Trabajo de Fin de Grado

Web Service para la transcripción de facturas y recibos

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Antonio Javier Torres Bordón

TUTORIZADO POR:
José Juan Hernández Cabrera

Fecha 07/24

Agradecimientos

A mi familia, a la escuela y a los profesores que han sido clave en mi formación y experiencia académica.

Resumen

Este trabajo de fin de grado consiste en la implementación de un Web Service para el reconocimiento de datos de facturas, abarcando tanto la investigación del problema que abordamos (*NER: Named Entity Recognition*), como desarrollar y hacer fine tuning de un modelo de procesamiento de lenguaje natural (BERT) para aplicarlo a problemas de reconocimiento de entidades nombradas, como generar un dataset y como abordar el problema paso a paso, desde la extracción de texto útil de las facturas hasta su interpretación, además de la disponibilidad que ofrece su implementación como Web Service y la facilidad de consumir este Software e implementarlo en entornos escalables y productivos.

Abstract

This final degree work consists of the implementation of a Web Service for invoice data recognition, covering both the investigation of the problem we address (*NER: Named Entity Recognition*), how to develop and fine tune a natural language processing model (BERT) to apply it to named entity recognition problems, how to generate a dataset and how to approach the problem step by step, from the extraction of useful text from invoices to its interpretation, as well as the availability offered by its implementation as a Web Service and the ease of consuming this Software and implementing it in scalable and productive environments.

Índice general

Índice de figuras

Índice de Algoritmos

Capítulo 1

Introducción, competencias específicas y objetivos

En este capítulo se presenta la introducción al Trabajo de Fin de Grado, destacando las competencias específicas que se espera desarrollar y los objetivos que se pretenden alcanzar. Se ofrecerá una visión general del contenido del proyecto, proporcionando un marco conceptual que facilite la comprensión de los temas a tratar a lo largo del trabajo.

1.1. Introducción

En el mundo actual, las empresas gestionan un volumen masivo de información y documentos que deben ser procesados de manera eficiente. Entre estos documentos, las facturas representan un componente esencial del ciclo financiero y administrativo. La automatización de la extracción de información relevante de las facturas puede mejorar significativamente la eficiencia y productividad, reducir la repetición de tareas mecánicas y optimizar recursos. Es en este contexto donde se enmarca el presente Trabajo de Fin de Título, que se centra en el desarrollo y la implementación de un sistema de Reconocimiento de Entidades Nombradas (NER, por sus siglas en inglés) para la extracción automática de datos en facturas, abarcando desde el desarrollo de un dataset, el fine tuning de un Transformer *(BERT)*, hasta su implementación en un servicio con una interfaz de usuario, su API y además su contenerización que permite su sencilla implementación.

El Reconocimiento de Entidades Nombradas es una técnica fundamental en el procesamiento del lenguaje natural (NLP), que permite identificar y clasificar elementos clave en un texto, como nombres de empresas, fechas, costes, etc. Aplicar NER a la digitalización de facturas no solo facilita la identificación automática de estos elementos, sino que también permite su integración en diversos sistemas de gestión de datos, haciendo que esta tecnología sea accesible y económica tanto para grandes empresas como para individuos y pequeñas empresas.

La importancia de la extracción automatizada de datos de facturas radica en su capacidad para proporcionar una solución económica y accesible para la recopilación de datos. Esto es especialmente beneficioso para pequeños negocios e individuos que no pueden permitirse costosos sistemas de gestión de documentos. Un sistema automatizado de extracción de datos ofrece la posibilidad de integrar fácilmente la información extraída en diversos sistemas y aplicaciones, mejorando la accesibilidad y usabilidad de los datos financieros.

La realización de este proyecto responde a la necesidad creciente de modernizar los procesos y adoptar tecnologías avanzadas para mantenerse competitivos en un mercado cada vez más digitalizado. El objetivo principal es desarrollar un sistema capaz de leer, interpretar y extraer de manera precisa los datos relevantes de las facturas, utilizando técnicas de NER y OCR y detección de texto, que bautizaremos con el nombre **AVALON** [**13**]. Este sistema busca minimizar la intervención manual, reducir el tiempo de procesamiento y garantizar una mayor precisión en la gestión de documentos financieros.

En esta introducción, se detallarán los aspectos clave que motivan la implementación de este proyecto, los beneficios esperados para empresas y particulares, y la relevancia del uso de tecnologías de procesamiento del lenguaje natural en el ámbito de la automatización de procesos administrativos. Asimismo, se proporcionará una visión general de la estructura del trabajo, destacando las metodologías empleadas y los resultados esperados, sentando un precedente para la implementación propia de soluciones a problemas similares, como por ejemplo el reconocimiento de datos en identificación.

1.2. Competencias específicas

CI1 Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente

El sistema desarrollado garantiza fiabilidad, seguridad y calidad usando técnicas avanzadas de NER y BERT. La seguridad se refuerza al mantener los datos dentro de la organización, evitando riesgos asociados a soluciones en la nube y cumpliendo con la normativa vigente y principios éticos.

CI2 Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social

El proyecto, desde la planificación hasta el despliegue, ofrece una interfaz de usuario y una API robusta. Su impacto económico y social es positivo, beneficiando a pequeñas empresas con una solución accesible y económica. La mejora continua está asegurada con un enfoque iterativo.

Capítulo 2

Estado actual y objetivos iniciales

En este capítulo, trataremos el estado del arte de las herramientas de extracción de datos de facturas, explorando las tecnologías más avanzadas y las metodologías empleadas en este campo. Se revisarán las principales herramientas disponibles en el mercado, sus características, ventajas y limitaciones, así como los enfoques innovadores que están marcando tendencia en la automatización de la gestión de facturas. Además, se presentarán los objetivos iniciales originalmente estipulados en el Trabajo de Fin de Titulación (TFT01), proporcionando un marco de referencia claro y actualizado que fundamentará las decisiones y desarrollos posteriores en el TFT.

2.1. Estado del arte

Para el tratamiento del problema de extracción de entidades nombradas o NER, y específicamente en el sector de la extracción de datos en facturas, existen varios servicios y herramientas de pago que permiten realizar la tarea de extracción de información de manera precisa, pero sin embargo, ninguna herramienta popular Open-Source que permita la resolución de manera sencilla e integral de este problema. Analizaremos las distintas herramientas de pago y código cerrado, sin capacidad de observar cual es la arquitectura de sistema que implementan y como tratan el problema, pero aportando un marco sobre el que podemos deducir como han sido desarrollada las mismas.

Azure Document Intelligence

Azure Document Intelligence, específicamente su función de ***Invoice Data Extraction***, es una herramienta avanzada que utiliza Reconocimiento Óptico de Caracteres (OCR) para identificar y extraer información clave de facturas y documentos relacionados. Este servicio soporta diversos idiomas y puede manejar una amplia variedad de formatos de documentos, desde facturas tradicionales hasta órdenes de compra y facturas de servicios públicos.

Proporciona salidas estructuradas en JSON, integrándose fácilmente con sistemas de gestión. Automatiza la extracción de datos, mejora la precisión y reduce el esfuerzo manual. Su integración se hace a través de la interacción con una API, en la que envías el documento, se procesa y obtienes los datos en formato JSON. Las ventajas de este sistema es su fácil integración gracias a la nula necesidad del desarrollador de requerir de conocimientos sobre inteligencia artificial. Las desventajas de este sistema son principalmente la incapacidad de tratar los datos de forma local y el precio por factura procesada a pagar.[8]

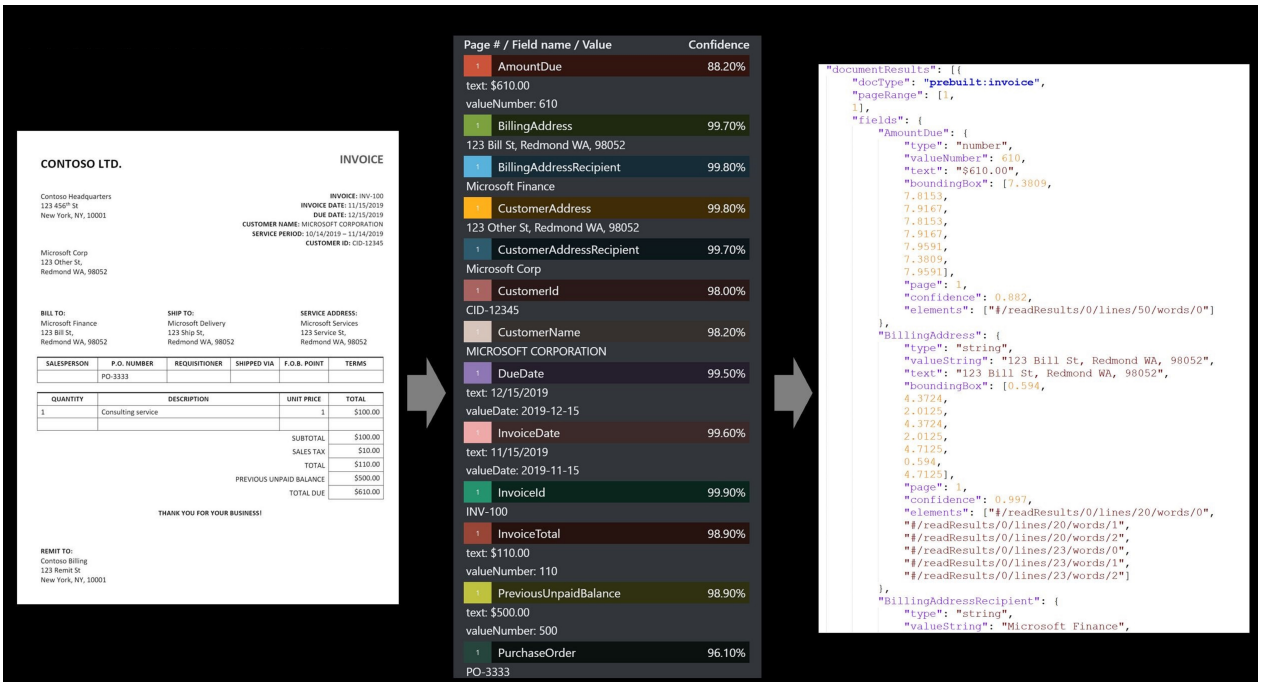


Ilustración 2.1: Explicación de funcionamiento del servicio proporcionada por Azure [8]

Google Cloud Document AI

Google Cloud Document AI es una herramienta avanzada que emplea la inteligencia artificial y el aprendizaje automático para analizar y extraer datos de documentos. Su función ***Invoice Parser*** permite la extracción automatizada de información clave de facturas y otros documentos financieros, es decir, es el homónimo de Google de Azure Document Intelligence y funciona de la misma manera.

Su implementación es exactamente igual a la que Azure ofrece, se interactúa a traves de una API en el que envías tus datos a la nube y estos son procesados devolviendo los datos estructurados en formato JSON, probablemente con una estructura ligeramente distinta. Una de las diferencias de implementación con Azure es la capacidad de aprendizaje continuo, donde en la solución de Microsoft se debe pagar por este mismo entrenamiento de forma alternativa.[4]

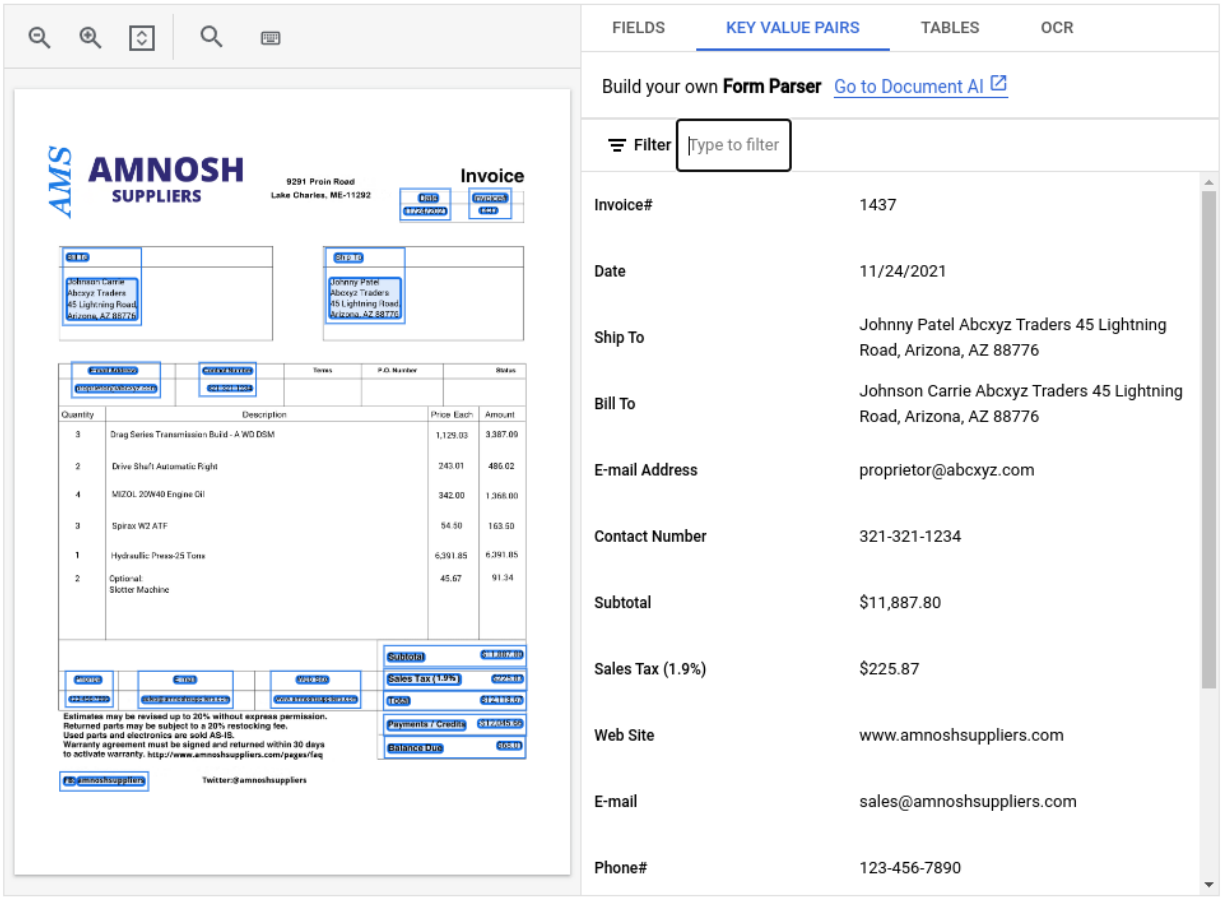


Ilustración 2.2: Herramienta de demostración ofrecida por Google[4]

Otros servicios similares

Existen múltiples soluciones que se implementan de manera similar, conectando con una API, enviando y recibiendo datos. Entre estas soluciones destacan:

- ***Amazon Textract*** [12]
- ***ABBYY FlexiCapture*** [1]
- ***Rossum*** [11]
- ***Kofax*** [6]
- ***Hypatos*** [5]
- ***Parascript*** [10]
- ***Docparser*** [2]
- ***Veryfi*** [16]
- ***Nanonets*** [9]
- ***UiPath Document Understanding*** [14]
- ***WorkFusion*** [17]

Estas herramientas comparten características comunes como el uso de OCR y machine learning para la extracción de datos, la provisión de datos estructurados en formato JSON y la integración a través de APIs.

2.2. Técnicas de Implementación

Las técnicas para extraer información de documentos semiestructurados han evolucionado significativamente, pasando por dos etapas principales: la extracción de información tradicional y la extracción basada en aprendizaje profundo. Los métodos tradicionales se apoyan en características de imagen para identificar caracteres en las imágenes y extraer información, utilizando principalmente reglas y la coincidencia de plantillas. Sin embargo, estos métodos tienen limitaciones en su capacidad para generalizar y son afectados por la complejidad y variabilidad de los documentos [7].

En contraste, los métodos basados en aprendizaje profundo permiten una identificación y extracción más precisa y robusta de características complejas en documentos semiestructurados, superando significativamente a los métodos tradicionales. Estos métodos integran información de características semánticas, espaciales y visuales, mejorando así el rendimiento de la extracción [7].

Arquitectura

La arquitectura técnica de la extracción de información basada en aprendizaje profundo involucra la detección y reconocimiento de texto, así como la extracción de información de texto mediante la obtención de características multimodales ricas. Estas características incluyen información semántica, espacial y visual. La extracción de información se puede clasificar desde varias perspectivas: pasos operativos (dos etapas o extremo a extremo), serialización del texto (serializado o sin serializar), análisis del diseño del texto (análisis unimodal o multimodal) y representación del documento (basado en secuencias, estructuras gráficas o cuadrículas) [7].

Tecnologías Clave

La extracción de información implica la aplicación de algoritmos de procesamiento de lenguaje natural (NLP) y visión por computadora (CV). Los avances en redes neuronales como CNN, RNN y GNN, junto con mecanismos de atención y modelos Transformer, han facilitado la evolución de la tecnología de extracción de información. Estas tecnologías han permitido mejoras significativas en la precisión y eficiencia de la extracción de información de documentos semiestructurados [7].

Detección e Identificación

La capa de detección y reconocimiento tiene como objetivo extraer, convertir y fusionar información de características críticas de una variedad de datos multimodales y de múltiples fuentes. Esta capa incluye módulos de representación de características modales, conversión de características modales y fusión de características modales [7].

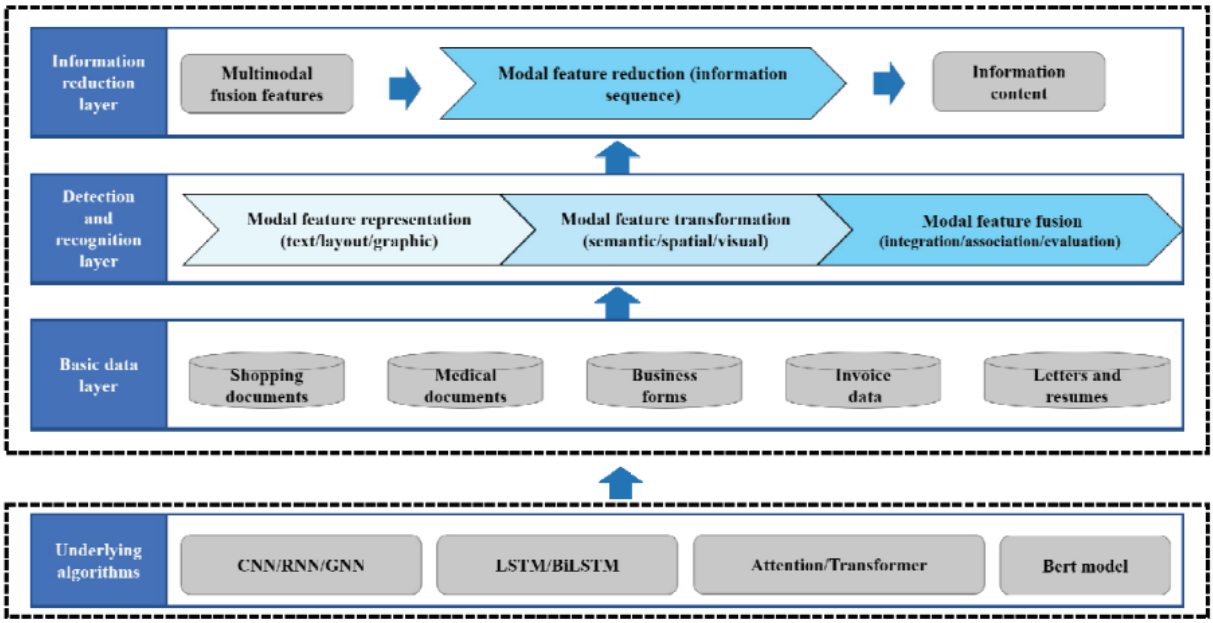


Ilustración 2.3: Arquitectura de solución de inteligencia artificial[7]

En resumen, la transición del uso de métodos tradicionales a técnicas basadas en aprendizaje profundo ha marcado una mejora notable en la precisión y capacidad de generalización en la extracción de información de documentos semiestructurados. La integración de características multimodales y el uso de arquitecturas avanzadas han sido claves en este progreso [7].

2.3. Objetivos iniciales

Objetivo Principal

El objetivo principal de este proyecto es desarrollar un modelo de inteligencia artificial que pueda transcribir facturas y recibos con una precisión que sea no solo aceptable, sino preferiblemente excelente. Esto permitirá reducir al mínimo la supervisión humana necesaria, mejorando así la eficiencia y efectividad del proceso.

Objetivos Adicionales

Además del objetivo principal, este proyecto se propone empaquetar la herramienta de manera que su ejecución sea lo más sencilla posible para los usuarios. Para lograrlo, se proporcionarán ejecutables accesibles en un repositorio de GitHub, lo que facilitará su uso y distribución.

Otro objetivo importante es la optimización del modelo para reducir los costos computacionales excesivos que suelen asociarse con las inteligencias artificiales actuales. Al reducir estos costos, también se disminuirá el gasto en infraestructura necesario para ejecutar la herramienta, haciendo que su implementación sea más económica y accesible.

Objetivos Específicos

Para alcanzar estos objetivos, se profundizará en el conocimiento de las redes neuronales y su aplicabilidad en el contexto de este proyecto. Esto incluirá el estudio de modelos de lenguaje que puedan ser aplicables para la interpretación del contexto de los datos transcritos.

Asimismo, se experimentará con técnicas avanzadas de visión por computadora y reconocimiento óptico de caracteres (OCR) basadas en redes neuronales. Estas técnicas se aplicarán al desarrollo de la herramienta, mejorando su capacidad para interpretar y transcribir con precisión los datos contenidos en facturas y recibos.

Finalmente, se designará el nombre AVALON [13] al sistema que se desarrollará, el cual se espera que cumpla con todos estos objetivos y se convierta en una referencia en el ámbito de la transcripción automatizada de facturas y recibos.

Capítulo 3

Metodología, herramientas

3.1. Metodología

La metodología empleada en este proyecto ha sido diseñada para abordar de manera eficiente y flexible los desafíos inherentes al desarrollo de un sistema de extracción de datos de facturas basado en inteligencia artificial. Este enfoque combina elementos de diversas prácticas ágiles, adaptadas a las necesidades específicas de un proyecto desarrollado por un solo individuo, con el objetivo de maximizar la productividad y la calidad del producto final.

3.1.1. Enfoque Ágil e Iterativo

El proyecto se ha desarrollado siguiendo una metodología ágil e iterativa, lo que ha permitido una evolución gradual desde la conceptualización hasta la implementación de un producto final. Este enfoque se materializó en cuatro iteraciones principales, cada una con una duración aproximada de un mes a mes y medio.

La primera iteración se centró en la investigación, explorando tecnologías y conceptos relevantes para el proyecto. Esta fase fue crucial para establecer una base sólida de conocimientos y definir la dirección del desarrollo. La segunda iteración se dedicó al desarrollo de un prototipo, creando una versión preliminar que permitió validar los conceptos clave identificados en la fase de investigación.

En la tercera iteración, el foco se desplazó hacia el desarrollo de un producto mínimo viable, implementando la cadena completa de procesamiento de datos de facturas. Esta fase fue fundamental para probar la viabilidad del sistema en condiciones más cercanas a la realidad. Finalmente, la cuarta iteración se centró en la contenerización y la preparación del producto final, refinando el sistema y asegurando su preparación para el despliegue.

3.1.2. Modularidad y Desarrollo Incremental

La modularidad fue un principio fundamental que guió el desarrollo del proyecto. Este enfoque permitió abordar componentes específicos de manera independiente, facilitando un desarrollo incremental y permitiendo pruebas y mejoras continuas en cada módulo.

Entre los componentes clave desarrollados se encuentran: la extracción y detección de texto en facturas, la clasificación del texto extraído, la conversión de clasificaciones de IA a formato JSON, el desarrollo de una librería integradora de la cadena de procesamiento, y la implementación de un servidor con API para gestionar peticiones. Cada uno de estos módulos se desarrolló de forma iterativa, permitiendo una evolución constante del sistema en su conjunto.

3.1.3. Gestión del Proyecto

Para la gestión del proyecto, se implementó un sistema Kanban tradicional utilizando post-its físicos. Esta herramienta visual resultó ser extremadamente útil para mantener una visión clara del flujo de trabajo y facilitar la priorización de tareas, especialmente en un contexto de desarrollo individual donde la autogestión es crucial.

Las historias gestionadas a través de este sistema Kanban se centraron en dos áreas principales: el desarrollo del proyecto en sí, que incluía tareas como la implementación de la generación de datasets, y la interacción con el sistema, que abarcaba el desarrollo de la API y la interfaz de usuario con diversas capacidades.

3.1.4. Priorización y Gestión de Cambios

La estrategia de priorización se enfocó en tres aspectos fundamentales: en primer lugar, el desarrollo de la solución de inteligencia artificial core; en segundo lugar, la implementación de herramientas periféricas para facilitar la integración, como el traductor a JSON; y finalmente, el desarrollo de un servicio basado en Flask para gestionar peticiones.

Este enfoque permitió una adaptación flexible a los cambios en los requisitos, manteniendo siempre el foco en la entrega de valor incremental. La capacidad de ajustar prioridades y responder a nuevos desafíos fue crucial para el éxito del proyecto, especialmente dado su carácter innovador y experimental.

3.1.5. Adaptación de Prácticas Ágiles

Aunque el proyecto fue desarrollado por una sola persona, lo que naturalmente limitó la aplicación de ciertas prácticas de Scrum, se mantuvieron los principios ágiles clave: el desarrollo iterativo, las entregas incrementales y la adaptación continua.

Esta metodología híbrida, que combinaba elementos de Kanban con principios ágiles generales, permitió mantener la agilidad y la capacidad de respuesta a lo largo de todo el ciclo de desarrollo del proyecto. La flexibilidad inherente a este enfoque fue fundamental para navegar los desafíos únicos que presenta un proyecto de inteligencia artificial, donde la experimentación y la adaptación constante son cruciales para el éxito.

3.2. Herramientas Utilizadas

En el desarrollo de este trabajo de fin de grado se han empleado diversas herramientas y tecnologías, cada una seleccionada por sus capacidades específicas para abordar los diferentes aspectos del proyecto. A continuación, se presenta una descripción detallada de cada herramienta utilizada:

3.2.1. Visual Studio Code

Visual Studio Code (VS Code) es un editor de código desarrollado por Microsoft, conocido por ser gratuito, de código abierto y altamente personalizable. En este proyecto, VS Code fue elegido como el editor de código principal. Su versatilidad y el amplio soporte para múltiples lenguajes de programación, además de su amplio catálogo de extensiones, fueron factores clave en esta decisión.

3.2.2. Python

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general, reconocido por su simplicidad y legibilidad. Su filosofía de diseño enfatiza la legibilidad del código, lo que lo hace accesible tanto para principiantes como para expertos. Python se utiliza en una variedad de campos, desde el desarrollo web y la ciencia de datos hasta la inteligencia artificial y la automatización de tareas. Para este trabajo de fin de grado, Python fue seleccionado como el lenguaje de programación principal. Su elección se basó en varios factores clave: su sintaxis clara y concisa permitió un desarrollo rápido y eficiente, mientras que su rico ecosistema de bibliotecas y gran soporte en el mundo de la inteligencia artificial, con librerías como PyTorch o las librerías de HuggingFace lo hicieron el lenguaje en el que se debía desarrollar este sistema.

3.2.3. Transformers de HuggingFace

Esta biblioteca proporciona implementaciones de última generación de arquitecturas de transformers, que son la base de muchos modelos de lenguaje avanzados. Transformers se utiliza ampliamente en tareas como clasificación de texto, generación de texto y traducción automática. En el contexto de este proyecto, la biblioteca Transformers fue fundamental para

implementar modelos de procesamiento de lenguaje natural basados en la arquitectura Transformer. Específicamente, se utilizaron los modelos relacionados con BERT, aprovechando su capacidad para entender el contexto y las relaciones semánticas en el texto, lo que fue crucial para las tareas de análisis y procesamiento de lenguaje natural requeridas en el proyecto.

3.2.4. BERT

BERT (Bidirectional Encoder Representations from Transformers) es un modelo de lenguaje desarrollado por Google que ha marcado un antes y un después en el campo del NLP. La versión multilingual Cased de BERT destaca por su capacidad para procesar texto en múltiples idiomas mientras mantiene la sensibilidad a mayúsculas y minúsculas. Este modelo se utiliza en una variedad de tareas multilingües, como clasificación de texto, reconocimiento de entidades nombradas y análisis de sentimientos en diferentes idiomas. En este trabajo, se eligió BERT multilingual Cased como el modelo preentrenado principal. La sensibilidad a mayúsculas y minúsculas del modelo fue especialmente útil para mantener la precisión en tareas que requerían distinguir entre nombres propios y comunes, o en situaciones donde la diferenciación entre mayúsculas y minúsculas era semánticamente relevante, como en los códigos de identificación como puede ser el CIF o el DNI.

3.2.5. PyTorch

PyTorch es una biblioteca de aprendizaje automático de código abierto desarrollada por Meta, conocida por su flexibilidad y eficiencia en el desarrollo de modelos de aprendizaje profundo. Su diseño permite una programación imperativa natural y proporciona aceleración GPU out-of-the-box, lo que la hace ideal para el desarrollo rápido de prototipos y la investigación en inteligencia artificial. En el contexto de este proyecto, PyTorch se utilizó como el framework principal de aprendizaje profundo. Su elección se basó en su flexibilidad para el diseño y entrenamiento de redes neuronales complejas, así como en su eficiencia en la ejecución de modelos de machine learning. PyTorch facilitó la implementación y el fine tuning de los modelos de lenguaje utilizados, permitiendo una integración fluida con las otras herramientas y bibliotecas del proyecto.

3.2.6. Jinja2

Jinja2 es un motor de plantillas para Python, inspirado en el motor de plantillas de Django. Es una herramienta poderosa para la generación dinámica de contenido, que se utiliza ampliamente en el desarrollo web para separar la lógica de la presentación. Jinja2 no se limita a la generación de páginas web; también se usa para crear correos electrónicos personalizados, generar informes y producir cualquier tipo de documento basado en plantillas. En este trabajo de fin de grado, Jinja2 desempeñó un papel crucial en la generación dinámica de documentos HTML, posteriormente renderizables como PDFs por Pyppeteer. Su flexibilidad y potencia fueron particularmente útiles para la creación de facturas y otros documentos estructurados.

La capacidad de Jinja2 para manejar lógica compleja dentro de las plantillas permitió generar documentos personalizados y detallados de manera eficiente, adaptándose a las necesidades específicas del proyecto.

3.2.7. Pyppeteer

Pyppeteer es una biblioteca de Python que proporciona una API de alto nivel para controlar Chrome o Chromium a través del protocolo DevTools. Esta herramienta es especialmente útil para la automatización de tareas en navegadores web, como pruebas automatizadas, web scraping de sitios con contenido dinámico, y generación de capturas de pantalla o PDFs de páginas web. En el contexto de este proyecto, Pyppeteer se utilizó en conjunto con Jinja2 para renderizar HTML y CSS. Esta combinación permitió la generación de facturas y otros documentos en formato PDF con un alto grado de fidelidad visual. Pyppeteer facilitó la conversión de los documentos HTML generados dinámicamente por Jinja2 en archivos PDF de alta calidad, asegurando que el formato y la presentación de los documentos fueran exactamente como se diseñaron.

3.2.8. Faker

Faker es una biblioteca de Python diseñada para generar datos falsos de manera realista y consistente. Es ampliamente utilizada en el desarrollo de software para crear conjuntos de datos de prueba, poblar bases de datos con información ficticia, y generar contenido de ejemplo para demostraciones. Faker puede producir una amplia variedad de tipos de datos, desde nombres y direcciones hasta números de teléfono y direcciones de correo electrónico, todo ello manteniendo una apariencia realista. En este proyecto, Faker jugó un papel importante en la generación de datos aleatorios realistas. Se utilizó para crear un conjunto de datos para el entrenamiento del modelo. La capacidad de Faker para generar datos consistentes y variados permitió generar una cantidad generosa de datos que le permitiría al modelo ser ciertamente robusto ante distintos escenarios.

3.2.9. Flask

Flask es un micro framework web escrito en Python, conocido por su simplicidad y flexibilidad. A diferencia de frameworks más grandes, Flask no impone una estructura rígida o requiere herramientas o bibliotecas específicas, lo que lo hace ideal para proyectos de diversos tamaños y complejidades. Es ampliamente utilizado para el desarrollo de aplicaciones web pequeñas a medianas y la creación de APIs RESTful. En el contexto de este trabajo de fin de grado, Flask se empleó como el framework principal para implementar el backend del proyecto y su API. Su naturaleza ligera y su facilidad de uso permitieron una rápida configuración y desarrollo del servidor. Flask facilitó la creación de endpoints para la API, la gestión de solicitudes HTTP, y la integración con otras componentes del sistema

3.2.10. React con Vite y TypeScript

React es una biblioteca de JavaScript ampliamente adoptada para la construcción de interfaces de usuario interactivas y reactivas. Vite, por su parte, es un empaquetador moderno que ofrece una experiencia de desarrollo más rápida y eficiente. TypeScript, un superconjunto tipado de JavaScript, añade un sistema de tipos a JavaScript, mejorando la calidad y mantenibilidad del código. La combinación de estas tres tecnologías se utilizó en este proyecto para el desarrollo de la interfaz de usuario. React proporcionó una base sólida para gestionar eficientemente el estado de la aplicación. Vite se eligió como bundler por su velocidad de compilación y su configuración mínima, lo que aceleró significativamente el proceso de desarrollo. TypeScript, por su parte, añadió una capa adicional de seguridad y claridad al código, facilitando el mantenimiento y reduciendo los errores potenciales. Esta combinación resultó en una interfaz de usuario robusta, eficiente y fácil de mantener.

3.2.11. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de software de visión artificial y aprendizaje automático de código abierto. Es ampliamente utilizada en aplicaciones de procesamiento de imágenes y vídeos, detección y reconocimiento de objetos, y en el desarrollo de sistemas de visión por computadora en tiempo real. En este proyecto, OpenCV se utilizó principalmente para el preprocesamiento y manipulación de imágenes. Sus capacidades avanzadas de procesamiento de imágenes fueron cruciales para preparar los datos visuales antes de aplicar técnicas de reconocimiento óptico de caracteres. OpenCV se usó principalmente para extraer los cuadros de texto y aplicar filtros, mejorando así la calidad y la precisión del proceso de OCR subsiguiente.

3.2.12. TesseractOCR y EasyOCR

TesseractOCR es un motor de reconocimiento óptico de caracteres (OCR) de código abierto, reconocido por su precisión y versatilidad. EasyOCR, por otro lado, es una biblioteca de Python que proporciona una interfaz fácil de usar para realizar OCR y reconocimiento de texto en más de 80 idiomas. Ambas herramientas son ampliamente utilizadas en la digitalización de documentos, la extracción de texto de imágenes, y la automatización de procesos que involucran datos textuales en formato de imagen. En el contexto de este trabajo, TesseractOCR se usó en primer lugar junto con OpenCV para extraer texto, sin embargo la implementación de EasyOCR de modelos de identificación de texto, la hizo la solución final al tipo de problema que nos encontramos.

3.2.13. Docker

Docker es una plataforma de contenedorización que ha revolucionado la forma en que se desarrollan, despliegan y ejecutan aplicaciones. Permite empaquetar una aplicación junto

con todas sus dependencias en un contenedor virtual estandarizado, lo que garantiza que la aplicación funcione de manera uniforme en cualquier entorno que soporte Docker. En este proyecto, Docker se utilizó para contenerizar el sistema completo. Esta decisión facilitó enormemente la gestión de dependencias y aseguró la consistencia entre los distintos entornos de ejecución. Además, simplificó el proceso de configuración del entorno de desarrollo, permitiendo replicar fácilmente el entorno de trabajo en diferentes máquinas.

3.2.14. CUDA

CUDA (Compute Unified Device Architecture) es una plataforma de computación paralela y un modelo de programación desarrollado por NVIDIA. Está diseñada para aprovechar la potencia de procesamiento de las unidades de procesamiento gráfico (GPUs) para tareas de cómputo general. CUDA se utiliza ampliamente en campos que requieren cálculos intensivos, como el aprendizaje profundo, el procesamiento de imágenes y vídeo de alto rendimiento, y las simulaciones científicas complejas. En el contexto de este trabajo de fin de grado, CUDA se aprovechó para la aceleración por GPU del entrenamiento y la inferencia del modelo de aprendizaje profundo. Esta tecnología fue crucial para mejorar significativamente el rendimiento en tareas computacionalmente intensivas, como el entrenamiento de modelos de lenguaje y el procesamiento de grandes volúmenes de datos. La utilización de CUDA permitió reducir drásticamente los tiempos de entrenamiento y mejorar la eficiencia en la fase de inferencia, lo que fue especialmente beneficioso dado el alcance y la complejidad del proyecto.

3.2.15. Git

Git es un sistema de control de versiones distribuido que se ha convertido en el estándar de facto para la gestión de código fuente en la industria del software. Desarrollado inicialmente por Linus Torvalds para el desarrollo del kernel de Linux, Git destaca por su diseño distribuido que permite a los desarrolladores trabajar de forma offline, realizar ramificaciones y fusiones de código de manera eficiente, y mantener un historial completo de cambios en el proyecto. En el contexto de este trabajo de fin de grado, Git desempeñó un papel fundamental en la gestión del código fuente.

3.2.16. GitHub

GitHub es una plataforma de gestión de código de que utiliza Git como sistema de control de versiones subyacente. Esta plataforma amplía las funcionalidades de Git, ofreciendo herramientas adicionales para la gestión de proyectos. Se ha usado en este proyecto para albergar el código de manera pública, relacionado con el carácter Open Source del mismo y el objetivo de proporcionar una puerta de entrada a este tipo de proyectos a todo aquel que quiera implementar una solución similar.

Capítulo 4

Desarrollo

El presente capítulo aborda el proceso de desarrollo del proyecto, estructurado en tres iteraciones fundamentales. Inicialmente, se llevó a cabo una exhaustiva fase de investigación y generación de datos sintéticos, estableciendo así los cimientos para las etapas subsiguientes. La segunda iteración se centró en la creación de un prototipo funcional que implementara las tareas de extracción automatizada propuestas, permitiendo evaluar la viabilidad técnica del proyecto. Finalmente, la tercera fase consistió en la optimización y generalización del prototipo inicial, incluyendo su implementación como servicio, el desarrollo de una interfaz de usuario y la creación de un contenedor para facilitar su ejecución y despliegue.

A lo largo de las siguientes secciones, se detallarán los procesos, desafíos y logros asociados a cada una de estas etapas, proporcionando una visión integral del desarrollo del sistema de extracción automatizada.

4.1. Primera iteración

Esta fase del proyecto se inició con la conceptualización de las funcionalidades clave del sistema propuesto. Posteriormente, se llevó a cabo una exhaustiva investigación de mercado y una revisión de la literatura académica pertinente. Paralelamente, se desarrolló un esquema preliminar que delineaba la arquitectura y el flujo de operaciones del proyecto.

4.1.1. Investigación inicial

La etapa inicial de este proyecto, que precedió a la implementación del sistema y su adopción como trabajo de fin de titulación, se centró en una exhaustiva investigación. Esta fase consistió principalmente en la búsqueda y análisis de herramientas y productos capaces de extraer datos estructurados de facturas, independientemente de su formato, con el propósito de facilitar tareas de análisis financiero, gestión financiera automatizada e incluso gestión automatizada de impuestos.

En primera instancia, la investigación se enfocó en el entorno comercial. Las soluciones de software como servicio (SaaS, por sus siglas en inglés) destacaron por su competitividad, atribuible a los significativos recursos económicos invertidos en su desarrollo. Sin embargo, al reorientar nuestra atención hacia la implementación de un sistema similar, se evidenció una notable carencia: la ausencia de herramientas de código abierto que ejecutaran el proceso completo de procesamiento de facturas o documentos financieros para la obtención de datos estructurados. Esta constatación reveló una oportunidad para contribuir significativamente a la comunidad de código abierto mediante el desarrollo de nuestro proyecto.

Habiendo verificado la viabilidad de desarrollar este tipo de soluciones, nuestra investigación se dirigió hacia fuentes académicas en busca de tecnologías de vanguardia aplicables al proyecto. En este contexto, identificamos la denominación científica del problema que abordamos: Reconocimiento de Entidades Nombradas (NER, por sus siglas en inglés). Típicamente, estos sistemas implementan una serie de modelos multimodales que integran capacidades visuales de reconocimiento de texto, extracción textual y procesamiento del lenguaje natural, junto con la interpretación de datos posicionales en documentos. Esta combinación permite la extracción precisa de información de facturas y otros documentos estructurados similares, como recibos o documentos de identificación.

El artículo titulado *Review of Semi-Structured Document Information Extraction Techniques Based on Deep Learning* [7] resultó particularmente esclarecedor. Esta revisión exhaustiva de sistemas similares nos proporcionó una excelente base y una dirección clara sobre cómo implementar nuestro sistema y qué tecnologías podíamos utilizar. La síntesis de diversas técnicas y enfoques presentada en este trabajo fue fundamental para estructurar nuestra aproximación al problema.

4.1.2. Generación del conjunto de datos

Dadas las limitaciones económicas y temporales inherentes al proyecto, se optó por una estrategia innovadora para la generación del conjunto de datos de facturas. Esta aproximación implicó la creación de plantillas utilizando HTML y CSS, lo que permitió no solo dotar a las facturas de un aspecto profesional, sino también facilitar la generación masiva de documentos en formato PDF, acompañados de sus correspondientes representaciones estructuradas en JSON.

Ilustración 4.1: Prototipo inicial del modelo de factura

El formato de los datos de salida del modelo se diseñó para seguir una estructura específica, representativa de la información que el sistema debía extraer de cada factura procesada. A continuación, se presenta un ejemplo ilustrativo de esta estructura en formato JSON:

```json
{
    "invoice_id": 216103,
    "issue_date": "2024-06-17",
    "due_date": "2024-07-28",
    "issuer": {
      "name": "Ruth Reguera Macías S.L.L.",
      "address": "Alameda Ani Peña 48 Puerta 7 \nGuipúzcoa, 32788",
      "phone": "+34822 616 384",
      "email": "gilbertolobato@mora.es",
      "tax_id": "SHh622SRc270"
    },
    "recipient": {
      "name": "Alex del Santiago",
      "address": "Calle María José Solís 66 Puerta 1 \nLas Palmas, 01330",
      "phone": "+34922 29 17 19",
      "email": "blancomaura@yahoo.com",
      "tax_id": "XPX524zSR471"
    },
    "items": [
      {
        "description": "syndicate end-to-end architectures",
        "quantity": 2,
        "unit_price": "74.19",
        "total": "148.38"
      },
      {
        "description": "cultivate 24/7 e-services",
        "quantity": 9,
        "unit_price": "59.11",
        "total": "531.99"
      },
      {
        "description": "engage impactful niches",
        "quantity": 3,
        "unit_price": "27.12",
        "total": "81.36"
      },
      {
        "description": "transform ubiquitous e-commerce",
        "quantity": 8,
        "unit_price": "95.07",
        "total": "760.56"
      }
    ],
    "subtotal": "1522.29",
    "taxes": [
      {
        "description": "VAT",
        "percentage": 16,
        "amount": "243.57"
      }
    ],
    "total": "1765.86",
    "payment_method": "Bank Transfer",
    "currency": "EUR"
}
```

Algoritmo 4.1: Estructura JSON de una factura de ejemplo

Para la implementación efectiva de este proceso de generación, se emplearon dos herramientas fundamentales:

1. **Jinja2**: Este potente motor de plantillas, ampliamente utilizado en el framework Django, se seleccionó para la sustitución dinámica de campos en las plantillas HTML. Su flexibilidad y eficiencia permitieron una generación fluida de facturas.
2. **Faker**: Esta biblioteca se utilizó para la generación de datos aleatorios pero verosímiles. Faker facilitó la creación de un amplio espectro de información ficticia pero realista, esencial para simular la diversidad de datos que se encuentran en facturas reales.

La combinación de estas tecnologías permitió la creación de un conjunto de datos robusto y variado, fundamental para el entrenamiento y evaluación del sistema de extracción de información de facturas.

4.1.3. Boceto inicial del sistema

El boceto inicial del sistema se conceptualizó como una arquitectura de dos componentes principales, diseñada para procesar y extraer datos de facturas de manera eficiente. La Figura 4.2 ilustra este diseño preliminar.

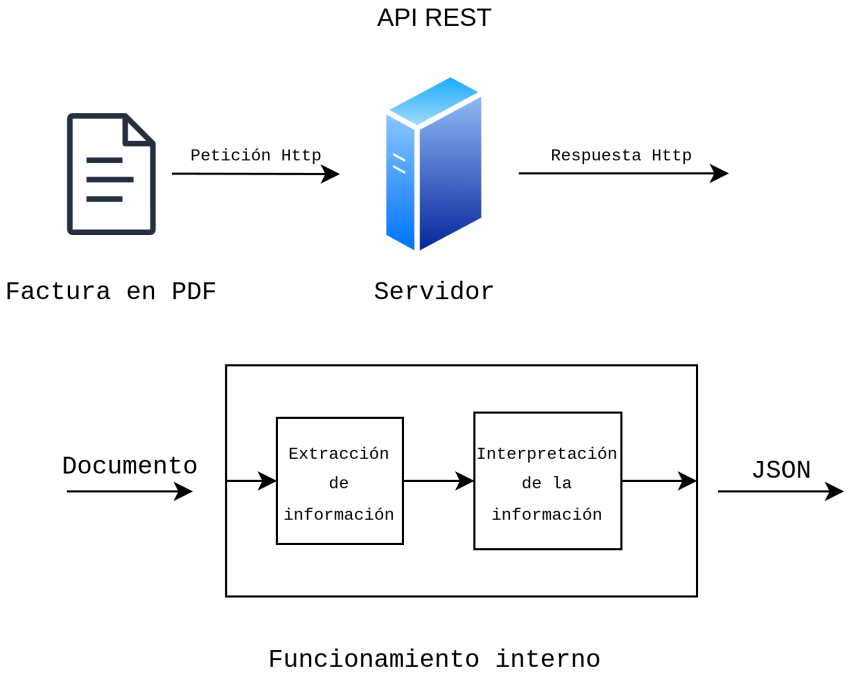


Ilustración 4.2: Boceto del sistema

El primer componente es una interfaz API REST, que permite la comunicación entre el cliente y el servidor:

- ✔ El cliente envía una factura en formato PDF mediante una petición HTTP.
- ✔ El servidor procesa esta petición y devuelve una respuesta HTTP con los datos extraídos.

El segundo componente representa el funcionamiento interno del sistema, que se divide en dos etapas principales:

1. Extracción de información: En esta fase, el sistema analiza el documento de entrada para extraer los datos relevantes.
2. Interpretación de la información: Aquí, los datos extraídos se procesan y estructuran en un formato coherente.

El resultado final es un documento JSON que contiene la información estructurada de la factura.

4.1.4. Natural Language Processing

En el ámbito del Procesamiento del Lenguaje Natural (NLP), la búsqueda de arquitecturas eficientes para el Reconocimiento de Entidades Nombradas (NER) ha sido un área de intensa investigación y desarrollo. Nuestro proyecto, centrado en la extracción precisa de datos de facturas, nos llevó a explorar diversas opciones dentro de este campo en evolución.

Inicialmente, las redes neuronales recurrentes (RNN) se presentaban como una opción prometedora. Estas redes, diseñadas para procesar secuencias de datos, parecían adecuadas para la tarea de identificar y clasificar entidades en el texto de las facturas. Su capacidad para mantener un estado interno que se actualiza con cada elemento de la secuencia las hacía particularmente atractivas para capturar el contexto en documentos estructurados como las facturas.

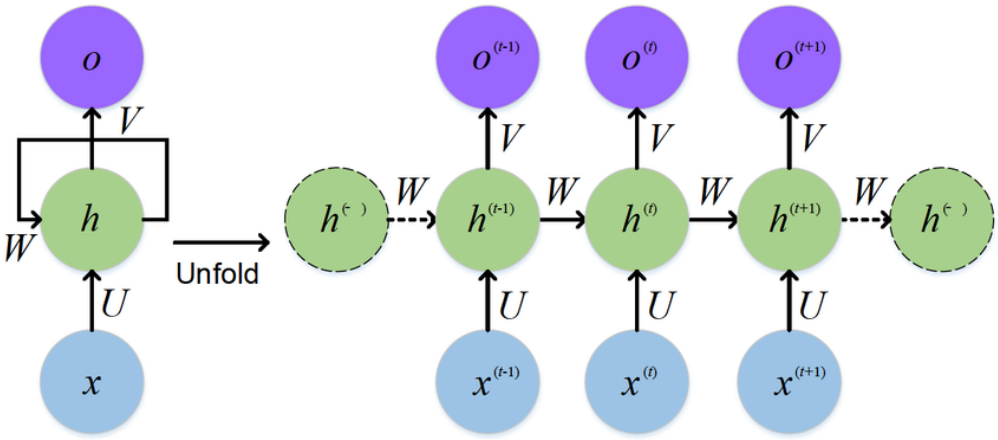


Ilustración 4.3: Arquitectura de las redes neuronales recurrentes [3]

Sin embargo, a medida que profundizamos en nuestra investigación, se hicieron evidentes las limitaciones de las RNN, especialmente en el manejo de dependencias a largo plazo en

documentos extensos. Este aspecto era crucial para nuestro proyecto, dado que la interpretación precisa de una factura a menudo requiere relacionar información dispersa a lo largo del documento.

El panorama cambió significativamente con la introducción de la arquitectura Transformer, presentada en el influyente artículo "Attention Is All You Need" de Vaswani et al. [15]. Los Transformers, basados únicamente en mecanismos de atención, ofrecían ventajas significativas que parecían abordar directamente los desafíos que enfrentábamos:

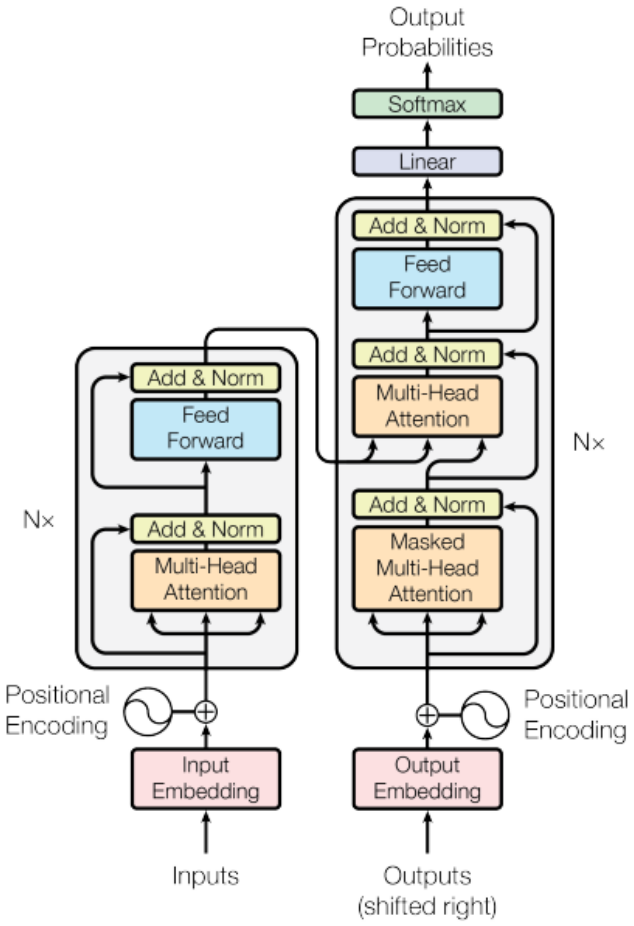


Ilustración 4.4: Arquitectura del modelo Transformer [15]

Es importante destacar las diferencias fundamentales entre las RNN y los Transformers:

- ✔ Procesamiento secuencial vs paralelo: Las RNN procesan los datos de forma secuencial, lo que puede ser ineficiente para secuencias largas. Los Transformers, por otro lado, pueden procesar todos los elementos de una secuencia en paralelo, lo que resulta en un procesamiento más rápido.
- ✔ Manejo de dependencias a largo plazo: Las RNN pueden sufrir del problema de desvanecimiento del gradiente, lo que dificulta la captura de dependencias a largo plazo.

Los Transformers, gracias a su mecanismo de atención, pueden capturar eficientemente relaciones entre elementos distantes en una secuencia.

- ✔ Complejidad computacional: La complejidad de las RNN aumenta linealmente con la longitud de la secuencia, mientras que los Transformers tienen una complejidad constante en términos de pasos de procesamiento, aunque requieren más memoria.
- ✔ Capacidad de paralelización: Los Transformers son altamente paralelizables, lo que los hace más eficientes en hardware moderno como GPUs y TPUs.

Considerando estas ventajas y las características específicas de nuestra tarea de NER en facturas, determinamos que BERT (Bidirectional Encoder Representations from Transformers), un modelo basado en la arquitectura Transformer, era la opción más adecuada para comenzar nuestro desarrollo. BERT ofrecía un equilibrio óptimo entre precisión en tareas de NER, eficiencia computacional, adaptabilidad a nuestro dominio específico, y disponibilidad de recursos y soporte comunitario.

En las siguientes secciones, exploraremos en detalle cómo implementamos y optimizamos nuestro modelo basado en BERT para la tarea que nos acontece.

4.2. Segunda iteración

La segunda iteración de nuestro proyecto se centró en el desarrollo de un Producto Mínimo Viable (MVP), marcando un paso crucial en la evolución de nuestro sistema. Este MVP consistió en un sistema simplificado pero funcional, capaz de procesar un modelo único de facturas y generar una salida estructurada en formato JSON.

En esta fase, implementamos los componentes esenciales del sistema. Utilizamos OpenCV para la extracción de bounding boxes con texto útil, delimitando las áreas de interés en las imágenes de las facturas. Para convertir el texto de las imágenes en datos procesables, hicimos uso de TesseractOCR. Además, desarrollamos herramientas para el formateo en BIO (Beginning, Inside, Outside), preparando así los datos para el entrenamiento de BERT. Finalmente, creamos un intérprete para transformar los tokens clasificados por BERT en una estructura JSON coherente.

Este MVP nos permitió validar conceptos clave, identificar desafíos técnicos y establecer una base sólida para futuras iteraciones. A lo largo de esta sección, exploraremos en detalle el proceso de desarrollo, los problemas encontrados y las soluciones implementadas, proporcionando una visión clara de cómo evolucionó nuestro sistema desde un concepto inicial hasta un prototipo funcional.

4.2.1. Entrenamiento del modelo

Inicialmente, consideramos que los archivos JSON serían suficientes para entrenar el modelo, esperando que fuera capaz de interpretar las facturas de manera autónoma. Sin embargo,

pronto nos percatamos de que este enfoque era inadecuado para nuestro objetivo.

Los modelos de clasificación de texto, como BERT, funcionan etiquetando cada palabra con su correspondiente etiqueta semántica, indicando su función o significado en el contexto que la rodea. Esta realización nos llevó a buscar una solución más apropiada para generar un dataset de entrenamiento para BERT.

Para entrenar un modelo de lenguaje que interprete los elementos de interés, es crucial definir las etiquetas a utilizar y su formato. Optamos por el formato BIO (Beginning, Inside, Outside), que consiste en tres tipos de etiquetas para cada elemento semántico. Este formato no se limita necesariamente a una palabra, sino que puede abarcar estructuras más complejas como nombres compuestos. Por ejemplo, BERT interpretaría mi nombre de la siguiente manera:

```
Antonio => B-name
Javier  => I-name
Torres  => I-name
Bordón  => O-name
```

Algoritmo 4.2: Ejemplo de etiquetado BIO para nombres

Como se observa, B-name marca el inicio del nombre, I-name indica la continuación, y O-name señala la palabra que cierra ese nombre. Este formato se considera robusto para tareas de clasificación de texto según investigaciones previas, por lo que decidimos adoptarlo.

Las etiquetas se ajustan a los elementos que necesitamos extraer de la factura, previamente definidos en el JSON:

```
labels = [
  "O",  # Para tokens que no son parte de ninguna entidad nombrada
  "B-invoice_id", "I-invoice_id",
  "B-issue_date", "I-issue_date",
  "B-due_date", "I-due_date",
  "B-issuer_name", "I-issuer_name",
  "B-issuer_address", "I-issuer_address",
  "B-issuer_phone",
  "B-issuer_email",
  "B-issuer_tax_id",
  "B-recipient_name", "I-recipient_name",
  "B-recipient_address", "I-recipient_address",
  "B-recipient_phone",
  "B-recipient_email",
  "B-recipient_tax_id",
  "B-item_description", "I-item_description",
  "B-item_quantity",
  "B-item_unit_price",
  "B-item_total",
  "B-subtotal",
  "B-tax_description", "I-tax_description",
  "B-tax_percentage",
  "B-tax_amount",
  "B-total",
  "B-payment_method",
  "UNK"
```

```
]
```

Algoritmo 4.3: Definición de etiquetas para el modelo BERT

Es importante distinguir entre las etiquetas O y UNK. La etiqueta O se asigna a palabras que, aunque no son entidades nombradas de interés, contribuyen al contexto. Por otro lado, la etiqueta UNK se utiliza para tokens especiales como los de apertura y cierre, o para partes no utilizadas de la secuencia de entrada, dado que BERT tiene una entrada fija de 512 tokens, que rara vez se llena completamente en nuestro caso.

Para continuar con el entrenamiento de BERT, es necesario comprender el concepto de token. BERT no interpreta palabras enteras; su tokenizador divide las palabras en unidades más pequeñas llamadas tokens. El siguiente ejemplo ilustra cómo BERT interpreta el dataset:

```
[CLS] ['UNK']
Con ['B-issuer_name']
##sul ['B-issuer_name']
##tor ['B-issuer_name']
##ía ['B-issuer_name']
Domingo ['I-issuer_name']
...
```

Algoritmo 4.4: Ejemplo de tokenización de BERT

Esta muestra, extraída del notebook de Jupyter en los repositorios del sistema, evidencia que la tokenización depende de la palabra. Esto presenta un desafío, ya que el dataset que hemos generado, cuyo funcionamiento detallado y justificación se explicarán en la siguiente sección, tiene el siguiente formato:

```
Factura -> O
F302087 -> B-invoice_id
Fecha -> O
de -> O
emisión -> O
: -> O
2024-04-03 -> B-issue_date
Fecha -> O
de -> O
vencimiento -> O
: -> O
2024-05-04 -> B-due_date
Datos -> O
del -> O
Emisor -> O
: -> O
Datos -> O
del -> O
Receptor -> O
: -> O
Escobar -> B-issuer_name
Ltd -> I-issuer_name
Dawn -> B-recipient_name
Huff -> I-recipient_name
23753 -> B-issuer_address
Kevin -> I-issuer_address
```

```
...
```

Algoritmo 4.5: Formato del dataset generado

Para abordar esta discrepancia, desarrollamos una función que ajusta la lista de etiquetas a los nuevos tokens generados por cada palabra. Esta función, implementada en el notebook, expande la etiqueta original para cubrir todos los tokens derivados de una palabra:

```
...

texts = []  # Lista de textos de factura
tags = []  # Lista de etiquetas (cada etiqueta es una lista de ids de etiquetas)

def load_tags(file_path):
    tags = []
    with open(file_path, 'r') as file:
        for line in file:
            # Dividir la línea por ' -> ' y tomar el segundo elemento, que es la etiqueta
            parts = line.strip().split(' -> ')
            if len(parts) > 1:
                tags.append(parts[1])  # Agrega la etiqueta a la lista
    return tags

def load_text(file_path):
    texts = []
    with open(file_path, 'r') as file:
        for line in file:
            # Dividir la línea por ' -> ' y tomar el segundo elemento, que es la etiqueta
            parts = line.strip().split(' -> ')
            if len(parts) > 1:
                texts.append(parts[0])  # Agrega la etiqueta a la lista
    return texts

tags = []
for i in range(1000):  # Ajusta el rango según la cantidad de facturas
    tags.append(load_tags(f'dataset_output/train/train{i}.tokens'))
    texts.append(load_text(f'dataset_output/train/train{i}.tokens'))

...
```

Algoritmo 4.6: Código para cargar y procesar el dataset

Una vez preparado el dataset para el entrenamiento, procedemos a entrenar el modelo. Dividimos el dataset en conjuntos de entrenamiento y validación, y configuramos el proceso de la siguiente manera:

```
...

train_texts, val_texts, train_labels, val_labels = train_test_split(texts, tags, test_size=0.1)

train_dataset = InvoiceDataset(train_texts, train_labels)
val_dataset = InvoiceDataset(val_texts, val_labels)
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = BertForTokenClassification.from_pretrained(MODEL_NAME, num_labels=len(labels))
```

```
model.to(device)  # Mover el modelo a la GPU si está disponible

optimizer = AdamW(model.parameters(), lr=5e-5)

def train(model, dataloader, optimizer):
    model.train()
    total_loss = 0
    for batch in dataloader:
        # Mover los datos al dispositivo correcto
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(dataloader)

def evaluate(model, dataloader):
    model.eval()  # Pone el modelo en modo evaluación
    total_accuracy = 0
    total_tokens = 0

    with torch.no_grad():
        for batch in dataloader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask)
            predictions = outputs.logits.argmax(dim=-1)  # Obtén la clase predicha para cada token

            # Ignorar tokens con etiquetas -100 (usadas en algunos \gls{datasets} para ignorar ciertos tokens)
            mask = (labels != -100)
            correct_predictions = (predictions == labels) & mask
            total_accuracy += correct_predictions.sum().item()
            total_tokens += mask.sum().item()

    return total_accuracy / total_tokens

...
```

Algoritmo 4.7: Código para el entrenamiento y evaluación del modelo BERT

Durante el entrenamiento, que se realiza mediante la ejecución de la función train, observamos que BERT alcanza resultados sobresalientes en muy pocas iteraciones. Esto se debe principalmente a que BERT ya está preentrenado con una gran cantidad de datos, y con solo dos iteraciones logramos adaptarlo a nuestra tarea específica:

```
# Entrenamiento
for epoch in range(30):
    train_loss = train(model, train_loader, optimizer)
    val_accuracy = evaluate(model, val_loader)
    print(f'Epoch {epoch + 1}, Train Loss: {train_loss}, Val Accuracy: {val_accuracy:.2%}')

###  Salida
###  Epoch 1, Train Loss: 0.3294720137739076, Val Accuracy: 99.97%
###  Epoch 2, Train Loss: 0.007293984328965302, Val Accuracy: 100.00%
###  ---------------------------------------------------------------------------
###  KeyboardInterrupt                         Traceback (most recent call last)
```

```
###  Cell In[23], line 3
###        1 # Entrenamiento
###        2 for epoch in range(30):
###  ----> 3     train_loss = train(model, train_loader, optimizer)
###        4     val_accuracy = evaluate(model, val_loader)
###        5     print(f'Epoch {epoch + 1}, Train Loss: {train_loss}, Val Accuracy: {val_accuracy:.2%}')
###
###  Cell In[20], line 34
###       32     loss.backward()
###       33     optimizer.step()
###  ---> 34     total_loss += loss.item()
###       35 return total_loss / len(dataloader)
###
###  KeyboardInterrupt:
```

Algoritmo 4.8: Resultados del entrenamiento de BERT

Los resultados obtenidos del entrenamiento, procesados para unir los tokens, son los siguientes:

```
[CLS] -> UNK
Factura -> O
FH600301 -> B-recipient_tax_id
Fechadeemisión: -> O
2024-03-11 -> B-issue_date
Fechadevencimiento: -> O
2024-04-15 -> B-due_date
DatosdelEmisor:DatosdelReceptor: -> O
Williams,York andSchwartz KatrinaFritz540StrongGreenNorthApril,AK086280936 ButlerVillagesApt.228
   Williamhaven,NY240612803178625 -> B-recipient_name
+1-905-539-8849x13583 -> B-recipient_phone
hamptondanielle(Ogriffin.commariah650gmail.com -> B-recipient_email
vsP770FmMb986arF309WLD229 -> B-recipient_tax_id
DescripciónCantidadPrecioUnitarioTotal -> O
expedite24/7systems -> B-item_description
9 -> B-item_quantity
80.86 -> B-item_unit_price
727.74 -> B-item_total
orchestrateweb-enabledmodels -> B-item_description
7 -> B-item_quantity
65.31 -> B-item_unit_price
```

Algoritmo 4.9: Resultados de clasificación del modelo BERT entrenado

Como se puede apreciar, hemos logrado desarrollar un modelo capaz de clasificar eficazmente los elementos de la factura. Para un análisis más detallado del proceso de entrenamiento y los resultados, se recomienda consultar el notebook adjunto en el repositorio de entrenamiento de BERT.

4.2.2. Extracción de características visuales y texto

En esta sección, abordaremos el proceso de extracción de características visuales y texto de las facturas utilizando las bibliotecas OpenCV para el procesamiento de imágenes y Tesseract para el reconocimiento óptico de caracteres (OCR). Este paso es crucial para transformar la información visual de las facturas en datos procesables para nuestro modelo BERT.

Preparación de la imagen

Inicialmente, cargamos la imagen de la factura y la preparamos para el procesamiento. Utilizamos la biblioteca PIL (Python Imaging Library) para abrir la imagen y luego la convertimos al formato compatible con OpenCV:

```
from PIL import Image
import cv2
import numpy as np

image = Image.open("factura0.jpg")
image_cv2 = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
```

Algoritmo 4.10: Carga y conversión de la imagen

A continuación, aplicamos una serie de transformaciones para mejorar la calidad de la imagen y facilitar la extracción de texto:

```
image_gray = cv2.cvtColor(image_cv2, cv2.COLOR_BGR2GRAY)
image_binary = cv2.threshold(image_gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
```

Algoritmo 4.11: Preprocesamiento de la imagen

Estas transformaciones incluyen la conversión a escala de grises y la aplicación de un umbral binario utilizando el método de Otsu, que ayuda a separar el texto del fondo.

Extracción de texto con Tesseract

Una vez preprocesada la imagen, utilizamos Tesseract para extraer el texto:

```
import pytesseract

def to_string(image):
    return pytesseract.image_to_string(image)

text = to_string(image_binary)
print(text)
```

Algoritmo 4.12: Extracción de texto con Tesseract

Este proceso nos proporciona una primera versión del texto contenido en la factura. Sin embargo, la precisión puede variar dependiendo de la calidad de la imagen y la complejidad del layout.

Extracción de características visuales con OpenCV

Para mejorar la precisión de la extracción de texto, implementamos un proceso de detección de regiones de interés (ROI) utilizando OpenCV. Este enfoque nos permite identificar y extraer áreas específicas de la factura, como la tabla de productos o los datos del emisor y receptor.

Primero, aplicamos una serie de transformaciones morfológicas para resaltar las regiones de texto:

```
blur = cv2.GaussianBlur(gray, (7, 7), 0)
thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (13,5))
dilate = cv2.dilate(thresh, kernel, iterations=15)
```

Algoritmo 4.13: Transformaciones morfológicas

Luego, detectamos los contornos en la imagen dilatada y los convertimos en bounding boxes:

```
cnts = cv2.findContours(dilate, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
boxes = [cv2.boundingRect(c) for c in cnts]
```

Algoritmo 4.14: Detección de contornos y creación de bounding boxes

Para mejorar la detección de líneas de texto completas, implementamos una función que agrupa las bounding boxes que están aproximadamente a la misma altura:

```
def merge_boxes_on_same_row(boxes, y_threshold):
    # Código de la función (omitido por brevedad)
    return merged_boxes

grouped_boxes = merge_boxes_on_same_row(boxes, y_threshold=10)
```

Algoritmo 4.15: Agrupación de bounding boxes

El resultado obtenido es el siguiente:

Finalmente, aplicamos OCR a cada región detectada:

```
for (x, y, w, h) in grouped_boxes:
    roi = gray[y:y+h, x:x+w]
    text = pytesseract.image_to_string(roi, config='--psm 6')
    print(text)
```

Algoritmo 4.16: OCR por región

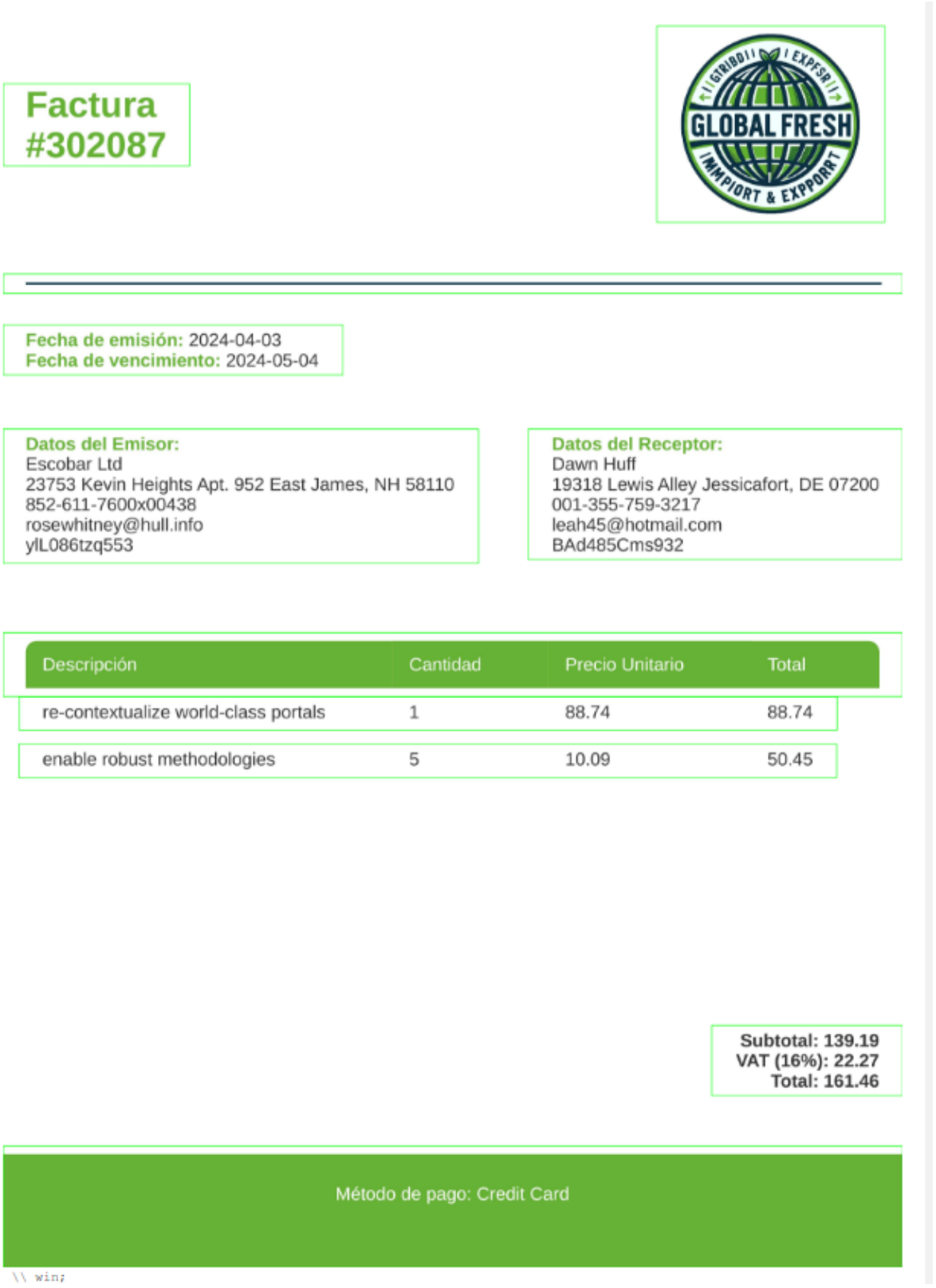


Ilustración 4.5: Resultado de la detección de regiones de texto con filtros

Este enfoque nos permite obtener una extracción de texto más precisa y estructurada, facilitando la posterior clasificación y etiquetado de la información para nuestro modelo BERT.

4.2.3. Generación del dataset, síntesis de datos similares a la extracción visual

Una vez completado el proceso de extracción de cuadros de texto de las facturas, el siguiente paso crucial fue la generación de un dataset sintético que simulara esta salida. Este dataset es esencial para entrenar nuestro modelo BERT de manera efectiva, asegurando que pueda adaptarse a las características específicas de las facturas procesadas por nuestro sistema de extracción de datos visuales.

Para lograr esto, desarrollamos una librería especializada que transforma los datos estructurados en formato JSON a un formato de texto etiquetado que emula la salida del OCR. Esta librería consta de una serie de funciones, cada una diseñada para manejar un aspecto específico de la factura, como el nombre del emisor, la dirección del receptor, o los detalles de los ítems.

A continuación, se presenta un ejemplo de una de estas funciones, específicamente la que maneja el nombre del emisor:

```
def issuer_name_to_bio(data):
    json_data = \GLS{JSON}.loads(data)

    # Extract the issuer name
    issuer_name = json_data['issuer']['name']

    issuer_name = issuer_name.split()

    text = textwrap.dedent(f"""
          Datos -> O
          del -> O
          Emisor -> O
          : -> O
          """)

    for i, name in enumerate(issuer_name):
        if i == 0:
            text += f"{name} -> B-issuer_name\n"
        else:
            text += f"{name} -> I-issuer_name\n"

    return textwrap.dedent(text).strip()
```

Algoritmo 4.17: Función que convierte el nombre del emisor a formato etiquetado

Esta función toma los datos JSON, extrae el nombre del emisor, y lo convierte en un formato etiquetado que simula cómo aparecería en un documento procesado por OCR. Incluye etiquetas contextuales (como "Datos del Emisor:") y etiqueta cada palabra del nombre del emisor según su posición.

Nuestra biblioteca incluye funciones similares para cada campo relevante de la factura. Por brevedad, se omite el contenido detallado de estas funciones, pero se presenta a continuación una lista de sus firmas:

```
def issue_date_to_bio(data):
def due_date_to_bio(data):
def invoice_id_to_bio(data):
def issuer_address_to_bio(data):
def issuer_phone_to_bio(data):
def issuer_email_to_bio(data):
def issuer_tax_id_to_bio(data):
def recipient_name_to_bio(data):
def recipient_address_to_bio(data):
def recipient_phone_to_bio(data):
def recipient_email_to_bio(data):
def recipient_tax_id_to_bio(data):
def item_description_to_bio(data, item_index=0):
def item_quantity_to_bio(data, item_index=0):
def item_unit_price_to_bio(data, item_index=0):
def item_total_to_bio(data, item_index=0):
def subtotal_to_bio(data):
def tax_description_to_bio(data):
def tax_percentage_to_bio(data):
def tax_amount_to_bio(data):
def total_to_bio(data):
def payment_method_to_bio(data):
def table_header_to_bio():
```

Algoritmo 4.18: Firmas de las funciones de parseo a formato etiquetado

Estas funciones son los bloques de construcción fundamentales para nuestro generador de dataset. Se combinan para crear fragmentos más grandes de texto etiquetado, simulando diferentes partes de una factura procesada.

El proceso de generación del dataset se implementa en un script principal, que utiliza estas funciones para crear un conjunto diverso de ejemplos de entrenamiento. Este script genera facturas aleatorias en formato JSON y luego las convierte a texto etiquetado, creando un dataset heterogéneo que abarca diferentes aspectos de las facturas.

```
import libraries.random_data_generator as rdg
import libraries.bio_text_parser as btp
import \GLS{JSON}

def invoice_id_chunk(json_input):
    return btp.invoice_id_to_bio(json_input)

def date_chunk(json_input):
    return btp.issue_date_to_bio(json_input) + "\n" +  btp.due_date_to_bio(json_input)

# ... [otras funciones de chunks omitidas por brevedad]

def generate_full_bio_text(json_input):
    text = invoice_id_chunk(json_input) + "\n"
    text += date_chunk(json_input) + "\n"
    text += issuer_data_chunk(json_input) + "\n"
    text += recipient_data_chunk(json_input) + "\n"
    text += full_items_chunk(json_input) + "\n"
    text += total_chunk(json_input) + "\n"
    text += payment_method_chunk(json_input)
```

```
    return text

if __name__ == "__main__":
    directory = "dataset_output/train/"
    for i in range(1000):
        json_input = rdg.generate_invoice()
        selector = i % 8
        file_name = f"train{i}.tokens"
        with open(directory + file_name, 'w') as file:
            if selector == 0:
                file.write(invoice_id_chunk(json_input) + '\n')
            elif selector == 1:
                file.write(date_chunk(json_input) + '\n')
            # ... [otros casos omitidos por brevedad]
            elif selector == 7:
                file.write(generate_full_bio_text(json_input) + '\n')
```

Algoritmo 4.19: Código del generador del Dataset

Este script genera 1000 ejemplos de entrenamiento, alternando entre diferentes tipos de contenido (número de factura, fechas, datos del emisor, etc.) para crear un dataset equilibrado y diverso. Utiliza un enfoque de Round Robin para asegurar una distribución uniforme de los diferentes tipos de datos.

Cabe destacar que este código está accesible en commits antiguos de los repositorios, ya que las nuevas versiones no usan estos generadores como veremos más tarde.

4.2.4. Pipeline de procesamiento, primer mínimo viable

Una vez completadas las etapas de extracción visual de datos, generación de un dataset ajustado a los datos visuales, y entrenamiento de un modelo capaz de clasificar estos datos, el siguiente paso crítico fue la implementación de un pipeline de procesamiento completo. Este pipeline integra todas las etapas desarrolladas para producir un resultado estructurado y útil a partir de una factura de entrada.

Un componente clave de nuestro pipeline es el traductor que convierte la salida del modelo BERT en un formato JSON estructurado. Este traductor es esencial para transformar las etiquetas y tokens clasificados en datos utilizables por sistemas de gestión empresarial.

La función principal de este traductor se implementó de la siguiente manera:

```
def convert_words_and_labels_into_json(labels, tokens):
    result = {}
    result['issuer'] = {}
    result['recipient'] = {}
    result['items'] = []
    result['taxes'] = []

    current_item = {}
    current_tax = {}

    for j in range(len(labels)):
        if labels[j].startswith('B-issuer'):
            issuer_key = parse_issuer(tokens[j],labels[j])
            if issuer_key:
```

```
15                 result['issuer'][issuer_key] = tokens[j]
16         elif labels[j].startswith('B-recipient'):
17             recipient_key = parse_recipient(tokens[j],labels[j])
18             if recipient_key:
19                 result['recipient'][recipient_key] = tokens[j]
20         elif labels[j].startswith('B-item'):
21             item_key = parse_item(tokens[j],labels[j])
22             if item_key:
23                 if item_key == 'description' and current_item:
24                     result['items'].append(current_item)
25                     current_item = {}
26                 current_item[item_key] = tokens[j]
27         elif labels[j].startswith('B-tax'):
28             tax_key = parse_tax(tokens[j],labels[j])
29             if tax_key:
30                 if tax_key == 'description' and current_tax:
31                     result['taxes'].append(current_tax)
32                     current_tax = {}
33                 current_tax[tax_key] = tokens[j]
34         elif labels[j].startswith('B-'):
35             generic_key = parse_generic(tokens[j], labels[j])
36             if generic_key:
37                 result[generic_key] = tokens[j]
38
39     if current_item:
40         result['items'].append(current_item)
41     if current_tax:
42         result['taxes'].append(current_tax)
43
44     return result
```

Algoritmo 4.20: Función de conversión de etiquetas y tokens a JSON

Esta función recorre las etiquetas y tokens generados por BERT, utilizando funciones auxiliares de parsing para determinar la estructura correcta del JSON resultante. Estas funciones auxiliares (`parse_generic`, `parse_issuer`, `parse_recipient`, `parse_item`, y `parse_tax`) se encargan de interpretar las etiquetas específicas y mapearlas a las claves correspondientes en el JSON.

El pipeline completo de procesamiento se implementó en una función que integra todos los componentes desarrollados:

```
1  def process_pdf(pdf_path):
2      image = pdf.pdf_to_img(pdf_path)  # Convierte el pdf a imagen
3      text = ocr.ocr(image)  # Extrae los cuadros de texto
4
5      tokens, labels = ai.predict(text)  # Clasifica el texto
6
7      current_tokens, current_labels = tr.unify_tokens(tokens, labels)  # Unifica los tokens partidos
8      current_tokens, current_labels = tr.merge_same_labels(current_tokens, current_labels)  # Une las
       palabras que componen un concepto semántico
9
10     json_result = jp.convert_words_and_labels_into_json(current_labels, current_tokens)  # Traduce a \GLS{
       JSON} la salida
11
12     json_result = \GLS{JSON}.dumps(json_result, indent=4)  # Formatea el \GLS{JSON} para mejor legibilidad
13     return json_result  # Devuelve el resultado como string
```

Algoritmo 4.21: Función del pipeline de procesamiento completo

Este pipeline realiza las siguientes operaciones en secuencia:

1. Convierte el PDF de entrada en una imagen.
2. Aplica OCR a la imagen para extraer el texto.
3. Utiliza el modelo BERT para clasificar el texto extraído.
4. Unifica los tokens que pudieron haber sido divididos durante el proceso de OCR.
5. Fusiona las etiquetas contiguas que pertenecen al mismo concepto semántico.
6. Convierte las etiquetas y tokens procesados en una estructura JSON.
7. Formatea el JSON para mejorar su legibilidad.

Visualización del Pipeline

Para ilustrar mejor el flujo de procesamiento, se presenta a continuación un diagrama que muestra las etapas principales del pipeline:

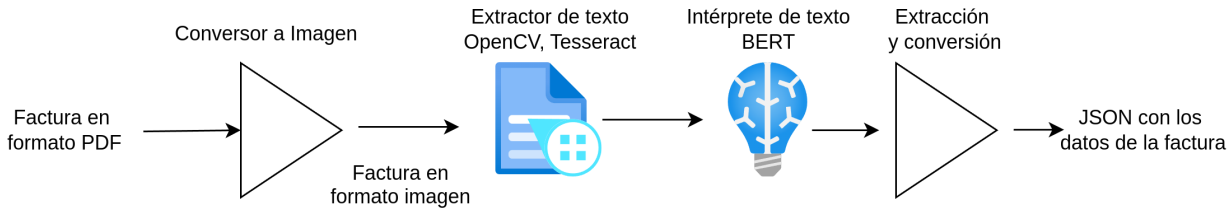


Ilustración 4.6: Diagrama del pipeline de procesamiento

Este diagrama proporciona una visión general de cómo los diferentes componentes del sistema interactúan para procesar una factura desde su formato PDF original hasta un JSON estructurado con la información extraída y clasificada.

La implementación de este pipeline marca un hito significativo en nuestro proyecto, ya que representa un sistema funcional capaz de procesar facturas de principio a fin. Este primer mínimo viable nos permite evaluar la eficacia de nuestro enfoque y proporciona una base sólida para futuras mejoras y optimizaciones.

4.2.5. Resultados

En esta iteración, se logró implementar un producto mínimo viable, aunque con numerosas restricciones. Una de ellas, por ejemplo, es que el modelo está adaptado únicamente al tipo de factura que creamos, es decir, a esa primera plantilla. Por lo tanto, en la segunda iteración, el objetivo es desarrollar diversas plantillas para generalizar la solución una vez observada su viabilidad.

Aunque la clasificación es precisa, la extracción de características visuales y la interpretación del texto están fuertemente acopladas, ya que el conjunto de datos se generó en función de esa salida. Además, los filtros de OpenCV solo funcionan para este modelo de facturas. Por lo tanto, en la siguiente iteración, implementaremos una solución a este problema, tanto para la interpretación de características visuales como para la interpretación de texto, ya que no todos los modelos de facturas necesariamente tienen el mismo formato.

4.3. Tercera iteración

En esta tercera y última iteración desarrollaremos nuevos modelos de facturas, adaptaremos la extracción de datos visuales a los nuevos modelos, generaremos los datos BIO de una manera alternativa para manejar los nuevos modelos, implementaremos el web service que permita desplegar este sistema además de empaquetarlo en un Docker para facilitar su despliegue y la gestión de dependencias y desarrollaremos una interfaz de usuario que nos permita facilmente iteractuar con la herramienta

4.3.1. Nuevos modelos de factura

Para lograr una generalización efectiva del modelo, es fundamental que este sea capaz de interpretar no solo un tipo de factura, sino un conjunto diverso de formatos. Con este objetivo, hemos desarrollado una variedad de modelos de facturas, que se detallan a continuación: VER ANEXO

4.3.2. Implementación de un nuevo sistema OCR

La introducción de nuevos modelos de factura, caracterizados por una mayor complejidad visual, ha requerido una revisión de nuestro enfoque en la extracción de texto. Los filtros previamente utilizados con OpenCV resultaron insuficientes para abordar esta nueva diversidad de diseños. En respuesta a este desafío, hemos implementado EasyOCR, una solución más avanzada y versátil.

EasyOCR incorpora modelos de detección de bounding boxes de texto, lo que añade una capa de inteligencia al proceso de extracción. Además, utiliza modelos de OCR más sofisticados en comparación con TesseractOCR, nuestro sistema anterior. Esta transición nos ha permitido desarrollar un sistema de extracción de texto significativamente más robusto y generalizable, capaz de adaptarse a diversos formatos de factura sin depender de variables tan específicas como el contraste entre el fondo y la tipografía.

A continuación, se presenta la implementación del nuevo sistema OCR utilizando EasyOCR:

```
1  import easyocr
2
3  reader = easyocr.Reader(['en'], gpu=True)
4
5  def ocr(image):
6      ## Image from opencv to PIL
7      result = reader.readtext(image, paragraph=True,
8                               text_threshold=0.3, low_text=0.3, y_ths=0.4)
9
10     output = ''
11     for (box, text) in result:
12         output += text + ' '
13
14     return output
```

Algoritmo 4.22: Implementación de OCR con EasyOCR

Como se puede apreciar, EasyOCR ofrece un nivel de abstracción considerablemente mayor, simplificando notablemente el proceso de extracción de texto. Esta implementación no solo mejora la precisión de la extracción, sino que también aumenta la flexibilidad del sistema para manejar una amplia gama de diseños de facturas.

4.3.3. Adaptación del dataset

La evolución en el formato de las facturas ha requerido una revisión del enfoque para la generación del dataset destinado a BERT. Manteniendo el concepto inicial de generación automática de datos en formato BIO, hemos implementado una metodología más práctica y eficiente.

El nuevo proceso se basa en la sustitución de datos en plantillas predefinidas, utilizando la información extraída por EasyOCR de las facturas. Esta aproximación ha demostrado ser más efectiva y adaptable a la diversidad de los nuevos modelos de factura. A continuación, se presenta un ejemplo de la implementación de este nuevo método de generación de datos:

```
def schema_structure5(data):
    json_data = \GLS{JSON}.loads(data)

    text = textwrap.dedent(f"""
    FACTURA -> O
    Datos -> O
    del -> O
    cliente -> O
    : -> O
    CIF -> O
    : -> O
    {json_data['recipient']['tax_id']} -> B-recipient_tax_id
    Telf -> O
    : -> O
    {json_data['recipient']['phone']} -> B-recipient_phone
    Nombre -> O
    : -> O
    {format_tokens(json_data['recipient']['name'], 'recipient_name')}
    Correo -> O
    : -> O
    {json_data['recipient']['email']} -> B-recipient_email
    {format_tokens(json_data['recipient']['address'].strip(), 'recipient_address')}
    No -> O
    FACTURA -> O
    : -> O
    {json_data['invoice_id']} -> B-invoice_id
    FECHA -> O
    : -> O
    {json_data['issue_date']} -> B-issue_date
    Descripcion -> O
    Cantidad -> O
    TOTAL -> O
    """).strip()

    # Código adicional para ítems, subtotal, impuestos, total y datos del emisor...
```

```
36
37     return text
```

Algoritmo 4.23: Generación de datos BIO para nuevos modelos de factura

Aunque esta implementación puede parecer más compleja, ha demostrado ser altamente efectiva y se adapta mejor a nuestras limitaciones temporales. La sustitución de datos en plantillas nos permite generar datasets precisos y consistentes para cada uno de los nuevos modelos de factura.

Es importante destacar que se han desarrollado un total de seis esquemas diferentes, correspondientes a las distintas disposiciones presentes en los nueve modelos de facturas implementados. Algunos modelos comparten esquemas similares o idénticos, lo que optimiza el proceso de generación de datos.

Esta nueva metodología ha resultado en una mejora significativa en el rendimiento de BERT, que ahora presenta resultados excepcionales para todos los modelos de factura implementados, demostrando la eficacia de este enfoque adaptativo.

4.3.4. Implementación de facturas enriquecidas

Durante el desarrollo del proyecto, surgió un concepto innovador: las facturas enriquecidas. Esta idea consiste en almacenar los datos extraídos en los metadatos del PDF, lo que permite crear una base de datos de documentos con información fácilmente accesible mediante herramientas como Acrobat Reader.

Este enfoque abre nuevas posibilidades para casos de uso específicos, como por ejemplo, permitir que un usuario, al recibir un PDF, pueda visualizar instantáneamente los datos relevantes, facilitando su inserción en sistemas o su utilización inmediata.

La implementación de esta funcionalidad se ha realizado introduciendo el JSON extraído en los metadatos del PDF bajo la etiqueta *AVALON [13]*, nombre asignado a nuestra aplicación final. A continuación, se presenta el código que permite tanto el enriquecimiento como la extracción de estos datos:

```
1  import PyPDF2
2
3  def add_metadata(text_input, pdf_route):
4      """
5      Agrega un metadato personalizado al PDF especificado.
6
7      :param text_input: Texto que se usará como metadato.
8      :param pdf_route: Ruta al PDF a enriquecer.
9      """
10     with open(pdf_route, 'rb') as file:
11         reader = PyPDF2.PdfReader(file)
12         writer = PyPDF2.PdfWriter()
13
14         # Copia todas las páginas del PDF original al nuevo objeto PdfWriter
15         for page_num in range(len(reader.pages)):
16             page = reader.pages[page_num]
```

```
            writer.add_page(page)

        # Añade metadatos personalizados
        metadatos = {
            '/AVALON \cite{avalon2024}': text_input,
        }

        writer.add_metadata(metadatos)

        # Guarda el nuevo PDF con los metadatos añadidos
        with open(pdf_route, 'wb') as new_file:
            writer.write(new_file)

def check_for_AVALON \cite{avalon2024}_sign(pdf_route):
    """
    Verifica si el PDF contiene la firma AVALON \cite{avalon2024}.

    :param pdf_route: Ruta al PDF a verificar.
    :return: Boolean indicando la presencia de la firma AVALON \cite{avalon2024}.
    """
    with open(pdf_route, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        metadatos = reader.metadata

    return '/AVALON \cite{avalon2024}' in metadatos


def extract_AVALON \cite{avalon2024}_sign(pdf_route):
    """
    Extrae los datos de la firma AVALON \cite{avalon2024} del PDF.

    :param pdf_route: Ruta al PDF del cual extraer los datos.
    :return: Contenido de la firma AVALON \cite{avalon2024}.
    """
    with open(pdf_route, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        metadatos = reader.metadata

    return metadatos['/AVALON \cite{avalon2024}']
```

Algoritmo 4.24: Implementación de facturas enriquecidas

Esta implementación proporciona una serie de funciones que permiten enriquecer los PDFs añadiendo la firma AVALON [13] con los datos pertinentes, verificar la presencia de estos datos, y extraerlos cuando sea necesario. Además, se incluye una función para comprobar si un PDF ya ha sido procesado, evitando así duplicidades en el procesamiento.

El concepto de facturas enriquecidas no solo mejora la usabilidad de los documentos procesados, sino que también abre la puerta a nuevas formas de interacción y análisis de datos en facturas digitales.

4.3.5. Implementación del Web Service

Para la implementación de nuestro web service, optamos por utilizar Flask, un framework de Python conocido por su simplicidad y flexibilidad. Esta elección se basó en la naturaleza relativamente sencilla de nuestros requerimientos, que se limitaban a la implementación de dos endpoints principales.

Flask, siendo un framework no opinionado, nos proporcionó la libertad necesaria para crear una API eficiente y adaptada a nuestras necesidades específicas. Los dos endpoints implementados son:

1. Un endpoint para devolver los tokens extraídos en formato JSON.
2. Otro endpoint para devolver los PDFs enriquecidos con metadatos.

A continuación, se presenta la implementación detallada de estos endpoints:

```
@app.route("/pdf_upload/json", methods=["POST"])
def pdf_upload_json():
    if 'files' not in request.files:
        return jsonify({"error": "No file part"}), 400

    files = request.files.getlist('files')

    pdf_routes = save_pdf(files)
    responses = []
    for pdf_route in pdf_routes:
        pdf_path = pdf_route["path"]
        pdf_name = pdf_route["filename"]
        pdf_response = \GLS{JSON}.loads(pp.process_pdf(pdf_path))
        pdf_response["pdf_name"] = pdf_name
        responses.append(pdf_response)

    return jsonify(responses)

@app.route("/pdf_upload/signed_pdf", methods=["POST"])
def pdf_upload_signed_pdf():
    if 'files' not in request.files:
        return jsonify({"error": "No file part"}), 400

    files = request.files.getlist('files')

    pdf_routes = save_pdf(files)
    response = ""
    result = []

    for pdf_route in pdf_routes:
        pdf_path = pdf_route["path"]
        response = pp.process_pdf(pdf_path)
        ps.add_metadata(response, pdf_path)
        result.append(pdf_path)

    zip_buffer = io.BytesIO()
    with zipfile.ZipFile(zip_buffer, 'a', zipfile.ZIP_DEFLATED, False) as zip_file:
        for pdf_path in result:
            zip_file.write(pdf_path, os.path.basename(pdf_path))
    zip_buffer.seek(0)

    return send_file(zip_buffer, mimetype='application/zip', as_attachment=True, download_name='
    signed_pdfs.zip')
```

Algoritmo 4.25: Implementación de endpoints con Flask

Características clave de la implementación:

✔ **Manejo de múltiples archivos:** Ambos endpoints están diseñados para procesar uno o varios PDFs simultáneamente, lo que aumenta la eficiencia del sistema.

- ✔ **Procesamiento JSON:** El endpoint `/pdf_upload/json` extrae la información de los PDFs y la devuelve en formato JSON, facilitando su integración con otros sistemas.
- ✔ **Enriquecimiento de PDFs:** El endpoint `/pdf_upload/signed_pdf` no solo procesa los PDFs, sino que también los enriquece con metadatos utilizando la función `add_metadata`.
- ✔ **Compresión de resultados:** Para el caso de múltiples PDFs enriquecidos, el sistema los comprime en un archivo ZIP antes de enviarlos, optimizando así la transferencia de datos.
- ✔ **Manejo de errores:** Se implementa una verificación básica para asegurar que se han subido archivos, devolviendo un error 400 en caso contrario.

Esta implementación proporciona una interfaz robusta y flexible para interactuar con nuestro sistema de procesamiento de facturas. La capacidad de manejar múltiples archivos y devolver resultados en formatos útiles (JSON para datos extraídos, PDFs enriquecidos para visualización) hace que el sistema sea versátil y adaptable a diferentes necesidades de integración y uso.

4.3.6. Interfaz de usuario

La interfaz de usuario se ha implementado con React, TypeScript, CSS y HTML, usando algunos componentes de la librería de Material UI y consiste en lo siguiente, una página que permite interactuar con la API de forma amigable, sin necesidad de implementar ninguna herramienta que consuma la API, ideal para una demostración:

Página principal

La página principal es el punto de inicio de la interfaz. Aquí, los usuarios pueden arrastrar y soltar archivos PDF o hacer clic para seleccionar archivos desde su sistema. Esta página está diseñada para ser intuitiva y fácil de usar, facilitando la interacción inicial con la aplicación.

Ilustración 4.7: Página principal

Página de espera

Esta página se muestra mientras la aplicación procesa los archivos cargados por el usuario. Incluye un indicador de carga para informar al usuario que el procesamiento está en curso y que debe esperar. Esto mejora la experiencia del usuario al mantenerlo informado sobre el estado de la operación.

Ilustración 4.8: Página de espera mientras se procesan

Página de descarga

Una vez que el procesamiento de los archivos ha finalizado, esta página permite a los usuarios descargar los resultados. Si no se selecciona la opción de .ºnly JSON", se descarga un archivo en el formato adecuado (por ejemplo, PDF). La página confirma que la descarga ha comenzado, proporcionando una experiencia de usuario fluida y coherente.

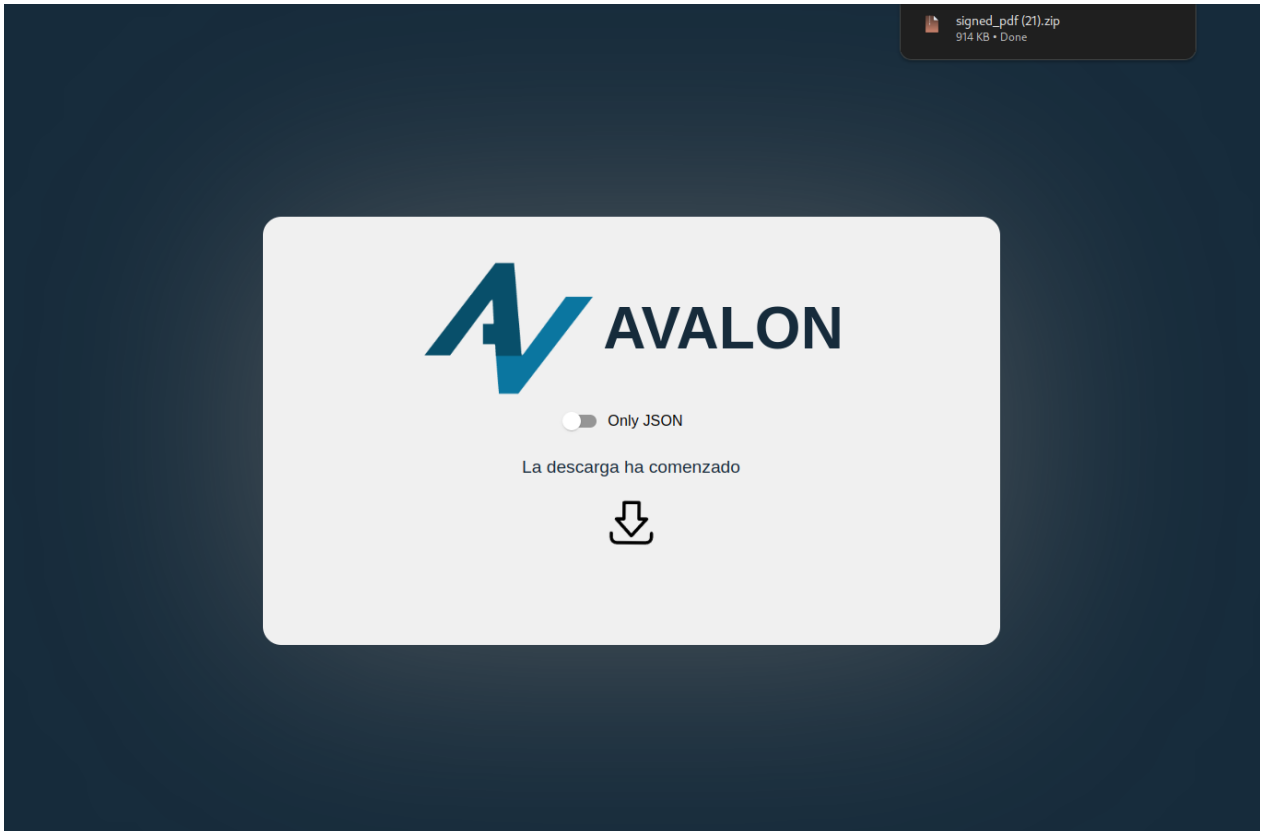


Ilustración 4.9: Página de descarga, si no se selecciona la opción de solo JSON

Página de selección

Si el usuario selecciona la opción .ºnly JSON", se presenta esta página después del procesamiento. Aquí, los usuarios pueden ver una lista de archivos JSON generados a partir de los PDFs procesados, permitiendo una fácil visualización y descarga de los mismos. Esta funcionalidad es útil para usuarios que requieren datos en formato JSON para integraciones o análisis adicionales.

Ilustración 4.10: Página de selección, si se selecciona la opción de solo JSON

Modal con datos de factura sin extender

Este modal muestra una vista resumida de los datos de una factura procesada. Incluye información básica como el ID de la factura y la fecha de emisión. Los usuarios pueden interactuar con esta ventana modal para ver más detalles, proporcionando una vista rápida y accesible de la información clave.

Ilustración 4.11: Modal con los datos de las facturas, sin extender

Modal con datos de facturas extendido

Este modal muestra una vista detallada y extendida de los datos de una factura procesada. Incluye información completa del emisor, destinatario, artículos, impuestos y resumen de la factura. Esta vista detallada es útil para usuarios que necesitan verificar o analizar la información específica de cada factura de manera exhaustiva.

Ilustración 4.12: Modal con los datos de las facturas, extendido

4.3.7. Contenedorización con Docker

Dado que el proyecto es ciertamente complicado y no ejecutable con librerías nativas, se ha optado por empaquetar todo el sistema en un contenedor Docker. Esto permite gestionar las dependencias y la ejecución del servicio, simplificando el proceso a entrar en el directorio del proyecto y ejecutar el comando ***docker compose up***

La imagen Docker utiliza Python versión 3.12, que es la versión en la que se ha desarrollado todo el proyecto:

```
FROM python:3.12.3-slim

WORKDIR /app

COPY requirements.txt .

RUN apt-get update && apt-get install -y \
    build-essential \
    python3-dev \
    && rm -rf /var/lib/apt/lists/*

RUN apt-get update && apt-get install -y poppler-utils

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

Algoritmo 4.26: Dockerfile

El archivo ‘docker-compose.yml‘ se encarga de ejecutar el contenedor con soporte para tarjeta gráfica Nvidia, si está disponible:

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/app
    environment:
      FLASK_APP: app.py
      FLASK_ENV: development
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: 1
              capabilities: [gpu]
```

Algoritmo 4.27: docker-compose.yml

Capítulo 5

Conclusiones

El trabajo concluye con AVALON [13], un sistema que permite la extracción automatizada de datos de facturas, acompañado de una guía detallada sobre su desarrollo, plasmada en esta memoria. Además, AVALON [13] sirve como punto de entrada para la comunidad de software interesada en desarrollar sistemas similares.

En los siguientes apartados, destacaremos el estado actual del proyecto, su rendimiento y resultados; el trabajo futuro para mejorar la precisión en la extracción de datos; las conclusiones y el impacto personal de la realización de este trabajo, así como las aportaciones realizadas.

5.1. Resultados

Esta sección recapitula lo mencionado en las iteraciones anteriores, resaltando los hitos clave y el estado actual del proyecto. A lo largo de tres iteraciones fundamentales, se han establecido las bases de un sistema robusto y eficiente para la extracción automatizada de datos de facturas, denominado AVALON [13].

La primera iteración se centró en una exhaustiva fase de investigación y en la generación de datos sintéticos, estableciendo un conjunto de datos inicial y un modelo preliminar de facturas. Se exploraron herramientas y tecnologías clave, y se generaron plantillas de facturas en HTML y CSS para crear un conjunto de datos diverso.

En la segunda iteración, se desarrolló un Producto Mínimo Viable (MVP) que incluía la implementación de un prototipo funcional para la extracción de datos. Este MVP permitió validar conceptos y ajustar el enfoque técnico del proyecto, utilizando tecnologías como OpenCV y Tesseract OCR, además del entrenamiento del modelo BERT para la clasificación de datos.

La tercera iteración ha sido crucial para la generalización y optimización del sistema. Se desarrollaron múltiples modelos de facturas y se mejoró significativamente el proceso de extracción de datos visuales mediante la adopción de EasyOCR, una herramienta más

avanzada y adaptable. Además, se implementó un servicio web utilizando Flask y una interfaz de usuario basada en React, lo cual ha facilitado la interacción con AVALON [13] y permitido una demostración práctica y efectiva de sus capacidades.

El rendimiento del sistema AVALON [13] ha mejorado notablemente a lo largo de las iteraciones, especialmente en la última, donde se ha enfocado en la generalización y optimización. La implementación de EasyOCR ha permitido una extracción de texto más precisa y adaptable a diversos modelos de facturas, superando las limitaciones de los métodos anteriores basados en OpenCV. Sin embargo, es importante destacar que AVALON [13] sigue dependiendo principalmente del texto extraído, sin considerar otras características de la factura, lo que podría limitar su capacidad para discriminar datos en formatos más complejos o variados.

El modelo BERT, entrenado con datos BIO generados específicamente para cada tipo de factura, ha demostrado una alta precisión en la clasificación de elementos, logrando resultados sobresalientes con pocas iteraciones de entrenamiento. A pesar de estos logros, el enfoque actual de AVALON [13], basado únicamente en texto, puede no ser suficiente para manejar todas las variaciones posibles en las facturas.

La pipeline de procesamiento completo de AVALON [13], que integra la extracción de texto, la clasificación y la conversión a formato JSON, ha sido optimizada para manejar facturas de diferentes formatos de manera eficiente. La introducción de facturas enriquecidas, con datos almacenados en los metadatos del PDF, ha mejorado la usabilidad y la accesibilidad de la información extraída. No obstante, la usabilidad se verá realmente mejorada una vez se desarrolle una herramienta específica que facilite la interacción con estos metadatos.

Estas iteraciones han culminado en un sistema capaz no solo de procesar y extraer datos de facturas con alta precisión, sino también de manejar una variedad de formatos y adaptarse a nuevas necesidades y requisitos. El proyecto se encuentra en una fase avanzada de desarrollo, con AVALON [13] siendo un sistema robusto, eficiente y listo para su implementación práctica.

Aunque AVALON [13] ha mostrado mejoras significativas, existen limitaciones inherentes a la arquitectura actual que depende únicamente del texto. Abordar estas limitaciones será crucial para mejorar la capacidad del sistema para manejar una variedad más amplia de formatos y características de facturas.

5.2. Contribuciones

Las principales contribuciones del proyecto incluyen el desarrollo de AVALON [13], un sistema robusto y adaptable de extracción de datos visuales, capaz de manejar múltiples modelos de facturas. La implementación de un modelo BERT altamente preciso para la clasificación de elementos de facturas ha sido un avance significativo, permitiendo una extracción eficiente y estructurada de información.

El desarrollo de un servicio web con Flask y una interfaz de usuario intuitiva basada

en React ha facilitado la interacción con AVALON [13], proporcionando una herramienta práctica y accesible para los usuarios. Además, la introducción del concepto de facturas enriquecidas ha mejorado la usabilidad de los documentos digitales, permitiendo una gestión más eficiente de la información extraída.

El proyecto será lanzado como open source, con planes de seguir mejorándolo y documentándolo mejor. Se prevé incluso que AVALON [13] pueda convertirse en un Trabajo de Fin de Máster, lo que subraya su relevancia académica y práctica. La importancia de contar con AVALON [13] empaquetado en Docker no puede subestimarse, ya que permite una implementación sencilla para desarrolladores sin conocimientos específicos en inteligencia artificial.

El impacto potencial de AVALON [13] en aplicaciones de gestión financiera o fiscal es enorme, tanto para individuos como para empresas. La democratización de estos sistemas permitirá mejorar la eficiencia económica de aquellos que los utilicen, ofreciendo una herramienta vital que contribuye a la mejora económica de los usuarios.

Estas contribuciones han sentado las bases para futuras mejoras y expansiones de AVALON [13], abriendo nuevas posibilidades para su aplicación en diversas áreas y sectores, y representando una contribución significativa y accesible al ámbito de la tecnología y la gestión financiera.

5.3. Aprendizaje

El proyecto ha demostrado ser viable y efectivo en la extracción automatizada de datos de facturas. A través de iteraciones sucesivas, se han abordado y superado diversos desafíos técnicos, logrando una implementación robusta y eficiente. La combinación de herramientas avanzadas como EasyOCR y BERT ha permitido desarrollar un sistema capaz de extraer y estructurar información de manera precisa y rápida.

La implementación de un servicio web y una interfaz de usuario intuitiva ha facilitado la demostración y el uso práctico de AVALON [13], permitiendo una interacción fluida y eficiente con los usuarios. Además, la introducción de facturas enriquecidas ha mejorado significativamente la usabilidad de los documentos digitales, facilitando su manejo y análisis.

La participación en este proyecto ha tenido un impacto significativo tanto a nivel personal como profesional. Ha permitido adquirir y profundizar conocimientos en áreas clave como el procesamiento de imágenes, el reconocimiento óptico de caracteres (OCR) y el procesamiento de lenguaje natural (NLP). La implementación de un sistema completo, desde la extracción de datos hasta la creación de un servicio web y una interfaz de usuario, ha sido una experiencia enriquecedora que ha contribuido al desarrollo de habilidades técnicas y de gestión de proyectos.

Además, la resolución de desafíos técnicos complejos y la colaboración en equipo han fomentado el crecimiento personal y profesional, mejorando la capacidad de abordar problemas de manera creativa y efectiva. Este proyecto no solo ha potenciado mis habilidades como desarrollador e ingeniero, sino que también me ha capacitado para crear un producto a partir

de una investigación propia. Este afán de conocimiento ha culminado en un proyecto útil, que ha cimentado en mí una seguridad y un sentimiento de profesionalidad que no había obtenido durante mi etapa de estudiante.

Este proyecto ha sido una experiencia transformadora, permitiéndome crecer como profesional y persona, y brindándome la confianza para enfrentar futuros desafíos con una perspectiva innovadora y segura, consolidando mi camino hacia la excelencia en el desarrollo de soluciones tecnológicas.

5.3.1. Trabajo futuro

El trabajo futuro se centrará en la ampliación de AVALON [13] para admitir una mayor variedad de formatos de facturas y otros documentos financieros. Esto incluirá la incorporación de nuevas plantillas y la adaptación del sistema de extracción de datos para manejar esta diversidad. También se prevé mejorar la precisión del modelo BERT mediante el entrenamiento con conjuntos de datos más amplios y variados, incluyendo datos reales obtenidos de diferentes fuentes.

Un aspecto crucial del desarrollo futuro será el cambio de la arquitectura actual de AVALON [13] a una que aproveche más información de la factura, como la posición de los cuadros de texto y la relación entre ellos. Esto permitirá una discriminación más precisa de los datos y mejorará la capacidad del sistema para manejar formatos más complejos. Además, se planea la implementación de un modelo más óptimo como ALBERT, que permite una ejecución más rápida y eficiente, incluso en equipos menos potentes, facilitando su despliegue en una variedad más amplia de entornos.

Otra área de desarrollo será la integración de tecnologías avanzadas de procesamiento de lenguaje natural y aprendizaje profundo para optimizar aún más la extracción y clasificación de datos. Además, se explorarán nuevas funcionalidades, como la integración con sistemas de gestión empresarial y plataformas de análisis de datos, para ampliar las aplicaciones prácticas del sistema AVALON [13].

El trabajo futuro se enfocará en mejorar la generalización de AVALON [13], incorporar una arquitectura más sofisticada que utilice información posicional y relacional, e implementar modelos más eficientes que permitan un desempeño superior en una gama más amplia de dispositivos y entornos.

Bibliografía

[1] ABBYY (2024). Abbyy flexicapture. https://www.abbyy.com/flexicapture/.

[2] Docparser (2024). Docparser. https://www.docparser.com/.

[3] Feng, W., Guan, N., Li, Y., Zhang, X., and Luo, Z. (2017). Audio visual speech recognition with multimodal recurrent neural networks. In *2017 International Joint Conference on neural networks (IJCNN)*, pages 681–688. IEEE.

[4] Google (2024). Google cloud document ai. https://cloud.google.com/document-ai?hl=es. Accessed: 2024-06-30.

[5] Hypatos (2024). Hypatos. https://www.hypatos.ai/.

[6] Kofax (2024). Kofax. https://www.kofax.com/.

[7] Li, Y., Jiang, W., and Song, S. (2023). Review of semi-structured document information extraction techniques based on deep learning. In *2023 2nd International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*, pages 112–119. IEEE.

[8] Microsoft (2024). Document intelligence invoice model. https://learn.microsoft.com/en-us/azure/ai-services/document-intelligence/concept-invoice?view=doc-intel-4.0.0. Accessed: 2024-06-30.

[9] Nanonets (2024). Nanonets. https://nanonets.com/.

[10] Parascript (2024). Parascript. https://www.parascript.com/.

[11] Rossum (2024). Rossum. https://rossum.ai/.

[12] Services, A. W. (2024). Amazon textract. https://aws.amazon.com/textract/.

[13] Torres Bordón, A. J. (2024). Avalon. https://github.com/OpenInvoiceScan. Software disponible en https://github.com/OpenInvoiceScan.

[14] UiPath (2024). Uipath document understanding. https://www.uipath.com/product/document-understanding.

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[16] Veryfi (2024). Veryfi. https://www.veryfi.com/.

[17] WorkFusion (2024). Workfusion. https://www.workfusion.com/.

Glosario

API REST Interfaz de Programación de Aplicaciones que sigue los principios de la Transferencia de Estado Representacional, permitiendo la comunicación entre sistemas utilizando métodos HTTP estándar. 21

BERT Representaciones de Codificador Bidireccional de Transformers, un modelo de lenguaje desarrollado por Google basado en la arquitectura Transformer, que ha demostrado un rendimiento sobresaliente en diversas tareas de NLP. viii, 2, 13, 24–26, 28–30, 33, 35–37, 48, 49, 60–63

BIO Un esquema de etiquetado para la anotación de secuencias en tareas de procesamiento del lenguaje natural, donde B denota el comienzo de una entidad, I denota la continuación de una entidad, y O denota fuera de cualquier entidad. viii, 24, 25, 39, 48, 49, 61

datasets Conjuntos de datos utilizados para entrenar, evaluar y validar modelos en tareas de aprendizaje automático. 11, 49

datos estructurados Datos organizados en un formato predefinido, como tablas en una base de datos, que facilitan su interpretación y procesamiento por sistemas automatizados. 4, 6, 18, 33

GPUs Unidades de Procesamiento Gráfico, hardware especializado en el procesamiento rápido de gráficos y cálculos paralelos, ampliamente utilizado en aplicaciones de aprendizaje automático y procesamiento de datos. 24

JSON Notación de Objetos de JavaScript, un formato de texto ligero y fácil de leer para el intercambio de datos. viii, 4, 6, 11, 19, 20, 22, 24, 33–37, 50–52, 61

NER Reconocimiento de Entidades Nombradas, una tarea del procesamiento del lenguaje natural (NLP) que involucra la identificación y clasificación de entidades mencionadas en el texto en categorías predefinidas como nombres de personas, organizaciones, ubicaciones, expresiones de tiempo, etc.. 1–3, 24

NLP Procesamiento del Lenguaje Natural, una rama de la inteligencia artificial que se centra en la interacción entre computadoras y el lenguaje humano. 13

OCR Reconocimiento Óptico de Caracteres, una tecnología utilizada para convertir diferentes tipos de documentos, como documentos escaneados en papel, archivos PDF o imágenes capturadas por una cámara digital, en datos editables y buscables. v, viii, 2, 6, 15, 31, 33, 37, 48, 60

RNN Redes Neuronales Recurrentes, un tipo de red neuronal artificial donde las conexiones entre las unidades forman un grafo dirigido a lo largo de una secuencia temporal, permitiendo mantener información de estados anteriores. 7, 22–24

TPUs Unidades de Procesamiento Tensorial, hardware desarrollado por Google específicamente para acelerar las cargas de trabajo de aprendizaje automático. 24

Transformer Una arquitectura de red neuronal que utiliza mecanismos de atención para procesar datos secuenciales, ampliamente utilizada en modelos de procesamiento del lenguaje natural. vi, 1, 7, 13, 23, 24

Anexos

Anexo 1. Modelos de facturas

Modelos de facturas desarrollados para la generalización del modelo, cada una con sus características específicas que supusieron un reto a nivel de visión por computador:

Factura
#216103

GLOBAL FRESH
IMPORT & EXPORT

Fecha de emisión: 2024-06-17
Fecha de vencimiento: 2024-07-28

Datos del Emisor:
Ruth Reguera Macías S.L.L.
Alameda Ani Peña 48 Puerta 7 Guipúzcoa, 32788
+34822 616 384
gilbertolobato@mora.es
SHh622SRc270

Datos del Receptor:
Alex del Santiago
Calle María José Solís 66 Puerta 1 Las Palmas, 01330
+34922 29 17 19
blancomaura@yahoo.com
XPX524zSR471

Descripción	Cantidad	Precio Unitario	Total
syndicate end-to-end architectures	2	74.19	148.38
cultivate 24/7 e-services	9	59.11	531.99
engage impactful niches	3	27.12	81.36
transform ubiquitous e-commerce	8	95.07	760.56

Subtotal: 1522.29
VAT (16%): 243.57
Total: 1765.86

Método de pago: Bank Transfer

Ilustración 5.1: Primer modelo de factura: diseño inicial implementado

FACTURA

Servicios Armengol & Asociados S.C.P
Ronda Maximiano Franco 48
Guipúzcoa, 17524
+34 810449192
molesroberto@anglada.com
VqK011rtf090

Datos del ciente:
Donato Rueda Gomila
Urbanización Matilde Cortés 51 Piso 8 Ávila, 20785
+34 973906406
silvestrevigil@gmail.com
smC008wZg596

Nº FACTURA: 376329
FECHA: 2024-06-10

CANTIDAD	DESCRIPCIÓN	PRECIO UNITARIO	TOTAL
4	repurpose back-end e-commerce	75.04	300.16
10	harness dynamic action-items	40.98	409.80

Subtotal: 709.96
VAT (16%): 113.59
TOTAL: 823.55

Ilustración 5.2: Segundo modelo de factura: variación en la disposición de elementos

Cañellas y Cal S.Coop.
Via Cintia Romeu 1 Puerta 1
Valencia, 14561
+34 856 65 72 08
xrodenas@banco.com
nnd211nvG471

Cliente:
Eugenio Carvajal Vicens
Urbanización de Eleuterio Yáñez 94 Puerta 3 Córdoba, 49716
+34 719625479
ramisadelaida@gmail.com
Lmn387AIW698

Nº FACTURA: 862741
FECHA: 2024-06-04
PAGADO CON: PayPal

CANTIDAD	DESCRIPCIÓN	PRECIO UNITARIO	TOTAL
7	productize virtual e-commerce	13.41	93.87
8	leverage frictionless relationships	62.56	500.48
7	utilize frictionless niches	28.36	198.52

Subtotal: 792.87
VAT (16%): 126.86
TOTAL: 919.73

Ilustración 5.3: Tercer modelo de factura: incorporación de elementos gráficos adicionales

FACTURA

Datos del cliente:

CIF: unv115jIN678
Telf: +34862 10 77 35
Nombre: Valero Badía Fuentes

Correo: ribaantonio@hotmail.com
Pasadizo de Inés Carrillo 9 Puerta 3 Navarra, 01200

Nº FACTURA:754975
FECHA: 2024-06-04

Descripción	Cantidad	TOTAL
enable distributed users	1	99.73
visualize world-class interfaces	5	406.40

Forma de pago:Credit Card

Subtotal: 506.13
VAT (16%): 80.98
TOTAL: 587.11

Datos del Emisor:

CIF: FFc215aen984
Telf: +34 620917030
Nombre: Parra & Asociados S.L.

Correo: gertrudisanton@bueno.es
Pasadizo Flor Lerma 37 Puerta 9 La Coruña, 03101

Ilustración 5.4: Cuarto modelo de factura: énfasis en la presentación de datos del cliente

FACTURA

Datos del cliente:

CIF: iNy466Fww818
Telf: +34973 387 926
Nombre: Cándida Martínez-Vidal

Correo: mazamacarena@gmail.com
Acceso de Selena Cervantes 763 Piso 7 Barcelona, 32120

Nº FACTURA:854599
FECHA: 2024-06-29

Descripción	Cantidad	TOTAL
exploit interactive channels	8	595.04
iterate B2C e-markets	2	140.24
streamline end-to-end info-mediaries	7	173.95

Forma de pago:PayPal

Subtotal: 909.23
VAT (16%): 145.48
TOTAL: 1054.71

Datos del Emisor:

CIF: kRT961tiI718
Telf: +34945 58 92 03
Nombre: Olivares y asociados S.L.L.

Correo: chaconlorenzo@clementina.com
Acceso Isidoro Coloma 70 Zamora, 14184

Ilustración 5.5: Quinto modelo de factura: diseño minimalista con enfoque en la claridad

FACTURA

Distribuciones Alberto S.A.
Cañada Yaiza Martorell 69 Piso 1
Málaga, 04119
+34871 122 520
lupealvarado@llabres.es
myg637WwJ102

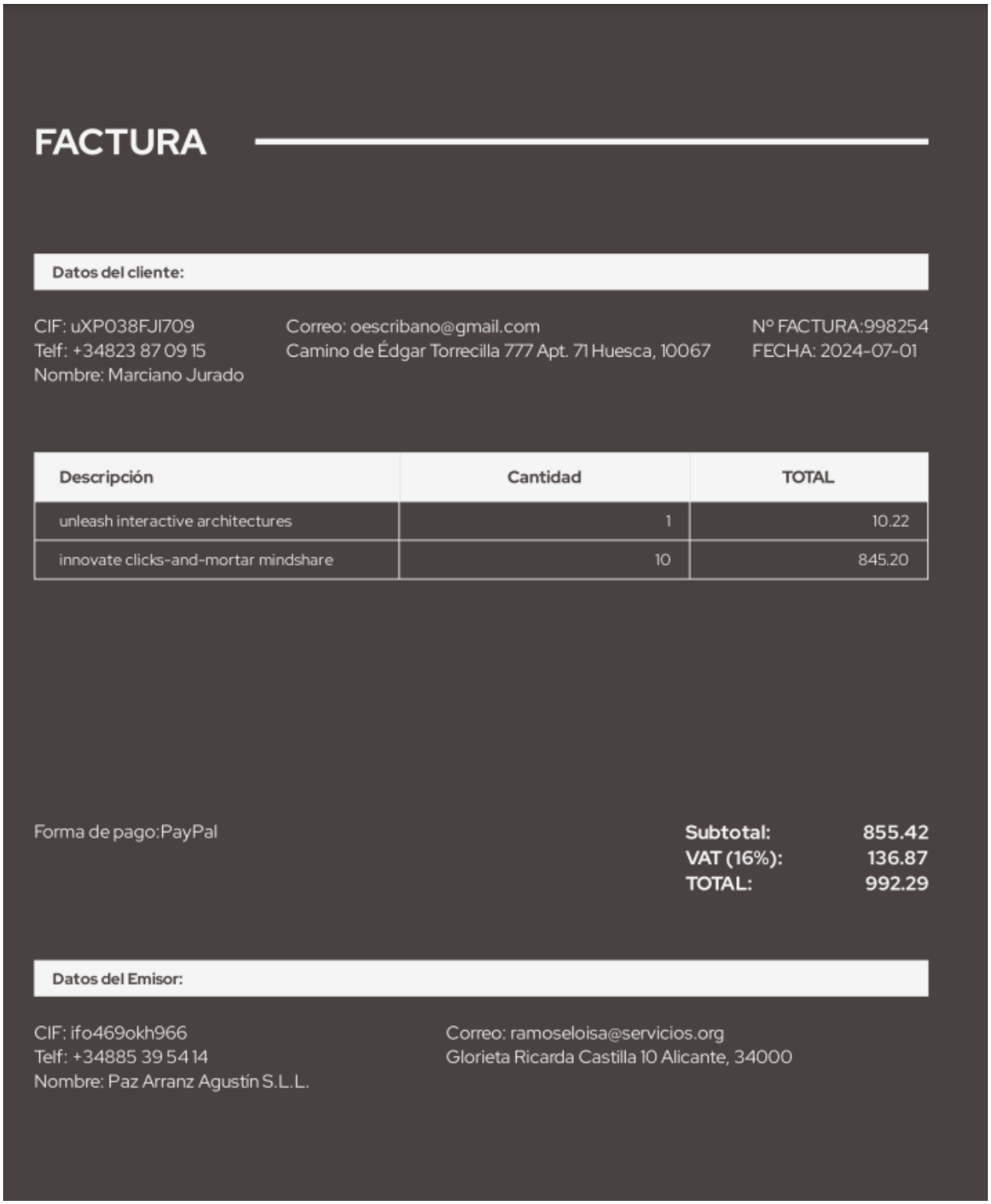
Datos del ciente:
Lino de Pulido
Rambla de Valero Codina 84 Granada, 38799
+34920 52 35 44
riveroramon@gmail.com
kKh207VuB510

Nº FACTURA: 274142
FECHA: 2024-06-06

CANTIDAD	DESCRIPCIÓN	PRECIO UNITARIO	TOTAL
10	aggregate compelling networks	61.04	610.40
9	transform bricks-and-clicks e-commerce	40.17	361.53
9	matrix ubiquitous deliverables	65.07	585.63

Subtotal: 1557.56
VAT (16%): 249.21
TOTAL: 1806.77

Ilustración 5.6: Sexto modelo de factura: integración de elementos visuales distintivos



FACTURA

Datos del cliente:

CIF: uXP038FJI709
Telf: +34823 87 09 15
Nombre: Marciano Jurado

Correo: oescribano@gmail.com
Camino de Édgar Torrecilla 777 Apt. 71 Huesca, 10067

Nº FACTURA:998254
FECHA: 2024-07-01

Descripción	Cantidad	TOTAL
unleash interactive architectures	1	10.22
innovate clicks-and-mortar mindshare	10	845.20

Forma de pago:PayPal

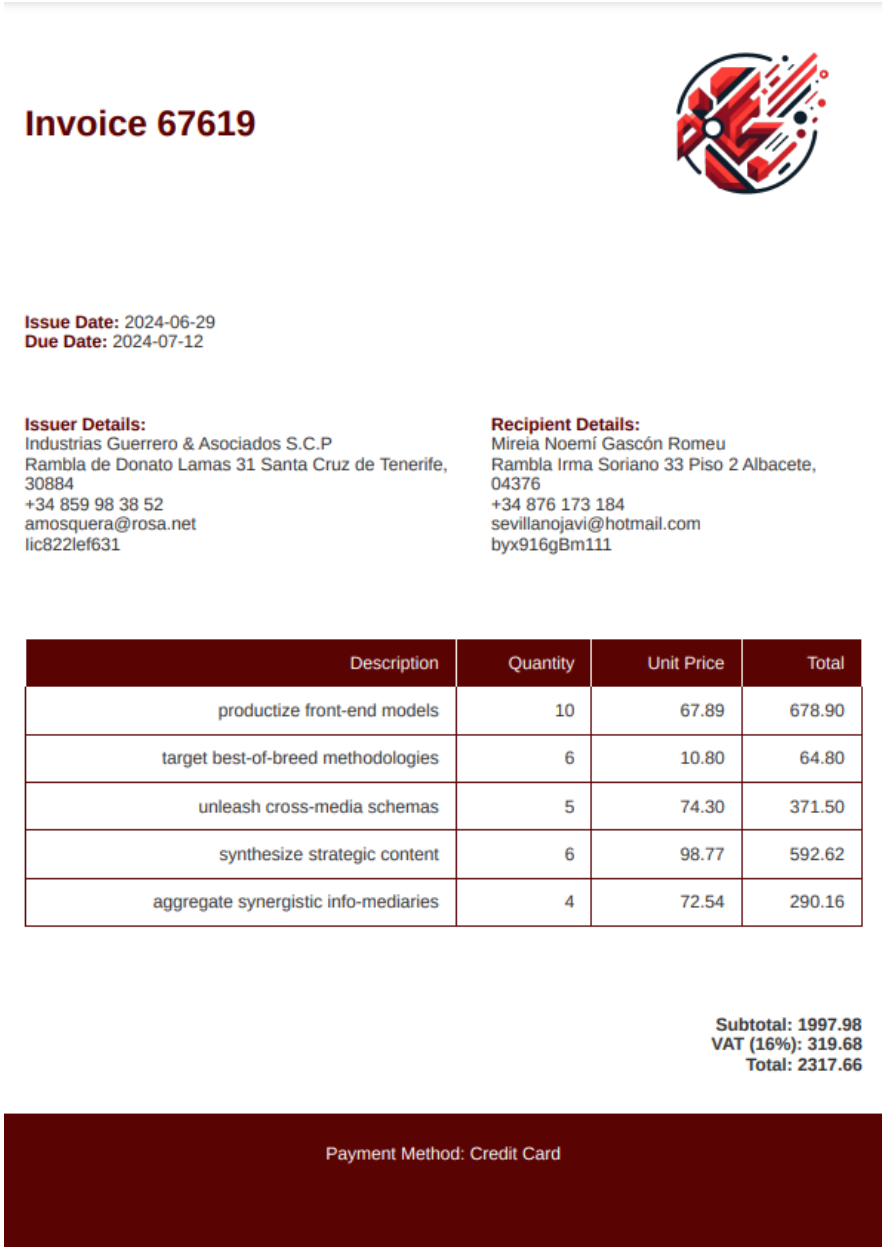
Subtotal: 855.42
VAT (16%): 136.87
TOTAL: 992.29

Datos del Emisor:

CIF: ifo469okh966
Telf: +34885 39 54 14
Nombre: Paz Arranz Agustín S.L.L.

Correo: ramoseloisa@servicios.org
Glorieta Ricarda Castilla 10 Alicante, 34000

Ilustración 5.7: Séptimo modelo de factura: estructura tabular avanzada



Invoice 67619

Issue Date: 2024-06-29
Due Date: 2024-07-12

Issuer Details:
Industrias Guerrero & Asociados S.C.P
Rambla de Donato Lamas 31 Santa Cruz de Tenerife, 30884
+34 859 98 38 52
amosquera@rosa.net
lic822lef631

Recipient Details:
Mireia Noemí Gascón Romeu
Rambla Irma Soriano 33 Piso 2 Albacete, 04376
+34 876 173 184
sevillanojavi@hotmail.com
byx916gBm111

Description	Quantity	Unit Price	Total
productize front-end models	10	67.89	678.90
target best-of-breed methodologies	6	10.80	64.80
unleash cross-media schemas	5	74.30	371.50
synthesize strategic content	6	98.77	592.62
aggregate synergistic info-mediaries	4	72.54	290.16

Subtotal: 1997.98
VAT (16%): 319.68
Total: 2317.66

Payment Method: Credit Card

Ilustración 5.8: Octavo modelo de factura: disposición innovadora de la información

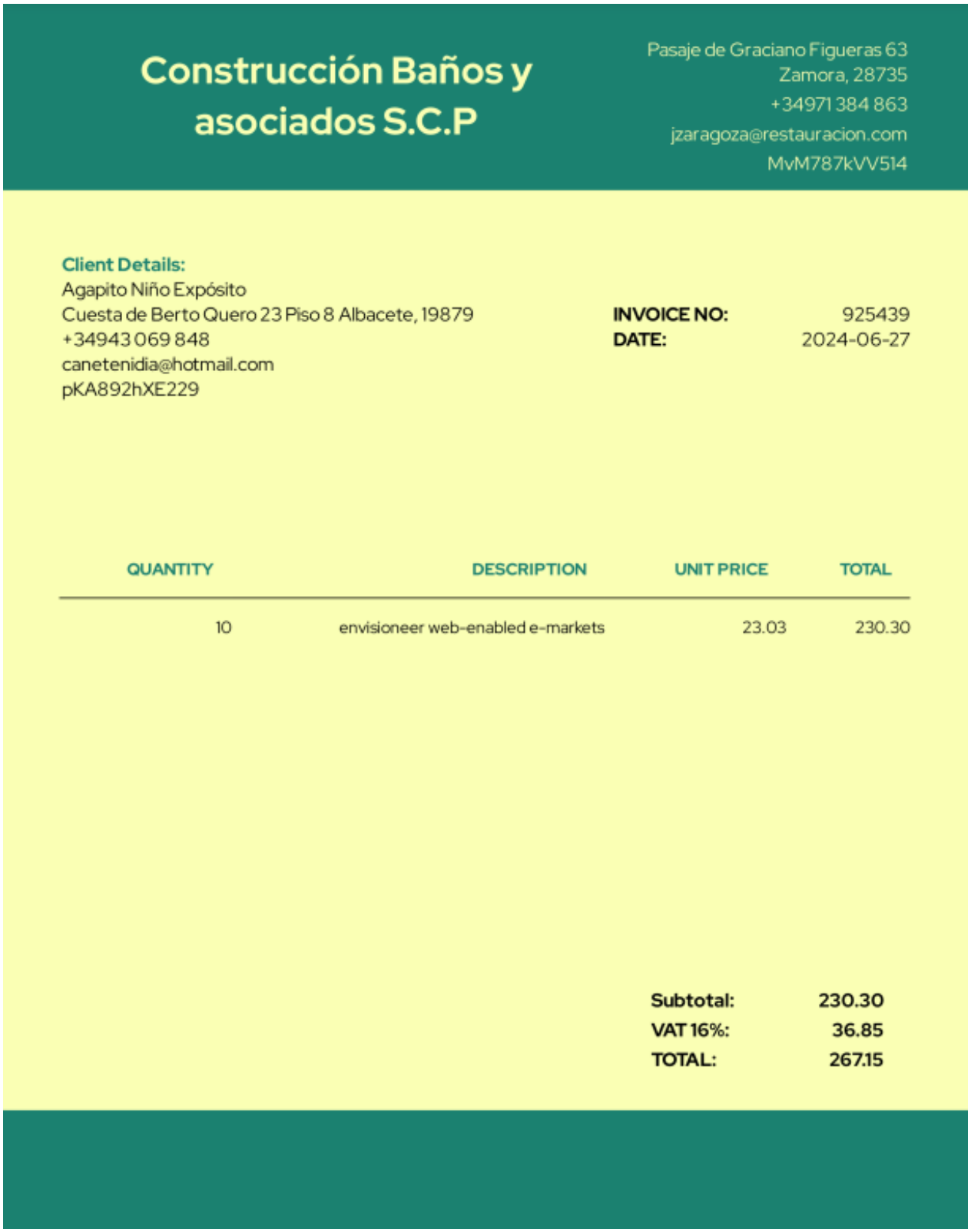


Ilustración 5.9: Noveno modelo de factura: combinación de elementos de diseños previos

Anexo 2. Interfaz de usuario

Anexo con las imagenes relacionadas a la interfaz de usuario, para más detalle en la explicación consultar 4.3.6. Interfaz de usuario

Ilustración 5.10: Página principal

Ilustración 5.11: Página de espera mientras se procesan

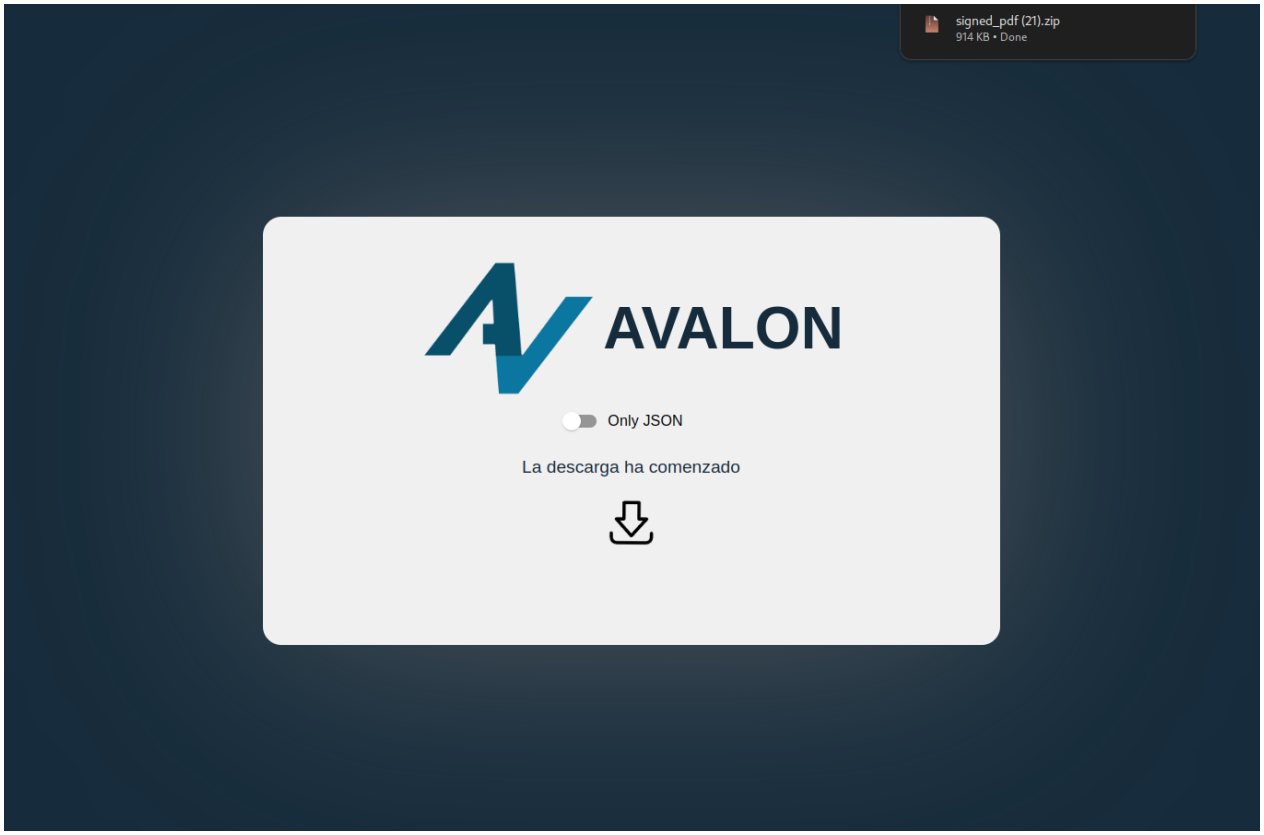


Ilustración 5.12: Página de descarga, si no se selecciona la opción de solo JSON

Ilustración 5.13: Página de selección, si se selecciona la opción de solo JSON

Ilustración 5.14: Modal con los datos de las facturas, sin extender

Ilustración 5.15: Modal con los datos de las facturas, extendido