CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

DESARROLLO DE UN MICROSERVICIO DE AUTOMATIZACIÓN DE DOCUMENTOS CONTABLES

Autor: Miguel Mayoral Martínez

Director: María Luisa Córdoba Cabeza

MADRID, ENERO 2017

ÍNDICE DE FIGURAS Y TABLAS

1 RESUMEN EN ESPAÑOL

El objetivo principal de este documento es explicar el desarrollo de un analizador de texto de facturas. Este analizador se lleva a cabo mediante un microservicio implementado en Python, que se basa en el uso de expresiones regulares y el análisis de patrones que se observan en un conjunto de pruebas entregado por un cliente.

El programa es capaz de extraer de todo el texto de las facturas, los campos más relevantes para el cliente, como podían ser el precio del artículo, el número de chasis o la fecha de la factura, entre otras.

2 RESUMEN EN INGLÉS

The main goal of this paper is to explain the implementation of a parser for the extracted text of invoices. This parser is developed using a Python microservice, which is based on regular expressions and pattern analysis that could be observed on invoices delivered by a customer.

The program is able to extract the most relevant data from the invoice, such as Total price of an item, chassis number or invoice date, among others.

3 INTRODUCCIÓN

Este trabajo se ha desarrollado en la empresa IECISA, en el departamento de onboarding digital, dentro de identidad y biometría. Un cliente de una empresa de vehículos acude con la necesidad de digitalizar las facturas en PDF que tenían almacenadas y procesar el texto extraído de ellas.

Lo que busca el cliente con este proceso es evitar el paso manual de los datos de las facturas, no teniendo que trasladar la información relevante de éstas a una base de datos de una en una. El microservicio tardará en responder unos segundos por factura, y buscando los datos necesarios manualmente se tardarían varios minutos.

Para ello se hace uso de un programa ya desarrollado en la empresa de reconocimiento óptico de caracteres para extraer el texto, y posteriormente, con un microservicio de análisis del texto se analizará y formateará la información obtenida. El analizador de texto es lo que se explica en este documento.

Un analizador de texto se encarga de comprobar si un texto sigue alguna estructura lógica y en base a eso, realizar una u otra acción. Los analizadores de texto se suelen basar en expresiones regulares para identificar reglas o patrones que les permiten encontrar la información que buscan.

En este caso, se quería obtener del texto de una factura distintos campos que les resultaban necesarios al cliente. De modo que el microservicio explicado en este trabajo contiene un analizador que, una vez ha procesado el texto, identifica los datos buscados y los devuelve en sus campos correspondientes.

4 OBJETIVOS

El objetivo principal de un analizador de texto en programación consiste en convertir una información estructurada de una manera en otra mucho más sencilla y manejable. El analizador debe entender la estructura del texto y suponer que habrá a continuación.

La petición del cliente marca los objetivos de este analizador, que consisten en analizar el texto proveniente de las facturas y procesarlo para extraer los datos pedidos. Esta implementación beneficiará al cliente permitiéndole tener la información de las facturas digitalizada y estructurada, lo que hará que puedan obtener distintas estadísticas de sus clientes, o tenerla almacenada.

También aparecen los objetivos personales de aprender python y familiarizarse con el marco de trabajo de django, ya que este microservicio debía integrarse junto con otros que ya estaban desarrollados de esta manera.

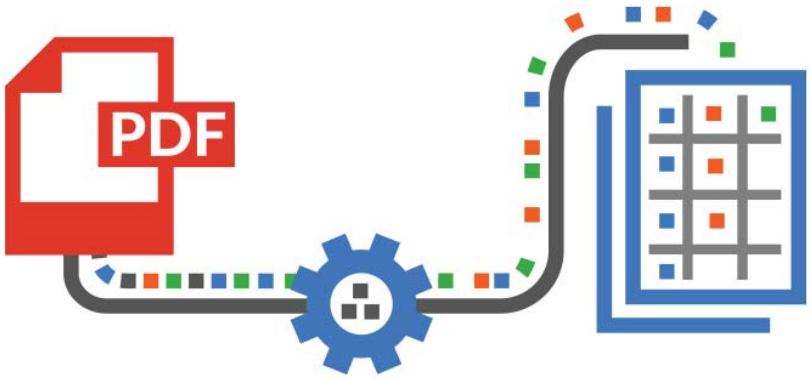

Figura 1. Ilustración de un analizador de texto [1]

5 ENTORNO DEL PROYECTO

5.1 PLANTEAMIENTO DEL PROYECTO

Para extraer la información de los PDFs se podía usar un programa ya desarrollado en la empresa, que consiste en un lector de documentos basado en tesseract, que es un mecanismo de reconocimiento óptico de caracteres (OCR). Este programa desarrollado en Python permite extraer el texto del PDF o de una imagen, y lo devuelve separado en varios bloques de texto o en uno sólo, separado por saltos de línea.

Inicialmente para suplir la necesidad del cliente se pensó en usar procesamiento del lenguaje natural para preprocesar el texto con el marco de trabajo keras [2], ya que se esperaba que las facturas fuesen muy diferentes entre ellas. Y así, facilitar la programación de las reglas para extraer la información. [3]

De modo que, una vez el cliente nos entrego el conjunto de facturas, se comenzó a hacer una base de datos manualmente con el texto etiquetado como se pretendía que la red neuronal debía preprocesarlo, esta base de datos se usaría para entrenar a la red, validarla y ponerla a prueba.

Nombre fichero	Nº documento	Sentencia	Codigo	Precio	Fecha
Nomina1	[redacted]	[redacted]	[redacted]	8.385,7400	21/06/2017
		3/3		8.385,7400	
		TL	21,00 %	-404,3400	
		[redacted]		250,0000	
				21,0000	
				8.231,4000	
				1.728,5940	
				9.959,99	
				9.959,99	
				8.231,40	
		FACTURA Nº:		1.728,59	
		Fecha:		9959,99	
		[redacted]			

Figura 2. Imagen de la base de datos con información del cliente tachada

Finalmente se descartó este planteamiento, debido principalmente al tiempo del que se disponía para entregarlo y que la complejidad que suponía entrenar una red neuronal no era necesaria para extraer los campos pedidos por el cliente para este tipo de facturas.

Se estudió el texto de 40 facturas y se dispuso en un Excel los campos que se querían obtener, incluyendo en algunos casos que palabras les precedían y las posteriores para buscar patrones que se pudiesen aprovechar. Después de este análisis se concluyó que se podía programar un conjunto de reglas para analizar el texto y extraer la información relevante.

Este analizador de texto, aunque fuese para un cliente específico, se podría usar como punto de partida para crear uno más genérico que entregar a un mayor número de clientes.

5.2 TECNOLOGÍAS USADAS

Tecnologías usadas	Definición
REST	Comunicación HTTP
JSON	Formato HTTP
POSTMAN	Cliente HTTP
PYTHON	Lenguaje de programación
PYCHARM	Entorno de programación
DJANGO	Marco de trabajo web

Tabla 1. Tecnologías usadas

Dado que este proyecto es un microservicio, las tecnologías usadas tienen relación con este tipo de programa. Para las peticiones HTTP se ha usado una arquitectura REST en formato JSON, ya que es un formato muy visual, y comparado con el XML es mucho más ligero al no contener etiquetas para representar la información. Las llamadas se han probado con Postman, un cliente HTTP.

También se ha usado Python y Django para la implementación del código. La razón principal de que se haya usado este lenguaje y este marco de trabajo es que el microservicio iba a ser integrado en la arquitectura de servicios de la empresa, que usan estas tecnologías. Estos servicios han sido implementados con Python por la utilidad de sus librerías para visión artificial y también, porque al ser un lenguaje interpretado, agiliza las pruebas de los servicios. Django es un marco de trabajo que otorga seguridad y facilidad para desarrollar aplicaciones web y servicios.

5.2.1 REST

REST es una técnica de arquitectura de software que se usa para desarrollar servicios web. Esta tecnología se basa en la comunicación mediante peticiones HTTP entre dos sistemas.

Cada petición contiene toda la información necesaria para realizar la operación que se quiere hacer, estas peticiones se estructuran de la siguiente manera:

Un verbo HTTP: que define la operación que se va a realizar. Hay cuatro operaciones básicas:

POST: para generar información.

GET: para obtener información.

PUT: para actualizar información.

DELETE: Para eliminar información.

Una cabecera: que tiene información sobre la petición. En este apartado se describe el tipo de mensaje que contiene la petición como por ejemplo JSON, XML, texto plano…

Una dirección URL: es un identificador único del recurso al que se quiere acceder. Las direcciones deben contener la información necesaria para especificar el recurso al que se accede.

Un cuerpo: información opcional que se añade al mensaje. [2]

Como en cualquier tecnología HTTP, disponemos de los códigos de estado. Estos códigos se envían en la respuesta de una petición HTTP, siguiendo el estándar RFC 7231 están formados por 3 dígitos, el primero indica el tipo de la respuesta y los dos siguientes no tienen ninguna categorización. [3]

Estos son las cinco categorías de códigos:

1xx (Informativo): solicitud recibida, se continúa con el proceso.

2xx (Exitoso): solicitud recibida con éxito, entendida y aceptada.

3xx (Redirigido): se necesitan más acciones para que se pueda completar la solicitud

4xx (Error en el cliente): la solicitud contiene sintaxis incorrecta o no puede ser entregada

5xx (Error en el servidor): el servidor ha fallado al procesar una solicitud aparentemente válida

5.2.2 JSON

JSON son las siglas de Notación de Objetos JavaScript en inglés. JSON es una sintaxis de texto para almacenar o intercambiar datos, permite a los ordenadores generar y formatear la información fácilmente y a la vez es comprensible por las personas para leerlo o escribirlo.

Está basado en el lenguaje de programación JavaScript que tiene parecidos a la familia de lenguajes C. JSON surge por la necesidad de cargar información rápidamente y de manera asíncrona.

En la comunicación entre un navegador y un servidor, la información que se intercambia está en un formato de texto. Es posible convertir objetos JavaScript a objetos JSON para enviarlo a un servidor y viceversa, lo que permite trabajar con la información como si fuesen objetos JavaScript directamente.

Su estructura consiste en una colección de parejas clave/valor. Un objeto en JSON es un conjunto de pares clave/valor que están dentro de llaves. Cada clave va seguida de dos puntos y los distintos pares van separados por una coma.

Los tipos de datos que hay en JSON son números, cadenas de texto, booleanos, null y Arrays.

Un ejemplo de JSON:

```
{

 "person": {
            "name": "Nombre Prueba",
            "lastname1": "Apellido1",
            "lastname2": "Apellido2",
            "country": "España",
            "age": 48,
            "eMail": "prueba@prueba.com",
            "driverlicense": true
            }


}
```

Figura 3. Ejemplo JSON

Se usará JSON para llamar al microservicio enviándole el texto extraído de las facturas y también en la respuesta, en la que se enviarán los campos pedidos una vez el texto ha sido procesado. [4]

5.2.3 POSTMAN

Postman es una aplicación que sirve para realizar comunicaciones HTTP. Dispone de una interfaz amigable para realizar peticiones y recibir respuestas HTTP.

Este software te permite guardar una serie de peticiones en colecciones para almacenarlas o exportarlas, con estas colecciones se pueden automatizar pruebas para que realice las llamadas que forman la colección en el orden que se le indique, lo que facilita el trabajo en servicios con un flujo de varias llamadas.

También dispone de espacios de trabajo, para que varios miembros de un mismo equipo trabajen sobre las mismas llamadas y se pueden depurar los servicios examinando las respuestas y añadiendo pruebas y funciones.

La llamada en Postman se divide en la URL con la que quieres comunicarte, en la que Postman tiene un cuadro de diálogo para que elijas el tipo de solicitud REST que es la llamada (POST, GET, DELETE, UPDATE). Después te divide en etiquetas la cabecera y el cuerpo de la llamada, con más desplegables ya configurados para el tipo de cabecera, o para especificar los parámetros en el cuerpo.

Además, incluye un visualizador de texto para interpretar las respuestas de una manera más intuitiva.

Se ha usado Postman para realizar las pruebas del microservicio. En las llamadas HTTP ese envía el texto que se tenía que analizar y se recibía la respuesta en un JSON. [5]

5.2.4 LENGUAJE PYTHON

Como la arquitectura en la que se iba a implementar este servicio de análisis de texto ya se encontraba en Python, éste también se ha programado en dicho lenguaje.

Las razones por las que la arquitectura está diseñada en Python son porque dispone de una sintaxis sencilla que permite realizar determinadas operaciones de una manera más simple.

Es un lenguaje interpretado, lo que agilizaba las pruebas con los microservicios, al compilarse automáticamente con cada cambio que se realiza.

También dispone de múltiples librerías útiles, sobre todo en el campo de visión e inteligencia artificial, que son los campos que se manejan en el departamento de

onboarding digital. En este caso, la que usaba el lector de documentos del que recibía el texto procesado el analizador de texto, llamada Tesseract, [6]

5.2.5 PYCHARM

Pycharm es el entorno de programación elegido para el desarrollo en Python del microservicio. Este entorno especializado en Python provee análisis de código, un depurador gráfico, un sistema integrado de pruebas unitarias y soporte para el desarrollo web con el marco de trabajo de Django. [7]

5.2.6 DJANGO

Django es un marco de trabajo web libre de alto nivel de Python que permite un diseño útil y sencillo de aplicaciones. Se creó en 2008 por programadores experimentados para simplificar la estructura del desarrollo de aplicaciones usando una estructura fija.

La estructura de Django se basa en un modelo vista controlador, que consiste en un sistema que media entre modelos de datos (clases de Python) y bases de datos relacionales (lo que en Django se llaman modelos), un sistema para procesar solicitudes HTTP (llamado vista) y un controlador URL basado en expresiones regulares (que es el controlador).

Aparte de eso incluye un servidor web para pruebas y desarrollo, un sistema de plantillas que usa el concepto de herencia de la programación orientada a objetos, soporte para clases de middleware, etc.

Django tiene distintos componentes para facilitar la estructura de la aplicación algunos de ellos son:

Capa de los modelos (models): Es la capa en la que se estructura y se manipula la información de tu aplicación web.

Capa vista (view): Django usa el concepto de vistas para agrupar la lógica responsable de procesar una solicitud del usuario y entregar su respuesta.

Capa de las plantillas (templates): las plantillas proveen una sintaxis amigable para presentar la información al usuario.

Una interfaz de administrador: Django dispone de una interfaz que te permite gestionar el contenido de del sitio web que se está desarrollando web.

También crea algunas clases para simplificarte el desarrollo de la aplicación y la estructura de entorno, como pueden ser:

Manage.py: sirve para que se puedan ejecutar determinados comandos sobre las aplicaciones.

Settings.py: es un archivo que contiene toda la configuración de la instalación de Django

Urls.py: En esta clase se encuentra un esquema URL en el que defines la URL de tu aplicación. [8]

6 DESARROLLO

La implementación que se ha decidido llevar a cabo, al observar el conjunto de facturas de muestra que entregó el cliente, fue un microservicio de análisis de texto basado en expresiones regulares. Este analizador de texto permitirá que el cliente pueda procesar todas las facturas que tiene almacenadas, al obtener los datos relevantes estructurados.

Se espera que el programa automático de digitalización de facturas tarde muy poco tiempo frente a una inserción manual de la información en la base de datos de gestión contable de facturas. Esta automatización permitirá no sólo la simplificación de los procesos de digitalización del cliente, sino la posibilidad de tratar un elevado número de facturas. Se han hecho pruebas comparativas, el tratamiento automático tarda menos de dos segundos por factura, mientras que el manual requiere de al menos un minuto por factura. Hay que tener en cuenta además que en todos los procesos manuales hay una componente de posible inserción de errores humanos que se eliminan con el programa automático.

Para extraer la información relevante en sus campos correspondientes, se ha diseñado una heurística, usando expresiones regulares y patrones del texto.

Como el tipo de cliente es una empresa del sector automovilístico, la mayoría de las facturas que procesemos serán sobre compras de vehículos, aunque también nos encontraremos con alguna simplemente relacionada con vehículos, como puede ser una operación de un taller.

Los campos que más le interesan al cliente son los siguientes:

- Código del concesionario
- Nombre del concesionario
- Número de chasis
- impuesto impositivo
- Porcentaje de iva
- Número de factura
- Fecha de factura
- Precio base, iva
- Porcentaje de iva
- Precio total.

En un principio este trabajo se desarrolla de manera que entregue los campos concretos pedidos para este cliente, pero se podría usar para otros clientes del mismo campo

automovilístico, adaptando los campos que necesiten estos. Se podrían aprovechar campos generales como el del IVA o la fecha de factura como punto de partida para clientes ajenos a este ámbito.

6.1 ESTRUCTURA DEL MICROSERVICIO

El microservicio está integrado con otros en un mismo paquete, las clases y métodos que conforman este microservicio son las que se describen a continuación.

El flujo del microservicio consiste en iniciarlo con una llamada REST de tipo POST, enviando el texto que se tiene que procesar.

Esta llamada se recibe en la clase OCRController, que llama a la clase OCROrchestrator.

La clase OCROrchestrator tiene la función separate_leasing, que se encarga de llamar a los métodos get de la clase LeasingParser en bucle, hasta que se rellena el objeto de respuesta.

Una vez se tienen los campos llenos, se devuelve una respuesta HTTP con ellos en la clase OCRController.

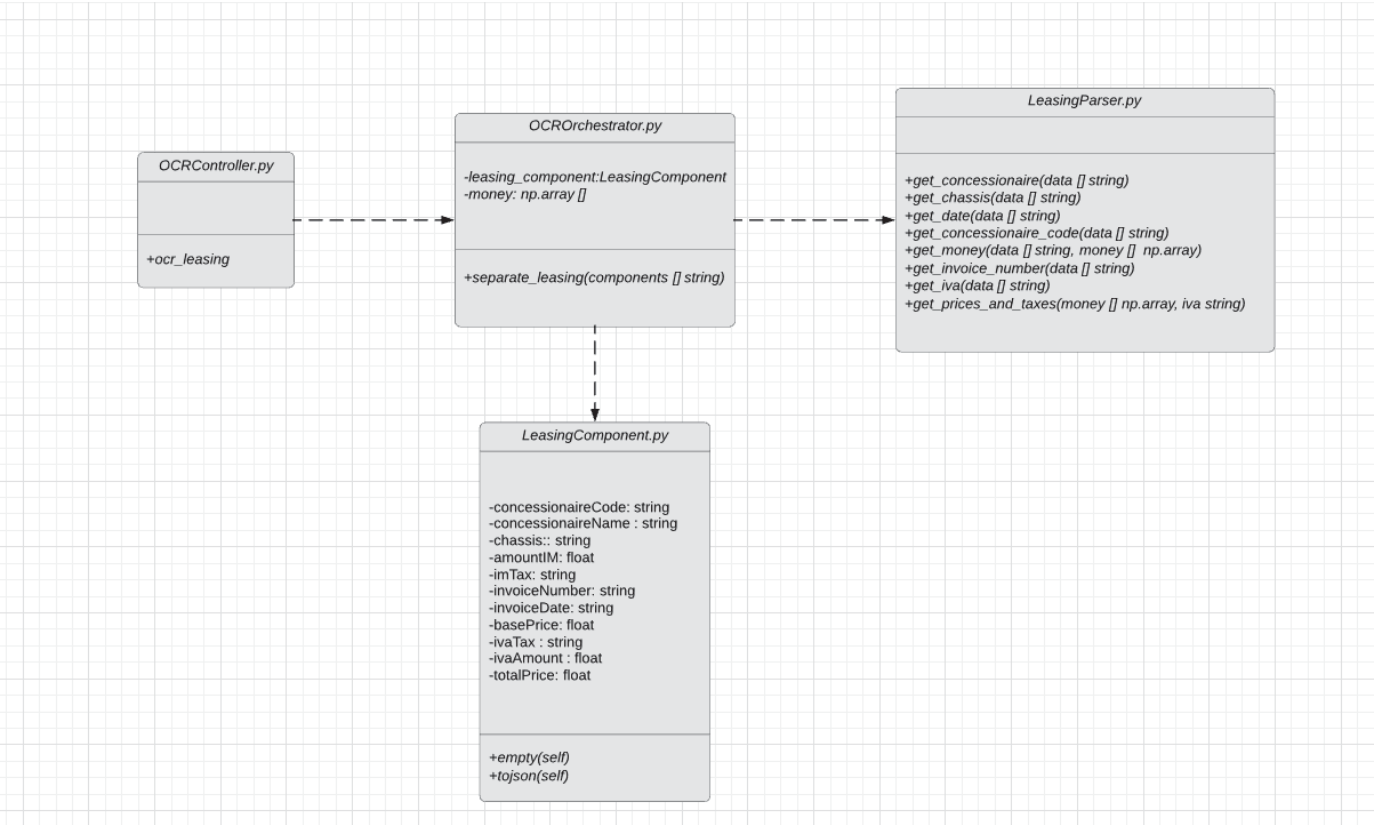

Figura 4. Diagrama del microservicio

6.1.1 CLASE LEASINGCOMPONENT

Esta clase contiene el objeto que tiene los campos que se deben rellenar.

MÉTODO __INIT__

Este método es el constructor del objeto LeasingComponent, se encarga de inicializar todas las variables del objeto. Éste consta de los campos:

concessionaireCode: Código del concesionario.

concessionaireName: nombre del concesionario.

chassis: número de chasis del vehículo.

amountIM: impuesto impositivo.

imTax: porcentaje del impuesto impositivo que se aplica.

invoiceNumber: número de factura.

invoiceDate: fecha de factura.

basePrice: precio base del vehículo.

ivaTax: porcentaje del IVA que se aplica.

ivaAmount: cantidad del IVA.

totalPrice: precio total.

MÉTODOS EMPTY

Uno por variable, se encarga de comprobar si está vacía.

MÉTODO TOJSON

Convierte el objeto a un JSON.

6.1.2 CLASE OCRCONTROLLER

La clase controller se encarga de recibir y enviar las peticiones HTTP.

MÉTODO OCR_LEASING

Este método se encarga de recibir la llamada HTTP. Comprueba que la llamada sea un POST, y que reciba un texto para procesar, en caso contrario devolverá error. Después

llama a la clase OCROrchestrator para que rellene los campos con la información correspondiente y posteriormente devuelve la respuesta HTTP con dicha información.

6.1.3 CLASE OCRORCHESTRATOR

Esta clase es la que se encarga de guardar los datos devueltos por la clase en la que se analiza el texto y devolverlos al controller.

MÉTODO SEPARATE_LEASING

Este método se encarga de rellenar los campos del objeto LeasingComponent recorriendo el texto y llamando a los gets de LeasingParser.

6.1.4 CLASE LEASINGPARSER

Esta clase es el núcleo de la lógica del programa. En ella se encuentran los métodos que usan las expresiones regulares que analizan el texto y extraen los campos pedidos.

6.1.5 CLASES DE DJANGO

En la estructura del microservicio también se usan las clases mencionadas anteriormente en la explicación de Django.

En la clase urls.py se añade la url del servicio:

url(r'^ocr/leasing/service$', OCRController.ocr_leasing, name='ocr_leasing'),
url(r'^ocr/leasing/service/$', OCRController.ocr_leasing, name='ocr_leasing'),

6.2 PROGRAMACIÓN DE LA HEURÍSTICA

Un programa heurístico es un programa formado por un grupo de reglas, definidas por el sentido común o procedimientos para obtener un resultado. Estas reglas se basan en expresiones regulares y se definen en las funciones o métodos de la clase LeasingParser.

En este apartado se explica brevemente el funcionamiento de estos métodos, la forma en que cada uno analiza el texto buscando el dato correspondiente.

6.2.1 MÉTODO GET_CONCESSIONAIRE

Este método se encarga de obtener el nombre del concesionario. Como todos los nombres de concesionario terminan en el tipo de sociedad al que pertenece (SA, SL, SLU…) se observa si la línea contiene una de estas siglas, y si es así se selecciona el texto que lo acompaña.

6.2.2 MÉTODO GET_CHASSIS

Este método obtiene el número de chasis del vehículo. Se encarga de filtrar palabras alfanuméricas de 17 caracteres que cumplan las restricciones establecidas por la ISO 3779. [9]

6.2.3 MÉTODO GET_DATE

Este método obtiene la fecha de la factura. Únicamente aparece una fecha en estas facturas, así que se filtra según el formato de fecha.

6.2.4 MÉTODO GET_CONCESSIONAIRE_CODE

Este método se encarga de obtener el código del concesionario (CIF). Se filtran las palabras que comiencen por una letra, y vayan seguidas de 8 dígitos. [10]

6.2.5 MÉTODO GET_MONEY

Es un método auxiliar que sirve para obtener un array con todos los precios que haya en la factura.

6.2.6 MÉTODO GET_INVOICE_NUMBER

Este método sirve para obtener el número de factura. Se busca en las líneas del texto hasta que se encuentra la palabra FACTURA. Una vez se ha encontrado, se selecciona el texto que la sigue, o en caso de que este vacío se selecciona la línea de texto siguiente.

6.2.7 MÉTODO GET_IVA

Este método se encarga de obtener el IVA. Se busca por la palabra IVA y se selecciona el texto que este al lado si hay un porcentaje. En caso de que no se encuentre junto a la palabra IVA, se busca en las líneas cercanas.

6.2.8 MÉTODO GET_PRICES_AND_TAXES

Este método se encarga de obtener el precio base, el precio total, el IVA y el porcentaje de impuesto impositivo y el impuesto impositivo. Recurriendo al array obtenido en get_money, se ordena y se busca el mayor precio, este será el precio total.

Con el precio total y el porcentaje de IVA se calcula la cantidad de IVA.

Con la cantidad de IVA y el precio total se obtiene el precio base.

Con el IVA, el precio base y el precio total se obtiene la cantidad de impuesto impositivo y su porcentaje.

7 RESULTADOS

Como ejemplo de operación, se presenta la salida o respuesta a una lectura de factura con el siguiente formato:

"concessionaireCode": "A12345678",
"concessionaireName": "CONCESIONARIO SA ",
"chassis": "XX0XX123456789012",
"amountIM": 0,
"imTax": "0%",
"invoiceNumber": "XX1234567",
"invoiceDate": "DD/MM/AAAA",
"basePrice": 12345.6,
"ivaTax": "21.0%",
"ivaAmount": 1234.56,
"totalPrice": 13580,16,

El servicio devuelve 11 campos, se procesaron inicialmente 40 facturas, y su salida se dispuso en un Excel con los datos que tenía que extraer el analizador de texto manualmente. Para analizar la calidad del resultado del proceso, posteriormente se fue comprobando el índice de éxito de campos y en cuáles fallaba, así como las posibles causas [Figura 5].

Código concesionario	Nombre concesionario	Chasis	Importe IM	Nº factura	Fecha factura	Precio base	Tipo impositivo	Impuestos	Total
-	##########	CHASIS:	##########	653,48	-	FACTURA:	##########	FECHA: #### TO	13.157,45	IVA:	21	%	2.869,07	Total a Pagar	17.300,01	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	10.252,40	% IVA (arriba)	21	%	2.153,00	TOTAL A PAGAR	12.405,40	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	10.661,15	% IVA (arriba)	21	%	2.238,84	TOTAL A PAGAR	12.899,33	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	13.865,23	% IVA (arriba)	21	%	4.111,28	TOTAL A PAGAR	24.034,50	-
CIF:	##########	Nº CHASIS (arriba)	##########	-	-	FACTURA Nº:	##########	Fecha: #### Ba	8.231,4000	% IVA (arriba)	21	%	1.728,59	Total a Pagar	9.959,99	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	8.231,40					TOTAL Liqui	9.959,99	-
-	##########	CHASIS:	##########	-	-	FACTURA:	##########	FECHA: #### TO	12.396,69	% IVA (arriba)	21	%	2.603,30	TOTAL A PAGAR	14.999,99	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	FECHA: #### TO	3.669,42	IVA:	21	%	2.030,58	TOTAL A PAGAR	11.700,00	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	17.625,40	Tipo de IVA: %	21,00	%	3.701,33	Total a Pagar	21.326,73	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	16.364,76	% IVA (arriba)	21	%	3.566,80	TOTAL A PAGAR	20.551,56	-
-	##########	CHASIS:	##########	1.226,85	-	Núm.Factura:	##########	Fecha: #### To	25.815,70	Tipo de IVA: %	21,00	%	5.421,30	Total a Pagar	31.237,00	-
-	##########	CHASIS:	##########	-	-	FACTURA:	##########	FECHA: #### TO	3.719,82	IVA:	21	%	2.054,18	Total a Pagar	11.836,00	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	13.512,40	Tipo de IVA: %	21,00	%	2.837,60	Total a Pagar	16.350,00	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	11.614,05	Tipo de IVA: %	21,00	%	3.696,95	Total a Pagar	21.310,00	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	16.342,15	Tipo de IVA: %	21,00	%	3.551,85	Total a Pagar	20.500,00	-
C.I.F.:	##########	Nº chasis:	##########	-	-	Nº DOCUMENTO:	##########	Fecha de: #### To	11.047,10	-	21,00	%I.V.A.	2.319,90	TOTAL A PAGAR	13.367,00	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	18.685,36	% IVA (arriba)	21	%	3.923,93	TOTAL A PAGAR	23.609,23	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	18.181,82	% IVA (arriba)	21	%	3.818,18	TOTAL A PAGAR	22.000,00	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	18.181,82	% IVA (arriba)	21	%	3.818,18	TOTAL A PAGAR	22.000,00	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	18.286,31	% IVA (arriba)	21	%	3.840,13	TOTAL A PAGAR	22.126,44	-
-	##########	CHASIS:	##########	-	-	FACTURA NUMERO:	##########	FECHA: #### TO	12.443,15	I.V.A.	21	%	2.613,06	TOTAL A PAGAR	15.056,21	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	14.108,51	% IVA (arriba)	21	%	2.962,79	TOTAL A PAGAR	17.071,30	-
-	##########	Nº CHASIS (arriba)	##########	MATRICULACION	-	750,00	-	FECHA: #### NE	21.884,30	% IVA (arriba)	21	%	5.855,70	TOTAL A PAGAR	34.490,00	-
-	##########	Nº CHASIS (arriba)	##########	IMP.MATRICULA	4,75%	1.153,45	-	FECHA: #### NE	24.283,19	% IVA (arriba)	21	%	5.093,47	TOTAL A PAGAR	29.376,66	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	3.856,93	% IVA (arriba)	21	%	2.063,96	TOTAL A PAGAR	11.926,89	-
-	##########	XASSIS:	##########	-	-	FACTURA NÚMERO:	##########	DATA: #### TO	3.952,89	I.V.A.	21,00	%	1.322,11	TOTAL A PAGAR	11.075,00	-
-	##########	-	##########	-	-	Factura Nº:	##########	Fecha: #### Ba	218,01	% (arriba)	21,00	IVA	45,79	Total Factu	863,86	EUROS
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	14.314,38	% IVA (arriba)	21	%	3.046,15	TOTAL A PAGAR	17.563,13	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	10.134,71	% IVA (arriba)	21	%	2.128,23	TOTAL A PAGAR	12.263,00	-
-	##########	Nº CHASIS (arriba)	##########	-	-	Nº DE FACTURA:	##########	FECHA: #### NE	10.607,92	% IVA (arriba)	21	%	2.227,66	TOTAL A PAGAR	12.835,58	-
-	##########	Nº CHASIS (arriba)	##########	IMPUESTO MATRI	1.117,61	cuot	Nº DE FACTURA:	FECHA: #### NE	24.795,39	% IVA (arriba)	21	%	5.207,16	TOTAL A PAGAR	30.003,15	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	17.206,88	Tipo de I.V.A. %	21,00	%	3.613,86	Total a Pagar	20.820,74	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	13.772,02	Tipo de IVA: %	21,00	%	2.892,12	Total a Pagar	16.664,14	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	17.873,56	Tipo de IVA: %	21,00	%	3.753,45	Total a Pagar	21.627,01	-
-	##########	Nº CHASIS (arriba)	##########	IMPUESTO MATRI	-	0,00	cuot	FACTURA Nº:	##########	Fecha	DATA: #### TO	9.421,43	I.V.A.	21,00	%	1.978,51	TOTAL A PAGAR	11.400,00	-
-	##########	XASSIS:	##########	-	-	FACTURA NÚMERO:	##########	DATA: #### TO	11.438,16	I.V.A.	21,00	%	2.402,01	TOTAL A PAGAR	13.840,17	-
-	##########	CHASIS (arriba)	##########	-	-	Núm.Factura:	##########	Fecha: #### To	16.538,88	Tipo de IVA: %	21,00	%	3.473,16	Total a Pagar	20.012,04	-
-	##########	CHASIS:	##########	-	-	FACTURA NUMERO:	##########	FECHA: #### TO	8.002,34	I.V.A.	21,00	%	1.680,47	TOTAL A PAGAR	3.682,71	-
-	##########	CHASIS:	XXXXXXXXXXXXXXXXX	(E.D.M.T. PAGADO	0%	-	FACTURA NUM:	XXXXXXXXXXXXXXXX	FECHA: #### TO	12.413,34	IVA:	21,00	%	2.620,66	TOTAL A PAGAR	15.100,00	-

Figura 5. Excel del resultado de las pruebas

Se analizaron 40 PDFs. Tras ver que algunos campos se devolvían vacíos, se analizaron las posibles causas. Se observó que la principal razón de que los campos se devolviesen vacíos era porque el lector OCR no era capaz de extraer esa información del PDF correctamente, de modo que en el texto a analizar no se encontraban los datos. Otra razón encontrada es que el lector OCR confunde a veces alguna letra o número que precede el dato, o que es parte del dato. Como el analizador de texto se basa en expresiones regulares que buscan distintos caracteres anteriores y posteriores para encontrar la información buscada, esto hace que falle, aunque el campo buscado se pudiera en principio leer correctamente.

Por tanto, hay tres causas de fallos:

Los primeros serían los errores totales debido a fallos en la lectura de OCR, en los que el texto extraído del PDF no contiene el dato a extraer. Contando estos errores, los campos restantes que sí se han devuelto correctamente son el 89,31%.

La segunda consiste en errores del OCR en algún carácter que afecte a algún dato, lo que implica que, si la lógica del programa fuese menos estricta, ya sea usando distancia de Levenshtein en estos casos o inteligencia artificial como el procesamiento del lenguaje natural, quizá podría obtenerlo. En estos casos el índice de acierto es del 97,95%.

Por último, si se tienen en cuenta los errores del OCR en la lectura de campos que no son datos, pero que la lógica del analizador usa para encontrarlos, el porcentaje de campos devueltos correctamente sería del 99,77.

Estos datos muestran que el programa no tiene ningún problema para extraer la información de un texto extraído correctamente por el lector, y que mientras las distintas facturas mantengan la misma estructura de los datos, el microservicio devolverá los campos correctamente.

8 LÍNEAS DE FUTURO

Este trabajo se ha realizado para el caso concreto pedido por un cliente, con lo cual, el mejor tema para mejorar este proceso es centrarse en la estandarización de éste, y que sirva a distintos clientes que pidan cosas diferente.

Se podría hacer configurable para cada cliente, de manera que cada uno active o desactive los campos que le interesase recibir. Eso no sólo serviría a nuevos clientes para decidir los campos que quieran obtener, si no que los que ya lo estuviesen usando podrían decidir la estructura de datos cuando ellos quisiesen, en caso de que ya no les interesase algún dato o, al contrario, que quisiesen obtener alguno nuevo.

También es posible añadir el procesamiento del lenguaje natural, que se descartó por cuestiones de complejidad y tiempo. Usando este procedimiento haría posible el uso del analizador de texto para documentos muy diferentes entre sí, de distintos clientes de diferentes ámbitos.

Dejando de lado el hacer el analizador de texto genérico, que es lo que más margen de mejora produciría. También se podrían aumentar los campos que se extraen del texto, para entregar una mayor información al cliente.

9 CONCLUSIONES

Los objetivos marcados han sido cumplidos como se observa en los resultados, El analizador de texto ha obtenido un índice de éxito de 99,77%, en los casos que implican directamente la lógica de las reglas implementadas. La mayor parte de los campos incompletos han ocurrido por la ausencia de éstos en el texto a analizar, incluyendo estos errores, que provienen del lector, el índice de éxito también ha sido bastante alto, de un 89,31%.

El texto entregado por el lector OCR podía no ser perfecto debido a la calidad de la imagen de la factura, la mayoría de estos documentos se obtienen escaneando una factura física, lo que puede provocar que en la imagen haya manchas o dobleces. Aparte de que la factura ha tenido que ser impresa primero pudiendo perder calidad en la impresión. Si la imagen o el PDF de la factura esta borrosa, tiene manchas o el propio texto no se puede leer bien, la información extraída puede estar incompleta, produciendo campos vacíos en la respuesta del analizador. Por esta razón, aparte de las maneras de mejorar el analizador de texto, expuestas en la línea de futuro, cualquier mejora que se hiciese al lector OCR para conseguir identificar mejor los caracteres de la factura tendría un efecto directo en el porcentaje de éxito del analizador. También se podría aplicar un tratamiento a la imagen usando filtros antes de usar el lector OCR, lo que produciría mejores resultados a costa de un aumento del tiempo del proceso.

Los objetivos personales también se han cumplido, he aprendido a desarrollar en python y he aprendido a manejarme con el marco de trabajo de Django con el diseño de este microservicio.

10 Bibliografía

[1] Imagen del analizador de PDF[Online]. Available: https://docparser.com/blog/pdf-parser/

[2] Software de procesamiento del lenguaje natural KERAS [Online]. Available: https://www.opencodez.com/python/text-classification-using-keras.htm

[3] Integración de palabras, procesamiento del lenguaje natural [Online]. Available: https://www.datascience.com/blog/word-embeddings-natural-language-processing

[4]Tecnología REST [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer

[5]Estandar RFC7231 Códigos de estado HTTP [Online]. Available: https://tools.ietf.org/html/rfc7231#section-6

[6]Cliente HTTP POSTMAN [Online]. Available: https://www.getpostman.com/products

[7] Formato JSON[Online]. Available: https://www.json.org/

[8] Lenguaje python[Online]. Available: https://www.python.org/

[9]Entorno de Desarrollo Pycharm [Online]. Available: https://www.jetbrains.com/pycharm/

[10]Marco de trabajo django [Online]. Available: https://www.djangoproject.com/

[11]Formato del numero de chásis [Online]. Available: https://es.wikipedia.org/wiki/N%C3%BAmero_de_chasis

[12] Formato del CIF[Online]. Available: https://es.wikipedia.org/wiki/C%C3%B3digo_de_identificaci%C3%B3n_fiscal

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Sun Jan 13 21:17:24 CET 2019
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)