



**UNIVERSIDAD
DE GRANADA**

**TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA**

Software para el uso de redes neuronales en el análisis de imágenes médicas

Aplicación basada en arquitectura cliente-servidor

Autor

Alejandro Ruiz Salazar

Directores

Alberto Durán López

María Bermudez Edo



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN**

Granada, 16 de junio de 2025

Software para el uso de redes neuronales en el análisis de imágenes médicas: Aplicación basada en arquitectura cliente-servidor

Alejandro Ruiz Salazar

Palabras clave: Inteligencia Artificial, Análisis Automatizado, Radiología Digital, Segmentación de Imágenes, Arquitectura Cliente-Servidor, Diagnóstico Clínico, Software Médico, Asistente Automatizado, Asistente Médico.

Resumen

Este trabajo presenta el desarrollo de una aplicación basada en inteligencia artificial (IA) para el análisis automatizado de imágenes, enfocada especialmente en la segmentación de imágenes médicas. La solución implementa una arquitectura cliente-servidor, utilizando Django para la gestión del backend, FastAPI para la ejecución de algoritmos avanzados de IA, y Docker para garantizar un despliegue flexible y escalable.

La herramienta diseñada permite cargar imágenes médicas, procesarlas mediante modelos específicos de segmentación basados en IA y generar resultados visuales que apoyan el diagnóstico clínico. Además, incorpora funcionalidades para la gestión interactiva de historiales y resultados de pacientes.

La validación técnica se lleva a cabo mediante pruebas tanto automáticas como manuales, asegurando su efectividad y estabilidad. El objetivo del proyecto es facilitar diagnósticos más rápidos y precisos, mejorar la calidad asistencial y promover un acceso equitativo a tecnologías avanzadas en centros médicos de cualquier tamaño.

Software for the Use of Neural Networks in Medical Image Analysis: Application Based on a Client-Server Architecture

Alejandro Ruiz

Keywords: Artificial Intelligence, Automated Analysis, Digital Radiology, Image Segmentation, Client-Server Architecture, Clinical Diagnosis, Medical Software, Automated Assistant, Medical Assistant.

Abstract

This work presents the development of an application based on artificial intelligence (AI) for the automated analysis of medical images, specifically focused on the analysis of radiographs. The solution implements a client-server architecture, using technologies such as Django for backend management, FastAPI for efficient execution of advanced AI algorithms, and Docker to ensure flexible and scalable deployment.

The designed tool allows medical images to be uploaded, processed through specific AI-based segmentation models, and generates clear visual results to support clinical diagnosis. In addition, it incorporates features for interactive management of patient histories and results.

Technical validation is carried out through both automated and manual testing, ensuring effectiveness and stability. The project aims to enable faster and more accurate diagnoses, improve the quality of care, and provide equitable access to advanced technologies in medical centers of any size.

Yo, **Alejandro Ruiz Salazar**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77557787Q, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alejandro Ruiz Salazar

Granada, a 16 de junio de 2025

D. **Alberto Durán López**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D. **María Bermudez Edo**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Software para el uso de redes neuronales en el análisis de imágenes médicas, Aplicación basada en arquitectura cliente-servidor*, ha sido realizado bajo su supervisión por **Alejandro Ruiz Salazar**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada, a 16 de junio de 2025.

Los directores:

Alberto Durán López María Bermudez Edo

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que han contribuido, directa o indirectamente, a la realización de este Trabajo de Fin de Grado.

En primer lugar, agradezco a mi tutor académico por su guía, disponibilidad y orientación durante todas las fases del proyecto. Su apoyo incondicional, a pesar de las adversidades y problemas surgidos por el proyecto o por factores externos, ha facilitado mucho el desarrollo correcto del mismo. También quiero agradecer a mi tutora por su disposición a cotutorizar el proyecto y hacerlo posible.

También me gustaría dar las gracias a los profesores del Grado en Ingeniería Informática por los conocimientos impartidos a lo largo de estos años, que han sido fundamentales para afrontar este proyecto con confianza y criterio.

A nivel personal, agradezco profundamente el apoyo de mi familia, amigos y pareja, cuyo apoyo constante me ha motivado en los momentos de mayor esfuerzo.

Por último, gracias a todas las personas que han ofrecido su tiempo para probar la aplicación (Elena, esta va por ti), dar feedback o compartir ideas. Sin su colaboración, este trabajo no habría sido posible en su forma actual.

Índice

1. Contextualización	19
1.1. Introducción	19
1.2. Estado del Arte	21
1.3. Consideraciones sobre Protección de Datos	25
2. Especificación del Sistema	27
2.1. Requisitos	27
2.1.1. Requisitos Funcionales	27
2.1.2. Requisitos No Funcionales	28
2.2. Casos de Uso	30
2.2.1. Desglose de los Casos de Uso	31
2.2.2. Funcionalidades Previstas	36
3. Diseño	39
3.1. Principios y criterios de diseño	39
3.2. Patrones de diseño utilizados	40
3.3. Diagrama de Clases UML	41
4. Arquitectura del Sistema	43
4.1. Tecnologías Seleccionadas	43
4.1.1. Django (backend web)	43
4.1.2. HTML, CSS y JavaScript (frontend)	43
4.1.3. FastAPI (servidor de inferencia)	44
4.1.4. Docker y Docker Compose	44
4.1.5. SQLite (base de datos)	44
4.1.6. Otras tecnologías	44
4.2. Arquitectura Propuesta	45
4.3. Clases y Base de Datos	47
4.3.1. Clase <code>ProcessedImage</code>	47
4.3.2. Clase <code>User</code>	48
4.3.3. Justificación del Modelo	48
4.4. Vistas HTML	48
4.5. Interacción entre Frontend y Backend	51

4.5.1. Funciones relevantes del Backend	51
4.6. Seguridad	52
4.7. Servidor de Inferencia con FastAPI	52
4.7.1. Estructura del Servidor	52
4.7.2. Carga y Ejecución de Modelos	53
4.7.3. Funcionamiento del Endpoint	53
4.7.4. Validación y Control de Errores	53
4.7.5. Ventajas del Desacoplamiento	54
4.8. Dockerización y Despliegue	54
4.8.1. Persistencia de Datos	55
5. Validación y Pruebas	57
5.1. Pruebas Automáticas	57
5.1.1. Coverage	59
5.1.2. Pruebas Manuales	59
5.1.3. Cobertura de Casos de Error	60
5.1.4. Medición de Rendimiento	60
6. Planificación y Costes	61
6.1. Planificación y Metodología	61
6.2. Recursos Humanos	64
6.3. Recursos Materiales	64
6.4. Presupuesto	65
7. Conclusiones y Líneas Futuras	67
7.1. Conclusiones Generales	67
7.2. Limitaciones Detectadas	68
7.3. Líneas Futuras de Mejora	68
7.4. Futuro del proyecto	69
A. Guía de Uso y Administración del Sistema	71
A.1. Requisitos previos	71
A.2. Lanzamiento del sistema	72
A.2.1. Acceso al sistema	72
A.3. Acceso a la interfaz de administración	73
A.4. Modificación de datos desde la interfaz	73

Índice de cuadros

1.1. Resumen de soluciones IA para imágenes médicas	24
4.1. Campos del modelo <code>ProcessedImage</code>	47
5.1. Cobertura de código por módulo	59
6.1. Estimación de costes del proyecto	65

Índice de figuras

2.1. Diagrama de casos de uso	30
3.1. Diagrama de clases UML general del patrón Estrategia. . . .	41
3.2. Diagrama de clases; paquete de Django.	42
3.3. Diagrama de clases UML. Se muestran 3 estrategias diferentes: ColumnaStrategy, SAMMed2DStrategy y FakeStrategy; paquete de inferencia	42
4.1. Diagrama de arquitectura	46
4.2. Vista de inicio de sesión	49
4.3. Vista principal	50
4.4. Vista del historial	51
6.1. Diagrama de Gantt en Marzo	61
6.2. Diagrama de Gantt en Abril	62
6.3. Diagrama de Gantt en Mayo	62
6.4. Diagrama de Gantt en Junio	63
6.5. Metodología en Cascada	64

Capítulo 1

Contextualización

1.1. Introducción

El avance de la inteligencia artificial (IA) en el ámbito sanitario ha experimentado un crecimiento exponencial en las últimas décadas, convirtiéndose en un pilar fundamental para la mejora de la atención clínica, el diagnóstico precoz y la eficiencia operativa en los sistemas de salud (Esteva et al., 2019; Jiang et al., 2017; Topol, 2019). En particular, el uso de técnicas avanzadas de procesamiento y análisis automatizado de imágenes médicas ha revolucionado la forma en que se abordan diversas patologías, proporcionando herramientas para asistir a los profesionales médicos en su práctica diaria (Litjens et al., 2017; Shen et al., 2017).

La radiología digital ha sido una de las áreas que más se ha beneficiado del desarrollo de IA. Modalidades como las radiografías convencionales, las tomografías computarizadas (TC) y las resonancias magnéticas (RM) generan grandes volúmenes de datos visuales que requieren un análisis minucioso para identificar posibles anomalías o condiciones médicas específicas. Tradicionalmente, este análisis ha sido realizado por radiólogos expertos, lo que implica una carga de trabajo significativa.

La integración de sistemas de IA como primer *screening* en la práctica radiológica ha demostrado reducir significativamente la carga de trabajo de los radiólogos sin comprometer el rendimiento diagnóstico. Por ejemplo, un metaanálisis reveló que la colaboración entre humanos y sistemas de IA redujo el tiempo de lectura en un 27,2 % y disminuyó en un 44,5 % los estudios que requerían revisión cuando la IA actuó como segundo lector (Faes et al., 2024).

Además, la IA ha mostrado ser eficaz en la reducción de errores diagnósticos. Un análisis indicó que los sistemas de IA presentan tasas de error más bajas, incluyendo falsos positivos y falsos negativos, en comparación con los radiólogos humanos en ciertos hallazgos específicos (Carvalho et al., 2024).

Estas evidencias respaldan la adopción de la IA como una herramienta

complementaria en la radiología, mejorando la eficiencia operativa y la precisión diagnóstica, y permitiendo a los profesionales centrarse en casos más complejos que requieren su experiencia especializada.

En este contexto, la aplicación de algoritmos de IA ofrece una solución eficaz al automatizar parcialmente la interpretación de estas imágenes, destacando rápidamente áreas sospechosas o anormales. Esto no solo mejora considerablemente la eficiencia diagnóstica, sino que también reduce significativamente los errores humanos asociados al cansancio, la repetición y la variabilidad subjetiva en la interpretación visual. Además, permite que los profesionales médicos puedan centrar su atención en los casos más complejos y urgentes, optimizando así los recursos hospitalarios y mejorando la atención al paciente.

El presente trabajo propone un software basado en IA para el análisis automático y preciso de imágenes médicas. Aunque el sistema ha sido validado principalmente con radiografías por su amplia disponibilidad y simplicidad de uso en entornos clínicos, su arquitectura permite aplicar los mismos principios a otras modalidades como resonancias magnéticas o tomografías computarizadas, siempre que los modelos estén debidamente entrenados. Esta versatilidad es especialmente relevante en entornos clínicos reales, donde los hospitales manejan una gran variedad de estudios de imagen médica digital. El sistema propuesto, por tanto, se adapta a las necesidades diagnósticas cotidianas de centros de salud de distintos niveles de complejidad tecnológica. Esta herramienta utiliza una arquitectura cliente-servidor que permite la separación eficiente de responsabilidades, facilitando tanto el procesamiento de imágenes usando redes neuronales (NN), para la gestión efectiva de los resultados obtenidos como para su presentación visual final.

La arquitectura propuesta integra tecnologías robustas y ampliamente adoptadas en la industria tecnológica actual. Por un lado, se utiliza Django para proporcionar un backend sólido para la gestión de usuarios, autenticación y almacenamiento estructurado de datos (Django Software Foundation, s.f.); por otro lado, para la implementación de un servidor de inferencia ágil y eficiente, se usa FastAPI, que es capaz de ejecutar las NN de segmentación en tiempos reducidos. Finalmente, Docker garantiza que la aplicación pueda desplegarse de manera rápida y segura en múltiples entornos, promoviendo la escalabilidad y la replicabilidad del sistema.

Los objetivos centrales del proyecto se enmarcan en la creación de software que sirva como soporte al diagnóstico clínico. El objetivo general es desarrollar una plataforma accesible, rápida y efectiva para apoyar el análisis diagnóstico mediante IA. Los objetivos específicos se desglosan de la siguiente manera:

- Implementar un sistema automatizado capaz de recibir imágenes médicas y procesarlas de manera eficiente mediante algoritmos de IA, proporcionando los resultados de las segmentaciones que producen dichos

modelos.

- Diseñar y desarrollar una interfaz web interactiva, intuitiva y accesible que permita a los profesionales médicos cargar imágenes, seleccionar algoritmos específicos de IA, gestionar y consultar historiales de análisis previos y descargar fácilmente los resultados para su posterior uso clínico.
- Garantizar la modularidad y escalabilidad del sistema mediante una arquitectura desacoplada y contenerizada, permitiendo la incorporación futura de nuevos modelos de IA o la integración sencilla con otros sistemas de información clínica existentes.
- Validar y asegurar la calidad del producto final mediante pruebas técnicas, incluyendo pruebas automatizadas y manuales que cubran la mayoría de escenarios posibles y garanticen la estabilidad, rendimiento y fiabilidad del sistema en condiciones reales.

Finalmente, la motivación principal del proyecto reside en su potencial impacto social y sanitario. Facilitar herramientas automatizadas de análisis radiológico mediante IA puede traducirse directamente en diagnósticos más rápidos y precisos, atención médica más eficiente y una reducción significativa en costes operativos. Este tipo de tecnología tiene la capacidad no solo de mejorar la calidad asistencial en hospitales y clínicas de mayor nivel económico, sino también de democratizar el acceso a servicios médicos avanzados en áreas rurales o centros con recursos económicos limitados, contribuyendo a la equidad y universalidad de la atención sanitaria, objetivo fundamental de la Organización Mundial de la Salud (OMS) (Organización Mundial de la Salud, 2023).

1.2. Estado del Arte

El interés en herramientas basadas en IA para analizar imágenes médicas ha crecido en los últimos años. En el campo de la radiología digital, muchos de estos programas se usan para detectar patrones anormales (como neumotórax, nódulos o fracturas) y ayudar a los médicos a priorizar o confirmar diagnósticos.

Este capítulo repasa algunas de las soluciones que ya existen y que hacen cosas similares a lo que se pretende lograr con este proyecto. Se explica brevemente qué tipo de imágenes analizan, qué patologías detectan, si se usan realmente en hospitales y qué tecnologías usan por detrás (cuando se conoce). Esto permite comparar enfoques y ver qué aporta realmente esta propuesta respecto a lo que ya existe.

Critical Care Suite (CCS) (GE Healthcare, 2021) es un software embebido en los equipos de rayos X de General Electric (GE). Contiene funciones avanzadas de control de calidad en tiempo real, como la detección de problemas de posicionamiento o campo visual incompleto. Thoracic Care Suite (TCS), desarrollado conjuntamente con Lunit, amplía esta solución incorporando detección de hasta 8 anomalías frecuentes mediante mapas de calor superpuestos directamente sobre las imágenes. Sin embargo, al estar firmemente integrado con los sistemas y hardware propietarios de GE, su escalabilidad es limitada y su adopción fuera de estos ecosistemas resulta restringida.

qXR (Qure.ai) (Biewer et al., 2024) es un software orientado al análisis de radiografías de tórax, especialmente diseñado para su uso en entornos clínicos con recursos limitados. Su principal ventaja es su capacidad para ejecutarse offline en hardware básico, lo que facilita su implementación en zonas rurales o campañas móviles. Puede integrarse con sistemas hospitalarios, aunque su grado de personalización o extensibilidad no está documentado públicamente. Su interfaz genera salidas visuales con anotaciones sobre posibles hallazgos, pero no produce informes estructurados ni se extiende a otras regiones anatómicas más allá del tórax, lo que limita su aplicabilidad en entornos generalistas.

Aidoc Radiology AI Platform (Writers, 2022) es una plataforma para la integración eficaz en sistemas PACS (Picture Archiving and Communication System, tecnología esencial para el manejo y gestión de imágenes médicas digitales). Destaca especialmente por la plataforma AiOS, que permite añadir fácilmente algoritmos adicionales de terceros, ampliando continuamente sus capacidades. Entre sus limitaciones destaca que requiere una infraestructura potente (servidores con GPU o acceso a la nube), lo que eleva los costes y genera dependencia tecnológica.

Lunit INSIGHT CXR (Lunit Inc.) (Hale, 2020; Lunit Inc., 2025) es un software clínico enfocado en la lectura asistida de radiografías torácicas. Se integra con plataformas PACS y equipos de fabricantes como GE o Philips, y permite desplegarse tanto en la nube como en servidores locales. Su funcionamiento se basa en la generación de mapas de calor sobre la imagen original, sin producir informes diagnósticos estructurados. Está optimizado para centros con infraestructura tecnológica consolidada, pero su dependencia de conectividad o hardware específico puede dificultar su adopción en instalaciones más modestas. Además, el alcance funcional está limitado a un conjunto cerrado de hallazgos definidos, lo que restringe su flexibilidad como herramienta generalista.

BoneView (Gleamer) (Pauling et al., 2023) es un software clínico especializado en el análisis automatizado de radiografías, centrado en la detección de fracturas óseas. Está diseñado para integrarse directamente con sistemas PACS, donde genera una copia del estudio original con anotaciones gráficas sobre las zonas sospechosas. La plataforma permite su ejecución tanto en

servidores locales como en la nube, adaptándose a distintos entornos clínicos según la infraestructura disponible. BoneView está optimizado para flujos de trabajo en urgencias, contribuyendo a reducir los tiempos de lectura sin requerir cambios drásticos en los sistemas existentes. Entre sus limitaciones se encuentran: una cobertura restringida a lesiones óseas (sin análisis de tejidos blandos u otras patologías) y la generación de una serie adicional de imágenes que requiere validación manual.

Behold.ai (red dot®) (Behold.ai, 2025) fue un software británico diseñado para el triaje automatizado de radiografías de tórax. Su integración con sistemas RIS (Radiology Information System, sistema informático que gestiona la información relacionada con los estudios de imagen médica, como radiografías, tomografías, resonancias magnéticas, entre otros)/PACS permitía analizar las imágenes inmediatamente tras su adquisición y marcar automáticamente aquellas sospechosas como “anormales”, generando un punto rojo en el listado de trabajo. Su enfoque binario simplificaba el flujo de revisión clínica y pretendía aliviar la carga de radiografías en espera mediante un modelo de pago por estudio. Sin embargo, ofrecía un dictamen simple sin detalle diagnóstico, y su utilidad dependía de la interpretación posterior del radiólogo. Desde 2025, la empresa cesó operaciones y el producto fue discontinuado, lo que limita su viabilidad actual al haber quedado sin soporte, sin actualizaciones y con una validación clínica escasa frente a otras soluciones más robustas.

Arterys Chest/MSK AI (Arterys Inc., 2025) es un módulo dentro de la plataforma en la nube de Arterys, diseñado para análisis automatizado de radiografías en contextos de urgencias. Funciona como solución web accesible desde el navegador, sin necesidad de instalación local, y permite integrar múltiples algoritmos en la misma infraestructura mediante un sistema tipo marketplace. Destaca por su escalabilidad y por la posibilidad de añadir algoritmos complementarios desde la misma interfaz. Sin embargo, su funcionamiento depende de conexión constante a internet, lo que puede limitar su uso en entornos sin infraestructura robusta. Además, al tratarse de una solución generalista, su cobertura es más limitada que la de herramientas especializadas, y su integración automática en flujos clínicos puede requerir desarrollo adicional. Se posiciona como una plataforma versátil y ampliable, pero con menor profundidad diagnóstica por área en comparación con soluciones más focalizadas.

Annalise Enterprise CXR (Annalise.ai) (Magazine, 2021) es un software clínico avanzado para la asistencia en la interpretación de radiografías de tórax. Destaca por su cobertura excepcionalmente amplia, capaz de identificar más de 120 hallazgos distintos, y por una interfaz configurable que permite filtrar los resultados por categoría y nivel de confianza. Su diseño está orientado a integrarse directamente en sistemas PACS y adaptarse a distintos flujos clínicos, incluyendo funciones de priorización de estudios en listas de trabajo. Está disponible como solución local o desplegable

vía plataformas como Sectra Amplifier. Como contrapartida, requiere una infraestructura computacional robusta (GPU o servidor dedicado), y su tiempo de análisis por imagen es mayor que en otras soluciones más acotadas. El volumen de información que presenta también puede resultar excesivo si no se configura adecuadamente.

Solución	Tipo de imagen	Patologías	Tecnología clave	Problema principal
CCS	Radiografía de tórax	Errores de posicionamiento; campo incompleto	Algoritmos embebidos de GE	Limitado al ecosistema GE
TCS	Radiografía de tórax	Hasta 8 anomalías torácicas	Mapas de calor sobre DL	Escasa escalabilidad fuera de GE
qXR	Radiografía de tórax	Anomalías torácicas generales	Modelos CNN offline	Sin informe estructurado; solo tórax
Aidoc	Rayos X, TC, RM	Múltiples, según módulos	GPU/nube (DL)	Requiere infraestructura potente
Lunit Inc.	Radiografía de tórax	Conjunto fijo de hallazgos	Mapas de calor (DL)	No genera informes diagnósticos
BoneView	Radiografía ósea	Fracturas	CNN (local o nube)	Cobertura limitada a huesos
Behold.ai	Radiografía de tórax	Normal/anormal	Modelo binario (DL)	Discontinuado y sin soporte
Arterys	Radiografía de tórax y MSK	Hallazgos torácicos y musculoesqueléticos	Plataforma web (nube)	Necesita conexión continua
Annalise.ai	Radiografía de tórax	> 120 hallazgos	CNN avanzado	Requiere GPU/servidor potente

Cuadro 1.1: Resumen de soluciones IA para imágenes médicas

Del análisis crítico realizado sobre el panorama actual de las plataformas de análisis automatizado de radiografías (1.2), pueden extraerse las siguientes conclusiones:

- Los softwares comerciales actuales han alcanzado un nivel notable de consolidación tecnológica, siendo utilizados en múltiples redes clínicas y contextos hospitalarios, especialmente en tareas de triaje o soporte diagnóstico estructurado.
- A pesar de su adopción creciente, muchas de estas soluciones presentan importantes limitaciones en cuanto a escalabilidad, flexibilidad de integración e interoperabilidad. Suelen depender de infraestructuras específicas (nube, GPU, PACS compatibles), licencias cerradas o ecosistemas propietarios, lo que dificulta su reproducción y mantenimiento en contextos diversos.

- El diseño de interfaz y la experiencia de usuario también representan un reto: muchas plataformas están orientadas a entornos tecnológicos avanzados, pero no han sido diseñadas pensando en su uso cotidiano por personal clínico generalista, lo que reduce su aplicabilidad en centros sanitarios con menor grado de especialización o soporte técnico.

En este contexto, el proyecto propuesto responde a una necesidad no cubierta: desarrollar una herramienta software accesible, adaptable y comprensible, que pueda integrarse fácilmente en distintos entornos clínicos sin requerimientos técnicos elevados ni costes restrictivos, y que aporte una cobertura funcional suficiente para asistir de forma efectiva en el análisis rutinario de radiografías.

1.3. Consideraciones sobre Protección de Datos

El sistema desarrollado en este Trabajo de Fin de Grado está orientado al procesamiento de imágenes médicas, las cuales pueden contener información sensible según lo establecido por el Reglamento General de Protección de Datos (RGPD, 2016).

Aunque durante el desarrollo se han utilizado imágenes de prueba sin información real de pacientes, se ha diseñado el sistema teniendo en cuenta las buenas prácticas para garantizar la confidencialidad, integridad y disponibilidad de los datos. Entre las medidas adoptadas se incluyen:

- El uso de identificadores no personales (como el DNI ficticio) en lugar de nombres o datos identificativos.
- El almacenamiento local de las imágenes procesadas sin transmisión a servidores externos.
- La posibilidad de eliminar cualquier imagen del historial a través de la interfaz de usuario.
- Autenticación de usuario obligatoria para acceder al sistema.

En un entorno real, se recomendaría además la anonimización previa de las imágenes y el uso de cifrado tanto en almacenamiento como en transmisión (por ejemplo, mediante HTTPS y cifrado en disco). Asimismo, se debería informar adecuadamente a los usuarios mediante un consentimiento informado y un aviso legal de privacidad.

Capítulo 2

Especificación del Sistema

2.1. Requisitos

La definición de requisitos constituye una fase fundamental en cualquier proceso de desarrollo de software, ya que establece las bases sobre las que se diseñará, implementará y evaluará la solución final. Una correcta identificación de los requisitos funcionales y no funcionales permite garantizar que el sistema se ajuste a las necesidades reales del usuario, sea técnicamente viable y pueda mantenerse en el tiempo. Además, actúa como guía estructurada durante todo el ciclo de vida del proyecto, facilitando la toma de decisiones, la planificación y la validación de resultados.

2.1.1. Requisitos Funcionales

Los requisitos funcionales describen el comportamiento esperado del sistema, es decir, las funciones específicas que el software debe ser capaz de realizar para satisfacer las necesidades del usuario final (Pressman & Maxim, 2014). Estos requisitos definen las entradas, salidas, procesos y respuestas del sistema ante distintas condiciones de uso. En esta sección se detallan los requisitos funcionales del software propuesto, que orientan el diseño y desarrollo de la aplicación hacia los objetivos establecidos en el proyecto.

RF1 - Carga de imágenes médicas

- **RF1.1** - La imagen cargada debe ser enviada al backend para su procesamiento mediante IA; si falla, mostrará un mensaje de error.
- **RF1.2** - La imagen cargada debe ser enviada al backend para su procesamiento con un DNI que facilitará su búsqueda posteriormente.

RF2 - Procesamiento de imágenes con IA

- **RF2.1** - Se deben poder elegir varios modelos de IA en función de lo que se quiera procesar.
- **RF2.2** - El sistema debe analizar la imagen utilizando un modelo de IA.

RF3 - Visualización de resultados

- **RF3.1** - El sistema debe permitir al usuario visualizar la imagen procesada y la imagen original en la interfaz web.
- **RF3.2** - La imagen procesada debe mostrar claramente las áreas identificadas por la IA.
- **RF3.3** - La IA debe devolver una imagen en un formato visible y visualmente claro.

RF4 - Descarga de la imagen procesada

- **RF4.1** - El sistema debe permitir al usuario descargar la imagen resultante en formato PNG.

RF5 - Comunicación entre frontend y backend

- **RF5.1** El sistema enviará la imagen al backend de forma asíncrona.

RF6 - Almacenamiento de imágenes

- **RF6.1** - El sistema debe permitir guardar el historial de imágenes procesadas para futuras referencias.
- **RF6.2** - El usuario debe poder acceder a imágenes procesadas anteriormente desde una sección de historial, así como descargarlas.
- **RF6.3** - El usuario debe poder eliminar imágenes procesadas anteriormente desde una sección de historial.
- **RF6.4** - El usuario debe poder buscar imágenes procesadas anteriormente a partir del DNI del paciente.
- **RF6.5** - El historial de imágenes debe ser accesible por cualquier usuario autenticado, independientemente de quién haya subido la imagen.

2.1.2. Requisitos No Funcionales

Los requisitos no funcionales especifican las restricciones y cualidades que debe cumplir el sistema, como el rendimiento, la usabilidad, la seguridad o la escalabilidad. Aunque no describen funcionalidades concretas, son esenciales para garantizar que el software sea eficiente, fiable y adecuado a

su contexto de uso (Sommerville, 2011). A continuación, se enumeran los requisitos no funcionales definidos para la solución propuesta en este proyecto.

RNF1 - Rendimiento

- **RNF1.1** - El procesamiento de la imagen por parte del modelo de IA debe completarse en un tiempo razonable (e.g. menos de 5 segundos) en la mayoría de los casos, salvo en la primera ejecución del modelo, que puede tardar más debido a su carga inicial.
- **RNF1.2** - El tiempo total desde la subida hasta la visualización de la imagen procesada no debe superar los 7 segundos, salvo en la primera petición de un modelo que no se encuentre aún cargado en memoria.

RNF2 - Compatibilidad

- **RNF2.1** - La aplicación debe funcionar correctamente en los navegadores principales: Google Chrome, Opera y Microsoft Edge.

RNF3 - Seguridad

- **RNF3.1** - Solo se deben aceptar archivos de imagen en formato JPG, JPEG o PNG, rechazando otros tipos de archivos.
- **RNF3.2** - El sistema debe validar y decodificar las imágenes de forma segura para prevenir vulnerabilidades como la inyección de código malicioso.

RNF4 - Escalabilidad y Mantenibilidad

- **RNF4.1** – El sistema permitirá incorporar nuevos modelos de IA sin modificar componentes de procesamiento existentes.
- **RNF4.2** – El sistema separará frontend, backend y servicio de inferencia en módulos con API REST bien definidos.
- **RNF4.3** - El backend debe poder manejar múltiples solicitudes concurrentes sin degradar significativamente el rendimiento.

RNF5 - Despliegue y Contenerización

- **RNF5.1** - El sistema debe estar contenerizado usando Docker y orquestado mediante docker-compose.
- **RNF5.2** - Los contenedores deben permitir el almacenamiento persistente de imágenes procesadas mediante volúmenes compartidos.
- **RNF5.3** - El sistema debe permitir su despliegue en servidores locales o sin necesidad de configuración manual previa.

2.2. Casos de Uso

Después de la fase de análisis de requisitos, los casos de uso permiten describir cómo interactúan los usuarios con el sistema para lograr objetivos funcionales. Esta técnica de modelado orientado a objetos ayuda a representar de forma estructurada los distintos escenarios operativos del software, facilitando la validación de requisitos desde el punto de vista del usuario y sirviendo como base para el diseño posterior (Jacobson et al., 1992).

En este proyecto se ha optado por elaborar un único diagrama general de casos de uso (2.2), ya que el sistema cuenta con un conjunto reducido y bien definido de actores y funcionalidades principales. Este enfoque resulta adecuado para representar con claridad las operaciones clave de la plataforma, sin necesidad de descomponer el modelado en múltiples diagramas individuales.



Figura 2.1: Diagrama de casos de uso

2.2.1. Desglose de los Casos de Uso

Caso de Uso: Subir Imagen Médica	
ID	CU-01
Actor	Profesional de la Salud
Tipo	Primario
Referencias	-
Precondición	El usuario debe tener acceso a una imagen en formato JPEG, JPG o PNG válida.
Postcondición	La imagen es aceptada y enviada al backend para su procesamiento.
Autor	Alejandro Ruiz Salazar
Versión	1.01
Propósito	Permitir al usuario cargar una imagen médica para su análisis mediante IA.
Resumen	El usuario selecciona y carga una imagen médica desde su dispositivo. El sistema valida el formato y lo envía al backend para su procesamiento.
Flujo de Eventos	
Paso 1	El usuario selecciona la imagen médica desde su dispositivo.
Paso 2	El sistema valida el formato del archivo.
Paso 3	El archivo es enviado al backend para su procesamiento.
Manejo de Excepciones	
CE-01	El usuario intenta subir un archivo con un formato no permitido (ej. PDF, TXT, GIF). <i>El sistema mostrará un mensaje de error y no permitirá la subida del archivo.</i>
CE-02	El usuario sube una imagen con un tamaño mayor a 50MB. <i>El sistema rechazará la imagen y sugerirá reducir su tamaño.</i>

Caso de Uso: Analizar Imagen con IA	
ID	CU-02
Actor	Profesional de la Salud
Tipo	Primario
Referencias	CU-01 (<i>include</i>), CU-05 (<i>extend</i>)
Precondición	La imagen médica ha sido cargada correctamente.
Postcondición	La imagen procesada con anomalías resaltadas es generada y almacenada.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Analizar la imagen utilizando un modelo de IA y resaltar posibles anomalías.
Resumen	El sistema procesa la imagen utilizando IA, detecta anomalías y genera una nueva imagen con las anomalías resaltadas para su evaluación.
Flujo de Eventos	
Paso 1	El sistema procesa la imagen con el modelo de IA.
Paso 2	Se detectan y resaltan las posibles anomalías en la imagen.
Paso 3	Se genera una imagen con las anomalías resaltadas.
Manejo de Excepciones	
CE-03	La IA no logra procesar la imagen correctamente. <i>El sistema mostrará un mensaje de error indicando que el análisis ha fallado y sugerirá intentarlo de nuevo.</i>

Caso de Uso: Mostrar Imagen con Análisis	
ID	CU-03
Actor	Profesional de la Salud
Tipo	Primario
Referencias	CU-02 (<i>include</i>), CU-06(<i>extend</i>), CU-04 (<i>extend</i>)
Precondición	La imagen procesada está disponible.
Postcondición	El usuario visualiza la imagen con las anomalías resaltadas.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Permitir al usuario visualizar la imagen procesada con las anomalías identificadas.
Resumen	El usuario accede a la interfaz web para ver la imagen procesada y revisar las áreas detectadas por la IA.
Flujo de Eventos	
Paso 1	El usuario accede a la sección de visualización.
Paso 2	El sistema muestra la imagen procesada con las anomalías resaltadas.
Paso 3	El usuario revisa la imagen para su evaluación.

Caso de Uso: Descargar Imagen con Análisis	
ID	CU-04
Actor	Profesional de la Salud
Tipo	Primario
Referencias	CU-03
Precondición	La imagen procesada está disponible.
Postcondición	El usuario obtiene la imagen procesada en formato PNG.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Permitir al usuario descargar la imagen con las anomalías resaltadas para su almacenamiento o evaluación.
Resumen	El usuario descarga la imagen procesada en su dispositivo para revisarla más adelante o compartirla con otros especialistas.
Flujo de Eventos	
Paso 1	El usuario selecciona la opción de descarga.
Paso 2	El sistema genera el archivo en formato PNG.
Paso 3	La imagen es descargada al dispositivo del usuario.
Manejo de Excepciones	
CE-04	El usuario intenta descargar una imagen que ya no está disponible. <i>El sistema notificará que el archivo no se encuentra en el servidor.</i>

Caso de Uso: Seleccionar Algoritmo de IA	
ID	CU-05
Actor	Profesional de la Salud
Tipo	Primario
Referencias	CU-01
Precondición	El usuario ha cargado una imagen médica en la plataforma.
Postcondición	Se selecciona un modelo de IA para el análisis de la imagen.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Permitir al usuario elegir el modelo de IA que se usará para analizar la imagen.
Resumen	El usuario elige entre diferentes modelos de IA disponibles en la plataforma según el tipo de análisis requerido.
Flujo de Eventos	
Paso 1	El usuario accede a la sección de selección de modelo de IA.
Paso 2	Se muestran las opciones de modelos de IA disponibles.
Paso 3	El usuario selecciona el modelo de IA deseado.

Caso de Uso: Consultar Historial de Imágenes	
ID	CU-06
Actor	Profesional de la Salud
Tipo	Primario
Referencias	CU-03, CU-04, CU-09
Precondición	
Postcondición	El usuario accede a imágenes previas para su consulta o análisis.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Permitir al usuario visualizar y acceder a imágenes médicas previamente procesadas.
Resumen	El usuario accede a una sección de historial donde puede consultar imágenes procesadas anteriormente para revisarlas o compararlas con nuevas imágenes.
Flujo de Eventos	
Paso 1	El usuario accede a la sección de historial de imágenes procesadas.
Paso 2	Se muestra la lista de imágenes procesadas con detalles relevantes.
Paso 3	El usuario selecciona una imagen para visualizarla o descargarla.

Caso de Uso: Administrar Plataforma	
ID	CU-07
Actor	Personal de TI
Tipo	Secundario
Referencias	-
Precondición	El administrador tiene acceso al sistema de gestión.
Postcondición	Se garantiza el correcto funcionamiento del sistema y sus componentes.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Asegurar que la plataforma y los modelos de IA operen de manera óptima y segura.
Resumen	El administrador supervisa el sistema, asegurando la disponibilidad de los modelos de IA, la seguridad de los datos y el correcto almacenamiento de imágenes procesadas.
Flujo de Eventos	
Paso 1	El administrador revisa el estado del sistema y los modelos de IA.
Paso 2	Se realizan ajustes o mantenimientos necesarios.
Paso 3	Se supervisa la seguridad y almacenamiento de imágenes.

Caso de Uso: Optimizar Modelos de IA	
ID	CU-08
Actor	Personal de TI
Tipo	Secundario
Referencias	-
Precondición	Se requiere probar o mejorar un modelo de IA en el sistema.
Postcondición	Se actualizan los modelos de IA para mejorar la detección de anomalías.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Optimizar los modelos de IA utilizados para el análisis de imágenes médicas.
Resumen	Los técnicos en IA prueban, ajustan y optimizan los modelos de detección de anomalías para mejorar la precisión del análisis de imágenes médicas.
Flujo de Eventos	
Paso 1	El especialista accede a la configuración de modelos de IA.
Paso 2	Se realizan ajustes y pruebas en los modelos de detección de anomalías.
Paso 3	Se implementan mejoras en el sistema para optimizar la precisión del análisis.

Caso de Uso: Eliminar Imagen del Historial	
ID	CU-09
Actor	Profesional de la Salud
Tipo	Primario
Referencias	CU-06
Precondición	El usuario ha accedido al historial de imágenes procesadas.
Postcondición	La imagen seleccionada se elimina de la base de datos y deja de estar disponible en el historial.
Autor	Alejandro Ruiz Salazar
Versión	1.0
Propósito	Permitir al usuario eliminar imágenes procesadas que ya no son necesarias, manteniendo limpio el historial.
Resumen	El usuario selecciona una imagen del historial y confirma su eliminación. El sistema elimina el registro de la base de datos y, en caso de que el archivo aún exista, también lo borra.
Flujo de Eventos	
Paso 1	El usuario accede al historial de imágenes.
Paso 2	El usuario selecciona la opción de eliminar en la imagen deseada.
Paso 3	El sistema solicita confirmación de eliminación.
Paso 4	Tras la confirmación, la imagen es eliminada de la base de datos (y opcionalmente del almacenamiento físico).
Manejo de Excepciones	
CE-05	El usuario intenta eliminar una imagen que ya no existe. <i>El sistema muestra un mensaje de error y permanece en la pantalla del historial.</i>

2.2.2. Funcionalidades Previstas

Actualmente, los casos de uso CU-07 (Administrar Plataforma) y CU-08 (Optimizar Modelos de IA) no están plenamente implementados como funcionalidades interactivas en la plataforma, pero se ha diseñado la arquitectura del sistema con el objetivo de soportarlos en versiones futuras.

- **CU-07 - Administrar Plataforma:** La arquitectura basada en contenedores (Docker + Docker Compose) permite supervisar y mantener el sistema de forma modular. Aunque no existe una interfaz de administración específica (a parte de la propia de Django para su base de datos), el despliegue actual facilita la gestión de servicios, modelos y almacenamiento.
- **CU-08 - Optimizar Modelos de IA:** No se ha incorporado todavía una funcionalidad para reentrenar, ajustar o comparar modelos de IA

directamente desde la aplicación. Sin embargo, el sistema utiliza un patrón Estrategia, que permite reemplazar o mejorar los modelos fácilmente en el backend sin modificar el flujo principal. Esto deja abierta la posibilidad de implementar una interfaz de ajuste y evaluación de precisión en el futuro.

Capítulo 3

Diseño

Este capítulo presenta el diseño del software propuesto, etapa clave del proceso de desarrollo situada entre el análisis de requisitos y la implementación efectiva. Durante esta fase, se definen claramente la estructura interna del sistema, los módulos funcionales y las interacciones entre los distintos componentes. El diseño del software se enfoca en asegurar que el sistema sea mantenible, escalable y eficiente, facilitando así tanto la fase de implementación como futuras ampliaciones o modificaciones.

3.1. Principios y criterios de diseño

El diseño del software propuesto se ha basado en varios principios fundamentales:

- **Modularidad:** el sistema se ha estructurado en módulos claramente definidos y separados según su funcionalidad específica (frontend, backend web, servidor de inferencia). Esta separación permite modificar o ampliar cualquier componente sin afectar significativamente a los demás.
- **Bajo acoplamiento y alta cohesión:** se ha buscado minimizar las dependencias entre módulos, garantizando que cada uno realice tareas específicas y relacionadas entre sí. Este enfoque facilita las pruebas individuales y aumenta la estabilidad general del sistema.
- **Escalabilidad:** los componentes han sido diseñados para permitir fácilmente un aumento en la carga de trabajo. Por ejemplo, el servidor de inferencia está desacoplado mediante FastAPI y Docker, lo que permite escalar este servicio independientemente del resto del sistema.

Respecto a la arquitectura de referencia, se ha adoptado un enfoque basado en una arquitectura cliente-servidor con componentes desacoplados,

que interactúan mediante una API RESTful. Esta estructura permite una clara diferenciación de responsabilidades, mejor distribución de recursos y mayor facilidad de mantenimiento.

En cuanto a las convenciones estructurales seguidas en el desarrollo, se han aplicado:

- Una organización clara por carpetas según la función del código (por ejemplo, carpetas diferenciadas para vistas, modelos, inferencia y utilidades).
- Una nomenclatura consistente y descriptiva para módulos, clases y métodos, facilitando la lectura y el entendimiento del código por parte de otros desarrolladores.
- Flujos de datos bien definidos y documentados, desde la recepción inicial de la imagen por parte del usuario, pasando por su procesamiento mediante IA, hasta la devolución del resultado visualizado en la interfaz.

3.2. Patrones de diseño utilizados

Durante el desarrollo del sistema se ha hecho uso del patrón de diseño Estrategia, el cual permite definir una familia de algoritmos, encapsulando cada uno de ellos y haciendo que sean intercambiables entre sí. Este patrón facilita seleccionar diferentes algoritmos o comportamientos en tiempo de ejecución sin afectar a la estructura del sistema, favoreciendo la extensibilidad y mantenibilidad del software desarrollado (Gamma et al., 1995; Hunt & Hunt, 2013).

En concreto, el patrón Estrategia ha sido utilizado para gestionar la selección dinámica del modelo de IA empleado en la inferencia de imágenes. Esto permite al usuario elegir entre diferentes modelos disponibles sin necesidad de modificar el código interno, simplemente cambiando el contexto de ejecución.

A continuación, se muestra el diagrama de clases UML del patrón Estrategia:

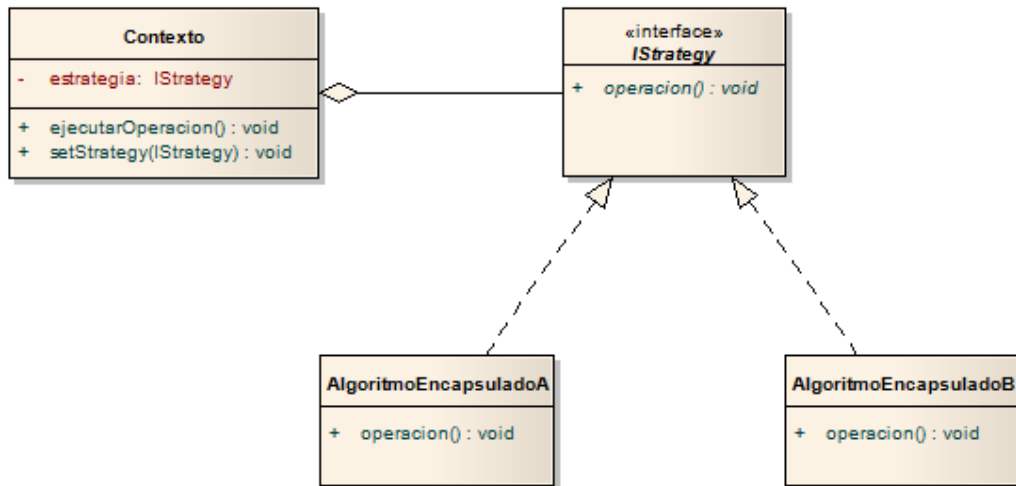


Figura 3.1: Diagrama de clases UML general del patrón Estrategia.

Este enfoque ofrece importantes ventajas al sistema:

- Facilita la incorporación de nuevos modelos de IA sin necesidad de modificaciones significativas en el código base. Basta con crear una clase que implemente («implements») la interfaz *IStrategy* y defina su método *operacion*.
- Mejora la flexibilidad al permitir cambios rápidos y sencillos en el comportamiento del sistema según necesidades específicas.
- Promueve un diseño limpio y modular, simplificando futuras tareas de mantenimiento y evolución del software.
- Al basarse en la herencia e interfaces/clases abstractas, el patrón permite escalar la funcionalidad del sistema mediante clases nuevas reutilizables y fácilmente integrables.

3.3. Diagrama de Clases UML

A continuación se muestra un diagrama de clases que refleja la relación entre los modelos principales del sistema:

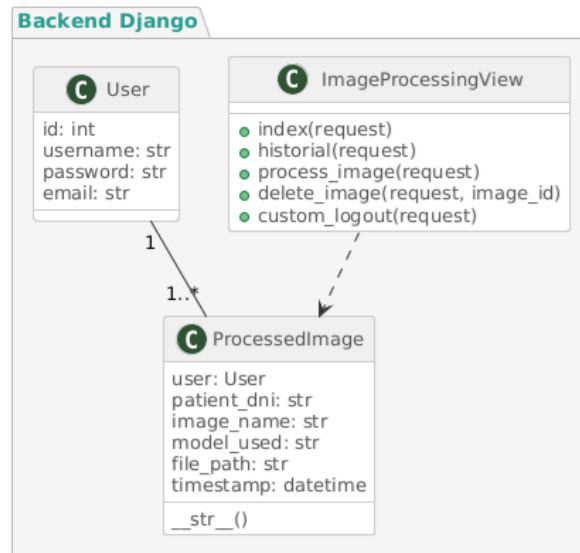


Figura 3.2: Diagrama de clases; paquete de Django.

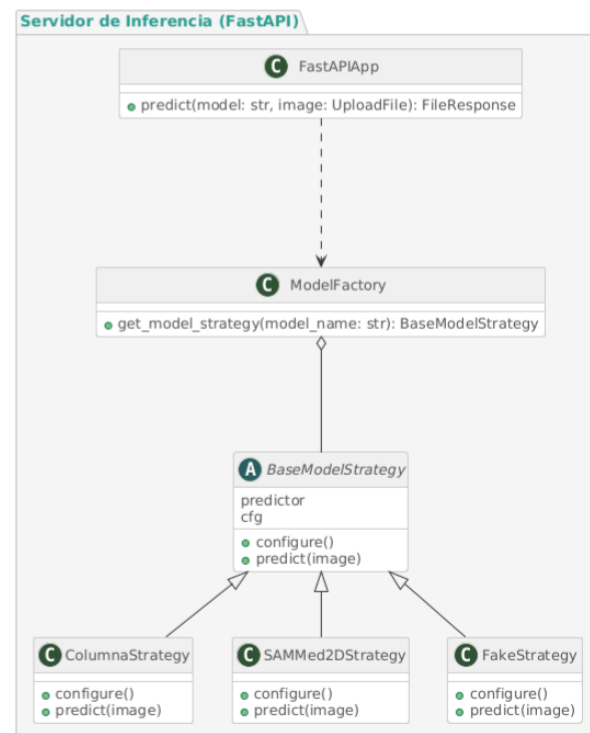


Figura 3.3: Diagrama de clases UML. Se muestran 3 estrategias diferentes: ColumnaStrategy, SAMMed2DStrategy y FakeStrategy; paquete de inferencia

Capítulo 4

Arquitectura del Sistema

Este capítulo describe la arquitectura del sistema propuesto, abordando cómo se estructura internamente y cómo se relacionan sus diferentes componentes. La definición de la arquitectura forma parte esencial de la fase de diseño del software. En ella se concretan las decisiones técnicas que permiten transformar los requisitos funcionales y no funcionales en una solución estructurada, mantenible y escalable. A lo largo del capítulo se detallan las tecnologías empleadas, la organización de los módulos principales, la gestión de los contenedores Docker y el flujo de datos entre frontend, backend y el servidor de inferencia.

4.1. Tecnologías Seleccionadas

Durante el desarrollo del proyecto se han utilizado diversas tecnologías, cada una seleccionada en función de los requisitos del sistema y la eficiencia en su implementación. A continuación, se describen las principales herramientas empleadas:

4.1.1. Django (backend web)

Django es un framework web en Python utilizado para implementar la lógica de negocio del sistema, así como la gestión de usuarios, autenticación y almacenamiento de información sobre las imágenes procesadas (Django Software Foundation, s.f.).

Django fue seleccionado por su facilidad de uso y rápida curva de aprendizaje personal, además de proporcionar una gestión robusta de bases de datos y autenticación.

4.1.2. HTML, CSS y JavaScript (frontend)

El sistema cuenta con una interfaz web desarrollada en HTML, CSS y JavaScript. Se ha optado por una solución que permite al usuario subir

imágenes, seleccionar modelos y visualizar los resultados de manera sencilla. JavaScript permite gestionar de forma dinámica la carga y presentación de imágenes procesadas, así como mostrar mensajes de error y feedback en tiempo real.

Además, se emplea la técnica de AJAX (Asynchronous JavaScript and XML) para enviar y recibir datos del servidor sin necesidad de recargar toda la página. Esto permite que el procesamiento de imágenes, la obtención de resultados y la actualización de la interfaz ocurran de forma fluida y continua, mejorando significativamente la experiencia de usuario (MDN Web Docs, s.f.).

4.1.3. FastAPI (servidor de inferencia)

FastAPI ha sido utilizado como microservicio encargado de ejecutar los modelos de IA. Gracias a su arquitectura asíncrona y su rendimiento, permite responder de forma eficiente a las peticiones del backend, facilitando el desacoplamiento entre componentes y mejorando la escalabilidad (FastAPI, s.f.-a).

FastAPI se escogió por su rendimiento y simplicidad para implementar APIs asíncronas, ofreciendo un tiempo de respuesta rápido y eficiente.

4.1.4. Docker y Docker Compose

Docker ha permitido la contenedorización de los diferentes servicios del sistema (backend y servidor de inferencia), asegurando portabilidad y replicabilidad del entorno. Docker Compose ha sido empleado para orquestar ambos servicios de forma conjunta, simplificando su despliegue y ejecución (Docker Inc., s.f.).

4.1.5. SQLite (base de datos)

SQLite es el sistema de gestión de bases de datos por defecto en Django, utilizado en este proyecto para almacenar de forma ligera y embebida la información relativa a las imágenes, usuarios y resultados procesados (SQLite, s.f.).

4.1.6. Otras tecnologías

Además de las tecnologías principales, se han empleado varias librerías y herramientas de apoyo que complementan el funcionamiento del sistema:

- **Uvicorn:** servidor ASGI (Asynchronous Server Gateway Interface) ligero y de alto rendimiento, utilizado para ejecutar la aplicación FastAPI de inferencia de forma asíncrona (Uvicorn, s.f.).

- **NumPy**: utilizado en el servidor de inferencia para manipular matrices y procesar imágenes.(NumPy Developers, s.f.).

4.2. Arquitectura Propuesta

La arquitectura del sistema sigue una estructura cliente-servidor modular compuesta por tres elementos fundamentales, cada uno con responsabilidades claramente definidas:

- **Frontend (cliente web)**: interfaz web interactiva para que el usuario pueda cargar imágenes médicas, seleccionar modelos y visualizar los resultados obtenidos.
- **Backend (Django)**: intermediario encargado de gestionar la lógica de negocio, incluyendo la autenticación, el almacenamiento persistente de imágenes y resultados, y la comunicación con el servidor de inferencia.
- **Servidor de inferencia (FastAPI)**: microservicio independiente que ejecuta los modelos de IA, recibiendo las imágenes desde Django y devolviendo las imágenes procesadas.

En la siguiente figura se muestra una representación visual simplificada y orientada al flujo de procesamiento de la imagen médica desde el navegador del usuario hasta la obtención del resultado final:

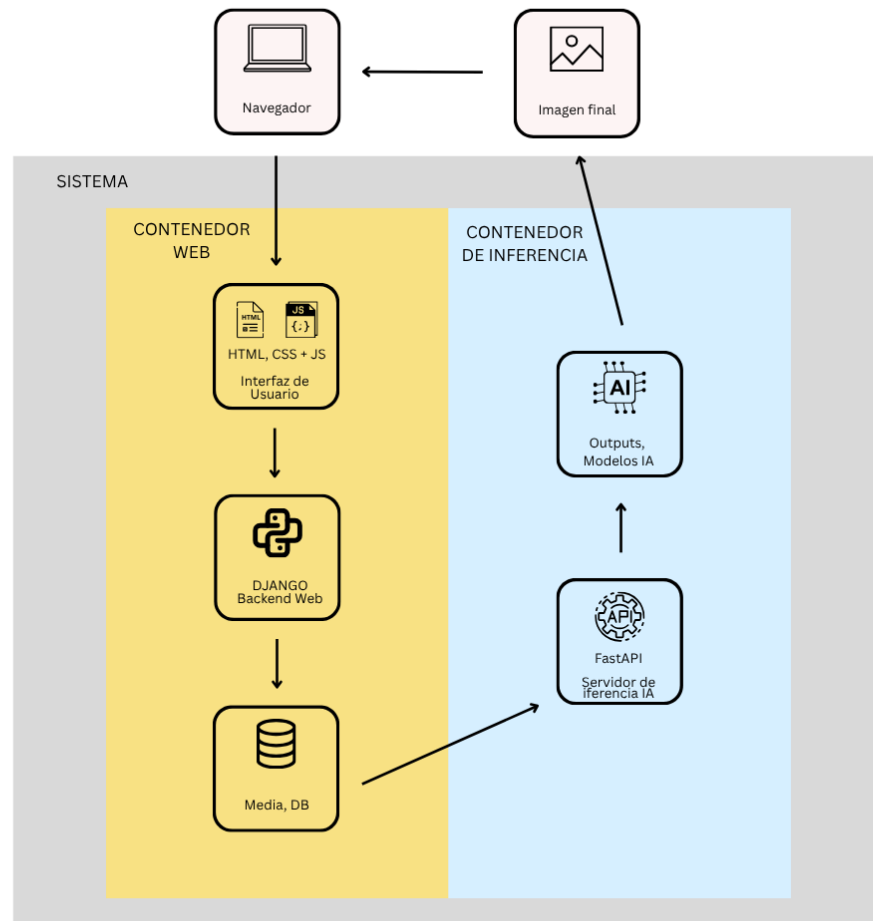


Figura 4.1: Diagrama de arquitectura

El flujo del sistema se describe de la siguiente forma:

1. **Cliente (Navegador):** El usuario accede a la plataforma a través de un navegador web, desde donde puede cargar una imagen médica para su análisis.
2. **Contenedor Web (Django):** Este contenedor aloja la aplicación principal del sistema, desarrollada con Django. Se encarga de gestionar la lógica de negocio, la autenticación de usuarios, el almacenamiento de información en la base de datos, y la persistencia de las imágenes en un volumen dedicado. También sirve los archivos estáticos para el frontend.

3. **Contenedor de Inferencia (FastAPI):** De forma desacoplada, un segundo contenedor ejecuta un servidor de inferencia desarrollado con FastAPI. Este se encarga de procesar la imagen médica usando un modelo de IA seleccionado. Los modelos se almacenan en un volumen dedicado y los resultados se guardan en un volumen de salida.
4. **Resultado final:** Una vez completado el procesamiento, la imagen con la segmentación resaltada es devuelta al navegador del usuario para su visualización o descarga. Todo el ciclo se completa en cuestión de segundos, ofreciendo una experiencia fluida y autónoma.

La contenerización del sistema, junto con la separación lógica de responsabilidades, garantiza facilidad de despliegue, mantenibilidad y posibilidad de escalabilidad futura.

4.3. Clases y Base de Datos

El sistema utiliza una estructura de datos simple y funcional. Se han definido modelos que permiten almacenar de forma persistente la información de las imágenes procesadas, asociadas a cada usuario autenticado. A continuación, se describen las clases principales utilizadas.

4.3.1. Clase ProcessedImage

Este modelo representa una imagen médica que ha sido procesada por IA. Se asocia con un usuario específico y almacena información clave para identificarla, consultarla o incluso borrarla posteriormente.

Aunque cada imagen procesada queda vinculada al usuario que la ha subido, el historial es compartido entre todos los médicos de la plataforma. De esta forma, al buscar por DNI, cualquier profesional puede acceder al historial de análisis de un mismo paciente, garantizando la continuidad asistencial y evitando duplicidad de estudios.

Campo	Tipo	Descripción
id	Integer (PK)	Identificador único de la imagen procesada.
user	ForeignKey (User)	Usuario propietario de la imagen.
patient_dni	CharField(20)	DNI del paciente al que corresponde la imagen.
image_name	CharField(255)	Nombre del archivo de imagen original.
model_used	CharField(50)	Modelo de IA utilizado para procesar la imagen.
file_path	CharField(500)	Ruta del archivo procesado en el sistema.
timestamp	DateTimeField	Fecha y hora del procesamiento.

Cuadro 4.1: Campos del modelo **ProcessedImage**

4.3.2. Clase User

El modelo `User` es proporcionado por Django e incluye campos estándar como nombre de usuario, contraseña, correo electrónico, etc. Se utiliza para gestionar la autenticación y permitir que cada usuario tenga acceso a la app.

4.3.3. Justificación del Modelo

La estructura del modelo `ProcessedImage` está diseñada para ser sencilla, flexible y suficiente para cubrir los requisitos funcionales del sistema:

- La relación con `User` permite un historial personalizado por usuario.
- El campo `patient_dni` facilita búsquedas posteriores por paciente.
- El campo `model_used` permite registrar qué estrategia de IA se aplicó en cada imagen.
- El `file_path` permite recuperar la imagen procesada sin necesidad de almacenar grandes blobs en la base de datos.
- El `timestamp` facilita el orden cronológico y permite filtrar resultados.

4.4. Vistas HTML

Dentro del módulo frontend del sistema, se han diseñado varias vistas HTML que conforman la interfaz de usuario. Estas vistas son servidas por el backend y permiten al usuario interactuar con las funcionalidades principales del sistema. A continuación, se describen las vistas implementadas.

- **login.html:** formulario de autenticación para acceder al sistema. Está protegido contra ataques CSRF (“Cross-Site Request Forgery”, un tipo de ataque que manipula a un usuario autenticado para que realice acciones no deseadas en un sitio web en el que ya está autenticado) y muestra mensajes de error si las credenciales son incorrectas.

Iniciar sesión

Usuario

Contraseña

Entrar

Figura 4.2: Vista de inicio de sesión

- **index.html**: página principal desde la que se puede subir una imagen, introducir el DNI del paciente, seleccionar el modelo de IA, procesar la imagen y posteriormente descargarla. Contiene también un botón para acceder al historial y cerrar sesión.

Analizar Imagen

Bienvenido, admin

[Historial](#)
[Cerrar sesión](#)

Columna ▼

77557787Q

Choose file

prueba.jpg

Procesar Imagen

Descargar imagen procesada

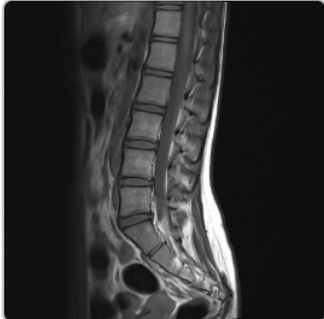
Imagen Original


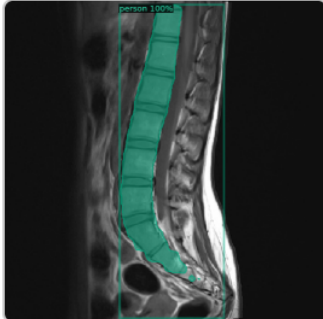
Imagen Procesada


Figura 4.3: Vista principal

- **historial.html**: muestra una tabla con todas las imágenes procesadas, incluyendo el DNI, nombre de la imagen, modelo utilizado, fecha y acciones (visualizar, eliminar). Incluye una barra de búsqueda para filtrar por DNI.

Historial de Imágenes Procesadas

Buscar

DNI	Imagen	Modelo	Fecha	Acciones
77557787Q	prueba.jpg	fake	21/05/2025 19:02	 
77557787Q	prueba.jpg	columna	21/05/2025 19:07	 
77557787Q	prueba.jpg	torax	21/05/2025 19:07	 

[← Volver al inicio](#)

Figura 4.4: Vista del historial

4.5. Interacción entre Frontend y Backend

El flujo de datos entre la interfaz y el servidor sigue este esquema:

1. El usuario selecciona una imagen, introduce el DNI del paciente y elige un modelo.
2. Al pulsar “Procesar Imagen”, se lanza una función JavaScript que crea una petición AJAX al backend mediante `fetch()`, enviando los datos como `FormData`.
3. Django recibe la petición en la vista `process_image`, que a su vez reenvía la imagen al servidor de inferencia FastAPI.
4. El servidor de inferencia procesa la imagen con el modelo indicado y devuelve una nueva imagen con los hallazgos resaltados.
5. Django guarda esta imagen procesada junto con el DNI, modelo utilizado y fecha en la base de datos.
6. El frontend recibe la respuesta y actualiza dinámicamente la interfaz para mostrar la imagen original y la procesada, permitiendo también su descarga.

4.5.1. Funciones relevantes del Backend

Las funciones más importantes definidas en el módulo `views.py` son:

- **process_image**: recibe la imagen y los datos del formulario, los envía al servidor de inferencia y guarda el resultado en el modelo `ProcessedImage`.
- **index**: renderiza la página principal con el formulario de carga de imágenes.
- **historial**: muestra el historial de imágenes procesadas. Permite filtrar por DNI y devuelve todos los resultados del sistema, ya que todos los usuarios de la plataforma son profesionales médicos.
- **delete_image**: permite eliminar una imagen del historial. Solicita confirmación y actualiza la base de datos.

4.6. Seguridad

El sistema incorpora varias medidas de seguridad:

- **Autenticación**: solo los usuarios autenticados pueden acceder a las funciones de análisis e historial.
- **Protección CSRF**: todos los formularios están protegidos mediante tokens CSRF para evitar ataques de falsificación de peticiones.
- **Validación de archivos**: solo se aceptan imágenes en formato JPG, JPEG o PNG, y se valida su contenido antes de procesarlas.
- **Gestión de errores**: el sistema controla errores típicos como imágenes corruptas, falta de campos o fallos del servidor de inferencia, mostrando mensajes comprensibles al usuario.

4.7. Servidor de Inferencia con FastAPI

El procesamiento de las imágenes médicas se realiza en un componente desacoplado del backend principal, implementado con el framework **FastAPI**. Este servidor actúa como microservicio de inferencia, recibiendo peticiones desde el backend Django, aplicando el modelo de IA correspondiente y devolviendo una imagen procesada con los hallazgos resaltados.

4.7.1. Estructura del Servidor

El servidor de inferencia se compone de los siguientes módulos clave:

- **inference_server.py**: archivo principal que define los endpoints y gestiona la lógica de inferencia.

- `factory.py`: módulo que implementa una factoría para instanciar dinámicamente distintas estrategias de modelo según el nombre recibido.
- `columna.py`, `sammed2d.py`, `fake.py`: módulos que implementan diferentes estrategias de predicción (una por modelo disponible).
- `base.py`: clase base abstracta que define la interfaz común para todas las estrategias de modelo.

4.7.2. Carga y Ejecución de Modelos

El sistema implementa el **patrón Estrategia**, de forma que cada modelo hereda de una clase base `BaseModelStrategy` y sobrescribe los métodos `configure()` y `predict()` (3.2).

La función `get_model_strategy()` en `factory.py` recibe un nombre de modelo como string y devuelve una instancia de la clase correspondiente, ya configurada.

4.7.3. Funcionamiento del Endpoint

El servidor expone un único endpoint:

- `POST /predict/`

Este endpoint recibe dos campos: el nombre del modelo y una imagen en formato JPG, JPEG o PNG. El flujo que sigue es:

1. Se valida que el archivo sea una imagen soportada.
2. Se decodifica la imagen con `OpenCV` para obtener una matriz `NumPy`.
3. Se instancia o reutiliza el modelo correspondiente, aplicando caché si ya estaba cargado.
4. Se aplica la predicción y se genera una imagen con las anomalías resaltadas.
5. El resultado se guarda temporalmente y se devuelve como archivo PNG al backend.

4.7.4. Validación y Control de Errores

El servidor implementa múltiples controles:

- Verificación de tipo MIME del archivo de imagen.
- Comprobación de que la imagen es legible tras la decodificación.

- Manejo de excepciones internas con respuestas en formato JSON y mensajes informativos.

En caso de éxito, la respuesta incluye el archivo de imagen procesado. En caso de error, se retorna un mensaje descriptivo con código HTTP adecuado (400 o 500).

4.7.5. Ventajas del Desacoplamiento

Separar el servidor de inferencia del backend principal ofrece varias ventajas:

- Permite escalar o sustituir los modelos sin afectar la lógica web.
- Reduce la complejidad del backend y mejora el mantenimiento.
- Facilita pruebas, despliegue modular y paralelización futura.

4.8. Dockerización y Despliegue

Con el fin de facilitar el despliegue, la replicabilidad y la portabilidad del sistema, se ha optado por contenerizar los distintos componentes utilizando Docker y orquestarlos con Docker Compose. Esta decisión permite ejecutar la aplicación completa en cualquier entorno compatible con Docker sin necesidad de configurar manualmente cada dependencia.

Contenedores Definidos

Para implementar de manera práctica y replicable la arquitectura descrita, se han definido dos servicios Docker mediante `docker-compose.yml`:

- **web**: contiene el backend Django, responsable de servir la interfaz web y gestionar las comunicaciones internas.
- **inference**: contiene el servidor FastAPI, dedicado exclusivamente a ejecutar los modelos de IA.

Ambos servicios están conectados en una red interna creada por Docker Compose, permitiendo una comunicación fluida mediante peticiones HTTP internas. Esto simplifica significativamente el despliegue y el mantenimiento del sistema.

4.8.1. Persistencia de Datos

Para garantizar que los archivos y bases de datos no se pierdan al reiniciar los contenedores, se han definido volúmenes persistentes:

- `./media`: almacena las imágenes procesadas para su posterior consulta.
- `./outputs`: guarda temporalmente los resultados generados por FastAPI.

Capítulo 5

Validación y Pruebas

La fase de pruebas constituye una etapa fundamental dentro del ciclo de vida del desarrollo de software, ya que permite verificar que el sistema cumple con los requisitos definidos y funciona correctamente bajo distintas condiciones. Esta etapa es crítica para detectar errores, asegurar la calidad del producto final y reducir riesgos antes de su puesta en producción. De hecho, se considera la última fase del proceso de desarrollo antes del despliegue del sistema en un entorno real (Sommerville, 2011). El sistema incluye una batería de pruebas automáticas desarrolladas con los frameworks unittest (vía TestCase de Django) y TestClient de FastAPI (Python Software Foundation, s.f.) (FastAPI, s.f.-b). Estas pruebas se complementan con comprobaciones manuales orientadas a validar la interfaz web, la carga de imágenes y la experiencia del usuario.

5.1. Pruebas Automáticas

El fichero `tests.py` contiene una clase unificada llamada `UnifiedTests` que abarca tests tanto del backend Django como del servidor de inferencia FastAPI. Se han implementado los siguientes grupos de tests:

1. Pruebas sobre vistas Django

- **test_django_missing_image_field:** Verifica que no se permite enviar una petición sin imagen.
- **test_process_requires_authentication:** Comprueba que las peticiones no autenticadas son redirigidas.
- **test_process_creates_processedimage_entry:** Asegura que tras un análisis se crea un objeto `ProcessedImage`.
- **test_historical_dni_filter:** Verifica que la búsqueda por DNI devuelve las entradas correctas.

- **test_index_view_renders_for_logged_user:** Comprueba que la vista principal funciona para usuarios autenticados.
- **test_historial_requires_login:** Confirma que la vista de historial exige autenticación.
- **test_delete_nonexistent_image_graceful:** Verifica que se maneja con gracia el intento de eliminar una imagen inexistente.

2. Pruebas sobre el servidor FastAPI

- **test_fastapi_invalid_mimetype:** Rechaza archivos con MIME no válido.
- **test_fastapi_valid_fake_model:** Valida una imagen con el modelo `fake` (útil para tests rápidos).
- **test_fastapi_corrupt_image:** Rechaza imágenes corruptas o no decodificables.
- **test_fastapi_invalid_model_name:** Rechaza nombres de modelo inválidos.
- **test_fastapi_missing_image_field y missing_model_field:** Verifican que ambos campos son obligatorios.
- **test_fastapi_unsupported_mime_type:** Rechaza tipos como GIF que no están permitidos.

3. Pruebas de disponibilidad (*health/up*)

- **test_django_health_up:** Comprueba que la vista de login responde correctamente, confirmando que el backend está levantado.
- **test_fastapi_health_up:** Verifica la disponibilidad del esquema `/openapi.json`, asegurando que el servidor de inferencia está accesible.

4. Pruebas de carga básicas

- **test_django_process_load:** Simula múltiples envíos consecutivos al endpoint `/process/` y comprueba que todas las entradas se almacenan correctamente.
- **test_fastapi_predict_load:** Ejecuta varias predicciones seguidas con FastAPI para comprobar que responde de forma coherente bajo cierta carga repetida.

En conjunto, las pruebas implementadas cubren el 88,24 % de los requisitos funcionales definidos, lo que demuestra un alto grado de verificación automatizada del sistema.

5.1.1. Coverage

Para evaluar el grado en que las pruebas implementadas ejercen el código del sistema, se ha llevado a cabo un análisis de cobertura mediante la herramienta `coverage.py`. Este tipo de análisis permite cuantificar qué porción del código ha sido efectivamente ejecutada durante los tests, proporcionando una medida objetiva del alcance de la validación automática.

En este proyecto, la cobertura fue medida ejecutando los tests definidos en `tests/tests.py` dentro de un contenedor Docker independiente (`test`)(Ned Batchelder, s.f.). Se obtuvo una cobertura total del 92 %.

Módulo	Cobertura
<code>backend/settings.py</code>	100 %
<code>image_processing/views.py</code>	90 %
<code>image_processing/models.py</code>	93 %
<code>inference/inference_server.py</code>	91 %
<code>inference/model_strategies/factory.py</code>	64 %
TOTAL	92 %

Cuadro 5.1: Cobertura de código por módulo

5.1.2. Pruebas Manuales

Además de las pruebas automatizadas, se realizaron comprobaciones manuales orientadas a validar aspectos que no pueden ser evaluados únicamente mediante tests de unidad. Este tipo de pruebas se centró en la experiencia de usuario, el comportamiento de la interfaz web, la usabilidad general del sistema y la estética de la presentación. También permitió detectar posibles errores en la interacción entre componentes (como cargas incorrectas, rutas rotas o respuestas inesperadas) en situaciones reales de uso. Estas validaciones manuales son especialmente relevantes en sistemas con interfaz gráfica, ya que permiten asegurar una experiencia fluida y comprensible para el usuario final.

A continuación, se detallan los principales flujos funcionales que fueron validados manualmente:

- Subida de imagen, selección de modelo y procesamiento.
- Visualización y descarga del resultado procesado.
- Consulta y filtrado del historial por DNI.
- Eliminación de imágenes desde la interfaz.

Se verificó también la compatibilidad con los principales navegadores: Google Chrome, Microsoft Edge y Opera, y la respuesta del sistema frente a entradas incorrectas.

5.1.3. Cobertura de Casos de Error

Además de las funcionalidades principales, el sistema ha sido diseñado para manejar los errores más comunes que pueden producirse durante su uso. La gestión de excepciones y validaciones de entrada es fundamental para garantizar la estabilidad, la seguridad y la experiencia del usuario. A continuación, se enumeran los principales casos de error contemplados explícitamente en el desarrollo:

- Formatos de imagen no soportados.
- Archivos corruptos o ilegibles.
- Peticiones incompletas o campos ausentes.
- Modelo no reconocido.
- Imagen ya eliminada o no existente.

Los errores se gestionan con respuestas claras al usuario (en interfaz o en JSON según el caso) y no comprometen la estabilidad del sistema.

5.1.4. Medición de Rendimiento

Además de validar la corrección funcional del sistema, se ha considerado relevante evaluar su comportamiento en términos de tiempo de respuesta. Esta métrica resulta especialmente significativa en aplicaciones de análisis de imágenes médicas, donde la agilidad en el procesamiento puede influir directamente en la experiencia del usuario y la utilidad clínica del sistema. Para ello, se ha automatizado un script que simula peticiones consecutivas al microservicio de inferencia, midiendo el tiempo total y obteniendo una media aproximada por petición (para 30 peticiones, el tiempo medio de respuesta es de 2.5 segundos).

- **Procesamiento sin recarga de modelo:** entre 1.2 y 2.5 segundos.
- **Carga inicial del modelo + análisis:** entre 5 y 6 segundos.
- **Tiempo total desde subida hasta visualización:** típicamente inferior a 7 segundos.

Capítulo 6

Planificación y Costes

6.1. Planificación y Metodología

Durante el desarrollo de este Trabajo de Fin de Grado, se ha ido llevando un seguimiento detallado de las tareas y actividades realizadas mediante un diagrama de Gantt, el cual permite visualizar de manera clara y ordenada las distintas fases del proyecto, así como los plazos establecidos para cada una de ellas.

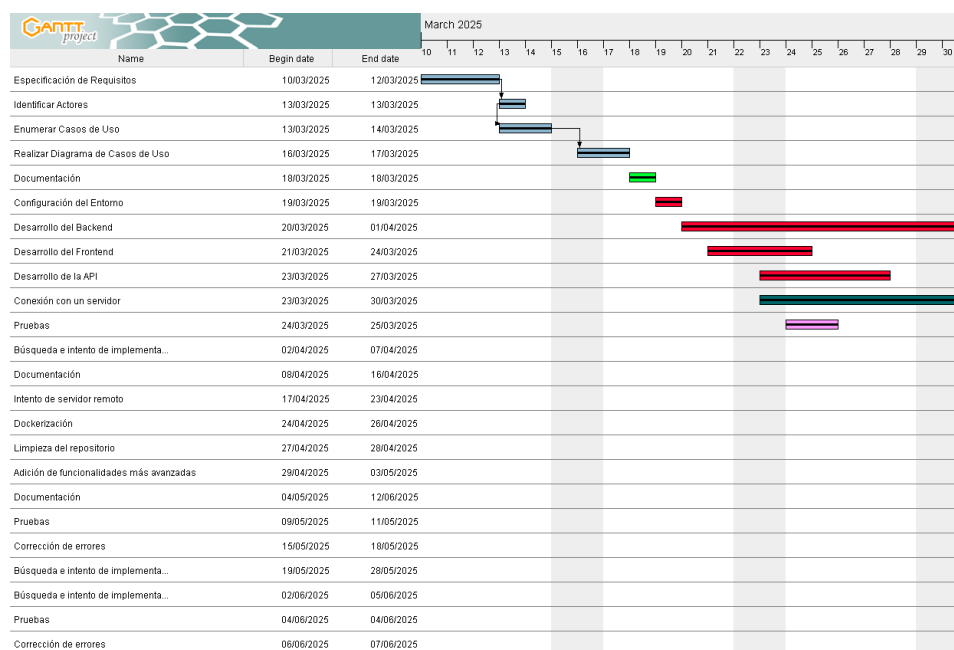


Figura 6.1: Diagrama de Gantt en Marzo

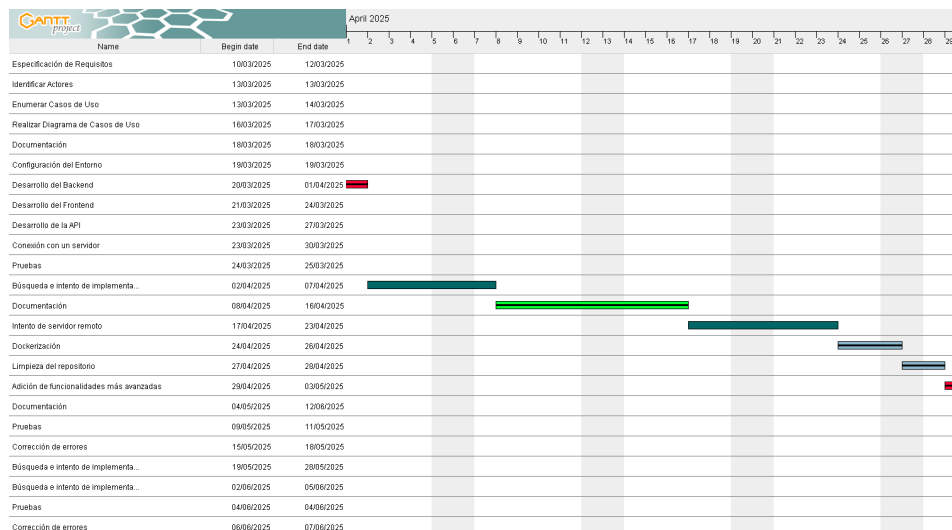


Figura 6.2: Diagrama de Gantt en Abril

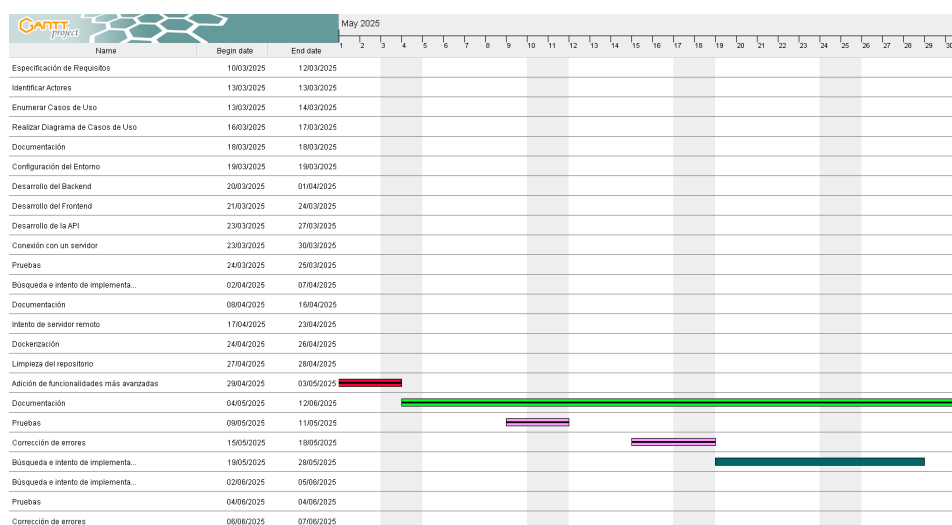


Figura 6.3: Diagrama de Gantt en Mayo

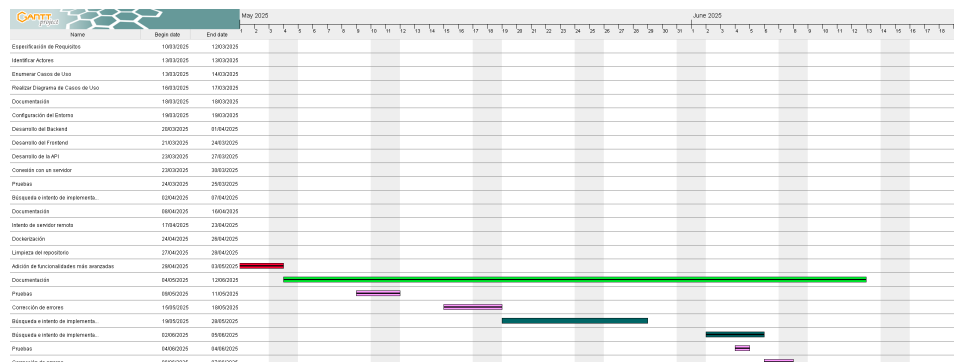


Figura 6.4: Diagrama de Gantt en Junio

En consonancia con esta planificación, se ha adoptado una metodología de desarrollo en cascada (Raquel Brull, 2018), caracterizada por la ejecución ordenada y progresiva de las distintas fases del ciclo de vida del software: especificación de requisitos, diseño, implementación, pruebas y documentación (6.1). Esta metodología resulta especialmente adecuada para proyectos con un alcance cerrado y bien definido desde el inicio, como es el caso de este proyecto. Además, permite mantener una trazabilidad clara entre las fases y facilita el control del avance.

Aunque en la actualidad es común recurrir a metodologías ágiles como Scrum o Kanban, especialmente en entornos colaborativos e iterativos, estas no resultan tan convenientes para el contexto del presente proyecto. Las metodologías ágiles están diseñadas para adaptarse a cambios constantes en los requisitos y fomentar ciclos cortos de desarrollo con entregas continuas, lo cual no se ajusta completamente a la naturaleza individual, académica y con un objetivo cerrado de un TFG. En este caso, la previsibilidad y estructura de la metodología en cascada permiten una mejor organización personal y un seguimiento más claro de los hitos del desarrollo.

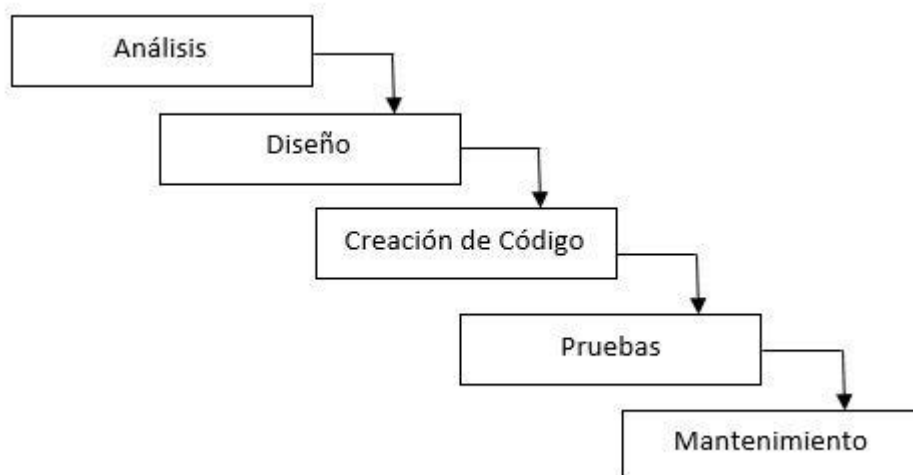


Figura 6.5: Metodología en Cascada

6.2. Recursos Humanos

El desarrollo del proyecto ha implicado la participación de distintos perfiles profesionales, cuya estimación de dedicación y coste resulta fundamental para valorar el esfuerzo necesario si el sistema fuese implementado en un contexto real o profesional. Realizar una evaluación de los recursos humanos permite dimensionar el alcance del proyecto y contextualizar su viabilidad técnica y económica. A continuación, se detalla el coste estimado de las personas implicadas, considerando tarifas aproximadas según el perfil y el mercado laboral actual:

- **Desarrollador junior:** se estima un coste de 25 €/hora, equivalente a un salario bruto anual aproximado de 22000 €, según los datos salariales medios publicados para este perfil en España (Glassdoor, s.f.-a).
- **Tutor académico:** se estima un coste de 40 €/hora, correspondiente a la banda salarial de un perfil senior o profesor universitario con experiencia, cuyo salario bruto anual ronda los 45000 € según diversas fuentes (Glassdoor, s.f.-b).

6.3. Recursos Materiales

A continuación, se detalla el coste estimado de los materiales implicados en el desarrollo del proyecto:

- **Portátil personal:** se utilizó un ordenador portátil adquirido en 2020 por un precio de 1200 €. Dado que han transcurrido cinco años desde su compra, se considera completamente amortizado. No obstante, se imputa un coste residual estimado del 10 % de su valor original (120 €), prorrateado por un uso exclusivo de tres meses (25 % del año), correspondiente al periodo activo del proyecto (de marzo a mayo de 2025). Esto resulta en un coste asignado de **30 €**.

6.4. Presupuesto

Esta sección presenta una estimación detallada de los costes asociados al desarrollo del proyecto, siguiendo un enfoque realista y proporcional al alcance del trabajo realizado. Se han tenido en cuenta tanto los costes de personal como los gastos derivados del uso de hardware y recursos indirectos necesarios durante el periodo activo del proyecto. Aunque se trata de un Trabajo de Fin de Grado sin finalidad comercial, el análisis económico permite valorar el esfuerzo invertido y establecer una aproximación razonable del coste si se tratara de un encargo profesional.

Cuadro 6.1: Estimación de costes del proyecto

	Cálculo	Coste €
<i>Costes de personal</i>		
Estudiante	315 h \times 25 €/h	7875
Tutor	10 h \times 40 €/h	400
Subtotal personal		8275
<i>Costes de hardware</i>		
Portátil (prorrata 3/12 meses)		30
<i>Costes operativos indirectos</i>		
Electricidad + Internet	50 €/mes \times 3	150
Coste total		8455

Capítulo 7

Conclusiones y Líneas Futuras

7.1. Conclusiones Generales

El desarrollo de esta aplicación ha permitido demostrar la viabilidad de integrar técnicas de inteligencia artificial en un entorno web interactivo orientado al análisis de imágenes médicas. A través de una arquitectura desacoplada y modular, se ha logrado construir un sistema funcional que permita a profesionales médicos cargar imágenes, analizarlas con modelos preentrenados y obtener resultados visuales explicables en cuestión de segundos.

Entre los logros más destacables del proyecto se encuentran:

- La implementación de un backend en Django con funcionalidades completas de autenticación, historial, y gestión de imágenes.
- La integración exitosa de un microservicio de inferencia basado en FastAPI, utilizando modelos de segmentación.
- La contenerización completa del sistema con Docker, lo que facilita su despliegue y escalabilidad futura.
- El desarrollo de una interfaz clara, funcional y centrada en la experiencia de uso médico.
- La cobertura mediante pruebas automáticas de los principales flujos de funcionamiento y errores comunes.

Además de los resultados técnicos, el proyecto ha servido como experiencia integradora de múltiples áreas de conocimiento: IA, desarrollo web, buenas prácticas de ingeniería de software y despliegue de sistemas distribuidos.

7.2. Limitaciones Detectadas

A pesar del éxito general del proyecto, existen algunas limitaciones reconocidas:

- Los modelos de IA utilizados son estáticos y no pueden reentrenarse ni personalizarse desde la interfaz actual.
- No se ha implementado aún una gestión avanzada de roles o permisos, lo que limitaría su uso en entornos clínicos con distintos perfiles de usuario.
- La interfaz de historial no incluye paginación ni opciones avanzadas de filtrado o exportación de resultados.
- El sistema no contempla actualmente la subida de estudios en lote ni la comparación entre múltiples análisis de un mismo paciente.

7.3. Líneas Futuras de Mejora

Para evolucionar esta plataforma hacia un producto más completo y cercano a un entorno de producción clínica, se proponen las siguientes líneas de mejora:

- Implementar una interfaz de administración para gestionar modelos, usuarios, y recursos del sistema (actualmente lo único parecido a esto es la propia interfaz de administración de Django).
- Permitir la incorporación dinámica de nuevos modelos desde una API externa o una interfaz de configuración.
- Ampliar el sistema de historial con exportación a PDF, paginación...
- Diseñar una API que permita conectar esta plataforma con otros sistemas clínicos ya existentes, como los que almacenan imágenes médicas (PACS) o gestionan la información de pacientes (HIS). Esto permitiría automatizar el envío y recepción de datos entre la herramienta de análisis y los entornos hospitalarios reales, integrándola completamente en el flujo de trabajo médico.
- Estudiar la viabilidad de desplegar el sistema en servidores cloud o entornos hospitalarios reales bajo requisitos de seguridad y protección de datos.

Estas mejoras permitirían ampliar el alcance funcional y el impacto clínico del sistema, convirtiéndolo en una herramienta real de soporte al diagnóstico médico asistido por IA.

7.4. Futuro del proyecto

Desde sus comienzos, este Trabajo de Fin de Grado se ha concebido como un proyecto abierto a toda la comunidad, basado en código abierto, con la intención de contribuir a la mejora social, aunque sea de forma modesta.

La comercialización del software desarrollado no se contempla, puesto que entra en conflicto directo con mis principios éticos y la filosofía central que fundamenta esta aplicación: el acceso libre y universal a herramientas tecnológicas en salud. La salud, bajo ningún concepto, debería convertirse en una mercancía.

Este proyecto constituye un primer acercamiento a una herramienta que, aunque actualmente se encuentre en una fase inicial y necesite mejoras técnicas y funcionales significativas para aplicarse a gran escala, ha sido diseñado explícitamente con un enfoque modular y extensible. De esta manera, se facilita la futura incorporación de mejoras, ampliaciones o integraciones con otros sistemas de información clínica.

Mi aspiración es que, mediante contribuciones abiertas y colaborativas, este proyecto pueda evolucionar y consolidarse como una alternativa tecnológica relevante, de utilidad pública, y accesible para centros médicos de cualquier nivel, contribuyendo así a reducir la brecha tecnológica en el ámbito sanitario.

Apéndice A

Guía de Uso y Administración del Sistema

Esta sección tiene como objetivo describir el proceso de despliegue, uso y administración del sistema desarrollado, proporcionando una referencia clara tanto para usuarios finales como para posibles desarrolladores o administradores que deseen reutilizar, adaptar o ampliar la solución.

El código fuente completo se encuentra disponible públicamente en el repositorio de GitHub: <https://github.com/aleruizs/TFG>. No obstante, dado que el proyecto requiere el uso de modelos de IA con archivos de gran tamaño, se ha recurrido también a la plataforma Hugging Face para alojar los modelos de análisis, permitiendo así una descarga eficiente desde el servidor de inferencia durante el proceso de inicialización.

A.1. Requisitos previos

Para ejecutar el sistema en un entorno local se requiere:

- Tener instalado Docker y Docker Compose.
- Clonar el repositorio completo, que ya incluye la configuración necesaria para instalar los contenedores con los modelos de IA necesarios.
- Este proyecto requiere una `SECRET_KEY` para funcionar. Se puede generar una ejecutando:

```
from django.core.management.utils
import get_random_secret_key;
print(get_random_secret_key())
```

y guardarla en un archivo llamado `settings_secret.json`.

A.2. Lanzamiento del sistema

Compatibilidad multiplataforma: el sistema ha sido probado tanto en entornos Linux como en Windows mediante Docker Desktop. En ambos casos, puede lanzarse directamente con el comando:

```
docker-compose up --build web inference
```

Esto lanzará dos servicios:

- Backend Django accesible en `http://localhost:8000`
- Servidor de inferencia FastAPI en `http://localhost:8001` (uso interno)

El comando especifica el nombre de los dos servicios a ejecutar para evitar lanzar los tests también.

No se requiere ninguna configuración especial adicional en Windows, siempre que Docker Desktop esté instalado y funcionando con soporte para WSL 2 (que se gestiona automáticamente). Esto garantiza una experiencia de despliegue coherente y portable en cualquier sistema compatible con Docker.

Eliminación completa de contenedores e imágenes (opcional)

En algunos casos, especialmente durante el desarrollo o al detectar errores inesperados (como conflictos en volúmenes o imágenes desactualizadas), puede ser útil realizar una limpieza completa del entorno Docker.

Para ello, se recomienda ejecutar los siguientes comandos desde la raíz del proyecto:

```
docker-compose down --volumes
docker image prune -a
```

El primer comando detiene y elimina todos los contenedores, redes y volúmenes definidos por docker-compose. El segundo comando elimina todas las imágenes locales no utilizadas, liberando espacio y evitando inconsistencias al reconstruir.

A.2.1. Acceso al sistema

Para acceder a la aplicación es necesario iniciar sesión con un usuario previamente registrado. Por defecto, se incluye un usuario administrador con las siguientes credenciales:

- **Usuario:** admin

- **Contraseña:** admin123

Este usuario permite el acceso completo al sistema, incluida la interfaz de administración. Desde dicha interfaz (descrita en la siguiente sección), se pueden crear nuevos usuarios, modificar los existentes o eliminar registros.

Además, el repositorio incluye un conjunto de imágenes de prueba ubicadas en la carpeta `imagenesPrueba/`. Para poder utilizarlas correctamente desde la interfaz web o el historial, es posible que sea necesario ajustar sus permisos de lectura. Para ello, se proporciona un script que debe ejecutarse con permisos adecuados:

```
./fix_permisos.sh
```

Este script otorga los permisos necesarios a los archivos para garantizar su accesibilidad dentro del contenedor.

A.3. Acceso a la interfaz de administración

Django proporciona una interfaz administrativa integrada, accesible desde el siguiente enlace:

`http://localhost:8000/admin/`

El superusuario por defecto (`admin`) también tiene acceso a esta interfaz. Desde aquí se puede gestionar la base de datos de usuarios e imágenes.

En caso de necesitar crear un nuevo superusuario, puede hacerse ejecutando el siguiente comando desde la raíz del proyecto:

```
docker-compose exec web python backend/manage.py createsuperuser
```

Este comando abrirá un asistente interactivo que solicitará el nombre de usuario, dirección de correo (no necesaria) y contraseña. Una vez creado, se podrá iniciar sesión en la interfaz administrativa con las credenciales introducidas.

A.4. Modificación de datos desde la interfaz

Desde la interfaz de administración, es posible:

- Crear, editar o eliminar usuarios.
- Consultar el historial de imágenes procesadas.
- Eliminar registros o corregir errores manualmente.

Bibliografía

- Arterys Inc. (2025). *Arterys - Medical Imaging Cloud AI Platform*. <https://www.arterys.com/>
- Behold.ai. (2025, febrero). *red dot Chest X-Ray (discontinued)*. <https://healthairegister.com/products/holdai-red-dot/>
- Biewer, A. M., Tzelios, C., Tintaya, K., Roman, B., & Hurwitz, S. (2024). Accuracy of digital chest x-ray analysis with artificial intelligence software as a triage and screening tool in hospitalized patients being evaluated for tuberculosis in Lima, Peru. *PLOS Global Public Health*, 4(1), e0002031. <https://doi.org/10.1371/journal.pgph.0002031>
- Carvalho, E., Silva, M., Lopes, A., & Teixeira, R. (2024). AI versus human performance in radiographic diagnosis: A comparative analysis in veterinary radiology. *Frontiers in Veterinary Science*, 11, 1437284. <https://doi.org/10.3389/fvets.2024.1437284>
- Django Software Foundation. (s.f.). Django documentation (version 5.2). <https://docs.djangoproject.com/en/5.2/>
- Docker Inc. (s.f.). Docker Documentation. <https://docs.docker.com>
- Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., & Dean, J. (2019). A guide to deep learning in healthcare. *Nature Medicine*, 25(1), 24-29. <https://doi.org/10.1038/s41591-018-0316-z>
- Europea, U. (2016). Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016. <https://www.boe.es/doue/2016/119/L00001-00088.pdf>
- Faes, L., Liu, X., Wagner, S. K., Denniston, A. K., & Keane, P. A. (2024). Human-AI collaboration for screening: A systematic review and meta-analysis. *NPJ Digital Medicine*, 7, 63. <https://doi.org/10.1038/s41746-024-01328-w>
- FastAPI. (s.f.-a). FastAPI documentation. <https://fastapi.tiangolo.com>
- FastAPI. (s.f.-b). Testing - FastAPI. <https://fastapi.tiangolo.com/advanced/testing/>
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- GE Healthcare. (2021, noviembre). *X-ray Critical Care Suite 2.0 [Brochure USA]*. <https://www.gehealthcare.com/-/jssmedia/gehc/us/files/>

- products / radiography / mobile- xray - systems / critical- care - suite - on - optima - xr240amx / xray - critical - care - suite - 2_0 - usa - brochure - jb06158us.pdf?rev=-1
- Glassdoor. (s.f.-a). Junior Software Developer Salaries in Spain. https://www.glassdoor.es/Salaries/spain-junior-software-developer-salary-SRCH_IL.0,5_IN219_KO6,31.htm
- Glassdoor. (s.f.-b). Software Engineer Sueldos en España. https://www.glassdoor.es/Sueldos/espa%C3%B1a-software-engineer-sueldo-SRCH_IL.0,6_IN219_KO7,24.htm
- Hale, C. (2020, junio). GE Healthcare rolls out new AI-powered chest X-ray suite. <https://www.fiercebiotech.com/medtech/ge-healthcare-taps-lunit-s-ai-for-its-new-chest-x-ray-suite>
- Hunt, J., & Hunt, J. (2013). Gang of four design patterns. *Scala Design Patterns: Patterns for Practical Reuse and Design*, 135-136.
- Jacobson, I., Christerson, M., Jonsson, P., & Övergaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.
- Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S., Wang, Y., Dong, Q., Shen, H., & Wang, Y. (2017). Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology*, 2(4), 230-243. <https://doi.org/10.1136/svn-2017-000101>
- Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., van der Laak, J. A. W. M., van Ginneken, B., & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42, 60-88. <https://doi.org/10.1016/j.media.2017.07.005>
- Lunit Inc. (2025). *Lunit INSIGHT CXR*. <https://www.lunit.io/en/products/cxr>
- Magazine, R. (2021, agosto). *Comprehensive chest X-ray AI launch is backed by positive study results*. <https://www.radmagazine.com/comprehensive-chest-x-ray-ai-launch-is-backed-by-positive-study-results/>
- MDN Web Docs. (s.f.). AJAX - Web technology for developers. <https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX>
- Ned Batchelder. (s.f.). coverage.py Documentation. <https://coverage.readthedocs.io>
- NumPy Developers. (s.f.). NumPy Documentation. <https://numpy.org/doc>
- Organización Mundial de la Salud. (2023). Harness digital health for universal health coverage. <https://www.who.int/southeastasia/news/detail/20-03-2023-harness-digital-health-for-universal-health-coverage>
- Pauling, C., Kanber, B., Arthurs, O. J., & Shelmerdine, S. C. (2023). Commercially available artificial intelligence tools for fracture detection:

- the evidence. *BJR Open*, 6(1), tzad005. <https://doi.org/10.1093/bjro/tzad005>
- Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8.^a ed.). McGraw-Hill Education.
- Python Software Foundation. (s.f.). unittest — Unit testing framework. <https://docs.python.org/3/library/unittest.html>
- Raquel Brull. (2018). Metodología en cascada. <https://medium.com/@raquelbrull/metodolog%C3%ADa-cascada-f114683031e9>
- Shen, D., Wu, G., & Suk, H.-I. (2017). Deep learning in medical image analysis. *Annual Review of Biomedical Engineering*, 19, 221-248. <https://doi.org/10.1146/annurev-bioeng-071516-044442>
- Sommerville, I. (2011). *Software Engineering* (9.^a ed.). Addison-Wesley.
- SQLite. (s.f.). SQLite Documentation. <https://www.sqlite.org/docs.html>
- Topol, E. J. (2019). High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, 25(1), 44-56. <https://doi.org/10.1038/s41591-018-0300-7>
- Uvicorn. (s.f.). Uvicorn Documentation. <https://www.uvicorn.org>
- Writers, A. S. (2022, marzo). *Aidoc gets FDA nod for pneumothorax AI triage software*. <https://www.auntminnie.com/clinical-news/digital-x-ray/article/15630645/aidoc-gets-fda-nod-for-pneumothorax-ai-triage-software>