



viu

**Universidad
Internacional
de Valencia**

Sistema de extracción de información de documentos

De:

 Planeta Formación y Universidades

Trabajo final de grado

Título	Sistema de extracción de información de documentos
Universidad	Universidad Internacional de Valencia
Titulación	Grado de ingeniería informática
Mención	Ingeniería del software
Curso	2023/24
Convocatoria	Primera convocatoria, Junio
Autor	Daniel González Cerviño, 49012312 W
Directora	Marlene Goncalves da Silva, Y7592198 G

Dedicatoria

A Arancha, mi compañera, cuyo apoyo ha sido fundamental en este proceso. Gracias por asumir con amor la carga extra de nuestras responsabilidades familiares, permitiéndome así quitarme la espinita de no haber completado mis estudios de ingeniería.

Tu fortaleza y dedicación son el cimiento de nuestra familia y de cada uno de mis logros.

A mis queridas hijas, Aroa y Enara, gracias por llenar cada día con vuestra alegría y entusiasmo por la vida.

Este trabajo es tan vuestro como mío.

Dedicado con todo mi amor y gratitud a las tres, por ser mi mayor inspiración y la razón de mi esfuerzo.

Resumen

El presente trabajo se centra en el desarrollo de un sistema de extracción de información de documentos de cualquier tipo y formato, utilizando tecnologías avanzadas y enfoques modernos de arquitectura de software.

Dado que este objetivo general puede resultar demasiado ambicioso para un TFG, el objetivo específico será crear una herramienta con capacidad para procesar documentos *PDF*, enfocándose en dos tipos de documentos concretos: los contratos de arrendamiento de vivienda entre particulares y los contratos de compraventa de vehículos entre particulares.

Aplicando los principios de código limpio y arquitectura limpia, se podrá asegurar la calidad del código y la capacidad para añadir soporte a nuevos formatos de documento y a nuevos tipos de contratos.

Para el desarrollo del sistema, se utilizó un enfoque modular, implementando dos componentes principales: el *Generator*, que convierte documentos *PDF* en texto, y el *Reader*, que extrae la información requerida del texto generado.

El análisis de los resultados demostró una precisión del 100 % en la identificación y extracción de datos para los conjuntos de prueba utilizados. Sin embargo, dado que los modelos LLM como *ChatGPT* no son sistemas deterministas y la muestra es relativamente pequeña, es posible que los resultados reales sean ligeramente inferiores.

En términos de rendimiento, se constató que el núcleo del sistema es rápido, aunque el rendimiento general depende de las herramientas seleccionadas para la capa de infraestructura. Las pruebas de rendimiento mostraron tiempos de ejecución bajos para herramientas locales como *PDF To Text*, pero tiempos más elevados para la *API* de *ChatGPT*.

Palabras clave: extracción de información, código limpio, arquitectura limpia, PHP, Symfony, PDF, LLM, ChatGPT.

Abstract

The present work focuses on the development of an information extraction system for documents of any type and format, using advanced technologies and modern software architecture approaches.

Since this general objective may be too ambitious for a final year project, the specific objective will be to create a tool capable of processing *PDF* documents, focusing on two specific types of documents: residential lease agreements between individuals and vehicle sale contracts between individuals.

By applying the principles of *clean code* and *clean architecture*, the quality of the code and the ability to add support for new document formats and new types of contracts can be ensured.

For the development of the system, a modular approach was used, implementing two main components: the *Generator*, which converts PDF documents into text, and the *Reader*, which extracts the required information from the generated text.

The analysis of the results demonstrated 100 % accuracy in the identification and extraction of data for the test sets used. However, since LLM models like *ChatGPT* are not deterministic systems and the sample is relatively small, the actual results may be slightly lower.

In terms of performance, it was found that the core of the system is fast, although the overall performance depends on the tools selected for the infrastructure layer. Performance tests showed low execution times for local tools like *PDF To Text*, but higher times for the *ChatGPT API*.

Keywords: Information extraction, clean code, clean architecture, PHP, Symfony, PDF, LLM, ChatGPT.

Índice general

Índice general	5
Índice de figuras	7
Índice de tablas	8
1. Introducción	9
1.1. Antecedentes	9
1.2. Planteamiento del problema	9
1.3. Justificación	10
1.4. Objetivos	11
2. Marco teórico	14
2.1. Código limpio	14
2.2. Arquitectura limpia	14
2.3. Procesamiento de lenguaje natural	16
2.4. Modelos de lenguaje de gran escala	16
3. Metodología	19
3.1. Metodología ágil	19
3.2. Gestión visual de procesos	20
4. Desarrollo	23
4.1. Planificación	23
4.2. Tecnologías utilizadas	25
4.3. Diseño del sistema	31
4.4. Implementación y programación	36
4.5. Pruebas y evaluación	41
5. Resultados	46
5.1. Descripción detallada de la solución informática desarrollada	46
5.2. Rendimiento y eficiencia	47
5.3. Calidad del código y mantenibilidad	49

5.4. Análisis de los resultados	53
6. Conclusiones	54
6.1. Recapitulación de los objetivos	54
6.2. Síntesis de los resultados	54
6.3. Implicaciones y relevancia	56
6.4. Reflexión sobre el proceso de desarrollo	57
6.5. Trabajo futuro	58
Bibliografía	62
A. Instrucciones de instalación	66

Índice de figuras

1.1. Esquema del funcionamiento general del sistema	11
1.2. Esquema de los objetivos específicos del TFG	12
1.3. Esquema de los conjuntos de datos prueba e interfaces interactuando con el sistema	13
2.1. Esquema de una arquitectura limpia	15
3.1. Captura del tablero kanban del proyecto	21
4.1. Esquema de los componentes principales del sistema	32
4.2. Diagrama UML del componente <i>Generator</i>	32
4.3. Esquema de la competición entre procesadores del componente <i>Generator</i>	33
4.4. Diagrama UML del componente <i>Reader</i>	35
4.5. Captura del código fuente del procesador <i>PDF To Text Processor</i>	36
4.6. Esquema de comunicación con la herramienta de la capa de infraestructura pdftotext	37
4.7. Captura del código fuente del procesador <i>Residential Lease Agreement Processor</i>	38
4.8. Esquema del sistema de procesamiento de logs	41
4.9. Esquema del sistema de construcción automática de documentos en el conjunto de datos de prueba	44
4.10. Captura del resultado de la ejecución de la batería de pruebas automáticas en el entorno de desarrollo	45
4.11. Captura del resultado de la ejecución de la batería de pruebas automáticas en el entorno de integración continua	45
5.1. Captura del resultado de procesar un documento a través de la interfaz web	47
5.2. Captura del resultado de procesar un documento a través de la interfaz de línea de comandos	48
5.3. Esquema de las herramientas en la capa de infraestructura	49

Índice de tablas

4.1. Resumen de los resultados de la ejecución procesador <i>Residential Lease Agreement Processor</i>	39
4.2. Límite de tokens por contexto para los modelos más recientes de <i>ChatGPT</i>	40
4.3. Resumen de los conjuntos de datos de prueba	43
5.1. Evaluación del rendimiento de las herramientas de la capa de infraestructura	49
5.2. Evaluación de la cobertura de código	50
5.3. Evaluación de la complejidad del código	51
5.4. Evaluación de los resultados por tipo de conjunto	53
A.1. Dependencias del proyecto y versiones mínimas	66

1 Introducción

Este Trabajo de Fin de Grado (TFG) aborda la necesidad de automatizar la extracción de información de documentos.

La solución propuesta, tiene como objetivo general procesar documentos de cualquier formato y tipo, siendo los objetivos específicos, dar soporte a documentos en formato *PDF* y documentos de tipo contrato de arrendamiento de vivienda y de compraventa de vehículos.

1.1 Antecedentes

El desarrollo de este TFG surge de una necesidad surgida en mi faceta profesional, donde la tarea de leer y extraer información de documentos representa una carga significativa para la empresa en la que trabajo.

Este problema no es exclusivo de mi actual empresa, sino que es una realidad común en una variedad de sectores incluyendo entre otros a compañías de seguros, instituciones educativas, empresas del sector sanitario, empresas de gestión de recursos humanos o entidades financieras.

Estas empresas y organizaciones se enfrentan el reto de gestionar grandes volúmenes de documentación, lo cual resalta la importancia y la necesidad de soluciones automatizadas que permitan extraer información de dichos documentos.

Además, durante la asignatura del Grado de Ingeniería Informática 47 Proyecto de Ingeniería del Software, tuve la oportunidad de desarrollar una base preliminar que he utilizado como la base para la realización de este TFG [1].

1.2 Planteamiento del problema

Supongamos una empresa que gestiona seguros de coche, para dar de alta un nuevo cliente deberá recibir y procesar un paquete de datos que contendrá entre

otros los siguientes documentos:

- Documentos de identidad del titular y los tomadores
- Permiso de conducción de los tomadores
- Ficha técnica y permiso de circulación del vehículo
- Recibo del impuesto de vehículo de tracción mecánica

La forma tradicional de tramitar la información que contienen estos documentos consiste en que un operario reciba los documentos, los abra e introduzca los datos en el sistema. Esta metodología enfrenta la siguiente problemática:

- El personal dedicado a estas tareas genera un **elevado coste** que impacta directamente en los costes operativos de la organización.
- La tramitación manual implica la **demora en los tiempos de tramitación** de documentos que no van a ser procesados en el momento en que son recibidos, sino que deberán esperar a que un operario esté disponible para ocuparse de esta tarea.
- **Pobre asignación de recursos humanos** que podrían ser asignados a actividades que aporten un mayor valor a la organización. Esto incluye tareas como la innovación, el desarrollo estratégico y el servicio de atención al cliente, entre otros.
- **Errores manuales** en la tramitación de documentos.

Ante esta situación, se hace necesario el desarrollo de soluciones con capacidades de automatizar el proceso de extracción de la información de documentos.

1.3 Justificación

Como hemos visto en la sección 1.2 Planteamiento del problema, la relevancia de este TFG se fundamenta en la necesidad de implementar soluciones que permitan automatizar el proceso de extracción de la información contenida en documentos de cualquier tipo y cualquier formato.

- Desde el punto de vista profesional este proyecto responde a una necesidad real de desarrollar una solución que permita la recuperación de la información contenida en documentos de diversa índole.
- Desde la perspectiva académica, el desarrollo de un sistema de extracción automática de información aborda competencias clave en la ingeniería

informática, el diseño de sistemas, la programación, y las buenas prácticas de desarrollo de software.

En resumen, este TFG no solo es una oportunidad para aplicar habilidades y conocimientos técnicos adquiridos durante el grado de ingeniería informática, sino también una contribución innovadora que aplica directamente en mi ámbito profesional.

1.4 Objetivos

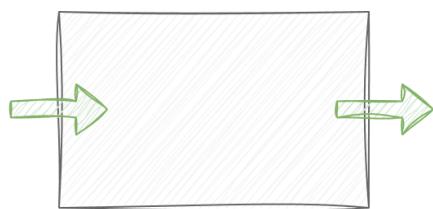
Como hemos visto en las secciones anteriores el propósito general de este TFG es la creación de un sistema que permita la extracción automática de documentos de cualquier tipo y cualquier formato.

Soporte:

1) Documentos en cualquier formato



2) cualquier tipo de documento



DataMiner

Estructura de datos con la información relevante

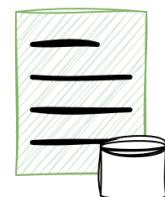


Figura 1.1: Esquema del funcionamiento general del sistema

Provisionalmente, hemos denominado al sistema *DataMiner*. Por lo tanto, de aquí en adelante nos referiremos a él indistintamente como: “sistema de extracción de información de documentos”, “el sistema” o “*DataMiner*”.

Tal y como se puede ver en la figura 1.1 el funcionamiento es el siguiente:

1. El sistema recibe un documento de cualquier tipo en cualquier formato.
2. El sistema genera una estructura de datos que contiene la información relevante del documento recibido.

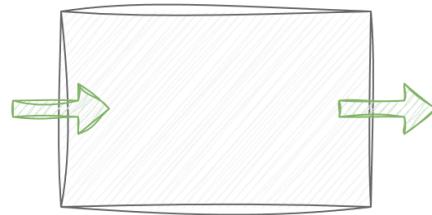
Objetivos específicos

Como este objetivo puede resultar demasiado ambicioso, el alcance de este TFG quedará limitado a los siguientes objetivos específicos que aparecen en la figura 1.2.

1. **Requisito 1** desarrollar un sistema capaz de recibir documentos PDF [2] .
2. Implementar dos casos de uso dentro del sistema:
 - a) **Requisito 2** procesar contratos de arrendamiento de vivienda entre particulares.
 - b) **Requisito 3** procesar contratos de compraventa de vehículo entre particulares.

Soporte:

1) Documentos en formato PDF



Estructura de datos con la información relevante



2a) Contratos de alquiler de vivienda

DataMiner

2b) Contratos de compraventa de vehículos

Figura 1.2: Esquema de los objetivos específicos del TFG

Además, tal como aparece en la figura 1.3 vamos a necesitar dos objetivos complementarios para completar este trabajo.

1. Desarrollar un conjunto de pruebas, que permita realizar las pruebas oportunas durante el desarrollo del proyecto, y que en su conclusión nos permita evaluar cuán preciso es el sistema.
2. Desarrollar interfaces que permitan a un usuario comunicarse con el sistema:
 - a) Una interfaz web
 - b) Una interfaz de línea de comandos

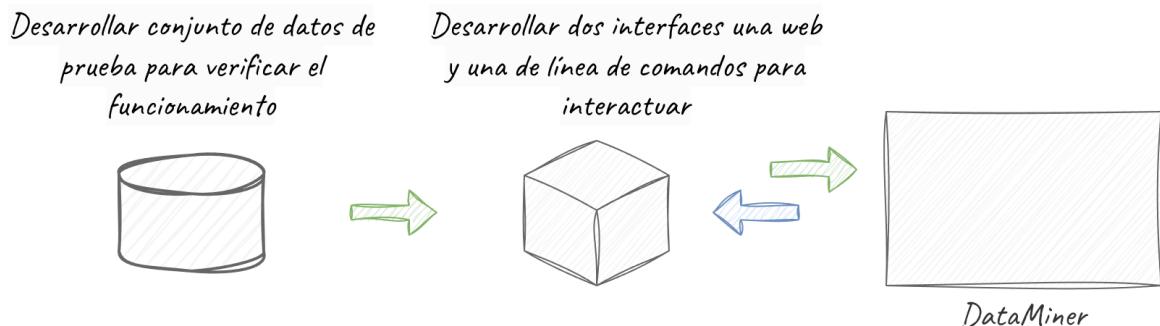


Figura 1.3: Esquema de los conjuntos de datos prueba e interfaces interactuando con el sistema

El sistema estará diseñado de tal forma que se puedan introducir nuevas características de una forma sencilla. Por ejemplo estas son algunas características que pueden ser añadidas fácilmente.

1. Procesar nuevos formatos de documentos, como por ejemplo formatos de office como microsoft word [3] o microsoft excel [4] o formatos de video o audio, etc.
2. Procesar nuevos tipos de documentos, como documentos de identidad, acuerdos de confidencialidad, documentos estatales de impuestos, etc.

2 Marco teórico

2.1 Código limpio

El concepto de Código Limpio ha sido fundamental en la programación de software desde los primeros días del desarrollo de software, pero fue articulado y popularizado por Robert C. Martin en el libro *Clean Code: A Handbook of Agile Software Craftsmanship* [5]

Este libro se convirtió en una guía esencial para muchos desarrolladores al enfatizar la importancia de escribir código que no solo funcione, sino que también sea fácil de entender, modificar y mantener y hace exactamente lo que se espera que haga.

Según Robert C. Martin, el código limpio puede ser leído y mejorado por un desarrollador que no sea su autor original con un mínimo esfuerzo necesario.

Se caracteriza por su simplicidad, la ausencia de duplicación, la expresión clara de la intención del desarrollador y la atención a los detalles en el nivel de código.

En el desarrollo de este proyecto, permitirá que otros desarrolladores puedan entender y modificar el código con facilidad, lo cual es esencial para futuras extensiones y mejoras del sistema.

2.2 Arquitectura limpia

En el ámbito del desarrollo de software, la arquitectura de un sistema es crucial para determinar su escalabilidad y mantenibilidad a largo plazo. Una de las metodologías que ha cobrado relevancia en este contexto son las “arquitecturas limpias”, un enfoque para el diseño de software promovido por Robert C. Martin en el libro *Clean Architecture: A Craftsman’s Guide to Software Structure and Design* [6].

La arquitectura limpia es un estilo de diseño que organiza el sistema de manera

que sea independiente de los elementos externos a los que considera “detalles de implementación” y que son intercambiables unos por otros. Ejemplos de detalles de implementación son los *frameworks*, las *interfaces*, las bases de datos o las *API* entre otros.

La esencia de esta arquitectura radica en su diagrama de capas, donde cada capa tiene una responsabilidad claramente definida y depende solo de las capas más internas. Esto se logra mediante el principio de inversión de dependencias, lo que significa que los detalles dependen de las abstracciones y no al contrario.

Existen diferentes implementaciones de “arquitectura limpia”, cada una con sus características intrínsecas, en este proyecto hemos decidido implementar la versión en tres capas que aparece en la figura 2.1.

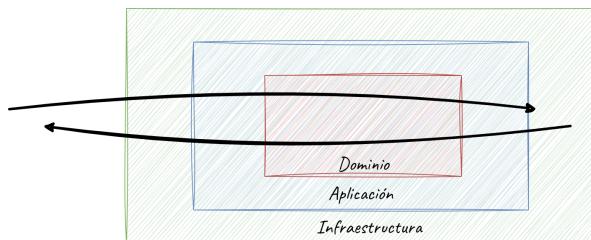


Figura 2.1: Esquema de una arquitectura limpia

La capa más interior se denomina **capa de dominio** y es el corazón del proyecto, encapsula la lógica y las reglas de negocio. Esta capa es fundamentalmente agnóstica respecto a tecnologías externas y se centra exclusivamente en definir los conceptos fundamentales. Algunos de los elementos que contiene son: entidades, objetos de valor, servicios de dominio, y abstracciones.

Esta capa debe ser autocontenido, fácilmente testable y aislada de tecnologías externas.

La capa que aparece a continuación es la **capa de aplicación**. La capa de aplicación actúa como un mediador entre la capa exterior y la capa de dominio, coordinando las operaciones de alto nivel que involucran múltiples aspectos del dominio.

La capa más exterior se denomina **capa de infraestructura**. La capa de infraestructura proporciona las capacidades tecnológicas necesarias para que la capa de dominio pueda realizar sus funciones sin tener que preocuparse por los detalles de implementación. Toda la comunicación o interacción con

elementos externos a la aplicación deben estar contenidos dentro de la capa de infraestructura.

El uso de una arquitectura limpia en el desarrollo de este sistema asegura que cada componente en la capa de infraestructura sea fácil de modificar o reemplazar. Como veremos, esto permitirá una integración sencilla de nuevas tecnologías y, por lo tanto, la inclusión del soporte de nuevos tipos de formato o nuevos tipos de documento en el sistema a futuro.

2.3 Procesamiento de lenguaje natural

El procesamiento de lenguaje natural o *Natural Language Processing* (NLP) es un campo que se sitúa en la intersección de la informática, la inteligencia artificial y la lingüística.

Se dedica al desarrollo de sistemas que permiten a las computadoras entender y generar lenguaje humano de una manera útil. La historia del NLP comienza en la década de 1950, marcada por el trabajo pionero de Alan Turing y su famoso test de Turing, que planteaba la cuestión de si una máquina puede emular el lenguaje humano de manera convincente [7].

En los años 60 y 70, el esfuerzo principal estaba en la traducción automática, como los esfuerzos del proyecto Georgetown [8].

Con la introducción de la inteligencia artificial (IA) en la década de 1980, surgieron métodos basados primero en reglas y luego en modelos estadísticos, culminando con el desarrollo de modelos de aprendizaje automático [9] en la década de 1990.

El verdadero cambio paradigmático llegó con el advenimiento de las redes neuronales y el aprendizaje profundo en la década de 2010. Este período vio la creación de modelos de lenguaje avanzados [10], que han revolucionado la capacidad de las máquinas para procesar el lenguaje con un grado de sutileza y profundidad sin precedentes.

2.4 Modelos de lenguaje de gran escala

La historia de los modelos de lenguaje de gran escala o *large language models* (LLMs) comienza con los primeros modelos estadísticos de lenguaje en la década

de 1980, que utilizaban métodos simples como los modelos de Markov y n-gramas para predecir la probabilidad de secuencias de palabras [11].

Estos métodos, aunque efectivos para algunas tareas básicas, estaban limitados por su incapacidad para capturar contextos más largos y por su dependencia de grandes corpus de texto para entrenamiento.

En la década de 2000, con el advenimiento de modelos más sofisticados como los modelos ocultos de Markov y especialmente las redes neuronales, comenzó a vislumbrarse el potencial de los modelos de lenguaje más complejos.

Sin embargo, no fue hasta la introducción de las redes neuronales recurrentes o *recurrent neural networks* (RNN) y, más tarde, las redes neuronales de memoria a largo plazo o *long-term memory neural networks* (LSTM) que los investigadores pudieron abordar el problema del “desvanecimiento del gradiente” y mejorar significativamente la capacidad de los modelos para aprender dependencias a largo plazo en el texto [12].

El verdadero cambio paradigmático llegó en 2017 con el desarrollo de la arquitectura *Transformer* [13]. Esta arquitectura introdujo el mecanismo de atención, que permite a los modelos ponderar diferentes partes de la entrada de texto de manera dinámica, mejorando la capacidad de los modelos para manejar secuencias de texto largas y complejas.

En este proyecto se utilizaron modelos de lenguaje de gran escala como *GPT*, para la extracción de información de documentos, tal y como veremos en la sección 4.4 Implementación y programación.

Principales modelos

GPT

GPT o *Generative Pre-trained Transformer* es un avanzado modelo de lenguaje desarrollado por *OpenAI* [14]. Es uno de los modelos más reconocidos en la actualidad. Aunque es un modelo privado, *OpenAI* ofrece acceso a través de una API de pago que incluye una capa gratuita limitada. *OpenAI* ofrece varios modelos, cada uno con un esquema de precios que varía según su complejidad. Una de las principales ventajas de este servicio es que proporciona una solución *plug-and-play*, es decir, no requiere instalación ni configuración adicional por

parte del usuario.

BERT

BERT, desarrollado por *Google* [10], es un modelo de lenguaje basado en la arquitectura *Transformer* que se entrenó utilizando un gran corpus de texto en una manera bidireccional, lo que le permite entender el contexto de una palabra basada en todas sus apariciones en un texto.

Hugging Face's Transformers

Hugging Face's Transformers es una biblioteca de NLP [15] que simplifica el uso de modelos de aprendizaje profundo basados en la arquitectura *Transformer*.

Esta biblioteca ofrece una amplia variedad de modelos pre entrenados y libres desarrollados por la comunidad para diversos fines. Además de la posibilidad de entrenar modelos personalizados.

En contraposición, los modelos disponibles en *Hugging Face* suelen ser menos potentes que los modelos disponibles a través en plataformas como *OpenAI* [16].

Otros

Además de los descritos existen muchas otras plataformas o proyectos como por ejemplo *Eleuther AI* [17], *Fairseq* [18] o *GPT 2* [19] que ofrecen alternativas LLM para desarrolladores interesados en incorporar este tipo de capacidades a sus aplicaciones.

En este proyecto se utilizarán las tecnologías introducidas en la sección 2.3 Procesamiento de lenguaje natural y sobre todo en esta sección 2.4 Modelos de lenguaje de gran escala para entender y extraer la información relevante de los documentos que describimos en la sección 1.4 Objetivos.

3 Metodología

En este proyecto se utilizaron metodologías ágiles para mejorar la calidad y eficiencia del desarrollo de software. La programación extrema facilitó ciclos de desarrollo cortos y una comunicación fluida con la directora del TFG, mientras que el desarrollo dirigido por pruebas garantizó un diseño bien probado. La metodología iterativa incremental permitió evaluar y ajustar avances rápidamente, y la gestión visual de procesos con Kanban mejoró la visibilidad y gestión del flujo de trabajo.

3.1 Metodología ágil

En este proyecto se utilizaron las siguientes metodologías enmarcadas dentro de las metodologías ágiles. La *programación extrema* introdujo ciclos de desarrollo cortos y frecuentes, así como la comunicación fluida con el resto de miembros del equipo, en este caso con la dirección del TFG. El *desarrollo dirigido por pruebas* ayudó a obtener un diseño óptimo y bien probado, mientras que la *metodología iterativa incremental* permitió evaluar los avances en pequeños pasos, permitiendo tomar decisiones rápidamente cuando fue necesario.

Programación extrema

La programación extrema o *extreme programming* (XP) es una metodología ágil de desarrollo de software que se enfoca en mejorar la calidad del software y la capacidad de respuesta a las necesidades cambiantes del cliente. Fue introducida por Kent Beck en el libro *Extreme Programming Explained: Embrace Change* [20], XP promueve la colaboración intensa entre los desarrolladores y los clientes, ciclos de desarrollo cortos y frecuentes, y la entrega continua de pequeñas mejoras.

Desarrollo dirigido por pruebas

El desarrollo dirigido por pruebas o *test driven development* (TDD) es una práctica de desarrollo de software. Este enfoque fue popularizado por Kent Beck, uno de

los pioneros de las metodologías ágiles, en su libro *Test Driven Development: By Example* [21]. TDD se basa en ciclos cortos de desarrollo donde se escribe una prueba, se implementa el código necesario para pasar la prueba y luego se reescribe el código para mejorar su estructura sin cambiar su comportamiento.

Iterativo incremental

El enfoque iterativo incremental es una metodología utilizada en el desarrollo de software que combina dos modelos: el iterativo y el incremental. Esta metodología es popular en el desarrollo ágil y se centra en desarrollar un sistema a través de ciclos repetidos (iteraciones) y en la construcción gradual de funcionalidad (incrementos).

El término “iterativo” se refiere a la repetición de un conjunto de actividades a lo largo del ciclo de vida del desarrollo del software. En cada iteración, se planea, desarrolla y evalúa una parte del sistema.

El término “incremental” se refiere a la construcción del sistema mediante adiciones sucesivas de componentes y funcionalidades. Cada incremento agrega una parte funcional del sistema hasta que el producto está completo.

3.2 Gestión visual de procesos

Kanban o gestión visual de procesos es un método de gestión de proyectos que se originó en la industria manufacturera japonesa, específicamente en Toyota, como una forma de mejorar la eficiencia de la producción y que se encuentra fuertemente implantado dentro de la industria del desarrollo de software [22].

El término significa “tarjeta visual” en japonés, y el método se basa en el uso de tarjetas visuales para representar tareas en un tablero, permitiendo a los miembros del equipo ver el estado del progreso del trabajo de un simple vistazo.

Kanban promueve la mejora continua, la flexibilidad y la eficiencia, ayudando a los equipos a gestionar el flujo de trabajo y a identificar cuellos de botella rápidamente.

En la figura 3.1 puede una captura de cómo hemos usado *kanban* para la gestión de las tareas, teniendo en todo momento una visión completa del estado de avance de cada *iteración*.

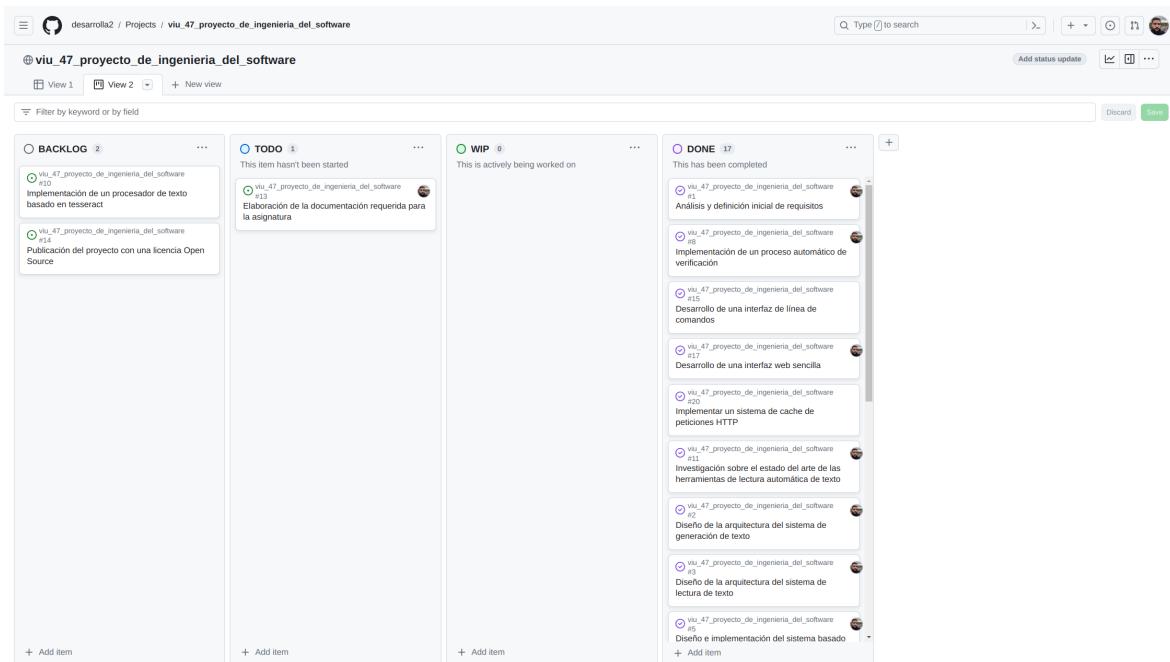


Figura 3.1: Captura del tablero kanban del proyecto

Al inicio del proyecto las tareas fueron definidas e incluidas en la columna **backlog**. Durante el desarrollo del proyecto, si una nueva tarea era identificada, se incluía dentro de esta columna.

Al inicio de cada iteración, se realiza una planificación donde algunas tareas son incluidas dentro de la iteración y las tareas son desplazadas a la columna **todo**. Estas tareas deben encontrarse correctamente definidas y claras y el volumen total del trabajo debe estar adecuado a la capacidad del equipo y del tiempo disponible en la iteración.

Durante la iteración las tareas se van desplazando a la columna **WIP** o *work in progress* para indicar que están siendo realizadas en este momento y finalmente a la columna **DONE**, una vez que son completadas.

Al terminar cada iteración, todo el trabajo no completado, se desplaza de nuevo al *backlog*. Se realizan dos ceremonias, en una denominada *demo* o demostración se realiza una demostración del trabajo realizado, mientras que en la segunda, la *retrospectiva* se realiza un análisis de los puntos fuertes y puntos débiles del proceso de trabajo, con el objetivo de mitigar cualquier problema que pudiera surgir.

A modo de ejemplo añado los videos de ejemplo de una planificación [23], una demo [24] y una retrospectiva [25] de la asignatura Proyecto de Ingeniería del Software.

4 Desarrollo

4.1 Planificación

Este proyecto parte de una base realizada en la asignatura 47 GIIN Proyecto de ingeniería del software de este mismo grado universitario [26].

Objetivos

A partir del trabajo realizado en esa asignatura se plantean los siguientes objetivos:

1. **Definir un nuevo conjunto de datos de prueba,** En el trabajo previo existía un conjunto de datos de prueba. Sin embargo, para este TFG se quería definir un nuevo conjunto, utilizando si fuera posible algún conjunto existente de datos de fuentes abiertas.
2. **Evaluar la precisión del sistema,** se quería evaluar cómo de preciso es el sistema a la hora de la extracción de información de los documentos.
3. **Experimentar y seleccionar uno o más sistemas LLM** en el trabajo previo se había probado con los modelos *ollama* [27] y *gpt-3.5-turbo* [28]. Para este TFG se quería probar con nuevos modelos.
4. **Mejora de las interfaces** se querían realizar mejoras en la capa de presentación de ambas interfaces, en el sentido de hacerlas más visuales, concretamente en lo relacionado con la representación de los datos.
5. **Documentación formal,** será necesaria la elaboración de la documentación requerida para completar satisfactoriamente este TFG.

Debían respetarse además los tres requisitos definidos previamente en el capítulo 1 Introducción.

Plan de trabajo

Una vez que se han identificado los requisitos y los objetivos se estableció un plan de trabajo, que además tal y como se describió en la sección 3.1 Iterativo incremental, al final de cada sprint se realizó una pequeña retrospectiva, para

evaluar que tan bien estaba funcionando el plan de trabajo, así como para planificar la siguiente iteración y modificar si fuera necesario el plan de trabajo restante en función de los resultados obtenidos hasta ese momento.

Iteración 0: Hasta el 24 de marzo

Se realizó el planteamiento inicial del TFG, la elaboración y entrega del anteproyecto. Se realizó la instalación del proyecto y una nueva selección de las herramientas de trabajo.

Iteración 1: del 25 de marzo al 7 de abril

Se realizó una ampliación del conjunto de datos de prueba. Para ello se realizó una investigación sobre distintas bases de datos que ofrecen conjuntos de datos abiertos; sin embargo, estas fuentes de datos ofrecen conjuntos de datos anonimizados, es decir datos como nombres, apellidos, números de documentos, direcciones, no aparecen.

Es por este motivo que se descartó utilizar uno de estos conjuntos de datos.

Iteración 2: del 8 de abril al 21 de abril

Al final de la iteración anterior se determinó que el conjunto de datos de prueba debería realizarse manualmente. Para ello se desarrolló un sistema que permitiría, escribir los contratos en formato *Markdown* el cual es fácil de manipular y luego sería convertido a *PDF* utilizando *LaTeX* y diferentes plantillas para introducir variabilidad en los formatos de entrada.

Iteración 3: del 22 de abril al 5 de mayo

En apenas unos meses desde el final de la asignatura en la que se basa este trabajo, han aparecido una serie de nuevos modelos. Se hacía necesario investigar el funcionamiento de modelos como *Llama 3* [29] y *ChatGPT 4* [30] para determinar si eran más adecuados que los modelos que se utilizaron en dicho trabajo.

Después de realizar una serie de pruebas durante esta iteración, se determinó que si bien *Llama 3* es un modelo mucho más avanzado con respecto a su versión anterior, *ChatGPT 4* demostró ser muy superior, por lo que se determinó que se realizaría una nueva implementación del proyecto utilizando este modelo.

Iteración 4: del 6 de mayo al 19 de mayo

Esta iteración estuvo enfocada en realizar una reimplementación de los procesadores *Residential Lease Agreement Processor* y *Vehicle Sale And Purchase Agreement Processor* que desarrollaremos en la sección 4.3.

Iteración 5: del 20 de mayo al 2 de junio

Si bien durante todo el proyecto se han desarrollado las pruebas unitarias, de integración y funcionales oportunas, esta iteración se dedicó a realizar una revisión formal de cómo de bueno era la implementación realizada en comparación con los datos reales que contiene el conjunto de datos de prueba desarrollado.

En esta iteración, se realizó una primera versión del borrador del TFG. Para ello se utilizó la herramienta web *Google Docs* [31]. Si bien los resultados con esta herramienta fueron muy buenos en cuanto a productividad y acabado de los documentos generados, no cumplía con algunos de los requisitos necesarios para la realización del TFG.

Iteración 6: del 3 de junio al 16 de junio

En este segunda iteración dedicado a la elaboración de la memoria fue necesario realizar una nueva versión de este documento, pero esta vez utilizando *LaTeX* [32], una herramienta más compleja de utilizar pero mucho más adecuada a la hora de realizar este tipo de trabajos.

Iteración 7: del 17 de junio al 30 de junio

La última iteración se dedicó a realizar los ajustes finales y revisión de la memoria, y comenzar a preparar la presentación.

4.2 Tecnologías utilizadas

A continuación, se describen las principales tecnologías utilizadas en el proyecto, categorizadas en distintas áreas según su propósito y aplicación.

Servidor

Las tecnologías de servidor o *backend* son aquellas que se ejecutan en un servidor gestionado por la propia organización. Habitualmente se ocupan de la lógica de negocio y la persistencia de datos.

- **PHP** es un lenguaje de scripting de propósito general que está especialmente diseñado para el desarrollo web [33].
- **Symfony** es un *framework PHP* ampliamente utilizado para desarrollar aplicaciones web. Ofrece una arquitectura robusta y flexible, que facilita el desarrollo de aplicaciones mantenibles y escalables [34].
- **Twig** es un motor de plantillas para *PHP*, utilizado principalmente en proyectos basados en *Symfony* [35]. Un sistema de plantillas permite separar la capa de presentación de la capa de negocio.

En este proyecto hemos utilizado *PHP* como lenguaje principal para desarrollar nuestro sistema, mientras que *Symfony* se utiliza en la parte de infraestructura, incluida la interfaz web, para la que también hemos usado *Twig*.

Cliente

Las tecnologías de cliente o *frontend* son aquellas que se ejecutan en el navegador del usuario de una aplicación web. El *frontend* se encarga de la presentación y la experiencia del usuario.

- **HTML5** es el lenguaje de marcado para crear páginas web. *HTML5* introduce nuevas funcionalidades como elementos semánticos y multimedia, mejorando la accesibilidad y la interoperabilidad de las páginas web [36].
- **CSS3** es la tecnología de estilos para el diseño visual de páginas web. CSS3 permite aplicar estilos complejos y responsivos a los elementos **HTML**, mejorando la presentación y la experiencia del usuario [37].
- **JavaScript** es el lenguaje de programación para la interactividad del lado del cliente. *JavaScript* permite crear experiencias de usuario dinámicas, manejando eventos y actualizando el contenido de la página sin recargar [38].
- **Bootstrap** es un framework de *frontend* que facilita el diseño de sitios y aplicaciones web responsivas y móviles. *Bootstrap* proporciona una colección de componentes CSS y *JavaScript* predefinidos que aceleran el

desarrollo de interfaces de usuario [39].

- **Font Awesome** es una biblioteca de iconos vectoriales que proporciona una amplia gama de iconos escalables y personalizables para su uso en proyectos web. Facilita la inclusión de iconos de alta calidad sin tener que depender de imágenes [40].

En este proyecto hemos utilizado *HTML5*, *CSS3* y *JavaScript* para el desarrollo de la interfaz web. Además, hemos usado *Bootstrap* para realizar un prototipado rápido y finalmente hemos utilizado *Font Awesome* para la representación de iconos y logos.

Gestión de dependencias

La gestión de dependencias es un aspecto crucial en el desarrollo de software. Se encarga de administrar las bibliotecas y paquetes que un proyecto necesita para funcionar correctamente.

- **Composer** es una herramienta de gestión de dependencias para *PHP*, que permite declarar en un fichero las bibliotecas de las que depende el proyecto y asegura que las versiones correctas de las bibliotecas se instalen y mantengan actualizadas [41].

Hemos utilizado *Composer* para instalar las librerías *PHP* necesarias, principalmente *Symfony* y algunos de sus componentes.

Control de versiones

El control de versiones es una práctica en el desarrollo de software que permite rastrear y gestionar los cambios realizados en el código fuente a lo largo del tiempo.

- **Git** es un sistema de control de versiones distribuido, diseñado para manejar desde proyectos pequeños hasta muy grandes con rapidez y eficiencia. *Git* permite a los desarrolladores colaborar de manera efectiva, rastreando cambios y gestionando ramas y fusiones [42].
- **GitHub** es una plataforma de hospedaje de repositorios *Git*. Proporciona herramientas para la colaboración, la revisión de código y facilitando el trabajo en equipo [43].

Hemos utilizado *Git* y *GitHub* para gestionar el código fuente de este proyecto [44].

Se ha seguido el patrón de crear una nueva rama para cada nueva característica y fusionarla con la rama principal una vez completada. Además, hemos vinculado el código con el sistema de gestión de proyectos mediante los mensajes de cada commit.

Automatización

La automatización en el desarrollo de software se refiere al uso de herramientas y scripts para ejecutar tareas repetitivas de manera automática.

- **Make** es una herramienta de automatización de tareas muy popular que utiliza archivos *Makefile* para definir y ejecutar tareas de construcción y gestión de proyectos. Facilita la automatización de tareas repetitivas y la configuración del entorno de desarrollo [45].

Hemos usado *Make* para automatizar las tareas más comunes como son la construcción o la activación de los contenedores de docker, pero también para tener acceso directo a tareas que resultan repetitivas.

Pruebas

Las pruebas en el desarrollo de software son un proceso fundamental para garantizar la calidad y funcionalidad de una aplicación. Consisten en la ejecución de diversos tipos de evaluaciones, como pruebas unitarias, de integración y de sistema.

- **PHPUnit** es un marco de pruebas unitarias para *PHP*. Permite a los desarrolladores escribir y ejecutar pruebas automatizadas para asegurar que el código se comporta como se espera, facilitando el desarrollo de software de alta calidad [46].
- **GitHub Actions** es una plataforma de integración continua que permite automatizar flujos de trabajo directamente desde *GitHub*. *GitHub Actions* facilita la configuración de flujos de trabajo de integración continua, ejecutando pruebas y otras tareas automáticamente con cada cambio en el repositorio [47].

En la sección 4.5 Pruebas y evaluación, veremos que las pruebas son una parte

fundamental de este TFG.

Hemos utilizado *PHPUnit* para la definición y ejecución de una batería de pruebas o *test suite*.

Hemos utilizado *GitHub Actions* para que esta batería de pruebas se ejecutará automáticamente después de cada commit, en un proceso de integración continua.

Contenerización

La contenerización es una técnica de virtualización de sistemas operativos que permite empaquetar aplicaciones y sus dependencias en un contenedor. Estos contenedores aseguran que las aplicaciones se ejecuten de manera consistente en cualquier entorno, independientemente de la configuración del sistema subyacente.

- **Docker** es una plataforma que permite desarrollar, enviar y ejecutar aplicaciones dentro de contenedores. Proporciona un entorno consistente para el desarrollo, prueba y puesta en producción, asegurando que las aplicaciones funcionen de manera idéntica en diferentes entornos [48].
- **Docker Compose** es una herramienta para definir y ejecutar aplicaciones Docker multi-contenedor. Permite orquestar varios contenedores docker que componen una aplicación, facilitando la configuración y gestión de entornos de desarrollo más complejos [49].

Hemos utilizado *Docker* para definir dos conjuntos de contenedores, uno básico y otro extendido.

El conjunto básico contiene un único contenedor en el que se ejecuta la aplicación. El conjunto extendido contiene servicios que son complementarios, en este caso el conjunto *ELK* descrito en la subsección 4.2 Registros.

Registros

Los registros o *logs* contienen información de eventos que ocurren en un sistema, son utilizados para monitorear, diagnosticar y solucionar problemas. Capturan información vital sobre el funcionamiento de aplicaciones, servidores y otros componentes, proporcionando un historial que pueda ser analizado.

- **Monolog** es una biblioteca de *logs* para *PHP*. Permite enviar registros a varios destinos, como archivos, bases de datos y servicios de terceros, facilitando la monitorización y depuración de aplicaciones [50].
- **Elasticsearch** es un motor de búsqueda y análisis de texto completo o *full text search*. Permite almacenar y analizar grandes volúmenes de datos en tiempo real [51].
- **Logstash** es una herramienta de procesamiento de datos que procesa, transforma y envía datos a un sistema de destino [52].
- **Kibana** es una herramienta de visualización de datos que trabaja en conjunto con *Elasticsearch*. Permite a los usuarios crear gráficos y dashboards interactivos para visualizar y analizar los datos de *logs* almacenados en *Elasticsearch* [53].

Hemos utilizado *Monolog* para configurar los *logs* generados en la aplicación. El resto de elementos, *Elasticsearch*, *Logstash* y *Kibana*, conocidos como *ELK*, permiten interpretar en tiempo real los registros de la aplicación.

En la sección 4.4 Implementación y programación, ampliamos los detalles de cómo hemos configurado este sistema.

Componente Generator

El componente *Generator* está descrito en la subsección 4.3 Componente *Generator*.

- **Pdf to Text** es una herramienta que convierte documentos *PDF* en texto plano. Permite extraer el contenido textual de archivos *PDF*, lo cual es útil para la posterior manipulación y análisis [54].

Utilizamos esta tecnología dentro del procesador *PDF To Text Processor* para la conversión de documentos *PDF* en texto plano.

Componente Reader

El componente *Generator* está descrito en la subsección 4.3 Componente *Reader*.

- **ChatGPT API** proporciona acceso a modelos avanzados de procesamiento de lenguaje natural. Permite integrar capacidades de *IA* en una aplicación, como la generación de texto y el análisis de datos, mejorando las

funcionalidades y experiencias del usuario [55].

- **Symfony HttpClient** es un cliente *HTTP* flexible y eficiente para *PHP*. Permite realizar solicitudes *HTTP* a servicios externos, manejar respuestas y gestionar errores de manera sencilla y eficaz [56].
- **Symfony Cache** es un componente de *Symfony* que proporciona una implementación robusta y flexible para el almacenamiento en caché [57]. La cache permite mejorar el rendimiento de la aplicación mediante el almacenamiento temporal de datos, reduciendo la carga en los recursos externos.

Hemos utilizado la *API* de *ChatGPT*, para extraer la información de los documentos en los procesadores *Residential Lease Agreement Processor* y *Vehicle Sale And Purchase Agreement Processor*.

Las llamadas *HTTP* se realizaron mediante la librería *Symfony HttpClient* y finalmente *Symfony Cache* almacena las respuestas.

Veremos más en profundidad este proceso en la sección 4.4 Implementación y programación.

4.3 Diseño del sistema

En la figura 1.1 vimos un esquema general del sistema, el cual recibe documentos en formato *PDF* y genera objetos que contienen la información más importante de los mismos. Dependiendo del tipo de documento, la información contenida en estos objetos puede variar.

En la figura 1.3 vimos que se va a incluir también el desarrollo de dos interfaces, una web y una de línea de comandos.

En esta sección profundizaremos en el diseño de la arquitectura de este sistema. En la figura 4.1 vemos que el sistema tiene dos componentes principales: el componente *Generator* y el componente *Reader*. El primero será responsable de convertir los documentos en formato texto y el segundo de extraer la información.

Componente Generator

La responsabilidad de este primer componente, el *Generator* es convertir archivos de cualquier formato en una representación en texto plano del mismo. Tal y como

Recibe un documento de tipo PDF

Estructura de datos con la
información relevante

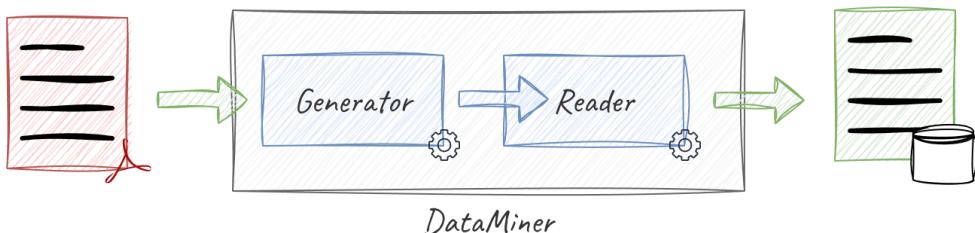


Figura 4.1: Esquema de los componentes principales del sistema

se ve en la figura 4.2 con el esquema UML el componente consta de cuatro tipos diferentes de elementos.

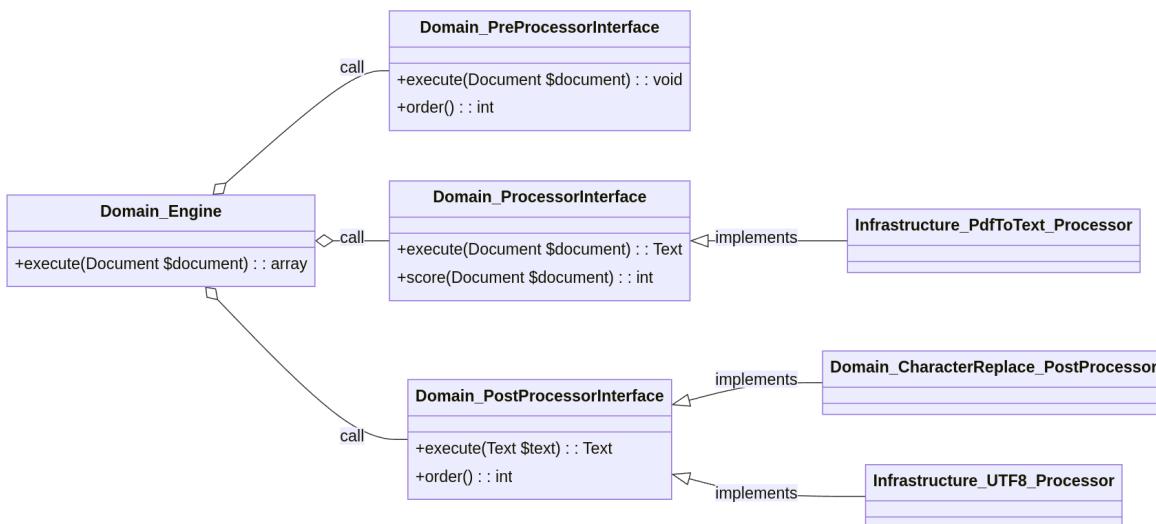


Figura 4.2: Diagrama UML del componente *Generator*

Es importante destacar que salvo el *Engine*, el resto de elementos son intercambiables, y dependiendo del caso de uso, pueden utilizarse los componentes descritos en este TFG, o ser necesaria la implementación de otros nuevos.

El motor o *Engine* es el corazón del componente. Es el encargado de realizar las llamadas a los demás elementos registrados en el mismo coordinando cuáles y en qué orden deben ser llamados.

Los *pre procesadores* preparan el documento original para facilitar su conversión a texto. En esta implementación no se ha desarrollado ningún preprocesador, ya

que no ha sido necesario para los casos de uso previstos.

Los procesadores realizan la conversión efectiva del documento a texto. Funcionan mediante un sistema competitivo donde varios procesadores evalúan su propia aptitud para manejar el formato del en cuestión, seleccionando cuál es el más adecuado para llevar a cabo la tarea.

En la figura 4.3 puede verse un ejemplo. El *Engine* pretende convertir un fichero PDF en texto y tiene registrados dos hipotéticos procesadores: el MP3 Processor y el PDF Processor.

Primero realiza una llamada a *MP3 Processor*, el cual evalúa el documento y responde con una puntuación de -1, lo cual indica que no es capaz de procesar este tipo de documento.

A continuación realiza una segunda llamada a *PDF Processor*, el cual evalúa el documento y responde con una puntuación de 90, lo cual significa que es un procesador adecuado para procesar este formato de documento.

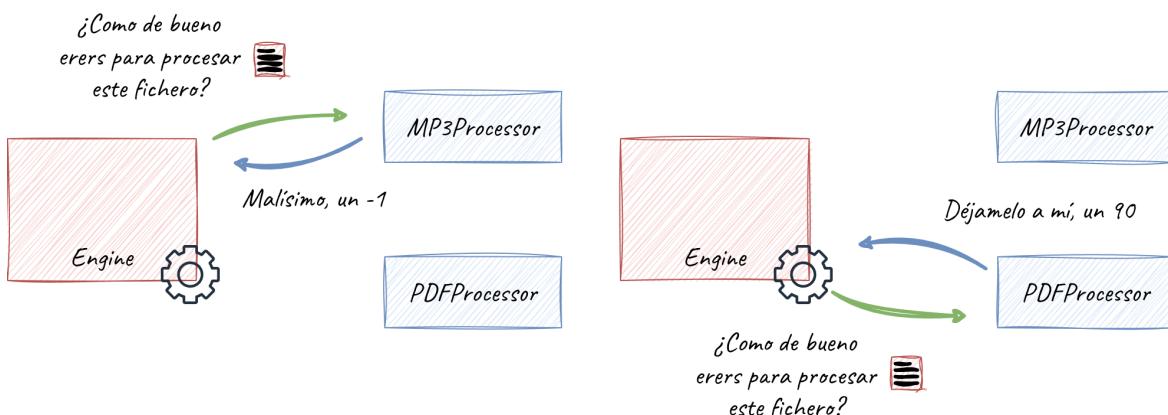


Figura 4.3: Esquema de la competición entre procesadores del componente Generator

Si existieran más procesadores registrados, se irían invocando secuencialmente, para que evaluaran como de adecuados son para procesar ese tipo de documento. Finalmente, una vez que todos los procesadores han sido invocados, el *Engine* enviará el documento al procesador que haya respondido con una puntuación más alta.

Como puede verse, es fácil añadir soporte a nuevos tipos de documentos. Se podría añadir soporte para documentos escaneados simplemente añadiendo un

procesador que utilice una tecnología de reconocimiento de caracteres u *Optical character recognition* (OCR) como por ejemplo *Tesseract OCR* [58].

El **Requisito 1** definido en la sección 1.4 Objetivos para este proyecto era “convertir documentos PDF en documentos de texto”.

El procesador *PDF To Text Processor* es el responsable de esta tarea mediante llamadas a una tecnología que se encuentra en la capa de infraestructura, en este caso *Pdf to Text* [54]. Añadiremos más detalles sobre como fue la implementación de este procesador en la sección 4.4 Implementación y programación.

Los **post procesadores** perfeccionan el texto generado, realizando los ajustes necesarios, lo que mejora la calidad del texto resultante.

Los post procesadores se llaman secuencialmente. El primer post procesador recibe como entrada la salida del procesador ejecutado. Los siguientes post procesadores reciben como entrada la salida del post procesador anterior.

En esta implementación, se han desarrollado los siguientes post procesadores:

- **UTF8 Post Processor:** Ya que algunos documentos pueden contener caracteres no *UTF8* este procesador los reemplazará por caracteres equivalentes en *UTF8*.
- **Character Replace Post Processor:** Después de eliminar los caracteres no *UTF8* se detectó que algunos caracteres todavía podrían resultar problemáticos, así que se implementó un nuevo procesador capaz de reemplazar caracteres problemáticos, por otros equivalentes. En este caso se intercambió el carácter salto de página *\f*, por el carácter salto de línea *\n*.
- **Word Limit Post Processor:** Más adelante en la implementación del componente *Reader*, y debido al uso de las tecnologías implementadas en el mismo, se vio que documentos con textos demasiado largos podrían generar errores. Es por esto que se decidió implementar este procesador que corta el texto después de las primeras N palabras.

En conjunto al ejecutarse los tres, primero se sustituyen los caracteres no *UTF8*, por otros equivalentes, a continuación se sustituye el carácter salto de página. Finalmente, se determinó, para el conjunto de datos de prueba que la información relevante se encuentra siempre dentro de las primeras dos páginas, y que en promedio cada página contenía unas cuatrocientas palabras, por motivos de

rendimiento y coste se limitó el tamaño del texto resultante a mil palabras.

Componente Reader

El componente Reader es el responsable de interpretar y procesar el texto plano obtenido a partir de la salida del componente Generator descrito en la subsección anterior.

Tal y como se ve en la figura 4.4 el componente consta de dos tipos diferentes de elementos: El motor o *Engine* es el corazón del componente. Es el encargado de hacer las llamadas a los demás elementos registrados en la aplicación, en este caso únicamente los procesadores.

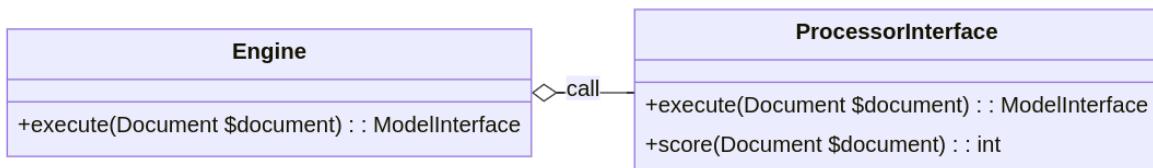


Figura 4.4: Diagrama UML del componente Reader

Los procesadores están organizados en una única capa y operan bajo el mismo mecanismo competitivo descrito en la subsección 4.3 Componente Generator.

Cada uno de estos procesadores es invocado secuencialmente para evaluar su idoneidad en el manejo de un tipo de documento específico.

Una vez seleccionado, el procesador elegido procede a ejecutar una serie de tareas que incluyen la identificación y extracción de la información clave.

En esta implementación, se han desarrollado los siguientes procesadores:

- **Residential Lease Agreement Processor** adecuado para evaluar contratos de arrendamiento de vivienda entre particulares tal y como indicamos en **Requisito 2**.
- **Vehicle Sale And Purchase Agreement Processor** adecuado para evaluar contratos de compra venta de vehículo entre particulares tal y como indicamos en **Requisito 3**.

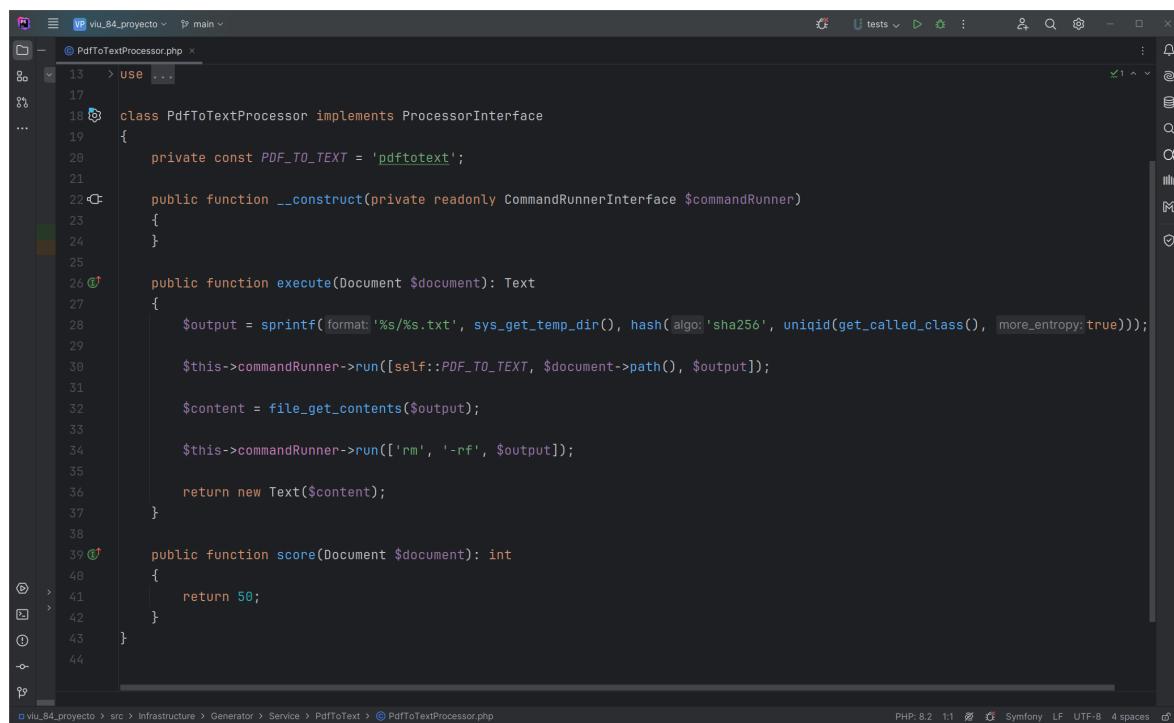
4.4 Implementación y programación

En esta sección vamos a explicar los detalles de implementación más importantes de aquellos componentes que creemos que es interesante que sean introducidos.

Componente Generator

Cómo introducimos en la sección 4.3 Diseño del sistema, el elemento clave de este componente es el *PDF To Text Processor*. Este procesador es responsable de convertir un fichero *PDF* en su representación en texto plano.

Tal y como puede verse en el extracto de código de la figura 4.5, el procesador *PDF To Text Processor* es simplemente un envoltorio o *wrapper* que utiliza el componente *Command Runner* para realizar llamadas a *Pdf To Text* en la línea de comandos.



```
use ...;

class PdfToTextProcessor implements ProcessorInterface
{
    private const PDF_TO_TEXT = 'pdftotext';

    public function __construct(private readonly CommandRunnerInterface $commandRunner)
    {
    }

    public function execute(Document $document): Text
    {
        $output = sprintf(format: '%s/%s.txt', sys_get_temp_dir(), hash(algo: 'sha256', uniqid(get_called_class(), more_entropy: true)));

        $this->commandRunner->run([self::PDF_TO_TEXT, $document->path(), $output]);

        $content = file_get_contents($output);

        $this->commandRunner->run(['rm', '-rf', $output]);

        return new Text($content);
    }

    public function score(Document $document): int
    {
        return 50;
    }
}
```

Figura 4.5: Captura del código fuente del procesador *PDF To Text Processor*

En la figura 4.6 puede verse un esquema de este comportamiento: el motor envía el documento *PDF* al *PDF To Text Processor*, este a su vez lo envía a *PDF To Text* el cual genera un fichero de texto plano. Finalmente, el *PDF To Text Processor* extrae el contenido de ese fichero y lo envía de vuelta al motor, donde comenzó todo.

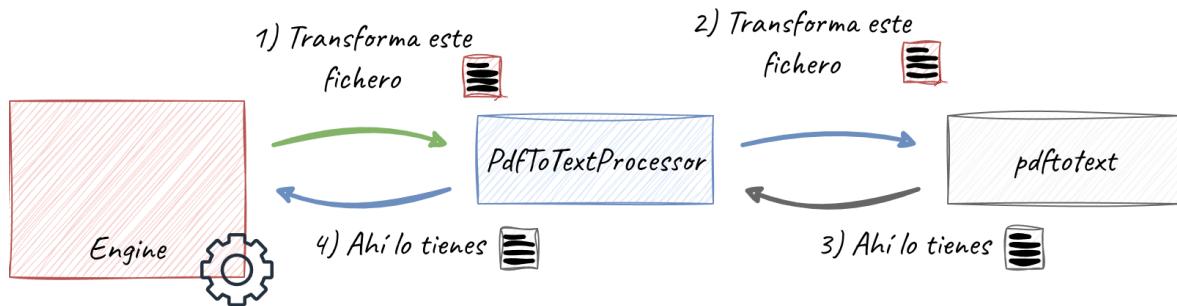


Figura 4.6: Esquema de comunicación con la herramienta de la capa de infraestructura pdftotext

Componente Reader

De este componente queremos destacar dos elementos: el *Residential Lease Agreement Processor* y el *Vehicle Sale And Purchase Agreement Processor* responsables de extraer la información de los contratos de alquiler de vivienda entre particulares y la compraventa de vehículos entre particulares respectivamente.

Cómo introducimos en la sección 4.3 Diseño del sistema, el procesador *Residential Lease Agreement Processor* es responsable de transformar el texto de un contrato de arrendamiento de vivienda entre particulares en un objeto con la información fundamental.

La implementación de este procesador es muy similar a la que hemos visto en el *PDF To Text Processor*, por lo que no vamos a incluir ni el código ni el esquema con la comunicación.

Simplemente, indicaremos que se trata de enviar peticiones sobre un documento que previamente hemos convertido en texto plano a un sistema que lo pueda procesar, en este caso nos hemos decantado por *ChatGPT 4 Turbo* [30]

Si merece la pena profundizar en las características de las peticiones que se envían. La parte más compleja de estos componentes es encontrar un *prompt* o entrada lo suficientemente precisa para que genere una respuesta que pueda ser utilizada a continuación.

Para encontrar una entrada suficientemente precisa, utilizamos dos técnicas:

La primera *Few-shot Prompting* [59] es una técnica en la cual se proporcionan al modelo ejemplos de entrada y salida para que aprenda el patrón específico.

La segunda técnica es el *Format-based Prompting* [60], una técnica en la cual se estructura el prompt para guiar al modelo a generar respuestas en un formato específico, como *JSON*, listas o tablas, las cuales pueden ser luego fácilmente parseadas. Por ejemplo para una persona se podría indicar que la respuesta debería ser similar a esto: `{"name": "John", "surname": "Doe", "number": "12345678A"}`.

El primer intento de implementación se trató que el modelo respondiera con un *XML*, utilizando ejemplos en formato *XSD*. Sin embargo, *ChatGPT 4* tiene limitaciones para trabajar con documentos *XML*, por lo que la implementación final se realizó utilizando modelos *JSON*.

Todo esto puede verse en el ejemplo de código que aparece en la figura 4.7, donde se muestra el código del método que define el *prompt*, donde puede apreciarse que la construcción correcta del *prompt* es importante para obtener una respuesta adecuada.

```

10  readonly class ResidentialLeaseAgreementProcessor extends AbstractAgreementProcessor
11  {
12      protected function getTemplateForContent(): string
13      {
14          return <><EO0>
15          Extrae la siguiente información desde el texto dado:
16
17          1. Fecha del acuerdo
18          2. Arrendatarios ( nombre, apellidos, número de documento )
19          3. Arrendadores ( nombre, apellidos, número de documento )
20          4. Dirección ( calle, ciudad, país, código postal )
21          5. Importe mensual de la renta.
22
23          Text:
24          <<
25
26          Por favor proporciona la información en el siguiente formato:
27
28          [
29              "date": "YYYY-MM-DD",
30              "landlords": [
31                  {
32                      "name": "John",
33                      "surname": "Doe",
34                      "number": "12345678A"
35                  },
36              ],
37              "tenants": [
38                  {
39                      "name": "Jane",
40                      "surname": "Smith",
41                      "number": "87654321B"
42                  }
43              ],
44              "property": {
45                  "address": {
46                      "address": "123 Main St",
47                      "city": "Madrid",
48                      "country": "Spain",
49                      "postal_code": "28004"
50                  }
51              },
52              "monthly_rent": "12000.00"
53          ]
54
55          Asegurate que la respuesta es un JSON válido.
56          EO0:
57      }
58
59  }
60
61  <<EO0>
62  <<EO0>
63
64  (App\Infrastructure\Reader\Service\ChatGPT -> ResidentialLeaseAgreementProcessor -> getTemplateForContent)

```

Figura 4.7: Captura del código fuente del procesador *Residential Lease Agreement Processor*

Una vez que se obtiene la respuesta en formato *JSON*, tan solo es necesario des serializar el mismo para convertirlo en un objeto del tipo indicado, en este caso un *Residential Lease Agreement* que contiene los tipos de datos esperados para un contrato de arrendamiento de vivienda entre particulares: nombres, apellidos y documentos de identidad de los arrendatarios y los arrendadores, la dirección del inmueble, la renta mensual y la fecha del contrato.

Sin embargo, llegados a este punto comprobamos que la *API* de *ChatGPT 4* presenta dos limitaciones que deben ser tenidas en cuenta: las peticiones son lentas y cuestan dinero. Los tiempos de respuesta pueden variar dependiendo del tipo de documento y del estado de la api. Los costes también pueden variar dependiendo del tipo de documento y del precio de la api en ese momento [61].

A continuación una tabla obtenida con el promedio de realizar 100 peticiones de procesamiento para el documento *001/001.pdf*.

Nombre	Valor
Total peticiones	3
Tiempo total	6.74 segundos
Total <i>input tokens</i>	6643
Total <i>output tokens</i>	159
Precio total	0.07 euros

Tabla 4.1: Resumen de los resultados de la ejecución procesador *Residential Lease Agreement Processor*

Un token en un modelo *ChatGPT* es una unidad básica de texto que puede ser una palabra, parte de una palabra o un carácter de puntuación.

Aunque los modelos *ChatGPT* tienen limitaciones en cuanto al tamaño máximo del contexto, como se muestra en la tabla 4.2, estas limitaciones no afectan la ejecución de este procesador [62].

Como ya dijimos, el post procesador *Word Limit Post Processor*, descrito en la sección 4.3 Diseño del sistema, asume que la información relevante para este caso de uso se encuentra dentro de las primeras mil palabras de cada documento, truncándose cualquier información que exceda este límite.

Modelo	Tokens por contexto
GPT-4 turbo	128,000 tokens
GPT-4	8,192 tokens
GPT-3.5 turbo	16,385 tokens

Tabla 4.2: Límite de tokens por contexto para los modelos más recientes de *ChatGPT*

Cabe mencionar que durante la fase de desarrollo de este proyecto se repitieron continuamente los mismos documentos, en un ciclo de prueba y error. Por lo que tanto para acelerar los tiempos de ejecución como para reducir los costes se hizo recomendable implementar un sistema de caché.

El sistema de caché genera un hash para cada petición y guarda la respuesta. Si se repite una misma petición recupera la respuesta de la caché, por lo que la petición se resuelve de forma prácticamente instantánea y sin ningún coste adicional.

Vehicle Sale And Purchase Agreement Processor

Este procesador es responsable de transformar el texto de un contrato de compraventa de vehículo entre particulares en un objeto con sus datos fundamentales.

Este procesador funciona de forma tan similar al procesador anterior que no será necesario entrar en nuevos detalles con respecto a su funcionamiento, hasta el punto de que en el momento del desarrollo fue evidente que podrían compartir una parte muy importante del código, simplemente cambiando ligeramente las peticiones que se debían realizar.

Registros

El registro de *logs* es una parte crucial del monitoreo y mantenimiento de cualquier aplicación. En proyectos de este tipo, por la cantidad de pequeños componentes y por interactuar con elementos de infraestructura externos, se hace necesario contar con un buen sistema de registro de logs.

En este proyecto, se ha implementado un sistema de registros utilizando

monolog, una biblioteca de registro para *PHP*.

Se han configurado tres canales.

- *Generator*: para los logs del componente *generator*.
- *Reader*: para los logs del componente *reader*.
- *HTTP*: para el componente que realiza las peticiones *HTTP*.

Un canal es la forma en la que *monolog*, agrupa un conjunto de información para poder filtrar y procesar adecuadamente. Además, los logs se almacenan en dos ficheros y formatos diferentes:

- **Monolog format**, es el formato estándar de ficheros de log generados por esta biblioteca.
- **Logstash format**, es el estándar de la herramienta *monolog*.

El formato *Monolog format* está indicado para entornos de desarrollo o proyectos de pequeña envergadura. Siempre que estos ficheros no sean demasiado grandes, se pueden trabajar a través de herramientas de línea de comandos como *grep* o *awk*.

El formato *Logstash format* es adecuado para ser procesado por un sistema compuesto por *Elasticsearch*, *Logstash* y *Kibana* (ELK). Este formato está indicado en entornos de producción, donde la monitorización de los logs sea una tarea relevante.

En la figura 4.8 puede verse un esquema de este comportamiento.

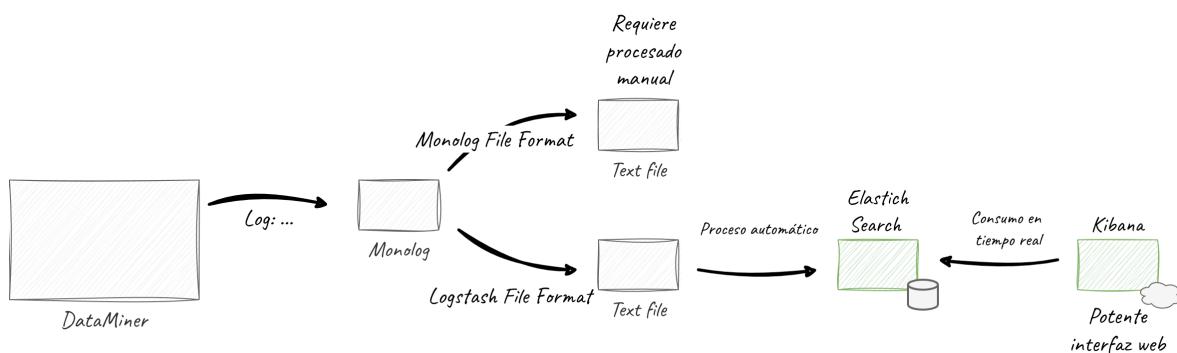


Figura 4.8: Esquema del sistema de procesamiento de logs

4.5 Pruebas y evaluación

Desarrollo del conjunto de datos de prueba

Una parte fundamental de este TFG fue construir un conjunto de pruebas que permitiera dar soporte tanto al desarrollo como a la validación del proyecto.

En una primera aproximación se realizó una investigación sobre distintas bases de datos que ofrecen conjuntos de datos abiertos, aquí una lista de algunas de las que investigamos:

- **Data.gov**: Portal de datos abiertos del gobierno de Estados Unidos [63].
- **European Union Open Data Portal**: Portal de datos abiertos de la Unión Europea [64].
- **Kaggle Datasets**: Conjuntos de datos para competencias y proyectos de análisis de datos [65].
- **Google Dataset Search**: Motor de búsqueda de conjuntos de datos [66].
- **Amazon Web Services (AWS) Public Datasets**: Conjuntos de datos públicos de AWS [67].

Sin embargo, no fue posible encontrar un conjunto adecuado. Los conjuntos de datos publicados en estos portales presentan datos anonimizados. No es posible encontrar datos como nombres, documentos de identidad, direcciones, etc., por lo que este enfoque no fue adecuado.

Se determinó, que el conjunto de datos de prueba debería generarse manualmente a partir de modelos descargados de internet.

También se determinó que los conjuntos de datos deberían ser lo suficientemente grandes como para realizar pruebas que permitieran alcanzar conclusiones relevantes en términos de precisión del sistema, pero lo suficientemente pequeños como para que la ejecución completa de la batería no fuera excesivamente lenta ni cara.

Dado que en la sección 4.4 Implementación y programación, se había estipulado que el coste de la ejecución de un documento eran aproximadamente siete céntimos de euro, se consideró que de nueve documentos para los conjuntos de datos de prueba y de tres elementos para los conjuntos de datos de contraste, ofrecía un balance adecuado.

En la tabla 4.3 pueden verse los conjuntos de datos de prueba desarrollados.

Conjunto	Contenido	Documentos
000	Datos de contraste	3
001	Contratos de arrendamiento de vivienda entre particulares	9
002	Contratos de compra venta de vehículo entre particulares	9

Tabla 4.3: Resumen de los conjuntos de datos de prueba

El primer conjunto de datos **000** contiene ruido o documentos de contraste es decir documentos que no corresponden a ninguno de los otros conjuntos.

El conjunto de datos **001** contiene modelos descargados de diferentes fuentes de internet. Se modificaron estos documentos con datos generados aleatoriamente para indicar nombres, apellidos, direcciones, etc., que no se correspondan con datos reales de personas físicas. Finalmente, el conjunto de datos **002** se generó de la misma forma que el anterior.

Este proceso manual aseguró tanto un conjunto de datos adecuado como el cumplimiento de la normativa de protección de datos.

Construcción del conjunto de datos de prueba

Para permitir modificar fácilmente los documentos de los diferentes conjuntos, estos fueron transformados a formato *Markdown* más sencillo de modificar, que los documentos originales en formato *Word*.

A través de *LaTeX* y un conjunto definido de plantillas para introducir variabilidad en los formatos de entrada, estos documentos son convertidos a formato *PDF*. En la figura 4.9 puede verse un esquema de este comportamiento.

Pruebas automáticas

Las pruebas automáticas son esenciales para garantizar la calidad y la robustez del software.

En este proyecto, se han implementado varios tipos de pruebas automáticas, cada una con un enfoque diferente para cubrir todas las facetas del desarrollo y la

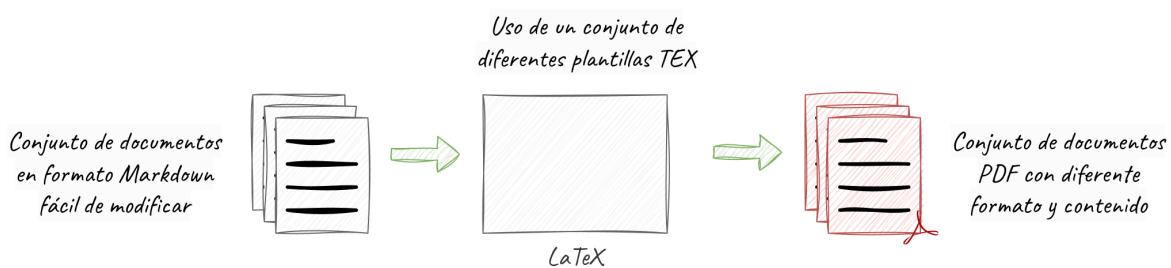


Figura 4.9: Esquema del sistema de construcción automática de documentos en el conjunto de datos de prueba

implementación del software.

1. Las **pruebas unitarias** verifican el funcionamiento de componentes individuales del sistema, como funciones o métodos.
2. Las **pruebas integración** comprueban el correcto funcionamiento de diferentes componentes interactuando
3. coordinadamente.
4. Las **pruebas funcionales** evalúan el sistema desde el punto de vista del usuario final.

En la figura 4.10 puede verse el resultado de la ejecución de la batería de pruebas [68].

Integración continua

La integración continua es una práctica esencial en el desarrollo de software moderno, que implica la integración frecuente y la ejecución automática de los test. Normalmente programado para ejecutarse cada vez que se añade un cambio al repositorio.

En este proyecto, hemos implementado la integración continua utilizando *GitHub Actions*, un servicio de automatización que permite crear flujos de trabajo personalizados para compilar, probar y poner en producción código directamente desde *GitHub*.

En la figura 4.11 puede verse el resultado de la ejecución de la batería de pruebas en el entorno de integración continua.

The screenshot shows the PhpStorm IDE interface with the code editor displaying the `SymfonyHttpClient.php` file. Below the editor is the test runner interface. The test results table shows 36 tests passed in 185ms. A large green progress bar indicates the test completion. A yellow bar at the bottom right of the results table highlights 14 remaining indirect deprecation notices.

Figura 4.10: Captura del resultado de la ejecución de la batería de pruebas automáticas en el entorno de desarrollo

The screenshot shows a GitHub Actions pipeline named "fix absolute paths" for the repository "desarrolla2 / viu_84_proyecto". The pipeline summary shows the job status. The detailed log shows the test execution process, including setting up the environment, cloning the repository, copying default configuration, building a Docker image, installing Composer dependencies, and running tests. The log output includes PHPUnit test results and deprecation notices.

```

test
Succeeded 4 days ago in 1m 48s

1  ➔ Run docker-compose run -T php bash -c "vendor/bin/phpunit --testsuite integration --testsuite unit"
2  4 Creating viu_84_proyecto.php_run ...
3  5 Creating viu_84_proyecto.php_run ... done
4  6 PHPUnit 9.6.19 by Sebastian Bergmann and contributors.
5  7
6  8 Testing
7  9 ..... 22 / 22 (100%)
8  10 Time: 00:00.127, Memory: 8.00 MB
9  11
10 12 OK (22 tests, 27 assertions)
11 13
12 14 Remaining indirect deprecation notices (14)
13 15
14 16 7x: Function utf8_decode() is deprecated
15 17 7x in UTF8PostProcessorTest::testPostProcessor from App\Tests\Unit\Infrastructure\Generator\Service\UTF8
16 18
17 19 6x: Implicit conversion from float 3.640625 to int loses precision
20 21 6x in UTF8PostProcessorTest::testPostProcessor from App\Tests\Unit\Infrastructure\Generator\Service\UTF8
22 23
23 1x: Implicit conversion from float 3.046875 to int loses precision
24 24 1x in UTF8PostProcessorTest::testPostProcessor from App\Tests\Unit\Infrastructure\Generator\Service\UTF8
25

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

```

Figura 4.11: Captura del resultado de la ejecución de la batería de pruebas automáticas en el entorno de integración continua

5 Resultados

5.1 Descripción detallada de la solución informática desarrollada

Se ha desarrollado un sistema capaz de extraer información de documentos de cualquier tipo y formato. Concretamente, tal y como se describe en la sección 1.4 Objetivos el sistema tiene soporte para documentos en formato *PDF* y para dos tipos de contrato, los contratos de alquiler de vivienda entre particulares y los contratos de compraventa de vehículo entre particulares.

Para el desarrollo, se ha seguido un conjunto de buenas prácticas como el uso código limpio y de una arquitectura limpia, descritas en las secciones 2.1 y 2.2 facilitan que la solución pueda ser extendida para implementar nuevos casos de uso.

El sistema está dividido en dos componentes, el primer componente, el componente *Generator*, tiene la capacidad para convertir documentos *PDF* en texto, tal y como se define en el **Requisito 1** que se definió en la sección 1.4 Objetivos.

Aunque en esta implementación se ha utilizado un sistema concreto para la conversión de documentos *PDF* en texto, el sistema permite incorporar otras implementaciones que realicen la conversión de otros formatos utilizando otros elementos de infraestructura.

El sistema tiene un segundo componente, el componente *Reader*, capaz de extraer información a partir de la salida del componente anterior. Se ha implementado soporte para dos tipos de documentos concretos: contratos de arrendamiento de vivienda entre particulares (**Requisito 2**) y contratos de compraventa de vehículos entre particulares (**Requisito 3**).

Aunque el sistema está diseñado para usarse en modo librería, integrado dentro de otros sistemas, se desarrollaron dos interfaces.

La interfaz web fue desarrollada con un diseño adaptativo o *responsive*, lo que

significa que se adapta al tamaño de la pantalla del dispositivo en el que se visualiza, como un ordenador de escritorio o un teléfono móvil.

En esta interfaz, al arrastrar y soltar un documento sobre el cuadro de diálogo, la web lo procesará y mostrará un mensaje con el resultado, que puede ser un error o los datos de un documento soportado en formato **JSON**. En la figura 5.1 se puede ver el resultado de arrastrar y soltar un contrato de arrendamiento de vivienda entre particulares.

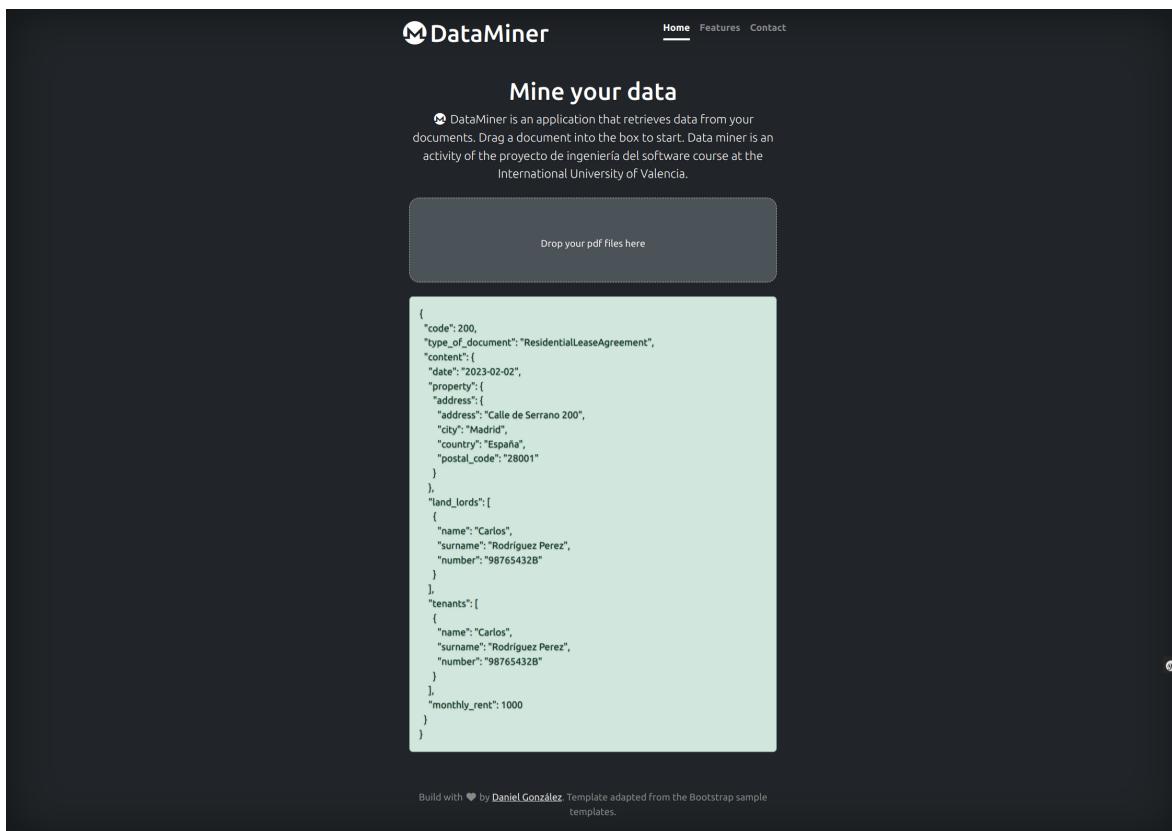


Figura 5.1: Captura del resultado de procesar un documento a través de la interfaz web

La interfaz de línea de comandos está desarrollada de forma que se debe pasar como parámetro la ruta al documento. En la figura 5.2 se puede ver el resultado de ejecutar un contrato de compraventa de vehículo entre particulares.

5.2 Rendimiento y eficiencia

El núcleo del sistema desarrollado es realmente liviano. Se trata de un pequeño conjunto de clases en **PHP** sin ningún tipo de dependencias externas, por

```

docker-compose exec php bash -c "php bin/console app:run /var/www/tests/data/output/002/001.pdf"
+-----+-----+
| Key  | Value |
+-----+-----+
| code | 200 |
| type_of_document | VehicleSaleAndPurchaseAgreement |
| content_date | 2024-05-15 |
| content.vehicle.make | Toyota |
| content.vehicle.model | COROLLA |
| content.vehicle.license_plate | 1234 ABC |
| content.sellers.0.name | Juan |
| content.sellers.0.surname | Pérez Martínez |
| content.sellers.0.number | 12345678A |
| content.buyers.0.name | María |
| content.buyers.0.surname | García López |
| content.buyers.0.number | 87654321B |
| content.price | 5000 |
+-----+-----+
make: Leaving directory '/home/daniel/Projects/desarrolla2/viu_84_proyecto'
Process finished with exit code 0

```

Figura 5.2: Captura del resultado de procesar un documento a través de la interfaz de línea de comandos

lo que su tiempo de ejecución y consumo de recursos pueden considerarse despreciables.

Sin embargo, tal y como se puede ver en la figura 5.3, la ejecución del sistema está ligada a las herramientas en la capa de infraestructura, que son las responsables finales de realizar el trabajo.

En este caso, el rendimiento y el consumo de recursos están directamente ligados a las herramientas seleccionadas y al documento que se esté analizando.

El rendimiento también estará relacionado con las características del entorno en el que se está ejecutando y la carga del mismo.

Sin embargo, hemos calculado los tiempos de ejecución y el consumo de memoria para el documento *001/001.pdf* con el objetivo de establecer una referencia, tal y como se aprecia en la tabla 5.1.

Como puede verse, el consumo de memoria son relativamente bajos, además la memoria se libera en el momento de completarse cada ejecución. Los tiempos de

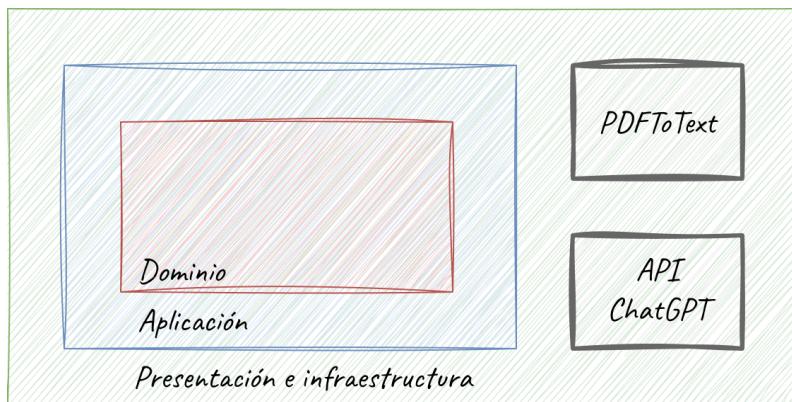


Figura 5.3: Esquema de las herramientas en la capa de infraestructura

Nombre	Ejecución	Memoria	Tiempo
PDF To Text	Local	288Kb	42.49 milisegundos
API de ChatGPT	Remota	612Kb	6.74 segundos

Tabla 5.1: Evaluación del rendimiento de las herramientas de la capa de infraestructura

respuetas son muy bajos para *PDF To Text*, que se ejecuta en local, mientras que son en el orden de cien veces más elevados para los que dependen del consumo de la API remota de *ChatGPT*.

Aun así, los tiempos de respuesta para la extracción de la información son realmente buenos si los comparamos con el tiempo que sería necesario para la extracción manual por parte de un usuario.

Possiblemente, estos tiempos de respuesta requieran en la mayoría de los casos una ejecución asíncrona, donde el usuario sube un paquete de documentación y esta se marca con el estado “*PROCESANDO*” mientras que una tarea es enviada a una cola, donde tras ser ejecutada marcará el paquete como “*PROCESADO*”.

5.3 Calidad del código y mantenibilidad

La calidad del código y su mantenibilidad es elevada gracias al uso de los estándares de la industria descritos en las secciones 2.1 Código limpio y 2.2 Arquitectura limpia.

Para evaluar la calidad del código, tendremos en cuenta los siguientes apartados.

Estándares de codificación

En cuanto a los estándares de codificación, se han seguido los estándares definidos por el marco de trabajo de *Symfony* [69]. Estos estándares aseguran que el código sea consistente a lo largo del tiempo.

Para verificar el cumplimiento de los estándares mencionados, se ha utilizado la herramienta *PHP Code Sniffer*. Esta herramienta analiza el código fuente y verifica que no exista ningún incumplimiento de los estándares definidos.

Cobertura de Pruebas

Para la ejecución de la batería de pruebas automáticas, hemos utilizado *PHPUnit*. En total se han programado 36 pruebas, que realizan a su vez 109 comprobaciones diferentes. La batería de pruebas se ejecuta en el orden de 200 milisegundos en una instalación local.

PHPUnit permite generar un reporte detallado de la cobertura de código, indicando qué partes del código han sido probadas y cuáles no.

En la tabla 5.2, se muestra la cobertura de código obtenida para diferentes capas del sistema.

Capa	Cobertura
Dominio	72.41 %
Infraestructura y aplicación	42.86 %

Tabla 5.2: Evaluación de la cobertura de código

Como puede verse, la capa de dominio tiene una cobertura del 72.41% lo que se considera bastante buena. La capa de dominio contiene la lógica central de negocio del sistema y es crucial que esté bien probada para asegurar que el comportamiento del sistema sea correcto. Una cobertura alta no garantiza que tengamos una buena batería de pruebas, pero si es un buen indicador.

En cambio, una cobertura del 42.86% en la capa de infraestructura y aplicación se considera una cobertura moderada. Esta capa incluye componentes que interactúan con sistemas externos, bases de datos, API, y otras herramientas de la capa de infraestructura. Esta cobertura más modesta se debe a que en ocasiones

la capa de aplicación es más complicada de testear, además de que se dan situaciones que no merece la pena incluir en la batería de pruebas debido a su complejidad o rareza.

Análisis estático

El análisis estático es el proceso de examinar el código fuente de un programa sin ejecutarlo para identificar posibles errores, malas prácticas y problemas de seguridad.

Para este análisis, hemos utilizado la herramienta **Scrutinizer-Cl**. Esta herramienta analiza el código en busca de posibles problemas proporcionando un informe detallado así sugerencias de mejora. Finalmente, nos ha calificado con una puntuación de diez sobre diez lo que indica que la herramienta no ha podido detectar errores ni advertencias que deban ser revisadas [70].

Complejidad del código

Para evaluar la complejidad del código, hemos utilizado la herramienta **phploc** (*PHP Lines of Code*), la cual reporta varias métricas importantes. En la tabla 5.3 que aparece a continuación, se presentan algunos de los resultados obtenidos.

Nombre	Valor	Porcentaje
Total Non-Comment Lines of Code (NCLOC)	1649	80.64 %
Total Logical Lines of Code (LLOC)	313	15.31 %
Average Class Length (LLOC)	7	
Average Complexity per Class	2.68	
Average Complexity per Method	1.38	

Tabla 5.3: Evaluación de la complejidad del código

La métrica *Total Non-Comment Lines of Code (NCLOC)* indica el número total de líneas de código que no son comentarios.

La métrica *Total Logical Lines of Code (LLOC)* mide el número de líneas lógicas de código, excluyendo las líneas vacías y los comentarios. Las líneas lógicas representan instrucciones de programación, lo que da una idea más precisa de la cantidad de trabajo realizado por el código.

Average Class Length (LLOC) muestra la longitud promedio de las clases en términos de líneas lógicas de código. Una menor longitud promedio de clase suele ser preferible, ya que indica clases más manejables y con responsabilidades bien definidas.

Average Complexity per Class mide la complejidad promedio de las clases utilizando la complejidad ciclomática, que cuantifica el número de caminos independientes a través del código. Una menor complejidad por clase sugiere un diseño más simple y fácil de entender.

Finalmente, *Average Complexity per Method* similar a la métrica anterior, mide la complejidad ciclomática promedio por método. Métodos con menor complejidad son más fáciles de mantener y menos propensos a errores.

Estas métricas nos proporcionan una visión de un software bien estructurado y con baja complejidad.

Documentación

El proyecto cuenta con una amplia documentación recogida en este TFG. Esta documentación incluye tanto la descripción técnica del sistema como guías para desarrolladores y usuarios, lo que facilita la comprensión y el uso del sistema. La documentación cubre aspectos como la instalación, configuración, uso y mantenimiento del sistema, asegurando que cualquier persona interesada pueda comprender y trabajar con el sistema de manera efectiva.

Esta documentación completa y detallada es esencial para garantizar la calidad y mantenibilidad del sistema a largo plazo.

En resumen, las métricas analizadas y las herramientas utilizadas proporcionan una visión integral de un software bien estructurado, con baja complejidad y alta mantenibilidad. La adopción de estándares de la industria, como los sugeridos por Robert C. Martin en Código Limpio y Arquitectura Limpia, aseguran que el código sea robusto. La extensa documentación también contribuye a la comprensión y uso eficiente del sistema, garantizando su calidad y mantenibilidad a largo plazo.

5.4 Análisis de los resultados

Para terminar evaluaremos la precisión de los datos obtenidos. Se realizó una revisión manual de todos los resultados obtenidos, confirmando que en todos los casos fueron correctos.

En la tabla 5.4 puede verse que el sistema fue capaz de identificar y extraer correctamente el 100 % de los datos para todos los conjuntos.

Conjunto	Documentos	Correctos	Precisión
000	3	3	100 %
001	9	9	100 %
002	9	9	100 %

Tabla 5.4: Evaluación de los resultados por tipo de conjunto

Sin embargo, dado que los modelos LLM y en concreto el modelo *ChatGPT* usado es un sistema no determinista y que la muestra utilizada en esta evaluación es relativamente pequeña, es posible que la precisión real del sistema sea ligeramente inferior en escenarios más amplios y variados. La naturaleza no determinista de *ChatGPT* implica que los resultados pueden variar entre ejecuciones, lo que debe ser considerado al interpretar estos resultados.

6 Conclusiones

6.1 Recapitulación de los objetivos

Al inicio de este proyecto en la sección 1.4 Objetivos, se establecieron tres objetivos principales, cada uno orientado a abordar diferentes aspectos del proceso de extracción de información automatizada desde documentos.

A continuación, se presenta una recapitulación de cada uno de estos objetivos y su grado de cumplimiento:

El **Requisito 1** era desarrollar un sistema capaz de convertir documentos *PDF* en texto plano. Este objetivo se logró con éxito mediante el desarrollo del componente *Generator*. Este componente utiliza la herramienta *Pdf to Text* para realizar la conversión.

El **Requisito 2** fue extracción de información de contratos de arrendamiento de vivienda entre particulares. Este objetivo se cumplió mediante el desarrollo del componente *Reader* que utiliza la *API* de *ChatGPT* interpretar y extraer información de los documentos convertidos. Las pruebas realizadas determinaron que el sistema puede extraer con precisión el 100% de la información relevante en los contratos de arrendamiento.

Finalmente, el **Requisito 3** consistía extracción de información de contratos de compraventa de vehículo entre particulares. Al igual que con los contratos de arrendamiento, el componente *Reader* se encargó de este objetivo.

Finalmente, destacar que el diseño modular implementado permite incorporar fácilmente soporte para nuevos formatos de y tipos de documentos.

6.2 Síntesis de los resultados

Durante el desarrollo del proyecto se alcanzaron resultados relevantes que demuestran la relevancia del sistema desarrollado para la extracción de

información de documentos. A continuación, se muestra un resumen los principales resultados:

Desarrollo de un sistema de extracción de información

Se implementó un sistema dividido en dos componentes principales: *Generator* y *Reader*. El componente *Generator* convierte documentos *PDF* en texto plano, mientras que el componente *Generator* extrae información relevante del texto procesado.

Dentro de cada componente se diseñó un sistema de pre procesadores, procesadores y post procesadores que permiten añadir fácilmente soporte para nuevos tipos de formato o de documentos.

Rendimiento y Eficiencia

En este proyecto, el rendimiento y la eficiencia no son apartados trascendentales, siempre y cuando el procesamiento de documentos se mantenga en el rango de unos pocos segundos. El núcleo del sistema es rápido y eficiente, pero el rendimiento general depende en gran medida de las herramientas seleccionadas para la infraestructura.

1. *PDF To Text* que se ejecuta localmente, ofrece un buen rendimiento debido a su rapidez en el procesamiento de documentos *PDF*.
2. El rendimiento de la *API* de *ChatGPT* dependerá significativamente del tipo de documento y de la carga del sistema en un momento concreto.

En general, el rendimiento no será tan bueno para las herramientas de ejecución remota como para las herramientas de ejecución local.

Calidad del código y mantenibilidad

La calidad del código y su mantenibilidad en este proyecto han sido elevadas gracias a la adopción de estándares de la industria. Estos estándares aseguran que el código sea robusto, consistente y fácil de mantener a lo largo del tiempo. La buena cobertura de pruebas en la capa de dominio, con un 72.41%, refleja un enfoque riguroso en la validación de la lógica de negocio, mientras que la capa de infraestructura, con un 42.86%, aunque moderada, indica es un área a la que también se le ha prestado atención.

El análisis estático del código, realizado mediante Scrutinizer-CI, ha proporcionado una puntuación perfecta de diez sobre diez, destacando la adherencia a las mejores prácticas y la ausencia de errores críticos ni advertencias.

La documentación detallada del proyecto, incluido este TFG asegura su mantenibilidad a largo plazo.

Precisión en la extracción de datos

El sistema demostró una precisión del 100 % en la identificación y extracción de datos para todos los conjuntos de prueba. Si bien los conjuntos de prueba eran relativamente pequeños, este nivel de precisión resalta la robustez del sistema y su capacidad para manejar diversos tipos de documentos con alta exactitud.

Pruebas e integración continua

Se implementaron prácticas de integración continua utilizando *GitHub Actions*, lo que permitió automatizar la ejecución de pruebas. Esta integración asegura que cualquier cambio en el código sea rigurosamente probado y validado.

Adaptabilidad y extensibilidad

El sistema fue diseñado con una arquitectura limpia, lo que facilita la incorporación de nuevos tipos de documentos y métodos de procesamiento. Esta flexibilidad asegura que el sistema pueda adaptarse a diferentes necesidades, manteniendo su relevancia y utilidad.

6.3 Implicaciones y relevancia

El desarrollo y la implementación del sistema de extracción de información de documentos tiene varias implicaciones significativas y una relevancia considerable en múltiples áreas, tanto en el ámbito profesional como en el académico.

En el ámbito académico, este proyecto demuestra un nuevo enfoque de los muchos posibles en los que se puede sacar partido a los sistemas de modelos de lenguaje (LLM). La capacidad de estos sistemas para procesar y extraer información relevante de documentos complejos abre nuevas oportunidades

para la investigación y la enseñanza en campos como la inteligencia artificial, la minería de datos y la gestión de información.

En el ámbito profesional, este tipo de tecnologías muestra un gran potencial para transformar la forma en que las compañías y organizaciones manejan la documentación. La automatización de la extracción de datos no solo puede aumentar la eficiencia y reducir los costos operativos, sino también minimizar los errores humanos y mejorar la consistencia en el manejo de información. En los próximos años, se espera que estas tecnologías tengan un impacto significativo en industrias como la legal, la financiera, la médica y entre otras, facilitando la tramitación de documentos y mejorando los procesos administrativos.

6.4 Reflexión sobre el proceso de desarrollo

El proceso de desarrollo de este proyecto ha sido una experiencia enriquecedora y formativa, proporcionando valiosas lecciones en diversas áreas de la ingeniería del software y la gestión de proyectos. A continuación, se presenta una reflexión sobre los aspectos más destacados y los desafíos enfrentados durante el desarrollo de este TFG.

1. La planificación y ejecución de un proceso de desarrollo utilizando Metodologías ágiles requirieron una constante evaluación y ajuste, lo que implicó un esfuerzo adicional. Sin embargo, este enfoque resultó en una mayor flexibilidad y capacidad de respuesta a las necesidades emergentes del proyecto.
2. La implementación de una arquitectura limpia aseguró que el núcleo del sistema no estuviera acoplado a ninguna herramienta externa, permitiendo intercambiar unas por otras y generando un sistema altamente modular y extensible.
3. La implementación del desarrollo dirigido por pruebas aseguró que cada componente del sistema fuera rigurosamente evaluado desde el principio. Esto no solo mejoró la calidad del código, sino que también facilitó la identificación y corrección temprana de errores, tanto de código como de diseño.
4. La Integración de tecnologías de última generación como modelos de lenguaje de gran escala (LLM), en este caso a través de la API de *ChatGPT* para la evaluación de documentos, *Docker* para la contenerización, *ELK*

para el registro de logs entre otras, permitió desarrollar un sistema robusto y novedoso. Sin embargo, la curva de aprendizaje asociada con la adopción de nuevas tecnologías y herramientas fue significativa. La configuración de estas herramientas para que funcionaran correctamente requirió un tiempo y esfuerzo considerable.

En conclusión, el proceso de desarrollo de este proyecto ha sido un recorrido desafiante pero gratificante, lleno de aprendizajes y crecimiento.

6.5 Trabajo futuro

A medida que la tecnología y las necesidades del mercado evolucionan, siempre existen oportunidades para mejorar y extender el trabajo realizado.

En este contexto, se presentan algunas sugerencias y recomendaciones para futuros trabajos en el ámbito de este proyecto.

Mejorar el conjunto de datos de prueba

Si bien el conjunto de datos de prueba es adecuado para el alcance de este TFG, es necesario ampliarlo para obtener conclusiones más relevantes sobre la precisión del sistema. Utilizar un conjunto de datos más grande y diverso permitirá evaluar mejor la robustez del sistema y su capacidad para manejar diferentes tipos de documentos y situaciones. Esto ayudará a asegurar que el sistema sea confiable y efectivo.

Otra mejora en este aspecto es designar un equipo externo que se ocupe de la evaluación de las pruebas para evitar que se produzca el sesgo del creador, un sesgo que se produce cuando la misma persona que desarrolla el código también lo prueba, lo que puede resultar en la falta de detección de errores [71].

Añadir soporte a otros formatos de documentos

Aunque una gran parte de la documentación gestionada por las organizaciones está en formato *PDF*, es conveniente incrementar el soporte para otros formatos igualmente comunes.

Por ejemplo se podría utilizar *LibreOffice* [72] para convertir documentos en formato *word* [3] o en formato *excel* [4] a texto.

Además, se podría emplear *Tesseract OCR* [58], una herramienta que permite convertir imágenes en texto, utilizando una tecnología de reconocimiento óptico de caracteres u *optical character recognition* (OCR) para convertir documentos escaneados o fotografías a texto.

Utilizar otros modelos

La rápida evolución de los modelos de lenguaje natural ofrece continuas oportunidades para mejorar la precisión, los tiempos de respuesta y el costo por documento procesado. Aunque en este trabajo se ha utilizado *ChatGPT 4*, es importante seguir probando nuevos modelos a medida que se desarrollan.

Evaluar estos modelos en términos de precisión, eficiencia y costo ayudará a identificar cuál es el más adecuado para cada caso de uso, asegurando que el sistema se mantenga a la vanguardia de la tecnología y sea capaz de ofrecer los mejores resultados posibles.

Paralelización de procesos

En el sistema desarrollado, la coordinación entre los diferentes procesadores se realiza de forma secuencial en ambos componentes *Generator* y *Reader*. Este enfoque es adecuado siempre y cuando el número de procesadores se mantenga muy limitado.

Sin embargo, si se añaden nuevos procesadores y el tiempo de ejecución se vuelve importante, será necesario cambiar de un esquema de llamadas consecutivas a uno donde todas las llamadas se ejecuten en paralelo. Esto permitirá una mayor eficiencia y una reducción significativa en el tiempo total de procesamiento, mejorando así el rendimiento del sistema en escenarios de alta demanda.

Añadir otras técnicas para extraer la información

Aunque los modelos LLM ofrecen una solución óptima para documentos con formato desconocido, para otros documentos el formato será conocido y tendrá pocas variaciones, como por ejemplo una declaración de la renta, facturas comerciales o los recibos de servicios públicos.

En estos casos, es más eficiente y económico extraer la información mediante técnicas más rudimentarias, como por ejemplo las expresiones regulares. Estas

técnicas permiten identificar y extraer datos específicos de manera rápida y con un menor costo, aprovechando la estructura predecible de estos documentos.

Incrementar la seguridad de los datos

La privacidad y la seguridad de los datos son aspectos críticos en cualquier sistema de procesamiento de información, especialmente cuando se manejan documentos sensibles.

La incorporación de prácticas de seguridad desde el diseño, conocido como *privacy by design*, puede ayudar a construir un sistema más seguro en este sentido. Para ello, se podrían implementar políticas de acceso, cifrado de datos y anonimización de los mismos cuando sea adecuado.

Estas medidas no solo protegen la información personal durante el procesamiento y almacenamiento, sino que también aseguran el cumplimiento de normativas de protección de datos.

Liberar el proyecto

Publicar el proyecto como código abierto permitirá que la comunidad de desarrolladores contribuya al mejoramiento y extensión del sistema. Esto puede incluir la identificación y corrección de errores y el soporte para nuevos tipos de formatos o documentos, entre otros.

Para ello se propone crear una documentación detallada sobre cómo contribuir al proyecto, incluyendo guías de estilo de código, procesos de revisión y pautas para la presentación de issues y pull requests.

Además, será necesario seleccionar una licencia adecuada para el proyecto que defina claramente los derechos y responsabilidades de los usuarios y contribuyentes. Licencias como MIT, Apache 2.0 o GNU GPL son opciones comunes que pueden ser consideradas.

Poner en producción

Desplegar el sistema en un entorno de producción es un paso vital que pone a prueba la capacidad del proyecto para operar bajo condiciones reales y cumplir con las expectativas. Esta prueba de fuego permite identificar y resolver

problemas que pueden no haber sido evidentes en un entorno de desarrollo o pruebas en entornos controlados.

Este paso final no solo valida la funcionalidad y la eficiencia del sistema, sino que también proporciona oportunidades para optimizar, asegurar y mejorar continuamente el sistema.

Bibliografía

- [1] Daniel González. *VIU 47 Proyecto ingeniería del software*. URL: https://github.com/desarrolla2/viu_47_proyecto_de_ingenieria_del_software.
- [2] Adobe Systems Incorporated. *Portable Document Format (PDF)*. URL: <https://www.adobe.com/pdf>.
- [3] Microsoft Corporation. *Microsoft Word*. URL: <https://www.microsoft.com/word>.
- [4] Microsoft Corporation. *Microsoft Excel*. URL: <https://www.microsoft.com/excel>.
- [5] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008. ISBN: 978-0132350884.
- [6] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson Education, 2017. ISBN: 978-0134494166.
- [7] A. Turing. «Computing Machinery and Intelligence». En: *Mind* 59.236 (1950), págs. 433-460. DOI: 10.1093/mind/LIX.236.433.
- [8] Theodore H. Smith y Bernard L. Merrick. *The Georgetown-IBM Experiment*. Inf. téc. Georgetown University, 1964.
- [9] Christopher D. Manning e Hinrich Schütze. «Foundations of Statistical Natural Language Processing». En: (1999).
- [10] Jacob Devlin, Ming-Wei Chang y et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». En: *arXiv preprint arXiv:1810.04805* (2019).
- [11] Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.
- [12] Sepp Hochreiter y Jürgen Schmidhuber. «Long Short-Term Memory». En: *Neural Computation* 9.8 (1997), págs. 1735-1780.
- [13] Ashish Vaswani et al. «Attention is All You Need». En: *arXiv preprint arXiv:1706.03762* (2017).
- [14] Tom B. Brown, Benjamin Mann y et al. «Language Models are Few-Shot Learners». En: *arXiv preprint arXiv:2005.14165* (2020).
- [15] Thomas Wolf et al. «Transformers: State-of-the-Art Natural Language Processing». En: *arXiv preprint arXiv:1910.03771* (2020).

- [16] Bob sharp. *ChatGPT vs. HuggingChat: Which Is Better?* 2024. URL: <https://www.makeuseof.com/chatgpt-vs-huggingchat/>.
- [17] EleutherAI. *EleutherAI GPT-Neo and GPT-J*. URL: <https://www.eleuther.ai/>.
- [18] Facebook AI Research. *Fairseq*: Facebook AI Research Sequence-to-Sequence Toolkit. URL: <https://github.com/pytorch/fairseq>.
- [19] OpenAI. *OpenAI GPT-2*. URL: <https://github.com/openai/gpt-2>.
- [20] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999. ISBN: 978-0201616415.
- [21] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, 2003. ISBN: 978-0321146533.
- [22] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010. ISBN: 978-0984521401.
- [23] Daniel González. VIU 47, Proyecto Ingeniería del Software, Sprint 6, Planificación. URL: <https://www.youtube.com/watch?v=AcML0-qcm5Q>.
- [24] Daniel González. VIU 47, Proyecto Ingeniería del Software, Sprint 6, Demostración. URL: <https://www.youtube.com/watch?v=f-00CNTTzwk>.
- [25] Daniel González. VIU 47, Proyecto Ingeniería del Software, Sprint 6, Retrospectiva. URL: <https://www.youtube.com/watch?v=Eg8BUbnQtJs>.
- [26] Daniel González. Proyecto de ingeniería del software. URL: https://github.com/desarrolla2/viu_47_proyecto_de_ingenieria_del_software.
- [27] Ollama. *Ollama Documentation*. URL: <https://ollama.com/documentation>.
- [28] OpenAI. *ChatGPT-3.5-turbo*. URL: <https://platform.openai.com/docs/models/gpt-3.5-turbo>.
- [29] Ollama. *Ollama3 Library*. URL: <https://ollama.com/library/llama3>.
- [30] OpenAI. *ChatGPT-4*. URL: <https://platform.openai.com/docs/models/gpt-4>.
- [31] Google. *Google Docs*. URL: <https://www.google.com/docs/about/>.
- [32] LaTeX Project. *LaTeX Project*. URL: <https://www.latex-project.org/>.
- [33] PHP Group. *PHP: Hypertext Preprocessor*. URL: <https://www.php.net/>.
- [34] Symfony. *Symfony - The leading PHP framework to create web applications*. URL: <https://symfony.com/>.
- [35] Twig. *Twig - The flexible, fast, and secure template engine for PHP*. URL: <https://twig.symfony.com/>.
- [36] W3C. *HTML5 Specification*. URL: <https://www.w3.org/TR/html5/>.
- [37] W3C. *CSS3 Specification*. URL: <https://www.w3.org/Style/CSS/>.

- [38] ECMA International. *ECMAScript 2021 Language Specification*. URL: <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>.
- [39] Mark Otto y Jacob Thornton. *Bootstrap Documentation*. URL: <https://getbootstrap.com/>.
- [40] Inc. Fonticons. *Font Awesome Documentation*. URL: <https://fontawesome.com/>.
- [41] Nils Adermann y Jordi Boggiano. *Composer Documentation*. URL: <https://getcomposer.org/doc/>.
- [42] Linus Torvalds y Junio Hamano. *Git Documentation*. URL: <https://git-scm.com/doc>.
- [43] Inc. GitHub. *GitHub Documentation*. URL: <https://docs.github.com/>.
- [44] Daniel González. *VIU 84 Trabajo de final de grado, proyecto*. URL: https://github.com/desarrolla2/viu_84_proyecto.
- [45] Free Software Foundation. *GNU Make Manual*. URL: <https://www.gnu.org/software/make/manual/>.
- [46] Sebastian Bergmann. *PHPUnit Documentation*. URL: <https://phpunit.de/manual/current/en/>.
- [47] Inc. GitHub. *GitHub Actions Documentation*. URL: <https://docs.github.com/en/actions>.
- [48] Inc. Docker. *Docker Documentation*. URL: <https://docs.docker.com/>.
- [49] Inc. Docker. *Docker Compose Documentation*. URL: <https://docs.docker.com/compose/>.
- [50] Jordi Boggiano. *Monolog Documentation*. URL: <https://github.com/Seldaek/monolog>.
- [51] Elastic NV. *Elasticsearch Documentation*. URL: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>.
- [52] Elastic NV. *Logstash Documentation*. URL: <https://www.elastic.co/guide/en/logstash/current/index.html>.
- [53] Elastic NV. *Kibana Documentation*. URL: <https://www.elastic.co/guide/en/kibana/current/index.html>.
- [54] XpdfReader. *PDF to Text Documentation*. URL: <https://www.xpdfreader.com/pdf2text-man.html>.
- [55] OpenAI. *ChatGPT API Documentation*. URL: <https://platform.openai.com/docs/api-reference/chat>.

- [56] Symfony. *Symfony HttpClient Documentation*. URL: https://symfony.com/doc/current/http_client.html.
- [57] Symfony. *Symfony Cache Documentation*. URL: <https://symfony.com/doc/current/components/cache.html>.
- [58] Ray Smith y Tesseract Contributors. *Tesseract OCR*. URL: <https://github.com/tesseract-ocr/tesseract>.
- [59] T. B. Brown et al. «Language Models are Few-Shot Learners». En: *arXiv preprint arXiv:2005.14165* (2020). URL: <https://arxiv.org/abs/2005.14165>.
- [60] L. Reynolds y K. McDonell. «Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm». En: *arXiv preprint arXiv:2102.07350* (2021). URL: <https://arxiv.org/abs/2102.07350>.
- [61] OpenAI. *ChatGPT API Pricing*. URL: <https://openai.com/api/pricing/>.
- [62] OpenAI. *Models*. URL: <https://platform.openai.com/docs/models/>.
- [63] Data.gov. *United States Open Data Portal*. 2024. URL: <https://www.data.gov/>.
- [64] European Union. *European Union Open Data Portal*. URL: <https://data.europa.eu/euodp/en/home>.
- [65] Kaggle Datasets. *Kaggle Datasets*. URL: <https://www.kaggle.com/datasets>.
- [66] Google. *Google Dataset Search*. Motor de búsqueda de conjuntos de datos. 2024. URL: <https://datasetsearch.research.google.com/>.
- [67] Amazon Web Services. *AWS Public Datasets*. 2024. URL: <https://registry.opendata.aws/>.
- [68] Daniel González. *VIU 84, Trabajo de final de grado, Integración continua*. URL: https://github.com/desarrolla2/viu_84_proyecto/actions/workflows/docker-tests.yml.
- [69] Symfony Project. *Symfony Code Standards*. URL: <https://symfony.com/doc/7.0/contributing/code/standards.html>.
- [70] Desarrolla2. *VIU 84 Proyecto*. URL: https://scrutinizer-ci.com/g/desarrolla2/viu_84_proyecto/.
- [71] Test IO. *How to Combat Bias in Software Testing*. URL: <https://test.io/resources/blog/how-to-combat-bias-in-software-testing>.
- [72] The Document Foundation. *LibreOffice*. URL: <https://www.libreoffice.org>.
- [73] Daniel González. *Trabajo de fin de grado, instrucciones de instalación*. URL: https://github.com/desarrolla2/viu_84_proyecto/blob/main/INSTALL.md.
- [74] OpenAI. *OpenAI Platform*. URL: <https://platform.openai.com/>.

A Instrucciones de instalación

Puedes encontrar una versión actualizada de este documento en la documentación del proyecto en *github* [73].

Instalar dependencias del proyecto

Para la ejecución de este proyecto, es necesario tener instalados los paquetes que aparecen indicados en la tabla A.1 junto con sus versiones mínimas.

Paquete	Versión mínima
docker [48]	20.10
docker-compose [49]	1.25
git [42]	2.34

Tabla A.1: Dependencias del proyecto y versiones mínimas

Proceso de instalación

Descargar el código fuente

Descargamos el código fuente desde el repositorio principal del proyecto.

```
git clone git@github.com:desarrolla2/viu_84_trabajo_fin_grado
```

Configurar el entorno docker

En este paso será necesario copiar archivos de configuración de docker, para ello ejecutamos lo siguiente:

```
cp docker/.env.dist .env
cp docker/docker-compose.yml.dist docker-compose.yml
cp docker/Makefile.dist Makefile
```

Si también se desea contar con los contenedores ELK descritos en la subsección 4.4 Registros, para la visualización de los logs deberás ejecutar:

```
cp docker/docker-compose.override.yml.dist docker-compose.override.yml
```

Finalmente, puedes reescribir la configuración si fuera necesario para adecuarla a tus necesidades.

Construir contenedores e instalar dependencias

Construir contenedores

Para la construcción de los contenedores hay una tarea definida en el fichero *MakeFile*

```
make docker-build
```

Instalar dependencias PHP

También hay una tarea definida para la instalación de las dependencias.

```
make php-composer-install
```

Levantar los contenedores

La siguiente tarea levanta los contenedores necesarios para la ejecución del proyecto.

```
make docker-up
```

Editar archivo .env

Finalmente, edita el archivo *.env* para actualizar los siguientes valores:

- **APP_SECRET**: establece un token único.
- **OPENAI_AUTHENTICATION_TOKEN**: genera un token válido a través de la plataforma de OpenAI [74]

Ejecutar pruebas

Una vez el proyecto está correctamente instalado, puedes ejecutar la batería de pruebas a través de la siguiente tarea.



```
make tests
```