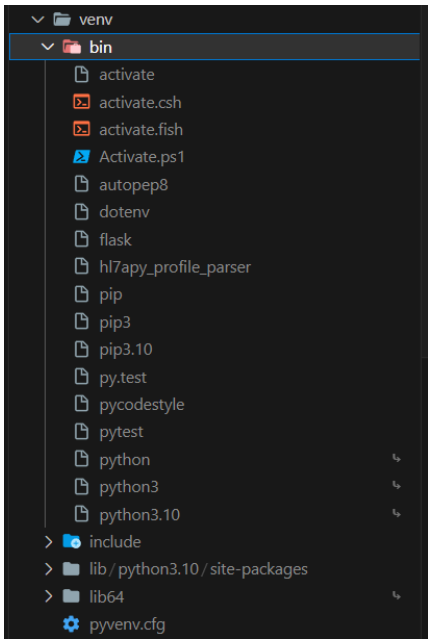

Ilustración 4.3-1 – Contenido de un entorno virtual de Python.

Del contenido de un entorno virtual, es importante resaltar la existencia de un directorio, en Linux "**/bin/**", en Windows "**/Scripts/**", que contiene un conjunto de ejecutables. Uno de los ejecutables más importantes es el de activación. La ejecución del comando de activación hará que los consecuentes comandos que sean ejecutados desde la terminal donde se realizó la activación usen programas pertenecientes al entorno virtual.

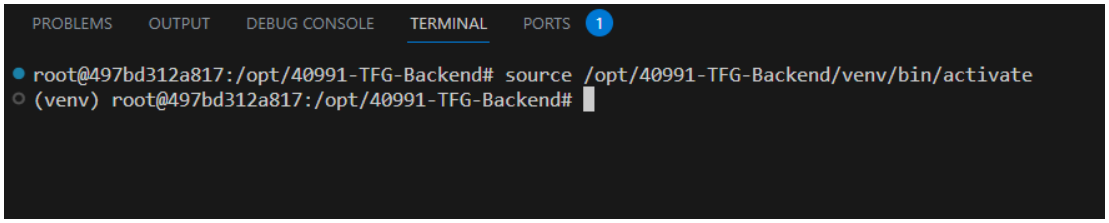

Ilustración 4.3-2 - Activación de un entorno virtual a través de la línea de comandos.

La ejecución de Python en la terminal que se muestra en la Ilustración 4.3-2 estaría ejecutando el intérprete de Python que se encuentra bajo el directorio "**/bin/**". De manera análoga, la ejecución del programa "***pip***", el gestor de paquetes de Python, en un terminal con un entorno virtual activado estaría actuando sobre los paquetes pertenecientes al entorno virtual. De hecho, **la ejecución tanto del intérprete como del gestor de paquetes bajo el directorio "/bin/" de un entorno virtual actuará sobre las dependencias del entorno esté este o no activado.**

Por último, para que la librería de importación reconozca un directorio como un módulo de Python, es necesario incluir un fichero de nombre "**__init__.py**", que puede estar o no vacío. En general, salvo excepciones, se considera una buena práctica dejarlo vacío.

4.3.2 Configuración, Flask y puesta en marcha

4.3.2.1 Configuración del contenedor

De manera análoga al aplicativo Frontend, se crea el directorio "**/opt/40991-TFG-Backend/**", que contendrá el aplicativo y estará sujeto a un control de cambios y versiones. En primer lugar, se crea un entorno virtual con la siguiente orden:

```
python -m venv venv
```

Se está llamando al módulo "*venv*" (*Virtual Environment*) nativo de Python y se crea un entorno virtual de nombre "venv". Posteriormente se instalan las dependencias haciendo uso del gestor de paquetes de Python, "*pip"*:

```
pip install flask
```

De esta manera se instala Flask y las dependencias necesarias para ejecutar el aplicativo. Se cuenta con un total de 24 dependencias, visibles en un archivo "*requirements.txt"*, generado con la orden:

```
pip freeze > requirements.txt
```

4.3.2.2 Configuración de Flask

Flask es un microframework web cuya versión base se encarga de manera casi exclusiva de la gestión de solicitudes HTTP. Cuenta con múltiples librerías y extensiones que facilitan ciertas operaciones, como la gestión de usuarios y control de sesiones, o la conexión y comunicación con una base de datos.

El arranque de una aplicación Flask implica la configuración de un conjunto importante de variables de entorno. Las variables de entorno de la configuración contienen información relacionada con el modo de ejecución o la identificación del recurso de la base de datos.

Adicionalmente, Flask permite cargar una configuración a partir de un objeto ya creado. Para simplificar la configuración de la aplicación, se crea un fichero de configuración "*config.py*" directamente bajo "**/opt/40991-TFG-Backend/**", donde se determinan los distintos modos de ejecución. Se crea un objeto de configuración base, posteriormente concretado por objetos hijos, encargados de modificar las variables de entorno pertinentes.

```python
from decouple import config

db_url = 'tfg-postgres-db:5432'
db_name = 'tfg-db'
db_user = 'postgres'
db_password = 'pass'
DATABASE_URI = f'postgresql://{db_user}:{db_password}@{db_url}/{db_name}'

class Config(object):
    DEBUG = False
    TESTING = False
    CSRF_ENABLED = True
    SECRET_KEY = config("SECRET_KEY", default="clave_super_secreta")
    SQLALCHEMY_DATABASE_URI = DATABASE_URI
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    BCRYPT_LOG_ROUNDS = 13
    WTF_CSRF_ENABLED = True
    DEBUG_TB_ENABLED = False
    DEBUG_TB_INTERCEPT_REDIRECTS = False


class DevelopmentConfig(Config):
    DEVELOPMENT = True
    DEBUG = True
    WTF_CSRF_ENABLED = False
    DEBUG_TB_ENABLED = True

class TestingConfig(Config):
    TESTING = True
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = "sqlite:///testdb.sqlite"
    BCRYPT_LOG_ROUNDS = 13
    WTF_CSRF_ENABLED = False


class ProductionConfig(Config):
    DEBUG = False
    DEBUG_TB_ENABLED = False
    WTF_CSRF_ENABLED = False
```

Extracto 4.3-1 - Fichero de configuración de la aplicación Flask.

Véase cómo se utiliza una base de datos distinta en la configuración de la aplicación cuando le será ejecutada la batería de test.

El resto de la aplicación Flask se encuentra bajo el directorio "**/opt/40991-TFG-Backend/src/**", donde la construcción y configuración de ésta se realiza en el fichero que define el directorio como módulo de Python "**/opt/40991-TFG-Backend/src/__init__.py**". Después de realizar las importaciones necesarias, el primer paso es configurar la aplicación a partir del fichero de configuración mostrado:

```python
from decouple import config
from flask import Flask, jsonify
from flask_cors import CORS
from flask_bcrypt import Bcrypt
from flask_login import LoginManager
from flask_sqlalchemy import SQLAlchemy
from flask_wtf.csrf import CSRFProtect, generate_csrf

# create flask application
app = Flask(__name__)
app.config.from_object(config("APP_SETTINGS"))
```

Extracto 4.3-2 - Inicialización y configuración de la aplicación Flask.

Donde "APP_SETTINGS" es una variable de entorno, que, junto a otras, se encuentra almacenada en el fichero "**/opt/40991-TFG-Backend/.env**". En entornos de producción, debería estar sujeto a varias medidas de protección.

```
export SECRET_KEY=093a9c0bc5dc3daad418a86b4dde1b6b5f3caa07857c2e202ebeef8e74036697
export DEBUG=True
export APP_SETTINGS=config.DevelopmentConfig
export FLASK_APP=src
export FLASK_DEBUG=1
```

Extracto 4.3-3 - Contenido del fichero "/opt/40991-TFG-Backend/.env".

Posteriormente en "Puesta en marcha" se describe el uso de este tipo de ficheros para arrancar la aplicación con distintos parámetros de configuración.

4.3.2.3 *Blueprints* y rutas

Las *Blueprints* en Flask son una forma de organizar las aplicaciones Flask en módulos reutilizables y con un contexto claro. Permiten dividir la aplicación en componentes más pequeños y manejables, que pueden ser registrados en una aplicación Flask. Las Blueprints se pueden utilizar para agrupar rutas y vistas relacionadas en un solo archivo o módulo, escribir extensiones reutilizables o reutilizar código en diferentes partes de la aplicación.

Las rutas son funciones precedidas por un decorador que define el nombre de la ruta y los métodos de conexión que soporta ("HTTP GET", "HTTP POST"). El contenido de la función será ejecutado cuando el servidor reciba una solicitud hacia una ruta concreta.

Con relación al trabajo, se procede a modularizar en función de las distintas entidades: registros, pipelines y estrategias. Para cada una de las entidades, se crea un módulo con un fichero de nombre "**views.py**" donde se define la *Blueprint* y se incluye la lógica de las rutas relacionadas con la interacción con la entidad concreta.

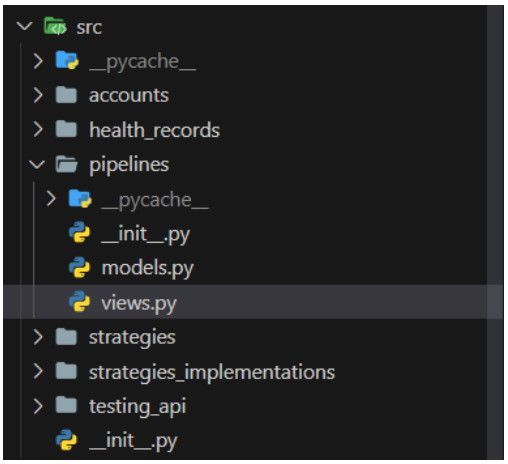

Ilustración 4.3-3 - Estructura de directorios de la aplicación Flask.

```python
from flask_login import current_user
from src import db
from src.pipelines.models import Pipeline

pipelines_bp = Blueprint("pipelines", __name__)

@pipelines_bp.route("/pipelines/get_all", methods=["GET"])
def get_all_pipelines():
    """Function used to fetch all pipelines from the database"""
    pipelines = Pipeline.query.all()
    dictionaries = [pipeline.as_dict() for pipeline in pipelines]
    response = jsonify(dictionaries)
    return response
```

Extracto 4.3-4 - Parte del fichero "views.py" de la entidad pipeline.

En el Extracto 4.3-4 se observa cómo en el fichero "**views.py**" se define la *Blueprint* con un nombre descriptivo y un atributo que hace referencia al contenido del fichero y se crea una ruta que acepta solicitudes de tipo "GET".

Posteriormente, las rutas son registradas en la aplicación Flask (fichero "**__init__.py**") de la siguiente manera:

```
# Registering blueprints
from src.accounts.views import accounts_bp
app.register_blueprint(accounts_bp)

from src.health_records.views import health_record_bp
app.register_blueprint(health_record_bp)

from src.pipelines.views import pipelines_bp
app.register_blueprint(pipelines_bp)

from src.strategies.views import strategies_bp
app.register_blueprint(strategies_bp)
```

Extracto 4.3-5 - Ejemplo de registro de Blueprints en la aplicación Flask.

4.3.2.4 Modelos y comunicación con la base de datos

En la Ilustración 4.3-3 se observa cómo además del fichero "*views.py*", bajo el directorio relativo a los pipelines se encuentra un fichero adicional "**models.py**". Como su nombre indica, contiene la definición del modelo de Pipeline que se almacenará en la base de datos.

La comunicación con la base de datos se hace empleando la funcionalidad de la librería Flask-SQLAlchemy [69]. Es una extensión de Flask que simplifica la adaptación y empleo de SQLAlchemy en la aplicación Flask, abstrayendo la creación de modelos, el manejo de sesiones y la configuración de motores. La librería no cambia el funcionamiento de SQLAlchemy ni cómo es usada.

SQLAlchemy se encarga de la técnica ORM (*Object Relational Mapping*). Esta técnica se encarga de crear una capa de conexión entre la definición de entidades desde un contexto de programación orientada a objetos (OOP) y una base de datos relacional. En consecuencia, la interacción con la base de datos queda enormemente simplificada y abstraída. No es necesario modificar el código en función de la base de datos utilizada. Un ejemplo de uso sería el siguiente:

```
@pipelines_bp.route("/pipelines/delete", methods=["POST"])
def delete_pipeline():
    # Get pipeline id from the request
    pipeline_id = request.json.get('id')
    # Find the pipeline in the database
    pipeline: Pipeline = db.session.execute(
        db.select(Pipeline).filter_by(id=pipeline_id)).scalar_one()
    # Delete the pipeline
    db.session.delete(pipeline)
    db.session.commit()

    response = jsonify(
        {'result': True, 'message': f'Pipeline "{pipeline.name}" eliminado con
éxito.', 'pipeline': None})
    return response
```

Extracto 4.3-6 - Ruta y lógica implicada en la eliminación de un pipeline.

El contenido del fichero “**models.py**” consiste exclusivamente en la definición de la clase relativa a la entidad. Véase como se define el nombre de la tabla a la que pertenece, los tipos de datos y restricciones y métodos adicionales.

```python
from datetime import datetime
from src import db


class Pipeline(db.Model):

    __tablename__ = "pipelines"

    # Pipeline specific fields
    name = db.Column(db.String, nullable=False)
    description =  db.Column(db.String, nullable=False)
    strategies = db.Column(db.JSON, nullable=False)

    # Additional data
    id = db.Column(db.Integer, primary_key=True)
    created_at = db.Column(db.DateTime, nullable=False)
    updated_at = db.Column(db.DateTime, nullable=False)
    created_by = db.Column(db.String, nullable=False)
    last_modified_by = db.Column(db.String, nullable=True)

    def __init__(self, name, description, strategies, created_by,
last_modified_by):
        self.name = name
        self.description = description
        self.strategies = strategies

        self.created_at = datetime.now()
        self.updated_at = datetime.now()
        self.created_by = created_by
        self.last_modified_by = last_modified_by

    def __repr__(self):
        return f"Name: {self.name}\nDone by: {self.created_by}"

    def as_dict(self):
        return {
            'name': self.name,
            'description': self.description,
            'strategies': self.strategies,

            'id': self.id,
            'created_at': self.created_at.strftime("%Y/%m/%d - %X"),
            'updated_at': self.updated_at.strftime("%Y/%m/%d - %X"),
            'created_by': self.created_by,
            'last_modified_by': self.last_modified_by,
        }
```

Extracto 4.3-7 - Contenido del fichero "models.py" relativo a la entidad Pipeline.

4.3.2.5 Puesta en marcha

La puesta en marcha se puede llevar a cabo desde la línea de comandos donde se debe ejecutar la orden:

```
flask run -h 0.0.0.0
```

Es necesario haber exportado previamente las variables de entorno específicas para el tipo de ejecución y haber activado el entorno virtual.

Como se puede ver en el Extracto 4.3-1, se definen tres tipos de ejecución principales:

- Modo desarrollo: **al detectarse cambios en ficheros bajo el directorio "/src/" que define la aplicación, la aplicación Flask se reinicia de manera automática**, permitiendo hacer cambios en la lógica de alguna ruta (normal en un entorno de desarrollo) y que dichos cambios se vean reflejados de manera casi instantánea. Además, por consola se muestra la salida de error e información de depurado adicional.
- Modo producción: la aplicación Flask no se reinicia al detectar cambios en ficheros y la información que aparece es limitada.
- Modo de prueba: se ejecuta la batería de pruebas. La ejecución en modo de pruebas se realiza con un comando distinto al presentado al principio del apartado y se cubre con más profundidad en "Baterías de test".

Para simplificar la ejecución en cualquiera de los modos, se crean tres ficheros de variables de entorno y tres *scripts* de *bash* que ejecutan la consecución de ordenes necesarias para arrancar la aplicación Flask en cualquiera de los tres modos:

```
#!/bin/bash
export FLASK_APP=./src/
source .env
source ./venv/bin/activate
flask run -h 0.0.0.0
```

Extracto 4.3-8 - Contenido del fichero "/bootstrap.bash".

En el extracto se observa el contenido de uno de esos scripts. El arranque a través de un script tiene la ventaja adicional de mantener la exportación de las variables de entorno en el contexto del proceso de ejecución del script y no del terminal. Esto evita desastres, como la ejecución de la batería de test (que elimina y crea la base de datos desde cero) sobre una configuración de la aplicación donde en vez de usar la base de datos de test, use la de producción o desarrollo.

Tras ejecutar el script presentado aparece la siguiente salida:

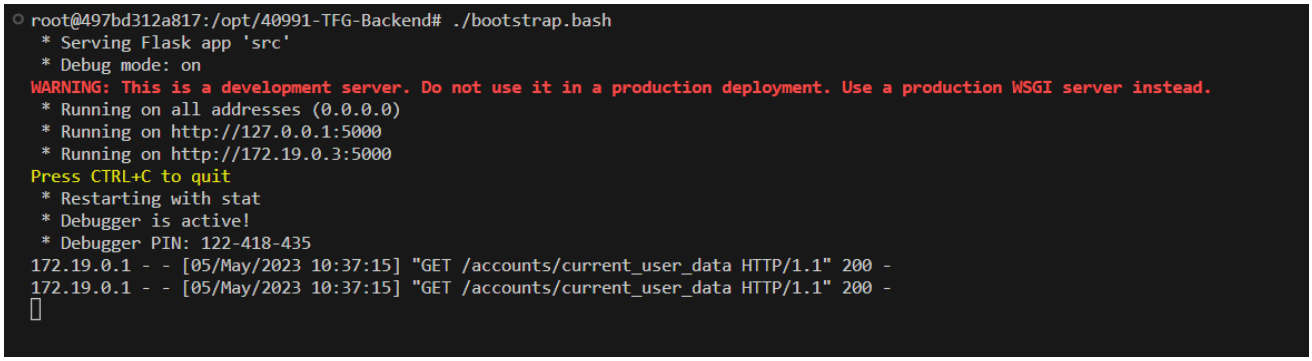

Ilustración 4.3-4 - Puesta en marcha del aplicativo en modo desarrollo.

Se destaca por una parte como el servidor escucha solicitudes HTTP en la dirección de loopback y 172.19.0.3 a través del puerto 5000. Es por eso por lo que se configuró el contenedor de tal manera que redirigiese las solicitudes al anfitrión de su puerto 9000 al 5000 interno al contenedor.

Por último, se observa la ocurrencia de dos solicitudes “HTTP GET” a la ruta “/accounts/current_user_data” por parte del aplicativo Frontend, cuya respuesta ha sido correcta (código 200 OK).

4.3.3 Creación de estrategias

En este apartado se describe la implementación de la funcionalidad de creación de estrategias, junto con ejemplos del sistema trabajando para facilitar la comprensión y contextualización de los matices destacados de la implementación.

Las estrategias son conceptualizadas como lógica de procesamiento independiente a la aplicación Flask. Para garantizar esa independencia y evitar conflictos de con el entorno virtual de Flask, se crea un módulo y un entorno virtual por cada estrategia. En cada entorno virtual se instalarán las dependencias específicas de cada estrategia, obtenidas como resultado de la ejecución del comando "**pip freeze > requirements.txt"**. En resumen, para crear una estrategia, será necesario el fichero con extensión "*.py*" que implemente el procesamiento concreto de la estrategia y un fichero "*requirements.txt"* con sus dependencias.

4.3.3.1 Vista desde el Frontend

Desde el Frontend, en primer lugar, se pedirá rellenar un formulario con información básica de la estrategia, como su nombre, descripción y datos de entrada y salida. Este paso se ve reflejado en la Ilustración 4.3-5.

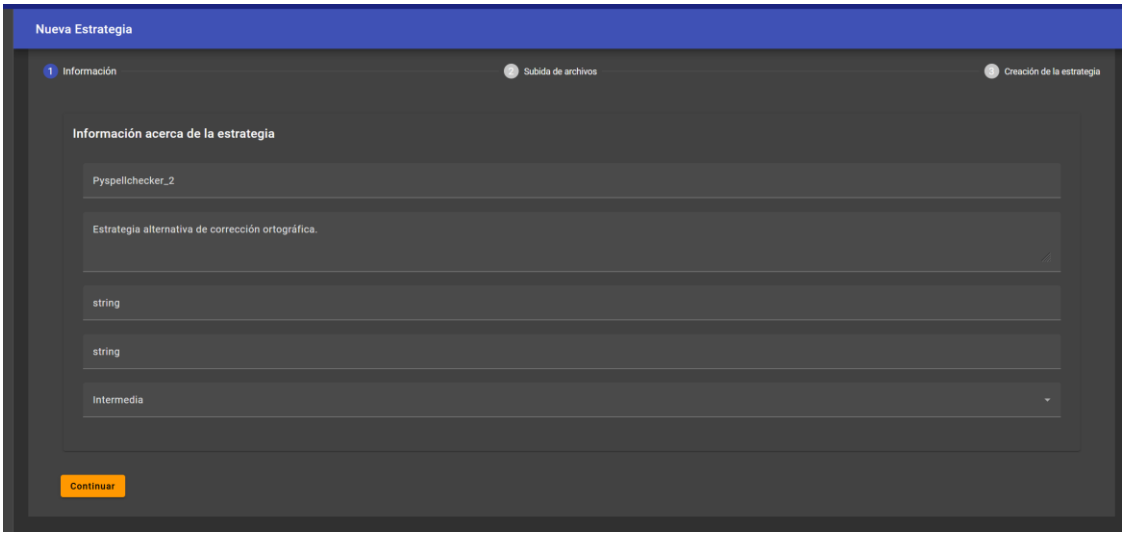

Ilustración 4.3-5 - Formulario de relleno de información relacionada con una nueva estrategia.

Se prosigue con la subida de los dos ficheros (Ilustración 4.3-6 e Ilustración 4.3-7):

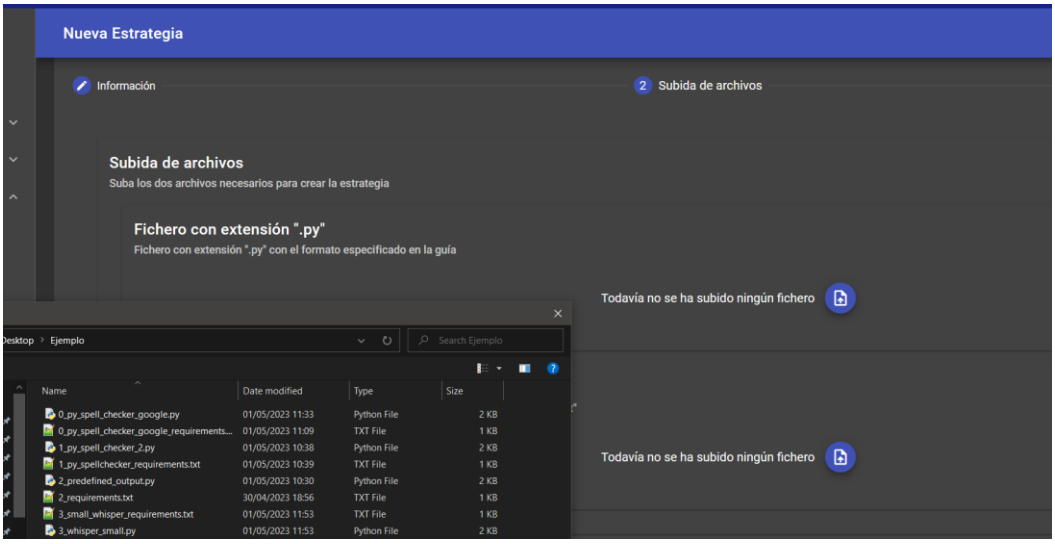

Ilustración 4.3-6 - Subida de los ficheros que definen la estrategia.

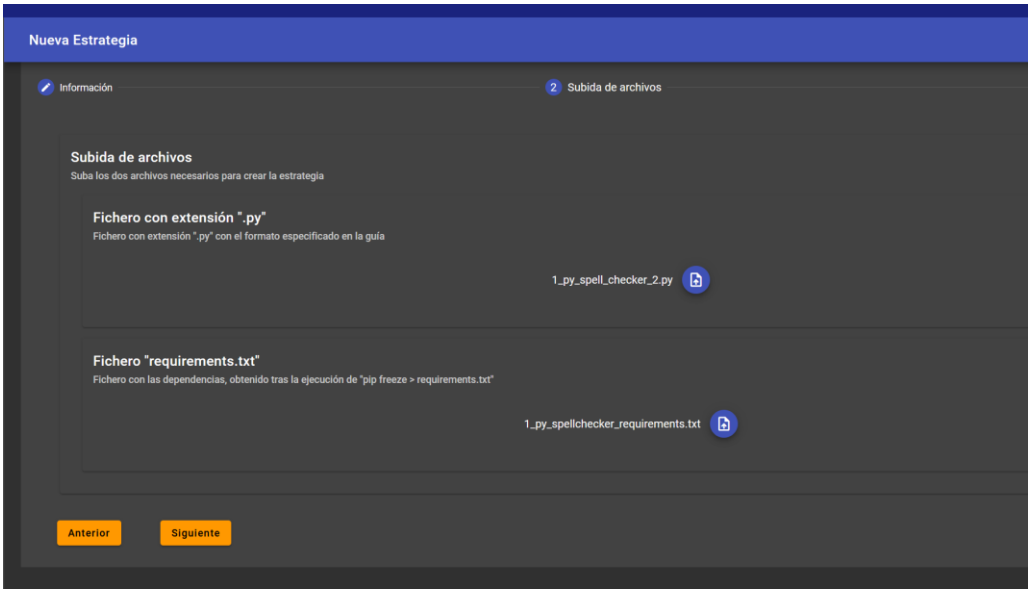

Ilustración 4.3-7 - Ambos ficheros subidos.

Una vez están los ficheros subidos, se presenta una pestaña con un resumen de lo introducido en las dos pestañas anteriores. Esta información se ve reflejada en la Ilustración 4.3-8.

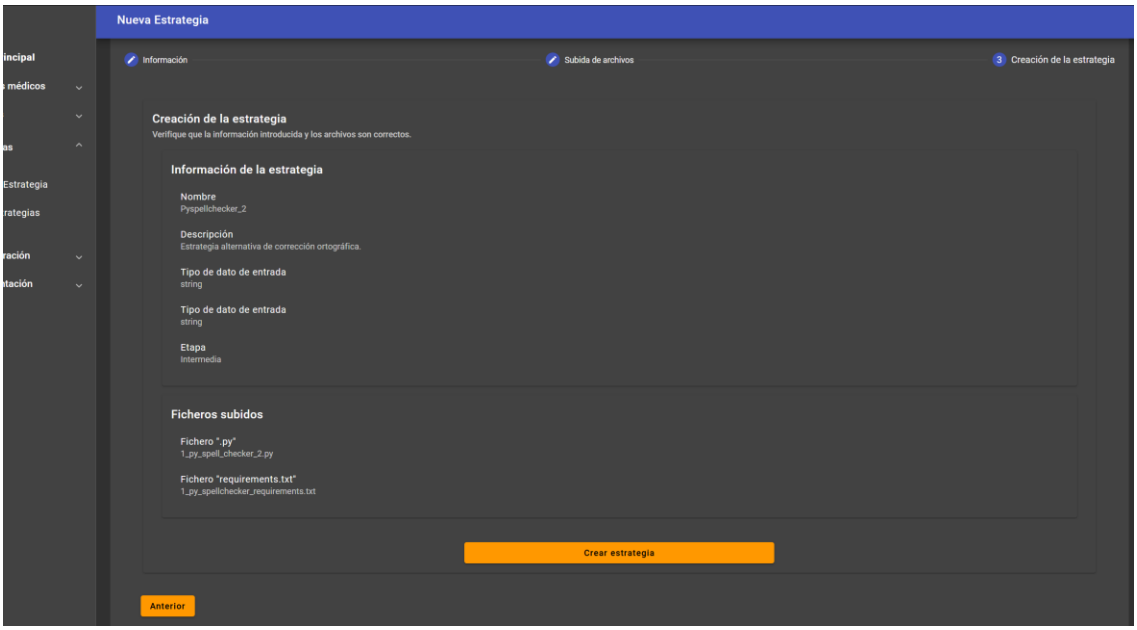

Ilustración 4.3-8 - Resumen de datos introducidos en la creación de estrategias.

Tras darle al botón de “Crear estrategia”, aparece una barra de progreso que notifica acerca de la etapa de creación. La barra de progreso aparece se muestra en la Ilustración 4.3-9.

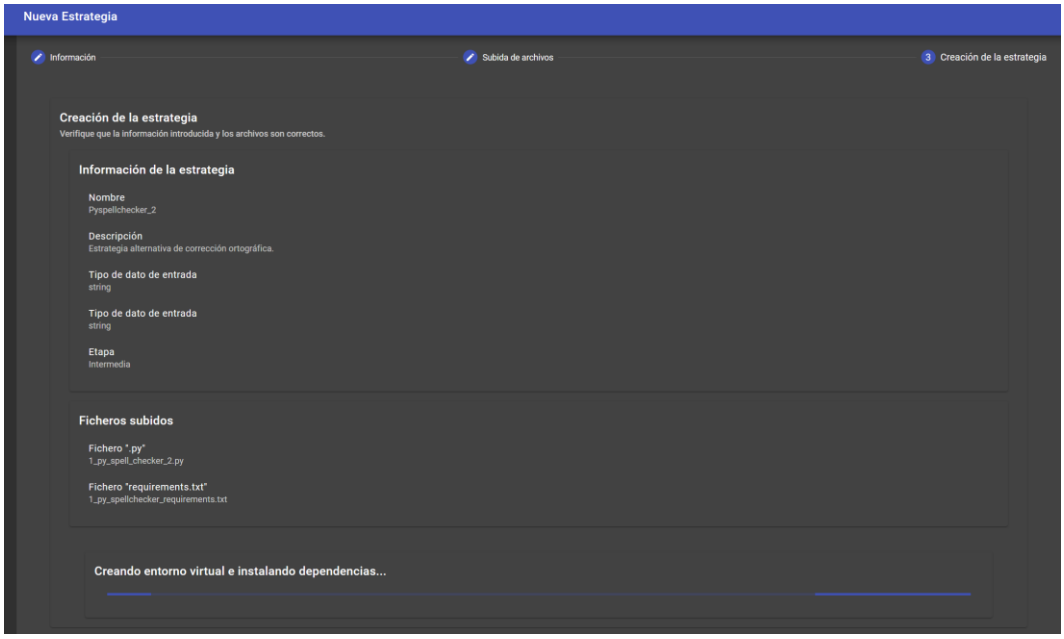

Ilustración 4.3-9 - Espera durante el proceso de creación de estrategias.

Por último, se muestra un mensaje con el resultado de la operación. En el caso de la Ilustración 4.3-10, exitoso.

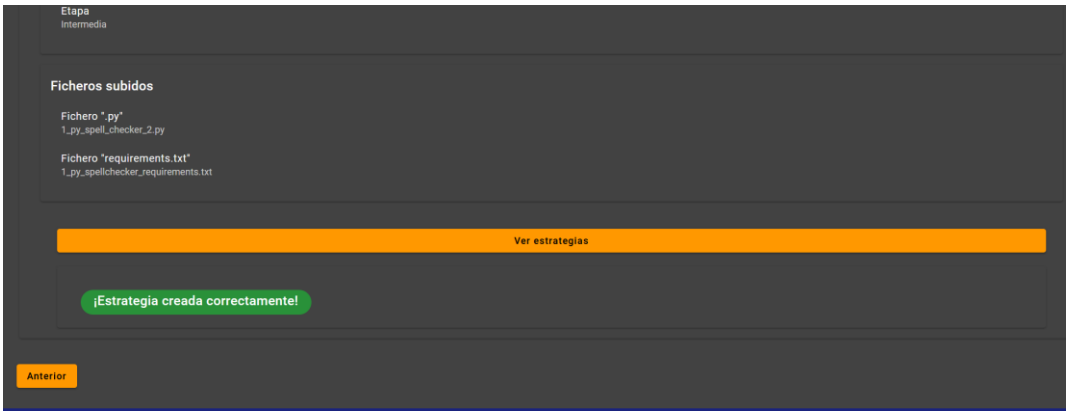

Ilustración 4.3-10 - Éxito en la creación de estrategias.

4.3.3.2 Operativa en el Backend

El proceso de creación de estrategias tiene un total de cuatro etapas: la creación del directorio y fichero "*__init__.py*", la subida y el guardado del fichero "*.py*", la subida y el guardado de "*requirements.txt*" y la creación de un entorno virtual y la instalación de las dependencias de la estrategia.

Se destaca la implementación del último paso en concreto. Para crear el entorno virtual, se importa la librería "*venv*" nativa y se realiza una llamada a su método de creación indicando además que se instale el gestor de paquetes "*pip*". Posteriormente, se instalan las dependencias haciendo uso de la librería "*subprocess*" que permite la ejecución de procesos desde el propio Python. Se crea un subproceso que instala las dependencias en el entorno virtual a partir del fichero "*requirements.txt*" subido. Lo hace ejecutando el programa "*pip*" que se encuentra bajo el directorio "*/bin/*" del entorno virtual.

En el Extracto 4.3-9 se adjunta el código que realiza la creación de un nuevo entorno virtual y la instalación de las dependencias adjuntas en el fichero "*requirements.txt*".

```
def create_virtual_env(strategy):
    # Venv path
    env_dir = f'/opt/40991-TFG-
Backend/src/strategies_implementations/{strategy.name}-
strategy/{strategy.name}-venv'
    # Requirements file
    requirements_file = f'/opt/40991-TFG-
Backend/src/strategies_implementations/{strategy.name}-
strategy/requirements.txt'

    try:
        print("[DEBUG] - [VENV CREATION]: About to venv.create()")
        # Virtual environment creation
        venv.create(env_dir, with_pip=True)
    except Exception as e:
        print(f"[ERROR] - [ATTEMPTING VIRTUAL ENV CREATION]:\n{e}")
        return {'result': False, 'message': "Error during virtual environment
creation"}

    try:
        print("[DEBUG] - [DEPENDENCY INSTALL]: About to subprocess.run")
        # Install dependencies
        subprocess.run([f"{env_dir}/bin/pip", "install",
                        "-r", requirements_file], check=True)
    except Exception as e:
        print(f"[ERROR] - [ATTEMPTING DEPENDENCY INSTALLATION]:\n{e}")
        return {'result': False, 'message': "Error during dependency
installation"}

    return {'result': True, 'message': "Virtual environment created
successfully!"}
```

Extracto 4.3-9 - Creación e instalación de dependencias en un entorno virtual de una nueva estrategia.

4.3.4 Creación de registros

De manera análoga al apartado anterior, se describe la implementación de la funcionalidad de creación de registros, y algunos ejemplos del sistema trabajando para contextualizar las decisiones de implementación.

4.3.4.1 Vista desde el Frontend

La creación de registros puede ser llevada a cabo de dos maneras: a través de una grabación o a partir de otro registro. En el primer caso, el procedimiento consiste en tres pasos: la realización de la grabación, la selección del pipeline que se usará en el procesamiento y un último paso en el que se verifica que la grabación y el pipeline son correctos y se procede a realizar el procesamiento.

En la Ilustración 4.3-11 se muestra la pantalla de creación de registros a partir audio. La Ilustración 4.3-12 junto con la Ilustración 4.3-13 muestran el proceso de captura de audio.

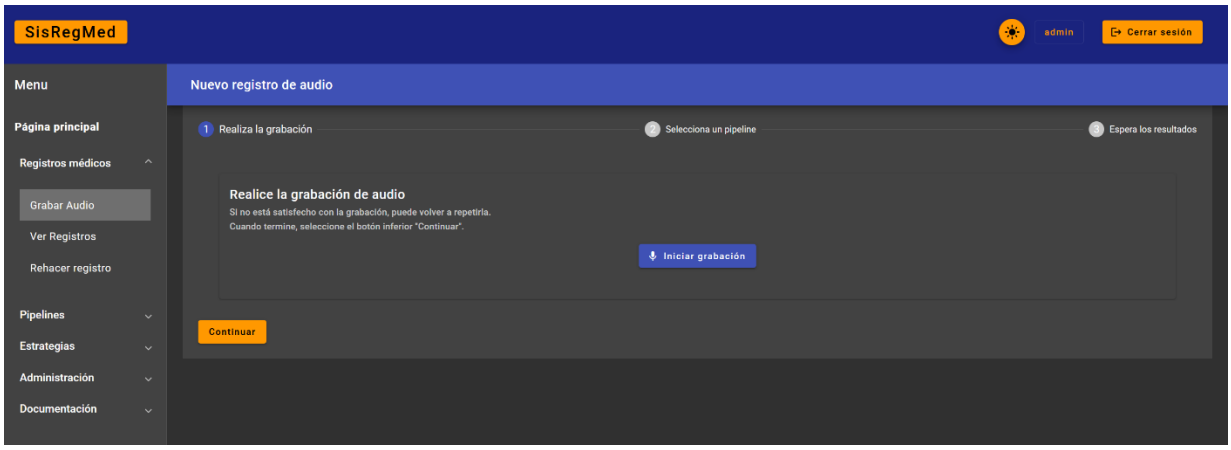

Ilustración 4.3-11 - Pantalla de creación de registros a partir de audio.

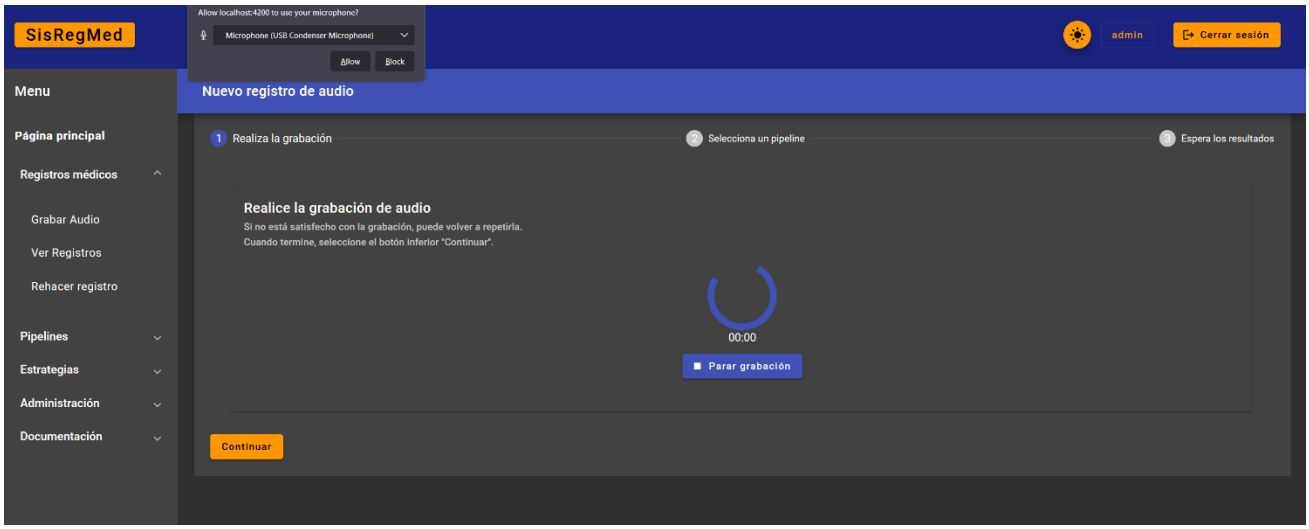

Ilustración 4.3-12 - Solicitud de permisos del micrófono por parte del navegador.

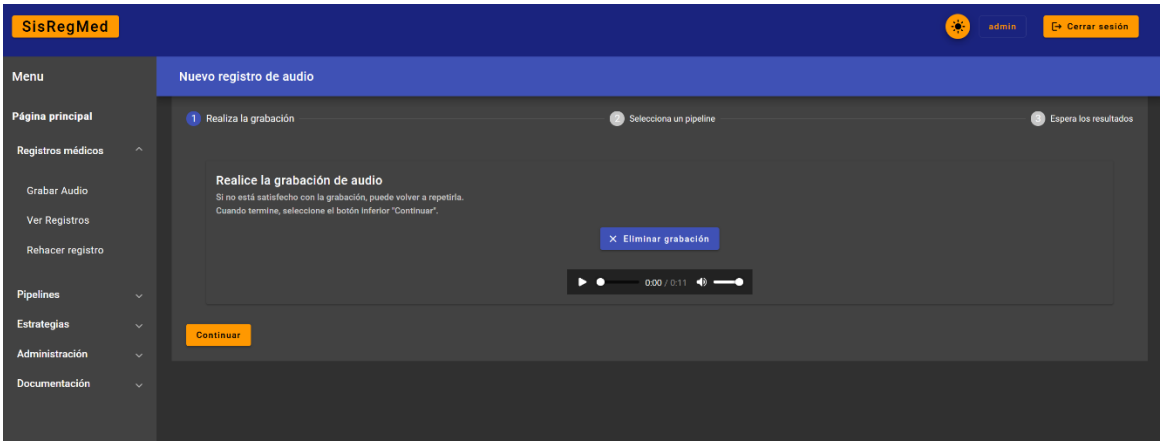

Ilustración 4.3-13 - Finalización de la grabación.

Posteriormente, como se observa en la Ilustración 4.3-14, se selecciona el pipeline que se empleará en el procesamiento. En la Ilustración 4.3-15, se muestra un resumen de lo introducido y se da lugar al comienzo del procesamiento.

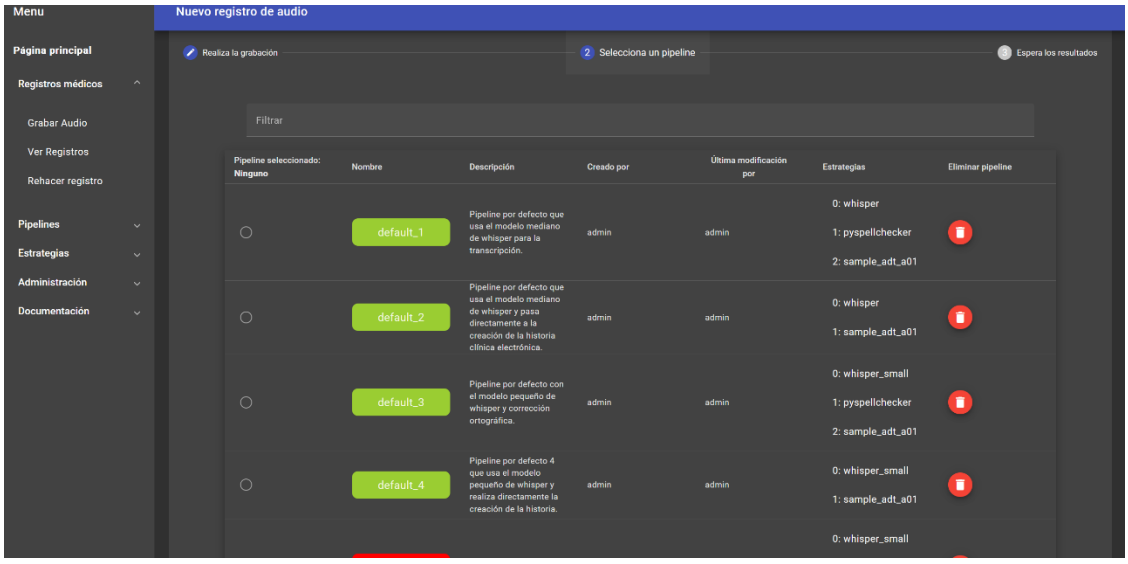

Ilustración 4.3-14 - Selección del pipeline.

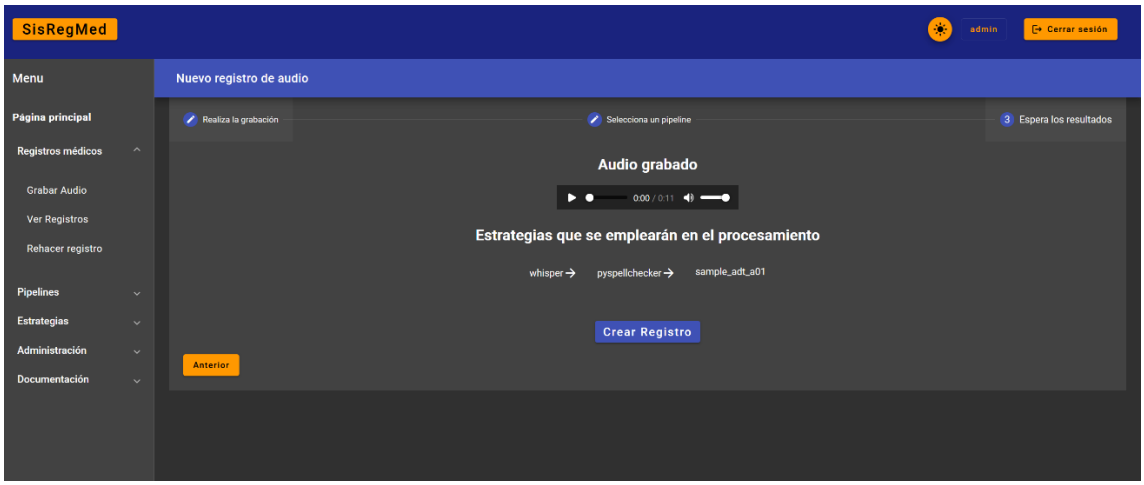

Ilustración 4.3-15 – Resumen de lo seleccionado/introducido y comienzo de procesamiento.

Finalmente, en la Ilustración 4.3-16 se observa el resultado del procesamiento del registro creado. En la Ilustración 4.3-17 se muestra la parte de resultado donde se exponen las salidas de las estrategias empleadas en el procesamiento. El reconocimiento de la entidad nombrada expuesto es parte de la última estrategia.

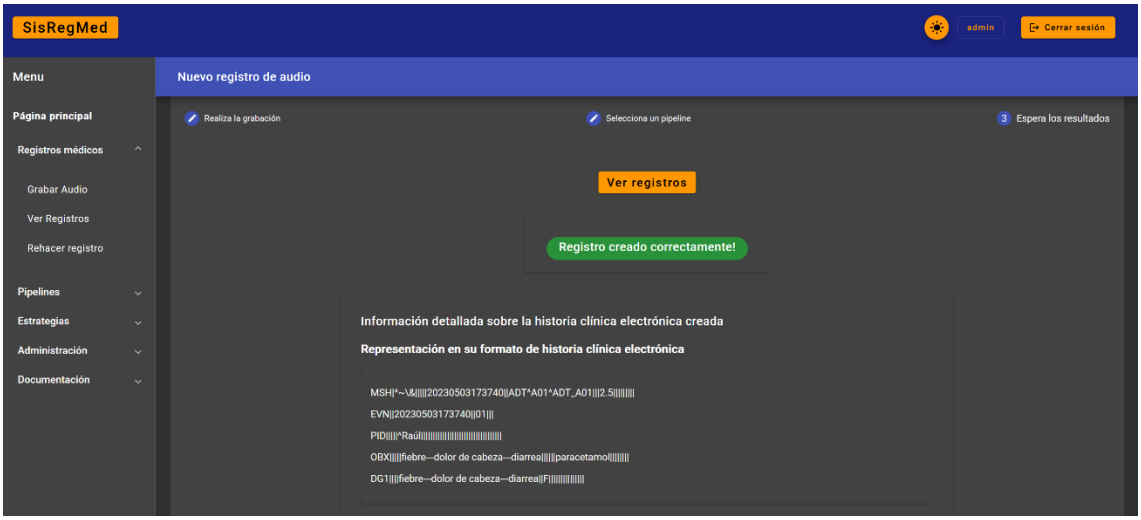

Ilustración 4.3-16 - Resultado exitoso en la creación del registro.

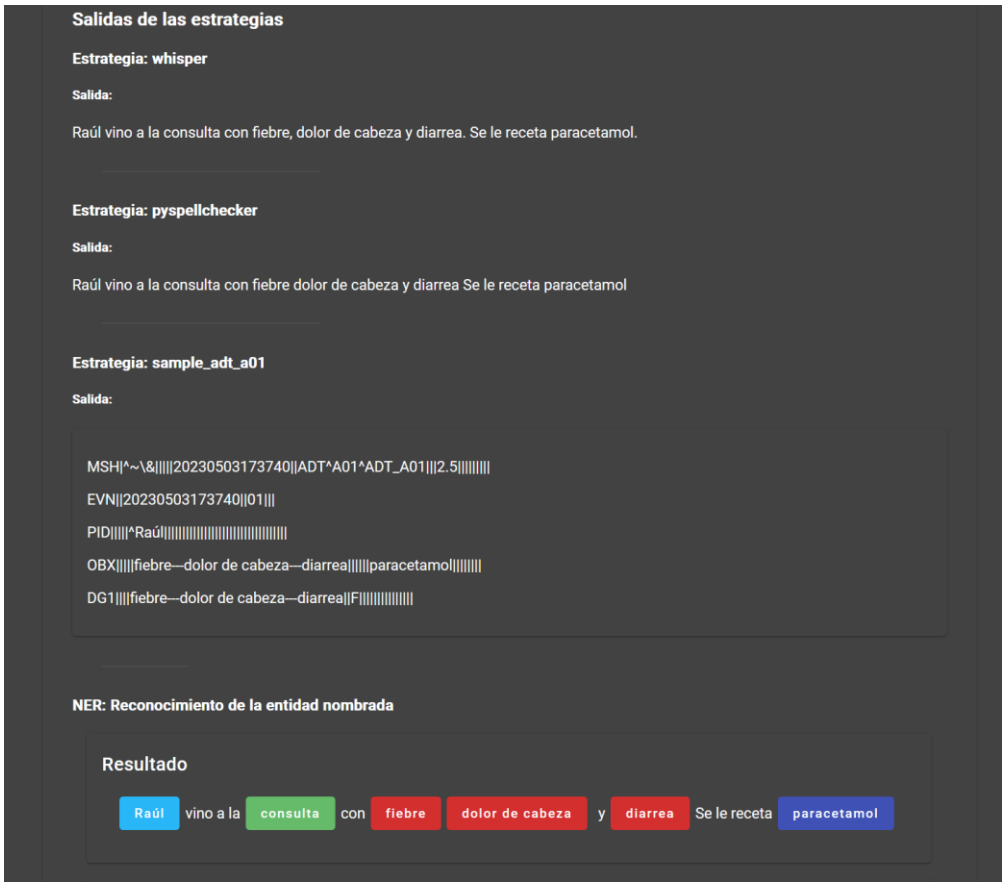

Ilustración 4.3-17 - Vista del registro creado presentada bajo el resultado.

La creación a partir de otro registro varía en los primeros pasos, donde, en primer lugar, se selecciona el registro del que se parte, posteriormente se elige el pipeline empleado en el procesamiento. En la tercera etapa aparecen dos secciones. En la de arriba se selecciona el dato de entrada que se empleará en el procesamiento. En este caso, se trata de la salida de la transcripción o la etapa intermedia de corrección ortográfica. En la sección inferior, se elige la etapa del pipeline a partir de la cual se llevará a cabo el procesamiento. Por último, se muestra un resumen de lo seleccionado antes de realizar el procesamiento, que, al terminar, notifica el éxito de éste y muestra el registro creado. Desde la Ilustración 4.3-18 hasta la Ilustración 4.3-21 se muestran los pasos descritos en el proceso de creación de un registro a partir de otro.

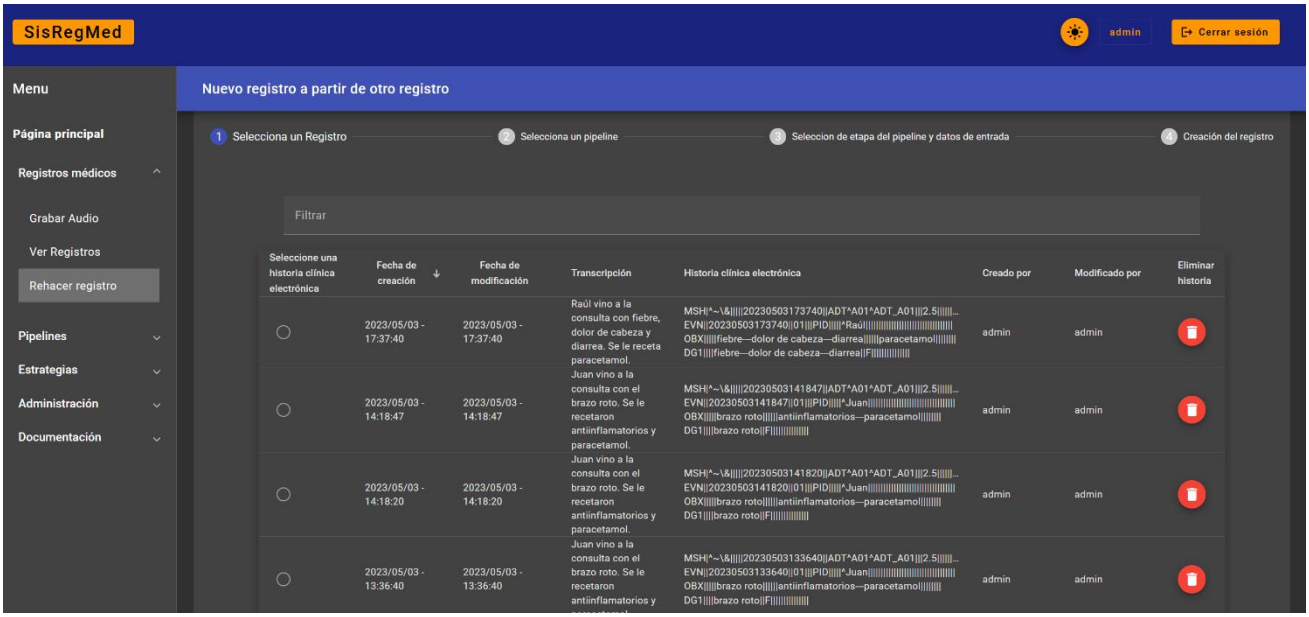

Ilustración 4.3-18 - Selección del registro padre en la creación a partir de otro registro.

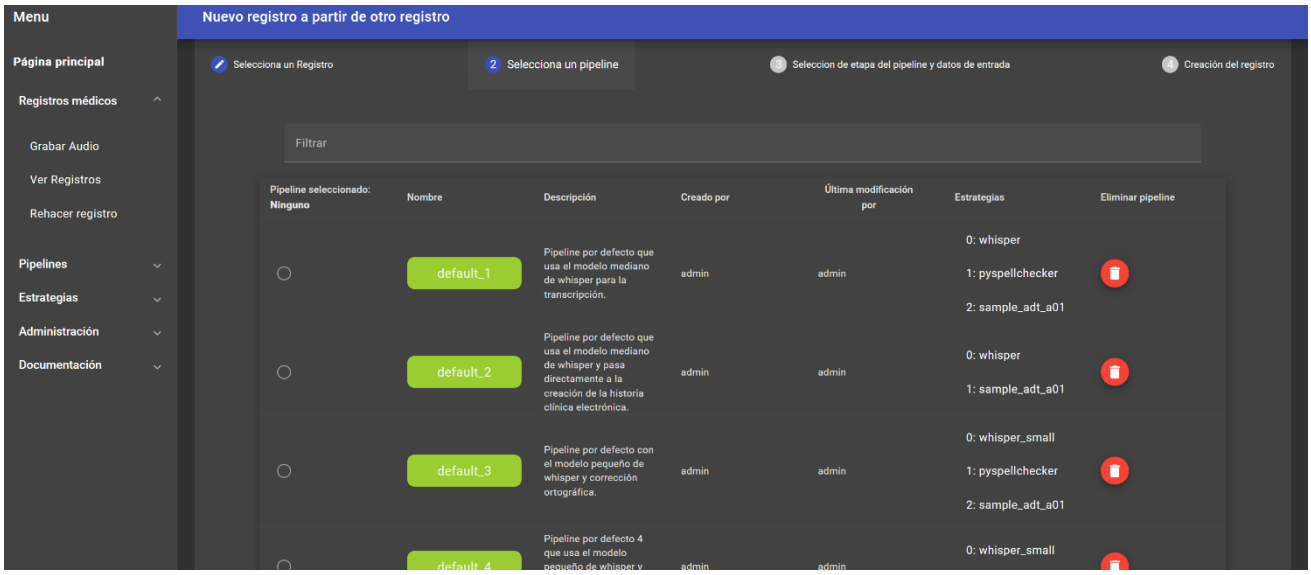

Ilustración 4.3-19 - Selección del pipeline en la creación a partir de otro registro.

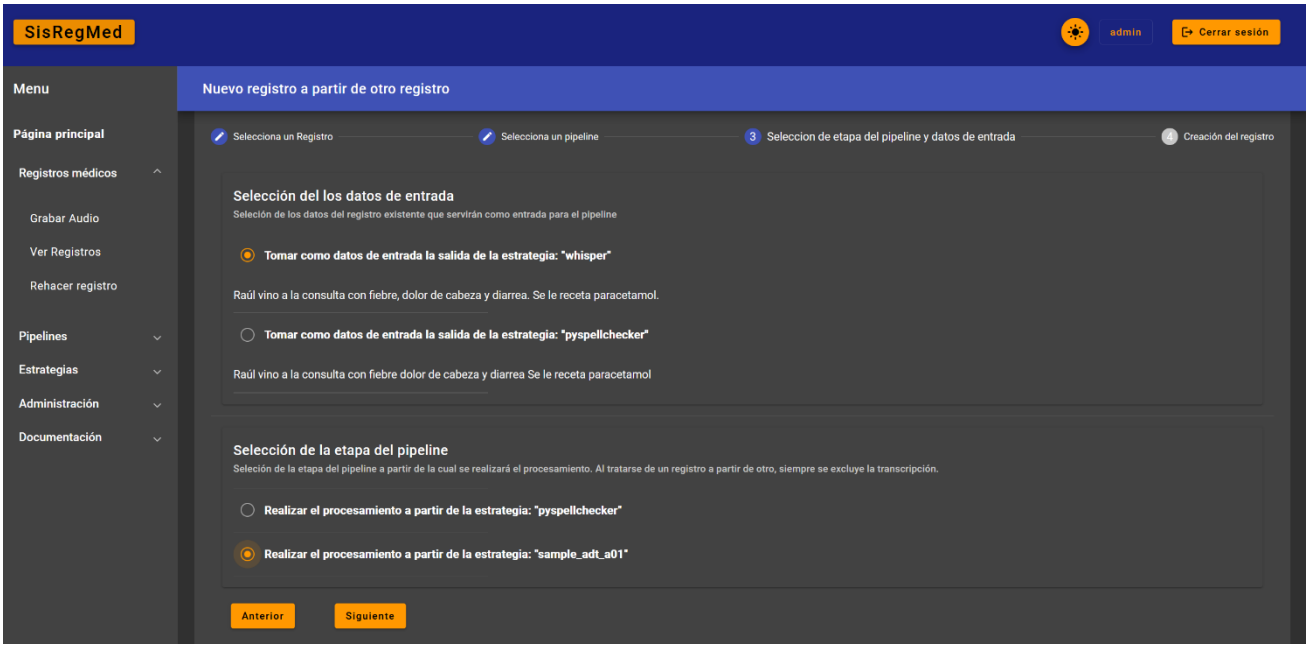

Ilustración 4.3-20 - Selección de la salida de procesamiento que se utilizará como entrada y etapa del pipeline a partir de la que realizar el procesamiento.

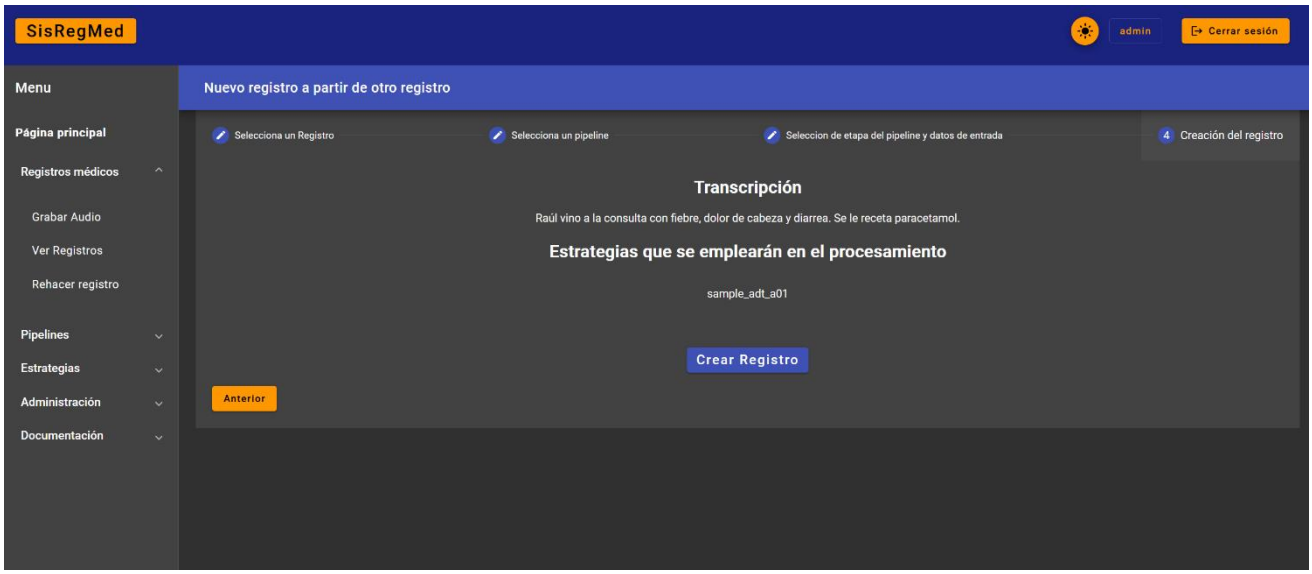

Ilustración 4.3-21 - Resumen de la información seleccionada.

4.3.4.2 Operativa en el backend

En ambos casos, se acaban ejecutando dos funciones, una que *ejecuta* el pipeline y otra que ejecuta de manera individual una estrategia.

La función de ejecución de pipelines puede saltarse partes del pipeline, en función de si el registro es creado a partir de otro registro.

Destaca la forma en la que se ejecuta una estrategia de manera individual. Tras obtener la información relativa a la estrategia, se hace uso de la interfaz "*Popen*" de la librería "*subprocess*". Se define el proceso que se va a ejecutar, es decir, el fichero "*.py*" de la estrategia con el intérprete de Python de su entorno virtual. También se configura la comunicación con el subproceso, definiendo los canales de entrada y salida estándar y salida de error.

Para poder comunicar tipos de datos nativos de Python con el subproceso a través de la entrada o salida estándar, estas variables deben ser serializadas. La serialización se realiza con la librería JSON nativa de Python. Los objetos serializados son de tipo de dato "*string*". En consecuencia, se configura la interfaz "*Popen*" para que las comunicaciones sean de tipo texto y no "*bytes*". Finalmente se comienza la comunicación con el subproceso donde se espera a que éste termine, realizando la deserialización de su salida y retornándola como resultado de ejecución.

```
def run_strategy(strategy_input, strategy_id):
    """Will run a single strategy as a subprocess and return the output

    :param strategy_input: Input data to the strategy
    :type strategy_input: any | string
    :param strategy_id: Unique identifier of the strategy
    :type strategy_id: int
    :return: any: Output of the strategy
    :rtype: any
    """
    # Get strategy information from the database
    strategy: Strategy = db.session.execute(
        db.select(Strategy).filter_by(id=strategy_id)).scalar_one()

    # Run strategy as subprocess
    # As JSON will be used to pass dictionary objects, the text_mode of the
underlying object is true.
    process = subprocess.Popen([strategy.env_path, strategy.python_file_path],
                               stdin=subprocess.PIPE, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)

    # Prepare the strategy input
    strategy_input_dict = {'input': strategy_input}
    serialized_input_dict = json.dumps(strategy_input_dict)

    # Wait for the process to finish
    stdout, stderr = process.communicate(input=serialized_input_dict)

    try:
        strategy_output = json.loads(stdout)
    except Exception:
        print(
            f"\n\n[ERROR] - Couldn't load output of the strategy:
'{strategy.as_dict()['name']}'")
        print(f"\n[ERROR] - Stderr:\n{stderr}")
        strategy_output = {'output': 'Error durante esta estrategia'}

    return strategy_output
```

Extracto 4.3-10 - Ejecución de una estrategia de manera individual.

4.4 Conexión Frontend-Backend

La conexión entre el frontend y el backend se realiza a través de conexiones HTTP. Como se observa en la Ilustración 4.3-4, una vez el backend entra en funcionamiento, sólo espera solicitudes. En consecuencia, es el aplicativo frontend, desde el navegador, el que se encargará de tomar la iniciativa en las comunicaciones. Para ello, se define una variable de entorno con la URL (*Uniform Resource Locator)* del backend.

```
export const API_URL = 'http://127.0.0.1:9000';
```

Extracto 4.4-1 - Variable de entorno en el frontend con la URL del backend.

Nótese cómo el puerto destino de la conexión es el 9000. Este puerto es empleado por el contenedor que aloja el aplicativo backend y se encargará de redirigir las peticiones a su 5000 interno, puerto en el que escucha la aplicación Flask.

Los distintos componentes pueden tomar la iniciativa en las conexiones con el backend a través de servicios. Por ejemplo, se observa cómo el componente encargado de mostrar una tabla con los pipelines registrados usa el servicio relacionado con pipelines para hacer una consulta de todos los pipelines.

```
constructor(
  private pipelineAPI: PipelineAPIService,
  public dialog: MatDialog,
  public globalService: GlobalService,
  private strategyAPI: StrategyAPIService,
) {
  this.dataSource = new MatTableDataSource(this.pipelineList);
  this.fetchPipelines();
}

fetchPipelines() {
  this.pipelineAPI.getAllPipelines().subscribe({
    next: (pipelines) => {
      if (this.debug) {
        console.log("[DEBUG] - [PIPELINE-TABLE-COMPONENT]: Pipelines fetched response:");
        console.log(pipelines);
        console.log("*---*");
      }
      this.pipelineList = pipelines;
      this.dataSource = new MatTableDataSource(this.pipelineList);
      this.dataSource.paginator = this.paginator;
      this.dataSource.sort = this.sort;
    }
  })
}
```

Extracto 4.4-2 - Uso de un servicio por parte de un componente.

A continuación, se muestra el servicio empleado por el componente anterior. Se observa cómo en las llamadas, el identificador de la ruta del backend viene precedido por la variable de entorno definida anteriormente. Se hace uso del cliente HTTP interno de Angular para realizar las solicitudes. Las funciones de "GET" y "POST" devuelven variables de tipo "*Observable*", que tienen la peculiaridad de funcionar a través de suscripciones, dando la posibilidad de esperar una respuesta o establecer un tiempo máximo de espera.

```
import { API_URL } from "../env";
import { catchError } from "rxjs";
import { Injectable } from "@angular/core";
import { Observable, throwError } from "rxjs";
import { HttpClient } from '@angular/common/http'
import { GlobalService } from "./global.service";
import { Pipeline, PipelineRelatedResponse } from "../models/pipeline.model";

@Injectable()
export class PipelineAPIService {
    constructor(
        private http: HttpClient,
        public globalService: GlobalService,
    ) { }

    getAllPipelines(): Observable<Pipeline[]> {
        return this.http.get<Pipeline[]>(`${API_URL}/pipelines/get_all`).pipe(
            catchError(error => {
                return throwError(() => new Error(error.message))
            }))
    }

    getPipeline(id: number): Observable<PipelineRelatedResponse> {
        return this.http.post<PipelineRelatedResponse>(`${API_URL}/pipelines/read`, { 'id': id
}, { withCredentials: true })
    }

    createPipeline(pipeline: Pipeline): Observable<PipelineRelatedResponse> {
        return this.http.post<PipelineRelatedResponse>(`${API_URL}/pipelines/create`, {
'pipeline': pipeline }, { withCredentials: true })
    }

    updatePipeline(pipeline: Pipeline): Observable<PipelineRelatedResponse> {
        return this.http.post<PipelineRelatedResponse>(`${API_URL}/pipelines/update`, {
'pipeline': pipeline }, { withCredentials: true })
    }

    deletePipeline(id: number): Observable<PipelineRelatedResponse> {
        return this.http.post<PipelineRelatedResponse>(`${API_URL}/pipelines/delete`, { 'id':
id }, { withCredentials: true })
    }
}
```

Extracto 4.4-3 - Código relativo a un servicio de comunicación con el backend.

4.4.1 Sesiones y cookies

En Flask, una sesión es un objeto que permite almacenar información entre solicitudes consecutivas realizadas por el mismo usuario. En el aplicativo, esta funcionalidad es usada para actualizar los valores de creación o modificación de entidades. También puede ser usado para trazar y guardar registros acerca de quién hace qué.

La sesión se implementa usando cookies que se envían al navegador del usuario. Estas cookies contienen un identificador único de sesión que se utiliza para asociar las solicitudes posteriores con la sesión actual, permitiendo, por ejemplo, refrescar la pestaña del navegador y seguir apareciendo como que se ha iniciado sesión. Las sesiones y cookies relativas a los usuarios son gestionadas a través de una librería, Flask-Login [70].

Para que el navegador guarde y gestione correctamente las cookies en las comunicaciones, es necesario declarar como verdadero el parámetro "*withCredentials*", tal y como se observa en el Extracto 4.4-3. La Ilustración 4.4-1 muestra el contenido de la cookie almacenada en el navegador.

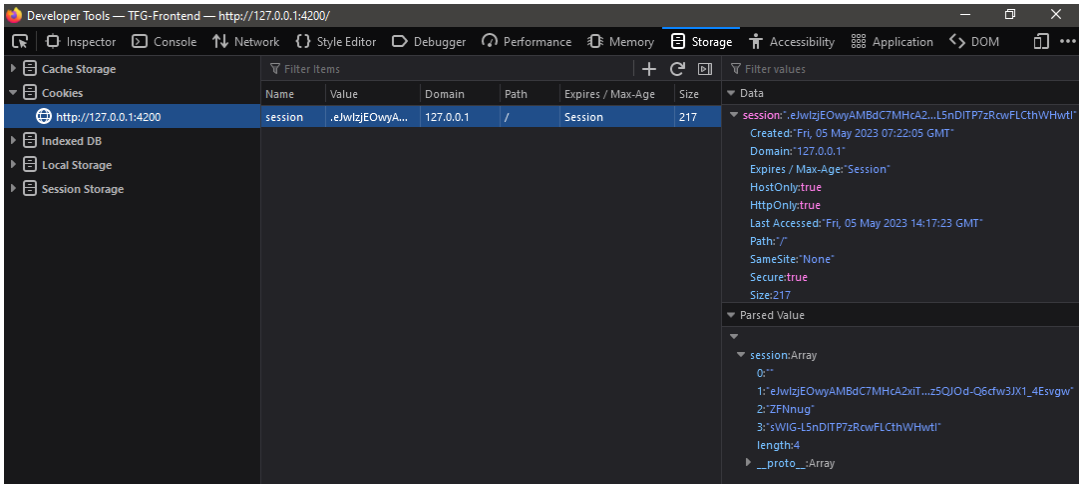

Ilustración 4.4-1 - Cookie de sesión almacenada en el navegador.

Al cerrar sesión, el backend devuelve una cabecera en su respuesta, donde se establece la fecha de expiración de la cookie de sesión al 1 de enero de 1970, haciendo que el navegador la descarte. En la Ilustración 4.4-2 se muestra la respuesta del servidor al cerrar sesión, cambiando la fecha de expiración de la cookie de sesión.

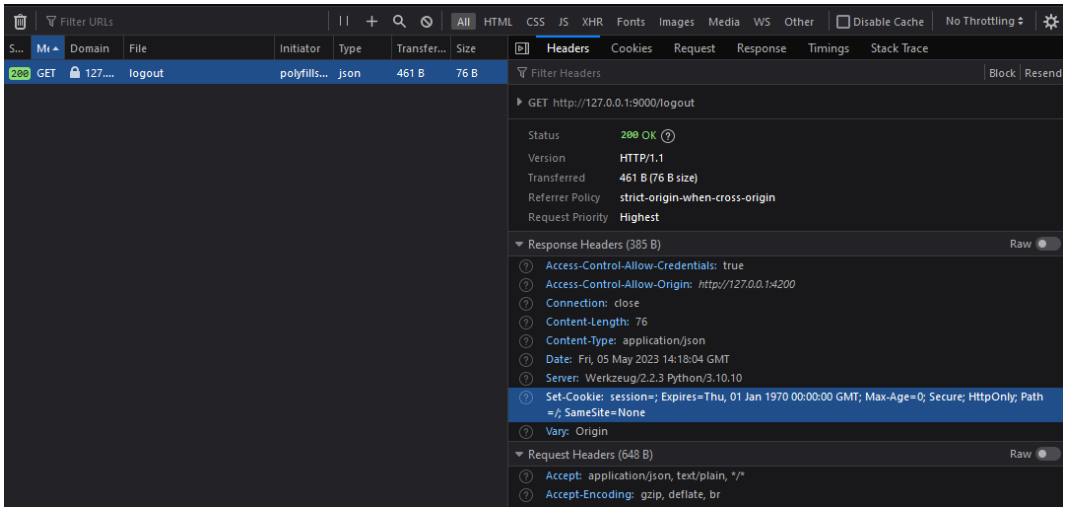

Ilustración 4.4-2 - Expiración de la cookie de sesión al cerrar sesión.

4.5 Baterías de test

4.5.1 Uso de pytest

Las baterías de test se han desarrollado haciendo uso de la librería *"pytest"* [71], puesto que es la manera recomendada por la documentación oficial de Flask [72]. Pytest es un marco de pruebas de Python que permite la escritura de éstas de manera sencilla y eficiente. Ofrece características avanzadas como la detección automática de pruebas, la ejecución paralela de pruebas, la integración con herramientas de cobertura de código y la capacidad de escribir pruebas parametrizadas.

La detección automática de pruebas se basa en convenciones de nomenclatura y ubicación de archivos. Por defecto, pytest busca todos los archivos que coinciden con el patrón "*test_.py"* o "*test.py"* en el directorio actual y sus subdirectorios. Posteriormente, busca dentro de estos ficheros todas aquellas funciones que comiencen con el prefijo "*test"*.

Por lo tanto, se crea el directorio "**/opt/40991-TFG-Backend/test/**" con la estructura que se observa en la Ilustración 4.5-1.

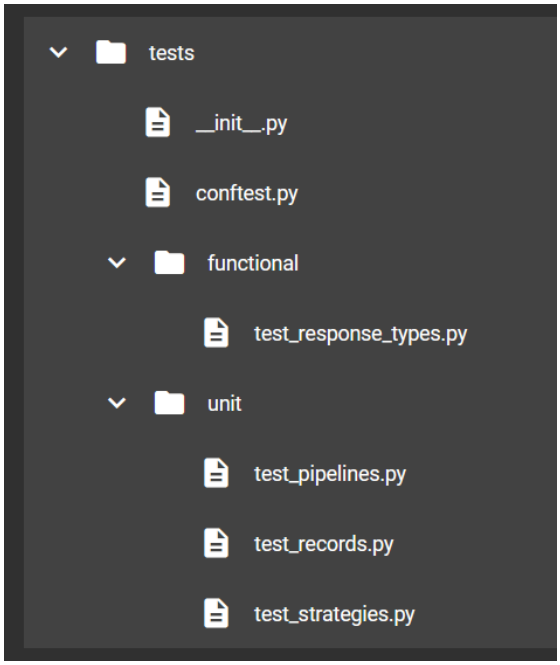

Ilustración 4.5-1 - Estructura del directorio de pruebas.

La estructura permite organizar los test en función de su tipo, permitiendo agrupar test unitarios bajo un mismo directorio. Se observa cómo en el módulo principal se crea un fichero, "*conftest.py"*. Este fichero especial contiene una serie de "*fixtures*" que serán registrados automáticamente al iniciar la batería de test.

Entre otras ventajas, *pytest* simplifica la sintaxis empleada para validar las pruebas. La validación consiste en la agregación de la directiva "*assert"* seguida de una condición booleana.

4.5.2 Fixtures

Los *fixtures* son funciones que se ejecutan antes de cada prueba para preparar un contexto específico de ejecución. Es importante resaltar el contexto de alcance o "*scope*" de un fixture. El alcance determinará el orden o veces que se ejecuta un *fixture* en la batería de test. El alcance puede ser a nivel de función, clase, módulo, paquete o sesión. Los *fixtures* se pueden declarar tanto en el fichero "*conftest.py*" como en módulos específicos que contienen test.

Por ejemplo, en el fichero "*test_pipelines.py*" se declara un *fixture* al principio del módulo. Este *fixture* tiene como objetivo proveer de un pipeline de prueba al resto de test que se ejecuten en el módulo. El *fixture* se declara a nivel de módulo, por lo que será inicializado en el momento que *pytest* comience a ejecutar las pruebas contenidas en el fichero.

```python
import json
import pytest
from src.pipelines.models import Pipeline


@pytest.fixture(scope="module")
def test_pipeline():
    pipeline = Pipeline(
        name="test_pipeline",
        description="test_pipeline_description",
        strategies=[],
        created_by="test",
        last_modified_by="test"
    )
    return pipeline


def test_pipeline_creation(client, test_pipeline: Pipeline, register):
    """
    GIVEN a Flask application configured for testing
    WHEN a new Pipeline is created
    THEN check the name and description fields are defined correctly
    """
    response = client.post("/pipelines/create", json={
        "pipeline": test_pipeline.as_dict()
    })
    parsed_response = json.loads(response.data)
    result = parsed_response["result"]
    response_pipeline = parsed_response["pipeline"]
    assert True == result
    assert response_pipeline["name"] == "test_pipeline"
```

Extracto 4.5-1 - Declaración de un fixture a nivel de módulo y ejemplo de prueba.

```
        TEARDOWN F client
        TEARDOWN F app
tests/unit/test_pipelines.py
    SETUP    M test_pipeline
        SETUP    F app
        SETUP    F client (fixtures used: app)
        SETUP    F register (fixtures used: client)
        tests/unit/test_pipelines.py::test_pipeline_creation (fixtures used: app, client, register, test_pipeline).
        TEARDOWN F register
        TEARDOWN F client
        TEARDOWN F app
        SETUP    F app
        SETUP    F client (fixtures used: app)
        SETUP    F register (fixtures used: client)
        tests/unit/test_pipelines.py::test_pipeline_creation_and_retrieval (fixtures used: app, client, register, test_pipeline).
        TEARDOWN F register
        TEARDOWN F client
        TEARDOWN F app
        SETUP    F app
        SETUP    F client (fixtures used: app)
        SETUP    F register (fixtures used: client)
        tests/unit/test_pipelines.py::test_pipeline_update_and_retrieval (fixtures used: app, client, register, test_pipeline).
        TEARDOWN F register
        TEARDOWN F client
        TEARDOWN F app
        SETUP    F app
        SETUP    F client (fixtures used: app)
        SETUP    F register (fixtures used: client)
        tests/unit/test_pipelines.py::test_pipeline_deletion (fixtures used: app, client, register, test_pipeline).
        TEARDOWN F register
        TEARDOWN F client
```

Extracto 4.5-2 - Ejecución del fixture a nivel de módulo.

En el Extracto 4.5-2, se observa cómo se realiza el Setup del *fixture* al empezar con el módulo. Esto se indica con la M intermedia. En cambio, los *fixtures* configurados a nivel de función realizan el Setup y Teardown antes y después de cada test respectivamente.

4.5.3 Ejecución de los test

A diferencia de los otros modos de ejecución, la batería de pruebas se ejecuta con el siguiente comando:

```
python -m pytest
```

De nuevo, para simplificar la operativa, se crea un script en bash que exporta las variables de entorno correspondientes e inicia las pruebas. La Ilustración 4.5-2 muestra una ejecución normal y exitosa de la batería de pruebas.

```
root@497bd312a817:/opt/40991-TFG-Backend# source /opt/40991-TFG-Backend/venv/bin/activate
(venv) root@497bd312a817:/opt/40991-TFG-Backend# ./run_tests.bash
============================================================ test session starts =============================================================
platform linux -- Python 3.10.10, pytest-7.3.1, pluggy-1.0.0
rootdir: /opt/40991-TFG-Backend
collected 19 items

tests/functional/test_response_types.py ....                                                                                        [ 21%]
tests/unit/test_pipelines.py ......                                                                                                 [ 52%]
tests/unit/test_records.py ....                                                                                                     [ 73%]
tests/unit/test_strategies.py .....                                                                                                 [100%]

============================================================ 19 passed in 1.08s ==============================================================
(venv) root@497bd312a817:/opt/40991-TFG-Backend#
```

Ilustración 4.5-2 - Resultado exitoso de la ejecución de pruebas.

Capítulo 5

5 Resultados

Los resultados son muy favorables, puesto que se ha conseguido cumplir con los objetivos previstos y obtener un prototipo funcional que cumple con los requisitos identificados y analizados.

5.1 Ejemplos del sistema trabajando

Ya en los apartados de “Creación de estrategias” y “Creación de registros” se ha cubierto la funcionalidad descrita en sus nombres, por lo que se procede a describir el proceso de creación de pipelines y vista y modificación de las tres entidades.

5.1.1 Creación de pipelines

La creación de pipelines está compuesta por tres pasos. En el primero, se comienza con un breve cuestionario donde es necesario introducir el nombre del pipeline y una descripción. No se podrá continuar hasta que el cuestionario esté relleno. La Ilustración 5.1-1 muestra este primer paso, con los campos vacíos y el botón de continuar deshabilitado. La Ilustración 5.1-2 expone el mismo paso con los campos rellenos.

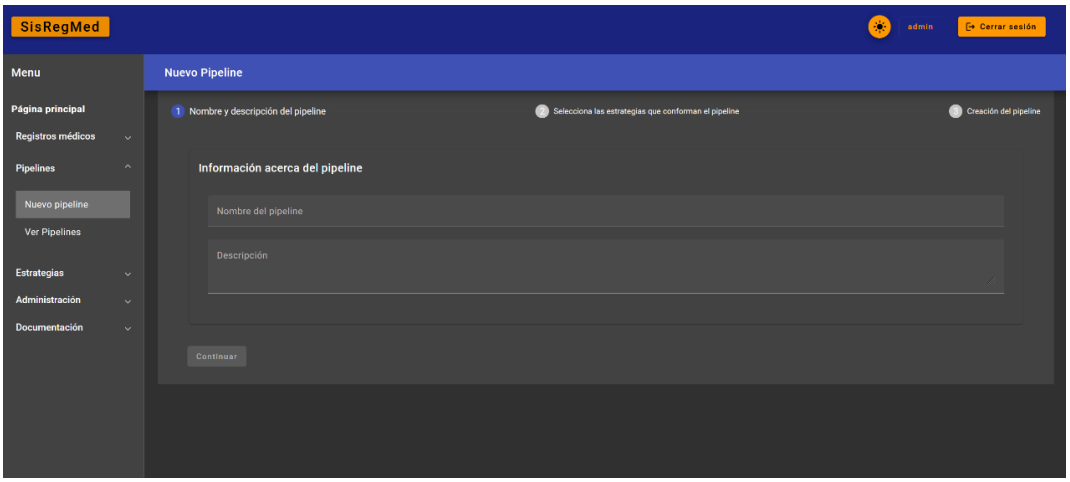

Ilustración 5.1-1 - Etapas implicadas en la creación de un pipeline.

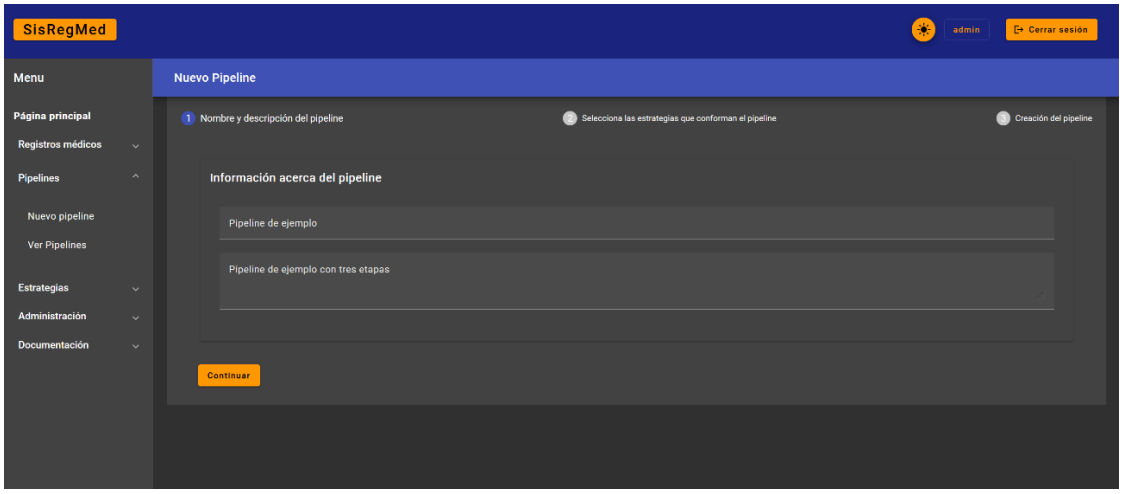

Ilustración 5.1-2 - Relleno del formulario de información acerca de un pipeline.

El segundo paso es el más importante: la selección de estrategias. En primer lugar, el conjunto de estrategias que conforman el pipeline aparece vacío. Sólo aparecen tres indicadores de los tipos de estrategias según el color de la etiqueta en la que aparecen. En la parte inferior de la página aparece una tabla con las estrategias registradas. Sin embargo, la tabla está limitada a las estrategias de la etapa "Voz a texto". Esto se debe a que la primera estrategia de un pipeline debe ser una estrategia que realice la transcripción de un archivo de audio. La Ilustración 5.1-3 muestra el estado inicial de la selección de estrategias.

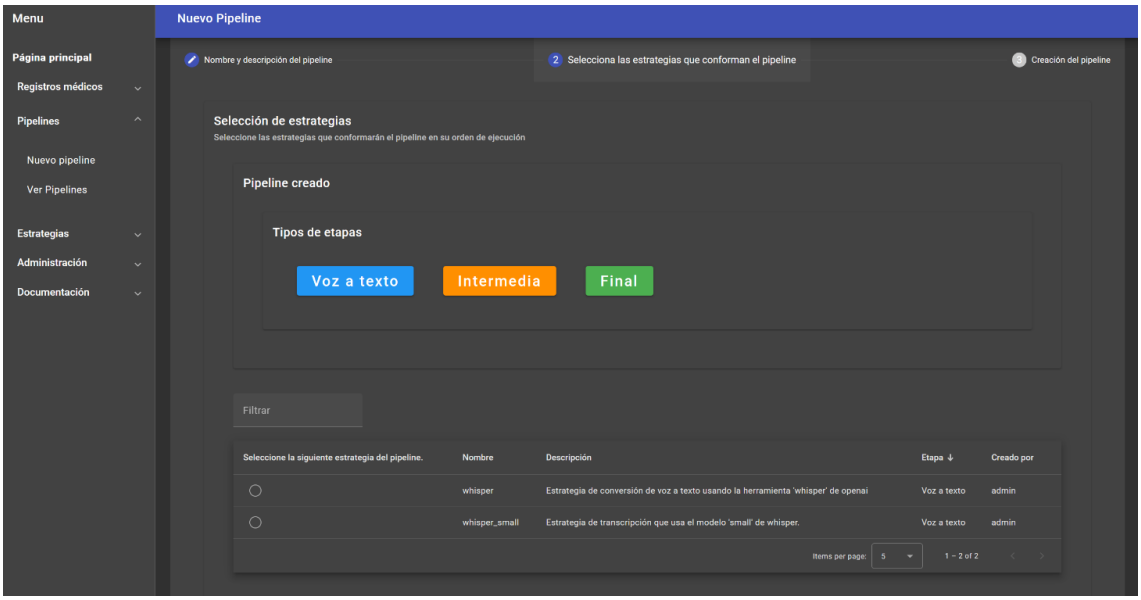

Ilustración 5.1-3 - Selección de una estrategia de tipo "Voz a texto".

Al seleccionar una estrategia de la tabla, ésta se actualiza, mostrando ahora estrategias de tipo intermedio y final. También se actualiza la vista, donde aparece la estrategia de voz a texto seleccionada, junto a un botón que permite su eliminación. La Ilustración 5.1-4 muestra el proceso de selección de estrategias tras haber seleccionado la primera estrategia.

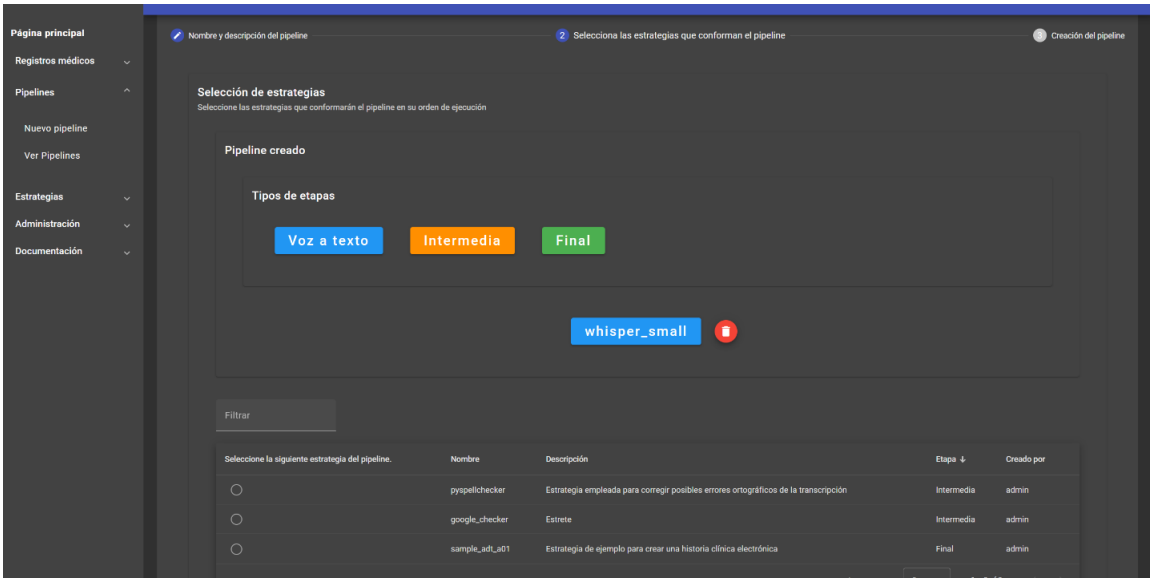

Ilustración 5.1-4 - Selección de una estrategia de tipo "Intermedia" o "Final".

Esta etapa concluye cuando se elige una estrategia “Final”. Entonces, la tabla de selección de pipeline desaparece y se puede continuar al último paso. En la Ilustración 5.1-5 se muestra está última acción en el paso de selección de estrategias.

Ilustración 5.1-5 - Finalización de la selección de estrategias que conforman el pipeline.

Finalmente, aparece un resumen con lo seleccionado y un botón que concluye la operación de creación, visible en la Ilustración 5.1-6. El resultado de la creación se expone en la Ilustración 5.1-7.

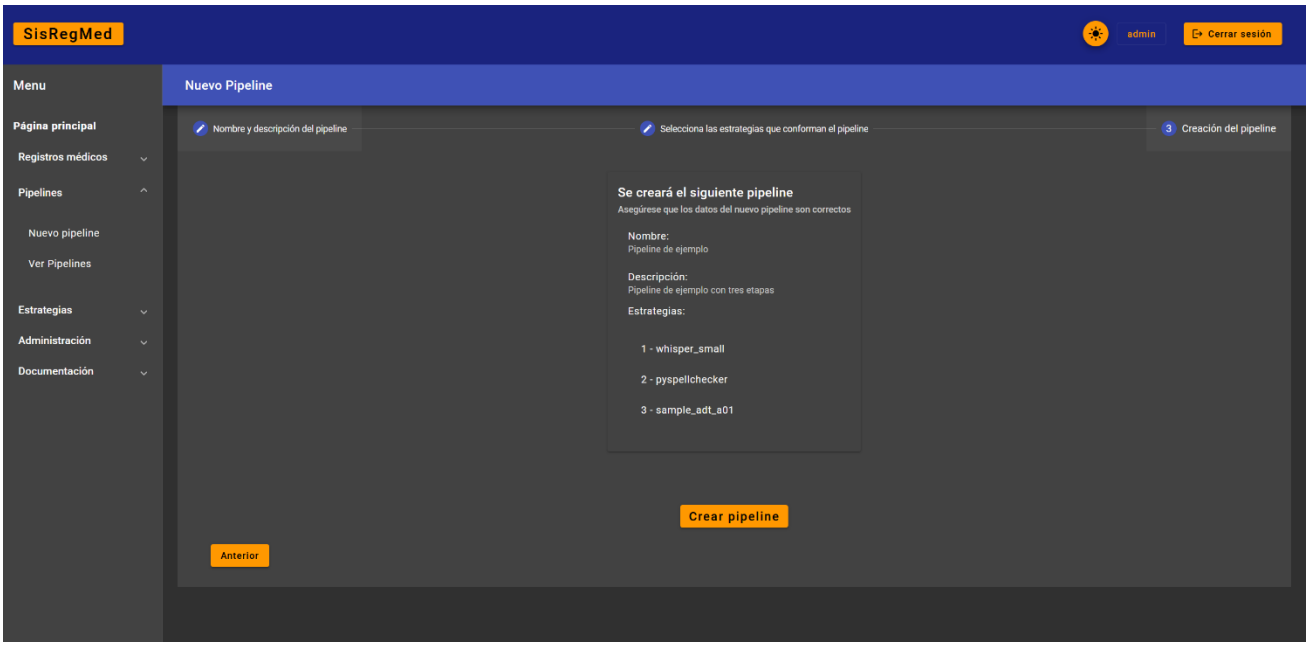

Ilustración 5.1-6 - Resumen de lo creado.

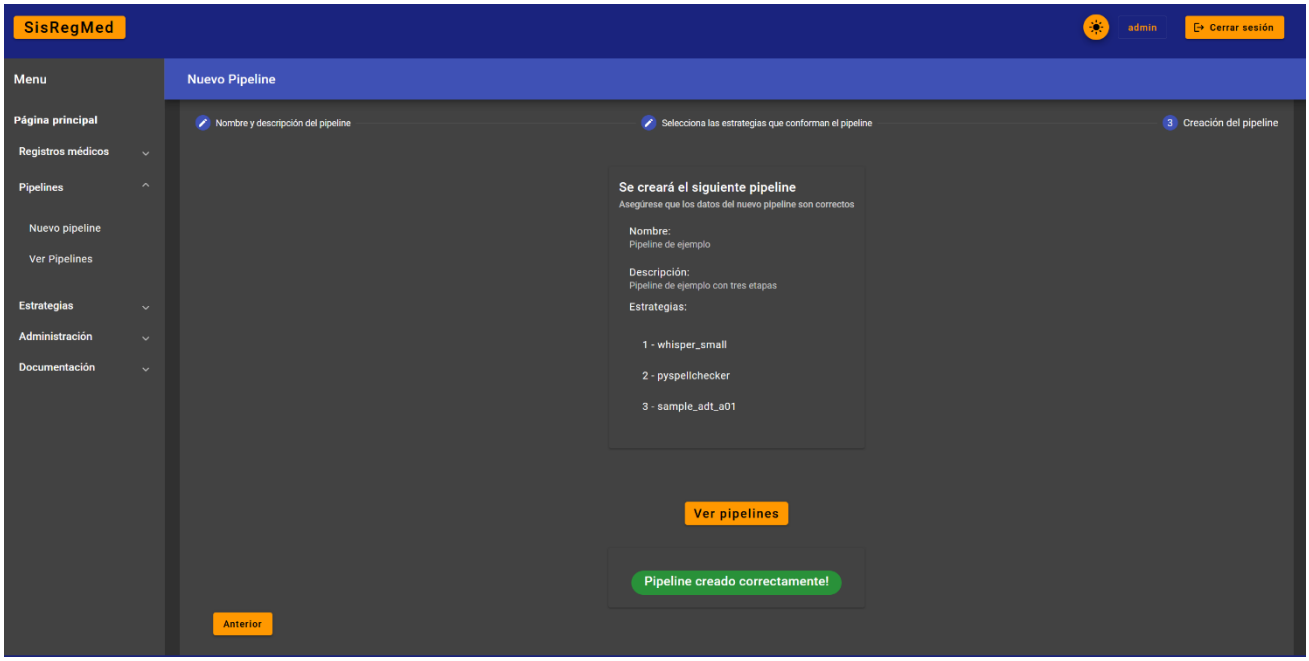

Ilustración 5.1-7 - Creación correcta del pipeline.

5.1.2 Vista y modificación de pipelines

Se prosigue comentando las funcionalidades provistas con respecto a la vista y modificación de pipelines. Al entrar en la página de vista de pipelines aparece una tabla, inicialmente no seleccionada (Ilustración 5.1-8). Contiene información relevante sobre los pipelines, como su nombre, descripción y etapas. El aspecto a destacar es el color que aparece en torno al nombre. Puede ser verde o rojo. En caso de ser rojo implica que una de las estrategias que conforman al pipeline no se ha encontrado registrada, es decir, es susceptible de haber sido borrada y, en consecuencia, sirve como indicación de que no se debe usar ese pipeline.

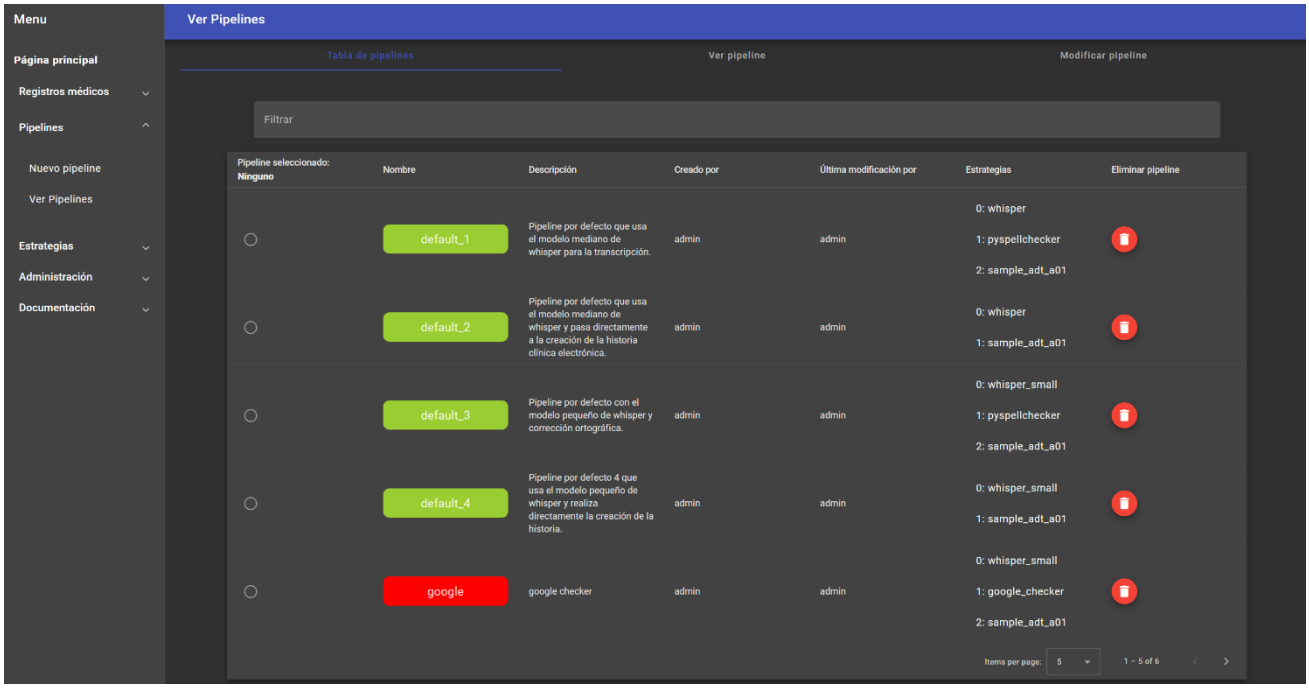

Ilustración 5.1-8 - Tabla de pipelines.

Adicionalmente, la tabla posee, en su parte inferior, una funcionalidad de selección del número de registros por página. Por defecto está configurado a 5. En la Ilustración 5.1-9 se establece a 10 para ver como aparece el pipeline creado en el apartado anterior.

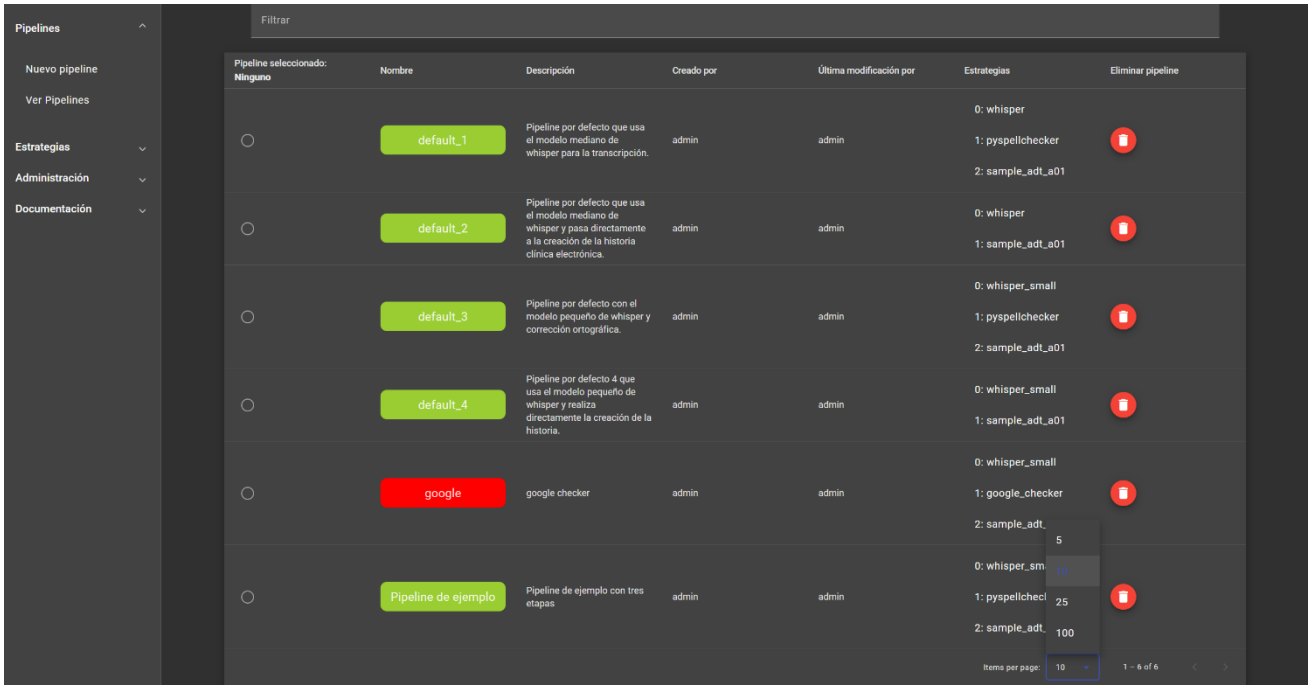

Ilustración 5.1-9 - Tabla de pipelines, mostrando un máximo de 10.

La vista de un pipeline concreto (Ilustración 5.1-10) se limita a mostrar la información relacionada con un pipeline: su nombre, descripción, estrategias, fechas de creación y modificación y usuarios encargados de la creación y modificación.

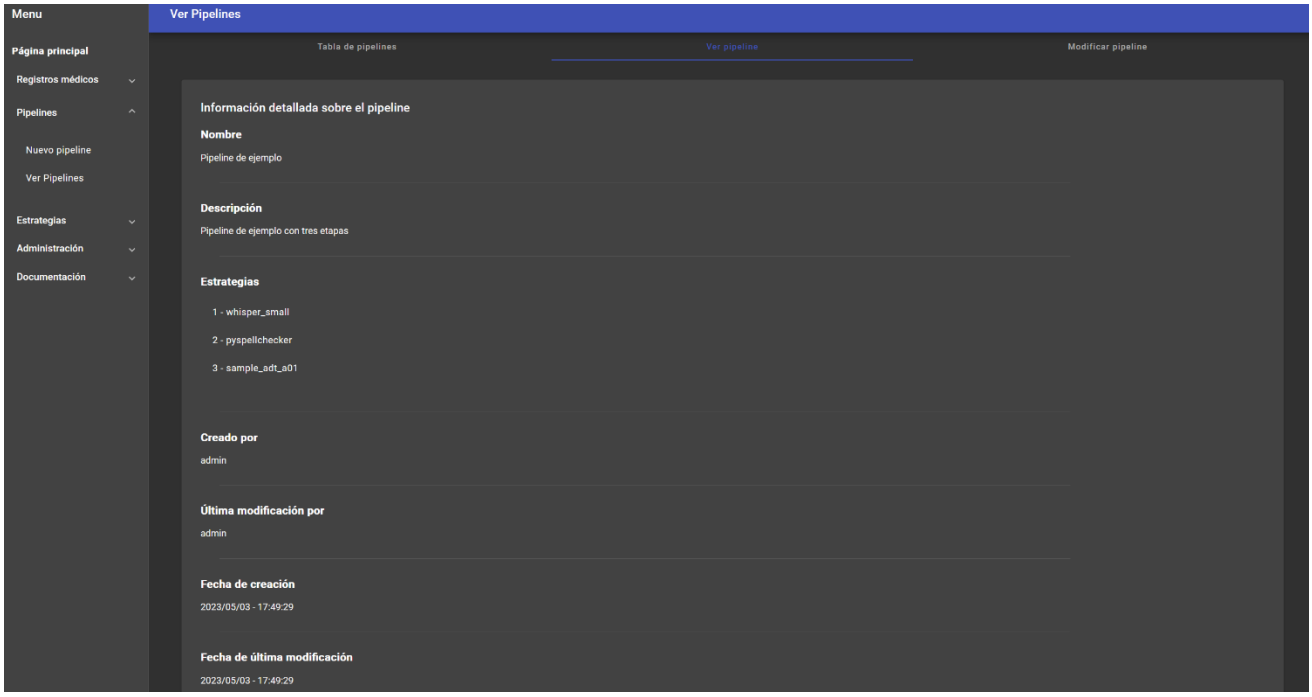

Ilustración 5.1-10 - Vista de la información detallada de un pipeline.

Adicionalmente, en el caso de tratarse de un pipeline susceptible de tener una estrategia eliminada, se muestra un mensaje de advertencia claro y diferenciado. Este ejemplo se muestra en la Ilustración 5.1-11.

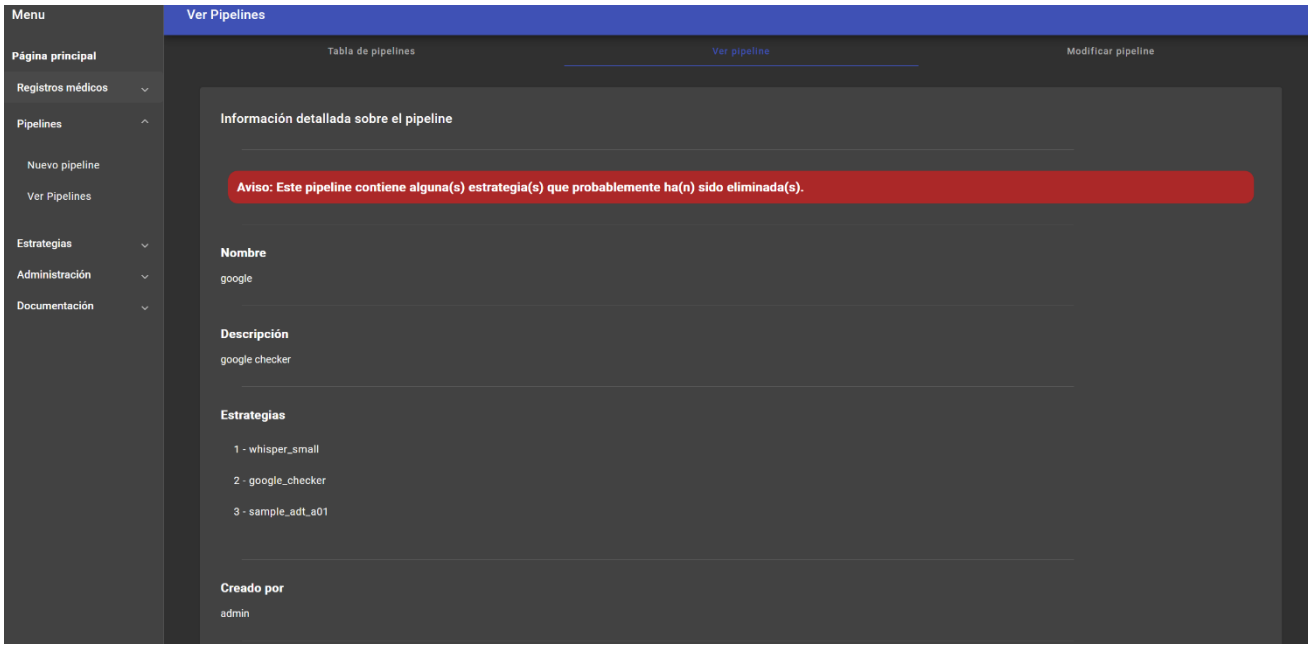

Ilustración 5.1-11 - Vista de un pipeline con un mensaje de advertencia.

Por último, se comenta acerca de la modificación de pipelines. La última pestaña contiene un formulario con la información del registro seleccionado. El formulario se puede modificar, teniendo como consecuencia la modificación del nombre o descripción del pipeline. La Ilustración 5.1-12 muestra la pestaña de modificación del pipeline, cuyo formulario aparece relleno con la información concreta del pipeline seleccionado.

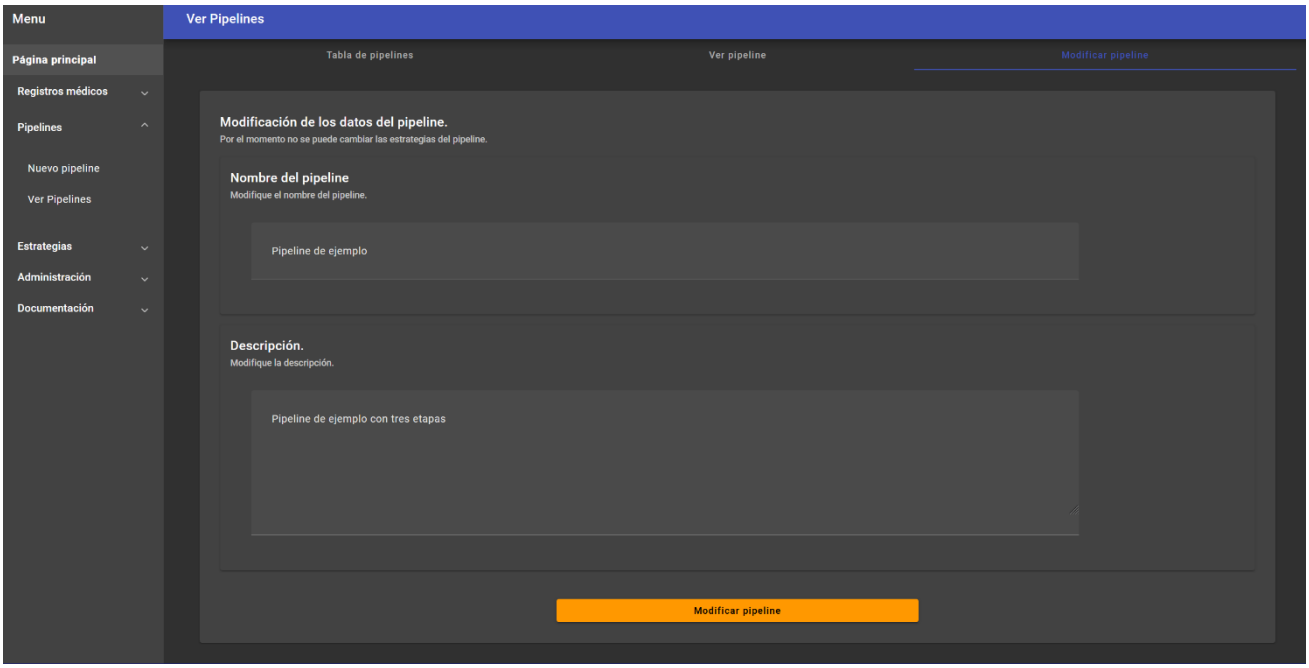

Ilustración 5.1-12 - Pestaña de modificación de pipeline.

La Ilustración 5.1-13 muestra el resultado de una modificación exitosa de los datos del pipeline.

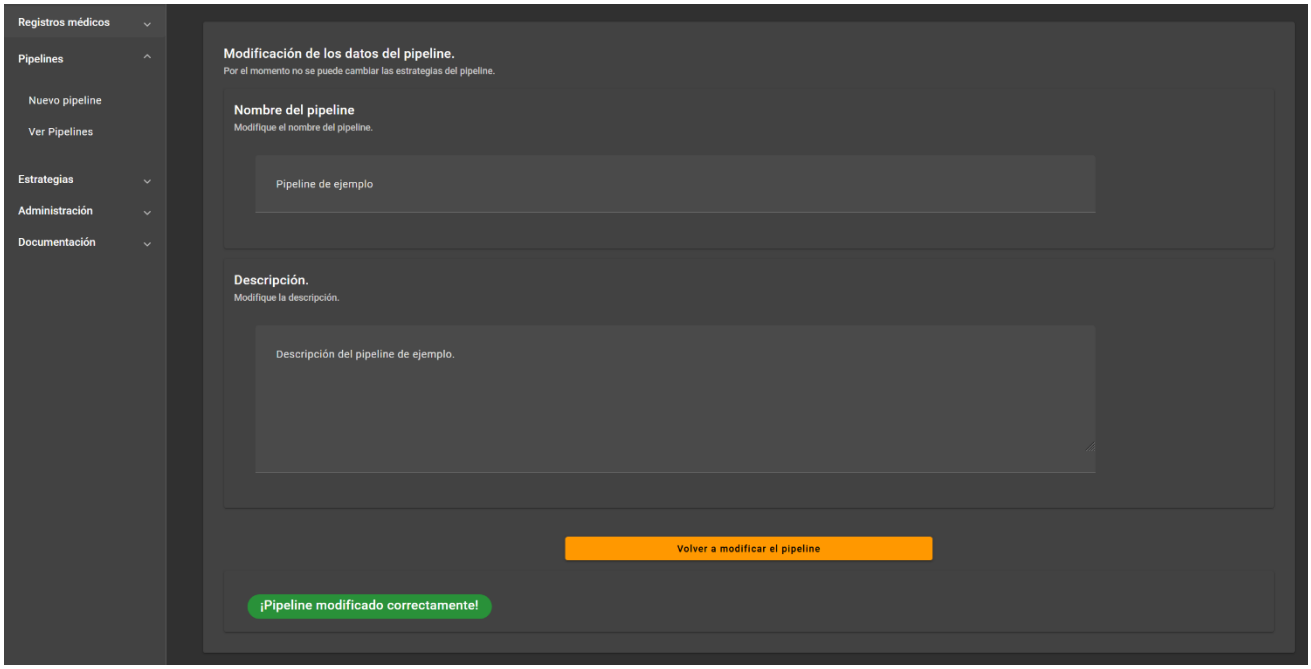

Ilustración 5.1-13 - Modificación exitosa de un pipeline.

En la Ilustración 5.1-14 se observa como los cambios son reflejados en la pestaña de vista concreta del pipeline.

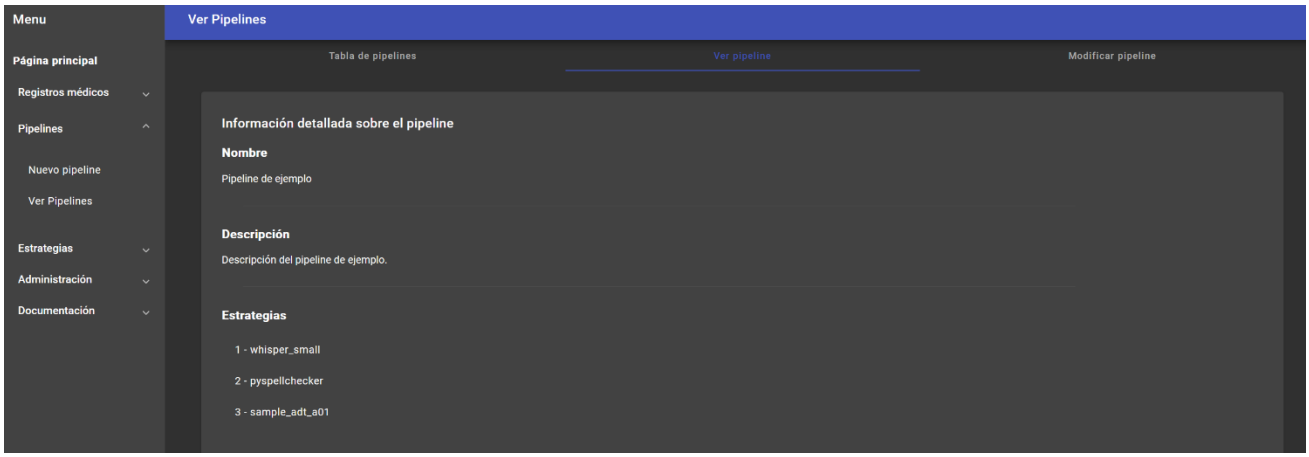

Ilustración 5.1-14 - Cambios reflejados tras la modificación del pipeline

5.1.3 Vista y modificación de registros

La vista y modificación de registros es muy similar a la de pipelines. Se trata de una página que cuenta con tres pestañas: una en la que aparece la tabla de registros, otra con la vista de los campos de un registro y una última donde se permite modificar el registro seleccionado. La Ilustración 5.1-15 muestra esta tabla, inicialmente no seleccionada.

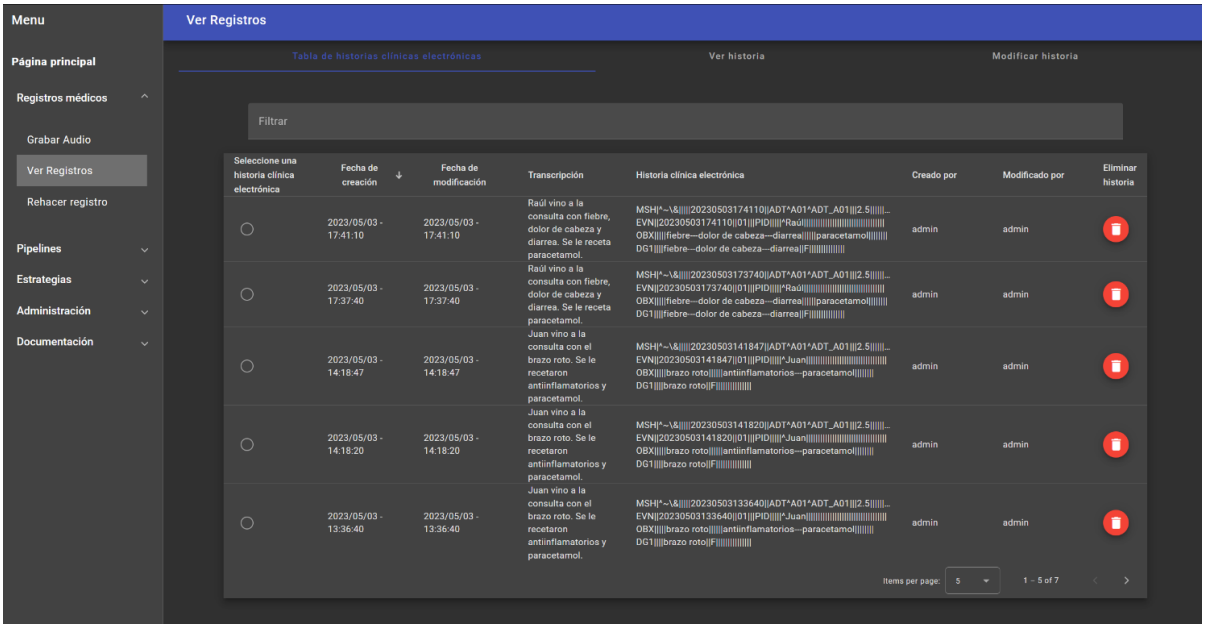

Ilustración 5.1-15 - Tabla de registros médicos.

De la vista concreta de un registro seleccionado se destaca la posible vista de una etapa de procesamiento concreta, como es el caso del reconocimiento de la entidad nombrada, donde se resaltan las entidades reconocidas y sus respectivas etiquetas. La Ilustración 5.1-16 muestra los datos más importantes que conforman el registro: la Historia Clínica Electrónica, la transcripción y el audio empleado.

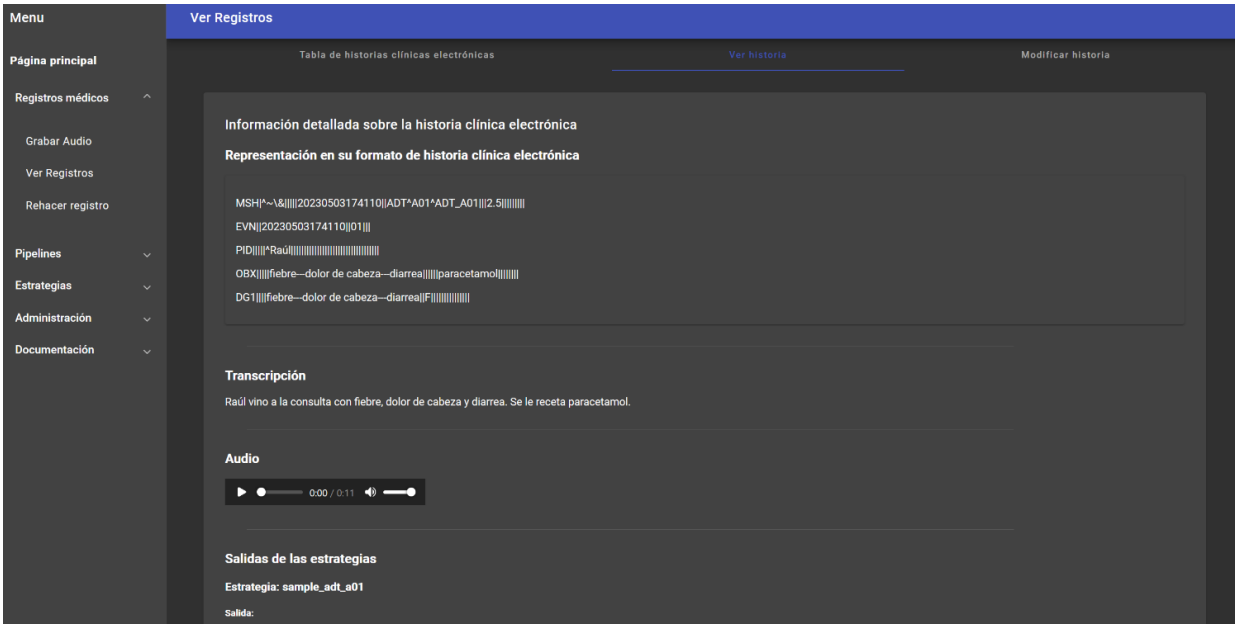

Ilustración 5.1-16 - Vista de la historia, transcripción y audio de un registro.

La Ilustración 5.1-17 muestra la sección que contiene las salidas de la última etapa de procesamiento. La salida principal es la Historia Clínica Electrónica mostrada. Adicionalmente, también muestra el reconocimiento de la entidad nombrada realizado en la etapa.

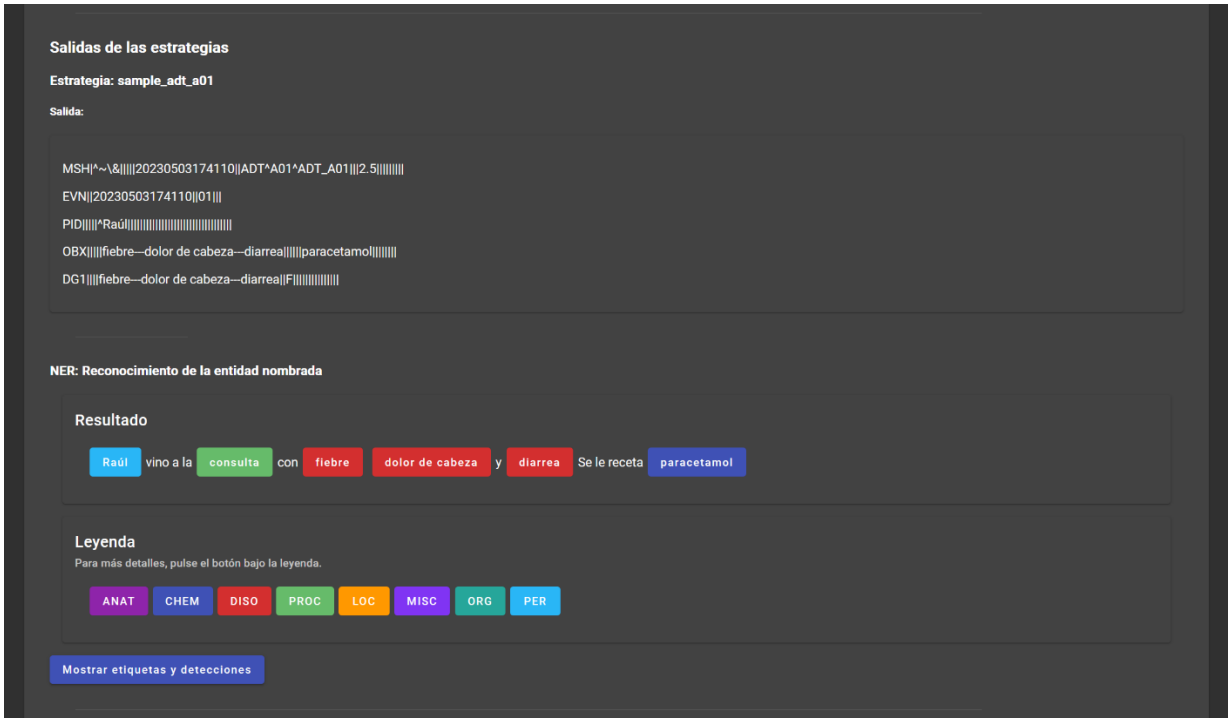

Ilustración 5.1-17 - Vista del reconocimiento de la entidad nombrada realizada sobre un registro.

La modificación de registros permite cambiar su Historia Clínica Electrónica y la transcripción generada. Esta transcripción modificada puede ser empleada posteriormente para crear nuevos registros. La Ilustración 5.1-18 muestra una modificación exitosa del registro, donde se añade “náusea” en la transcripción.

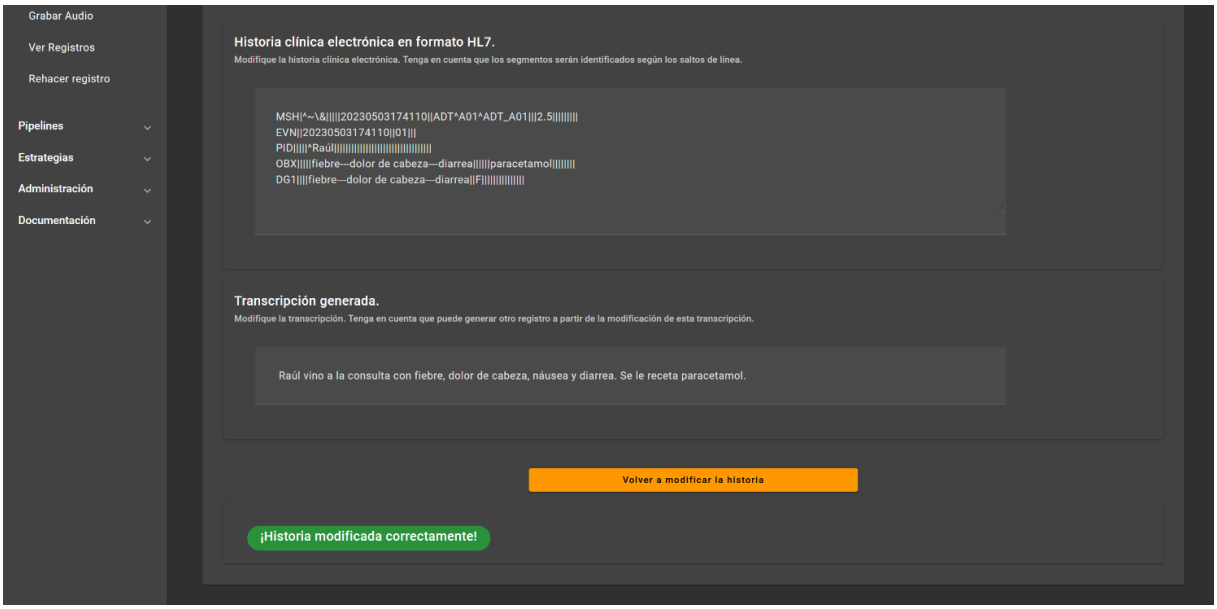

Ilustración 5.1-18 - Modificación de registros exitosa.

Tras modificar un registro, en su vista quedará marcado como modificado. Se pueden contrastar las modificaciones con las salidas de las estrategias. La Ilustración 5.1-19 muestra la transcripción actualizada y un aviso indicando el estado modificado del registro.

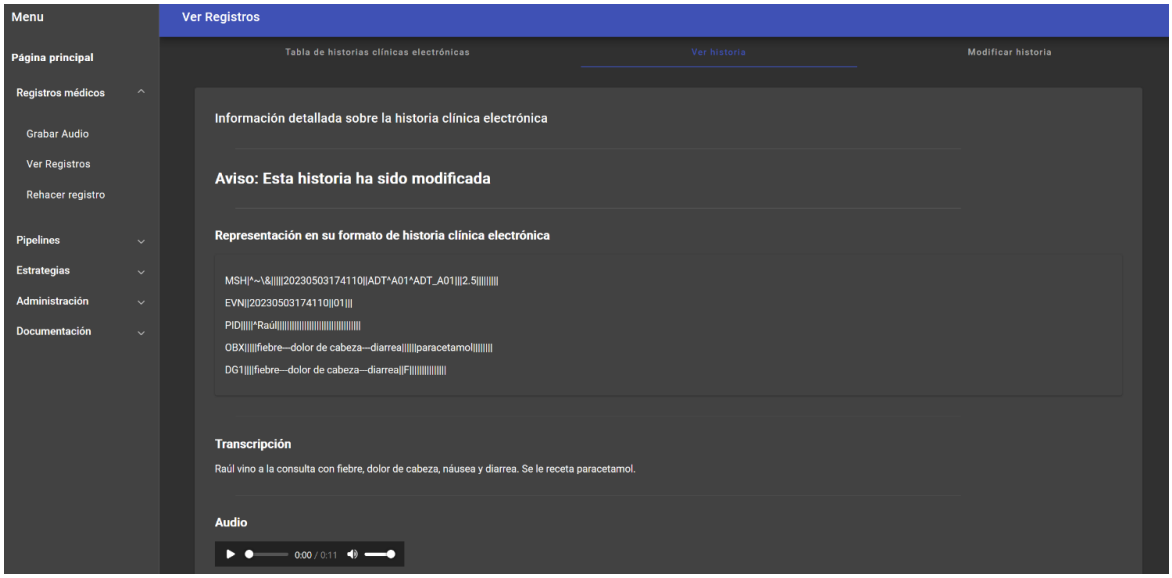

Ilustración 5.1-19 - Mensaje de modificación de un registro.

5.1.4 Vista y modificación de estrategias.

La vista de estrategias sigue el mismo estilo y formato: una tabla de selección seguida de una pestaña de vista concreta de la estrategia seleccionada. Sin embargo, se destaca la modificación de estrategias, donde además de permitir modificar la información a través de un formulario, se permite volver a subir el fichero "*.py*" que la compone. La única restricción es que las dependencias deben ser las mismas, puesto que no se modificará el entorno virtual. La Ilustración 5.1-20 muestra la pantalla de modificación de información relacionada con la estrategia.

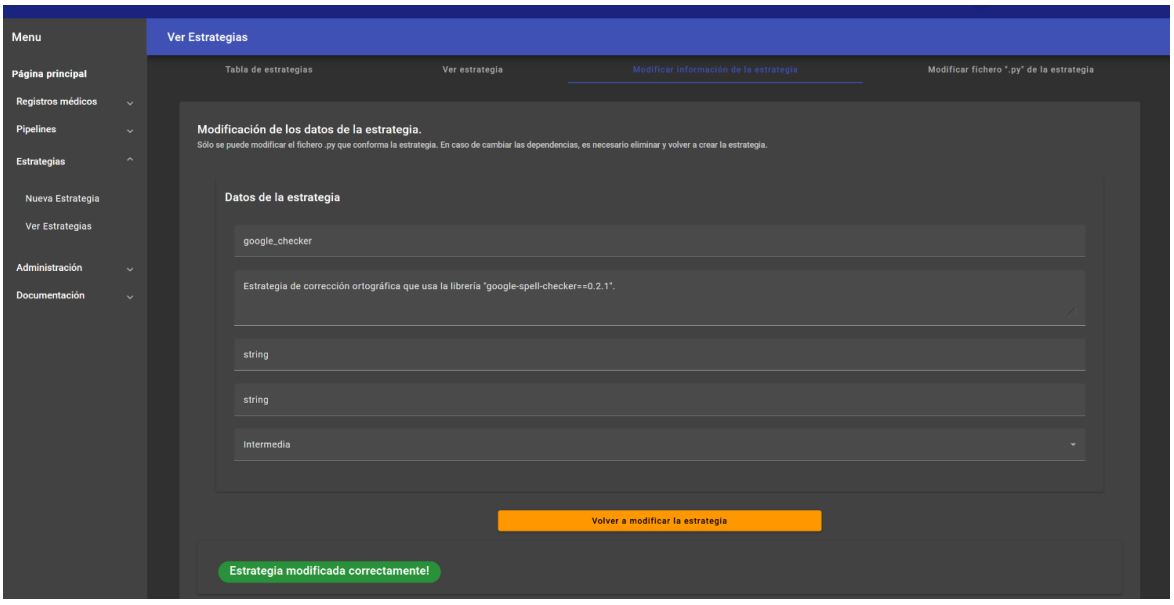

Ilustración 5.1-20 - Modificación de la información relativa a una estrategia.

Las ilustraciones 5.1-21 – 5.1-23 muestran el proceso de resubida del fichero "***.py***" empleado en la estrategia.

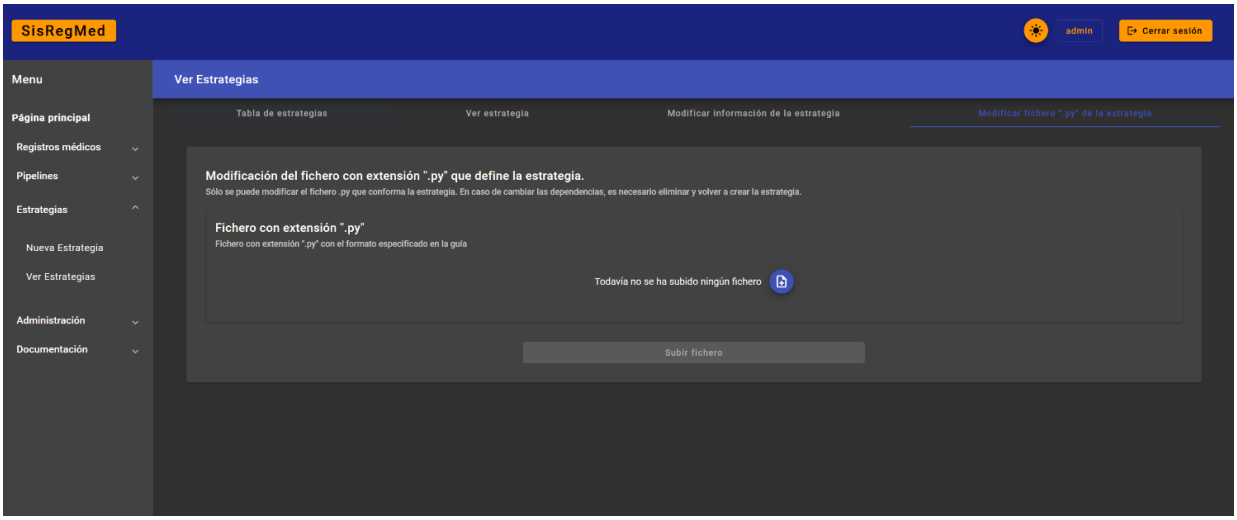

Ilustración 5.1-21 - Pantalla de subida del fichero ".py".

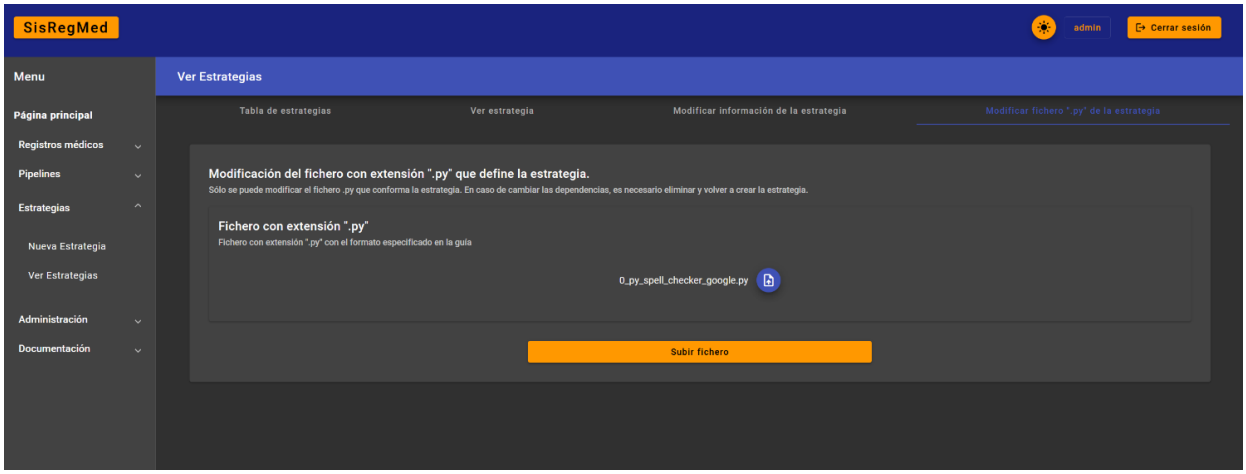

Ilustración 5.1-22 - Subida del fichero.

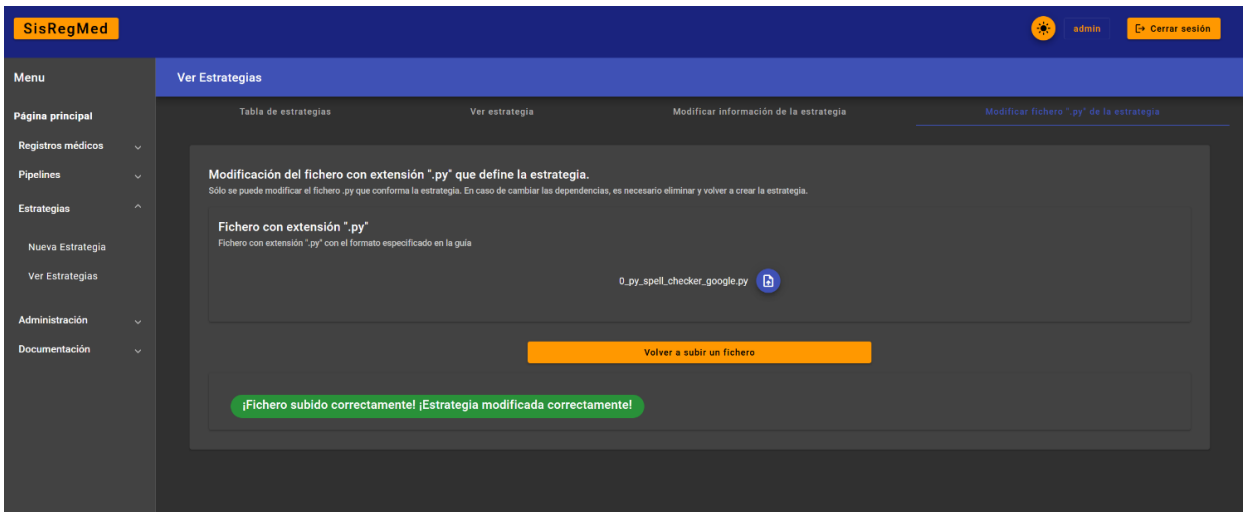

Ilustración 5.1-23 - Resultado de la actualización del fichero en el backend.

5.2 Documentación generada (Funcional y Técnica)

La documentación generada se puede encontrar en la propia vista del aplicativo Frontend. Se dispone de un total de cinco páginas.

Se incluye una introducción donde se define el marco del proyecto y se hace un resumen de éste. Se explica el concepto detrás del procesamiento y se definen las entidades de registros, estrategias y pipelines. También contiene un apartado donde se explica el resto de la documentación.

Existen un total de tres páginas de guías que sirven a modo de documentación funcional. Las páginas de guías describen en detalle cómo utilizar el aplicativo. Se explica la operativa de creación, vista y modificación de registros, pipelines y estrategias. Se incluyen ejemplos e imágenes.

Se agrega una página de código como documentación técnica, donde se explica el funcionamiento interno de los mecanismos de creación de estrategias y registros.

Por último, se agrega una página de documentación de pruebas, donde se detalla la estructura de directorios empleada en las pruebas y se describe el funcionamiento de la librería *pytest*, el uso de *fixtures* y su implementación.

Capítulo 6

6 Conclusiones, trabajos futuros y perspectiva

6.1 Conclusiones

El Trabajo de Fin de Grado ha consistido en la implementación de un prototipo de sistema de gestión de registros médicos que servirá como punto de partida para explorar y alcanzar objetivos más ambiciosos y mejores formas de optimizar y gestionar información sanitaria.

Durante la elaboración del proyecto se ha seguido la metodología Scrum, adaptándola al caso particular de este trabajo, logrando correcciones conceptuales y distintos puntos de vista en determinadas situaciones, permitiendo orientar el desarrollo en la dirección correcta antes de haber sido demasiado tarde.

Durante las fases de análisis y diseño se han aplicado técnicas de Ingeniería del Software como la realización de un análisis de requisitos para poder enfocar y definir el proyecto en el *qué* y posteriormente centrarse en el *cómo*.

Se ha cumplido correctamente con todos los objetivos previstos.

Desde un punto de vista técnico se han adquirido los siguientes conocimientos que, con total seguridad, serán útiles en el futuro:

- Diseño y posterior implementación de un sistema basado en el modelo cliente-servidor.
- Gestión de peticiones HTTP, manejo de sesiones y cookies y conexión de un aplicativo frontend con un backend.
- Afianzamiento, adición y adquisición de experiencia con respecto al funcionamiento del lenguaje de programación Python.
- Gestión de entornos virtuales y dependencias en Python.
- Ejecución de subprocesos en Python.
- Comunicación entre subprocesos serializando información.
- Uso de librerías adicionales de terceros en Angular.

6.2 Trabajos futuros

6.2.1 Aspectos relacionados con la seguridad

En el desarrollo del aplicativo se han identificado numerosas situaciones en las que, haciendo uso de software, tanto herramientas ya existentes como implementaciones propias, se podrían ofrecer medidas de protección adicionales. Sin embargo, este nivel programático de seguridad adicional ha sido omitido por quedarse fuera del alcance del proyecto, dadas las restricciones temporales. La agregación de estas medidas de seguridad podría ser un proyecto en sí mismo.

En primer lugar, la seguridad del sistema se puede mejorar desde el diseño de éste, realizando un análisis de riesgos muy detallado a nivel técnico, donde se comparen los distintos tipos de herramientas disponibles desde un punto de vista orientado a la seguridad. Tras haber realizado un análisis de riesgos, es importante revisarlo e identificar salvaguardas y medidas de protección adicionales para garantizar como mínimo las siguientes dimensiones de la seguridad: disponibilidad, autenticación, integridad, confidencialidad y trazabilidad.

Para garantizar la autenticación, se puede implementar un sistema acceso basado en certificados digitales y otros sistemas de autenticación, garantizando en la medida de lo posible que la información es vista por personal autorizado y evitando el repudio en las acciones realizadas.

En relación con la integridad, se debe establecer una política de copias de seguridad con carácter periódico, donde los datos sean almacenados en ubicaciones seguras. Se deben establecer políticas y procedimientos claros y bien definidos en relación con el empleo de las funcionalidades provistas por el aplicativo. Se deben utilizar técnicas de cifrado y resumen para verificar que la información transmitida es correcta.

Con respecto a la confidencialidad, tanto las comunicaciones como el contenido de la base de datos deberán estar sujetos a técnicas de cifrado y protocolos y políticas estrictas de acceso.

El aseguramiento de la disponibilidad pasa por agregar redundancia haciendo uso de herramientas de gestión de servidores como clústeres. El uso de clústeres permite funcionalidades como el balanceo de carga, para evitar la situación en la que un servidor recibe más peticiones de las que puede soportar, o una monitorización constante, capaz de aislar con efecto inmediato algún servidor sospechoso de haber sido comprometido.

La trazabilidad se refiere a la capacidad de rastrear un evento a lo largo del proceso. Para ello, se deben establecer sistemas de registro, donde queden guardado de manera sistemática y eficiente los accesos y operaciones realizados en el aplicativo.

Con respecto a la seguridad desde un punto de vista del diseño global, es crítico acotar las redes de comunicaciones en la que se alojan los servicios y hacer uso de cortafuegos y herramientas de análisis del tráfico para minimizar las posibilidades de infección.

El aspecto más importante es la identificación del tipo de información y contexto de tratamiento (si está relacionado o no con administraciones públicas) para darse de alta y cumplir con toda la normativa, legislación y protocolos provistos por el RGPD, LOPDYGDD, CCN, ENS, etc.

6.2.2 Paso a producción

Con perspectivas a un empleo de la herramienta en un entorno de producción, el sistema debería ser extendido y encapsulado.

Como se ha comentado anteriormente, se recuerda que el alcance del proyecto es limitado y tiene como objetivo principal servir a modo de prueba de concepto para demostrar que la idea de automatizar el registro y procesamiento de información sanitaria tiene potencial de ser explotado extensamente.

Debería haber un aplicativo de producción específico para el personal sanitario, otro para los perfiles de científico de datos y otro adicional para desarrolladores. De manera intrínsecamente relacionada con el apartado anterior de seguridad, una separación de sistemas permite facilitar el análisis, diseño y desarrollo en múltiples facetas, desde la seguridad, hasta la identificación de requisitos y funcionalidades adicionales especializadas en cada contexto.

El aplicativo de producción específico para el personal sanitario acotaría las funcionalidades y la posibilidad de ejecución y acceso a los datos únicamente a los requisitos del personal sanitario, dando un limitado acceso a la información a los científicos de datos. Los servidores de Frontend en este caso servirían unas páginas específicas, directamente eliminando los otros roles y con funcionalidades de empleo de claves criptográficas. Los servidores en el Backend constarían únicamente de la lógica necesaria para gestionar las solicitudes realizadas desde el punto de vista del personal sanitario y emplear las últimas tecnologías de procesamiento de datos provistas por los desarrolladores. La base de datos contaría con un plan de copias de seguridad y estaría cifrada.

Los científicos de datos tendrían también su propio aplicativo (Frontend, Backend y base de datos). De manera puntual, y tras haber pedido permiso, podrían solicitar registros de manera masiva desde el aplicativo del personal sanitario, donde se podrían implementar funcionalidades para garantizar la confidencialidad de la información. Por ejemplo, si se quiere hacer una estadística del número de pacientes atendidos en las distintas partes del hospital (19% traumatología, 10% neurología…), se pueden hacer consultas donde no se recupere la totalidad de los registros, sino las partes relevantes a la estadística, omitiendo así la información personal de los pacientes. Por supuesto, el aplicativo empleado por los científicos de datos tendría implementada una serie de funcionalidades específicas, así como su propia infraestructura de seguridad.

Por último, el aplicativo empleado por desarrolladores contaría con herramientas específicas para agregar y probar nuevas funcionalidades de manera cómoda, por ejemplo, el contraste de una nueva estrategia con una batería de ejemplos predeterminada y unas salidas esperadas. Adicionalmente, desde este aplicativo se controlarán las últimas versiones de la lógica de procesamiento empleada en el aplicativo de producción del personal sanitario.

Glosario de acrónimos

- **TFG**: Trabajo de Fin de Grado.
- **ITC**: Instituto Tecnológico de Canarias.
- **NLP**: *Natural Language Processing*.
- **EHR**: *Electronic Health Record*.
- **S2T**: *Speech to Text*.
- **IDE**: *Integrated Development Environment*.
- **HTTP**: *Hypertext Transfer Protocol*.
- **HL7**: *Health Level 7*.
- **UE**: Unión Europea.
- **RGPD**: Reglamento General de Protección de Datos.
- **HIPAA**: *Health Insurance Portability and Accountability Act*.
- **DICOM**: *Digital Imaging and Communications in Medicine*.
- **API**: *Application Programming Interface.*
- **UML:** *Unified Modeling Language.*
- **AWS:** *Amazon Web Services.*
- **ECS:** *Amazon Elastic Container Service.*
- **EC2:** *Amazon Elastic Compute Cloud.*
- **SPA:** *Single Page Application*
- **NPM:** *Node Package Manager*
- **YAML**: *Yet Another Markup Language*
- **CLI:** *Command Line Interface*
- **HTML:** *Hyper Text Markup Language*
- **CSS:** *Cascading Style Sheets*
- **ORM:** *Object Relational Mapping*
- **OOP:** *Object Oriented Programming*
- **JSON:** *Java Script Object Notation*
- **URL:** *Uniform Resource Locator*

7 Bibliografía

[1] HL7 International, «Home Page,» HL7 International, 2023. [En línea]. Disponible: https://www.hl7.org/. [Último acceso: 07 04 2023].

[2] Caristix, «HL7-Definition V2,» Caristix, [En línea]. Disponible: https://hl7-definition.caristix.com/v2/. [Último acceso: 07 04 2023].

[3] Wikipedia, «Health Level 7,» Wikipedia, 17 03 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Health_Level_7. [Último acceso: 07 04 2023].

[4] Wikipedia, «Prueba de concepto,» Wikipedia, 22 09 2022. [En línea]. Disponible: https://es.wikipedia.org/wiki/Prueba_de_concepto. [Último acceso: 07 04 2023].

[5] Google, «Angular Home Page,» Google, [En línea]. Disponible: https://angular.io/. [Último acceso: 16 04 2023].

[6] Wikipedia, «Angular (web framework),» Wikipedia, 11 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Angular_(web_framework). [Último acceso: 16 04 2023].

[7] Google, «Angular Material,» Google, [En línea]. Disponible: https://material.angular.io/. [Último acceso: 16 04 2023].

[8] Wikipedia, «Material Design,» Wikipedia, 12 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Material_Design. [Último acceso: 16 04 2023].

[9] V. a. Pallets, «Flask User's Guide,» Pallets, [En línea]. Disponible: https://flask.palletsprojects.com/en/2.2.x/. [Último acceso: 16 04 2023].

[10] Wikipedia, «Flask (web framework),» Wikipedia, 30 03 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Flask_(web_framework). [Último acceso: 16 04 2023].

[11] Wikipedia, «Microframework,» Wikipedia, 25 01 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Microframework. [Último acceso: 16 04 2023].

[12] Wikipedia, «Microframework (es),» Wikipedia, 8 10 2020. [En línea]. Disponible: https://es.wikipedia.org/wiki/Microframework. [Último acceso: 16 04 2023].

[13] OpenAI, «Whisper,» Github, 11 04 2023. [En línea]. Disponible: https://github.com/openai/whisper. [Último acceso: 16 04 2023].

[14] Docker, «Docker Home Page,» Docker, 2023. [En línea]. Disponible: https://www.docker.com/. [Último acceso: 16 04 2023].

[15] Wikipedia, «Docker (software),» Wikipedia, 04 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Docker_(software). [Último acceso: 16 04 2023].

[16] Microsoft, «Visual Studio Code Home Page,» Microsoft, 2023. [En línea]. Disponible: https://code.visualstudio.com/. [Último acceso: 16 04 2023].

[17] Wikipedia, «Visual Studio Code,» Wikipedia, 6 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Visual_Studio_Code. [Último acceso: 16 04 2023].

[18] Microsoft, «Developing inside a Container,» Microsoft, 30 03 2023. [En línea]. Disponible: https://code.visualstudio.com/docs/devcontainers/containers. [Último acceso: 16 04 2023].

[19] P. y. C. Europeo, «Reglamentos,» BOE, 27 04 2016. [En línea]. Disponible: https://www.boe.es/doue/2016/119/L00001-00088.pdf. [Último acceso: 16 04 2023].

[20] Wikipedia, «General Data Protection Regulation,» Wikipedia, 14 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/General_Data_Protection_Regulation. [Último acceso: 16 04 2023].

[21] John Snow Labs, «Spark NLP,» John Snow Labs, 2023. [En línea]. Disponible: https://nlp.johnsnowlabs.com/. [Último acceso: 16 04 2023].

[22] Wikipedia, «Spark NLP,» Wikipedia, 10 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Spark_NLP. [Último acceso: 16 04 2023].

[23] Explosion, «SpaCy Home Page,» Explosion, 2023. [En línea]. Disponible: https://spacy.io/. [Último acceso: 16 04 2023].

[24] Wikipedia, «spaCy,» Wikipedia, 16 01 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/SpaCy. [Último acceso: 16 04 2023].

[25] Google, «Using the Healthcare Natural Language API,» Google, 13 04 2023. [En línea]. Disponible: https://cloud.google.com/healthcare-api/docs/how-tos/nlp. [Último acceso: 16 04 2023].

[26] Azure, «Azure for healthcare,» Microsoft, 2023. [En línea]. Disponible: https://azure.microsoft.com/en-us/solutions/industries/healthcare/?WT.mc_id=ptx-azblog-dahouldi#customer-stories. [Último acceso: 16 04 2023].

[27] Hugging Face, «Home Page,» Hugging Face, 2023. [En línea]. Disponible: https://huggingface.co/. [Último acceso: 16 04 2023].

[28] Wikipedia, «Hugging Face,» Wikipedia, 15 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Hugging_Face. [Último acceso: 16 04 2023].

[29] lcampillos, «Roberta-es-clinical-trials-ner,» Hugging Face, [En línea]. Disponible: https://huggingface.co/lcampillos/roberta-es-clinical-trials-ner. [Último acceso: 16 04 2023].

[30] Meditech, «Home Page,» Meditech, 2023. [En línea]. Disponible: https://ehr.meditech.com/. [Último acceso: 16 04 2023].

[31] AthenaHealth, «Home Page,» AthenaHealth, 2023. [En línea]. Disponible: https://www.athenahealth.com/. [Último acceso: 16 04 2023].

[32] Real Academia Española, «Metodología,» RAE, [En línea]. Disponible: https://dle.rae.es/metodolog%C3%ADa. [Último acceso: 25 04 2023].

[33] Wikipedia, «Scrum (software development),» Wikipedia, 24 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Scrum_(software_development). [Último acceso: 25 04 2023].

[34] Scrum Manager, «Home Page,» Scrum Manager, 2023. [En línea]. Disponible: https://www.scrummanager.com/website/. [Último acceso: 25 04 2023].

[35] Wikipedia, «Scrum (desarrollo de software),» Wikipedia, 19 04 2023. [En línea]. Disponible: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software). [Último acceso: 25 04 2023].

[36] G. L. J. P. Alexander Menzinsky, Scrum Master, Lubaris, 2019.

[37] Wikipedia, «Use Case,» Wikipedia, 23 01 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Use_case. [Último acceso: 26 04 2023].

[38] Wikipediia, «Caso de uso,» Wikipedia, 25 10 2020. [En línea]. Disponible: https://es.wikipedia.org/wiki/Caso_de_uso. [Último acceso: 26 04 2023].

[39] Django Software Foundation, «Home Page,» Django Software Foundation, 2023. [En línea]. Disponible: https://www.djangoproject.com/. [Último acceso: 26 04 2023].

[40] Wikipedia, «Django (web framework),» Wikipedia, 17 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Django_(web_framework). [Último acceso: 26 04 2023].

[41] MariaDB Foundation, «Home Page,» MariaDB Foundation, 2023. [En línea]. Disponible: https://mariadb.org/. [Último acceso: 26 04 2023].

[42] Wikipedia, «MariaDB,» Wikipedia, 27 03 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/MariaDB. [Último acceso: 26 04 2023].

[43] CVE Details, «Mariadb: Security Vulnerabilities,» CVE Details, 2023. [En línea]. Disponible: https://www.cvedetails.com/vulnerability-list/vendor_id-12010/Mariadb.html. [Último acceso: 26 04 2023].

[44] DB-Engines, «Ranking,» DB-Engines, 2023. [En línea]. Disponible: https://db-engines.com/en/ranking. [Último acceso: 26 04 2023].

[45] Amazon, «About Amazon Web Services,» Amazon, 2023. [En línea]. Disponible: https://www.aboutamazon.com/what-we-do/amazon-web-services. [Último acceso: 26 04 2023].

[46] Amazon, «What is Amazon Elastic Container Service?,» Amazon, [En línea]. Disponible: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html. [Último acceso: 26 04 2023].

[47] Amazon, «Amazon Elastic Compute Cloud Documentation,» Amazon, [En línea]. Disponible: https://docs.aws.amazon.com/ec2/?icmpid=docs_homepage_compute#amazon-ec2. [Último acceso: 26 04 2023].

[48] Amazon, «Amazon EC2 pricing,» Amazon, [En línea]. Disponible: https://aws.amazon.com/ec2/pricing/on-demand/. [Último acceso: 26 04 2023].

[49] Amazon, «Free Cloud Computing Services - AWS Free Tier,» Amazon, [En línea]. Disponible: https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-

order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all. [Último acceso: 26 04 2023].

[50] Amazon, «Application and Network Traffic Distribution - Elastic Load Balancing Pricing - AWS,» Amazon, [En línea]. Disponible: https://aws.amazon.com/elasticloadbalancing/pricing/?nc=sn&loc=3. [Último acceso: 26 04 2023].

[51] Amazon, «Serverless Compute Engine–AWS Fargate Pricing–Amazon Web Services,» Amazon, [En línea]. Disponible: https://aws.amazon.com/fargate/pricing/. [Último acceso: 26 04 2023].

[52] Wikipedia, «Software architecture - Wikipedia,» Wikipedia, 22 02 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Software_architecture. [Último acceso: 27 04 2023].

[53] Wikipedia, «Client–server model - Wikipedia,» Wikipedia, 25 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Client%E2%80%93server_model. [Último acceso: 27 04 2023].

[54] D. E. Comer, Internetworking with TCP/IP Principles, Protocols, and Architectures - Fourth Edition, Upper Saddle River, New Jersey: Prentice Hall, 2000.

[55] Wikipedia, «Strategy pattern - Wikipedia,» Wikipedia, 03 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Strategy_pattern. [Último acceso: 27 04 2023].

[56] Docker, «Docker Hub Container Image Library | App Containerization,» Docker, [En línea]. Disponible: https://hub.docker.com/. [Último acceso: 04 05 2023].

[57] Docker, «node - Official Image | Docker Hub,» Docker, [En línea]. Disponible: https://hub.docker.com/_/node. [Último acceso: 04 05 2023].

[58] Docker, «python - Official Image | Docker Hub,» Docker, [En línea]. Disponible: https://hub.docker.com/_/python. [Último acceso: 04 05 2023].

[59] Docker, «postgres - Official Image | Docker Hub,» Docker, [En línea]. Disponible: https://hub.docker.com/_/postgres. [Último acceso: 04 05 2023].

[60] Docker, «Use Docker Compose | Docker Documentation,» Docker, [En línea]. Disponible: https://docs.docker.com/get-started/08_using_compose/. [Último acceso: 04 05 2023].

[61] Wikipedia, «YAML - Wikipedia,» Wikipedia, 04 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/YAML. [Último acceso: 04 05 2023].

[62] I. A. Santana, «isacas Repository,» Docker, [En línea]. Disponible: https://hub.docker.com/repositories/isacas. [Último acceso: 04 05 2023].

[63] The Linux Documentation Project, «/opt,» The Linux Documentation Project, [En línea]. Disponible: https://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/opt.html. [Último acceso: 04 05 2023].

[64] MomentJS Team, «moment - npm,» npmjs, 07 06 2022. [En línea]. Disponible: https://www.npmjs.com/package/moment. [Último acceso: 04 05 2023].

[65] M. Khan, «recordrtc - npm,» npmjs, 09 03 2021. [En línea]. Disponible: https://www.npmjs.com/package/recordrtc. [Último acceso: 04 05 2023].

[66] Python Software Foundation, «https://www.python.org/downloads/,» Python Software Foundation, [En línea]. Disponible: https://www.python.org/downloads/. [Último acceso: 05 05 2023].

[67] Microsoft, «Use the winget tool to install and manage applications | Microsoft Learn,» Microsoft, [En línea]. Disponible: https://learn.microsoft.com/en-us/windows/package-manager/winget/. [Último acceso: 05 05 2023].

[68] Wikipedia, «Chocolatey - Wikipedia,» Wikipedia, 28 04 2023. [En línea]. Disponible: https://en.wikipedia.org/wiki/Chocolatey. [Último acceso: 05 05 2023].

[69] Pallets, «Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (3.0.x),» Pallets, [En línea]. Disponible: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/. [Último acceso: 05 05 2023].

[70] M. Countryman, «Flask-Login — Flask-Login 0.7.0 documentation,» readthedocs, [En línea]. Disponible: https://flask-login.readthedocs.io/en/latest/. [Último acceso: 05 05 2023].

[71] Holger krekel and pytest-dev team, «Full pytest documentation — pytest documentation,» pytest.org, [En línea]. Disponible: https://docs.pytest.org/en/7.1.x/contents.html. [Último acceso: 05 05 2023].

[72] Pallets, «Testing Flask Applications — Flask Documentation (2.2.x),» Pallets, [En línea]. Disponible: https://flask.palletsprojects.com/en/2.2.x/testing/. [Último acceso: 05 05 2023].

[73] R. B. Pardo, «Trabajo Fin de Grado,» Universidad Autónoma de Madrid, Junio 2017. [En línea]. Disponible: https://repositorio.uam.es/bitstream/handle/10486/679808/blazquez_pardo_roberto_tfg.pdf?sequence=1. [Último acceso: 07 04 2023].

Anexo

En este anexo se presenta el conjunto de cuadros respectivo a las historias de usuario identificadas.

Cuadro 2 - HU 01.

ID	HU 01
Título	Realizar grabación
Rol	Personal sanitario
Descripción	El profesional sanitario podrá realizar una grabación para determinar si será objecto o no de procesamiento con la finalidad de crear un nuevo registro.
Criterios de validación	• Interfaz simple e intuitiva que indique claramente cómo grabar, el hecho de que se está grabando y la finalización del proceso de grabación. • Se permite reproducir la grabación una vez finalizada.

Cuadro 3 - HU 02.

ID	HU 02
Título	Selección de etapas de procesamiento
Rol	Personal sanitario
Descripción	El profesional sanitario podrá seleccionar un *pipeline* con una serie de etapas de procesamiento que serán aplicadas al audio grabado para crear un nuevo registro.
Criterios de validación	• Los *pipelines* aparecen en una tabla. • Se puede filtrar y ordenar la tabla. • La selección de un *pipeline* es sencilla e intuitiva.

Cuadro 4 - HU 03.

ID	HU 03
Título	Creación de un registro de a partir de una grabación
Rol	Personal sanitario
Descripción	El profesional sanitario podrá crear un nuevo registro a partir de la grabación realizada y el conjunto de etapas de procesamiento seleccionado.
Criterios de validación	• Antes de crear el registro debe aparecer la grabación (deberá ser reproducible) y el *pipeline* seleccionado. • Tras seleccionar el botón de creación debe haber alguna indicación de que el registro se está creando. • Al finalizar, se mostrará el resultado del procesamiento.

Cuadro 5 - HU 04.

ID	HU 04
Título	Ver registros
Rol	Personal sanitario
Descripción	El profesional sanitario podrá ver la lista de registros existente.
Criterios de validación	• Los registros deben aparecer en una tabla cuyos campos puedan ser ordenados en orden ascendente o descendente. • La taba debe poder filtrarse por texto. • Debe aparecer la fecha del registro. • Debe aparecer la transcripción y resultado del procesamiento (HL7)

Cuadro 6 - HU 05.

ID	HU 05
Título	Ver registro concreto
Rol	Personal sanitario
Descripción	El profesional sanitario podrá seleccionar un registro de la tabla para ver el registro en detalle.
Criterios de validación	• Debe poder seleccionarse el registro de manera intuitiva. • Se debe poder acceder a una vista del registro. • Debe aparecer la grabación empleada para crear el registro y ser reproducida. • Debe aparecer la historia clínica electrónica y la transcripción. • Deben aparecer los resultados del procesamiento implicados en la creación del registro. • Debe aparecer información adicional del registro, como el usuario creador de éste, fecha de creación o fecha de última modificación.

Cuadro 7 - HU 06.

ID	HU 06
Título	Modificación de registro
Rol	Personal sanitario
Descripción	El profesional sanitario podrá modificar la transcripción de un registro, o la historia clínica resultante, para corregir posibles errores que hayan tenido lugar durante el procesamiento.
Criterios de validación	• Interfaz intuitiva para realizar las modificaciones • Al volver a la vista del registro, los cambios serán reflejados de manera inmediata.

Cuadro 8 - HU 07.

ID	HU 07
Título	Eliminar registro
Rol	Personal sanitario
Descripción	El profesional sanitario podrá eliminar un registro.
Criterios de validación	• Interfaz intuitiva para eliminar el registro. • Antes de eliminar el registro debe aparecer un mensaje de confirmación.

Cuadro 9 - HU 08.

ID	HU 08
Título	Seleccionar salida de etapa de procesamiento
Rol	Personal sanitario
Descripción	El profesional sanitario podrá seleccionar la salida de una etapa de procesamiento de un registro ya existente para crear un registro a partir de otro.
Criterios de validación	• Las salidas deben aparecer identificadas y visibles bajo el nombre de la etapa de procesamiento empleada en la generación de esta. • Interfaz de selección simple e intuitiva. • Al seleccionar una salida, debe quedar claro que ha sido seleccionada correctamente.

Cuadro 10 - HU 09.

ID	HU 09
Título	Selección de etapa de procesamiento de un *pipeline*
Rol	Personal sanitario
Descripción	El profesional sanitario podrá seleccionar una etapa de procesamiento de un pipeline ya existente, a partir de la cual se realizará el procesamiento para crear un nuevo registro a partir de otro.
Criterios de validación	• Interfaz simple e intuitiva. • Al seleccionar una etapa debe quedar claro que ha sido seleccionada correctamente.

Cuadro 11 - HU 10.

ID	HU 10
Título	Creación de un registro a partir de otro
Rol	Personal sanitario
Descripción	El profesional sanitario podrá crear un registro a partir de otro, habiendo seleccionado anteriormente la salida de una etapa del procesamiento del registro padre y las etapas de procesamiento que se efectuarán en el nuevo registro hijo.
Criterios de validación	• Antes de proceder con la creación del registro, debe aparecer un resumen de lo seleccionado, es decir, lo que se tomará como entrada y las etapas implicadas en su procesamiento. • Al seleccionar la opción de creación de registros, se debe hacer ver al usuario que el proceso se ha iniciado.

Cuadro 12 - HU 11.

ID	HU 11
Título	Creación de *pipelines*
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá crear *pipelines* para crear registros médicos.
Criterios de validación	• Interfaz intuitiva • Deben poder seleccionarse las estrategias existentes. • Antes de finalizar la creación, debe aparecer un resumen del pipeline que se va a crear.

Cuadro 13 - HU 12.

ID	HU 12
Título	Eliminar pipelines
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá eliminar un pipeline.
Criterios de validación	• Interfaz intuitiva para eliminar el pipeline. • Antes de eliminar el pipeline debe aparecer un mensaje de confirmación.

Cuadro 14 - HU 13.

ID	HU 13
Título	Modificar pipelines
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá modificar un pipeline para cambiar las etapas de procesamiento que lo componen.
Criterios de validación	• Interfaz intuitiva • Los cambios deben verse reflejados de manera inmediata.

Cuadro 15 - HU 14.

ID	HU 14
Título	Ver pipelines
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá ver una lista de todos los pipelines existentes.
Criterios de validación	• Los pipelines aparecerán en una tabla. • La tabla podrá ser ordenada de manera ascendente o descendente según sus campos • La tabla debe ser filtrada. • Debe aparecer la fecha de creación del pipeline. • Deben aparecer las estrategias que componen al pipeline.

Cuadro 16 - HU 15.

ID	HU 15
Título	Ver pipeline concreto
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá ver en detalle el contenido de un pipeline, es decir, las estrategias que lo componen.
Criterios de validación	• Debe ser visible desde la tabla.

Cuadro 17 - HU 16.

ID	HU 16
Título	Crear estrategia
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá crear una estrategia para agregar nuevas posibles implementaciones concretas de una etapa de procesamiento.
Criterios de validación	• La interfaz de subida de fichero debe ser clara e intuitiva. • Debe haber un mensaje de información comunicando la correcta subida e integración de la estrategia.

Cuadro 18 - HU 17.

ID	HU 17
Título	Eliminar estrategia
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá eliminar una estrategia.
Criterios de validación	• Interfaz intuitiva para eliminar la estrategia. • Antes de eliminar el registro debe aparecer un mensaje de confirmación.

Cuadro 19 - HU 18.

ID	HU 18
Título	Modificar estrategia
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá modificar la información relacionada con la estrategia, como su nombre o descripción, pero no el código en sí. Para cambiar el código será necesario crear una nueva estrategia.
Criterios de validación	• Interfaz clara e intuitiva. • Los cambios deben verse reflejados de manera inmediata.

Cuadro 20 - HU 19.

ID	HU 19
Título	Ver estrategias
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá ver una lista de estrategias existentes.
Criterios de validación	• Las estrategias aparecerán en una tabla. • La tabla podrá ser ordenada de manera ascendente o descendente según sus campos • La tabla debe ser filtrada. • Debe aparecer la fecha de creación de la estrategia. • Debe aparecer el autor de la estrategia. • Debe aparecer la etapa a la que pertenece la estrategia (voz a texto, intermedia, final).

Cuadro 21 - HU 20.

ID	HU 20
Título	Ver estrategia concreta
Rol	Científico de datos/desarrollador
Descripción	El científico de datos/desarrollador podrá ver la información específica asociada a una estrategia concreta.
Criterios de validación	• Interfaz clara e intuitiva. • Deben aparecer todos los campos asociados a la estrategia.

Cuadro 22 - HU 21.

ID	HU 21
Título	Ver usuarios
Rol	Administrador
Descripción	El administrador podrá ver una lista con todos los usuarios que se han dado de alta en el sistema.
Criterios de validación	• Los usuarios aparecerán en una tabla. • La tabla podrá ser ordenada de manera ascendente o descendente según sus campos • La tabla debe ser filtrada. • Debe aparecer la fecha de alta del usuario. • Debe aparecer el nombre del usuario.

Cuadro 23 - HU 22.

ID	HU 22
Título	Ver usuario concreto
Rol	Administrador
Descripción	El administrador podrá ver toda la información registrada relativa a un usuario.
Criterios de validación	• Interfaz clara e intuitiva. • Fácil selección de usuarios desde la tabla.

Cuadro 24 - HU 23.

ID	HU 23
Título	Modificar usuario
Rol	Administrador
Descripción	El administrador podrá modificar la información registrada de un usuario para labores administrativas, como un cambio forzoso de contraseña o solicitud de cambio de nombre.
Criterios de validación	• Interfaz clara e intuitiva • Los cambios son reflejados de manera inmediata.

Cuadro 25 - HU 24.

ID	HU 24
Título	Eliminar usuario
Rol	Administrador
Descripción	El administrador podrá eliminar la información relativa a un usuario para cesar su uso del aplicativo.
Criterios de validación	• Interfaz intuitiva para eliminar al usuario. • Antes de realizar la eliminación debe aparecer un mensaje de confirmación.