Shopping List Automator: Geração Automática de Listas de Compras de Supermercado

Universidade Fernando Pessoa

Hugo David Santos

Faculdade de Ciência e Tecnologia

Universidade Fernando Pessoa

Uma tese submetida para obtenção do grau de

Mestre em Engenharia Informática, ramo Computação Móvel

2021

Resumo

O ritmo de vida atual é cada vez mais assoberbado por tarefas repetitivas que dificultam muitas vezes a gestão do tempo e da vida pessoal e familiar. Muitas destas tarefas, que são efetuadas rotineiramente, podem no entanto ser otimizadas ou mesmo automatizadas. Falamos em particular da realização de compras de supermercado, que se repetem diária, semanal ou mensalmente, tirando-nos muito tempo e colocando stress na organização do dia-a-dia.

Este trabalho foca-se num sistema capaz de sugerir listas de comprar automáticas, baseando-se no padrão de consumo do utilizador. Tipicamente temos por costume verificar repetidamente que produtos nos faltam em casa, sempre que nos deslocamos às compras. Este sistema procura simplificar este planeamento, sugerindo ao utilizador uma lista que tem em consideração o consumo médio registado de cada produto. A solução desenvolvida baseia-se na utilização de uma aplicação móvel usada para registar o histórico de faturas de compras realizadas. Esta aplicação permite também submeter as faturas a um serviço de *backend* para registo e análise de consumos. As faturas são importadas através da captura de fotografias ou selecionando imagens ou ficheiros pré-existentes no *smartphone*. No servidor efetua-se inicialmente a união de imagens de faturas mais longas que não cabem numa única foto, seguindo-se o pré-processamento da imagem, extração de texto e categorização de informação que permite armazenar numa base de dados toda a informação relativa aos produtos adquiridos pelo utilizador. Posteriormente, os dados históricos de consumo do utilizador são usados para estimar as necessidades de produtos e sugerir automaticamente uma lista de compras na próxima ida ao supermercado.

As informações relativas às faturas importadas, respetivos produtos adquiridos e a lista de compras automática são apresentadas na aplicação móvel. O serviços de *backend* permite identificar o supermercado e a data de cada fatura, os produtos contidos na mesma e a respetiva quantidade, capacidade e preço unitário. Para a extração de texto a partir de imagens das faturas de compras recorreu-se a uma ferramenta de OCR denominada *Tesseract*, que foi combinada com um componente desenvolvido neste trabalho para interpretar e categorizar a informação recolhida pelo OCR. A avaliação da precisão da extração do texto e da sua categorização em produtos e quantificação

de consumo, foi efetuada através de comparações entre os resultados obtidos pelo componente proposto e os valores de teste esperados. Para esta avaliação recorreu-se a métricas de comparação de *strings* e a um *script* que analisa e compara a informação recolhida com uma estrutura JSON pré-definida de produtos. Todo o sistema desenvolvido foi descrito nesta dissertação, juntamente com os testes efetuados que permitiram aferir o cumprimento dos requisitos inicialmente propostos.

Abstract

The current pace of life is increasingly overwhelmed by repetitive tasks that often make it difficult to manage time and personal and family life. Many of these tasks, which are performed routinely, can however be optimized or even automated. We are talking in particular about grocery shopping, which is repeated daily, weekly or monthly, taking a lot of time and putting stress in the organization of our day-to-day activities.

This work focuses on a system capable of suggesting automatic shopping lists, based on the user's consumption pattern. Typically, it is our custom to repeatedly check what products we are lacking at home, whenever we go shopping. This system seeks to simplify this planning, suggesting to the user a list that takes into account the average consumption registered for each product. The developed solution is based on the use of a mobile application used to record the history of purchase invoices made. This application also allows submitting invoices to a backend service for recording and analyzing consumption. Invoices are imported by capturing photos or selecting pre-existing images or files on the smartphone. On the server, images of longer invoices that do not fit into a single photo are initially joined, followed by image pre-processing, text extraction and information categorization that allows all information relating to the products purchased by the user. Subsequently, the user's historical consumption data is used to estimate product needs and automatically suggest a shopping list on the next trip to the supermarket.

Information regarding imported invoices, the respective purchased products and the automatic shopping list are displayed on the mobile application. The backend services make it possible to identify the supermarket and the date of each invoice, the products contained in it and the respective quantity, capacity and unit price. To extract text from images of purchase invoices, an OCR tool called Tesseract was used, which was combined with a component developed in this work to interpret and categorize the information collected by OCR. The evaluation of the accuracy of the text extraction and its categorization into products and consumption quantification was carried out through comparisons between the results obtained by the proposed component and the expected test values. For this evaluation, we used string comparison metrics and a script that analyzes and compares the information collected with

a pre-defined JSON structure of products. The entire system developed was described in this dissertation, together with the tests carried out that allowed to assess compliance with the initially proposed requirements.

Porque acredito em Deus e coloco nas suas mãos o meu caminho, agradeço primeiramente a Ele por ter conduzido os meus passos até aqui e me ter dado as forças para concluir esta etapa. Uma vez mais mostrou-me que se formos fiéis à sua Palavra e confiarmos na Sua força e poder, todas as portas certas se abrem e no fim somos sempre recompensados.

Agradecido estou também a Deus por me ter preparado uma companheira maravilhosa, a Carla, que foi fonte de alegria e motivação em todos os momentos. Este amor puro foi chave.

Agradeço à minha família, em especial à minha avó, por toda a ajuda, apoio e sacrifício que me permitiu concluir o mestrado nas melhores condições.

Agradecimentos

Com distinção, agradeço aos Professores Doutor Rui Moreira (orientador) e Doutor Christophe Soares (co-orientador) pelo apoio técnico e acima de tudo pelas competências humanas que admirei não só no desenvolvimento desta tese de mestrado, mas também durante a minha licenciatura e restante período do mestrado.

Vejo este trabalho como uma representação final do meu percurso de cinco anos pela Universidade Fernando Pessoa, na licenciatura e mestrado em engenharia informática. Portanto, sinto de agradecer a todos os restantes docentes com que me fui cruzando. Admirei os seus comportamentos em diversas situações e deles retirei muita aprendizagem humana. Guardo também várias conversas e olhares simpatizantes, que contribuíram para que considerasse como muito feliz este capítulo da minha vida.

Agradeço aos meus bons amigos e companheiros, os quais conheci e convivi durante o meu percurso académico. As aventuras vividas e o companheirismo demonstrado serão eternamente recordados.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API *Application Programming Interface*

BLE *Bluetooth Low Energy* 11

BLoC *Business Logic Component* 56

COL *Continente Online* 7

CUTIE *Convolutional Universal Text Information Extractor* 17

DNN *Deep Neural Network* 17

GTIN *Global Trade Item Number* 12

IoT *Internet of Things* 11

JSON *JavaScript Object Notation*

JWT *JSON Web Token* 25

ML *Machine Learning* 78

NLR *Named Entity Recognition* 20

OCR *Optical Character Recognition* 4

SISR *Single Image Super-Resolution* 14

Capítulo 1

Introdução

A compra de bens essenciais para o consumo próprio é uma atividade indispensável e frequente para toda a população. Por vezes tentamos entender o que falta nos nossos lares somente recorrendo à nossa memória, sem planeamento preciso, e acabamos por nos esquecer de comprar alguns produtos. Outras vezes fazemos uma lista de compras detalhada, com a desvantagem de ainda nos consumir mais tempo e esforço de planeamento.

Na nossa rotina fazemos por vezes deslocações inesperadas a um supermercado, seja por necessidade de um produto específico a curto prazo (como a compra de carne/peixe para uma próxima refeição), ou por nos encontrarmos próximos de um estabelecimento e tiramos partido da proximidade para fazer algumas compras. Porém, pelo facto dessa deslocação ser decidida na hora ou por falta de tempo, muitas das vezes não temos a lista de compras elaborada ou acessível.

Neste momento existem já aplicações de supermercados, sobretudo das principais cadeias, com o intuito de facilitar a criação manual de listas de compras e a gestão e planeamento de compras. Porém, sentimos a necessidade de criar uma ferramenta automática e universal, que permita trabalhar com dados de vários supermercados em simultâneo visto que, geralmente, as famílias recorrem a diferentes cadeias e não estão limitadas a um único supermercado.

O sistema móvel proposto nesta dissertação, procura colmatar esta lacuna e permitir aos utilizadores saberem que bens essenciais necessitariam de comprar para casa usando como base o seu histórico de compras. Esta solução funciona de forma automatizada, sem a necessidade de elaborar as listas de compras, poupando por isso muita preocupação e tempo a grande parte da população.

1.1 Contexto e Problema

Nos últimos anos cada vez mais supermercados disponibilizam as faturas no formato digital, enviando-as opcionalmente para o e-mail do cliente. Estes podem assim visualizar

as faturas na aplicação móvel oficial dessa mesma superfície. Normalmente esta opção só é possível se o cliente utilizar o cartão de cliente associado às compras que efetuar e se instalar a aplicação móvel de cada supermercados que frequenta. Praticamente todas as grandes superfícies comerciais têm ou irão por certo oferecer soluções na forma de aplicações móveis, disponibilizando informações sobre as compras associadas aos cartões de cliente. Contudo, esta solução não é prática quando os utilizadores são clientes de vários supermercados.

Atualmente não existe uma solução prática e transversal a vários supermercados, que facilite aos clientes dessas superfícies o armazenamento e visualização das suas faturas de compras de forma digital e que permita a geração de listas de compras de forma automatizada. A criação de uma aplicação universal que permita armazenar faturas de todos os supermercado e reunir todas as informações, seria portanto vantajosa para qualquer consumidor que utilize vários supermercados com frequência. Em particular a informação recolhida poderia ser muito útil para a geração automática de listas de compras adequadas às necessidades de cada cliente.

A geração manual de listas de compras exige um conjunto de passos que são demasiado desgastantes e sujeitos a erros (Figura 1.1). Muitas vezes pelo facto de dispormos de pouco tempo, porque simplesmente nos esquecermos ou por outros motivos associados à gestão diárias dos nossos afazeres, acontece de não realizarmos um planeamento adequado das idas ao supermercado. Isto aumenta a probabilidade de nos esquecer-mos de produtos que necessitamos ou que acabarmos por adquirir em quantidades incorretas. Podendo, portanto, resultar em novas idas às compras ou na expiração de prazos de validade dos produtos adquiridos. Assim sendo, consideramos que a geração automática de listas de compras, baseada no histórico de consumos, será uma mais valia para a população em geral, simplificando a gestão do seu dia-a-dia.

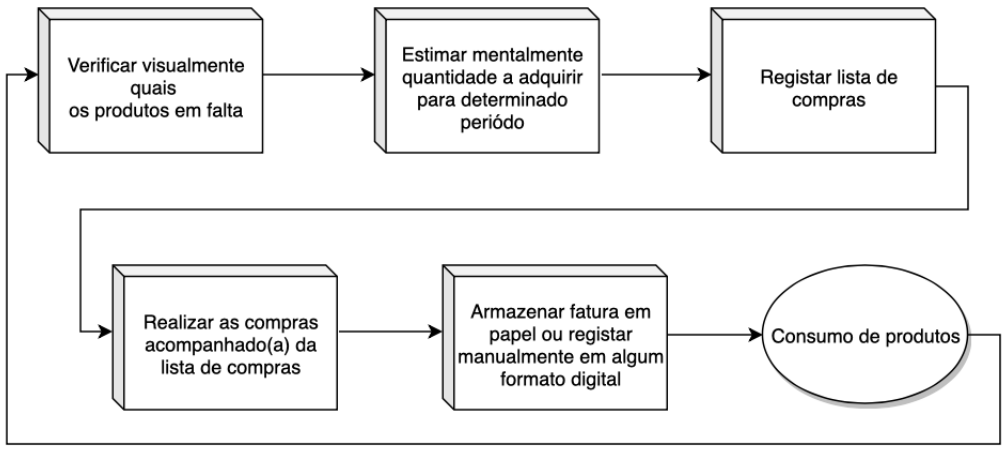

Figura 1.1: Procedimento comum para a realização de compras

1.2 Objetivos

O foco principal do trabalho proposto nesta dissertação é facilitar e reduzir a quantidade de tempo e atenção que todos os consumidores de supermercados despendem na elaboração das listas de compras e na ida ao supermercado. Com a elaboração automática da lista de compras pretende-se assegurar que não há esquecimento de nenhum produto necessário, nem compras em excesso, de acordo com os hábitos de consumo passados.

Neste sentido, pretende-se com este projeto criar um sistema prático que permita às pessoas ter os dados das suas compras automaticamente armazenados. Este sistema deverá simplificar a recolha individual da informação de consumo de cada cliente, recorrendo à extração e análise das faturas desse cliente. Após a extração, interpretação e armazenamento do histórico de consumo do cliente, o sistema deverá conseguir extrapolar a lista de compras personalizada ao utilizador, isto é, gerar de forma automática e sempre que necessário uma lista de compras devidamente adaptada às necessidades do utilizador.

Sabemos que não adquirimos exatamente os mesmos produtos, nem nas mesmas quantidades, de cada vez que vamos a um mercado local ou a um supermercado, pois as nossas preferências variam, por exemplo, conforme a época do ano ou dependendo de muitos outros fatores influenciadores. Ainda assim, consideramos que existem alguns padrões de consumo que nos permitirão apresentar ao utilizador uma sugestão de uma lista de compras base. Esta lista baseada no histórico de compras do utilizador, poderá ser uma boa referência para as próximas aquisições, poupando tempo e organização.

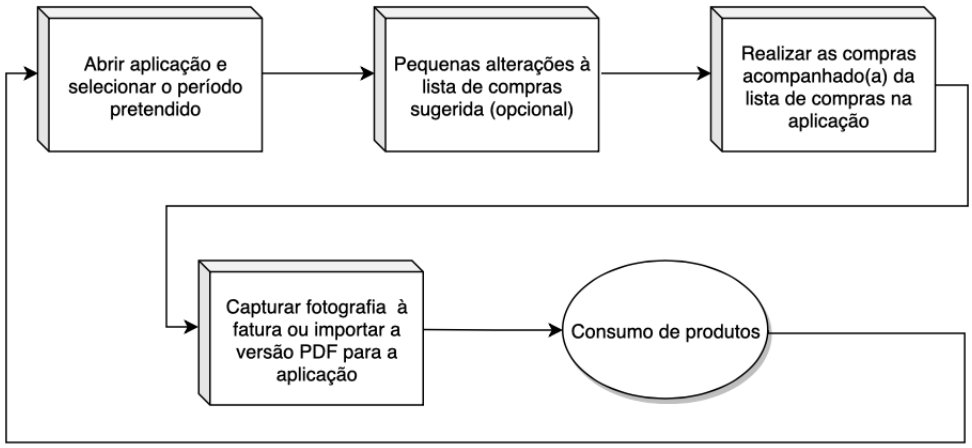

Figura 1.2: Procedimento pretendido para a realização de compras recorrendo ao *Shopping List Automator*

Como apresentado na figura 1.2, pretende-se que o utilizador, num momento que precede a ida às compras, tenha apenas que abrir a aplicação e selecionar a periodicidade das suas idas ao supermercado. Dessa forma o sistema poderá identificar e calcular a

quantidade de cada produto para incluir na lista de compras a efetuar. Com base na lista de compras, ao utilizador basta colocar os produtos no carrinho guiando-se pela lista disponível na aplicação do seu *smartphone*. Após as compras, a mesma aplicação permitirá capturar uma fotografia da fatura das compras ou a fatura em formato digital, obtida do supermercado. As faturas podem ser submetidas através da aplicação móvel ao *backend* para desencadear o processo de extração de informação, necessária à previsão de consumo do utilizador.

O sistema que denominamos *Shopping List Automator* irá assim permitir ao consumidor registar e gerir de forma automática os consumos e respetivas listas de compras. Este sistema será constituído por uma aplicação móvel, funcional tanto em Android como em iOS, e um serviço de *backend* de suporte à aplicação. Este sistema permitirá assim aos utilizadores armazenarem e visualizarem, de forma simples, as faturas de compras passadas. Adicionalmente, deverá também permitir automatizar a geração e organização de listas de compras em função do histórico e necessidades do utilizador. Para conseguir atingir este objetivo será necessário capturar, com recurso a fotos, as faturas das compras. Posteriormente, extrair o histórico de consumos a partir das fotos da faturas, recorrendo à utilização de um *Optical Character Recognition* (OCR), combinado com um componente de categorização do texto identificado pelo OCR. Com toda a informação categorizada e armazenada poderá construir-se um histórico de consumos que será utilizado na geração automática da lista de compras (ver Figura 1.3).

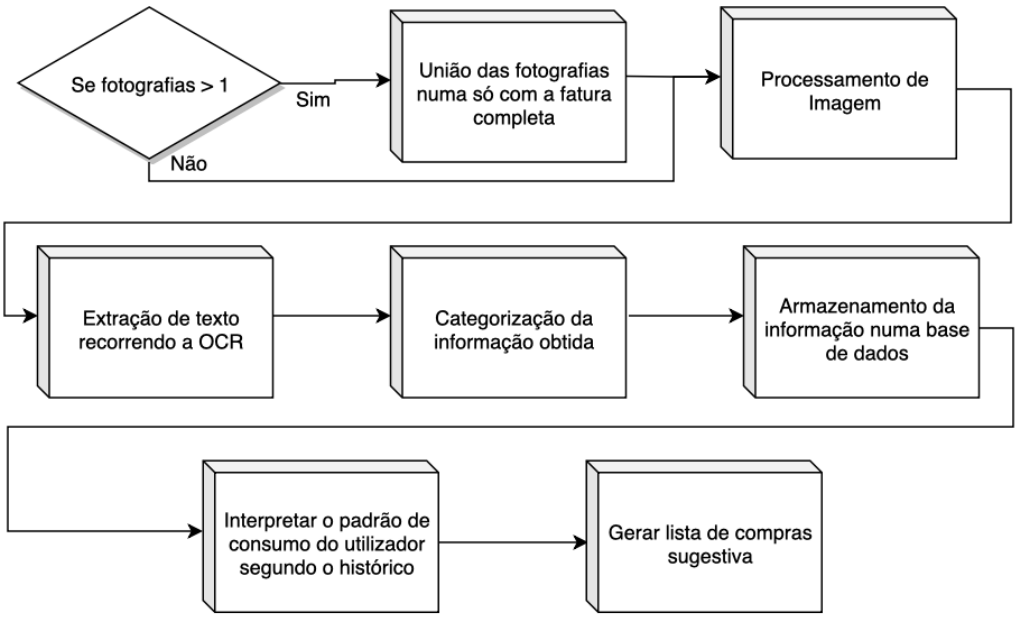

Figura 1.3: Processo Automático de Geração de Lista de Compras

1.3 Motivação

A vida pessoal torna-se cada vez mais complicada de gerir à medida que a responsabilidade de assegurar os bens essenciais para o agregado familiar aumenta. Vamos contudo apercebendo-nos que existe um certo padrão e uma repetição na quantidade de produtos que adquirimos, ou seja, reparamos que acabamos por elaborar listas de compras muito semelhantes sempre que nos deslocamos às compras.

Verificamos, através de pesquisas online, que não existem muitos sistemas que endereçem especificamente este problema, ou que, que consigam tornar a geração de listas de compras mais eficiente em termos de esforço e tempo. Por esse motivo, consideramos muito motivador o desenvolvimento de um sistema que integrasse a utilização de aplicações móveis para automatizar a geração e utilização de listas de compras.

1.4 Estrutura do Documento

Esta dissertação está dividida em cinco capítulos. O Capítulo 1 faz uma introdução ao tema geral do projeto, apresentando o problema e identificando os objetivos a alcançar no desenvolvimento do sistema *Shopping List Automator* para simplificação da criação de listas de compras. O Capítulo 2 apresenta as ferramentas tecnológicas relevantes fulcrais ao desenvolvimento do *Shopping List Automator*, assim como estudos relacionados com o problema. O capitulo 3 foca a especificação e desenvolvimento do sistema *Shopping List Automator*, apresentando inicialmente os requisitos. Posteriormente descreve a arquitetura geral do sistema, bem como os componentes mais relevantes na implementação da solução e a interdependência entre esses componentes. Neste capítulo são ainda detalhas as funcionalidades a que o cliente final tem acesso como utilizando da aplicação móvel. O capitulo 4 apresenta os testes efetuados para avaliar o sistema. Neste capítulo descreve-se a metodologia e critérios de avaliação dos componentes desenvolvidos. Apresenta-se ainda uma análise dos respetivos resultados de avaliação. Por fim, no Capitulo 5, encontram-se as conclusões finais deste projeto de dissertação e apontam-se possíveis desenvolvimentos futuros que permitam evoluir e tornar o sistema *Shopping List Automator* mais robusto e completo.

Capítulo 2

Revisão de Conceitos e Tecnologias

Neste capitulo é referida a importância da simplificação do processo anterior à compra de bens essenciais, e são apresentados alguns trabalhos relacionados com o *Shopping List Automator*. Tendo em conta que o sistema proposto é composto por vários componentes, nomeadamente o processamento de imagem, a extração de texto, a categorização de informação e o padrão de consumo de um utilizador, para cada um destes são mencionadas diferentes técnicas, e propostas que são importantes para atingir o objetivo principal do projeto desta dissertação. São também apresentadas bibliotecas vocacionadas para tratamento de imagem, reconhecimento ótico de caracteres (OCR), e para comparação de similaridade entre *strings*.

A realização de compras de bens essenciais para consumo tais como alimentação, higiene pessoal, produtos de limpeza, utensílios para o lar, vestuários, entre outros, é uma tarefa rotineira, e necessária em todos os agregados familiares. Existe a necessidade de se repor o que é consumido e sendo algo rotineiro, segundo Whatkins, presume algum tipo de planeamento (Watkins, 1984).

Uns têm por hábito realizar essas compras em diferentes mercados locais, outros realizam em superfícies maiores. Uns deslocam-se variadas vezes durante um mês efetuando compras pequenas, outros com menos frequência e em quantidades maiores. Cada um vai perante a sua disponibilidade mas a decisão final está sempre associada ao período que cada um entende como mais adequado para visitar o supermercado (Pashigian et al., 2003). Esta tarefa deve exigir um mínimo de planeamento para que a consigamos efetuar de forma organizada e eficiente, nem que seja através de um rápido raciocínio lógico e recorrendo à memória humana. Existem diversas soluções móveis relacionadas com a geração automática de listas de compras de acordo com o planeamento de refeições. A aplicação desenvolvida por Bird et al. é um dos casos, que apresenta o equilíbrio nutricional do seu carrinho de compras (Bird et al., 2013). Um estudo de dois meses realizado pelo desenvolvedor demonstrou que a aplicação levou a alterações significativas nos participantes, e melhorias no equilíbrio nutricional das suas dietas.

Planear e compor uma lista de compras é muito vantajoso para o consumidor. A

escolha ponderada do supermercado é também um ponto importante no planeamento. O tipo de consumidor que tem em conta estes dois fatores, despende de menos tempo nas compras e fá-las ao encontro das suas verdadeiras necessidades (Tang et al., 2001).

Segundo um estudo de Fernando Neves dos Santos foram feitos inquéritos a consumidores nos espaços comerciais logo após o ato da compra, questionando sobre a sua preparação para a compra (dos Santos, 2009). Verificou-se que os indivíduos do sexo feminino, ou indivíduos com filhos, auxiliam-se mais das listas de compras do que indivíduos do sexo masculino. Refere ainda que os clientes que recorrem a listas de compras dão maior valor a promoções e às marcas dos produtos de uma forma mais acentuada que os restantes.

Perante os resultados apresentados por Angelo verifica-se que a ausência de lista de compras leva a maiores erros na gestão de gastos com as compras (de Angelo et al., 2003). Isto que faz também com que o cliente fique mais exposto à compra por impulso, ou seja, não é tão racional. Entendeu-se através dos mesmos resultados que os gastos são muito sensíveis ao tempo em que as pessoas permanecem na loja. Além de que uma compra efetuada sem apoio de uma lista de compras, leva a um maior tempo de permanência na loja. Consequentemente, resulta numa maior probabilidade de haver maior gasto na compra.

O contínuo avanço da tecnologia resultou numa maior facilidade de grande parte da população no acesso a aplicações web e aplicações móveis. Estas aplicações para *smartphones* têm uma enorme utilidade e facilidade de uso (Yang, 2013). Consequentemente, existem atualmente diversas aplicações que tentam facilitar de alguma forma o utilizador antes e durante o processo de compra. Grandes superfícies como o caso da SONAE, conscientes de que ferramentas deste tipo facilitariam o cliente e tendo como objetivo fidelizar os seus clientes, têm constantemente redesenhado e evoluído as suas aplicações com mais funcionalidades. como é o caso do *Continente Online* (COL), Continente Siga e Cartão Continente (Santos, 2008). A SONAE tem a noção de que as soluções através de aplicativos têm como maior público alvo as camadas mais jovens em que o *smartphone* é atualmente essencial à vida quotidiana, porém é esta a tendência e cada vez mais se alastra para as camadas mais adultas (Pires, 2021).

2.1 Ferramentas Tecnológicas

Existem algumas bibliotecas que facilitam o processamento de imagens, a extração de texto através de reconhecimento ótico de caracteres, conversão de formatos de ficheiros, correção de palavras e a comparação de strings recorrendo a diferentes métricas. De seguida são revistas algumas delas.

O *OpenCV* é uma biblioteca de código aberto que é composta por uma grande coleção de algoritmos destinados principalmente ao processamento de imagens em tempo real

e ainda a análise de vídeo (Bradski, 2000). Desenvolvida pela Intel, é implementada recorrendo à linguagem de programação C/C++ e tem interfaces disponíveis para muitas outras linguagens de programação tais como Ruby, Matlab, Python, entre outras. Já existe também uma versão mais pequena desta biblioteca para desenvolvimento Android, iOS e outras *frameworks* multiplataforma como é o caso do Flutter.

O *Tesseract* é uma biblioteca também de código aberto, desenvolvida pela Google, para reconhecimento de texto (Kay, 2007). Desenvolvido por Hewlett Packard em 1990, foi apresentada no Teste Anual de Precisão OCR de 1995, onde demonstrou que tinha muito potencial em comparação à concorrência na época. Só em 2005 é que realmente passou a estar disponível em código aberto. A ferramenta permite detetar texto de diversas linguagens, tendo recentemente recebido suporte para a língua portuguesa. Esta retorna o texto identificado pela mesma ordem apresentada na imagem importada inicialmente. À semelhança do *OpenCV* também tem uma versão adaptada pronta a ser usada em *frameworks* para desenvolvimento de aplicações móveis.

A biblioteca *Enchant* é usada para verificar a ortografia de palavras e sugerir correções em casos que contenham erros ortográficos (Merejkowsky, 2006). Esta biblioteca mãe engloba muitas outras bibliotecas populares destinadas à verificação ortográfica incluindo: *ispell*, *aspell* e *MySpell*. É vantajosa no sentido em que o utilizador não tem de programar a interface de cada uma das bibliotecas. Estas lidam com vários idiomas nomeadamente a língua portuguesa. Está disponível para as linguagens Rust, Go e Python.

A *StrSimpy* é uma biblioteca disponível para Python que implementa diferentes medidas de similaridade e distância de strings. Composta por uma dúzia de algoritmos que permitem uma fácil comparação de strings, porém apenas os mais relevantes para o trabalho proposto desta dissertação serão abaixo apresentados.

Tabela 2.1: Principais Características de Algoritmos da biblioteca *StrSimpy*

Nome		Normalizada?	Métrica?	Custo	Uso típico
Levenshtein	distância	Não	Sim	O(m*n) [1]	
Levenshtein Normalizada	distância similaridade	Sim	Não	O(m*n) [1]	
Levenshtein com Pesos	distância	Não	Não	O(m*n) [1]	OCR
Jaro-Winkler	similaridade distância	Sim	Não	O(m*n)	Correção ortográfica

Na tabela 2.1 verificamos algumas das características de alguns algoritmos apropriados para a avaliação de resultados, em forma de *strings*, do *Shopping List Automator* e abaixo são explicitadas as características e cada um dos algoritmos:

No caso da similaridade de strings os algoritmos definem uma semelhança entre as strings em que zero significa que as strings são completamente diferentes. Para a distância de strings os algoritmos definem uma distância entre as strings em que zero significa que

as strings são idênticas, como o caso do algoritmo de Levenshtein, por exemplo. O valor da distância máxima depende do algoritmo.

Geralmente os algoritmos que implementam similaridade normalizada de strings também implementam a distância normalizada de *strings*. A similaridade é calculada subtraindo a distância a 1, mas existem algumas exceções.

A distância Levenshtein entre duas palavras é o número mínimo de edições de um único caratere, sejam elas inserções, eliminações ou substituições, necessárias para transformar uma palavra na outra. É uma distância métrica de strings, que usa programação dinâmica (algoritmo de Wagner-Fischer).

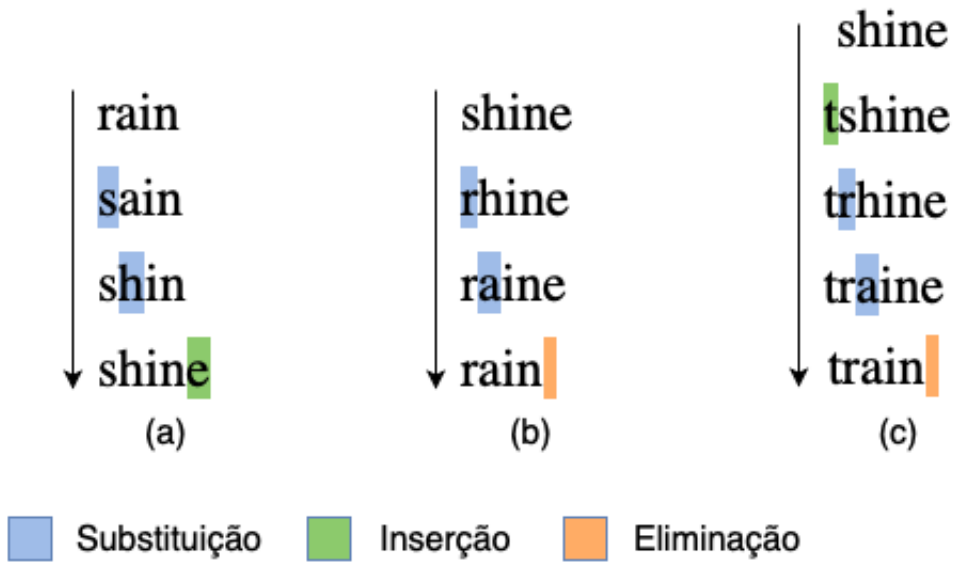

Figura 2.1: Exemplos de determinação da distância Levenshtein (Devopedia, 2019)

Na figura 2.1 verifica-mos que para os casos (**a**) e (**b**) a distância Levenshtein é de três pois é esse o número de operações necessárias para transformar a palavra inicial na final (Devopedia, 2019). No caso da transformação em (c) poderia-se utilizar cinco operações para obter a palavra "train", substituindo cada letra da palavra inicial pela letra final da mesma posição. Além dessa opção é possível realizar a transformação em menos operações. Neste caso, com quatro operações poder-se-ia efetuar a transformação reaproveitando a letra "i"e "n"que já estão alinhadas. Neste exemplo a distância Levenshtein é quatro pois é o número mínimo de operações necessárias.

A distância normalizada de Levenshtein é calculada como a distância de Levenshtein dividida pelo comprimento da string mais longa. O valor resultante está sempre dentro do intervalo [0,0; 1,0]. Esta similaridade é calculada subtraindo a distância normalizada a 1.

O algoritmo de Levenshtein com Pesos permite definir diferentes pesos para diferentes substituições de caracteres. Normalmente é usado para aplicações de OCR. No OCR, o custo de substituição de um *P* por um *R* é menor do que o custo de substituição de um *P* por um *M*, porque, do ponto de vista do OCR, *P* é semelhante a *R*. Também pode ser usado

para corrigir automaticamente a escrita a partir de um teclado. Neste exemplo, o custo de substituição de *E* e *R* é menor, por exemplo, por estarem próximos um do outro num teclado *AZERTY* ou *QWERTY*, portanto, a probabilidade do utilizador escrever incorretamente os caracteres é maior. Este algoritmo tem a desvantagem de não ser normalizado e de não disponibilizar o calco de similaridade que facilitaria na comparação direta entre strings por ser um número compreendido entre um e cem, que pode ser tratado como uma percentagem.

Jaro-Winkler é uma distância de edição de strings desenvolvida para a área de deteção de duplicados (Winkler, 1990). A métrica de distância Jaro – Winkler foi projetada a pensar em strings curtas, como é o caso de nomes de pessoas e para detetar erros de escrita.

$$\mathrm{sim_j} = \begin{cases} 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) \end{cases} \tag{2.1}$$

Acima temos a fórmula 2.1 para calcular a similaridade de Jaro, em que *sim j* representa a similaridade em questão, *sim i* representa o comprimento da *string*, *m* é o número de caracteres iguais entre si e *t* é o número de transposições necessárias para transformar uma *string* na outra.

$$\mathrm{sim}_w = \mathrm{sim}_j + \ell p\left(1 - \mathrm{sim}_j\right) \tag{2.2}$$

A similaridade de Jaro-Winkler é uma modificação da similaridade de Jaro, como demonstrado na fórmula 2.2. A variável *l* representa o maior número de caracteres iniciais da *string s1* que correspondem aos caracteres iniciais da *string s2*. Já o *p* é uma constante especificada anteriormente, Winkler no artigo de 1990 definiu essa constante como 0.1 (Winkler, 1990).

O Flutter é uma *framework* de código aberto desenvolvida pela Google a pensar no desenvolvimento multiplataforma de aplicações móveis (Miola, 2020). Neste momento, com o mesmo código fonte e de forma nativa, permite obter a aplicação em versões para Android, iOS, aplicação web e para os sistemas operativos Windows, MacOS e Linux. Baseia-se na linguagem de programação Dart, também desenvolvida pela Google, que é orientada a objetos e baseada em classes.

2.2 Soluções Relacionadas com o *Shopping List Automator*

Reconhecendo o aumento da exigência dos consumidores, existem atualmente diversas aplicações móveis que têm como objetivo apoiar o consumidor nesta tarefa que é a gestão das compras a realizar, girando à volta da útil ferramenta que é a lista de compras.

A popular aplicação Foodie.fi, em funcionamento na Finlândia permite os utilizadores planearem refeições e a partir das mesmas obter as respetivas listas de compras (Tukkinen and Lindqvist, 2015). A empresa desta aplicação realizou um caso de estudo em que investigaram o motivo a forma como as pessoas utilizam este tipo de aplicações. Os três motivos principais dos interrogados para o uso das aplicações foram: o planeamento holístico das refeições, o facto de servir como auxílio de memória e a possibilidade de comprar online, devido à impossibilidade de despender de tanto tempo no supermercado.

Um projeto apresentado por Silva é composto por uma aplicação móvel, para dispositivos Android que trata de gerir os produtos que estão em falta na casa do indivíduo, aliado a um sistema de deteção de consumo de produtos (Silva, 2020). Refere a importância em se focar em torno de uma lista de compras para que o consumidor tenda a ser mais controlado, organizado e consiga ser mais eficiente. O sistema de deteção de consumo envolve uma câmara de vídeo instalada no caixote do lixo de uma forma estratégica para que consiga ler os códigos de barras das embalagens dos produtos vazios que ali são colocados, seguido de análise à imagem capturada. A informação relativa aos produtos reconhecidos é enviada à aplicação, o sistema reconhece o que é que já foi consumido e passa, teoricamente, a faltar na despensa do utilizador. Este sistema baseia-se somente no consumo e não tem consciência do que o utilizador vai adquirindo do supermercado. Para o funcionamento deste sistema de análise foi utilizado a linguagem de desenvolvimento Python, a biblioteca *OpenCV* e a *framework* NestJS. A aplicação móvel foi desenvolvida no software Android Studio.

Existe um projeto muito semelhante ao referido anteriormente, apresentado por Khan, com a diferença que este utiliza recipientes inteligentes, e foca-se no uso à *Internet of Things* (IoT) (Khan, 2019). Os sensores de proximidade automaticamente detetam a quantidade atual de um dado produto e envia a informação para um *hub* que usa *Bluetooth Low Energy* (BLE). Este *hub* transfere a informação via *cloud* para o *smartphone* com a aplicação móvel. Tudo isto para que o utilizador não tenha a preocupação em estar atento a que produtos necessitará de comprar. Na figura 2.2 está representada a arquitetura do sistema de Khan em que os recipientes inteligentes, representados por boiões de vidro (**a-c**), contêm cada um um dispositivo eletrónico na tampa que por sua vez se conectam sem fios ao *hub* (**d**) principal utilizando BLE. O *hub* principal mostra informações como a quantidade de produtos, o estado da conexão e níveis de bateria de cada recipiente e através de Wi-fi, por estar conectado ao *router* de casa (**e**), envia estas informações via *cloud* (**f**) para a aplicação móvel no telemóvel (**g**).

Uma solução móvel desenvolvida por Harsha Jayawilal and Premeratne consiste em diversos módulos: uma lista iterativa, um seletor de supermercado e um algoritmo *a priori* para recomendação de produtos (Harsha Jayawilal and Premeratne, 2017). Na lista iterativa o utilizador consegue adicionar/remover/trocar produtos. O seletor de supermercado analisa o consumidor indicando-lhe quais os supermercados mais próximos em que

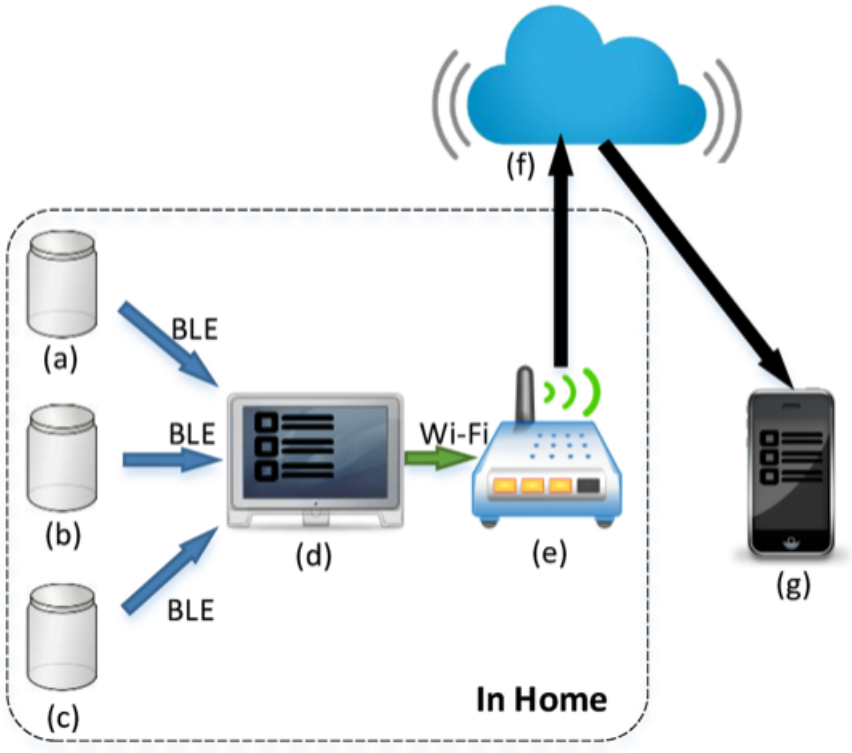

Figura 2.2: Arquitetura do Sistema de Recipientes Inteligentes (Khan, 2019)

este possa comprar maioria dos produtos no mesmo estabelecimento. O algoritmo de recomendação indica que produtos o utilizador poderá se ter esquecido de colocar na lista de compras. O autor afirma que o algoritmo *a priori* é ideal para identificar padrões de consumo do cliente. Isto é possível recorrendo aos dados de vendas em pequenas lojas de retalho. Tem a vantagem do algoritmo calcular mais conjuntos de produtos consumidos com mais frequência. Torna-se muito vantajoso principalmente quando a base de dados é menor e a geração de produtos mais frequentes é melhor se houverem mais sugestões. A figura 2.3 apresenta as páginas principais da aplicação móvel. Verifica-se a existência de um sistema de autenticação para armazenar as informações de diferentes utilizadores e também a página onde o utilizador visualiza, cria e edita listas de compras. Tem também uma página com um mapa onde pode descobrir quais os supermercados mais próximos e quanto ficaria o custo total da compra para uma das listas de compras que compôs.

Existem outras soluções que estão focadas em fornecer informações adicionais ao utilizador sobre a lista de compras que ele já compôs. O utilizador é guiado, por exemplo, na decisão de qual supermercado deverá se deslocar. Um sistema proposto por Cobalchini é composto por uma aplicação Android que permite ao utilizador selecionar os produtos que pretender, e no fim faz uma comparação de preços entre supermercados (Cobalchini, 2018). O sistema contém um sistema de *login* para armazenar as listas de cada utilizador, mostra as estatísticas de gastos por categoria e usa a padronização *Global Trade Item Number* (GTIN) para a identificação única de cada produto. Deste modo, ele consegue

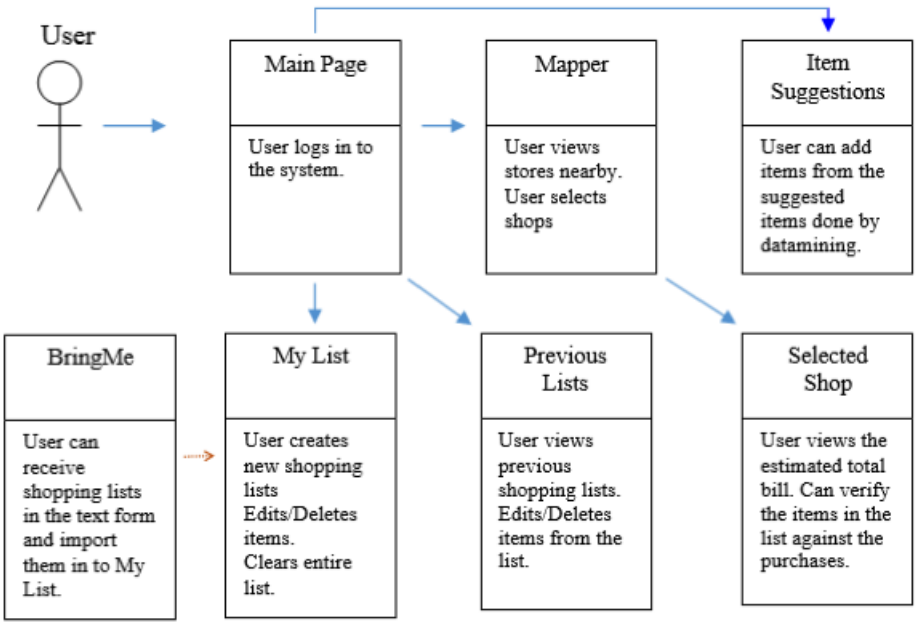

Figura 2.3: Mapa de Navegação do Smart Shopping List (Harsha Jayawilal and Premeratne, 2017)

associar exatamente o mesmo produto entre supermercados e posteriormente apresentar os diversos preços ao consumidor. Para facilitar o consumidor a encontrar o produto que pretende inserir na sua lista de compras era possível ler o código de barras de um produto através da câmara do *smartphone*.

Por lei é obrigatório a emissão de faturas numa aquisição de bens ou prestação de serviços independentemente se os destinatários o solicitaram. O mesmo se aplica nas típicas compras de supermercado, obtemos sempre uma fatura em papel ou então, podemos agora receber apenas em formato eletrónico, via e-mails ou na aplicação móvel do supermercado.

Nos últimos anos tem havido uma grande evolução de tecnologias que facilitam a extração de texto a partir de imagens e, consequentemente, tem havido um esforço no sentido de extrair informação de faturas de forma a ser acedida via digital posteriormente.

Inclusive, em 2019, houve um concurso que consistia na extração via OCR a recibos e recolha de informação importante (Huang et al., 2019). Na publicação relativa a esse desafio relata que apesar de muitos trabalhos terem sido publicados ao longo dos anos sobre análise de documentos administrativos, a comunidade tem avançado de forma relativamente lenta, sendo que a maioria das bases de dados continuam secretas. Este concurso era composto por três tarefas: localização do texto numa fatura digitalizada, extração via OCR e obtenção de informação relevante. Infelizmente não estão descritas as técnicas e tecnologias que foram utilizadas pelos participantes.

O trabalho apresentado por Kumar expõe uma aplicação de OCR que pode extrair informações de imagens de recibos, armazenar a mesma como texto processado por má-

quina de forma organizada para facilitar o acesso (Kumar et al., 2020). A aplicação funciona mesmo na presença de marcas d'água nos recibos. Os autores recorreram ao *OpenCV* para o processamento das imagens e *Tesseract*, um software de OCR, para o reconhecimento óptico de caracteres e extração de texto. A imagem é primeiramente processada usando *OpenCV* para a remoção de sombras ou outras marcas. Após este passo a imagem processada é passada para o *Tesseract* para a extração de texto. No fim é feita uma pesquisa para encontrar somente o valor total gasto e a data de emissão, recorrendo ao processamento de *strings*. Os autores afirmam que as sombras nas imagens podem reduzir significativamente a eficiência do *Tesseract*, daí ser importante a remoção de ruído da imagem antes da extração de texto. Além de reconhecerem que o sistema proposto poderia ser hospedado num servidor e depois ter uma aplicação móvel conectada ao mesmo, devido aos *smartphones* de hoje em dia terem poder de processamento suficiente, optaram por implementar todo o sistema dentro de uma aplicação Android. Para este feito recorreram ao conjunto de ferramentas *Tesseract-android-tools SDK* para o reconhecimento de texto e ao *OpenCv4Android SDK* para o processamento de imagem.

Na figura 2.4 temos o exemplo de uma fatura curta capturada com a câmara do *smartphone*, que por sua vez foi importada para a aplicação móvel, que após efetuar a extração de texto imprime o resultado numa página simples (Figura 2.5). Verifica-se que o texto extraído na segunda imagem está segundo a ordem apresentada na fatura física na vertical e que alguns dos caracteres não foram corretamente identificados como por exemplo na palavra "DUPLCATER"apresentada na quarta linha da figura 2.5.

Uma abordagem muito semelhante à anterior foi apresentada por Sidwha em que o objetivo final é o mesmo, obter texto extraído de imagens em formato digital (Sidhwa et al., 2018). Neste artigo é mais detalhado em certos aspetos tais como: avisar a necessidade da remoção das bordas escuras, o algoritmo utilizado para deteção das bordas e o pré-processamento aplicado. O sistema alerta aquando há a necessidade de se remover as bordas manualmente para que não sejam erradamente interpretadas como caracteres pelo *Tesseract*. Para a deteção da fatura numa determinada fotografia é usado um algoritmo chamado *Canny Edge* para a deteção de bordas. Antes da extração de texto trabalha com a imagem em escala de cinzentos seguido de um simples *threshold*. Os autores concluem que a técnica aplicada tem algumas limitações, como por exemplo reconhecer texto em faturas escritas manualmente visto que o *Tesseract* usa uma técnica de segmentação de texto que reconhece o primeiro caractere na linha e de acordo com a posição desse lê a linha inteira respetiva e como na escrita manual normalmente não está tudo alinhado, tem tendência a falhar. Este trabalho também recorreu ao *OpenCV* e ao *Tesseract*.

Uma investigação realizada por Robert and Talbot analisou o uso de *Single Image Super-Resolution* (SISR) para construir imagens de alta resolução a partir de imagens reais enviadas por clientes de empresas e avaliar o impacto das mesmas no desempenho da extração de texto via OCR (Robert and Talbot, 2020). Utilizando métodos integrados de

Figura 2.4: Exemplo de Fatura Importada Para a Aplicação Móvel (Kumar et al., 2020)

avaliação de desempenho, comprovaram que o uso de SISR pode melhorar significativamente o desempenho da extração de texto em casos de imagens que o reconhecimento foi baixo devido a baixas resoluções e em imagens com muito ruído. Por outro lado, também pode prejudicar o desempenho nos casos em que os recibos já tinham boa qualidade e tinham bons resultados na extração do texto. Concluindo, de uma forma geral, uma boa resolução de imagem das faturas permite obter melhores resultados mas até um certo ponto, resoluções altas demais como é o caso de super resoluções SISR aplicadas em fotografias originais de qualidade já aceitável, poderá resultar numa pior extração de texto.

O mercado de aplicações móveis contém inúmeras soluções que ajudam as pessoas a monitorizar a ingestão de alimentos ou fornecer sugestões para a perda de peso e conduzir o utilizador a ter uma alimentação mais saudável. Com o objetivo de consumir menos tempo ao utilizador final, em comparação à concorrência, foi desenvolvida uma aplicação

Figura 2.5: Texto Extraído da Fatura na Figura 2.4 (Kumar et al., 2020)

Android a pensar em pessoas com mais de setenta anos, especialmente para a a população que vive em locais rurais ou com pouco conhecimento técnico (Sainz-De-Abajo et al., 2020). Permite que estes introduzam os alimentos que adquirem através das faturas dos supermercados, facilitando-lhes o processo de inserção de dados. O algoritmo OCR extrai a informação das faturas e o utilizador pode manualmente acrescentar mais produtos à sua base de dados de compras, pesquisar pelos seus produtos comprados recorrendo a filtros, e ver estatísticas e sugestões alimentares (Figura 2.6). Testaram diversos serviços de páginas web para o processamento de OCR, ou seja, através de API's, tais como Newocr.com, Onlineocr.net, Ocr.space, e Tess4j juntamente com *Tesseract*, mas no fim acabaram por recorrer ao Google Vision, uma API da Google paga.

A Google Vision consegue automaticamente reconhecer vários formatos de fatura e convertê-los em números. Esforços manuais são necessários apenas para a verificação e os dados retornados de forma estruturada permitem aos utilizadores obter em formato JSON, de forma facilitada e digital, o conteúdo presente nas faturas importadas e manipulá-lo como bem entenderem (Cheng, 2021). Tem muitas vantagens, mas com algumas contradições. Para o tipo de tarefa referido como é o facto de não ser muito robusta em imagens com ruído (Hosseini et al., 2017).

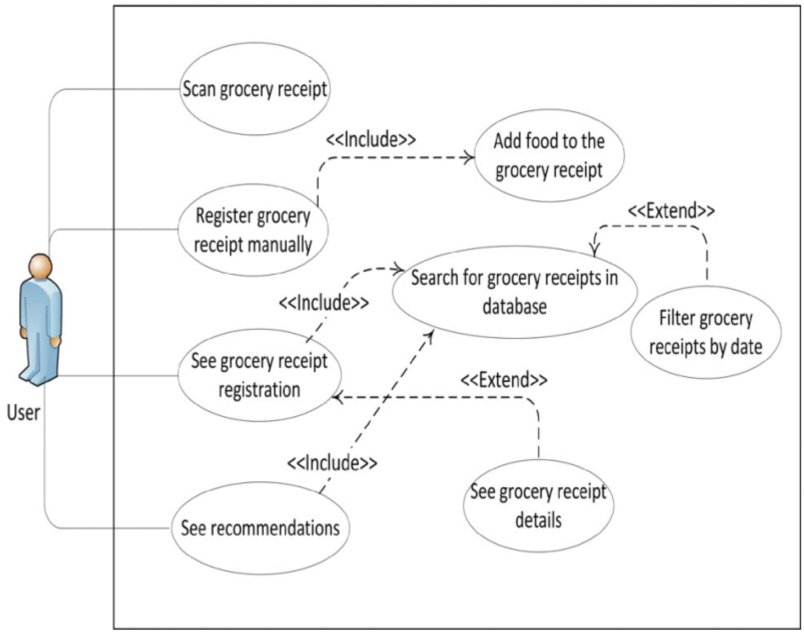

Figura 2.6: Diagrama de Casos de Uso da Aplicação FoodScan (Sainz-De-Abajo et al., 2020)

Em faturas antigas ou mal impressas o reconhecimento de carateres torna-se mais complexo. Nesse sentido, foi apresentado um projeto que recorre a redes neuronais (*Deep Neural Network* (DNN)) usando *Inception V3* para treinar e aplicar OCR (Wei et al., 2018). A rede *Inception V3* é treinada com 53.342 imagens com ruído com caracteres, obtidas a partir de recibos e jornais, e conseguiram melhor reconhecimento de texto com essa rede neuronal, com cerca de 21,5 por cento de redução de erros em comparação a outras soluções de OCR.

Para tentar contornar a criação de regras específicas para cada tipo de documento na categorização do texto extraído, têm sido publicados alguns trabalhos que tentam aplicar modelos que exploram o contexto semântico em sequências de texto. Um modelo chamado de *Convolutional Universal Text Information Extractor* (CUTIE) aplica redes neuronais convolucionais em texto organizado num estilo tipo tabela em que cada parte de texto é associado a conotações semânticas (Zhao et al., 2019). Esta solução que não necessita que se realize nada antes da sua aplicação ou pós processamento, é fácil de ser treinada e requer um conjunto de dados menor que as soluções de redes neuronais mais comuns. Explora essencialmente três fatores para conseguir o pretendido: i) relação espacial entre partes de texto, ii) a informação semântica do texto em si e iii) o mecanismo de mapeamento posicional em estilo grelha.

(a)

	BIEVENIDO	AL	SERVICIO	
TAXI	DEL	A.P.C	DE	MADRID
EUGENIO		DOMINGUEZ		GALLEG
N°	LICENCIA:			14670
...				
TOTAL	(I/IVA):	12.45		€
**	I.V.A.		INCLUTIDO	**
DIST.	SERVICIO:		5.9	km
TARIFAS	TR:			1
HORA	INICIO:			08:59
HORA	FINAL:			09:23
-DATOS	CLIENTE:			
-ORIGEN:				
-DESTINO:				

(b)

(c)

	BIEVENIDO	AL	SERVICIO	
TAXI	DEL	A.P.C	DE	MADRID
EUGENIO		DOMINGUEZ		GALLEG
N°	LICENCIA:			14670
...				
TOTAL	(I/IVA):	12.45		€
**	I.V.A.		INCLUTIDO	**
DIST.	SERVICIO:		5.9	km
TARIFAS	TR:			1
HORA	INICIO:			08:59
HORA	FINAL:			09:23
-DATOS	CLIENTE:			
-ORIGEN:				
-DESTINO:				

Figura 2.7: Framework do Método Proposto (Zhao et al., 2019)

A figura 2.7 apresenta em (**a**) o processo de extração de texto preservando o posicionamento relativo de cada palavra entre s. Em apresenta (**b**) o envio desta informação para a rede neuronal convolucional, e por fim em (**c**) temos uma inversão do mapa obtido com a informação chave que a rede detetou para que entendamos visualmente o que foi selecionado.

Na tabela 2.2 apresento algumas soluções comerciais que permitem a extração de texto através de OCR a partir de imagens de faturas. Estas soluções podem ser utilizadas pelos clientes via interface gráfica na aplicação web e/ou através da própria API. Comparo aqui as diferentes funcionalidades que cada plataforma permite e ainda os seus custos.

O software Klippa (Kli, 2021) é um dos serviços mais completos e, dos apresentados

Tabela 2.2: Principais Características de Aplicações para Extração de Informação de Faturas

Nome	App Web	App Móvel	API	Versão Teste	Versão Paga	NLR	Permite treinar modelo	Armazenamento Cloud	Exporta em JSON, PDF, CSV
Klippa	✓	✓	✓	✓	✓		✓	✓	✓
Nanonets	✓		✓	✓	✓		✓	✓	✓
Hypatos	✓		✓	✓	✓		✓	✓	✓
Rossum	✓		✓		✓		✓		✓
Turicode	✓		✓		✓		✓		✓
KlearStack	✓		?		✓	✓	✓		✓
Abby	✓		✓		✓		✓		✓
Typless			✓	✓	✓		✓		✓

na tabela 2.2, é o único com aplicação móvel e dos poucos que afirma conseguir extrair texto na língua portuguesa. Esta aplicação permite armazenar os documentos em *cloud* e partilhar os mesmos entre utilizadores pela própria aplicação visto que o objetivo do software é facilitar a relação dos trabalhadores independentes com os seus contabilistas. Supondo que os dois tipos de utilizadores se registam e trocam os documentos entre si pela aplicação, de uma forma mais organizada. Também pretendem facilitar a gestão das despesas das empresas e processamento das suas faturas, apoiando em automatização através da extração de informação. Disponibilizam uma versão de teste gratuita, porém limita o reconhecimento de texto automático à sua versão paga, disponibilizando apenas algumas funções para armazenar ficheiros e partilha dos mesmos.

Nanonets é uma aplicação que procura reduzir custos e mão de obra necessária à análise e verificação de faturas (Nan, 2021). Na aplicação é possível importar imagens de faturas e depois da análise, com recurso à interface gráfica, é pedida ao utilizador que manualmente confirme se os campos identificados foram corretamente categorizados. Se for o caso, o utilizador deve ainda selecionar outras informações que não foram destacadas e manualmente identificar o tipo de informação que é, melhorando o modelo para o tipo de documento em questão, favorecendo os resultados da análise a uma posterior fatura do mesmo tipo. O software afirma que o utilizador poderá pedir aos desenvolvedores da Nanonets que ajudem a ajustar um modelo específico para uma determinada fatura. Testamos com fotografias de faturas curtas em português do supermercado Continente, e apenas dois campos foram identificados automaticamente da fatura, o nome do supermercado e a data. A aplicação recorre ao sistema *CUTIE* (Zhao et al., 2019).

A Hypatos (Hyp, 2021) tem muitas semelhanças à Nanonets tirando o facto da versão gratuita não processar faturas em português. Rossum (Ros, 2021) utiliza um sistema de captura de dados cognitiva onde as redes neuronais sugerem candidatos para a identifi-

cação um certo campo, baseando-se num sistema de pontuações para a decisão final. As restantes aplicações Turicode (Tur, 2021), KlearStack (Kle, 2021) e Abby (Abb, 2021) são muito semelhantes às anteriores nos objetivos, pois permitem a inserção de faturas e exportação da informação obtida em diversos formatos, mas aparentam ser menos robustos na categorização da informação; não têm armazenamento *cloud*; não disponibilizam versões de teste; o preço é definido à medida do cliente e destes três o Turicode é o único que funciona com a língua portuguesa. A Klearstack recorre à técnica *Named Entity Recognition* (NLR) que consiste no reconhecimento e extração de entidades contidas no texto, tais como localizações, nomes, organizações, pessoas, entre outras. A técnica NLR utiliza regras gramaticais básicas para o efeito e a sua utilização não é referida pelos restantes softwares apresentados. O Typless (Typ, 2021) não dispõe sequer de uma aplicação web com interface gráfica, limita-se somente à disponibilização de uma API diversificada para os programadores utilizarem como bem entenderem. Funciona com faturas em português e tem um plano em que o utilizador paga conforme a utilização, ou seja, conforme o número de documentos que processar.

No que toca à automatização da previsão das compras de um utilizador com base no seu histórico é importante destacar o trabalho de Kuratli (Kuratli, 2015). Desenvolveu uma aplicação Android onde é possível criar manualmente listas de comprar compostas pelo nome de cada produto e respetiva quantidade. O grande foco deste trabalho foi o algoritmo de previsão de futuras compras. O algoritmo determina quão regularmente cada produto é comprado e verifica se já passou tempo suficiente desde a última compra. Para obter esta informação, para cada produto é calculada a média a da diferença de tempo entre cada par de recibos contendo o determinado produto. De seguida o algoritmo determina a diferença de tempo entre o momento em que a função é chamada e a última vez que o utilizador comprou um dado produto. Para verificar se já passou tempo suficiente o algoritmo calcula o rácio segundo a fórmula apresentada na fórmula 2.3.

$$r = \frac{t}{a} \tag{2.3}$$

O rácio r aumenta linearmente à medida que o tempo aumenta desde a última compra, e os produtos serão propostos para compra quando atingirem um rácio próximo de um.

Tabela 2.3: Cálculo de Rácio de Produtos Exemplo (Kuratli, 2015)

	Coca-Cola	Iced Tea	Leite	Batatas Fritas
a	128	256	162	228
t	120	120	120	120
r	0.94	0.47	0.74	0.53

Na tabela 2.3 são apresentados os valores para quatro produtos exemplo aplicando a fórmula 2.3, em que se verifica que a Coca-Cola seria um produto a ser recomendado

numa próxima compra, visto que o seu rácio está próximo de 1. Este sistema tem em consideração três fatores durante a sua utilização com o objetivo de melhorar a previsão: a quantidade média de produtos previstos em cada lista de compras, quantos produtos previstos é que realmente fizeram parte da compra final e a percentagem de previsões corretas de segundo a quantidade de produtos previstos totais (Kuratli, 2015).

Um artigo de Cumby et al. descreve um sistema de um assistente inteligente de compras desenhado para um *tablet* que seria utilizado pelos consumidores durante as suas compras, apoiado no próprio carrinho de compras (Cumby et al., 2005). O sistema recorre a algoritmos de aprendizagem de máquina para prever a uma lista de compras para o consumidor e apresenta-a no ecrã do *tablet*, assim como promoções personalizadas que à partida o consumidor teria interesse em aproveitar e seriam estratégicas também para o supermercado. Isto sempre de acordo à informação das compras passadas do utilizador associadas ao seu cartão de cliente. Para melhoria das técnicas de aprendizagem máquina na previsão dos produtos que o consumidor irá comprar, focaram-se nas categorias de produtos que o consumidor adquiriu na última compra, nos produtos da última lista de compras e nos produtos mais comprados pelo consumidor em todo o seu histórico.

Capítulo 3

Sistema de Lista de Compras Automáticas

O *Shopping List Automator* procura fornecer listas de compras automáticas ao utilizador baseadas nas suas compras passadas, pressupondo que as faturas foram sendo inseridas no sistema. Pretende-se que o sistema seja de fácil utilização e que todo o processo complexo relativo à análise das faturas importadas possa ficar impercetível ao utilizador. Caso contrário o objetivo final deste sistema não seria cumprido, que no fundo é poder simplificar o processo do planeamento das compras.

Neste Capítulo são apresentados os requisitos do sistema, seguido da arquitetura em que é explicitado que está dividido em duas partes. A primeira parte explicita a importância do servidor que detém todo o processamento pesado e a segunda é a aplicação móvel. Na aplicação móvel, disponível para Android e iOS, o utilizador pode autenticar-se, importar as faturas, visualizar dados relativos a essas faturas e visualizar listas de compras geradas automaticamente para si. Na implementação ambas as componentes estarão descritas separadamente.

O servidor opera em seis fases para conseguir gerar as listas de compras automáticas. Na primeira transforma as faturas importadas que estejam num ficheiro de formato PDF num ficheiro de imagem ou no caso de receber mais que uma fotografias tenta uni-las numa só. Na segunda fase aplica algumas transformações à imagem anterior para melhorar os resultados da terceira fase em que ocorre a extração de texto recorrendo a OCR. Na quarta fase o texto obtido é filtrado e categorizado, seguindo-se do armazenamento numa base de dados, a quinta fase. Por último, na sexta fase, quando o utilizador requisitar uma lista de compras automática, é feita a análise aos consumos passados do utilizador e gerada a lista de compras.

3.1 Requisitos

Nesta secção estão identificados e descritos os requisitos funcionais, não funcionais e os requisitos de sistema do *Shopping List Automator*, cumpridos com êxito.

3.1.1 Requisitos Funcionais

Os requisitos funcionais retratam o que o *Shopping List Automator* deverá ser capaz de executar, em termos de tarefas e serviços. Os requisitos funcionais são os seguintes:

ID: RF1.1
Título: Servidor
Descrição: O sistema deverá contar com um servidor com uma API disponível que permita chamar todas as suas funções.

ID: RF1.2
Título: Aplicação Móvel
Descrição: O sistema deverá contar com uma aplicação móvel que recorrerá à API do servidor.

ID: RF1.3
Título: Autenticação
Descrição: O sistema deverá permitir o registo e o *login* de utilizadores.

ID: RF1.4
Título: Importar Faturas
Descrição: O sistema deverá permitir a importação de fotografias ou de ficheiros em formato PDF a partir da aplicação móvel, que as passará para o servidor através da API.

ID: RF1.5
Título: Pré Processamento de Imagem
Descrição: O sistema deverá conseguir aplicar um tratamento à imagem antes da extração de texto, para que esta última possa ser executada com sucesso e obtenha melhores resultados.

ID: RF1.6
Título: Extração de Texto
Descrição: O sistema deverá permitir a extração do texto presente nas fotografias ou ficheiro PDF importado.

ID: RF1.7
Título: União de Imagens
Descrição: O sistema deverá tentar unir as fotografias referidas a uma mesma fatura, no caso desta se apresentar segmentada, se receber duas ou mais imagens.

ID: RF1.8
Título: Categorização do Texto Extraído
Descrição: O sistema deverá conseguir identificar, a partir do texto extraído de cada fatura analisada: a data da fatura, o nome do supermercado, o subtotal, o total e os produtos contidos na fatura sendo que para cada um destes deve identificar o nome, a quantidade que cada embalagem individual contém, a quantidade adquirida desse mesmo produto, e o preço total referente ao mesmo.

ID: RF1.9
Título: Armazenamento
Descrição: O sistema deverá armazenar os dados dos utilizadores para autenticação e a informação relativa aos produtos adquiridos e respetivas faturas.

ID: RF1.10
Título: Consulta de Informações de Faturas Registadas
Descrição: A aplicação móvel deverá ser capaz de apresentar as informações relativas a todas as faturas processadas e armazenadas anteriormente, para o utilizador autenticado.

ID: RF1.11
Título: Consulta de Produtos Adquiridos
Descrição: A aplicação móvel deverá ser capaz de apresentar as informações relativas a todos os produtos adquiridos, referentes nas faturas passadas que um dado utilizador autenticado tenha importado.

ID: RF1.12
Título: Lista de Compras Automática
Descrição: O sistema deverá conseguir gerar uma lista de compras automática, baseada nas compras passadas de um dado utilizador e apresentá-la na aplicação móvel.

ID: RF1.13
Título: Recurso à Câmara do *Smartphone*
Descrição: A aplicação móvel deverá possibilitar o uso direto da câmara fotográfica do *smartphone* para a captura da fotografia da fatura.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais explicitam de que forma as funcionalidades do sistema são entregues ao utilizador final. Estão relacionados ao uso da aplicação em termos de desempenho, usabilidade, fiabilidade, segurança, disponibilidade, manutenção e tecnologias envolvidas. Os requisitos funcionais são os seguintes:

ID: RNF1.1
Título: Bases Tecnológicas Gratuitas
Descrição: O sistema deverá ser composto por bibliotecas e *software* somente gratuito.

ID: RNF1.2
Título: Segurança
Descrição: O sistema permitirá a cada utilizador que apenas consiga visualizar as informações armazenadas relativas a si mesmo, estando estas protegidas pela autenticação.

ID: RNF1.3
Título: Encriptação
Descrição: As palavras passes dos utilizadores deverão ser processadas e armazenadas estando encriptadas, por uma questão de privacidade e segurança.

ID: RNF1.4
Título: Proteção do Acesso à API
Descrição: O *Shopping List Automator* deverá conter um sistema de *tokens* como o *JSON Web Token* (JWT) para que a relação entre o servidor e a aplicação móvel seja segura e apenas esta segunda possa chamar as funções do primeiro através da API.

ID: RNF1.5
Título: Usabilidade
Descrição: A aplicação móvel deverá dispor de uma interface gráfica de fácil utilização.

ID: RNF1.6
Título: Desempenho
Descrição: O sistema deverá de conseguir gerar e apresentar a lista de compras automática ao utilizador num curto espaço de tempo.

ID: RNF1.7
Título: Baixo Custo
Descrição: A aplicação móvel deverá ser compatível também em *smartphones* com pouco poder de processamento.

3.1.3 Requisitos de Sistema (Software e Hardware)

As necessidades de *software* e *hardware* necessárias para a implementação do *Shopping List Automator* são representadas pelos seus requisitos de sistema. Os requisitos de sistema são os seguintes:

ID: RS1.1
Título: Compatibilidade da Aplicação Móvel
Descrição: A aplicação móvel deverá ficar acessível para dispositivos Android e iOS, para as versões 9.0 e 4.4 ou superiores, respetivamente.

ID: RS1.2
Título: Câmara Fotográfica
Descrição: O sistema deverá ser capaz de processar as faturas a partir de fotografias capturadas pela a maioria das câmaras fotográficas de *smartphone* disponíveis no mercado.

3.2 Arquitetura

Este Subcapítulo apresenta o desenho da solução, desde como as duas componentes principais se relacionam uma com a outra no sistema até às diferentes tarefas que cada componente executa para atingir o propósito final. As ferramentas e bibliotecas utilizadas pelos componentes são indicados assim como a sua importância para o funcionamento do sistema e é evidenciado todo o processo lógico das diferentes fases de execução do sistema num cenário de utilização por parte de um utilizador que recorreria regularmente ao *Shopping List Automator*.

A figura 3.1 representa a arquitetura de todo o sistema. Um servidor foi desenvolvido na linguagem de programação Python devido a ter diversas bibliotecas apropriadas para as tarefas que se pretendiam. Este servidor conecta diretamente com uma base de dados MySQL para armazenar e ler informações dos utilizadores, respetivos produtos adquiridos e faturas registadas. Contém também uma REST API desenvolvida na *framework* Flask com o objetivo de trocar informações, em ficheiros tipo JSON, com a aplicação móvel.

A aplicação móvel foi desenvolvida na *framework* Flutter para que com o mesmo código fonte fosse possível correr a aplicação nos dois sistemas operativos mais comuns no mercado atual dos *smartphones*, Android e iOS. Porém com as atualizações recentes deste kit de desenvolvimento da Google o mesmo código fonte desenvolvido já compila para aplicação web e aplicações nativas para Windows, Linux e MacOS. A aplicação dis-

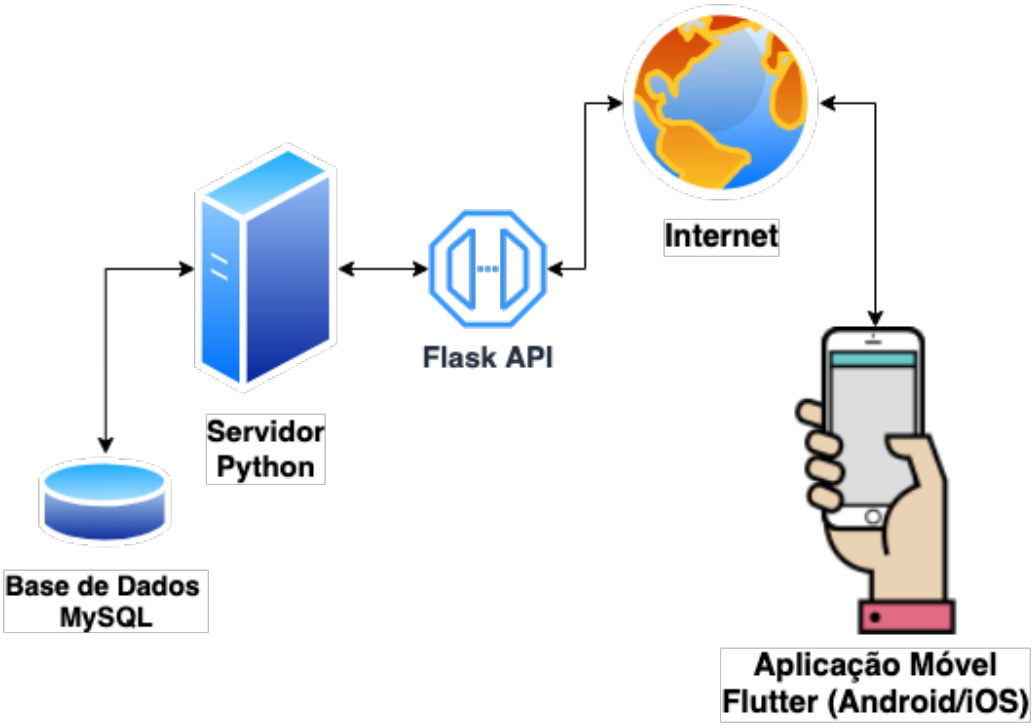

Figura 3.1: Diagrama da Arquitetura do *Shopping List Automator*

ponibiliza uma simples interface gráfica em que o utilizador se deve autenticar, poderá importar imagens ou ficheiros PDF, ver faturas armazenadas e poderá pedir uma lista de compras automática. Todas estes pedidos são enviados através da internet para a REST API do servidor, que os processa e retorna as respostas para a aplicação móvel, aliviando assim grande peso de processamento ao *smartphone* caso todo este sistema fosse desenvolvido somente dentro de uma aplicação móvel, desta forma permite também que a solução possa ser utilizada por telemóveis mais antigos e com menos poder de processamento.

Na figura 3.2 são apresentadas as principais funcionalidades implementadas no servidor do sistema. Os utilizadores devem registar-se e autenticar-se para poder aceder às restantes funcionalidades. O servidor armazena os dados de autenticação dos utilizadores de forma permanente na base de dados e para garantir que só é possível aceder aos dados e funções estando registado, foi implementado um sistema de *tokens* JWT. Consiste na troca de um *token* assinado para que futuramente haja uma validação do cliente que faz o pedido. Por outras palavras, quando o utilizador se tenta autenticar o servidor verifica se ambas as credenciais coincidem com as da base de dados identificando o utilizador respetivo e, caso se verifique, o servidor retorna um *token* assinado com uma chave secreta referente a esse utilizador para o cliente. Desta forma, o próximo pedido que fizer ao servidor envia também esse *token* e é verificado se é válido e reconhece a que utilizador o pedido se refere.

Uma das funcionalidades do sistema permite unir duas ou mais imagens, desde que

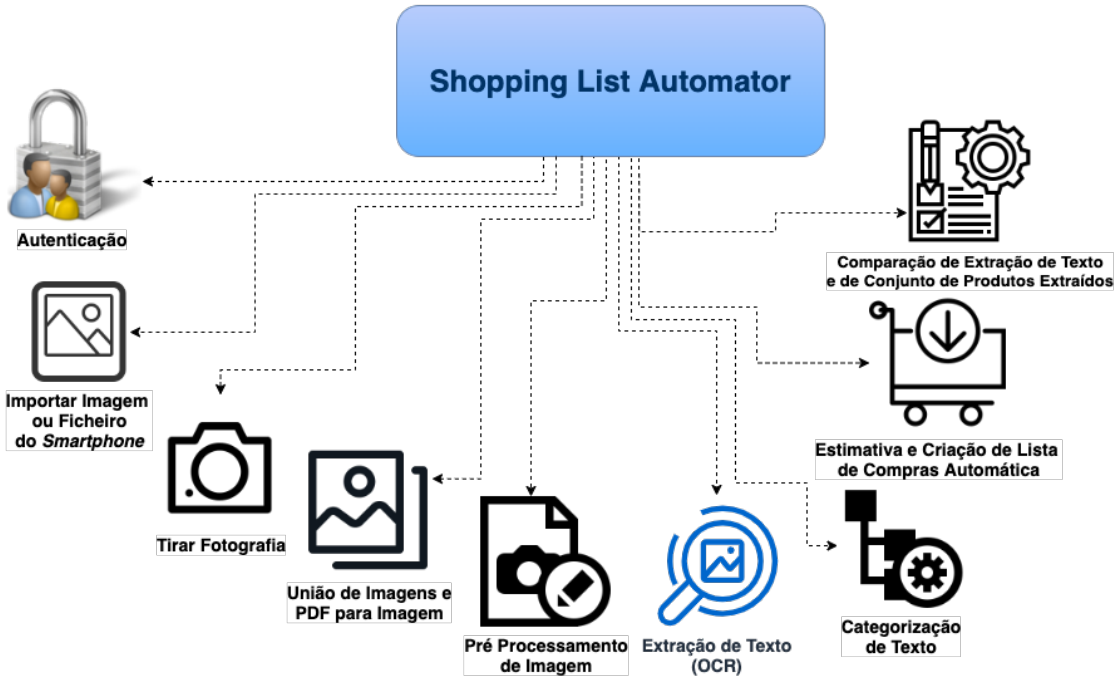

Figura 3.2: Componentes Funcionais do *Shopping List Automator*

tenham um mínimo de partes semelhantes e que estejam todas na mesma orientação. Esta funcionalidade foi desenvolvida tendo em mente as faturas que possam ser demasiado compridas. Para serem apanhadas na totalidade por uma só fotografia, teria de ser capturada com maior distância à fatura reduzindo assim a qualidade de imagem na zona do texto da fatura. Portanto, o sistema permite ao utilizador que capture diversas fotografias, mais aproximadas, das várias partes de uma fatura e esta funcionalidade encarrega-se das unir. Para esta função foram utilizadas bibliotecas como a *OpenCV* para carregar as imagens, para processamento de contornos e acima de tudo para a união das imagens. A biblioteca *Imutils* para apoio adicional no processamento dos contornos no recorte da maior forma retangular dentro da imagem unida. A biblioteca *SciPy* para as rotações de imagem. Para a transformação do ficheiro PDF numa imagem foi utilizada a biblioteca *PDF2Image* disponível para Python.

Precedentemente à extração de texto é aplicado um conjunto de transformações à imagem da fatura a processar com o intuito de obter os melhores resultados possíveis na fase seguinte. Transformações estas que serão detalhadas no subcapítulo seguinte. Uma vez mais a biblioteca *OpenCV* foi fortemente utilizada, acompanhada da *NumPy*, *Skimage*, *Pillow* e *Matplotlib*.

Para extrair o texto da imagem já preparada recorre-se essencialmente à biblioteca *PyTesseract* que automaticamente extrai o conteúdo a partir de imagens suportadas pela biblioteca *Pillow*, funcional com a língua portuguesa.

No tratamento do texto, em que é extraída dados chave de todo o texto obtido sobre a fatura e respetivos produtos na lista, foram utilizadas as bibliotecas *Enchant* para a

correção de palavras baseado em dicionário. Foi utilizada a biblioteca *Datetime* para manipular dados em formatos de data e hora.

Para se estimar o consumo médio de cada produto de cada utilizador e gerar uma lista de comprar automática o servidor apenas se apoia nas bibliotecas *Math* para cálculos matemáticos, MySql para as chamadas à base de dados e novamente à biblioteca *Datetime*.

Por último, o servidor contem duas funções destinadas à avaliação da precisão da extração de texto em comparação ao cenário ideal e outra para avaliar a identificação e categorização da informação chave obtida a partir de uma fatura. Foram aplicadas funções da biblioteca *StrSimpy* que contém os algoritmos de Levenshtein, Levenshtein Normalizado e de Jaro Winkler para análise da distância similaridade entre o cenário real e cenário ideal de cada fatura.

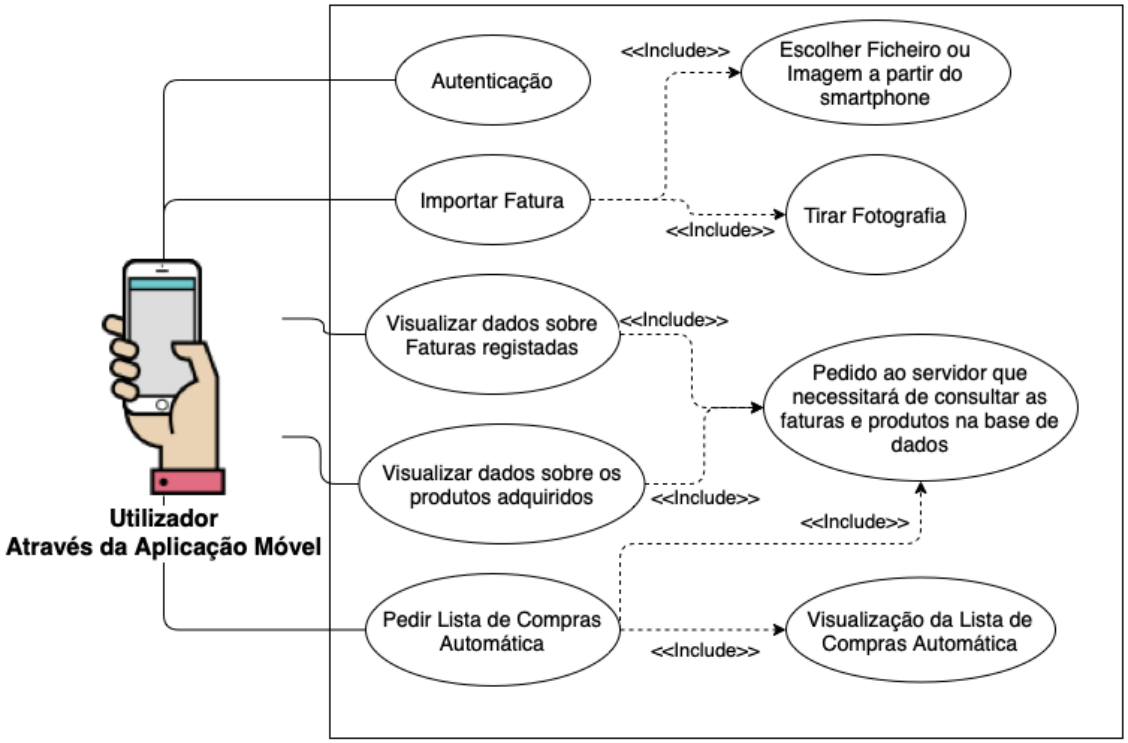

Figura 3.3: Diagrama de Casos de Uso da App *Shopping List Automator*

A aplicação móvel disponibiliza um conjunto de tarefas de simples processamento mas importantes para fazer a ponte entre o utilizador e o servidor, de forma prática. No diagrama de casos de uso da figura 3.3 verificamos várias unidades funcionais que a aplicação móvel fornece ao utilizador. Para cada uma destas, existe uma porta associada na API do servidor disposta a receber e responder aos pedidos da aplicação móvel. No caso do utilizador estar autenticado, pode aceder às restantes funcionalidades como importar faturas de uma nova compra que tenha realizado para fornecer mais informações ao sistema quanto aos seus consumos. Melhorando assim a previsão num próximo pedido de lista de compras automática ou pode também visualizar dados sobre faturas e produtos adquiridos anteriores. Para visualizar a lista de compras estimada para o dia em que reali-

zar o pedido, o utilizador passa antes por uma vista em que deve inserir o número de dias para os quais pretenderia não ter de se deslocar novamente a um supermercado.

Antes de se proceder à extração de texto através da biblioteca *PyTesseract*, precedem-se diversas transformações à imagem para obter melhores resultados. Estes passos acontecem na ordem especificada na figura 3.4. Tendo a imagem preparada para ser processada, inicia-se a duplicação da imagem original visto que de seguida são aplicadas transformações a uma versão da imagem com qualidade reduzida porque dessa forma o processamento é mais eficiente. Sobre a versão da imagem de qualidade reduzida aplica-se uma conversão para tons de cinzento seguido de um filtro de *Gaussian Blur* para remoção de ruído. Visto que a imagem pode ter vários objetos além da fatura e pretende-se somente a fatura em si, passa-se a um processo para detetar os contornos somente da fatura, terminando com um recorte dessa zona da imagem na imagem original e uma correção de perspetiva.

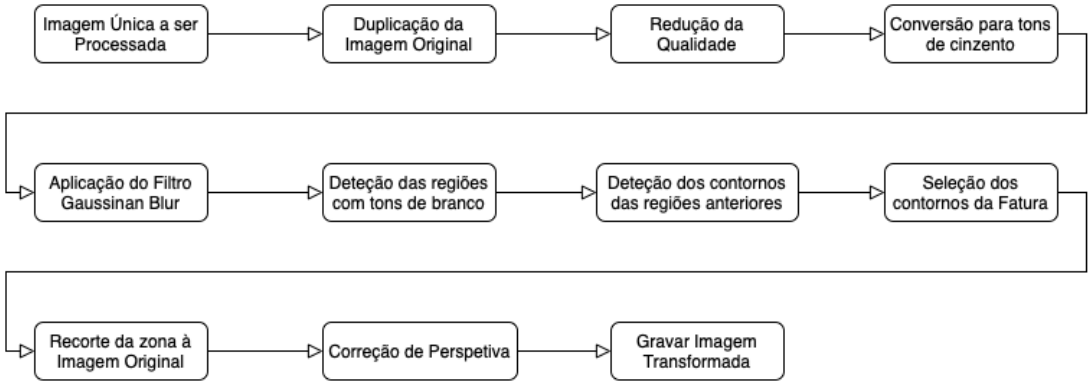

Figura 3.4: *Pipeline* do Pré Processamento de Imagem Anterior à Extração de Texto

Após o texto das faturas importadas ser extraído, organizado e armazenado na base de dados existe a possibilidade de, recorrendo a esses dados que representam o histórico de compras, estimar o consumo de um utilizador para cada produto e gerar uma lista de compras sugestiva. Esta lógica pressupõe que após todas as compras que o consumidor tenha feito, houve uma preocupação da sua parte em importar todas as faturas para o sistema. Assim sendo, o algoritmo assume que desde a compra mais antiga de um dado produto até ao dia atual tem controlo total da quantidade desse produto que foi supostamente consumida. Pode-se assim estimar e média de consumo diária. No fluxograma da figura 3.5 apresenta-se o raciocínio por trás da criação da lista de compras. Inicialmente é necessário que o utilizador informe quantos dias em diante pretende, desde o dia do momento do pedido, não ter de voltar ao supermercado, sugerindo assim ao algoritmo a quantidade de reservas de produtos que o utilizador deverá ficar em casa após a compra. O algoritmo obtém da base de dados todos os diferentes produtos que adquiriu até então e para cada um destes percorre todas as diferentes entradas. Em diferentes faturas o utilizador pode ter comprado o mesmo produto em quantidades diferentes, em datas dife-

rentes. Tendo o controlo de todas as entradas do mesmo produto é possível saber a data mais antiga e mais recente que esse produto foi adquirido, em que quantidades e a que preço unitário (em cada entrada diferente, pode ter adquirido a preços unitários diferentes devido a promoções ou por ser num supermercado diferente, por exemplo). Com estes dados é possível determinar durante quanto tempo foi consumido cada produto, chegando assim a uma média de consumo diária e a média de preço unitário estimado para uma próxima compra. Com as datas das compras dos produtos o algoritmo também estima qual a quantidade que o utilizador poderá ainda ter na dispensa de casa para subtrair ao que deveria de adquirir. Analisando a média diária de cada produto, o diferença de dias até o dia do pedido da lista de compras, a quantidade estimada do produto armazenado e o número de dias que o utilizador gostaria de ter reservas, é possível estimar a quantidade de cada produto que deverá de adquirir. Gerindo todas as informações, o algoritmo devolve um ficheiro JSON com os produtos que deverá de adquirir, em que quantidade, com estimativa de preço unitário e correspondente preço total para cada produto assim como o preço total estimado de toda a lista de compras sugestiva.

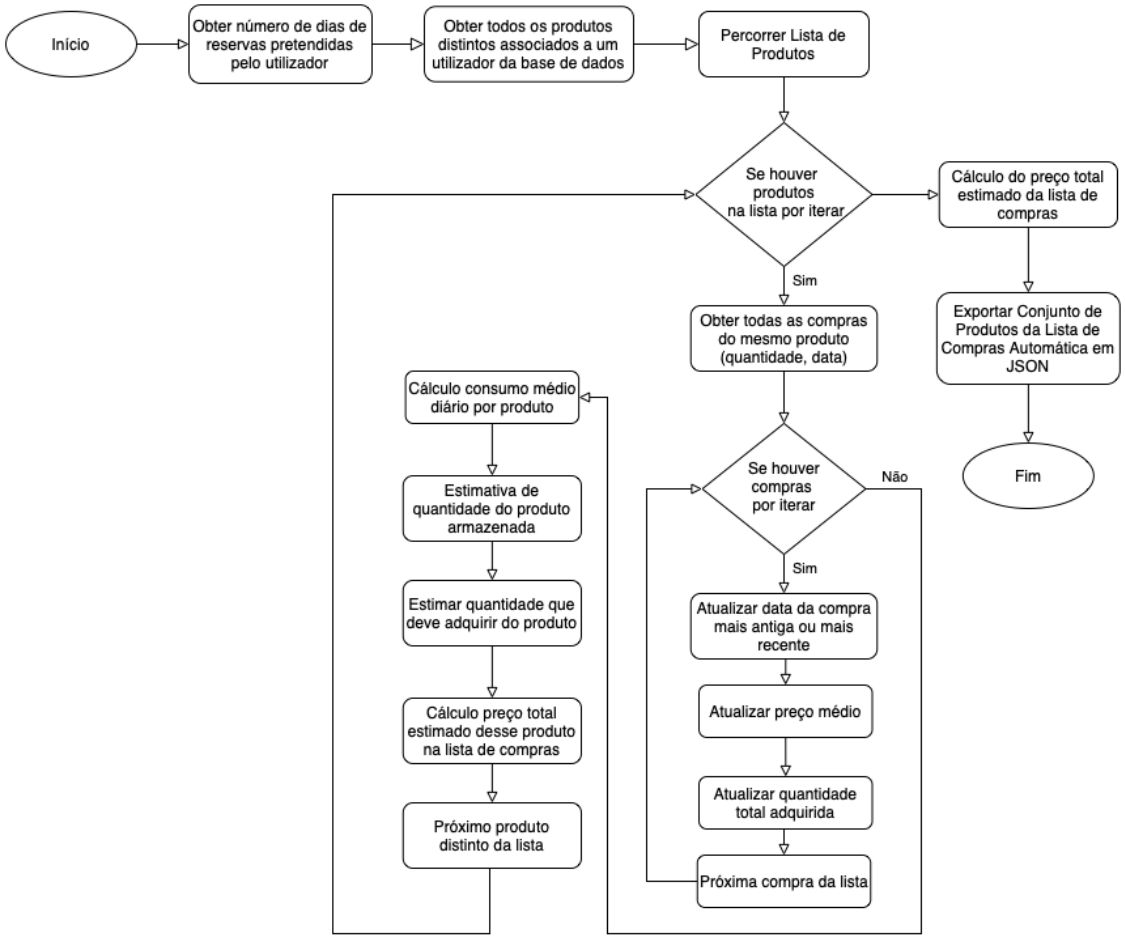

Figura 3.5: Fluxograma da Análise de Consumo e Criação Automática da Lista de Compras

A figura 3.6 da página seguinte apresenta os acontecimentos mais importantes desde

que o utilizador visualiza ou executa algo na aplicação móvel, passando pelos procedimentos intermédios que acontecem do lado do servidor, até à devolução de uma resposta com destino à aplicação em que o utilizador visualiza a informação esperada ou então uma simples mensagem de erro ou sucesso. Graficamente é demonstrado: a lógica da troca de *tokens* já descrita acima, a ordem com que as funcionalidades do servidor são chamadas para o processamento de cada fatura até o armazenamento da informação na base de dados, os passos executados para a apresentação da informação registada pelo utilizador até então na aplicação e por fim, as etapas necessárias para o pedido, criação e apresentação de uma lista de compras automática. Além das mensagens de alerta, o servidor retorna as informações como o conjunto de dados de faturas, produtos ou a lista de compras sugestiva em formato JSON. As fases de cada processo em que o mesmo transita para o lado do servidor ou quando existem chamadas à base de dados estão salientadas com um evento específico e ilustradas com um servidor e uma base de dados, se for o caso.

A base de dados, presente do lado do servidor, tem uma estrutura simples e com os campos essenciais para permitir as funcionalidades pretendidas, como demonstrado na figura 3.7. Contém uma tabela para armazenas as informações relativas aos utilizadores e ainda o seu respetivo *token* para garantir a segurança no acesso à API como referido anteriormente. Existe uma tabela dedicada às informações gerais de cada fatura tais como o nome do supermercado, a data de compra associada a essa fatura, o preço total e a quantidade de produtos que o sistema conseguiu extrair, reconhecer e armazenar na base de dados, associados à fatura em questão. A cada produto que o sistema extrai de cada fatura é criada uma nova entrada na tabela de produtos adquiridos em que se armazena o identificador da fatura a que cada produto corresponde, o nome do produto, a quantidade comprada, a capacidade unitária seja em unidades de massa ou de capacidade, o preço unitário a que este produto foi adquirido naquele dia e a data em que foi adquirido. Ambas as tabelas referidas contêm um identificador do utilizador que realizou a compra.

A lógica envolvente à funcionalidade do servidor relativamente à comparação de texto extraído por OCR e de conjuntos de produtos obtidos é detalhada no Capítulo 4 referente à avaliação do sistema. No Subcapítulo seguinte, referente à implementação do sistema, é apresentada uma análise mais profunda quanto às funcionalidades referidas acima assim como o esclarecimento de como é executado a categorização de texto.

3.3 Implementação

Neste Capítulo é apresentado, de forma pormenorizada, a implementação do *Shopping List Automator*. Esta secção faz referência às ferramentas tecnológicas utilizadas, apresenta os segmentos de código mais relevantes com as suas devidas explicações. Vão sendo apresentadas imagens que ilustram os resultados desses segmentos de código. A

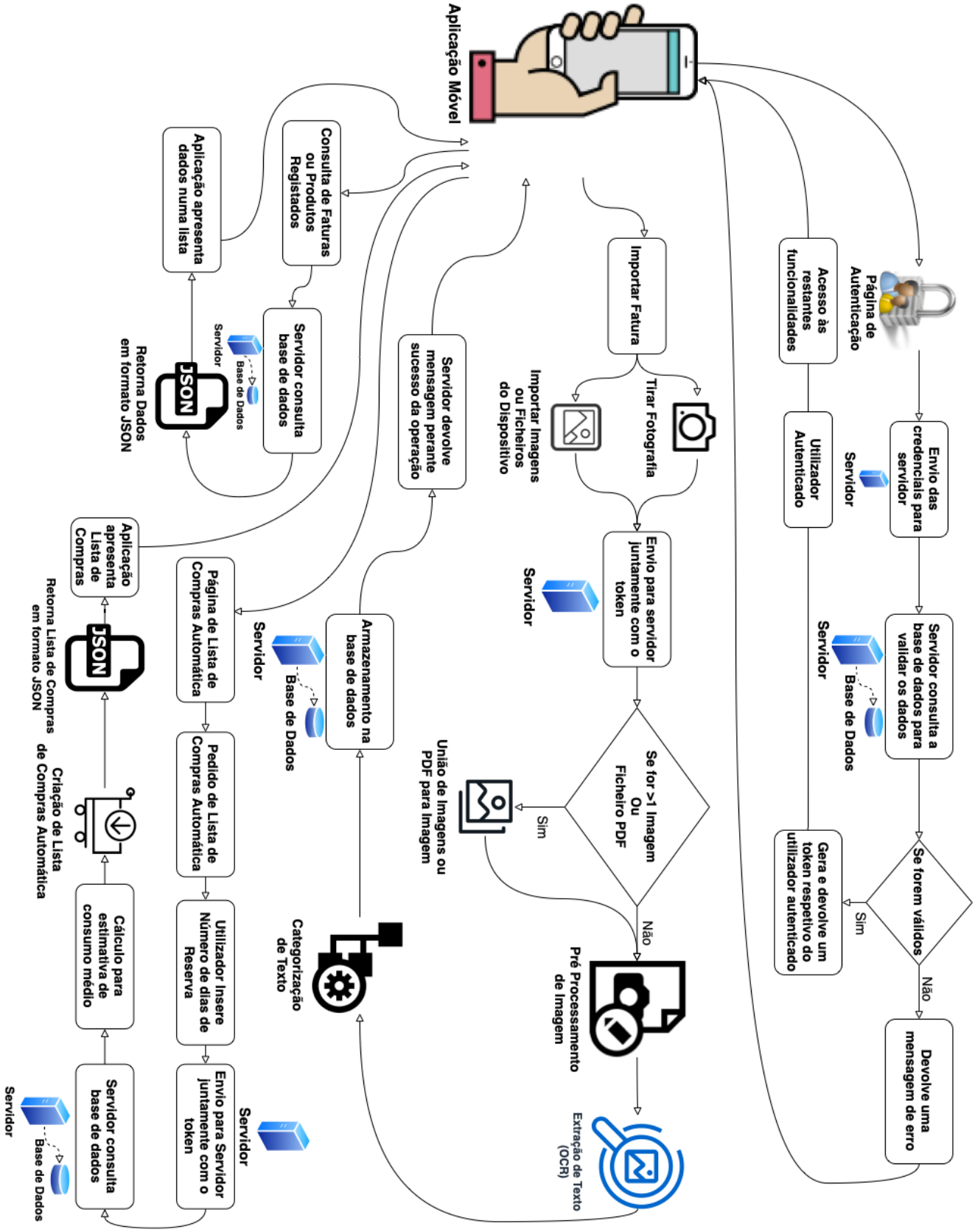

Figura 3.6: Diagrama de Interações entre Aplicação Móvel e *Backend* do Shopping List Automator

Figura 3.7: Diagrama de Relação de Entidades da Base de Dados do *Shopping List Automator*

implementação está divida em quatro partes que resultam nos quatro Subcapítulos aqui presentes. Cada um engloba uma ou mais tarefas das que foram apresentadas no Capítulo três, sobre a arquitetura do sistema. Os três primeiros Subcapítulos explicitam implementações do servidor *Python* e no quarto e último Subcapítulo apresenta-se a aplicação móvel desenvolvida em *Flutter* e a API do servidor que permite a comunicação entre ambos.

O grande propósito deste *Shopping List Automator* é conseguir gerar uma lista de compras automática a partir de faturas importadas pelo utilizador, processo esse que já foi apresentado na figura 3.6. A ordem dos Subcapítulos aqui exibidos e as explicações dentro de cada um deles segue o fluxo real de execução do sistema, com uma ligeira alteração do começo e fim. Na prática, o utilizador entra primeiramente em ação acedendo à aplicação móvel, importando as faturas, o servidor processa as informações e termina com o utilizador a visualizar os dados novamente no *smartphone*. Nesta secção inicia-se imediatamente pela fase em que o servidor recebe a fatura importada, procede-se a ordem normal do processamento até ao armazenamento dos dados e só no fim é que se apresenta

a implementação da aplicação móvel e API do servidor.

Durante o desenvolvimento deste sistema foram utilizadas diversas faturas reais, muitas em papel e outras em formato digital, de compras processadas em 2021, de supermercados e mercados locais como o Continente, Lidl, Super Milheirós, Auchan, Pingo Doce, entre outros. Estas faturas capturadas como imagens únicas, várias imagens segmentadas ou em ficheiro no formato PDF foram utilizadas para efeitos de teste e consequente melhoria de cada mecanismo deste *software*. No caso do mecanismo que permite a união de imagens também foram utilizadas imagens de diversos objetos de grande dimensão e do interior de uma casa, pelo facto de facilitar o trabalho de uma das bibliotecas usadas na união e facilitar também a deteção de erros existentes numa fase inicial do desenvolvimento dessa tarefa.

3.3.1 Pré-Processamento da Fatura

Posteriormente ao envio de um ou mais ficheiros referentes a uma fatura através da aplicação móvel para o servidor, é aplicado um pré-processamento. Para a extração de texto através de OCR é necessário que o ficheiro a ser processado seja uma imagem. Além de ser possível aplicar a técnica de reconhecimento ótico de caracteres a qualquer imagem, os resultados obtidos de uma imagem que além da fatura pudesse conter outros objetos, outras zonas com texto ou até mesmo simples ruídos na imagem, não seriam tão satisfatórios. Por tal motivo foi implementado um conjunto de operações que visam melhorar os resultados, que serão apresentados no Capítulo 4 sobre a avaliação em que é apresentada uma comparação de faturas executadas com ou sem pré-processamento.

Na perspetiva do servidor, aquando o recebimento de uma fatura, este analisa em qual das três formas a fatura foi recebida:

- Uma imagem do tipo PNG ou JPEG;
- Duas ou mais imagens do tipo PNG ou JPEG;
- Um ficheiro do tipo PDF.

No caso de receber duas ou mais imagens, estas são colocadas numa pasta temporária e é chamada uma função que tentará uni-las caso existam partes semelhantes entre si. Esta funcionalidade é importante para o sistema pois o utilizador deverá de enviar imagens da fatura com uma dada proximidade à mesma para obter melhores resultados, mas deve captar sempre todas as bordas da fatura para a ser possível um posterior recorte à imagem.

As seguintes imagens da figura 3.8 representam um conjunto exemplo de fotografias que foram importadas para o servidor referentes à mesma fatura. Há a necessidade de unir estas imagens, de forma a obter a fatura com as suas quatro bordas visíveis numa só imagem final preservando a boa qualidade de imagem que tinham anteriormente.

Figura 3.8: Imagens Importadas Antes da União no Servidor

Para unir as imagens recorreu-se à biblioteca *OpenCV* que disponibiliza uma função para tal efeito. Depois de vários testes durante o desenvolvimento descobriu-se que a função está preparada para criar imagens panorâmicas, ou seja, fazer uma união de imagens pelas suas laterais, unido-as horizontalmente. Tendo em conta que o utilizador procura capturar a fatura na vertical, no início da função que une as imagens realiza-se uma rotação de noventa graus positivos às diversas imagens. Na parte de excerto de código 1 recorre-se ao *OpenCV* para carregar cada uma das imagens e à função da biblioteca *SciPy*

para a rotação, acrescentando no fim todas elas a uma lista de imagens prontas a serem unidas.

Excerto de Código 1 Rotação de Imagens a 90 graus e Adição à Lista

```
for imagem in imagens:
    img = cv2.imread(imagem)
    img = rotate(img, 90)
    imagens.append(img)
```

De seguida é inicializado um objeto para a união de imagens do *OpenCV*, utilizando a função "*cv2.createStitcher()*". Executa-se depois a desejada união sobre a lista criada anteriormente através da função "*stitcher.stitch()*". Na variável chamada "*stitched*"ficará a imagem resultante e na variável "*status*"o estado da tentativa de união.

Segue-se uma verificação ao conteúdo da variável "*status*", e no caso de ser 0, significa que a união foi aplicada com sucesso. Neste ponto do algoritmo existe uma parte opcionalmente executável caso se pretenda obter uma imagem perfeitamente retangular, visto que a resultante pode ter um formato muito deformado dependendo das imagens iniciais. Esta parte de código contém um conjunto de transformações com o objetivo de cortar o maior retângulo existente na imagem resultante, passando pelos seguintes passos, segundo a ordem:

- Criação de uma fronteira de 10 pixeis a contornar a imagem;
- Converter a imagem para escala de cinza e aplicar *threshold*;
- Encontrar contornos externos à imagem com *threshold* e desses selecionar o mais comprido, que será o contorno geral da imagem unida original;
- Alocar memória para a máscara que irá conter a caixa delimitadora retangular;
- Criar duas cópias da máscara anterior em que uma servirá como referência para a menor região retangular e a segunda servirá como um contador de quantos pixeis necessitam de ser removidos para formar a menor região retangular;
- Remoção dos pixeis a 0;
- Extrair as coordenadas da máscara retangular e obter a imagem retangular final.

Muitos dos passos acima têm semelhanças no tratamento da imagem exatamente antes da extração de texto, referido abaixo neste Subcapítulo e por essa razão é que não é aqui aprofundado a nível de código.

Com a união de imagens bem sucedida, é aplicada uma rotação negativa de 90 graus para obter a imagem unida da fatura na vertical. A imagem unida poderia agora prosseguir para o tratamento seguinte.

A figura 3.9 representa a imagem unida processada pelo algoritmo descrito, com recorte retangular, referente às imagens da figura 3.8.

SUPER DE MILHEIROS
PALMIRA SUSANA FERREIRA SILVA
VIA CENTRAL DE MILHEIROS, N.155 A
4475-330 MILHEIROS
TELEFONE-229 734 602
CONTRIBUINTE-205988687

FATURA SIMPLIFICADA
ORIGINAL
DATA 2021/01/23 - 9:59 Nº A/20732

Cliente: Consumidor final
Morada.:
Cont.Nº: N/A

Artigo	Qtd	Preço €	Valor €
FUNDO			
6% VITELA FUNDO	0,610 kg x	7,99 =	4,87
6% COURGETE	1,250 kg x	10,90 =	13,63
6% CENOURA	0,500 Kg x	3,99 =	2,00
6% MOLHO FRANCEZINHA D.DUARTE	1,070 Kg x	0,99 =	1,06
23% BELGAS ORIGINAL	1 Un x	2,09 =	2,09
23% CHOCO CHIPS	1 Un x	1,99 =	1,99
23% TOMATE CACHO.	1 Un x	1,00 =	1,00
6% NABOS	0,410 KG x	2,95 =	1,21
6% BATATA ASSAR 2,5KG SACO	0,260 KG x	2,99 =	0,78
6% PÃO BIJU	1 Un x	3,80 =	3,80
6% #016	6 UN x	0,10 =	0,60

TOTAL..... 33,03
Multibanco..... 33,03
Troco..... 0,00

IVA Incluido à taxa indicada

Taxa%	Sujeito	IVA	Total
23,00	4,13	0,95	5,08
6,00	26,37	1,58	27,95

h/u4-Processado por programa certificado
Nº 88/AT - 6 A/20732

Opr: ANDREIA

OBRIGADO - VOLTE SEMPRE

Figura 3.9: Imagem Resultante do Algoritmo de União

Caso o servidor receba um ficheiro do tipo PDF, recorre-se à função da biblioteca *PDF2Image* "*convert_from_path(pdfPath, 500)*". Fornecendo à função o caminho do ficheiro ou o próprio em *bytes*, esta retorna uma lista em que cada elemento corresponde a cada página do PDF inicial em formato de imagem. Nas aplicações oficiais correspondentes aos supermercados Continente, Lidl e Pingo Doce, em que foram testadas as suas faturas em formatos digitais, todas elas eram fornecidas em formato PDF e eram compostas por apenas uma página. Ainda assim o algoritmo está preparado para percorrer mais que uma página e seguir com as próximas tarefas tranquilamente. Importante referir

que somente no caso do servidor receber a fatura em formato PDF, este não passará pelo pré-processamento descrito a seguir, passando diretamente para a extração de texto visto que o ficheiro já está simples o suficiente para ter os melhores resultados possíveis.

Detendo o sistema, nesta fase do processamento, um ficheiro do tipo imagem referente à fatura em que se apresentam as quatro bordas da mesma, aplicam-se algumas transformações com o objetivo de remover ruído e recortar a imagem segundo as bordas da fatura. Durante a explicação desta parte do algoritmo, irão sendo mostradas as transformações aplicadas à imagem de entrada da figura 3.10, de uma fatura do Continente. Verifica-se que nesta imagem também aparecem outros objetos, simulando um cenário real em que o utilizador, por exemplo, pousa a fatura em cima de uma mesa e rapidamente tira uma fotografia. Desta forma as capacidades do algoritmo em reconhecer somente a fatura também se tornam mais notórias.

Inicialmente o algoritmo reduz a qualidade da imagem recebida visto que para encontrar contornos numa imagem, neste caso de uma fatura, torna-se mais eficiente em imagens de baixas dimensões. Primeiramente definiu-se um rácio para redimensionamento da imagem, duplicou-se a imagem original para outra variável com o objetivo de se preservar a imagem com a qualidade máxima e por fim a aplicação do redimensionamento através da função "*opencv_resize(image, resize_ratio)*".

Segue-se uma conversão para tons de cinzento recorrendo à função "*cv2.cvtColor()*"do *OpenCV*. Na figura 3.11 é apresentada a imagem resultante desta conversão.

Logo depois é aplicado um simples filtro de Gaussinal Blur para remover possíveis ruídos com a chamada à função do *OpenCV* apresentada no excerto de código 2, com a figura 3.12 a representar a imagem resultante. O filtro Gaussian Blur permite suavizar e reduzir o detalhe de uma imagem, atenuando as frequências mais altas (Flusser et al., 2016). Este efeito de desfoque é aplicado misturando a imagem original com o *kernel* dado, resultando numa matriz.

Excerto de Código 2 Função para Apresentação de Imagem em Tons de Cinzento

```
imagem_desfocada = cv2.GaussianBlur(imagem_cores_cinzentas, (4, 4), 0)

plot_gray(imagem_desfocada)
```

Chega a fase em que se procura obter somente os contornos da fatura. Procura-se detetar todas as regiões com tons de branco, portanto aplica-se uma dilatação utilizando a função "*cv2.dilate(image_blurred, kernel)*", sendo que recebe um *kernel* instanciado antes com o formato e dimensões mais adequadas para o tipo de dilatação que desejavamos. O *kernel* foi criado utilizando a função "*cv2.getStructuringElement(cv2.MORPH_RECT, (8, 8))*". Esta dilatação sobre a imagem resulta na figura 3.13. A deteção de contornos das regiões mais claras, o passo seguinte, terá o trabalho facilitado devido a esta transformação..

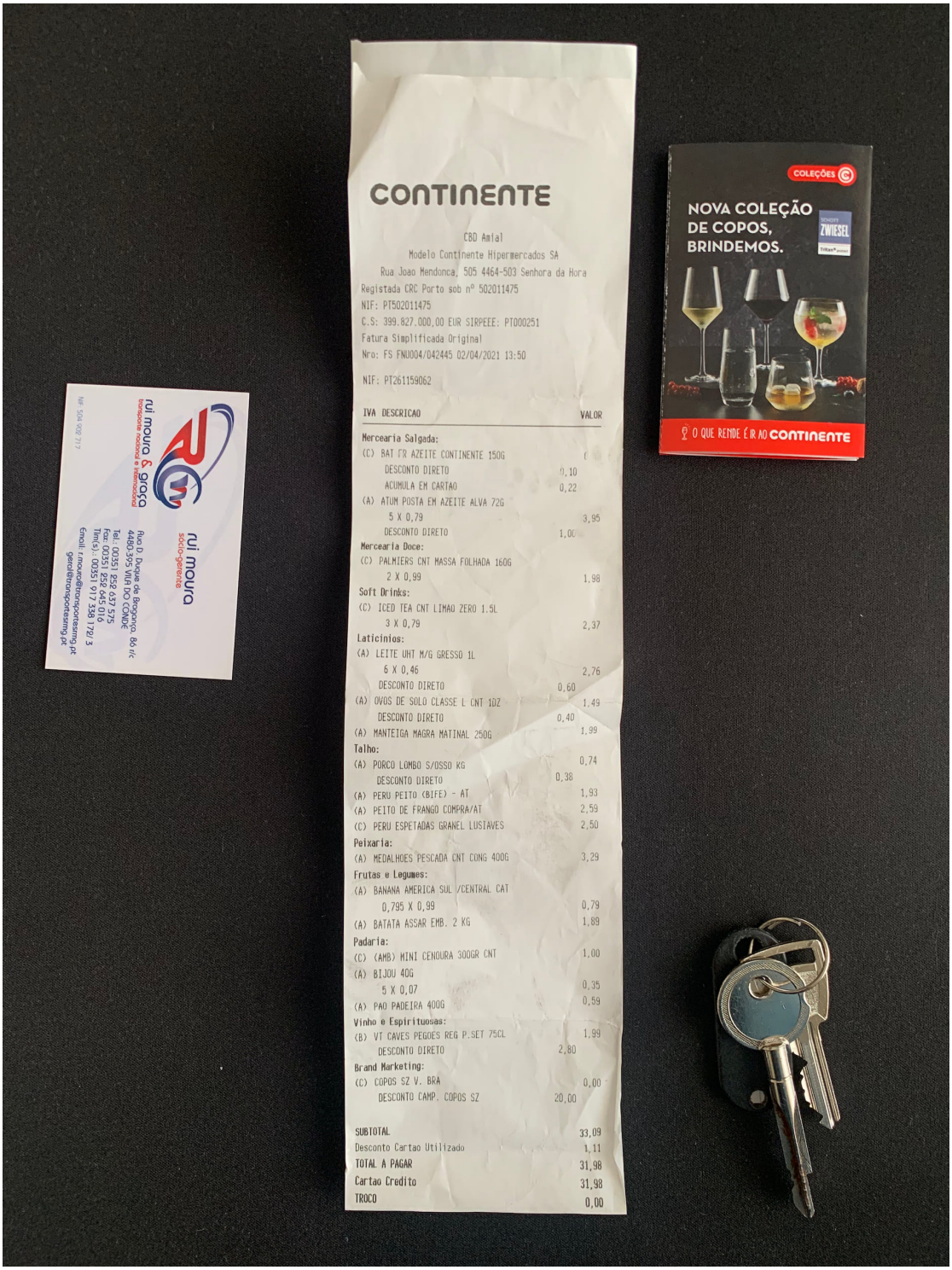

Figura 3.10: Imagem de Entrada de Fatura do Continente para o Pré-Processamento

Neste momento pode-se recorrer à deteção de Canny Edge. Num artigo publicado em 1986, Canny introduziu este algoritmo que visa detetar diversos contornos numa imagem, identificando as descontinuidades ou grandes variações nas intensidades dos pixeis vizinhos (Canny, 1986). Recorrendo à função "*cv2.Canny()*"obtemos os contornos detetados segundo a figura 3.14.

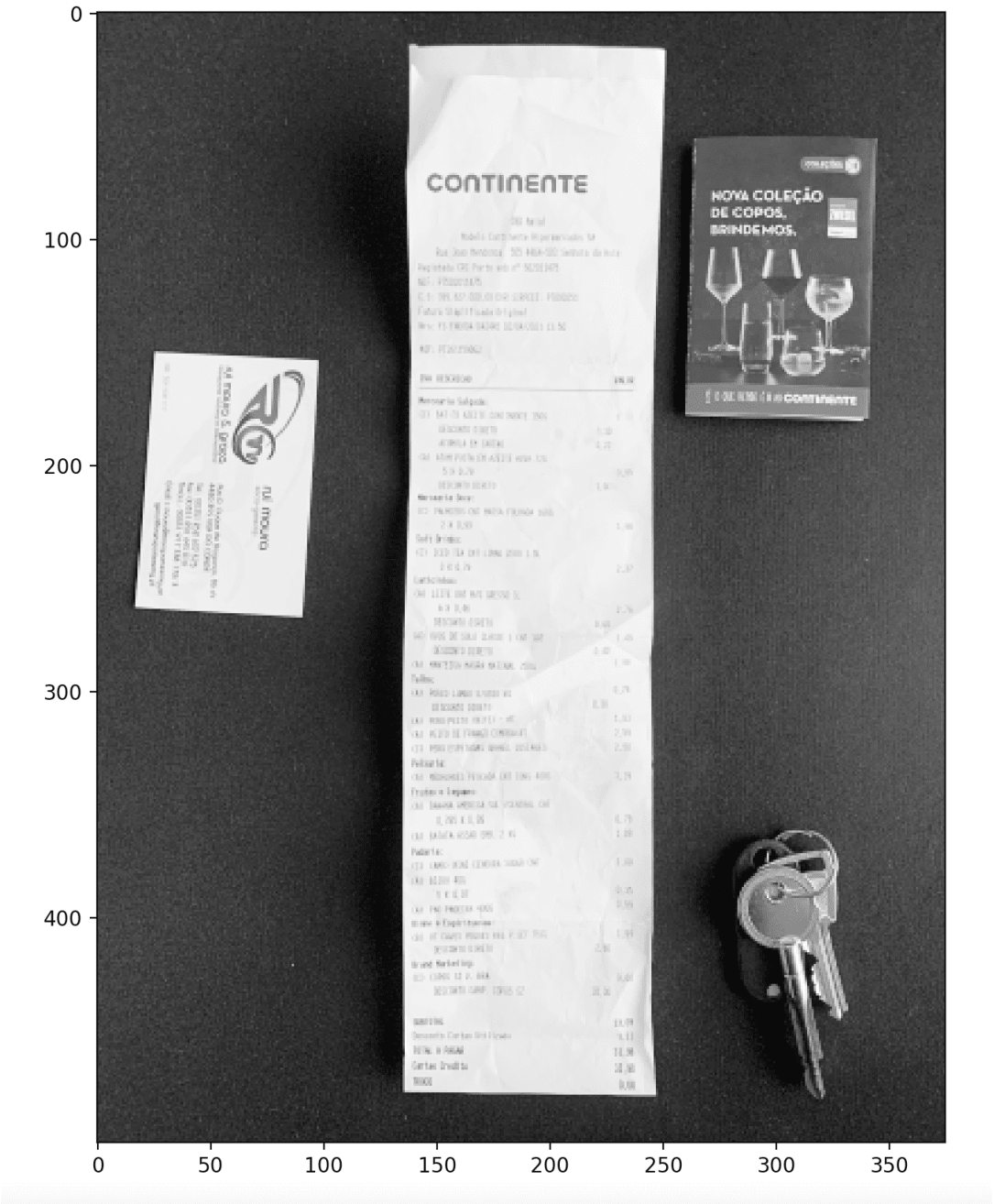

Figura 3.11: Imagem de Menor Resolução Transformada para Tons de Cinzento

Prosseguimos encontrando e armazenando os contornos obtidos pelo algoritmo de Canny Edge através da função $findContours()$ fornecida pelo *OpenCV*. Tendo todos os contornos detetados, sobrepose-mo-los à imagem original preservada para demonstrar visualmente o que foi possível detetar naquele ponto de execução, como apresentado na figura 3.15. De seguida, aproveita-se todos os contornos detetados e efetua-se uma filtragem inicial, ficando apenas com os 8 contornos mais compridos, resultado também revelado na figura 3.16. Por fim nesta parte de código 3 chama-se a função que procura fazer a filtragem final, ficando apenas com os contornos da fatura, assumindo que na imagem esta será o foco e terá o maior contorno e de forma retangular.

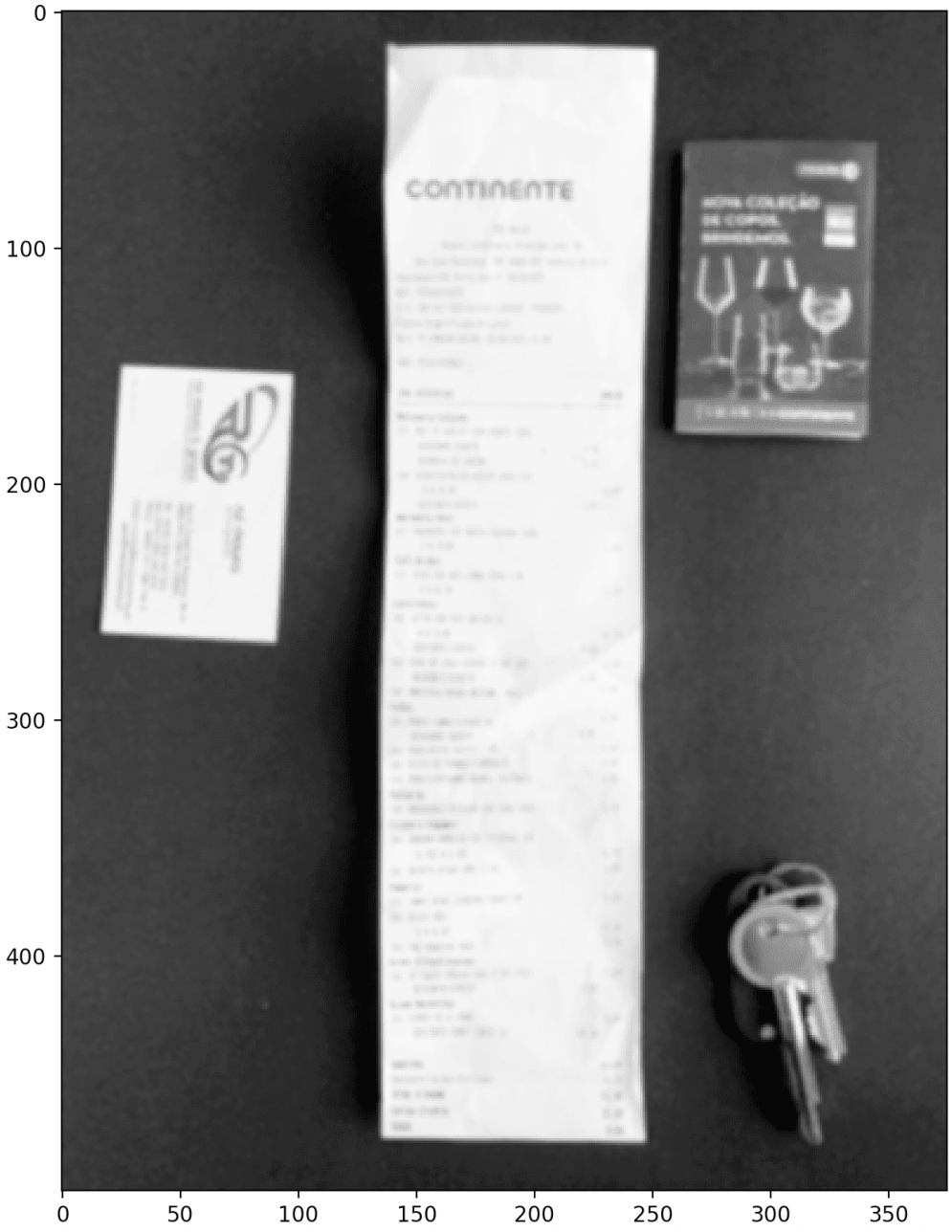

Figura 3.12: Imagem com filtro Gaussian Blur Aplicado

Para extrair somente os contornos da fatura a partir dos oito mais compridos, percorre-se cada um deles efetuando uma aproximação do seu formato ao polígono mais aproximado, que tenha tantos ou menos vértices que esse contorno. No excerto de código 4 demonstra como é utilizada a função de aproximação dos contornos ao polígono, que recebe como argumentos o contorno a aproximar, a distância máxima entre o contorno a transformar e o suposto polígono mais próximo e a variável que sugere se o contorno introduzido é fechado, respetivamente. A partir desta aproximação de cada contorno verificamos se o polígono resultante é composto por quatro vértices assumindo que a fatura terá sempre um formato quadrado ou retangular. Mesmo que hajam outras formas seme-

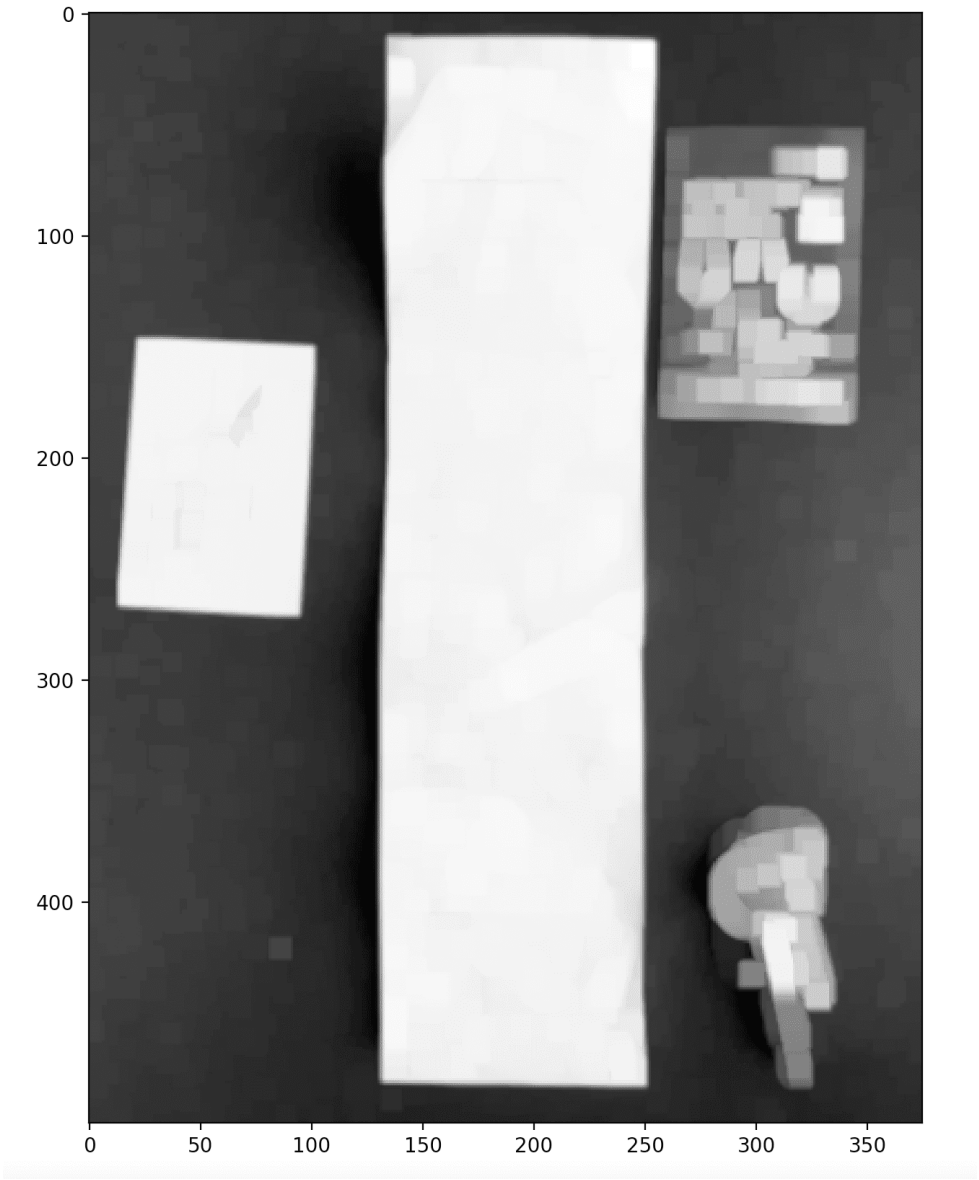

Figura 3.13: Imagem com Dilatação Aplicada

lhantes na imagem de tamanho inferior à da fatura, não afetará a seleção pois a lista dos oito contornos mais compridos estava ordenada por ordem decrescente. Por outras palavras, o primeiro contorno aproximado de quatro vértices será o retornado pela função, que corresponderá ao da fatura. A imagem original sobreposta pelos contornos detetados supostamente referentes à fatura presente é apresentada na figura 3.17. Como se pode verificar, mesmo com outros objetos presentes na imagem o algoritmo foi capaz de reconhecer somente o da fatura.

Nesta fase do algoritmo já conseguimos identificar a região correspondente à fatura. Posteriormente pretende-se recortar a imagem e restaurar a perspetiva da fatura para que esta fique alinhada com a orientação habitual de um documento. Para efetuar tal transformação recorreu-se a duas funções. A primeira serve para converter o contorno da fatura num *array* de coordenadas num formato retangular perfeito. Na segunda função utiliza-

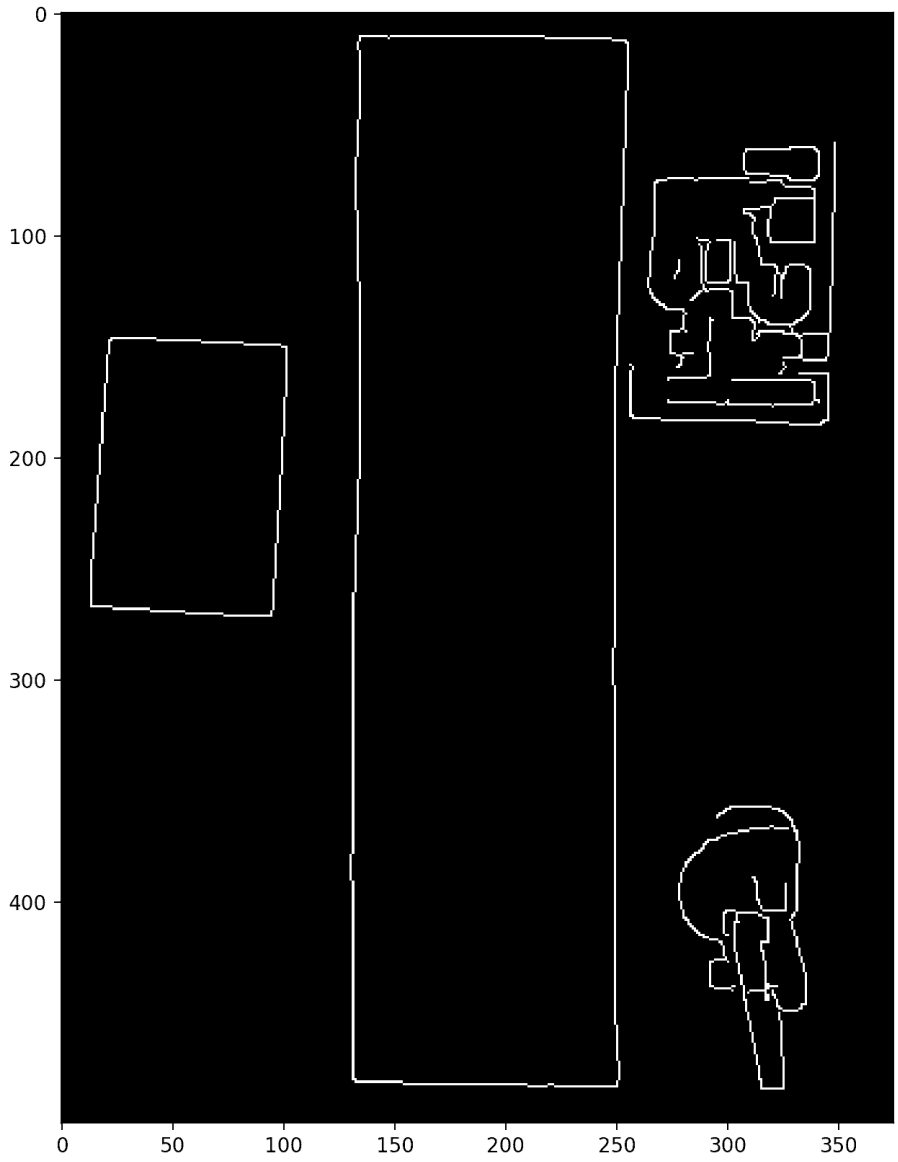

Figura 3.14: Imagem com Aplicação do Algoritmo Canny Edge

mos os pontos do retângulo obtido para calcular os pontos de destino da fatura com a perspetiva ideal, calcula-se a matriz transformada da perspetiva e termina aplicando essa transformada à imagem original. As duas funções de apoio referidas são apresentadas no excerto de código 5. As funções são utilizadas no momento de restauro da perspetiva, chamando-as da seguinte forma:
"*restauro_perspetiva(original.copy(),transformarContornoNumRetangulo(contornoRecibo,racioRedimens*
A imagem obtida é exibida na figura 3.18.

Por último nesta componente que é o pré-processamento da imagem, antes da extração de texto via OCR recorre-se a uma função do *OpenCV* para converter a imagem para tons de cinzento e a outra função da biblioteca *Skimage* para aplicar um efeito *threshold*, resultando numa imagem com aspeto de documento digitalizado, somente a preto e branco. Esta transformação de cores foi adicionada pois era esperado obter melhores resultados

Excerto de Código 3 Processo de Deteção dos Maiores Contornos

```
contornos, hchy = cv2.findContours(canny_edged, cv2.RETR_TREE,
                                   cv2.CHAIN_APPROX_SIMPLE)

contornos_imagem = cv2.drawContours(imagem.copy(), contornos, -1,
                                    (255, 0, 0), 3)
plot_rgb(contornos_imagem)

contornos_mais_largos = sorted(contornos, key=cv2.contourArea, reverse=True)
                                                                [:8]
imagem_com_contornos_mais_largos = cv2.drawContours(imagem.copy(),
                                contornos_mais_largos, -1, (255, 0, 0), 3)
plot_rgb(imagem_com_contornos_mais_largos)

contorno_recibo = contornoRecibo(contornos_mais_largos)
```

Excerto de Código 4 Funções Auxiliares para Aproximação e Filtragem dos Contornos Detetados

```
def poligno_a_partir_de_contorno(contorno):
    p = cv2.arcLength(contorno, True)
    return cv2.approxPolyDP(contorno, 0.03 * p, True)

def contornoRecibo(contornos):

    for c in contornos:
        aproximada = poligno_a_partir_de_contorno(c)

        if len(aproximada) == 4:
            return aproximada
```

por estar mais simplificada contendo apenas dois tons de cores e por ter um aspeto mais limpo. Foram realizados testes em faturas com e sem este filtro e os resultados e conclusões são apresentados no Capítulo seguinte sobre a avaliação do sistema. A parte de excerto de código 6 representa a implementação dessa transformação e a figura 3.19 a imagem resultante e final do pré-processamento.

3.3.2 Extração e Categorização de Informação

A fase de em que ocorre a extração de texto e se obtém as informações chave é dividida em quatro partes:

- Extração do texto recorrendo à biblioteca *PyTesseract*;
- Percorrer o texto obtido para obter informações gerais da fatura como o nome do supermercado e a data de compra;
- Percorrer o texto obtido para obter todos os produtos na lista;

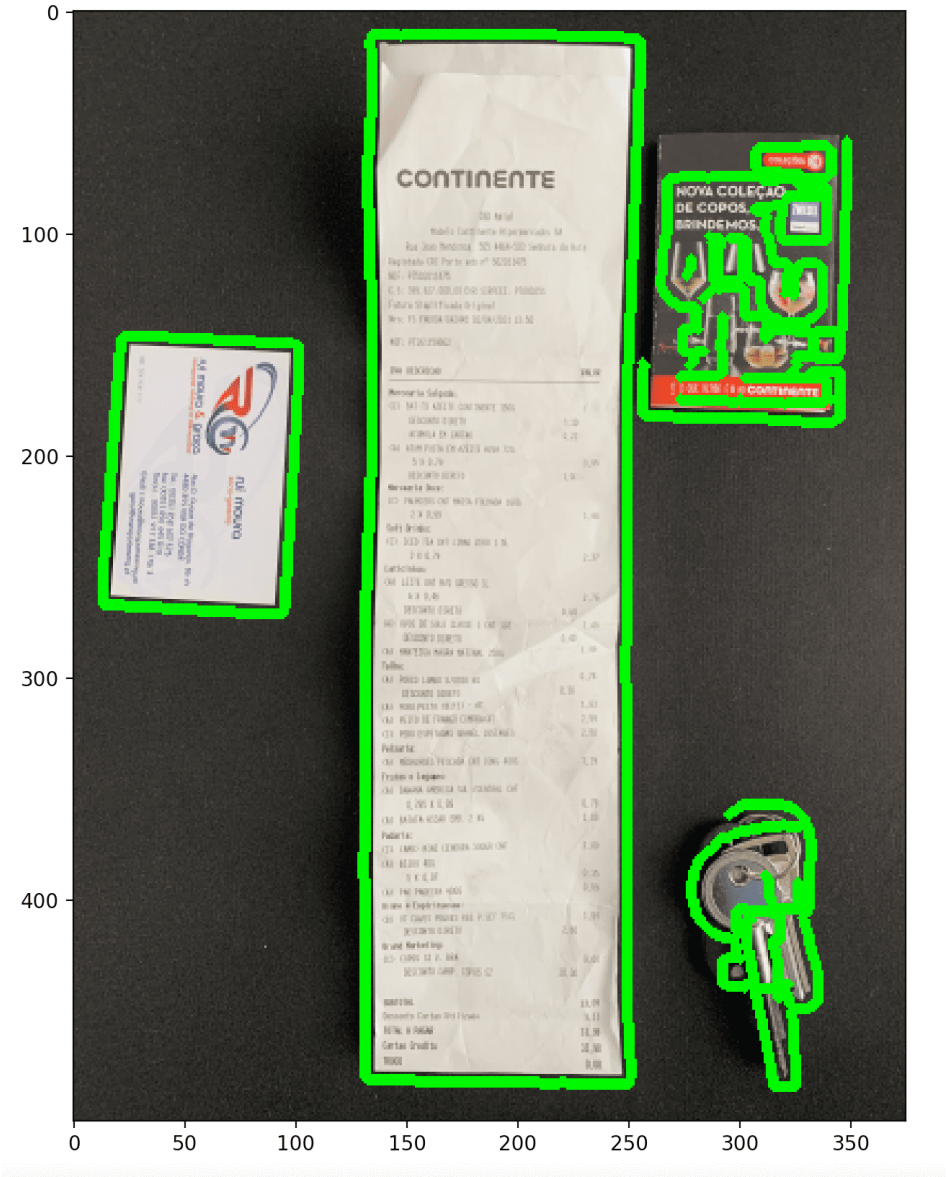

Figura 3.15: Todos os Contornos Detetados na Imagem

• Armazenar os dados de forma estruturada na base de dados.

Para extrair o texto da imagem processada recorremos à função "$image_to_data()$"da biblioteca *PyTesseract*. Ao contrário da função "$image_to_string()$"que apenas retorna uma única string com todo o texto obtido através do *Tesseract* OCR, além das palavras contidas no documento também nos indica coordenadas das posições dos blocos em que cada palavra se encontra, o nível de confiança que o algoritmo teve ao identificar determinada palavra, o número de palavras por linha, entre outras informações.

A partir das informações de saída fornecidas pela função "$image\_to\_data()$", percorre-se o texto extraído por cada linha e em cada uma destas, percorre-se cada palavra, efetuando a divisão através das entradas de novas linhas e parágrafos respetivamente. Numa função auxiliar são inicializadas algumas variáveis que irão armazenar, numa análise ini-

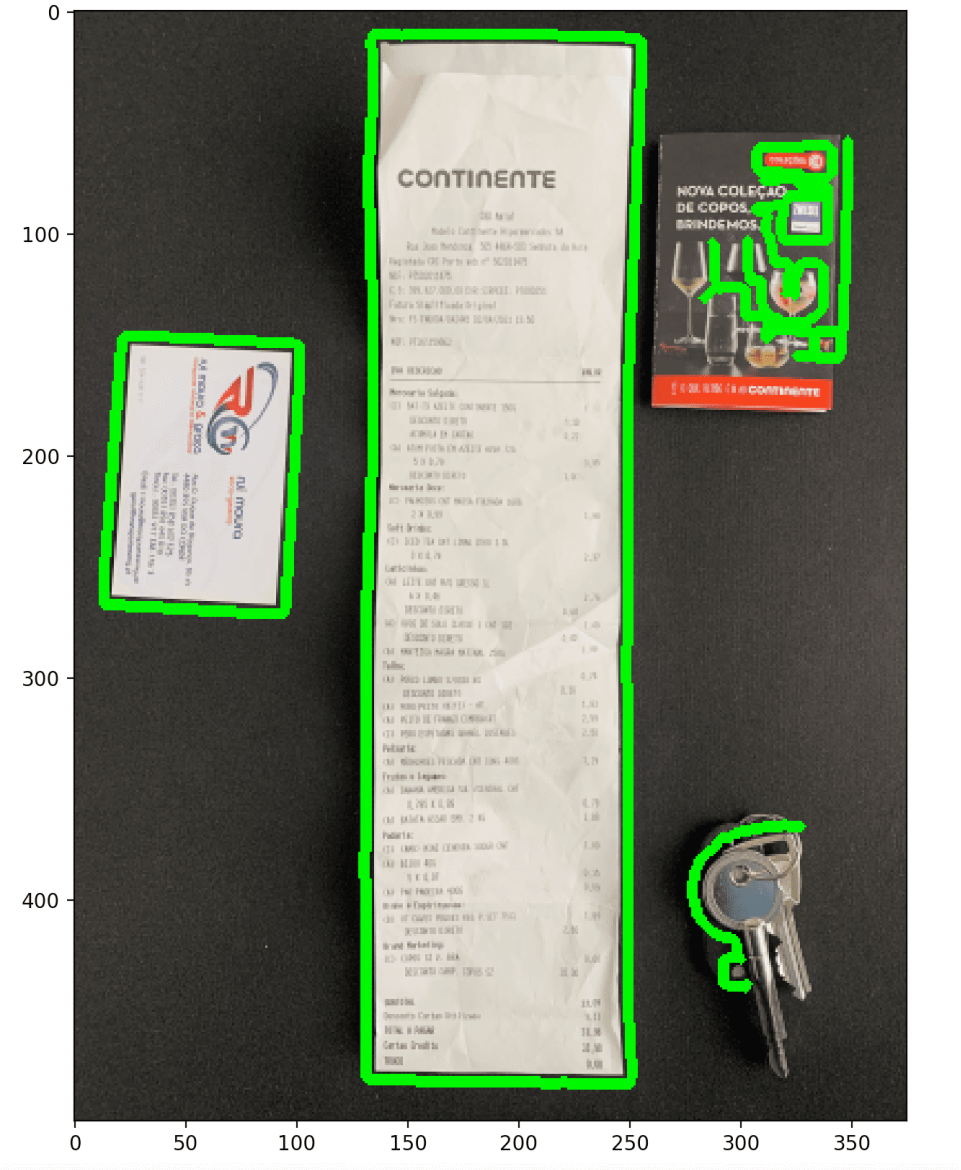

Figura 3.16: Os 8 Contornos Mais Compridos da Imagem

cial, informações da fatura como a data de compra e o supermercado. Para cada *string* detetada, o algoritmo tenta reconhecer se é um formato de data ou data e hora e se sim, extrai a mesma, tentando obter a data mais recente da fatura que geralmente é a referente ao dia de emissão. Um excerto da função que extrai o texto e procura a data é apresentado no excerto de código 7.

De seguida, se não reconhecer como data, analisa se encontra algum dos supermercados já armazenados numa lista pré definida que contém a maioria dos supermercados mais populares de Portugal, como demonstrado no excerto de código 8. Caso a data não seja encontrada no texto contido na fatura, é feita uma análise ao nome do ficheiro obtido tendo em conta que, as faturas em formato PDF descarregadas diretamente da aplicação de cliente do Continente, contêm também a data de compra no nome do ficheiro, e as

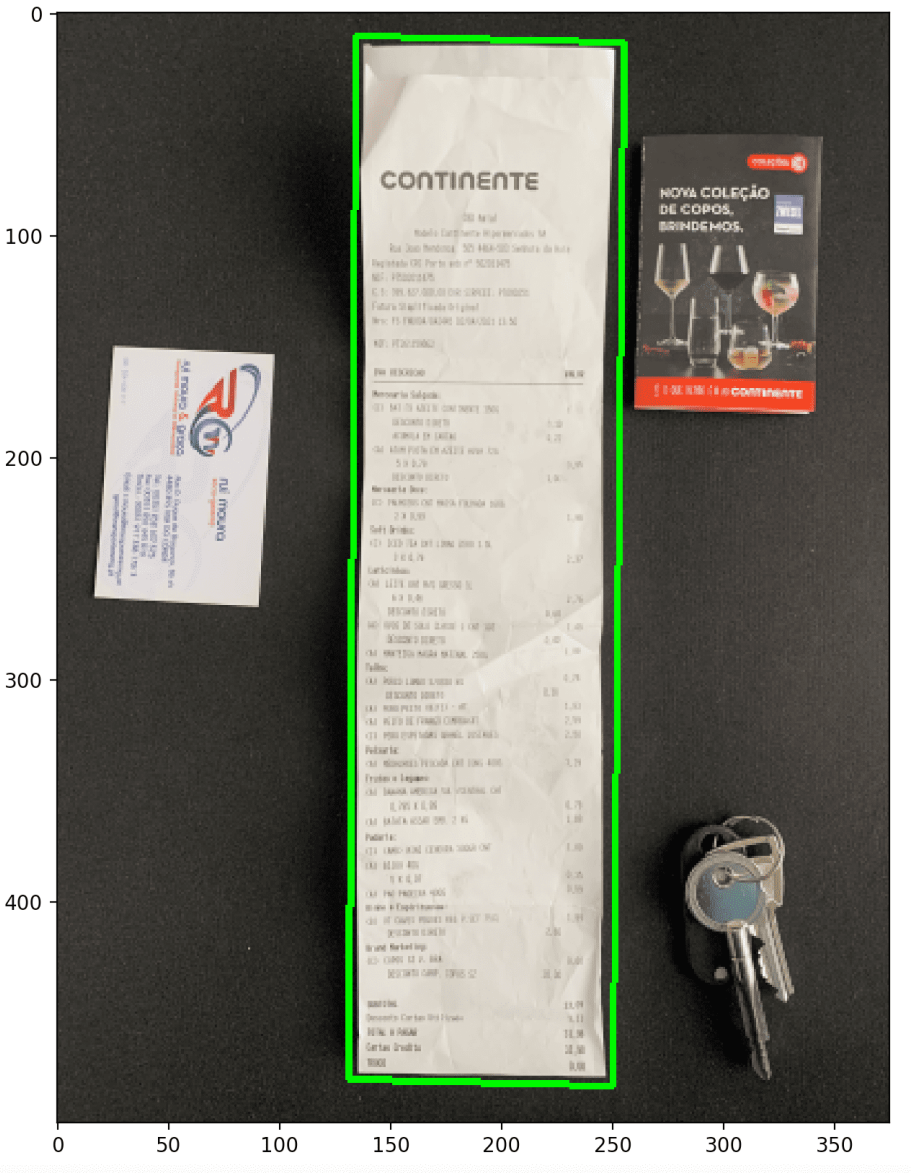

Figura 3.17: Contornos da Fatura Identificados

novas aplicações da concorrência têm vindo a adotar o mesmo padrão.

Aplicou-se um conjunto de manipulações com as informações relativas a cada bloco associado a cada *string*, nomeadamente as suas posições relativas para, em cada linha da lista dos produtos compreender de que característica do produto se poderia tratar (nome, quantidade, capacidade, preço unitário ou preço total). Desenvolveu-se assim um pedaço de algoritmo que extrai os produtos de faturas com sucesso. Estas regras foram derivadas de análises às estruturas das faturas dos supermercados portugueses de maior dimensão e de alguns mercados locais também, recorrendo a funções auxiliares para transformação e comparações de *strings*. Algumas das funções auxiliares permitem:

- Verificar certas expressões que, se detetadas, o algoritmo geral deverá ignorar a análise à linha corrente pois o elemento não interessará;

Excerto de Código 5 Funções de Apoio para Obter Retângulo a Partir de Contorno e Restaura de Perspetiva de Imagem

```
def transformarContornoNumRetangulo(contorno, racioRedimensionamento):
    pontos = contorno.reshape(4, 2)
    retangulo = np.zeros((4, 2), dtype = "float32")
    ss = pontos.sum(axis = 1)
    retangulo[0] = pontos[np.argmin(ss)]
    retangulo[2] = pontos[np.argmax(ss)]
    diferenca = np.diferenca(pontos, axis = 1)
    retangulo[1] = pontos[np.argmin(diferenca)]
    retangulo[3] = pontos[np.argmax(diferenca)]
    return retangulo / racioRedimensionamento

def restauro_perspetiva(imagem, retangulo):
    (tl, tr, br, bl) = retangulo
    largura_1 = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
    largura_2 = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
    altura_1 = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
    altura_2 = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
    max_L = max(int(largura_1), int(largura_2))
    max_A = max(int(altura_1), int(altura_2))
    dst = np.array([
        [0, 0],
        [max_L - 1, 0],
        [max_L - 1, max_H - 1],
        [0, max_A - 1]], dtype = "float32")
    M = cv2.getPerspectiveTransform(retangulo, dst)
    return cv2.warpPerspective(imagem, M, (max_L, max_A))
```

Excerto de Código 6 Aplicação de Filtro de Estilo de Documento Digitalizado

```
imagem_cinzenta = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
thresh = threshold_local(imagem_cinzenta, 22, offset = 4,
                                                method = "gaussian")
filtro_final_digitalizado = (imagem_cinzenta > thresh).astype("uint8")
                                                            * 255

plot_gray(filtro_final_digitalizado)
```

• Detetar se o texto a ser analisado no momento já pertence à zona da fatura posterior ao preço total, ou seja, a uma zona que já não contém informação relevante;

• Efetuar substituições a *strings* que, baseando-se em pares de dados pré definidos, provêm de erros típicos do OCR em determinados caracteres, deixando assim algumas palavras já corrigidas;

• Identificar se uma determinada *string* é referente à quantidade e/ou capacidade ou a um preço, baseando-se nas abreviaturas das unidades de volume respetivas e na forma de apresentação desses dados escolhida pelos supermercados;

Figura 3.18: Imagem Recortada pelos Contornos da Fatura com Perspetiva Restaurada

A certo ponto no desenvolvimento houve a tentativa de implementação de correção de palavras, devido a pequenos erros que poderiam ocorrer na extração de texto. Isto foi feito através da biblioteca *Enchant* para Python que foi desenvolvida com o objetivo de verificar palavras com erros ortográficos. O algoritmo utiliza um dicionário português pré definido e fazia substituições automáticas pela palavra do dicionário mais semelhante porém, depois de alguns testes foi removido pelo facto de nas faturas existirem nomes de produtos escritos com abreviações ou com nomes específicos de marcas ou de modelos que não existem num dicionário, o que resultava em correções erradas ou desnecessárias. Estas substituições indesejadas faziam com que o sistema armazenasse nomes de produtos muito diferentes dos originais, culminando em análises erradas ao consumo do utilizador.

O algoritmo geral para identificação das informações chave da fatura e dos produtos contidos é composto por cerca de duzentas linhas de código e, à medida que identifica

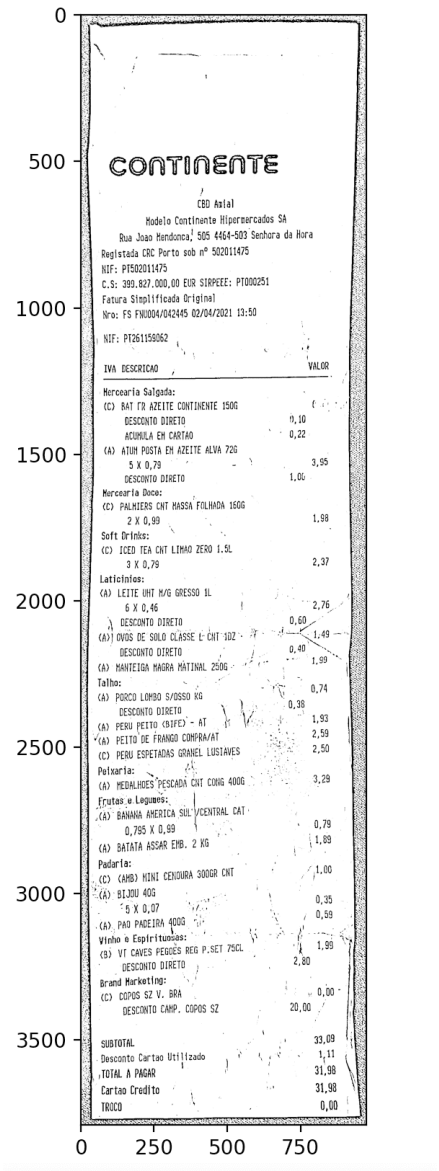

Figura 3.19: Imagem com Transformação de Cores de Estilo de Documento Digitalizado

um produto na totalidade, vai adicionando essa informação a um formato JSON. No fim do algoritmo principal, é devolvido esse ficheiro JSON para o servidor, e se necessário, armazena a mesma informação na base de dados. No Capítulo seguinte sobre as análises de resultados será exibido para uma imagem exemplo de uma fatura testada, a informação em formato JSON que o sistema consegue devolver.

3.3.3 Previsão de Consumo e Criação de Lista de Compras Automática

No momento em que o utilizador, através da aplicação móvel, pede uma sugestão de lista de compras automática, o servidor necessita inicialmente de saber quantos dias, a partir do dia atual, o utilizador pretende não necessitar de regressar a um supermercado

Excerto de Código 7 Identificação de Data nas Palavras Extraídas da Fatura

```
linhas = test_str.split('\n')
data_compra = None
supermercado = None
for linha in linhas[1:]:
    tokens = linha.split('\t')
    palavra = clean_word(tokens[-1])
    for fmt in ['%d.%m.%y', '%Y-%m-%d', '%d.%m.%y %H:%M']:
        try:
            nova_data_compra = datetime.datetime.strptime(palavra, fmt)
            if not data_compra:
                data_compra = nova_data_compra
            else:
                if nova_data_compra > data_compra
                    and nova_data_compra < datetime.datetime.now():

                    data_compra = nova_data_compra
```

Excerto de Código 8 Identificação de Supermercado nas Palavras Extraídas da Fatura e Identificação da Data no Nome do Ficheiro

```
    if not supermercado:
        for supermercado_aux in SUPERMERCADOS:
            if supermercado_aux in palavra.lower():
                supermercado = t_supermercado

#Mais abaixo no algoritmo geral:
    if not data_recibo:
        try:
            data_recibo = datetime.datetime.strptime(path.split('/')
                                                [-1],'Fatura_%d-%m-%Y.pdf')
        except:
            pass
```

para repor produtos na sua casa. De seguida o servidor acede à base de dados para obter todos os produtos diferentes (segundo o seu nome) armazenados, associados ao utilizador, realizando um pedido à base de dados.

De seguida inicia-se um primeiro ciclo que irá para cada produto distinto, inicializar ou restaurar algumas variáveis auxiliares para cálculos futuros e tem dentro de si um segundo ciclo que analisará todas as entradas na base de dados para o mesmo nome de produto. Cada entrada destas foi armazenada na base de dados a partir de diferentes faturas. No segundo ciclo o algoritmo calcula a média do custo unitário a que o produto foi comprado nas diferentes entradas para estimar quando poderá custar numa próxima compra, faz um somatório da quantidade total adquirida desse produto e identifica a data de compra mais antiga e mais recente desse produto, segundo as faturas que o utilizador registou no sistema. O excerto de código 9 apresenta o início do primeiro ciclo em que há

mais um pedido à base de dados, seguido da inicialização de variáveis e o segundo ciclo referido.

Excerto de Código 9 Análise às Entradas de Produtos Distintos na Base de Dados

```
    for x in nome_produtos_distintos_existentes:
        mycursor.execute("SELECT * FROM grocery_products_bought
                         WHERE name = '{}' AND user_ID = {};"
                            .format(x[0], user_ID))
        todos_produtos_iguais = mycursor.fetchall()
        total_quantidade_por_item = 0.0
        preco_unitario = 0.0
        preco_total_produto_quantidade_sugerida = 0.0
        data_mais_antiga = ''
        data_mais_recente = ''
        media_consumo_diario_por_produto = 0.0
        quantidade_comprada_compra_mais_recente = 0.0
        for y in todos_produtos_iguais:
            total_quantidade_por_item = total_quantidade_por_item
                                        + y[4]
            preco_unitario = round(y[6] / y[4], 2)
            data_em_analise_atual = datetime.strptime(y[7],
                                    '%Y-%m-%d %H:%M:%S')
            if data_mais_antiga == '' or ((data_em_analise_atual
                                    - data_mais_antiga).days < 0):
                data_mais_antiga = data_em_analise_atual
            if data_mais_recente == '' or ((data_em_analise_atual
                                   - data_mais_recente).days > 0):
                data_mais_recente = data_em_analise_atual
                quantidade_comprada_compra_mais_recente = y[4]
```

Ainda referente ao primeiro ciclo e após o término do segundo, através dos dados obtidos o algoritmo calcula o número de dias desde a compra mais antiga até ao dia atual do produto em análise, desde a compra mais recente até ao dia atual e o número de dias entre a compra mais antiga e a mais recente. Assume-se que o consumo médio do utilizador entre as duas compras mais distantes no tempo se manterá, então calcula-se a média dividindo a quantidade total do produto por esse número de dias. Para prever a quantidade de produto que o consumidor terá ainda armazenado em sua casa desde a última compra calcula-se a quantidade que terá consumido desde a última compra até ao dia presente (utilizando a quantidade de dias passados e a média diária de consumo estimada anteriormente). De seguida subtrai-se este bolo à quantidade de produto adquirida na compra mais recente. Caso o valor anterior resulte num número negativo é atualizado para zero. O seguinte excerto de código 10 apresenta os últimos cálculos referidos e ainda como foi calculada a quantidade estimada que o utilizador deverá então adquirir segundo o seu consumo e quantidade armazenada, o preço total para cada produto segundo a quantidade que o algoritmo irá sugerir na lista de compras final e por último, o somatório de todos os

preços totais de todos os produtos na lista de compras ou seja, o total geral.

Excerto de Código 10 Estimativa da Quantidade e Preços dos Produtos Referidos na Lista de Compras Automática

```
quantidade_produto_armazenado_estimado =
        round_down(quantidade_comprada_compra_mais_recente
        - (nr_dias_compra_mais_recente_ate_dia_atual
        * media_consumo_diario_por_produto))

if quantidade_produto_armazenado_estimado < 0:
    quantidade_produto_armazenado_estimado = 0

quantidade_estimada_que_deve_adquirir = round((nr_dias_min_prox_ida
                            * media_consumo_diario_por_produto)
                            - quantidade_produto_armazenado_estimado)

preco_total_produto_quantidade_sugerida = round(preco_unitario
                            * quantidade_estimada_que_deve_adquirir, 2)

preco_total_lista_sugestiva = round(preco_total_lista_sugestiva
                           + preco_total_produto_quantidade_sugerida, 2)
```

No fim do algoritmo, os dados obtidos durante a sua execução são armazenados numa estrutura JSON que contém: i) o nome do produto, ii) a quantidade estimada que deve adquirir desse produto, iii) o preço unitário estimado, e no fim, iv) o preço total estimado da lista de compras sugestiva. Estes dados no formato JSON são entregues à aplicação móvel através da API do servidor.

3.3.4 Aplicação Móvel e API

No servidor foi utilizado a micro *framework* Flask, escrita em Python, que permitiu desenvolver a API para comunicar com aplicações externas, neste caso, com uma aplicação móvel. Nesta API estão implementadas as funções que esperam lidar com os pedidos vindos da aplicação móvel devolvendo-lhes a informação de que necessitará para apresentar ao utilizador. As rotas mais importantes definidas na API permitem:

- Registar um novo utilizador;
- Autenticar um utilizador;
- Receber fatura em formatos de imagem para as informações serem extraídas e armazenadas;
- Receber fatura em formatos de imagem para as informações serem extraídas e armazenadas;
- Devolver as informações relativas às faturas ou produtos armazenados associados a um utilizador;

• Devolver uma lista de compras automática para um dado utilizador a partir do número de dias de segurança que indicou.

Na função responsável por registar um utilizador ocorre uma verificação antes do armazenamento dos dados do novo utilizador na base dados, impossibilitando que alguém se registe com um nome de utilizador ou email já existente. Já na função de *login* além da verificação se os dados da tentativa de autenticação correspondem a algum utilizador registado na base de dados. No caso de se confirmar a correspondência, é gerado um novo *token* e devolvido à aplicação móvel, como apresentado no excerto de código 11, para esta poder posteriormente aceder às restantes rotas com sucesso. O mesmo excerto de código 11 mostra inicialmente a função que gera o *token* em que é definido quando expira, ou seja, o momento em que deixa de ser válido e o algoritmo usado na encriptação. O HS256 baseia-se na combinação de uma função de *hash* e uma chave secreta, que se encontra apenas do lado do servidor, pois a mesma chave serve para validação e a segurança não pode ser comprometida.

Excerto de Código 11 Criação de um Token na Autenticação

```
def codificar_token_autenticacao(user_id):
    try:
        payload = {
            'exp': datetime.datetime.utcnow()
                    + datetime.timedelta(days=0,
                                         minutes=5),
            'iat': datetime.datetime.utcnow(),
            'sub': user_id
        }
        return jwt.encode(
            payload,
            app.config.get('SECRET_KEY'),
            algorithm='HS256'
        )
    except Exception as e:
        return e


token = codificar_token_autenticacao(x[0])
        valor = {
            "token": token,
            "status": 200,
        }
        return valor
```

As restantes rotas irão requisitar um *token* para continuarem a sua execução, confirmando que existe um utilizador autenticado. A função responsável por descodificar e validar o *token* devolve o identificador do utilizador correspondente assim como uma

mensagem de erro caso o tempo viável para uso já tenha sido ultrapassado ou no caso do *token* ser simplesmente inválido perante a chave secreta conhecida.

As funções associadas às rotas remanescentes fazem simples pedidos à base de dados quando necessário e chamam as funções principais do servidor para processar a fatura e a lista de compras. No momento em que recebem as imagens ou o ficheiro PDF codificados no formato *base64*, estes são convertidos para objetos de tipo *bytes*.

A aplicação móvel desenvolvida em Flutter contém também as funções que fazem os pedidos HTTP para as rotas da API. Estas enviam os dados do utilizador em formato JSON quando necessário, esperam a resposta do servidor e lançam uma exceção no caso de não receber um resultado de sucesso. São funções assíncronas para que a aplicação não tenha de parar por completo enquanto são feitos os pedidos e se aguarda as respetivas respostas. O seguinte excerto de código 12 exemplifica a estrutura destas funções, semelhantes entre si, com a função de autenticação que começa por preparar o caminho de destino para envio do pedido, prepara o pedido com os cabeçalhos e os dados da tentativa de autenticação do utilizador e espera a resposta descodificando a resposta no caso de ser bem sucedido ou retornando uma exceção.

Excerto de Código 12 Envio e Receção de Pedido de Login HTTP A Partir da Aplicação Móvel

```
Future<Token> login(UserLogin userLogin) async {
  final _loginURL = _base + "login";

  final http.Response response = await http.post(
    _loginURL,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(userLogin.toDatabaseJson()),
  );
  if (response.statusCode == 200) {
    return Token.fromJson(json.decode(response.body));
  } else {
    print(json.decode(response.body).toString());
    throw Exception(json.decode(response.body));
  }
}
```

Para a autenticação a partir da aplicação móvel foi utilizado o *Business Logic Component* (BLoC), um padrão criado pela Google que faz a gestão de estados da aplicação através de programação reativa controlando o fluxo de dados. Neste caso, foi implementado um BLoC que serve de intermediário entre a resposta da API do servidor e os estados que a aplicação deverá apresentar. Ao iniciar a aplicação, é instanciado um repositório para armazenar os dados de um utilizador e o BLoC para autenticação. Perante o estado da autenticação, no caso do utilizador não estar autenticado é levado para a vista de *login*

(figura 3.20) ou se estiver é levado para a vista onde pode importar uma fatura.

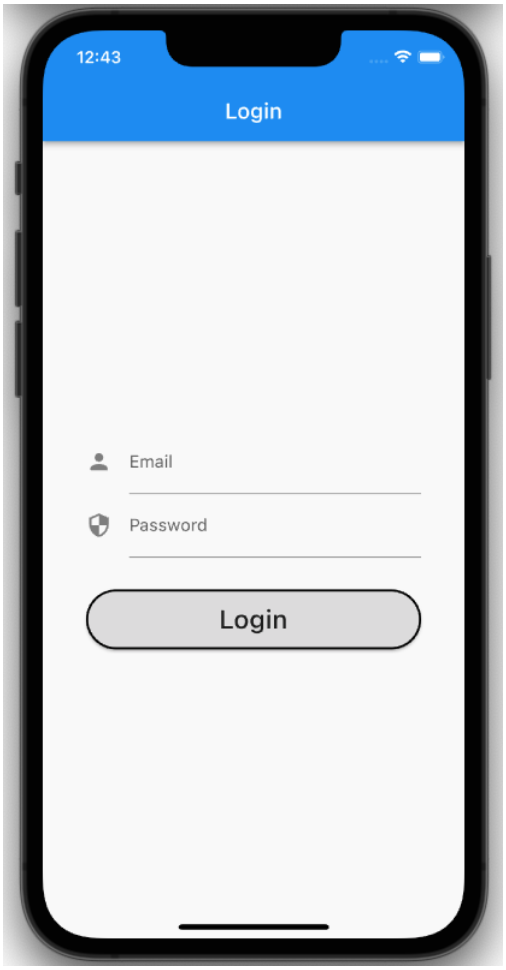

Figura 3.20: Página de Login da Aplicação Móvel

Na vista apropriada para o envio de faturas para o servidor, figura 3.21, contém inicialmente um botão em que o utilizador pode selecionar uma das três opções, figura 3.22, nomeadamente tirar uma fotografia no momento, selecionar fotografias do álbum de imagens do *smartphone* ou selecionar um ficheiro da pasta de documentos, figura 3.23. Para a implementação desta vista foram utilizadas as bibliotecas *FilePicker* e *ImagePicker* para facilitar o acesso aos documentos do telemóvel.

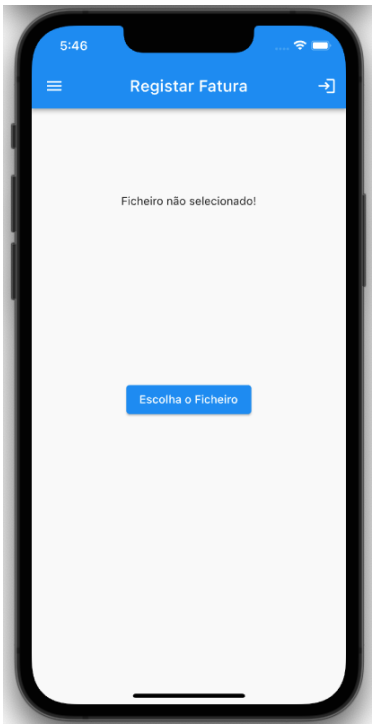

Figura 3.21: Página da Aplicação Móvel para Envio de Fatura

Figura 3.22: Escolha do Método de Seleção de Fatura

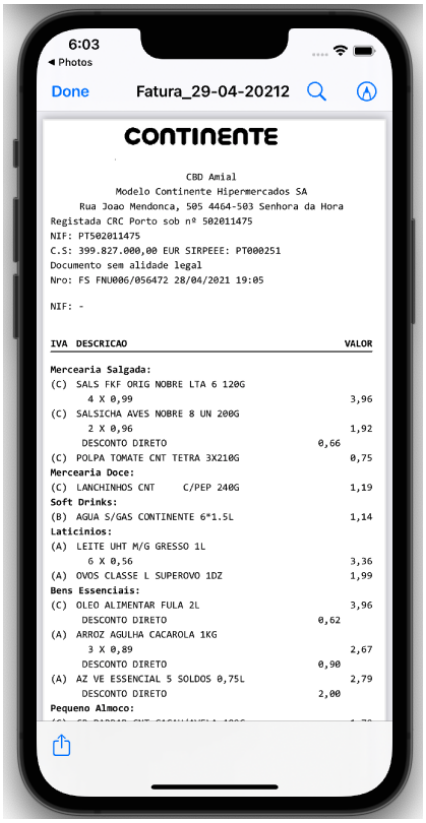

Figura 3.23: Exemplo de Seleção de PDF a Partir dos Documentos

A aplicação móvel contém um *widget* chamado AppBar que consiste num menu late-

ral que permite selecionar outras páginas disponíveis segundo apresentado na figura 3.24. A figura 3.25 representa uma listagem exemplo de faturas armazenadas na base de dados do servidor, sendo que a vista de listagem de produtos tem aspeto visual semelhante. A página para pedido de lista de compras automática, figura 3.26, exige apenas que o utilizador insira o número de dias para os quais pretende ficar com reservas de produtos suficientes e de seguida é mostrada segundo o exemplo da figura 3.27. Em todas as páginas referidas anteriormente, quer no momento em que o utilizador prime o botão para submeter os dados ou no momento em que abre as páginas, as vistas chamam as funções dos controladores da aplicação móvel que por sua vez comunicam com a API do servidor à espera da resposta associada. Recorrendo ao sistema de autenticação via BLoC torna-se mais simples enviar o *token* do utilizador autenticado no momento, juntamente nas chamadas aos controladores.

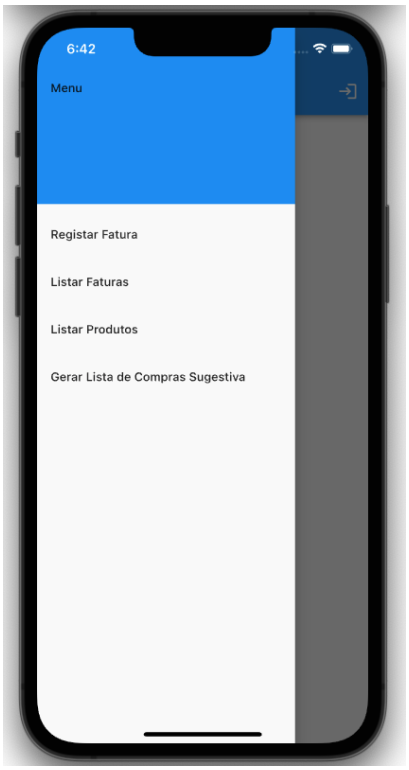

Figura 3.24: Menu Lateral da Aplicação Móvel

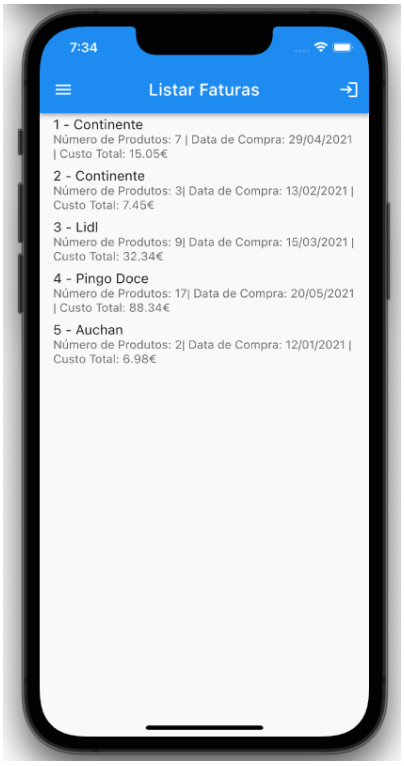

Figura 3.25: Exemplo de Listagem de Faturas Registadas de Um Utilizador

Figura 3.26: Página para Pedido de Lista de Compras Sugestiva

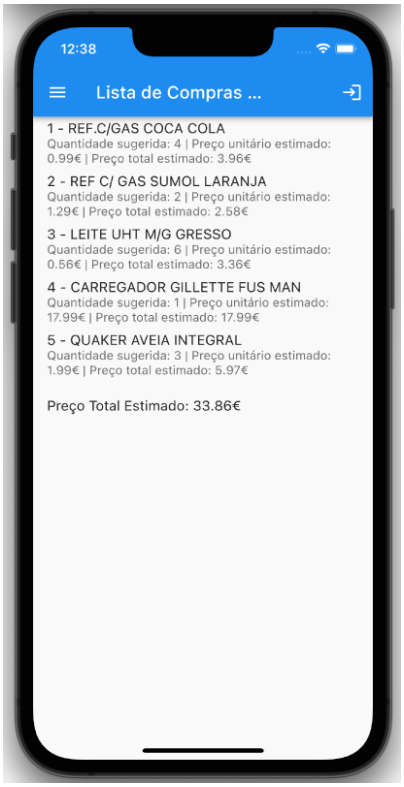

Figura 3.27: Exemplo de Lista de Compras Sugestiva

Capítulo 4

Avaliação do Sistema de Lista de Compras Automáticas

Este Capítulo é composto por três partes. Na primeira, apresenta-se para um exemplo de fatura importada, o texto e os produtos que o sistema consegue extrair. Na segunda parte descreve-se o método utilizado para comparar a precisão da extração de texto via OCR e analisa-se os resultados obtidos para diferentes níveis de pré processamento de imagem da fatura. Por fim, apresenta-se a lógica do algoritmo que avalia o grau de precisão das informações extraídas relativas à fatura e aos produtos incluídos a partir do texto extraído às mesmas faturas e com as mesmas variações de pré processamento da segunda parte.

Para as duas últimas partes deste Capítulo foram utilizadas seis faturas diferentes para os testes, em que para cada uma se utilizaram diferentes versões, nomeadamente a fatura capturada inteiramente com apenas uma fotografia, diversas imagens da fatura segmentada e a versão PDF da mesma, quando disponível. Estas seis faturas pertencem a quatro estabelecimentos diferentes, tais como Continente, Modelo, Lidl e SportZone. As versões PDF utilizadas foram obtidas a partir da aplicação móvel oficial do respetivo estabelecimento, através da utilização do cartão de cliente. Os testes com o método de importação de fatura através de várias imagens parciais da mesma foram realizados de forma a simular um cenário real, com uma só tentativa de captura, ou seja, não foram feitas diversas tentativas nem manipulação de imagens até que as imagens fossem todas corretamente unidas. Consequentemente houveram alguns resultados prejudicados por falta de partes da fatura nos casos em que a união não foi bem sucedida.

4.1 Apresentação de Resultados

Nesta secção são apresentados os resultados de alguns passos do sistema, para um exemplo de uma fatura importada, com o objetivo de descrever todo o processo e os

dados que foram utilizados na avaliação dos componentes do sistema.

Uma fatura do Continente em formato PDF foi importada para o servidor, descarregada diretamente a partir da aplicação móvel "Cartão Continente", como apresentada na figura 4.1. A figura seguinte não representa a totalidade do ficheiro PDF, uma vez que foi recortada, pelo facto do restante não interessar para o algoritmo de categorização da informação.

CONTINENTE

CBD Amial
Modelo Continente Hipermercados SA
Rua Joao Mendonca, 505 4464-503 Senhora da Hora
Registada CRC Porto sob nº 502011475
NIF: PT502011475
C.S: 399.827.000,00 EUR SIRPEEE: PT000251
Documento sem alidade legal
Nro: FS FNU006/056471 28/04/2021 19:03

NIF: -

IVA	DESCRICAO		VALOR
Mercearia Salgada:			
(A)	ATUM POSTA EM AZEITE GALEAO 72		
	6 X 0,82		4,92
	DESCONTO DIRETO	1,62	
Soft Drinks:			
(A)	NECTAR COMP 3*20CL		
	2 X 1,19		2,38
	DESCONTO DIRETO	0,40	
(A)	NECTAR COMPAL CL LAR.ALGARVE 1L		
	2 X 1,24		2,48
	DESCONTO DIRETO	0,50	
(B)	AGUA S/GAS LUSO 6*50CL		1,88
	DESCONTO DIRETO	0,47	
Beleza:			
(A)	DISC DE ALGODAO DESMAQ MYLABEL		0,85
Bens Essenciais:			
(A)	FAR.TR.EXT.FINA S/F NACIONAL 1K		0,96
(C)	PAO RALADO CONTINENTE 500G		0,69
Padaria:			
(A)	PAO FORMA S/CODEA INTEGRAL CNT		
	2 X 1,99		3,98

SUBTOTAL	**18,14**
Desconto Cartao Utilizado	2,12
TOTAL A PAGAR	**16,02**
Cartao Credito	**16,02**
TROCO	**0,00**

Figura 4.1: Parte da Fatura Exemplo do Continente

Visto que se trata de um ficheiro em formato PDF ao invés de uma ou mais fotografias reais da fatura, como explicitado na implementação, não ocorrerá pré processamento, passando diretamente para a fase de extração de texto. O texto obtido para a fatura da

imagem 4.1 é apresentado na figura 4.2.

CONTINENTE
CBD Amial
Modelo Continente Hipermercados SA
Rua Joao Mendonca, 505 4464-503 Senhora da Hora
Registada CRC Porto sob nº 502011475
NIF: PT502011475
C.S: 399.82/7.000,00 EUR SIRPEEE: PTOORO251
Documento sem alidade legal
Nro: FS FNUBO6/056471 28/04/2021 19:03
NIF: -
IVA DESCRICAO VALOR
Mercearia Salgada:
(A) ATUM POSTA EM AZEITE GALEAO 72
6 X 9,82 4,92
DESCONTO DIRETO 1,62
Soft Drinks:
(A) NECTAR COMP 3*20CL
2 X1,19 2,38
DESCONTO DIRETO 0,40
(A) NECTAR COMPAL CL LAR.ALGARVE 1L
2X1,24 2,48
DESCONTO DIRETO 0,50
(B) AGUA S/GAS LUSO 6*50CL 1,88
DESCONTO DIRETO 0,47
Beleza:
(A) DISC DE ALGODAO DESMAQ MYLABEL 0,85
Bens Essenciais:
(A) FAR.TR.EXT.FINA S/F NACIONAL 1K 0,96
(C) PAO RALADO CONTINENTE 500G 0,69
Padaria:
(A) PAO FORMA S/CODEA INTEGRAL CNT
2 X 1,99 3,98
SUBTOTAL 18,14
Desconto Cartao Utilizado 2,12
TOTAL A PAGAR 16,02
Cartao Credito 16,02
TROCO 0,00

Figura 4.2: Texto Extraído da Fatura Exemplo do Continente

A partir do texto extraído apresentado acima, que obteve uma precisão de 98,7% segundo a similaridade de Levenshtein normalizada explicada na parte seguinte sobre a avaliação da extração de texto, o algoritmo de categorização da informação conseguiu extrair os dados apresentados nas duas caixas de texto seguintes.

([{"item":"ATUM POSTA EM AZEITE GALEAO",
"capacidade unitária":,
"quantidade":"505 502011475 72 6 X",
"preço":4.92},
{"item":"NECTAR COMP X1,19",
"capacidade unitária":"3*20CL",
"quantidade":"2",
"preço":2.38},
{"item":"NECTAR COMPAL CL LAR.ALGARVE 2X1,24",
"capacidade unitária":"1L",
"quantidade":,
"preço":2.48},
{"item":"AGUA S/GAS LUSO",
"capacidade unitária":"6*50CL",
"quantidade":,
"preço":1.88},

{"item":"DISC DE ALGODAO DESMAQ MYLABEL",
"capacidade unitária":,
"quantidade":,
"preço":0.85},
{"item":"FAR.TR.EXT.FINA S/F NACIONAL 1K",
"capacidade unitária":,
"quantidade":,
"preço":0.96},
{"item":"PAO RALADO CONTINENTE",
"capacidade unitária":"500G",
"quantidade":,
"preço":0.69},
{"item":"PAO FORMA S/CODEA INTEGRAL CNT",
"capacidade unitária":,
"quantidade":"2 X",
"preço":3.98},
{"Subtotal":18.14},
{"Total":16.02},],
datetime.datetime(2021,4,28,0,0),
"continente")

Verificamos que o algoritmo foi capaz de reconhecer os oito produtos existentes na fatura, além de reconhecer o subtotal, o total, a data e o nome do supermercado. Não

reconheceu o texto de todos os campos com um grau de exatidão de cem por cento, mas foi muito próximo e esta mesma fatura será analisada mais detalhadamente nas duas partes seguintes. Alguns dos campos encontram-se vazios pelo facto dos produtos não conterem essa informação ou por o algoritmo não ter sido capaz de detetar corretamente. Estas informações seguiriam para a fase de armazenamento na base de dados, associadas ao utilizador que a importou, e os dados seriam utilizados futuramente para consulta ou previsão de consumo.

4.2 Avaliação da Extração de Texto

Numa primeira análise compararam-se os resultados da extração de texto a partir da fatura para averiguar a importância dos métodos de pré processamento de imagem que se antecede e que, por sua vez teriam impacto na obtenção das informações dos produtos que ocorre numa fase seguinte. Mesmo para as faturas em formato PDF, que é um cenários preferível face às imagens devido à sua qualidade e não existência de ruídos que facilita a extração de texto, é importante verificar se o nível de precisão na extração é perfeito para não se culpar na totalidade o algoritmo de categorização de texto, caso não identifique corretamente todos os produtos e informações gerais.

Para esta análise, para cada uma das seis faturas utilizadas, foi criado manualmente o resultado esperado, ou seja, as comparações foram feitas sempre face a uma *string* com o texto que representaria o cenário ideal. As variáveis deste estudo, além das três formas de inserção da fatura no sistema, foram definidas por um de três níveis de pré processamento: nenhum, todos processamentos exceto o último filtro de *threshold* que dá um efeito tipo documento digitalizado, e por último, o pré processamento completo.

A biblioteca *StrSimpy* foi utilizada para aceder às implementações dos algoritmos de comparação de *strings*, nomeadamente os algoritmos de Levenshtein, Levenshtein normalizada e o de JaroWinkler. Instancia-se um simples objeto para cada um dos algoritmos de comparação e de seguida utiliza-se as funções associadas relativas ao cálculo de distância ou similaridade que recebe como argumentos duas *strings* e imprime o valor calculado, como demonstrado no excerto de código 13.

Na primeira fatura analisada, do Continente com um preço total de dezasseis euros, apresentada anteriormente na figura 4.1, verifica-se perante o gráfico da figura 4.3 que a versão PDF obtém uma percentagem de precisão de extração de texto correta mais alta, com um valor muito próximo do máximo possível. Assume-se perante este exemplo que o sistema conseguirá resultados melhores se a fatura for importada com o ficheiro original descarregado da aplicação do supermercado.

A versão da fatura capturada numa só imagem face à versão composta por várias partes da fatura em diferentes imagens obteve resultados muito próximos, o que significa que a união das imagens pode ser realizada com sucesso. No método de imagens da fatura

Excerto de Código 13 Implementação do Cálculo da Distância e Similaridade Entre Strings

```
levenshtein = Levenshtein()
normalized_levenshtein = NormalizedLevenshtein()
jarowinkler = JaroWinkler()

print('Distancia Levenshtein: '
                + str(levenshtein.distance(string1, string2)))

print('Simiralidade Levenshtein Normalizada: '
    + str(normalized_levenshtein.similarity(string1, string2)))

print('Simiralidade Jarowinkler: '
              + str(jarowinkler.similarity(string1, string2)))
```

segmentada, a união foi bem sucedida, pois toda a fatura foi analisada. Esta fatura era de dimensão curta, pelo que não haveria necessidade de se capturar diferentes imagens, mas serviu para teste, que concluiu ser um sucesso, visto que o resultado final batalha diretamente com uma só imagem que apanharia toda a fatura com boa aproximação e consequentemente boa qualidade. O gráfico da figura 4.3 representa os níveis de similaridade sem nenhum pré-processamento aplicado, em nenhum dos casos.

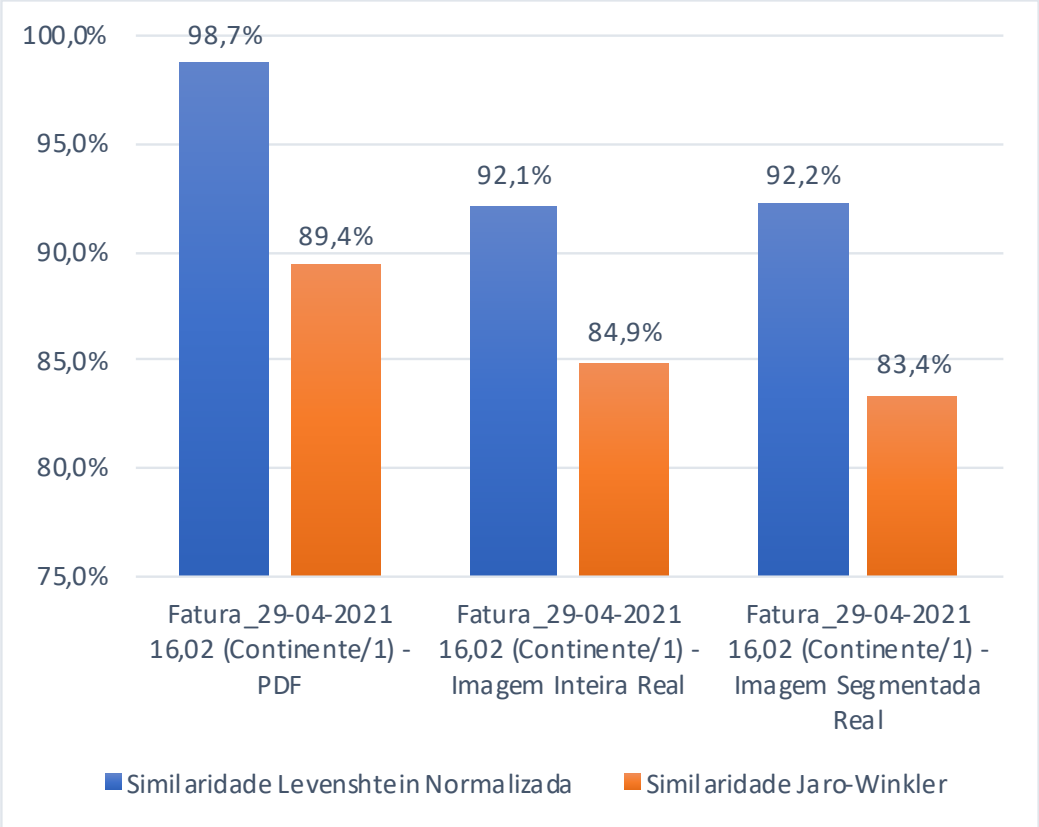

Figura 4.3: Comparação aos Três Métodos de Inserção de Fatura

No gráfico da figura 4.4 temos uma comparação do nível de precisão da extração de texto de duas faturas diferentes, novamente sem pré processamento algum, nas versões de imagens únicas ou de várias imagens da fatura segmentada. Constata-se que a segunda fatura do Continente, mais à direita do gráfico, obteve uma descida acentuada no grau de precisão. Isto deveu-se à união não ter sido realizada com sucesso, ficando a faltar a última parte da fatura na imagem final em que ocorreu a extração de texto. Como esperado, em comparação ao cenário ideal que contém todo o texto esperado da fatura completa, essa falta de texto resultou em apenas 31,7% de precisão segundo a similaridade de Levenshtein normalizada. Nota-se também que os resultados da imagem inteira da segunda fatura se aproximam dos resultados da primeira.

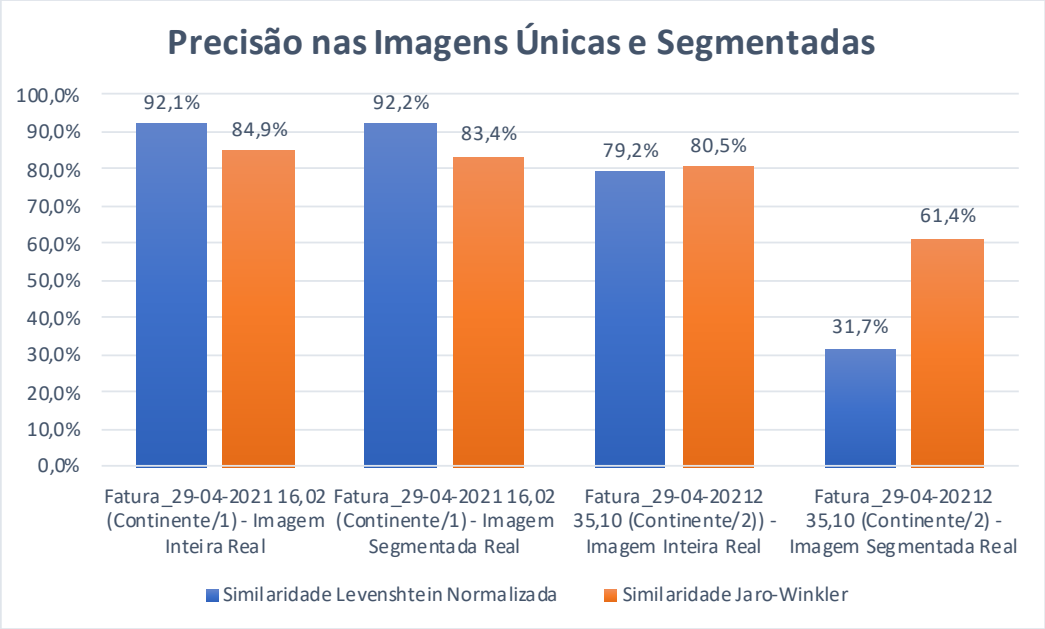

Figura 4.4: Comparação a Duas Faturas em Imagens Únicas e Imagens Segmentadas

Ainda sobre a precisão da extração de texto das imagens únicas, sem pré processamento, observa-se pelo gráfico da figura 4.5 que o sistema consegue valores próximos a 80%, em média, na similaridade de Levenshtein normalizada. Já nas faturas do Lidl e SportZone, por serem as que se encontravam num estado pior de impressão e de preservação, têm assim os valores mais baixos justificados.

Para o exemplo da fatura da SportZone não foi possível obter a versão digital diretamente de uma aplicação oficial, pelo facto desta não existir. Nas restantes cinco faturas analisou-se o nível de precisão aos ficheiros obtidos diretamente das suas aplicações oficiais, sendo que na aplicação do Lidl obtemos um ficheiro em JPEG, mas no servidor foi inserido pelo mesmo método caso fosse um ficheiro PDF. Apercebe-se pelo gráfico da figura 4.6 que nestas versões digitais de qualidade alta e sem ruídos, se obtém similaridades de Levenshtein normalizada próximas de 100% e de Jaro-Winkler próximas de 90% com consistência.

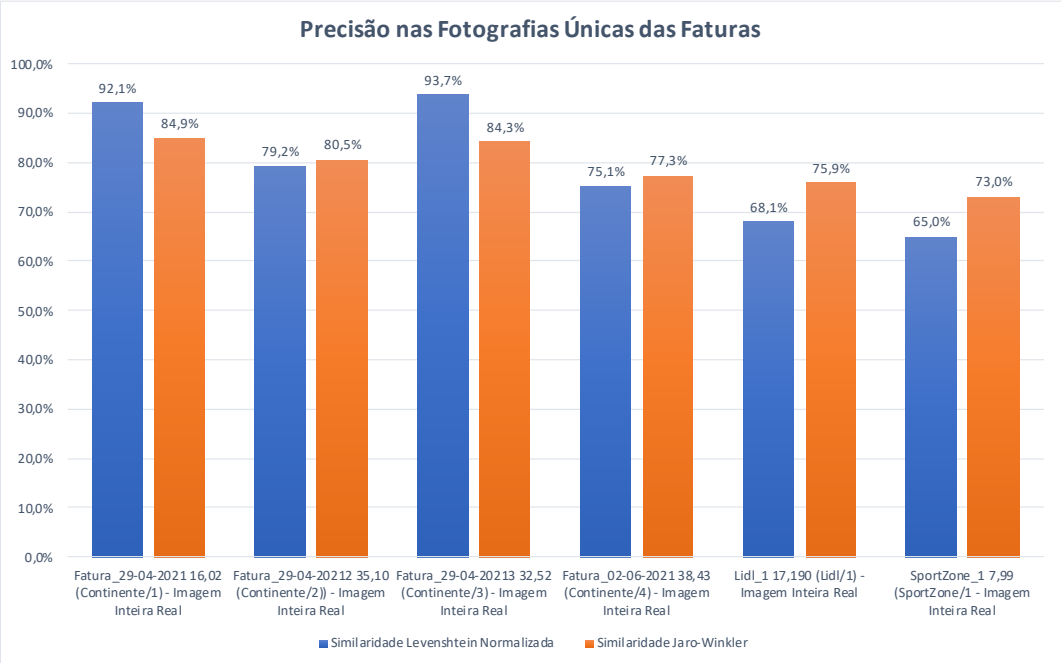

Figura 4.5: Comparação da Precisão em Imagens Únicas dem Pré-Processamento

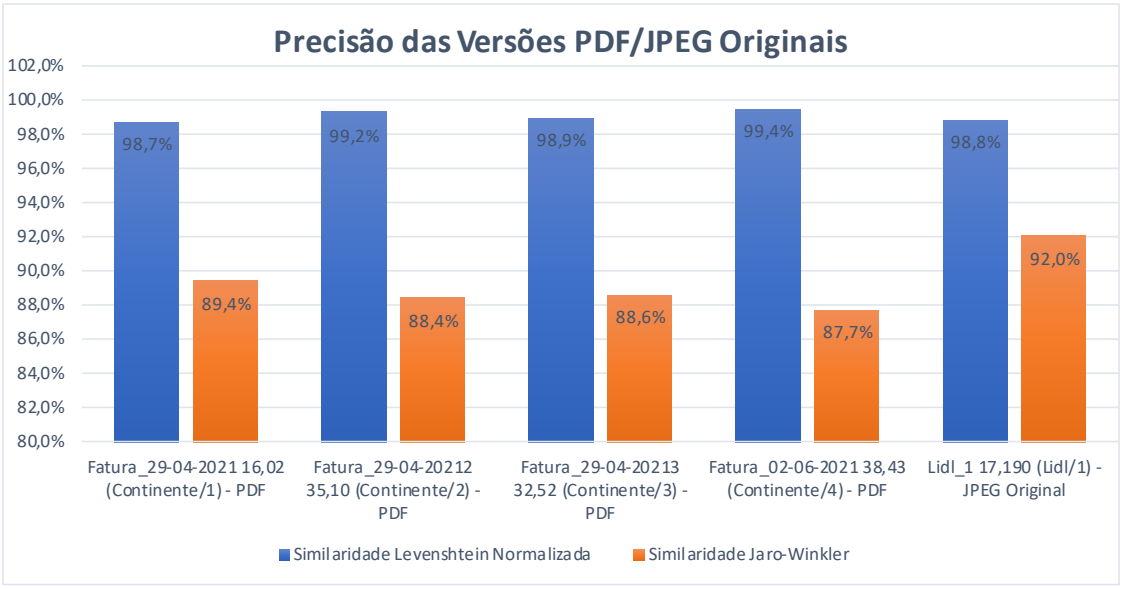

Figura 4.6: Comparação da Precisão nas Faturas Obtidas das Aplicações do Respetivo Supermercado

O seguinte gráfico da figura 4.7 apresenta os resultados relativos à análise da primeira fatura do Continente referida anteriormente, desta vez comparando os diferentes níveis de pré-processamento aplicados. Os três conjuntos de cores, cor preta, alaranjada e verde, representam os resultados da precisão da extração de texto sem pré-processamento à imagem, com pré-processamento mas sem o efeito final de *threshold* e com pré-processamento com o efeito, respetivamente. Em cada par de barras, a de tom mais escuro representa a versão da fatura em imagem única e a barra de tom mais claro

representa a fatura submetida em diferentes imagens. Constatamos que, de forma geral, há um aumento na precisão quando se aplica pré-processamento sem efeito *threshold* face ao sem pré-processamento. Há também uma descida acentuada nos resultados com pré-processamento e efeito *threshold*, atingindo até níveis bem abaixo da versão sem pré-processamento.

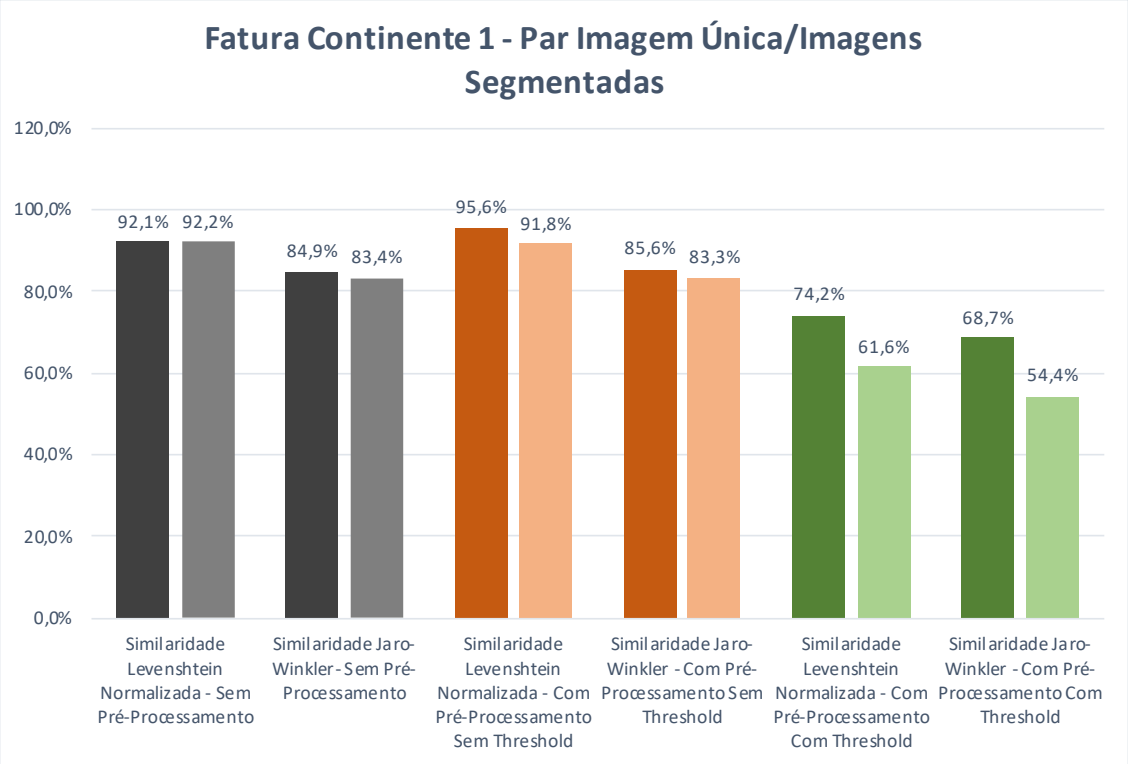

Figura 4.7: Comparação de uma Fatura em Imagem Única e Imagens Segmentadas em Diferentes Níveis de Processamento

Observando o gráfico da figura 4.8 que apresenta as similaridades de Levenshtein normalizada para as diversas imagens únicas das diversas faturas testadas, conclui-se que na maior parte dos casos se obtém melhores resultados aplicando o pré-processamento desenvolvido sem o efeito final de *threshold*. Porém no caso da fatura do Continente 2 houve um aumento de precisão em 0,3% com a aplicação do efeito face à sua não aplicação e, na fatura de Continente 3 houve uma diminuição de precisão de 0,8% no caso da aplicação de pré-processamento sem efeito face à versão sem processamento algum, parece que neste último caso o pré-processamento não ajudou. Para suportar a análise aos resultados gerais do sistema, a fatura da SportZone apresentou resultados com pré-processamento sem *threshold* muito superiores aos dois outros níveis de pré-processamento, diferenças superiores a 25% para ambas as comparações.

Perante a similaridade de Jaro-Winkler, nos testes feitos às faturas com imagens únicas aplicando o pré-processamento sem o *threshold* final obtiveram sempre os melhores resultados face aos outros dois níveis de pré-processamento. No gráfico da figura 4.9, da esquerda para a direita, verifica-mos que existe um padrão que contém uma barra só,

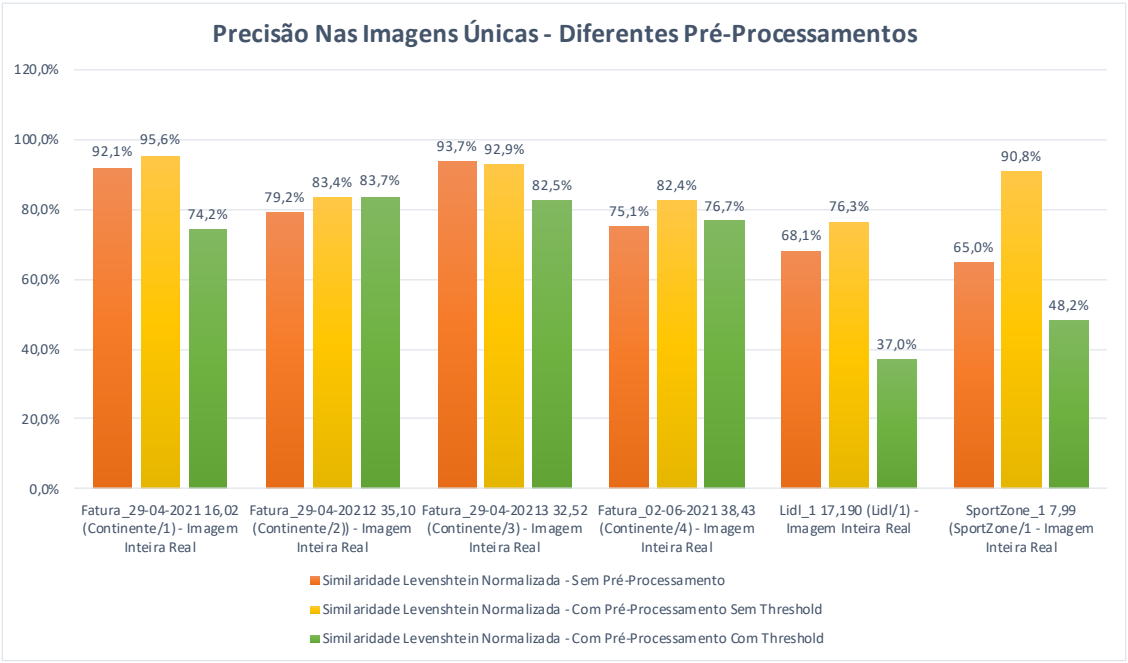

Figura 4.8: Comparação das Imagens Únicas das Faturas para Diferentes Níveis de Pré-Processamento

seguida de um conjunto de três barras juntas, sendo que a primeira representa a similaridade de Jaro-Winkler da fatura obtida digitalmente e as três barras seguintes representam os resultados da mesma fatura capturadas com numa imagem única, nos diferentes modos de pré-processamento. Além da confirmação da preferência pela não aplicação do efeito *threshold*, constata-se que as versões PDF/JPEG originais têm sempre melhores resultados de extração de texto do que com qualquer tipo de pré-processamento nas fotografias capturadas pelo utilizador.

4.3 Avaliação da Categorização do Texto Extraído

Visto que obter a informação correta dos diversos produtos apresentados nas faturas consiste numa das capacidades fulcrais para o funcionamento de todo o sistema, também se procedeu à avaliação da categorização do texto, que está também dependente da qualidade da extração do texto ocorrida anteriormente.

Para a análise a este componente foi desenvolvido um algoritmo que recebe duas *strings* num formato JSON, preparado para percorrer os seus conteúdos segundo a estrutura de dados que o algoritmo de categorização de texto retorna, e efetuar comparações entre os seus elementos. O algoritmo espera receber numa *string* os dados do cenário ideal, preenchido manualmente com exatamente o conteúdo que uma dada fatura contém e com a informação associada à categoria correta e recebe outra *string* que representa o resultado de uma categorização de texto ocorrida pelo sistema.

Após a descodificação da informação no formato JSON através da função "json.loads()",

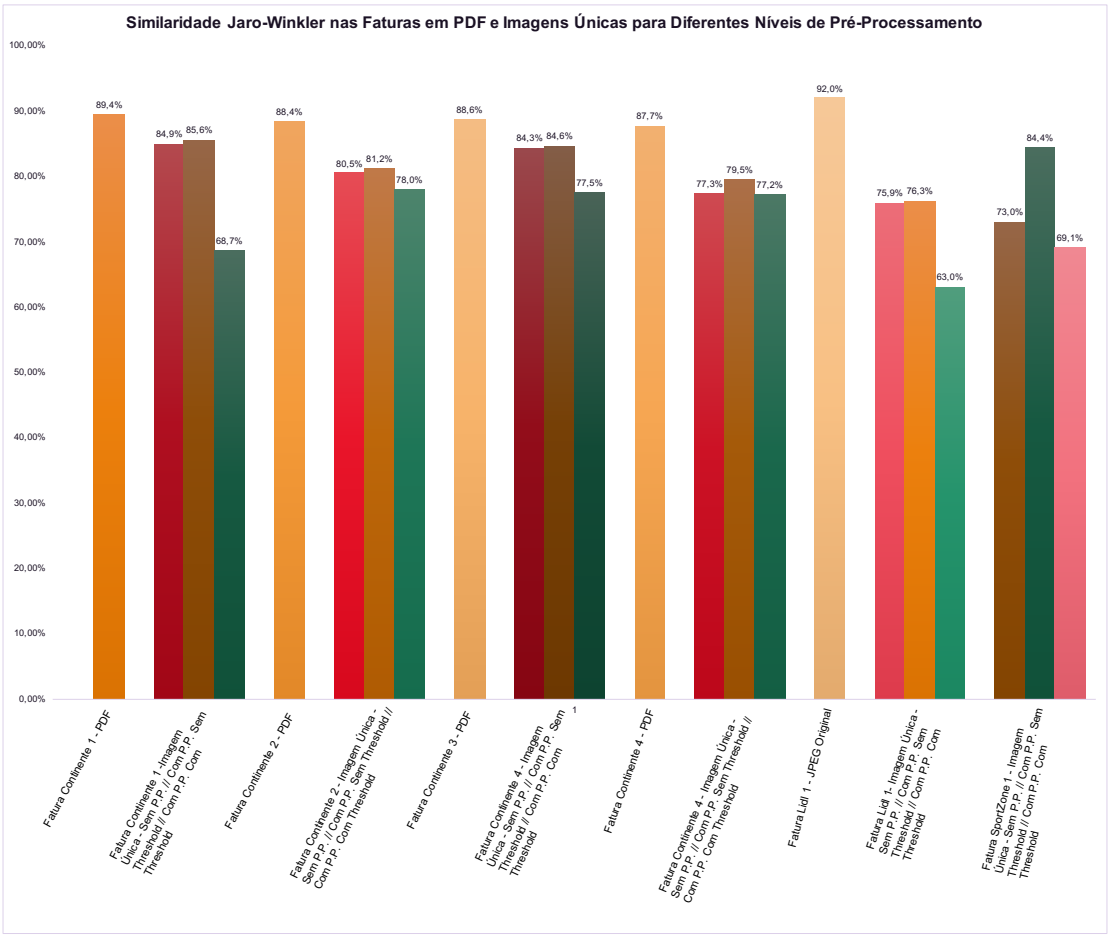

Figura 4.9: Comparação das Faturas em PDF e Imagens Únicas para Diferentes Níveis de Pré-Processamento

para cada produto identificado e para cada uma das categorias de informação (nome, capacidade, quantidade, preço), é calculada a similaridade de Levenshtein normalizada entre as *strings* desse elemento, por exemplo, o nome do produto do cenário ideal "ovos"é comparado com o nome "ovoss", o preço esperado de "1.99"é comparado com o preço obtido de "7.99", e todos os valores de similaridade vão sendo guardados em diferentes variáveis. À medida que o algoritmo percorre todas as categorias de todos os produtos, para cada categoria, vai atualizando a média da similaridade para cada categoria. No final, o algoritmo retorna o número de produtos identificados com sucesso face ao número de produtos esperado e a média de similaridade de Levenshtein normalizada para cada uma das quatro categorias.

Através dos resultados apresentados na tabela 4.1 pode-se verificar que informações foram corretamente categorizadas a partir do texto extraído através de OCR para as diversas faturas sem a aplicação de algum pré-processamento. Nas cinco primeiras faturas, nas versões PDF e JPEG original, o algoritmo identificou corretamente todos os produtos existentes na fatura à exceção da fatura Continente 2. De notar que a forma como se considerou se um produto foi identificado corretamente ou não depende apenas de um mínimo

valor de similaridade calculado entre o nome do produto, não tendo em conta a precisão das restantes informações relativas ao produto. Estes valores altos relativos ao número de produtos identificados face às restantes versões é uma consequência do grau de precisão com que o texto foi extraído anteriormente, observado na secção anterior. Constata-se também que nestas faturas obtidas digitalmente diretamente do supermercado foi sempre identificado corretamente o nome do supermercado, a data da compra e o valor total. O algoritmo só foi capaz de identificar dois de catorze produtos na fatura Continente 2 em imagens segmentadas devido ao problema ocorrido na união de imagens, em que se perdeu uma boa parte da fatura, aliado ao ruído que prejudica também a categorização.

Perante o gráfico da figura 4.10 que apresenta as médias de similaridades de Levenshtein normalizada por cada categoria de informação dos produtos para uma das faturas analisadas sem pré-processamento aplicado, verifica-se que na versão PDF se conseguiu identificar de forma mais exata a informação devida. O preço e a capacidade foram extraídos e categorizados com o máximo de sucesso. Semelhantemente aos resultados da extração de texto, a versão de fatura capturada numa imagem única é considerada a segunda melhor opção para obtenção de melhores resultados, sendo porém surpreendida pelo grau de precisão na categoria nome pela versão da fatura com imagens segmentadas. Parte-se do princípio de que este último caso tenha sido a exceção à regra.

O gráfico da figura 4.11 seguinte permite comparar as médias de similaridade na extração dos produtos para uma mesma fatura nas diferentes versões e para diferentes pré-processamentos. O valor máximo do gráfico no eixo de Y é 400% pelo facto de se tratarem de quatro categorias diferentes por cada barra, sendo que a média de similaridade de cada uma seria 100%. Assim observa-se mais facilmente qual das versões, na totalidade, obteve melhores resultados na categorização. Temos como casos extremos a versão PDF que obteve os melhores valores, próximo do valor máximo em todas as quatro categorias. Na versão de imagens segmentadas com pré-processamento com efeito *threshold* não se vê barra alguma porque nenhum produto foi identificado com sucesso. Tendo como base os resultados anteriores da extração de texto já se esperava que, a seguir à versão PDF, a versão de imagem única fosse a melhor, mas nesta análise há uma surpresa nos níveis de pré-processamento. Ao contrário do esperado, com pré-processamento e com o efeito *threshold* extraiu-se com maior exatidão as informações dos produtos face a sem pré-processamento e até mesmo com pré-processamento sem efeito *threshold*.

Uma análise geral ao nível de precisão na categorização de informação relativa aos produtos de todas as faturas testadas, excluindo as suas versões em imagens segmentadas para diminuição da dimensão do gráfico, pode ser visualizada na figura 4.12. As versões PDF dominam com consistência também na categorização e que a fatura da SportZone 1 obteve resultados muito baixos, sendo nulos em algumas das categorias. Os resultados desta última fatura devem-se ao facto desta ser muito pequena, com apenas um produto e num mau estado de impressão, ou seja, no caso de haver algumas falhas de reconheci-

Tabela 4.1: Informações das Faturas sem Pré-Processamento Categorizadas pelo Algoritmo

Nome/ Tipo	**Identificou Supermercado?**	**Identificou Data?**	**Identificou valor Total correto?**	**Quantos items identificou? (%)**
Continente 1 - PDF	✓	✓	✓	100% (8/8)
Continente 1 - Imagem Única	✓		✓	50% (4/8)
Continente 1 - Imagens Segmentadas	✓	✓		62,5% (5/8)
Continente 2 - PDF	✓	✓	✓	85,7% (12/14)
Continente 2 -Imagem Única	✓			42,9% (6/14)
Continente 2 - Imagens Segmentadas	✓	✓		14,3% (2/14)
Continente 3 - PDF	✓	✓	✓	100% (13/13)
Continente 3 - Imagem Única	✓			53,8% (7/13)
Continente 4 - PDF	✓	✓	✓	94,1% (16/17)
Continente 4 - Imagem Única	✓			35,3% (6/17)
Lidl 1 - JPEG Original	✓	✓	✓	100% (6/6)
Lidl 1 - Imagem Única	✓			66,7% (4/6)
SportZone 1 - Imagem Única				100% (1/1)

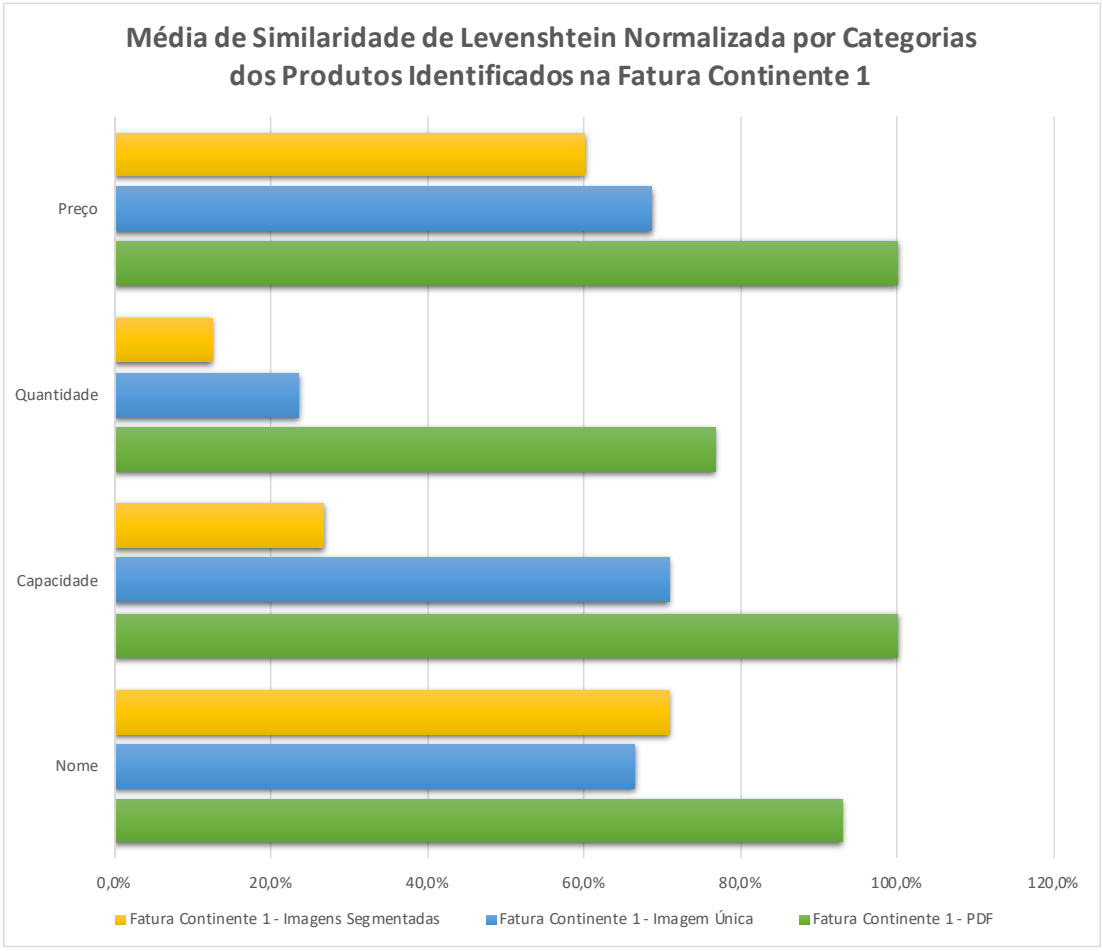

Figura 4.10: Média de Similaridade de Levenshtein Normalizada por Categorias dos Produtos Identificados em Diferentes Versões de uma Fatura

mento de carater e/ou categorização do texto contido, em percentagem, tem um grande impacto. Nesta forma de avaliação apenas foram considerados os níveis de similaridade dos produtos que foram identificados corretamente segundo o seu nome, o que justifica que nas faturas em versões com pré-processamento com *threshold* em que a percentagem total de produtos identificados tenham também as médias de similaridade mais altas. Isto porque nos poucos que identificaram corretamente, ocorreu de ser suficiente para bater as médias das outras versões que envolveram similaridade a um maior número de produtos categorizados. Confirma-se perante as barras inexistentes da fatura Lidl 1 e SportZone 1 com pré-processamento com *thresold* pois nenhum produto foi sequer reconhecido, obtendo 0% de média. Tendo esta última afirmação em conta, concluiu-se que como visualizado na análise de extração de texto, as imagens únicas com o pré-processamento sem efeito *threshold* é realmente a melhor combinação para obter os melhores resultados, a seguir das versões obtidas diretamente das aplicações móveis dos respetivos supermercados.

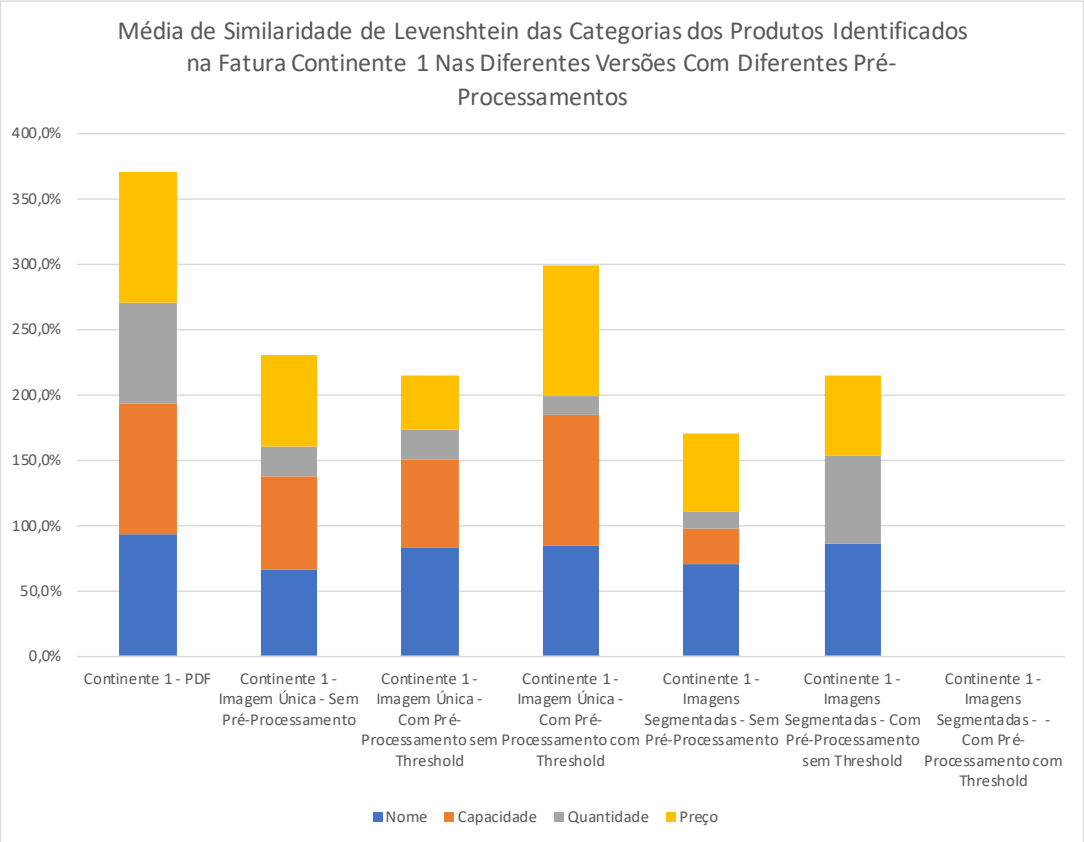

Figura 4.11: Média de Similaridade de Levenshtein por Categorias dos Produtos Identificados numa Fatura nas suas Diferentes Versões com Diferentes Níveis de Pré- Processamento

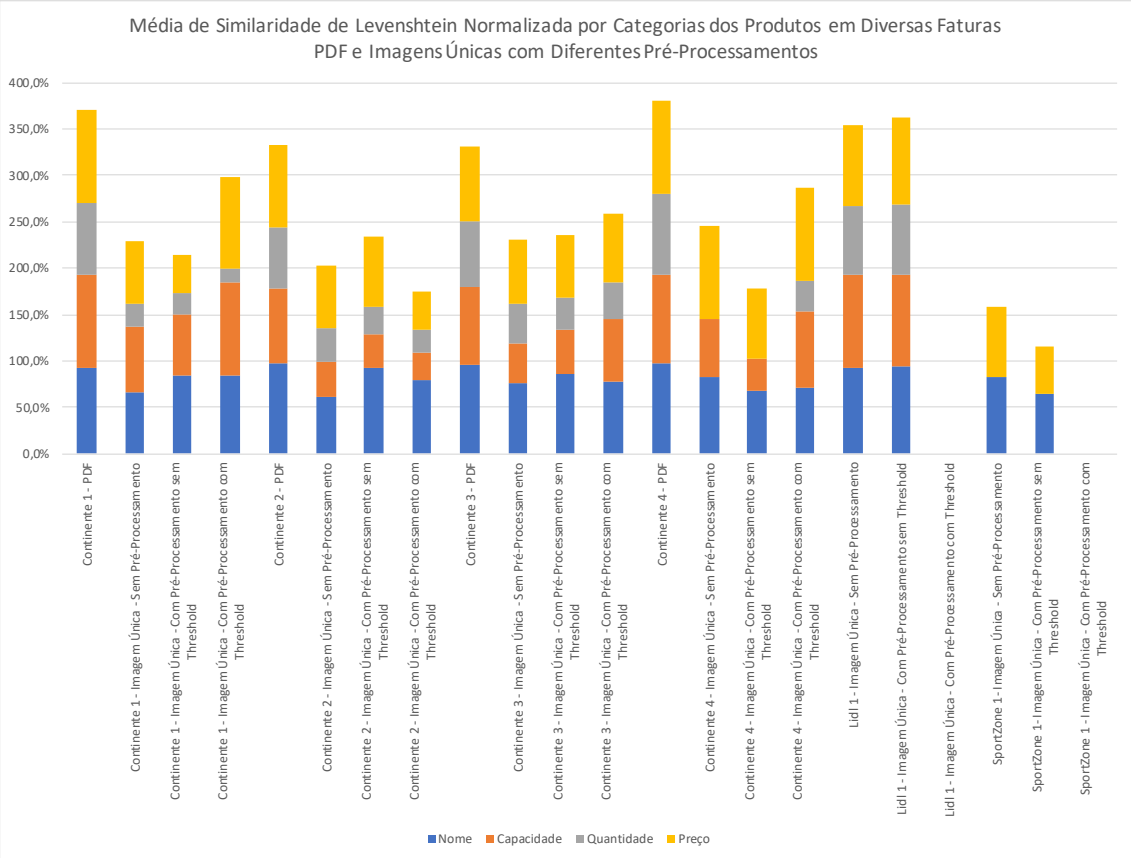

Figura 4.12: Média de Similaridade de Levenshtein Normalizada por Categorias dos Produtos em Diversas Faturas PDF e Imagens Únicas com Diferentes Pré-Processamentos

Capítulo 5

Conclusão

Esta dissertação propõe um sistema para facilitar o planeamento de uma ida às compras, sugerindo uma lista de compras elaborada automaticamente, com base no histórico de consumo do utilizador. Para que tal seja possível o utilizador terá apenas que usar a aplicação móvel para digitalizar a fatura das compras realizadas. As fotos das faturas são carregadas no *backend* do sistema que consegue extrair e classificar informação sobre os produtos e respetivas quantidades consumidas ao longo do tempo. Com base nessa informação recolhida automaticamente, o sistema gera uma lista de compras sugestiva sempre que o utilizador necessitar. O protótipo do sistema designa-se por *Shopping List Automator* e foi desenvolvido com recurso a uma biblioteca de reconhecimento de caracteres, a partir de imagens, e um algoritmo de classificação do texto extraído por forma a tipificar as compras efetuadas.

Os diversos objetivos definidos no início deste trabalho foram atingidos na totalidade, nomeadamente a criação de um sistema que recebe faturas em forma de imagem ou ficheiro, consegue automaticamente extrair devidamente a informação relevante, armazenando-a numa base de dados associada aos respetivos utilizadores. Usando essa informação consegue também analisar o padrão de consumo e gerar uma lista de compras sempre que o utilizador necessitar. Toda a estrutura do sistema e a forma como os seus componentes interagem para implementar o *pipeline*, desde a captura das faturas até à geração da lista de compras, foi descrito no Capítulo 3. A avaliação do sistema foi realizada através de testes com faturas modelo utilizadas para aferir a eficiência do sistema de extração e classificação da informação. Avaliaram-se e compararam-se ainda diferentes combinações de pré-processamento das fotos das faturas para averiguar quais obteriam melhores resultados.

Neste trabalho foram desenvolvidas todas as valências do sistema, desde a aplicação móvel até ao *backend* de suporte à aplicação. Parte significativa do processamento foi transferido para o **backend**, permitindo dessa forma que a aplicação móvel fosse mais leve. Embora o protótipo do sistema desenvolvido esteja totalmente funcional, existe

sempre espaço para melhorar e expandir as funcionalidades do mesmo. Por exemplo, seria interessante realizar estudos de usabilidade, com o intuito de aferir e melhorar a qualidade da interface gráfica da aplicação móvel. Relativamente ao tratamento das imagens das faturas, existe sempre a possibilidade de explorar e combinar mais formas de pré-processamento na tentativa de obter ainda melhores resultados na extração de texto. Adicionalmente, a categorização do texto extraído pelo OCR também pode ser sujeito a trabalho futuro na tentativa de melhorar a informação recolhida, por exemplo através de algoritmos de *Machine Learning* (ML). Da mesma forma, a análise de padrões de consumo do utilizador, também poderia explorar a utilização de algoritmos de ML, de modo a poder analisar adicionalmente variações do consumo do utilizador por categorias de produtos perante e em função da época do ano, ou até apresentar recomendações de outros produtos que nunca tenha adquirido com base no perfil de outros utilizadores com padrões de consumo similares. Por fim, pensamos também futuramente evoluir o sistema para suportar a criação de grupos de utilizadores, a pensar nos agregados familiares. Desta forma, diversos utilizadores poderiam inserir múltiplas faturas com os seus *smartphones* e desta forma contribuir para o perfil de consumo do mesmo agregado familiar. O sistema atual foi desenhado no sentido de ser uma base aberta a diversas melhorias futuras tornando-o mais robusto, prático e útil.

Referências

Abb (2021). Abbyy flexicapture. Visitado a: 26/08/2021.
URL: *https://www.abbyy.com/pt/flexicapture/* 20

Bird, J., Fozzati, D., Harrison, D. and Marshall, P. (2013). Healthy shopping: A longitudinal study of a mobile app to encourage a balanced diet, *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, Association for Computing Machinery, New York, NY, USA, p. 1047–1054.
URL: *https://doi.org/10.1145/2494091.2496012* 6

Bradski, G. (2000). The OpenCV Library, *Dr. Dobb's Journal of Software Tools* . 8

Canny, J. (1986). A computational approach to edge detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-8**(6): 679–698. 40

Cheng, T. (2021). Introduction to google cloud vision - ocr, apis, examples and pricing,.
URL: *https://nanonets.com/blog/google-cloud-vision/* 16

Cobalchini, M. E. (2018). *Mobile application for a market store list*, Master's thesis, Universidade Tecnológica Federal do Paraná.
URL: *http://repositorio.roca.utfpr.edu.br/jspui/handle/1/10941* 12

Cumby, C., Fano, A., Ghani, R. and Krema, M. (2005). Building intelligent shopping assistants using individual consumer models, *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, Association for Computing Machinery, New York, NY, USA, p. 323–325.
URL: *https://doi.org/10.1145/1040830.1040915* 21

de Angelo, C. F., João Paulo de Lara, S. and Fávero, L. P. L. (2003). As compras não planejadas em supermercados: a importância do tempo e da organização da loja na determinação dos gastos, *Revista de Administração Contemporânea* **7**(3): 149–162. 7

Devopedia (2019). Levenshtein distance. Visitado a: 24/07/2021.
URL: *https://devopedia.org/levenshtein-distance* x, 9

dos Santos, F. N. (2009). Hábitos de compras e uso de lista de compras, *Revista Portuguesa e Brasileira de Gestão* pp. 63–75. 7

Flusser, J., Farokhi, S., Höschl, C., Suk, T., Zitová, B. and Pedone, M. (2016). Recognition of images degraded by gaussian blur, *IEEE Transactions on Image Processing* **25**: 790–806. 39

Harsha Jayawilal, W. A. and Premeratne, S. (2017). The smart shopping list: An effective mobile solution for grocery list-creation process, *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*, pp. 124–129. x, 11, 13

Hosseini, H., Xiao, B. and Poovendran, R. (2017). Google's cloud vision api is not robust to noise. 16

Huang, Z., Chen, K., He, J., Bai, X., Karatzas, D., Lu, S. and Jawahar, C. V. (2019). Icdar2019 competition on scanned receipt ocr and information extraction, *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1516–1520. 13

Hyp (2021). Document processing with deep learning. Visitado a: 26/08/2021.
URL: *https://hypatos.ai/en* 19

Kay, A. (2007). Tesseract: An open-source optical character recognition engine, *Linux J.* **2007**(159): 2. 8

Khan, T. (2019). A Low Power IoT-Connected Smart Canister System Creating Automatic Shopping List, *Journal of Sensor and Actuator Networks* **8**(3).
URL: *https://www.mdpi.com/2224-2708/8/3/38* x, 11, 12

Kle (2021). Template-less data extraction for unstructured documents. Visitado a: 26/08/2021.
URL: *https://www.klearstack.com* 20

Kli (2021). Soluções de processamento automatizado de documentos. Visitado a: 26/08/2021.
URL: *https://www.klippa.com/pt/home-pt/* 18

Kumar, V., Kaware, P., Singh, P., Sonkusare, R. and Kumar, S. (2020). Extraction of information from bill receipts using optical character recognition, *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, pp. 72–77. x, 14, 15, 16

Kuratli, J. (2015). *Shopping 2.0*, Bachelor's thesis, ETH Zurich. xii, 20, 21

Merejkowsky, D. (2006). pyenchant. Visitado a: 25/07/2021.
URL: *https://pypi.org/project/pyenchant/* 8

Miola, A. (2020). *Flutter Complete Reference: Create Beautiful, Fast and Native Apps for Any Device*, Independently Published.
URL: *https://books.google.pt/books?id=y372zQEACAAJ* 10

Nan (2021). Soluções de processamento automatizado de documentos. Visitado a: 26/08/2021.
URL: *https://nanonets.com/invoice-ocr* 19

Pashigian, B., Peltzman, S. and Sun, J.-M. (2003). Firm Responses to Income Inequality and the Cost of Time, *Review of Industrial Organization* **22**(4): 253–273.
URL: *https://ideas.repec.org/a/kap/revind/v22y2003i4p253-273.html* 6

Pires, A. d. M. (2021). *Estratégias de posicionamento usadas pelo continente: a perceção do consumidor*, Master's thesis, Universidade de Évora.
URL: *http://hdl.handle.net/10174/30063* 7

Robert, V. and Talbot, H. (2020). Does super-resolution improve ocr performance in the real world? a case study on images of receipts, *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 548–552. 14

Ros (2021). Automated document communication. Visitado a: 26/08/2021.
URL: *https://rossum.ai* 19

Sainz-De-Abajo, B., García-Alonso, J. M., Berrocal-Olmeda, J. J., Laso-Mangas, S. and De La Torre-Díez, I. (2020). Foodscan: Food monitoring app by scanning the groceries receipts, *IEEE Access* **8**: 227915–227924. x, 16, 17

Santos, S. R. F. F. d. (2008). *Melhoria de serviços de base tecnológica Aplicação ao caso do retalho*, Master's thesis, Faculdade de Engenharia da Universidade do Porto.
URL: *http://hdl.handle.net/10216/61608* 7

Sidhwa, H., Kulshrestha, S., Malhotra, S. and Virmani, S. (2018). Text extraction from bills and invoices, *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pp. 564–568. 14

Silva, E. G. (2020). *Lista de compras inteligente*, Master's thesis, Instituto Superior de Engenharia do Porto.
URL: *http://hdl.handle.net/10400.22/16774* 11

Tang, C. S., Bell, D. R. and Ho, T.-H. (2001). Store Choice and Shopping Behavior: How Price Format Works, *California Management Review* **43**(2): 56–74.
URL: *https://doi.org/10.2307/41166075* 7

Tukkinen, P. and Lindqvist, J. (2015). Understanding motivations for using grocery shopping applications, *IEEE Pervasive Computing* **14**(4): 38–44. 11

Tur (2021). Intelligent document processing. Visitado a: 26/08/2021.
URL: *https://www.turicode.com* 20

Typ (2021). Invoice ocr for developers. Visitado a: 26/08/2021.
URL: *https://typless.com* 20

Watkins, T. (1984). Consumer Purchasing of Low-involvement Goods: Routine or Impulse?, *Marketing Intelligence & Planning* **2**(2): 51–66. 6

Wei, T. C., Sheikh, U. U. and Rahman, A. A.-H. A. (2018). Improved optical character recognition with deep neural network, *2018 IEEE 14th International Colloquium on Signal Processing Its Applications (CSPA)*, pp. 245–249. 17

Winkler, W. (1990). String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage, *Proceedings of the Section on Survey Research Methods* . 10

Yang, H. C. (2013). Bon Appétit for Apps: Young American Consumers' Acceptance of Mobile Applications, *Journal of Computer Information Systems* **53**(3): 85–96.
URL: *https://doi.org/10.1080/08874417.2013.11645635* 7

Zhao, X., Niu, E., Wu, Z. and Wang, X. (2019). Cutie: Learning to understand documents with convolutional universal text information extractor. x, 17, 18, 19