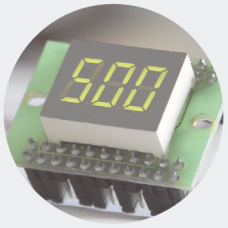
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Lista de compras inteligente

EVA GRAÇA SILVA
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Lista de compras inteligente

Mestrado em Engenharia Eletrotécnica e de Computadores

Eva Silva

Orientação
Lino Manuel Baptista Figueiredo

Ano Letivo: 2019-2020

Instituto Superior de Engenharia do Porto
Departamento de Engenharia Eletrotécnica
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Resumo

Devido à crescente exigência dos consumidores, as tecnologias envolventes neste âmbito têm de se encontrar à altura de atender os desejos destes, facilitando-lhes o seu dia-a-dia. Neste sentido, as diversas tecnologias atuais no mercado permitem construir e desenvolver algo que responda aos mais diversos problemas ou que, simplesmente, favoreça a experiência no ato de compra, existindo atualmente diversas aplicações móveis que assistem e apoiam o consumidor nesta atividade.

O presente Projeto consiste na implementação de uma aplicação que gere os produtos que se encontram em falta na despensa do consumidor, juntamente com um sistema que deteta esses itens que foram consumidos/gastos. Ao dispor de uma lista de compras no seu *smartphone*, o consumidor tende a ser mais controlado, organizado e a aplicar menos tempo nesta tarefa rotineira.

O sistema de captura de produtos encontra-se no caixote do lixo de forma estratégica de modo a ler os códigos de barras das embalagens dos produtos vazios a colocar no lixo. Este utiliza uma câmara de vídeo para detetar os códigos de barras em conjunto com a análise da imagem capturada.

A linguagem de desenvolvimento Python, a biblioteca OpenCV, assim como a *framework NestJS* e o *software Android Studio* foram algumas das ferramentas aplicadas neste Projeto, tornando possível auxiliar o consumidor de forma simples e acessível.

Foram seguidas várias etapas, desde a implementação de um detetor de códigos de barras através de uma câmara até ao desenvolvimento de uma aplicação móvel, que emergiram no sentido de enriquecer a experiência do consumidor no planeamento e ato de compra. Dito isto, é de salientar que os resultados obtidos foram satisfatórios, originando no sucesso de um sistema capaz de analisar que produtos se encontram em falta assim como gerir a elaboração de uma lista de compras.

Palavras-Chave

Lista de compras, Visão computacional, Raspberry Pi, Aplicação *Android*, Análise de Imagem em Tempo Real

Abstract

Due to the increase of demand by the consumers, technologies involved in this area must be able to meet their clients' needs, in order to make their daily lives easier. In this context, the several technologies existing in today's market, make it possible to construct and develop the solution to numerous problems or just to favour the experience when purchasing, currently many mobile applications assist and support the consumer during these activities.

This Project consists in developing an app which manages product needs in the consumer's pantry as well as a system that detects consumed items. Owning a list of the consumer's needs in their smartphone, allows them to manage, have control and spend less time with this task.

The scanning product system lies in the consumer's garbage can, as a strategic point of view, allowing it to regard the empty product's barcode when put away. A video camera is used to capture the barcode and examine the captured image.

The lingo Python, the library OpenCV, the framework NestJS and the software Android Studio were some of the application tools used in this project, making it possible to assist the consumer in an easy and simple way.

Through the making of this project there were several stages, from the implementation of a barcode camera detector, to the mobile App, emerging so the consumer's experience enhances when planning and purchasing. It must be underlined that the obtained results were satisfactory, originating a successful system able to acknowledge what products are missing and produce a list with said items.

Keywords

Shopping list, Computer vision, Raspberry Pi, Android app, Real-Time Image Analysis

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

Abreviatura	Descrição
API	*Application Programming Interface*
CCD	*Charge-Couple Device*
CODIPOR	Associação Portuguesa de Identificação e Codificação de Produtos
DTO	*Data Transfer Object*
EAN	*European Article Numbering*
EAN.UCC	*European Article Numbering/ Uniform Code Council*
FP	*Functional Programming*
FRP	*Functional Reactive Programming*
GEPIR	*Global Eletronic Party Information Registry*
GPIO	*General Purpose Input/Output*
GTIN	*Global Trade Item Number*
HDMI	*High-Definition Multimedia Interface*
HTTP	*Hypertext Transfer Protocol*
IA	Inteligência Artificial
IBM	*International Business Machines*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
MMC	*International Business Machines*
OOP	*Object Oriented Programming*
OpenCV	*Open Source Computer Vision Library*
ORM	*Object-Relational Mapping*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
RPC	*Remote Procedure Calls*
SD	*Secure Digital*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
UPC	*Universal Product Code*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*

Abreviatura	Descrição
UVC	USB *Video Class*
XML	*Extensible Markup Language*

Agradecimentos

Eu gostava de começar por agradecer à minha família pelo apoio e confiança.

Ao meu irmão Mateus, pela dedicação incondicional e apoio desde sempre.

Queria também agradecer ao Engenheiro Lino Figueiredo pela orientação, dedicação e apoio moral para a conclusão deste projeto.

Por último, quero agradecer aos meus amigos e colegas do ISEP por me ajudarem ao longo do meu percurso académico e me apoiarem a conquistar esta etapa.

Capítulo 1

Introdução

No presente capítulo é feita uma introdução a este trabalho de dissertação desenvolvido no âmbito da Unidade Curricular Tese Dissertação, inserida no plano de estudos do 2º ano do Mestrado em Automação e Sistemas do Instituto Superior de Engenharia do Porto. É apresentada uma pequena contextualização do trabalho a desenvolver, os seus principais objetivos a atingir e calendarização. Por fim, é exposta a organização do relatório indicando os temas/tópicos abordados em cada capítulo.

1.1 Contextualização

Com a constante evolução tecnológica, os consumidores estão cada vez mais informados e, consequentemente, mais exigentes, conscientes e sensíveis em determinados aspetos. Quando lhe é apresentado uma tecnologia que acelere e/ou facilite um processo, ou até que evite alguns procedimentos, o consumidor não hesitará em usufruir desta solução.

Quando se fala em consumidor deve-se sempre ter em conta o processo de tomada de decisão deste, que difere de acordo com o tipo de decisão estabelecida e o seu envolvimento, bem como o papel que desempenha.

A tomada de decisão é condicionada pelos objetivos que persegue no momento de compra: seja entretenimento, satisfação de necessidades ou aquisição de bens fundamentais para a sua vivência diária na qual incluem as compras de bens alimentares e produtos de mercearia. Este último caso trata-se de uma atividade muito rotineira e, predominantemente, guiada por comportamentos habituais e de baixo envolvimento. Esta realidade conhecida pela (quase) totalidade dos indivíduos pode ser muitas vezes cansativa e stressante, levando frequentemente a comprar mais produtos do que o necessário. É aqui que en-

tra a aplicação da tecnologia para melhorar a experiência do consumidor nesta tarefa tão comum no seu dia-a-dia.

Normalmente, o consumidor leva alguns pontos em consideração antes de realizar esta atividade. A escolha do local ou estabelecimento é um desses pontos e é influenciada pela localização, preços e diversidade de produtos. Esta decisão deve ser feita em função das suas expetativas e necessidades. A seleção do período para efetuar as compras, ou seja, o momento definido pela disponibilidade do consumidor juntamente com o período que considera mais oportuno para visitar a loja, é um dos fatores importantes para que este não encontre maior aglomeração de pessoas, o que aumenta o tempo despendido dentro da loja. Nestas condições, o uso de uma lista de compras constitui um grande benefício, reduzindo o tempo atribuído a esta tarefa, tal como muitas outras vantagens.

A lista de compras, tal como nome indica, é o conjunto de produtos que na perspetiva do consumidor devem ser comprados e deve ser planeada previamente de acordo com as necessidades. Este guia contribui para uma maior organização e controlo no momento de compra, sendo vários os motivos para a sua utilização: relembrar o consumidor todos os produtos a adquirir, não correndo o risco de se esquecer de nenhuma intenção de compra; controlar o orçamento gasto na compra tal como evitar compras impulsivas ou não planeadas; reduzir o tempo empregue nesta atividade; entre outros.

Nos dias que correm existem inúmeras aplicações que facilitam a vida aos consumidores no que diz respeito ao processo de compra, mais especificamente, na criação de listas de compras. O presente Projeto visa incentivar o uso de listas de compras através um sistema inteligente bastante acessível para os utilizadores, de modo a proporcionar uma experiência mais agradável nesta atividade tão rotineira [20].

1.2 Objetivos

O sistema a desenvolver foca-se na criação de uma lista de compras simples e prática para que no momento de compra o utilizador compre especialmente os produtos em falta na sua despensa. A lista de compras será acedida via uma aplicação para *smartphone*, onde o utilizador poderá visualizar os produtos que se encontram na lista, bem como editar informações relativas aos produtos, adicionar ou até mesmo eliminar itens da lista.

Ao invés do utilizador conferir os produtos que faltam e adicioná-los manualmente, o sistema inclui um leitor de código de barras inserido num caixote do lixo, que tem como função ler os códigos dos produtos vazios antes de serem

colocados no lixo que, por sua vez, serão inseridos automaticamente na lista de compras do utilizador.

Produtos como fruta e legumes deverão ser introduzidos manualmente pelo utilizador visto que não possuem código de barras. Desta forma, e indo ao encontro da descrição anteriormente definida são estabelecidos os principais objetivos:

- Estudar as soluções existentes no mercado;
- Definir os requisitos necessários para a implementação da solução proposta;
- Identificar/selecionar o *hardware* necessário ao desenvolvimento do projeto;
- Implementar um sistema para captura e leitura de códigos de barras mediante uma câmara;
- Desenvolver uma aplicação para *smartphone* para que o utilizador tenha acesso à lista de compras, tal como a diversas funcionalidades;
- Interligar a aplicação com o sistema de captura de produtos;
- Testar a solução proposta e validar em cenário experimental.

1.3 Calendarização

Para executar este projeto foram estabelecidas metas que visam a organização pessoal e o percurso a tomar. Na Tabela 1.1 encontra-se representado a distribuição das tarefas ao longo do tempo.

Tabela 1.1: Calendarização

Etapas	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro
Definição dos requisitos e objetivos							
Estudo de conceitos, Visão computacional e alguns métodos							
Estudo das soluções existentes no mercado							
Identificar/selecionar o *hardware* necessário							
Desenvolvimento do *software* do sistema de captura de produtos							
Desenvolvimento do servidor e base de dados							
Desenvolvimento da aplicação móvel							
Testes práticos e validação da solução							
Elaboração do relatório							

1.4 Organização

Este relatório está estruturado em seis capítulos. No presente capítulo é realizada uma contextualização do tema principal do Projeto, assim como a apresentação dos objetivos a atingir. Neste capítulo é apresentado ainda a calendarização e organização do relatório.

No capítulo subsequente, intitulado Estado de Arte, é realizada uma introdução ao conceito código de barras, abordando um pouco da história e evolução que este sofreu ao longo dos anos, tal como o seu funcionamento. Neste capítulo foi referido ainda algumas técnicas e tecnologias úteis para o desenvolvimento do presente Projeto, assim como o estudo das soluções existentes no mercado, indicando as vantagens destas.

Designado como Arquitetura, o terceiro capítulo aborda a arquitetura do sistema, incluindo uma descrição detalhada do *hardware* e *software* utilizado no Projeto.

O quarto capítulo, Implementação, explana a implementação e execução do projeto, indicando em maior pormenor como o sistema funciona, assim como que componentes foram utilizados para o desenvolvimento da aplicação.

O quinto e penúltimo capítulo, Resultados, é utilizado para expor os resultados obtidos.

O último capítulo, Conclusão, apresenta uma breve reflexão sobre a experiência desenvolvida na realização deste Projeto, sendo apresentadas recomendações e sugestões que potenciem a melhoria de futuros projetos. Por fim, ao encerrar o documento são expostas as referências bibliográficas que apoiaram a realização deste e os anexos que lhe dão suporte.

Capítulo 2

Estado da arte

No contexto da problemática em estudo e de forma a apoiar a análise que vem responder e confrontar certas questões, é essencial aprofundar e formular certos conceitos antes de partir para a implementação de qualquer projeto. Neste capítulo serão abordados vários conceitos relativos ao código de barras, sintetizando o seu funcionamento e apresentando os diversos leitores existentes atualmente no mercado. Serão ainda introduzidas tecnologias que serão utilizadas para o desenvolvimento do Projeto, tanto a nível de *software* como de *hardware*.

2.1 Código de barras

Atualmente, o ser humano encontra-se rodeado de códigos que, apesar de trazerem grandes benefícios, tornam o seu dia-a-dia cada vez mais dependente destes. Existe uma grande diversidade de códigos para identificar todo o tipo de produto e até mesmo pessoas diferenciando-as através, p.e., do número de identificação civil, número de identificação fiscal, entre outros. Dentro de todos estes códigos destaca-se o código de barras encontrando-se nas mais diversas áreas como comércio, indústria, bancos, hospitais, transportes, controlos de acesso e muitos outros setores.

O código de barras não é mais do que uma representação gráfica de dados. Ele permite uma rápida captação destes por meio de leitura ótica, proporcionando uma maior velocidade nas transações, precisão nas informações, um aumento da produtividade e redução de possíveis erros, garantindo rapidez no atendimento a pedidos. São maioritariamente usados nos produtos de forma a serem lidos à saída do supermercado [2].

Os dados presentes nos códigos de barras são representados por várias barras brancas e pretas alternadas de espessura variável, tal como ilustra a Figura 2.1.

Figura 2.1: Exemplo de um código de barras [1]

2.1.1 História

Os primeiros estudos efetuados que originaram os códigos de barras utilizados atualmente foram realizados em meados do século XX, por *Joseph Woodland* e *Bernard Silver*, estudantes no Instituto de Tecnologia de Drexel. O norte-americano *Bernard Silver* ouviu no corredor do instituto o presidente de uma cadeia de supermercados, *Food Fair*, a pedir que criassem um sistema automático para realizar a leitura de produtos no momento de *checkout*. Os dois estudantes norte-americanos entusiasmados com o projeto juntaram-se e puseram mãos à obra, dando os primeiros passos em direção do que hoje conhecemos por código de barras. Foi então que, em 1952, os dois norte-americanos registaram uma patente que apresentava uma matriz de identificação baseada em círculos concêntricos de espessura variável, também conhecido por "olho-de-boi" devido ao seu formato [2, 21] .

Nos anos 70, a firma *McKinsey & Co.* juntamente com a *Uniform Grocery Product Code Council*, pediu a várias companhias que criassem um código acessível e adequado de modo a reunir com o formato numérico que estabeleceram para a identificação de produtos. A solução mais viável foi apresentada por *George J. Laurer*, engenheiro do *International Business Machines* (IBM). O código era composto por uma sequência de 12 dígitos que eram traduzidos em barras brancas e pretas, tal como os códigos de barras implementados nos dias de hoje. Este código apresentado por *Laurer* foi formalmente aceite em 1973 denominando-o de *Universal Product Code* (UPC), atualmente aplicado nos Estados Unidos e no

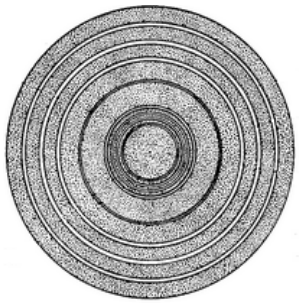


Figura 2.2: Modelo do primeiro código de barras [2]

Canadá. No ano 1974 fez-se a primeira compra de um produto marcado com um código de barras .

Figura 2.3: Exemplo do código de barras UPC [3]

Devido ao grande êxito do código criado por *Laurer* foi gerado um novo código de 13 dígitos. Este código foi adotado em 1976 e foi intitulado de *European Article Numbering* (EAN), sendo hoje aplicado em todo o mundo incluindo Portugal, exceto os Estados Unidos e Canadá que, como já foi referido, adotaram o código UPC. Foi então que nasceu a GS1 pondo o mundo dos negócios a falar a mesma linguagem, com soluções que melhoram a eficiência e visibilidade das cadeias de valor em todos os setores, unindo produtores, retalhistas, transportadores, *software developers*, alfândegas, hospitais, entidades reguladores e também consumidores [2].

Atualmente, a GS1 internacional é uma organização neutra, multissetorial, orientada para e pelos utilizadores e sem fins lucrativos, que se dedica ao desenho e implementação de normas segundo uma base comum (código de barras) para a identificação única, captura automática e partilha eletrónica de informação sobre produtos, localização e bens. Esta corporação conta agora com mais de um milhão de membros que vão desde grandes multinacionais até ao retalhista local [22].

A GS1 oferece um leque de serviços e ferramentas para tornar a adoção dos seus padrões mais fácil e acessível, de forma a contribuir para uma experiência positiva para os seus usuários. O GEPIR (*Global Eletronic Party Information Registry*) é um dos serviços disponibilizados pela GS1 que dá acesso gratuito a informações básicas de contato de empresas associadas à GS1. Através da simples inserção de um número de código de barras (GTIN – *Global Trade Item Number*) no GEPIR, qualquer utilizador pode ficar a saber quem é o proprietário do código (empresa), tal como o seu contato e a que organização membro GS1 que está associado. É possível ainda pesquisar no motor de pesquisa cedido por este serviço números de localização física, números de expedição e ainda a designação social da empresa. Toda esta informação pode ser acedida através de um *website* (https://gepir.gs1.org/), todavia dados como a descrição dos produtos, como peso, medidas, volume, entre outros, são reservados à empresa que os produzem [23].

2.1.2 Código de barras em Portugal

Em Portugal, a CODIPOR (Associação Portuguesa de Identificação e Codificação de Produtos), conhecida hoje por GS1 Portugal, foi constituída a 26 de novembro de 1985, de forma a introduzir no país o Sistema de Codificação EAN.UCC (*European Article Numbering/ Uniform Code Council*), atualmente GS1.

A GS1, organização internacional responsável pelo desenvolvimento e implementação de *standards* para produtos, licenciou, por contrato, a CODIPOR como sua organização-membro e representante exclusiva, para gerir, a nível nacional, o atual Sistema global GS1. Este sistema é um conjunto de normas integradas abertas e globais, reconhecidas internacionalmente, para gerir de forma eficiente as cadeias de valor multissectoriais, baseada numa identificação única e inequívoca de produtos, unidades de expedição, ativos, localizações e serviços, que agiliza todos os processos comerciais, incluindo o comércio eletrónico e a rastreabilidade, surgindo assim a implementação do código de barras em Portugal.

A CODIPOR tem como propósito a gestão, a nível nacional, do sistema implementado pela organização GS1 bem como o acompanhamento, investigação, estudo, formação, implementação e desenvolvimento de outros sistemas que conduzem à normalização e simplificação de procedimentos no âmbito da indústria, comércio e serviços.

A CODIPOR – GS1 Portugal é hoje uma das maiores associações empresariais em Portugal, reunindo mais de 8000 empresas que correspondem, relativamente a volume de faturação, a aproximadamente 50% do produto interno bruto nacional. Estas empresas associadas vão desde produtores de matérias-primas

a produtores de marcas, distribuidores e retalhistas, associações industriais, entre outros. Porém, em 2018, cerca de 26% das empresas associadas pertenciam à categoria de mercearia alimentar, o que reflete a maturidade alcançada pela organização no setor de retalho e bens de consumo, setor de origem da CODIPOR. Nesse mesmo ano, cerca de 38% das empresas agregadas à GS1 Portugal encontravam-se localizadas nos distritos de Lisboa e Porto, com um total de 1 966 empresas em Lisboa e 1 195 no Porto de um conjunto de 8 227 empresas afiliadas, conforme ilustra a Figura 2.4 [22, 24].

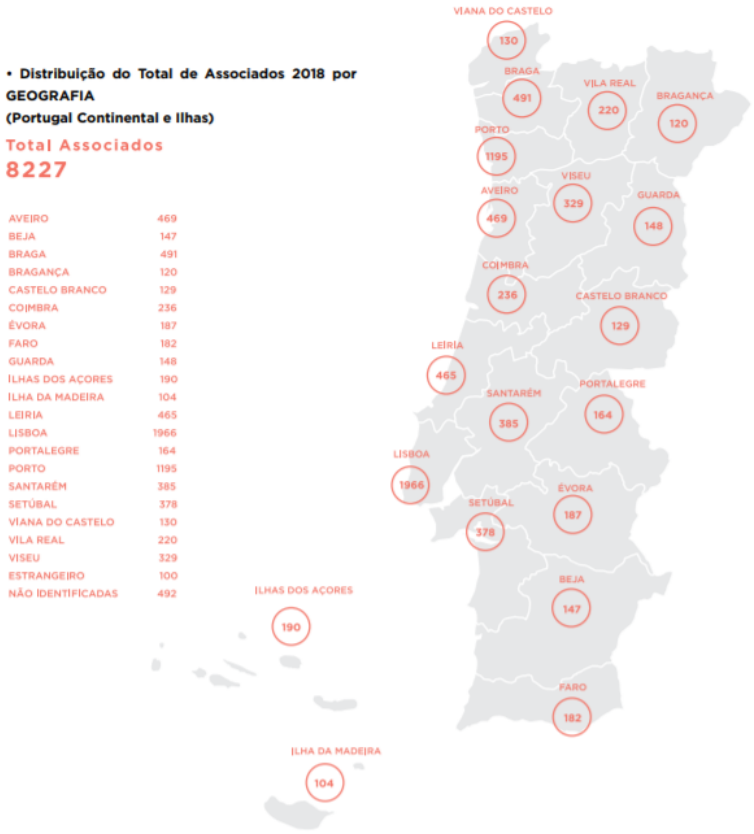


Figura 2.4: Distribuição geográfica das empresas associadas à GS1 Portugal em 2018 [4]

Segundo João Gaspar Lopes Ribeiro, membro fundador da CODIPOR, "a criação da CODIPOR, em novembro de 1985, representou uma revolução em áreas da produção e distribuição, permitindo a introdução de meios de controlo e funcionalidade que não seriam possíveis se o sistema de codificação não fosse introduzido. Com a CODIPOR, posteriormente GS1, a produção, a distribuição e o consumo passaram a auferir de informação, controlo de gestão e confiança

até àquela data inatingíveis. Hoje já não seria possível viver sem a codificação." [22].

2.1.3 Conceitos

O código de barras é a representação gráfica de uma sequência numérica ou alfanumérica para identificar um certo produto. É representado através de barras verticais, pretas e brancas, de espessuras variadas, que correspondem à sequência que se encontra por debaixo das barras, tal como se pode verificar na Figura 2.1. A sua principal função é identificar um produto, sendo que não existe a mesma sequência para dois produtos diferentes.

Após a leitura do código, os comerciantes têm acesso a uma grande quantidade de informação sobre o produto em questão, como o valor e uma pequena descrição. Os dados representados por barras verticais são analisados por um *software* apropriado que, por sua vez, irá exibir as informações relativas ao produto, identificando-o. A leitura dos dados é, geralmente, efetuada por um dispositivo denominado leitor de código de barras capaz de capturar, descodificar, e transferir os dados para o sistema computadorizado, tudo numa questão de segundos.

A rapidez e facilidade do processo de leitura e toda a informação disponibilizada através dos códigos de barras proporciona diversos benefícios para um negócio visto que, reduz a incidência de erros, pois a leitura é feita automaticamente reduzindo a ocorrência de falhas humanas; reduz do tempo gasto em inúmeras tarefas, tal como o registo de produtos; o custo de implementação e manutenção é baixo e não exige qualquer formação aos operadores. Consequentemente, estes fatos conduzem a uma redução de custos e a uma melhoria na eficiência operacional, contribuindo para o aumento da produtividade. A modernização destes processos proveniente da aplicação do código de barras também favorece a relação com os clientes e fornecedores, sendo um passo importante para uma possível fidelização.

Todos os códigos de barras têm uma zona de silêncio que são espaços sem impressão que se encontram antes e depois do código. Estas zonas são bastante importantes para que o leitor faça corretamente a captura do código. Os códigos EAN apresentam três blocos de barras ligeiramente mais compridas que as outras, denominados de delimitadores, como se pode verificar na Figura 2.5. Estes delimitadores têm a função de delimitar as áreas na qual se encontram as informações do código, dividindo o código em duas áreas, lado esquerdo e lado direito [25].

Figura 2.5: Estrutura de um código de barras EAN-13 (Adatado de [1])

2.1.4 EAN-13

Atualmente existem inúmeras simbologias de códigos de barras para as mais diversas áreas. Cada simbologia possui caraterísticas próprias como as regras aplicadas na codificação de cada caracter, requisitos de impressão, descodificação, verificação de erros, entre outros. Neste capítulo ir-se-á aprofundar o sistema de código de barras EAN de 13 dígitos, também conhecido por EAN-13, sendo este adotado em Portugal, como já foi mencionado anteriormente, contudo existem os códigos UPC, entre muitos outros.

O EAN-13 é o código de barras mais utilizado em todo o mundo e é composto por uma sequência numérica de 13 dígitos, em que os dois ou três primeiros dígitos fornecem a identificação da GS1 onde o código foi originado. Na Tabela 2.1 encontra-se uma lista dos vários prefixos GS1 de diversos países. O prefixo de identificação da GS1 Portugal é 560 [25].

Tabela 2.1: Prefixos GS1 de diversos países [17]

Prefixo	País
30-37	França
380	Bulgária
383	Eslovénia
385	Croácia
400 - 404	Alemanha
539	Irlanda
560	Portugal
569	Islândia

Os quatro ou cinco dígitos seguintes identificam a empresa que está associada à GS1, os próximos cinco representam o código do item para identificar o produto em si dentro da empresa e, por último, o décimo terceiro dígito é o dígito verificador, conforme ilustra a Figura 2.6. [25].

Figura 2.6: Composição de um código de barras EAN-13 (Adatado de [1])

Os códigos são formados por barras pretas e brancas de espessuras diferentes que podem ser classificadas como fina, média, grossa, ou muito grossa. Estas barras são interpretadas e descodificadas, através do leitor e do computador, em sequências de "zeros" (barras brancas) e "uns" (barras pretas), traduzindo assim os números que aparecem abaixo das barras em base binária [18].

Tabela 2.2: Classificação das barras [18]

Barras	Pretas	Brancas
Finas	1	0
Médias	11	00
Grossas	111	000
Muito Grossas	1111	0000

O processo de descodificação de códigos de barras respeita padrões de codificações estabelecidos e controlados internacionalmente, permitindo assim maior integração e troca de informações.

Cada dígito numérico decimal corresponde a uma sequência de zeros e uns de comprimento sete, formando quatro barras (duas pretas e duas brancas), no entanto para representar cada dígito é preciso levar em conta uma série de fatores. O primeiro é em que lado se encontra o dígito (lado esquerdo ou lado direito), pois se estiver do lado esquerdo a representação segue um critério e se estiver do lado direito a representação segue outro critério. Para representar os dígitos do lado esquerdo, i.e., do segundo ao sétimo dígito, é necessário saber o valor do primeiro dígito, pois este é o responsável por determinar que representação terá cada dígito, não sendo representado por barras, pois encontra-se implícito na codificação dos dígitos dependentes deste. Este dígito indica se o dígito é codificado com um número par ou ímpar de dígitos iguais a 1. Os dígitos que se encontram do lado direito, *i.e.*, do oitavo ao décimo terceiro dígito, obedecem todos ao mesmo padrão de codificação [18].

Na Tabela 2.3 está representada a codificação que os códigos EAN-13 seguem, sendo que o primeiro dígito é o dígito que define que codificação se deve optar para o lado esquerdo do código de barras, em que a letra "I" corresponde a um número ímpar de uns e a letra "P" a um número par de uns. Os dígitos correspondentes ao lado direito do código apresentam todos um número par de dígitos iguais a 1.

Tabela 2.3: Tabela de descodificação de código de barras [19]

Dígito	Codificação de Dígitos			Codificação em relação ao 1º dígito
	Esquerdo		Direito	
	Ímpar	Par	Par	
0	0001101	0100111	1110010	I-I-I-I-I-I P-P-P-P-P-P
1	0011001	0110011	1100110	I-I-P-I-P-P P-P-P-P-P-P
2	0010011	0011011	1101100	I-I-P-P-I-P P-P-P-P-P-P
3	0111101	0100001	1000010	I-I-P-P-P-I P-P-P-P-P-P
4	0100011	0011101	1011100	I-P-I-I-P-P P-P-P-P-P-P
5	0110001	0111001	1001110	I-P-P-I-I-P P-P-P-P-P-P
6	0101111	0000101	1010000	I-P-P-P-I-I P-P-P-P-P-P
7	0111011	0010001	1000100	I-P-I-P-I-P P-P-P-P-P-P
8	0110111	0001001	1001000	I-P-I-P-P-I P-P-P-P-P-P
9	0001011	0010111	1110100	I-P-P-I-P-I P-P-P-P-P-P

De forma a explicar melhor este procedimento, ir-se-á analisar o código de barras apresentado na Figura 2.1. Este é identificado pelo código 5-601234-502824. Segundo o primeiro dígito, que neste caso é o número 5, e a Tabela 2.3, a codificação do lado esquerdo do código será:

(6) ímpar - (0) par - (1) par - (2) ímpar - (3) ímpar - (4) par

Voltando a consultar a Tabela 2.3:

$$6 \rightarrow 0101111 \qquad 0 \rightarrow 0100111 \qquad 1 \rightarrow 0110011$$
$$2 \rightarrow 0010011 \qquad 3 \rightarrow 0111101 \qquad 4 \rightarrow 0011101$$

Para os dígitos do lado direito basta retirar a codificação diretamente da Tabela 2.3:

$$5 \rightarrow 1001110 \qquad 0 \rightarrow 1110010 \qquad 2 \rightarrow 1101100$$
$$8 \rightarrow 1001000 \qquad 2 \rightarrow 1101100 \qquad 4 \rightarrow 1011100$$

Na Figura 2.7 pode-se visualizar a sequência de zeros e uns que um leitor geraria na captação do código de barras 5-601234-502824, confirmando a correta codificação de cada dígito.

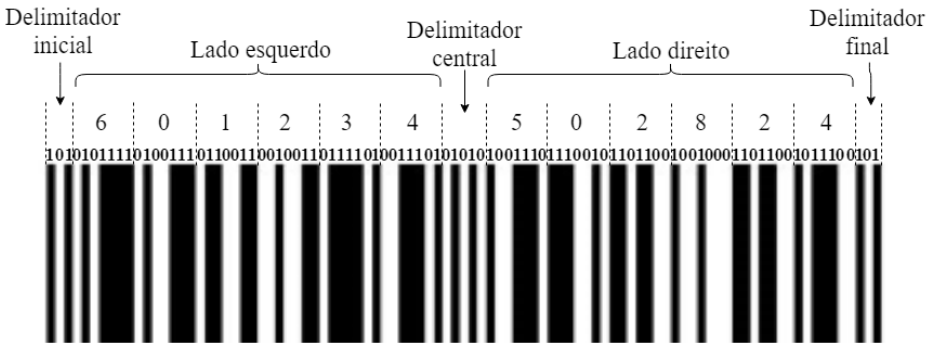


Figura 2.7: Descodificação de um código de barras (Adaptado de [1])

O dígito verificador, o último dígito (o mais à direita) do código de barras, é calculado em função dos dígitos anteriores e tem como objetivo assegurar a correta leitura do código de barras, verificando se é necessário fazer novamente a captação do código.

Considerando um produto que está identificado por um código de barras EAN-13 por uma dada sequência de dígitos: $a_1, a_2, a_3, \ldots, a_{13}$, sendo $x = a_{13}$ o dígito verificador, tem-se, então, $\alpha = (a_1, a_2, \ldots, a_{12}, x)$. O sistema EAN-13 recorre a um vetor fixo, denominado de vetor de pesos, ω, dado por:

$$\omega = (1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1)$$

Calcula-se, então, o produto escalar entre os dois vetores:

$$\alpha.\omega = (a_1, a_2, a_3, a_4, \ldots, a_{12}, x).(1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1)$$
$$\alpha.\omega = a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12} + x$$

O dígito de verificação x ($x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$) é determinado de forma a que a soma obtida seja múltiplo de 10, isto é, $\alpha.\omega \equiv 0 mod 10$ [18].

Exemplo:

Para o código 5-601234-502824, o dígito verificador é 4, podendo ser comprovado através da aplicação do algoritmo.

Sendo $\alpha = (5,6,0,1,2,3,4,5,0,2,8,2,x)$ e $\omega = (1,3,1,3,1,3,1,3,1,3,1,3,1)$, então tem-se:

$$\alpha.\omega = 5 + (3 \times 6) + 0 + (3 \times 1) + 2 + (3 \times 3) + 4 + (3 \times 5) + 0 + (3 \times 2) + 8 + (3 \times 2) + x$$

$$\alpha.\omega = 76 + x$$

Como $(76 + x) \equiv 0 mod 10$, então $x = 4$, tal como se pode verificar com o código de barras apresentado inicialmente, confirmando a correta identificação do produto.

2.1.5 Processo de leitura

De forma a descodificar os códigos de barras é necessário dispor de um dispositivo capaz de fazer a captação do padrão e traduzi-lo em informação. O leitor de código de barras, também conhecido por *scanner*, é a ferramenta responsável por realizar essa tarefa, facilitando processos como o registo e controlo de *stock*, pois basta apontar para o código na embalagem e visualizar as informações de uma forma instantânea.

Atualmente existem diversos leitores no mercado, no entanto a sua escolha deve ter em atenção o tipo de operação e as necessidades da empresa para que seja implementado o leitor mais adequado.

- *Leitor a laser*

O leitor de código de barras mais comum, leitor a laser, emite um raio de luz (laser) que percorre as barras, capturando e transformando as barras em carateres legíveis para o humano. Ao posicionar o feixe de luz de forma a percorrer todas as barras, a luz, que aos olhos do ser humano é apenas uma fina linha vermelha, será absorvida pelas barras pretas e refletida pelas barras brancas. A luz refletida é convertida em um sinal elétrico através de um fotosensor (dispositivo que converte energia luminosa em energia elétrica) que, por sua vez, será então

transformado em bits. Ao reduzir a representação gráfica numa sequência de zeros e uns com a ajuda de um conversor analógico/digital, esta será interpretada por um sistema computorizado e transformado em informação percetível ao ser humano.

Uma das vantagens deste tipo de leitor é que este realiza a leitura sem contato direto com o código podendo descodificar códigos que estejam em superfícies planas, curvas e irregulares [26].

- *Laser omnidirecional*

O leitor a laser omnidirecional, que tal como o leitor a laser, também utiliza o laser para fazer a identificação, porém este leitor aplica vários feixes de luz com o auxílio de um conjunto de espelhos. Este aspeto torna a leitura num processo mais rápido e acessível, pois os códigos podem ser capturados em qualquer posição, conseguindo inclusive fazer a leitura de códigos danificados.

Geralmente, estes leitores são fixos, pois devido à sua capacidade multidirecional torna-se mais simples este ficar imóvel de modo a que os produtos sejam apontados para o leitor e não o contrário, sendo muitas vezes encontrados em caixas de supermercados [27].

- *Leitor CCD*

Os *scanners* CCD (*Charge-Couple Device*), também conhecidos como leitores de contato, possuem uma matriz com centenas de pequenos sensores de luz alinhados em uma linha para captar o código de barras. O leitor mede a luz ambiente emitida pelo código, *i.e.*, a intensidade de luz recebida, sendo assim necessário encostar o leitor ao código de barras.

São muito baratos, leves e consumem pouca energia, contudo torna-se bastante difícil ler códigos pequenos e códigos que estão em embalagens curvas ou irregulares, o que torna inadequado para a utilização em alguns estabelecimentos como, por exemplo, supermercados [27].

- *Leitor baseado em câmara*

O mais recente leitor de códigos de barras atualmente disponível é o leitor baseado em câmaras e em técnicas de processamento de imagem. Tal como os *scanners* CCD, estes também utilizam centenas de sensores de luz, no entanto

estes leitores têm os sensores dispostos em uma matriz bidimensional de forma a gerar imagens [28].

Seja para aplicações comerciais ou industriais, os leitores baseados em câmaras são mais complexos e robustos no que toca à leitura comparativamente com os leitores tradicionais, pois têm de realizar várias operações até conseguirem finalmente identificar a informação pretendida. Por outro lado, estes leitores não necessitam do conhecimento prévio da localização do código de barras e podem localizar vários padrões ao mesmo tempo, o que o torna num leitor de alto desempenho. De forma a identificar os padrões presentes nas imagens capturadas pelo leitor é aplicado um sistema de processamento e reconhecimento de imagem [29].

2.2 Visão computacional

A compreensão do funcionamento do sistema visual dos seres humanos, como a sua rápida capacidade de aprendizagem e a aptidão em realizar inferências e ações baseadas em estímulos visuais são os alicerces para o desenvolvimento e evolução das técnicas aplicadas para a análise automática de informações retiradas de uma imagem com o apoio de um computador.

A recriação/replicação da visão humana não é de todo uma tarefa fácil, sendo que existe toda uma ciência que procura reproduzir, por meio de um modelo computacional, as funções da visão humana, essa tecnologia chama-se visão computacional. Esta recorre a sensores óticos, técnicas de processamento digital de imagens, reconhecimentos de padrões e aplicações de Inteligência Artificial (IA).

A visão computacional pode ser definida como um processo que, através de algoritmos computacionais, extrai, carateriza e interpreta informações provenientes de imagens captadas do meio ambiente tridimensional. Todavia, o objetivo da análise de imagens, seja pelo ser humano seja por uma máquina, é a extração da informação útil consoante a aplicação ou problema [30].

O sistema visual humano é um sistema extremamente complexo que consegue interpretar imagens em frações de segundo. Para replicar as tarefas deste sistema com a ajuda de máquinas requer por antecedência uma compreensão dos conhecimentos humanos, o que faz com que o processamento de imagens seja seriamente dependente do sistema no qual está associado. Neste sentido pode-se afirmar que não existe uma solução única que abrange todos os problemas, *i.e.*, não pode ser generalizado. Portanto, até então, não existe um sistema de visão computacional complexo capaz de solucionar todos os casos [31].

Contudo, devido aos avanços surpreendentes no campo da visão computacional é possível, atualmente, fazer a identificação de objetos e/ou pessoas, deteção da ocorrência de eventos, análise de movimentos, reconstrução de cenários em 3D e, ainda, o reconhecimento de padrões que é a funcionalidade aplicada nos leitores baseados em câmaras [32].

2.2.1 Etapas de um típico sistema de visão computacional

Geralmente, um sistema de visão computacional segue diversas etapas desde a formação da imagem até à interpretação propriamente dita das informações úteis, sendo elas: aquisição, pré-processamento, segmentação, extração de caraterísticas, reconhecimento e, por fim, interpretação, tal como ilustra a Figura 2.8.

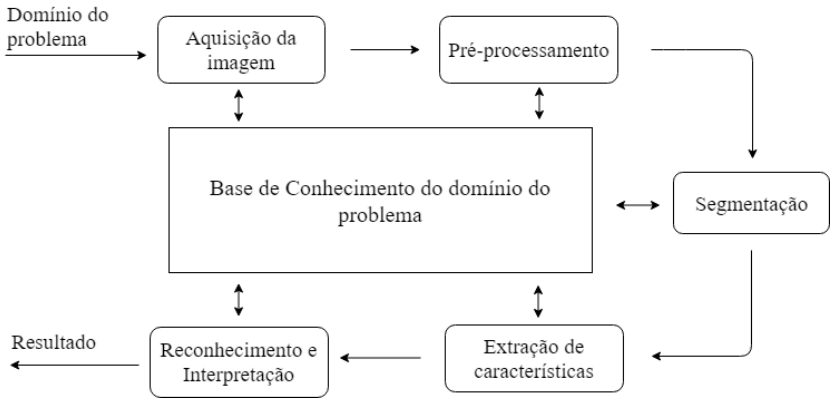


Figura 2.8: Estrutura de um sistema típico de Visão Computacional [5]

Todo o processo de um sistema de visão computacional inicia com a *aquisição de imagens* que podem ser representadas por duas ou mais dimensões. Essa tarefa de captura de imagens é muito importante, pois envolve o processo de digitalização, no qual uma imagem dever ser representada de forma apropriada ao tratamento computacional. Para isso, deve ser dada uma especial atenção às condições envolventes a este processo, como a iluminação do cenário, ajuste e preparação de imagem, de modo a obter uma imagem adequada para uma posterior análise, realçando toda a informação relevante. O dispositivo de captura escolhido, como uma câmara digital, câmara de vídeo, *scanner*, também interfere bastante no resultado final da imagem capturada, devido às caraterísticas internas que o equipamento apresenta.

Após a captura vem o *pré-processamento* que é responsável por filtrar os ruídos introduzidos pelo dispositivo e pelo ambiente físico que envolve a imagem capturada. Esta etapa tem como finalidade melhorar as caraterísticas relevantes na imagem, ou seja, diminuir ou, se possível, extrair a presenças de ruídos e certas anomalias e defeitos, introduzidos no processo de aquisição. Geralmente, neste processamento são aplicadas técnicas de filtragem de ruído, realce e reconstrução de imagem, entre outras, de modo a garantir a qualidade da imagem e uma adequada extração da informação pertinente.

Ao concluir o pré-processamento, a imagem fica pronta para ser fragmentada em unidades mais significativas, etapa conhecida como *segmentação*. O processo de segmentar uma imagem é de extrema importância, tal que o desempenho do sistema depende em grande parte do desempenho desta etapa. Esta consiste em dividir a imagem em diferentes regiões de interesse que serão posteriormente analisadas com algoritmos específicos em busca da informação relevante para a aplicação em questão. Existem vários fatores que tornam a precisão da segmentação num procedimento desafiante, como a iluminação reduzida ou excessiva, distorção da perspetiva, rotação/alinhamento do eixo longitudinal, resolução ou baixo foco da câmara. Contudo, existem várias técnicas de segmentação como: limiarização, segmentações baseadas em regiões, segmentação através de morfologia matemática, *watershed*, *wavelet*, contornos ativos e outras. Estas técnicas são baseadas na deteção de descontinuidades e similaridades de regiões da imagem, no entanto as técnicas aplicadas variam consoante as caraterísticas e finalidade da aplicação do sistema.

Com base nas áreas segmentadas na fase anterior, a etapa *extração de caraterísticas* procura extrair as caraterísticas ou propriedades dos objetos de interesse, que consiste na geração de descritores que caraterizam o objeto ou partes de uma imagem. Estes descritores podem ser representados através de um vetor de caraterísticas, contendo por exemplo, formas geométricas, entropia, energia, que poderão ser úteis para o reconhecimento e interpretação da imagem. Para a extração destes atributos são utilizadas técnicas como: código de cadeia, aproximação poligonal, descritores de Fourier, descritores regionais, medidas de texturas, etc [5, 33].

O *reconhecimento ou classificação de padrões* consiste em identificar objetos nas imagens em termos de padrões que as constituem e descobrir relações existentes entre grupos de objetos semelhantes baseados nas caraterísticas proveniente dos seus descritores. Atualmente existem muitas técnicas para a obtenção do reconhecimento de padrões que é normalmente dividido em três abordagens:

- *Estatística*: são usados métodos estatísticos para separar as classes de objetos de interesse. Dentro dos métodos utilizados, pode-se citar os classificadores Bayesianos, métodos probabilísticos e regras de decisão;

- *Estrutural*: os padrões são reconhecidos tendo como base a combinação de símbolos ou modelos que caracterizam e definem um padrão a ser reconhecido;
- *Neural*: o reconhecimento é obtido com o uso de redes neurais artificiais onde os padrões são obtidos pelo treinamento exaustivo da associação entre o padrão e a imagem analisada.

A *interpretação* da imagem consiste na fixação de um significado para um conjunto de objetos relacionados. Existem três principais formalismos utilizados para interpretar uma imagem:

- *Lógica formal*: a lógica clássica que é expressa por meio de regras lógicas;
- *Redes Semânticas*: faz o uso de grafos direcionados que permitem relacionar os elementos de uma imagem;
- *Sistemas baseados em conhecimento*: utiliza regras de produção para inferir o conhecimento. Outras abordagens podem ser agregadas a estas regras com o intuito de representar o conhecimento, tais como, lógica difusa, modal, epistémica, entre outras.

Por último, temos a *base de conhecimentos* que contém o conhecimento relativo ao domínio de problemas que são dependentes da aplicação para a qual esta base foi projetada. A complexidade e o seu tamanho variam e serve para realizar a comunicação entre os diversos módulos presentes no sistema conforme a complexidade da tarefa a ser realizada [5].

2.2.2 Histograma

O histograma de uma imagem digital é uma ferramenta bastante útil e utilizada na etapa de pré-processamento, pois fornece uma estrutura poderosa que viabiliza e facilita a realização de diversas técnicas de análise e processamento. Esta técnica passa pela representação do número total de pixéis da imagem em cada nível de intensidade de cinza, ou seja, fornece uma visão quanto à distribuição dos pixéis em relação ao contraste e níveis de iluminação da imagem. A informação gerada através do histograma é muitas vezes utilizada em técnicas aplicadas na etapa de segmentação como a limiarização.

Quando normalizado, o histograma de uma imagem digital com L níveis de cinza é definido por uma função discreta, do tipo [7]:

$$p(r_k) = \left(\frac{n_k}{n}\right) \tag{2.1}$$

onde:

- r_k representa o k-ésimo nível de cinza;
- $k = 0, 1, 2, \ldots, L-1$, onde L é o número de níveis de cinza da imagem digitalizada;
- n corresponde ao número total de pixéis na imagem;
- n_k representa o número de pixéis cujo nível corresponde a k;
- $p(r_\mathrm{k})$ revela a probabilidade estimada da ocorrência do k-ésimo nível de cinza.

O histograma pode ser considerado como uma função distribuição de probabilidades, obedecendo aos axiomas e teoremas da teoria de probabilidades, *i.e.* que [6]:

$$\sum_k p(r_k) = 1 \tag{2.2}$$

A Figura 2.9 mostra os histogramas de duas imagens com caraterísticas diferentes, sendo que a imagem (a) é uma imagem de baixo contraste, enquanto que a imagem (b) apresenta um maior contraste. Geralmente, os níveis de cinza encontram-se nas abcissas e a nas ordenadas a probabilidade estimada da ocorrência de um certo nível de cinza, tal como está representado na figura.

2.2.3 Limiarização

A limiarização, também conhecida por *thresholding*, é uma técnica de segmentação eficiente e simples no ponto de vista computacional, sendo, portanto, bastante utilizada em sistemas de visão computacional. O princípio da limiarização consiste em separar as regiões de uma imagem em duas classes, o objeto e o fundo, ou seja, colocando, de um modo geral, os objetos a preto e o fundo a branco [34].

Segundo [35], a limiarização consiste na definição de um ou mais limiares, por meio dos quais é estabelecido um critério para classificar os pixéis em uma

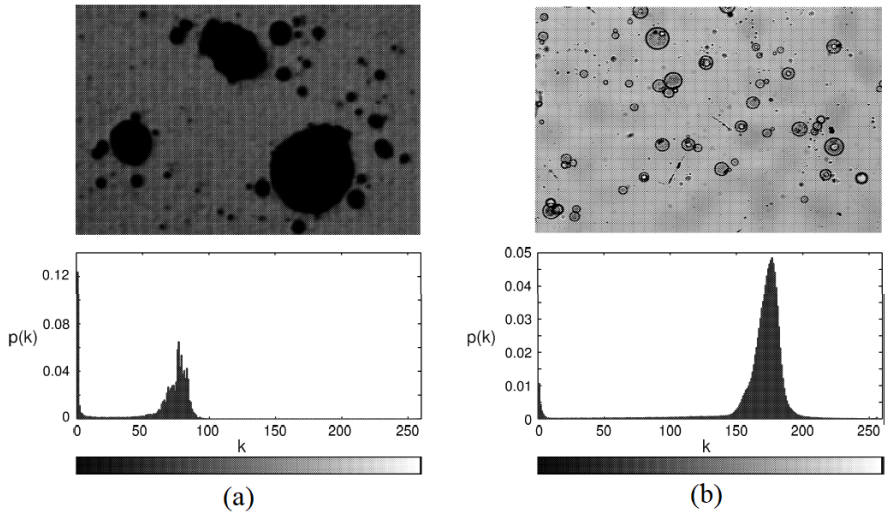


Figura 2.9: Exemplo de imagens com respetivos histograma: imagem de baixo contraste e seu histograma (a), imagem de alto contraste e seu histograma (b) [6]

imagem entre zero ou um, produzindo uma imagem binária, o qual esta técnica também pode ser denominada como binarização. Histogramas de níveis de cinza são muito utilizados para auxiliar este processo de segmentação.

Esta técnica parte da premissa de que níveis de cinza que pertencem a um objeto (região de interesse) se concentrem em um intervalo diferente dos níveis de cinza que constituem o fundo da imagem, permitindo assim realçar as regiões de interesse onde se encontra a informação útil. Portanto, o sucesso dos diversos métodos de limiarização existentes depende da determinação de limiares (certos níveis de intensidade de cinza), em que os pixéis são comparados e assim atribuído o valor zero ou um [35].

Em ambientes industriais, a iluminação é facilmente controlada deixando o objeto claro e o fundo escuro ou vice-versa, o que simplifica bastante este processo. Geralmente, nestes casos, as imagens captadas revelam um histograma bimodal, ou seja, apresentam dois picos ou regiões de maior incidência de pixéis. Todavia, nem todos os ambientes apresentam estas condições de iluminação, apresentando uma distribuição irregular com picos e vales pouco profundos ou multimodal, *i.e.* com mais de dois picos de maior incidência de pixéis, o que dificulta bastante a seleção dos limiares. No caso de distribuições bimodais, o limiar ou valor limite atribuído encontra-se no vale entre os dois picos separando as duas classes. Em distribuições irregulares e multimodais são estabelecidas técnicas de limiarização multinível capazes de definir múltiplos limiares de forma a isolar efetivamente as regiões de interesse [7, 34].

De acordo com [35], o processo de limiarização pode ser definido: com o uso de um limiar fixo L; com o uso de um intervalo fechado $[L1, L2]$; e com o uso de um esquema geral, no qual diferentes intervalos de limiares são definidos. Considerando $f(x, y)$ a intensidade (nível de cinza) do ponto (x, y), estas definições podem ser enunciadas nas seguintes formas:

- Quando um único limiar L for utilizado:

$$B(x,y) = \begin{cases} 1 & \text{se } f(x,y) \leq L \\ 0 & \text{caso contrário} \end{cases} \tag{2.3}$$

- Quando os valores de intensidade se situam em um intervalo $[L1, L2]$:

$$B(x,y) = \begin{cases} 1 & \text{se } L_1 \leq f(x,y) \leq L_2 \\ 0 & \text{caso contrário} \end{cases} \tag{2.4}$$

- Quando os valores de intensidade dos objetos de interesse são representados por um conjunto de intervalos de limiares Z:

$$B(x,y) = \begin{cases} 1 & \text{se } f(x,y) \in Z \\ 0 & \text{caso contrário} \end{cases} \tag{2.5}$$

A Figura 2.10 exemplifica a aplicação desta técnica de segmentação, em que a imagem capturada apresenta sombras e elementos indesejados que são eliminados após a limiarização, exceto o objeto que é a região de interesse.

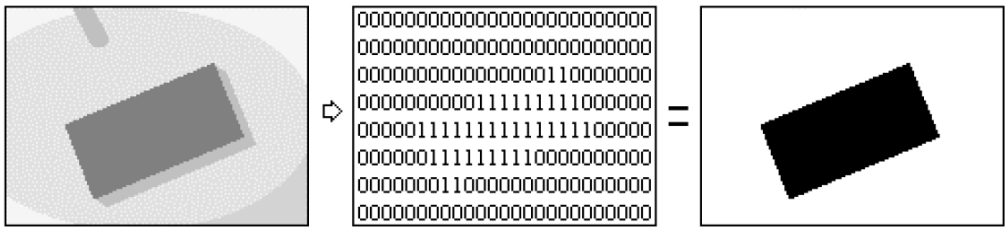


Figura 2.10: Exemplo de um processo de limiarização [7]

2.2.4 Deteção de bordas

A deteção de bordas é uma das operações de segmentação baseada em descontinuidade mais utilizada, sendo que envolve essencialmente a localização de objetos de interesse. Existem diversas definições possíveis para borda, no entanto a mais utilizada é a ideia de limite ou fronteira, onde um conjunto de pixéis são conectados para separar duas regiões com propriedade distintas de níveis de cinza. Deste modo, este método de segmentação passa por delinear, através da determinação das bordas, os objetos existentes na imagem, permitindo a separação dos mesmos [5].

Imperfeições no processo de captura podem provocar a suavização de pixéis e até mesmo gerar contornos borrados (ruído), o que dificulta a deteção de bordas. A Figura 2.11 mostra alguns exemplos do perfil de intensidade ao longo de duas imagens com diferentes níveis de ruído nas bordas, ou seja, na transição do nível de cinza.

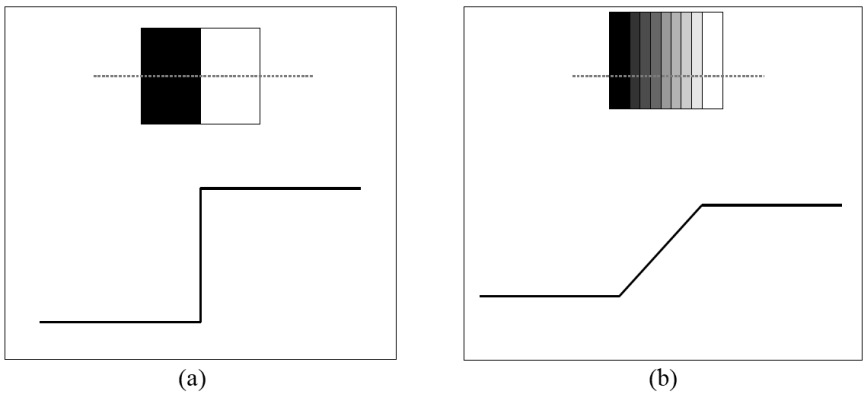


Figura 2.11: Exemplos de bordas: borda ideal (a), borda com ruído (b) [5]

Quanto maior é a mudança no nível de cinza (intensidade) ou quanto mais próximo estiver do caso ideal (Figura 2.11 (a)) mais fácil é a deteção da borda. No entanto, na prática, a deteção de bordas é uma tarefa complexa devido às imperfeições adquiridas no processo de captura, resultando em uma imagem com bordas borradas (Figura 2.11 (b)).

A maioria das técnicas de deteção de bordas tenta resolver este tipo de problemas obtidos na aquisição da imagem com métodos baseados no mecanismo básico de definir um operador derivativo de primeira ou segunda ordem, associados a alguma técnica de suavização para reduzir o efeito de ruídos.

Os operadores de *Sobel*, *Prewitt* e *Roberts* são dos métodos mais conhecidos na deteção de bordas que utilizam a primeira derivada para realizar a identificação das mudanças significativas de níveis de cinza. Esses operadores fazem uso da diferenciação de imagens chamado de gradiente, que se refere a um vetor que indica as regiões onde existe maior variação dos níveis de cinza [5].

O vetor gradiente de uma imagem $f(x, y)$ no ponto (x, y) pode ser calculado por derivadas parciais, tal como mostra a Expressão 2.6.

$$\nabla f(x,y) = \begin{bmatrix} G_x \\ \overline{G_y} \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{bmatrix} \tag{2.6}$$

A magnitude do vetor gradiente, que equivale à maior taxa de variação f por unidade de distância, e o ângulo de direção desse vetor são as duas medidas relevantes na deteção de bordas derivativas. Estas são expressas respetivamente por:

$$\nabla f(x,y) = mag(\nabla f) = \sqrt{G_x^2 + G_y^2} \tag{2.7}$$

$$\theta = arcotan \left(\frac{\frac{\delta f}{\delta x}}{\frac{\delta f}{\delta y}} \right) \tag{2.8}$$

Devido ao custo computacional requerido para calcular a magnitude do gradiente, esta pode ser calculada por valores absolutos ou pelo valor máximo entre

os gradientes na direção x e y. Uma forma simples de calcular a magnitude é através do uso de diferenças em ambas as direções, onde G_x e G_y poderiam ser expressas por:

$$G_x = f(x,y) - f(x+1,y) \qquad G_y = f(x,y) - f(x,y+1) \tag{2.9}$$

Desta forma, esses valores poderiam estar dispostos através de uma representação matricial de tamanho 3x3 que compõe uma região da imagem, tal como ilustra a Figura 2.12. Esta representação é intitulada de máscara e pode ser utilizada para implementar a Expressão 2.7, sendo considerada uma aproximação da equação.

f(x-1, y-1)	f(x, y-1)	f(x+1, y-1)
f(x-1, y)	f(x, y)	f(x+1, y)
f(x-1, y+1)	f(x, y+1)	f(x+1, y+1)

Figura 2.12: Representação Matricial de tamanho 3x3 pixéis [5]

Os operadores *Sobel* , *Roberts* e *Prewitts* podem ser definidos através de máscaras, tal como ilustra a Tabela 2.4.

Tabela 2.4: Máscaras dos vários operadores [5]

Operador	Máscara G_x	Máscara G_y
Roberts	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$
Prewitt	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

Em *Roberts*, a deteção identifica bordas diagonais da esquerda e direita, enquanto que em *Prewitt* e *Sobel* a preocupação está na obtenção de bordas horizontais e verticais. Em *Sobel* , é possível observar a utilização do valor 2 na posição central da máscara, que tem como finalidade suavizar a imagem nessa posição. De acordo com a necessidade e com a aplicação deste método, é possível adaptar as máscaras dos operadores para obter a deteção de bordas diagonais, tal como acontece no operador de *Roberts*. Isso pode ser realizado rotacionando os valores em 45 graus [5].

Na Figura 2.13 é possível observar a deteção de bordas com os vários operadores aplicados a uma mesma imagem.

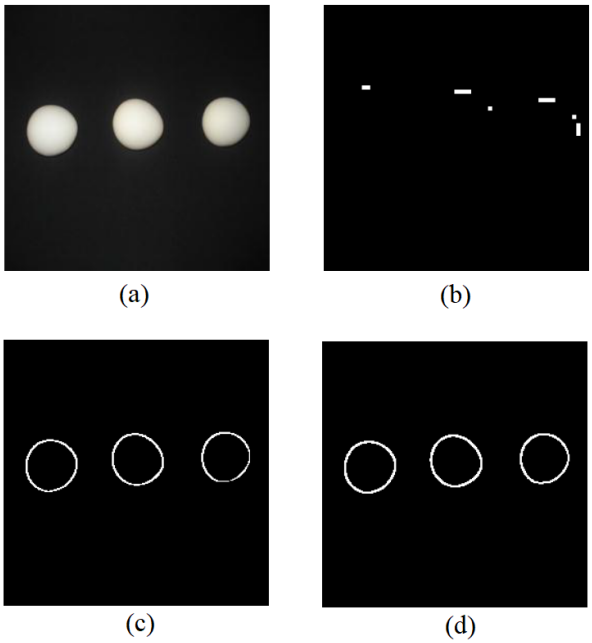


Figura 2.13: Deteção de bordas da imagem digital (a) com o operador de : *Roberts* (b), *Prewitt* (c) e *Sobel* (d) [8]

2.3 Raspberry Pi

Raspberry Pi é um computador de pequeno porte (microcomputador) de baixo custo, originalmente desenvolvido pela Fundação *Rasbberry Pi* em 2012, com a finalidade de promover o ensino básico em computação nas escolas, com produtos de preço acessível. Contudo, por ser um dispositivo bastante versátil, tornou-se rapidamente na *single-board computer* mais popular no mercado, com

milhões de vendas em todo o mundo. Esta placa possibilitou o desenvolvimento de aplicações que vão desde o desenvolvimento de jogos até ao desenvolvimento de soluções que envolvem visão computacional [36].

A nível de *hardware*, o Raspberry Pi é composto por um microprocessador com unidade de processamento gráfico integrado, memória cache, memória RAM, leitor de cartão SD como unidade de armazenamento, e ainda diversas entradas e saídas, tudo compactado numa só placa do tamanho de um cartão de crédito. Esta pequena placa apresenta portas USB, conetores HDMI que oferece saída de áudio e vídeo digital, uma porta *Ethernet* e ainda uma saída de áudio analógico de 3,5 milímetros, sendo que toda a placa é alimentada através de um USB tipo C para o modelo mais recente ou um micro-USB para modelos mais antigos. Existe ainda uma grande quantidade de pinos de entrada e saída de uso geral (GPIO - *General Purpose Input/Output*) no Raspberry Pi que permite o microcomputador comunicar com outros dispositivos.

O *Raspberry Pi 4 Model B* é o mais recente produto da gama Raspberry Pi. Esta é a placa que apresenta melhor desempenho até à data, permitindo desenvolver os mais variados projetos [12].

2.3.1 Linguagens associadas

Na década de 40, foram criados os primeiros computadores que eram programados através de milhares de interruptores, em que cada um deles podia assumir o valor 1 ou 0 consoante se o interruptor estava ligado ou desligado, tornando a sua programação difícil e cansativa. Devido à ausência de *software* e aos problemas existentes relativamente à tecnologia, nasceram os primeiros algoritmos descritos por intermédio de uma linguagem de programação.

Ao longo da história da computação, foram desenvolvidas diversas linguagens, introduzindo cada vez mais facilidades e recursos, o que tornou a tarefa de programar mais acessível e menos sujeita à ocorrência de erros.

As placas Raspberry Pi foram projetadas para incentivar os jovens a aprender a programar, tanto que a palavra "Pi" em Raspberry Pi vem da linguagem de programação *Python*. Contudo, num curto espaço de tempo foram adaptadas diversas linguagens de programação para esta placa, de forma a ceder aos seus utilizadores um maior suporte e escolha. *Python*, *Java*, *Javascript*, C, C++ são algumas das linguagens disponíveis no Raspberry Pi.

A linguagem *Python* é uma das linguagens de programação de código aberto mais populares em todo o mundo. Esta foi criada em 1991 por *Guido Van Rossum* que originalmente tinha como finalidade criar uma linguagem simples e de fácil compreensão para usuários como físicos e engenheiros. Atualmente, é gerenciado pela *Python Software Foundation* e é considerada uma linguagem de

programação de alto nível, *i.e.* mais próximo da linguagem humana, o que a torna mais intuitiva e simples para o programador, simplificando a tradução de um raciocínio em um algoritmo.

Esta linguagem de programação oferece um extenso repositório de bibliotecas e recursos e permite ainda a importação de bibliotecas criadas em linguagens como C, C++ e *Fortran*. A biblioteca padrão contém classes, métodos e funções para realizar essencialmente qualquer tarefa, desde o acesso a bases de dados até a interfaces gráficas com o utilizador, sendo, por isso, utilizada nas mais diversas aplicações (propósito geral).

Hoje, Python está disponibilizado em todas as distribuições populares do *Linux* e é usada como linguagem de referência do microcomputador Raspberry Pi [37].

2.3.2 OpenCV

Open Source Computer Vision Library (OpenCV), desenvolvida pela *Intel* no ano 2000, é uma biblioteca usada para o desenvolvimento de aplicações na área de visão computacional. Esta foi criada para proporcionar uma infraestrutura para aplicações de visão computacional e para acelerar o uso da perceção de máquinas em produtos comerciais, aumentando assim a eficiência computacional focando-se essencialmente em aplicações em tempo real. Trata-se de uma biblioteca de código aberto, o que faz com que seja simples de utilizar e, caso necessário, modificar o código, sendo totalmente livre tanto para o uso académico como comercial.

A biblioteca apresenta mais de 2 500 algoritmos otimizados, que abrange um extenso conjunto de algoritmos relacionados com visão computacional bem como inteligência artificial. Todas estas funções e algoritmos estão disponíveis no site *https://opencv.org/*, sendo que estes podem ser usados para detetar e reconhecer faces, identificar objetos, classificar ações em vídeos, identificar objetos em movimentos, extrair modelos em 3D de objetos, encontrar imagens semelhantes em um base de dados, reconhecer cenários, entre muitos outros.

Atualmente, mais de 47 mil pessoas utilizam o OpenCV e até ao momento o número estimado de *downloads* da biblioteca excede os 18 milhões. A biblioteca tem sido usada por uma extensa quantidade de empresas, investigadores e até mesmo órgãos governamentais. Organizações como a *Google*, *Yahoo*, *Microsoft*, *IBM*, *Sony*, *Honda*, *Toyota* são algumas das muitas empresas que fazem o uso da biblioteca.

Apesar de ser desenvolvida na linguagem de programação C e C++, esta possui interfaces C, C++, Python, *Java* e *Matlab*, no entanto a linguagem Python

tem sido cada vez mais adotada para o uso desta biblioteca. O OpenCV está disponível em diversos sistemas operativos, destacando o *Windows*, *Linux*, *Android*, *Mac OS* e *iOS* [38].

Relativamente à análise de códigos de barras, o OpenCV não possui módulos específicos para a realização da leitura e descodificação de códigos de barras, no entanto apresenta uma variedade de ferramentas de interpretação de imagens que facilita este processo, indo desde a captura de imagem de um vídeo, até ao processamento de imagens, incluindo diversos métodos para que os dados sejam extraídos da imagem com sucesso [39].

Após obter a imagem processada, esta será submetida à descodificação propriamente dita através de uma outra biblioteca. Zbar é uma das bibliotecas de código aberto que permite interpretar e descodificar códigos de barras. Ela suporta várias simbologias, incluindo a EAN-13, UPC-E, EAN-8, Código 128, Código 39 e QR Code.

A biblioteca ZBar usa uma abordagem que parte do mesmo raciocínio dos leitores a laser, ou seja, faz uma leitura linear sobre a imagem processada, considerando cada pixel como uma amostra de um único sensor de luz, que em conjunto produz o fluxo de dados pretendido. A união destas duas bibliotecas, OpenCV e Zbar, permite de uma forma acessível a deteção e descodificação de códigos de barras em tempo real [40].

2.4 Análise de mercado

Neste subcapítulo serão apresentadas algumas aplicações e tecnologias presentes no mercado usadas para facilitar o dia-a-dia do consumidor. Na atualidade, existem inúmeras aplicações capazes de ajudar o consumidor nas mais variadas formas, seja a pagar de forma segura e numa questão de segundos, seja a relembrar todos os produtos que estão em falta na despensa.

2.4.1 *Continente Siga*

A aplicação *Continente Siga* é disponibilizada para uso exclusivo dos clientes das lojas Continente possibilitando a estes partilhar listas de compras, realizar compras e pagar na própria aplicação, tudo de forma rápida, segura e cómoda.

Utilizável nas lojas da cadeia Continente, mais especificamente nas lojas Continente, Continente Modelo e Continente Bom Dia aderentes, esta permite ao utilizador usufruir de diversas funcionalidades que ajudam a organizar e controlar o processo de compra, tanto *online* como nas lojas.

Com a *Siga*, o utilizador pode:

- Criar listas de compras partilhadas, em tempo real, com a família, *i.e.*, utilizadores autorizados da conta Cartão Continente;
- Gerir as listas, através de voz, texto e leitura de códigos de barras com a câmara do *smartphone*;
- Comprar *online* com entrega em casa ou através do serviço *ClickGo* (serviço disponibilizado para a recolha das suas compras na loja ou no carro);
- Fazer a compra em loja diretamente no *smartphone*, através da leitura direta dos códigos de barras dos seus produtos;
- Aceder a campanhas, promoções e produtos mais relevantes para o cliente de acordo com o histórico de compras e as suas escolhas;
- Obter resultados de pesquisa inteligente que considera as preferências de compra do cliente;
- Utilizar cupões na compra;
- Fazer o *checkout* (pagamento) em loja e *online* de forma rápida e segura;
- Usar o saldo do Cartão Continente;
- Receber sugestões de descontos personalizadas;
- Aceder a informação detalhada das compras prévias nas lojas Continente;

Os clientes que usufruem deste serviço conseguem moderar quanto vão gastar, pois a aplicação apresenta sempre o valor total da compra, bem como quanto vai poupar, como se pode verificar na Figura 2.14.

O pagamento em loja pode ser feito de imediato a partir do telemóvel com o Continente Pay ou em caixas exclusivas nas lojas aderentes. O Continente Pay é um serviço onde é possível associar um ou vários cartões de débito e crédito, *Visa* ou *Mastercard* para pagamentos nas lojas Continente.

Esta aplicação que conta agora com mais de 120 mil utilizadores registados permite aos utilizadores diversas funcionalidades, como o *self-scan*, que transformam o processo de compra verdadeiramente autónomo e sem filas, reduzindo assim o tempo despendido na loja e, consequentemente, o tempo total atribuído a esta atividade. Todos estas funcionalidades tornam a aplicação cómoda para os clientes da insígnia alimentar pertencente ao universo da Sonae.

Em situações de pandemia, esta aplicação vem desempenhar um papel muito importante, pois permite realizar compras de forma rápida e, assim, diminuir o tempo e o contato com os clientes e funcionários nos supermercados, reduzindo

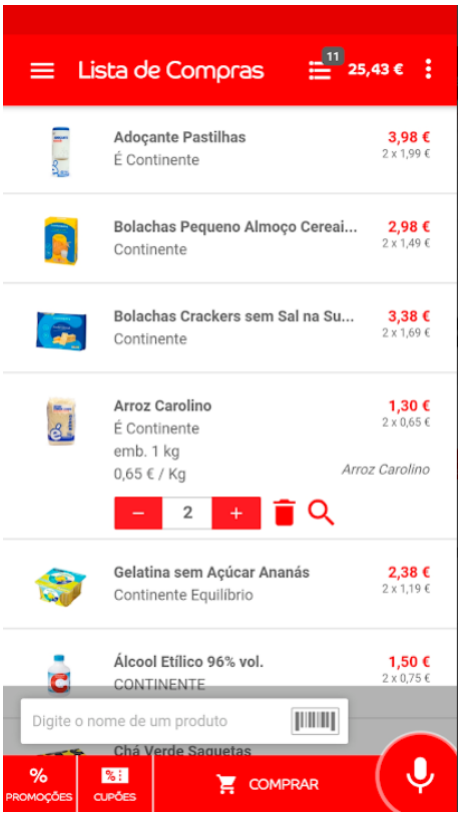


Figura 2.14: Aplicação *Continente Siga* [9]

o possível contágio do cliente. A *Continente Siga* verificou uma forte evolução na adesão deste serviço disponibilizado por esta cadeia alimentar, que registou mais 80% de transações desde o início da pandemia provocada pela Covid-19 em 2020 [9] [41].

2.4.2 *Auchan*

A aplicação móvel *Auchan* foi lançada em 2018 com o objetivo de disponibilizar aos seus clientes portugueses um serviço simples para o registo e gestão de listas de compras e ainda para efetuar compras *online* e em loja. Os clientes ao usufruir deste serviço podem poupar tempo nas compras e passar a ter ao dispor no seu *smartphone* a lista de compras de toda a família, sendo que cada um, no seu *smartphone*, pode adicionar e remover produtos.

A *Auchan Retail* afirma que "a principal vantagem para quem vai às compras é poder ir registando os produtos, através da leitura dos códigos de barras, à medida que se colocam no saco. Com a funcionalidade *Scan Expresso* controla-

se o total a pagar e no final não é necessário retirar os produtos do saco ou do carrinho. É só pagar, sem filas. Esta aplicação pode ajudar a reduzir em 30% o tempo gasto num percurso de compras típico em loja. É fácil e ganha-se tempo e conforto.".

Ao instalar a aplicação, o utilizador tem acesso a várias funcionalidades, nomeadamente a consulta de promoções, folhetos e informações sobre todos os produtos em loja, como preço, informações promocionais, ingredientes e alergénios. A leitura dos códigos de barras através da câmara do *smartphone* pode ser aplicada tanto na compra em loja como *online*, de forma a adicionar produtos à compra, tornando a compra mais prática. No momento de pagamento na loja, o cliente não precisa de tirar os produtos do carrinho nem perder tempo em filas, pois basta dirigir-se a uma caixa exclusiva a este serviço e efetuar o pagamento através de um código de barras gerido pela aplicação.

No lançamento da aplicação era necessário a leitura do QR code de *check in* que se encontrava à entrada da loja para dar início à compra. Atualmente, a *Auchan* pode identificar a loja onde o cliente se localiza de forma rápida e fácil através da geolocalização, iniciando assim a compra sem perder tempo. Para isso, o utilizador terá de ativar a geolocalização no seu telemóvel e permitir o seu uso na aplicação. É possível ainda procurar as lojas *Auchan* por distrito, concelho ou localidade e visualizar as lojas existentes mediante um mapa ou lista.

Todas estas funcionalidades tornam esta atividade do dia-a-dia do consumidor mais segura, simples e rápida. A aplicação *Auchan*, disponível para *download* na *App Store* e na *Google Play*, pode ser utilizada em 34 superfícies das insígnias Jumbo e Pão de Açúcar em Portugal, sendo que já mais de 200 mil portugueses desfrutam das funcionalidades que esta aplicação apresenta. A Figura 2.15 evidencia algumas dessas funcionalidades, tal como o acesso a diversos folhetos e pesquisa por seções [42] [43].

2.4.3 *Bring*

A *Bring!* foi criada especialmente para facilitar a tarefa de fazer compras, oferecendo uma maneira muito simples e prática de criar listas de compras. Além disso, tem a possibilidade de compartilhar a lista com membros familiares ou amigos. Algumas funcionalidades incluindo a compartilha das listas requerem que o utilizador se registe na plataforma, porém para criar uma lista pessoal não é necessário criar uma conta na aplicação.

Possui uma interface bastante amigável e intuitiva que torna todo o processo de criar e editar as listas acessível para qualquer utilizador. De modo a simplifi-

Figura 2.15: Aplicação *Auchan* [10]

car a pesquisa de produtos, estes estão organizados por categorias e são identificados facilmente por figuras/ícones que representam o produto, tal como ilustra a Figura 2.16.

A possibilidade de compartilhar as listas criadas ampliam ainda mais a flexibilidade desta aplicação, permitindo que outras pessoas possam contribuir no processo. Para além disso, existem muitas outras funcionalidades, tais como:

- Criar e personalizar listas de compras para qualquer ocasião;
- Gerir as listas de compras compartilhadas via *smartphone*, *tablet* ou *smartwatch* e com assistentes de voz *Alexa*, da *Amazon* e *Google*;
- Enviar mensagens inteligentes (notificações), pré-escritas, para informar os contatos alterações na lista ou até mesmo informar que o utilizador vai fazer compras;

Figura 2.16: Aplicação *Bring!* [11]

- Adicionar fotos aos itens da lista de compras;
- Adicionar produtos, caso o catálogo *Bring!*, que contém centenas de produtos, não inclua o item;
- Receber sugestões e lembretes de compras individuais, adaptados às necessidades do utilizador;
- Importar facilmente receitas e ingredientes de plataformas de receitas e *blogs* de comida.

Com mais de 5 milhões de *downloads*, esta aplicação permite ao utilizador poupar tempo, dinheiro e energia ajudando a organizar as compras de uma forma simples [11].

Capítulo 3

Arquitetura do sistema

O presente capítulo aborda a arquitetura geral do sistema. São expostos os requisitos e tecnologias necessárias para que o sistema proporcione uma melhor experiência ao utilizador no processo de compras. São apresentados ainda os *softwares* utilizados para o desenvolvimento do projeto.

3.1 Requisitos

Analisando os objetivos projetados anteriormente, o sistema deverá ter em consideração vários pontos essenciais para que auxilie eficazmente o utilizador. Para tal, deverá cumprir os seguintes pontos:

- Detetar produtos colocando o código de barras destes diante da câmara incorporada no caixote do lixo;
- Transmitir os códigos identificados de forma a serem inseridos numa lista de compras;
- Permitir a inserção de dados relativos ao produto quando o código de barras deste não é reconhecido;
- Permitir ao utilizador, através do seu *smartphone*, controlar todo o processo de criação de listas de compras tal como adicionar, editar e remover produtos da lista;

Considerando os tópicos mencionados, pode-se dividir este projeto em três partes: o sistema de captura de produtos inserido no caixote do lixo, a aplicação móvel, e o servidor, responsável por atender aos pedidos realizados.

Na Figura 3.1 é possível visualizar como se encontram interligados as várias partes que em conjunto formam o sistema.

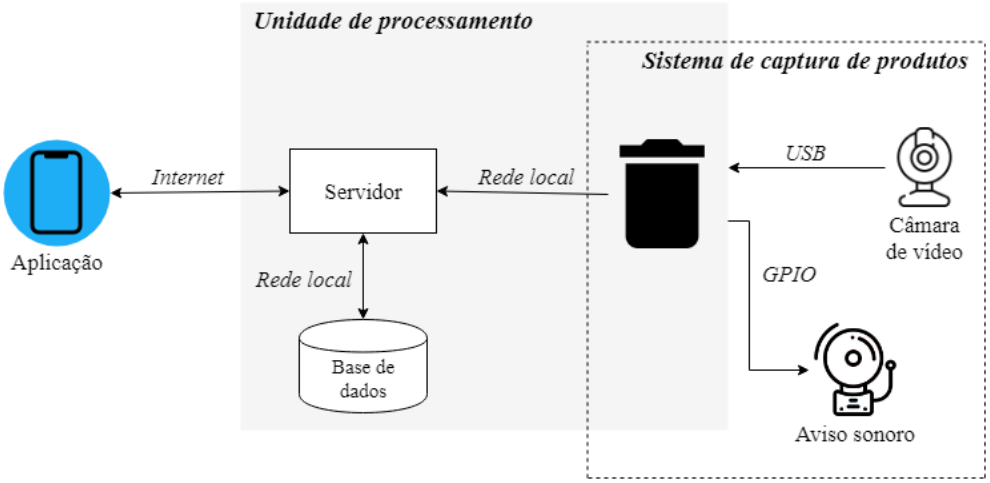


Figura 3.1: Arquitetura do Sistema

Tanto o servidor como a base de dados encontram-se na unidade de processamento central que engloba igualmente o sistema de captura de produtos. Por esse motivo, as ligações Servidor – Base de dados e Servidor – Sistema de captura de produtos são realizadas dentro da rede local, ou seja, na unidade de processamento, como explanado na Figura 3.1.

A aplicação precisa somente de aceder ao servidor para que esta apresente todos os dados necessários ao utilizador. Esta interligação entre ambos é realizada por *internet*, o que requer que tanto a aplicação móvel como a unidade de processamento estejam ligados a esta ao longo da sua utilização.

3.2 Sistema de captura de produtos

O sistema que se encontra na residência do utilizador, mais precisamente no seu caixote do lixo pessoal, é constituído por uma câmara, uma unidade de processamento e um alerta sonoro.

A câmara tem como função recolher informação em tempo real do ambiente circundante ao caixote do lixo em formato digital. Esta, por sua vez, estará ligada a uma unidade de processamento, que irá interpretar a informação capturada.

Atendendo aos requisitos e consequente estudo, o microcomputador *Raspberry Pi 4 Model B* foi o escolhido como unidade de processamento responsável por analisar as imagens vindas da câmara. Este, quando deteta um código de barras nas imagens recebidas, envia um pedido ao servidor com o código recolhido. Fica ao encargo do servidor verificar se já existe um produto associado ao código capturado ou se é necessário criar um novo produto.

No que concerne a nível do utilizador, este terá somente de posicionar o código de barras do produto em frente da câmara a uma distância considerável, de modo a que não esteja demasiado próximo ou excessivamente distante, dificultando a leitura do código. De modo a assistir o utilizador neste procedimento, o microcomputador integra um aviso sonoro. Este aviso é constituído somente por um único componente, por um pequeno besouro ou *buzzer*. Este é ativado sempre que é identificado um código no campo de visão da câmara. Desta forma, o utilizador saberá se o produto foi lido com sucesso, dando a indicação que poderá colocar o produto no lixo.

3.3 Aplicação móvel

Tento em conta que o utilizador requer uma interface para visualizar as listas de compras, uma aplicação móvel é uma das opções mais viáveis e práticas.

A aplicação deve englobar toda a informação que o utilizador necessita no momento de compra, i.e., os produtos a comprar assim como as quantidades correspondentes, indicando através de uma lista toda essa informação.

Os códigos de barras recolhidos pelo sistema são enviados para a aplicação para que o utilizador possa adicionar os produtos lidos nas listas que entender. Embora o objetivo é controlar o que se encontra em falta na despensa através do sistema de captura situado no caixote do lixo, também deverá ser possível adicionar produtos, mediante a aplicação.

3.4 Servidor

Um servidor é desenvolvido especificamente para gerir e transmitir informações aos seus clientes. Ao fornecer informação de forma centralizada, um servidor consegue servir vários clientes, partilhando dados entre estes.

Neste Projeto, o servidor tem um papel fundamental para interligar o sistema de captura de produtos e a aplicação. Em conjunto com uma base de dados, o servidor responde aos pedidos feitos pelos seus clientes (sistema de captura e a aplicação móvel), gerindo a informação e/ou enviando os dados pedidos.

O servidor foi desenvolvido através da *framework NestJS*, ideal para o desenvolvimento de servidores, oferecendo uma estrutura adequada. Combina *JavaScript* progressivo juntamente com *TypeScript* que se baseia neste primeiro. Estas duas linguagens de código aberto em conjunto com a arquitetura que a *framework* apresenta permitem construir facilmente um servidor devido à sua capacidade de criar aplicações altamente testáveis, escaláveis, e de fácil manutenção, dando um bom suporte no seu desenvolvimento. Na secção 3.6.1 é descrita com mais detalhe esta *framework* atualmente usada por todo o mundo.

A Figura 3.2 mostra como se encontra integrada a *framework* juntamente com o restante Projeto.

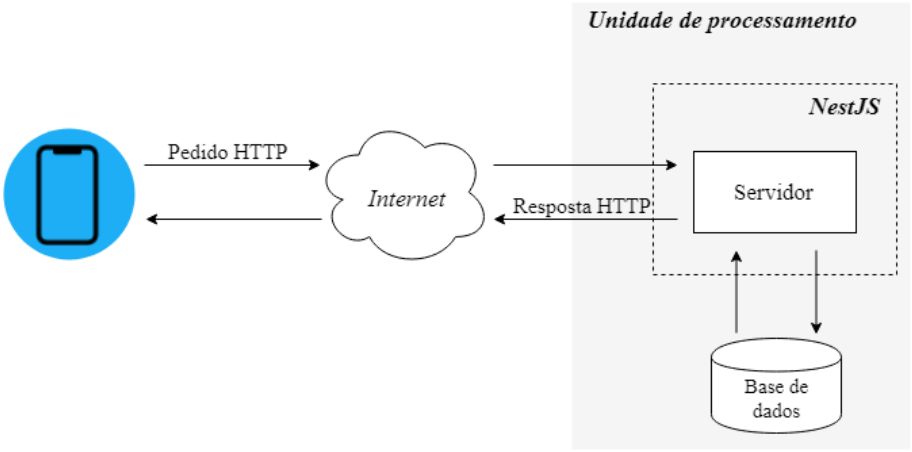


Figura 3.2: Pedido HTTP ao servidor

3.5 Tecnologia utilizada

O primeiro passo na implementação deste Projeto passou por analisar a tecnologia e *hardware* disponível no mercado atual, considerando os objetivos e requisitos definidos nas seções 1.3 e 3.1. O sistema de captura de produtos usado para este Projeto foi concebido agregando vários componentes, sendo estes:

- *Raspberry Pi 4 Model B*;
- Câmara de vídeo com resolução de 2 megapixéis;
- Besouro 3-24 V.

3.5.1 Raspberry Pi

Como já foi mencionado, a unidade de processamento selecionada foi o microcomputador *Raspberry Pi 4 Model B* que agrupa o sistema de captura de produtos, servidor e base de dados. Esta é a última placa da popular gama Raspberry Pi, sendo que comparativamente com a geração anterior (*Raspberry Pi 3*), esta oferece avanços inovadores na velocidade do processador, desempenho no que toca à multimédia, memória e conectividade, mantendo a compatibilidade similar em consumo de energia.

O *Raspberry Pi 4 Model B* pode ser comparável com sistemas PC x86 de nível básico no que se refere ao desempenho. Os principais recursos deste produto incluem um processador quad-core de 64 bits de alto desempenho, suporte a dois monitores em resoluções de até 4k através de duas portas micro-HDMI, decodificação de vídeo em *hardware* de até 4kp60, até 4 GB de RAM, USB 3.0, entre outros.

Inclui ainda um conjunto de 40 pinos GPIO que possibilita ao programador comunicar com outros dispositivos criando projetos inovadores, alargando assim o leque de possibilidades que a placa oferece [12]. A linguagem Python e as bibliotecas OpenCV e Zbar funcionam em conjunto neste Projeto para conseguir detetar e ler códigos de barras de forma a que essa informação seja tratada posteriormente.

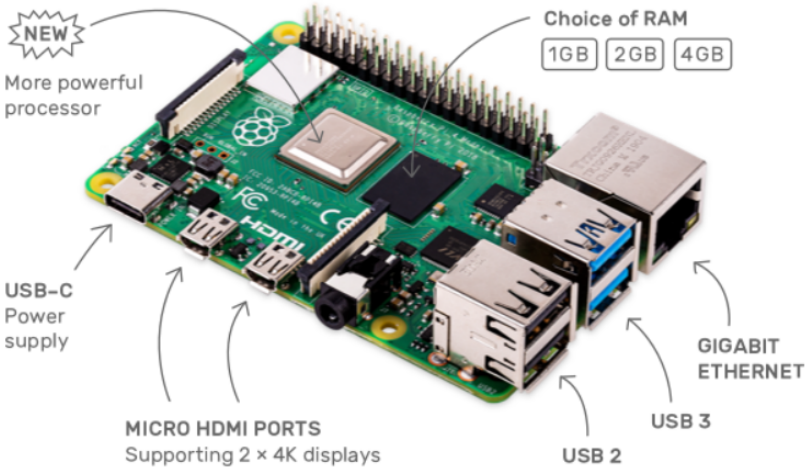


Figura 3.3: *Raspberry Pi 4 Model B* [12]

3.5.2 Câmara de Vídeo

Como câmara de vídeo foi escolhida uma câmara com uma resolução de 2 megapixéis (1920 x 1080 pixéis) (Figura 3.4). Esta câmara apresenta uma lente de 2,1 mm para uma ampla faixa de visão, dando um efeito de "olho de peixe". Esta lente tem um grande ângulo de visão possibilitando visualizar o meio ambiente circundante em torno dos 80 a 120 graus. Este faculta o *driver* USB *Video Class* (UVC) versão 1.1 que é muito utilizado para *streaming* de vídeo como *webcams*, não sendo necessário *drivers* adicionais.

O modo de comunicação é feito através do protocolo USB, tornando a sua ligação com o microcomputador *Raspberry Pi 4 Model B* muito simples. Esta câmara tem como função captar informação do meio ambiente envolvente em formato digital de modo a, posteriormente, detetar códigos de barras [13].

Figura 3.4: Câmara de vídeo [13]

3.5.3 Aviso sonoro

Tal como já foi mencionado anteriormente, o aviso sonoro tem como objetivo informar o utilizador que o produto colocado em frente ao caixote do lixo foi lido com sucesso, sendo da responsabilidade da unidade de processamento juntamento com a câmara de vídeo capturar cada fragmento de imagem e detetar a existência de um ou mais códigos de barras que identificam, por sua vez, os produtos a depositar no lixo. O besouro escolhido para o efeito tem um leque de funcionamento de 3 a 28 Volts (Figura 3.5) [14].

Este besouro será ligado aos pinos de GPIO do *Rasberry Pi 4 Model B* e será ativado sempre que o sistema capture um código de barras.

Figura 3.5: Besouro (3 a 24 Volts) [14]

3.6 Descrição dos *Softwares* utilizados

Nos mais diversos níveis de desenvolvimento do projeto, o *software* assume grande destaque, englobando o sistema de captura de produtos, a aplicação móvel e o servidor.

3.6.1 NestJS

NestJS é uma *framework* para desenvolvimento de aplicações *backend* em *Node.js*. Este usa *JavaScript* progressivo e é construído com *TypeScript* suportando igualmente este. Combina ainda elementos de *Object Oriented Programming* (OOP), traduzindo Programação Orientada a Objetos, *Functional Programming* (FP), e *Functional Reactive Programming* (FRP).

Nos últimos anos, graças ao *Node.js*, o *JavaScript* tornou-se a principal linguagem *web* para aplicações *front-end* e *back-end*. Isso originou projetos como *Angular*, *React* e *Vue*, que melhoram a produtividade do programador e permitem a criação de aplicações *front-end* rápidas, testáveis e extensíveis. No entanto, embora existam muitas bibliotecas e ferramentas excelentes para o *Node* (*JavaScript* do lado do servidor), nenhum deles proporciona uma arquitetura como a que *NestJS* oferece. Esta fornece uma arquitetura que permite aos programadores e equipas de desenvolvimento criar aplicações altamente testáveis, escaláveis e de fácil manutenção, sendo a sua arquitetura fortemente inspirada no *Angular*. Atualmente, empresas como Sanofi, Adidas, AutoDesk usam esta *framework* [44].

Neste Projeto, esta estrutura é utilizada para construir o servidor que irá responder às solicitações HTTP vinda dos seus clientes, ficando à escuta dos pedidos. Este servidor também considerado como uma *Web Application Programming*

Interface (API), utiliza o protocolo HTTP para a construção de serviços que podem ser depois acedidos em diferentes plataformas.

A Figura 3.6 demonstra como opera a *framework NestJS* quando recebe um pedido HTTP. O *Controller*, em português controlador, tem como objetivo lidar com as solicitações, ou seja, receber pedidos e devolver as respostas aos clientes. De forma a saber que controlador recebe o pedido, é utilizado um mecanismo de roteamento designado por prefixo. A inserção do prefixo permite agrupar facilmente o conjunto de rotas minimizando código repetitivo.

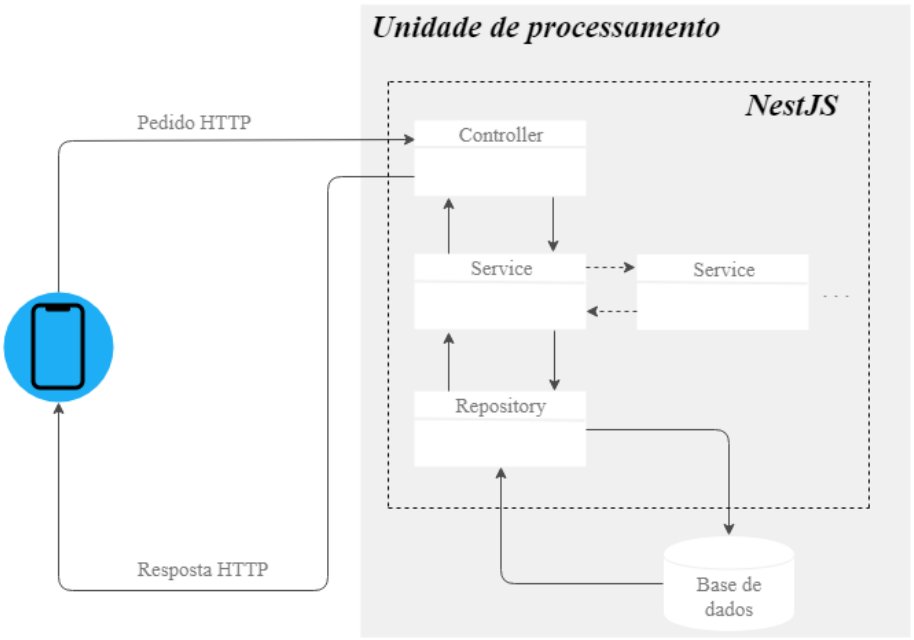


Figura 3.6: Funcionamento da *framework NestJS* num pedido HTTP

Dentro de um controlador, cada rota vem acompanhada com um decorador que indica que método HTTP se trata, de forma a que este execute diferentes ações. Por exemplo, um prefixo *products* combinado com o decorador *@GET('/category')* produz uma rota */products/category* do tipo GET. Por outro lado, o *Service*, traduzindo serviço, é responsável pelo levantamento e tratamento dos dados. Este pode importar outros serviços caso os dados solicitados pelo cliente impliquem tal.

O *Repository*, em português repositório, permite um encapsulamento da lógica de acesso a dados, contribuindo no isolamento da camada de acesso a dados. É no repositório que se trata de adquirir os dados da base de dados, sendo chamado pelos serviços com funções genéricas (como *find()*, *findOne()*, entre ou-

tras). Caso as funções não sejam suficientes para retornar os dados pretendidos, o serviço deverá invocar o repositório enviando os parâmetros necessários para que este recolha os dados mediante outras ferramentas, para além das funções.

Uma base de dados é um conjunto de informação que se relacionada entre si sobre um determinado tema ou domínio. Permite gerir grandes volumes de dados de modo a facilitar a organização, a manutenção e a pesquisa destes, bem como outros tipos de operações [45]. Neste Projeto, a base de dados irá armazenar todas as informações relacionados com os produtos, categorias, marcas, tal como os artigos em cada lista de compras.

3.6.2 TypeScript

TypeScript é uma linguagem de código aberto que se baseia em *JavaScript*, uma das ferramentas mais usadas do mundo, adicionando definições de tipo estático. Todo o código *JavaScript* válido também é válido em *TypeScript*, pois o código *TypeScript* é transformado em código *JavaScript* por meio de um compilador (*TypeScript* ou *Babel*).

Os *Types* (em português, tipos) fornecem uma forma de descrever um objeto, permitindo uma melhor documentação e permitindo que o *TypeScript* valide se o código está a funcionar corretamente. De um modo geral, esta linguagem economiza o tempo despendido ao detetar erros, sugerindo correções antes de executar o código. Pode ser interpretado em qualquer navegador, sistema operativo, ou melhor, em qualquer local que o *JavaScript* seja exequível, incluindo a plataforma *Node.JS* [46].

3.6.3 TypeORM

TypeORM é um *Object-Relational Mapping* (ORM) que pode ser executado nas plataformas *NodeJS*, *Browser*, *Cordova*, *PhoneGap*, *Ionic*, *React Native*, *NativeScript*, *Expo* e *Electron* e pode ser usado com *TypeScript* e *JavaScript* (ES5, ES6, ES7, ES8). O seu objetivo passa por oferecer suporte aos recursos *JavaScript* mais recentes e fornecer recursos adicionais que o ajudem a desenvolver qualquer tipo de aplicação que use base de dados, desde pequenas aplicações com algumas tabelas até aplicações corporativas de grande escala com várias bases de dados.

O *TypeORM* trabalha com entidades, colunas e repositórios para construir e gerir base de dados. Este suporta ainda *MySQL*, *MariaDB*, *Postgres*, *CockroachDB*, *SQLite*, entre outros [47].

3.6.4 Postman

Postman é uma ferramenta que dá suporte às solicitações feitas à API. Possui um ambiente ideal para execução de testes e pedidos em geral. Este *software*

torna mais eficiente trabalhar com API's, construindo solicitações de forma rápida e, ainda, guardá-las para uso posterior, além de conseguir analisar as respostas devolvidas. Através desta ferramenta é possível reduzir drasticamente o tempo necessário para testar e desenvolver API's [48].

Para fazer um pedido GET, tudo o que precisa ser feito é:

- Obter a conexão da rota na barra de endereços;
- Selecionar o método de resposta GET;
- Digitar a chave da API na seção "*Headers*";
- Especificar o formato de resposta, que poderá ser em JSON, por exemplo;
- Clicar em enviar;

Ao enviar, será obtido os dados de resposta em JSON acompanhado do código de *status 200* que confirma que a solicitação GET foi bem-sucedida.

Entre as principais vantagens em se usar o *Postman*, pode-se destacar:

- Possibilita qualquer tipo de chamada de API – REST, SOAP ou HTTP;
- Oferece suporte para formatos de dados populares, como *OpenAPI*, *GraphQL* e *RAML*;
- Fácil acessibilidade: Para usar o *Postman*, basta fazer *login*, facilitando o acesso a arquivos a qualquer momento, em qualquer lugar, desde que a ferramenta esteja instalada no computador;
- Uso de coleções: O *Postman* permite criar coleções para guardar diferentes solicitações de API;
- Elaborar testes: Pontos de verificação de teste, como a verificação do *status* de resposta HTTP, podem ser adicionados a cada pedido API, o que ajuda a garantir a cobertura do teste;

3.6.5 PostgreSQL

PostgreSQL é um poderoso sistema de base de dados *object-relational* de código aberto com mais de 30 anos de desenvolvimento que lhe rendeu uma forte reputação a nível de confiabilidade, robustez de recursos e desempenho.

Existe uma grande variedade de informações que descrevem como instalar e usar este sistema por meio de documentação oficial. A comunidade *PostgreSQL* oferece ainda muitas áreas para familiarizar com a tecnologia.

3.6.6 Android Studio

Android Studio é um *Integrated Development Environment* (IDE), em português ambiente de desenvolvimento integrado, criado para desenvolver plataformas *Android*, oferecendo todos os recursos necessários para criar as mais diversas aplicações. Foi desenvolvido pela Google para dispositivos móveis e é gratuito. Este está inserido neste Projeto, pois permite criar uma aplicação *Android* de forma a obter uma interface com o utilizador. Este *software* dispõe de duas linguagens distintas, *Java* e *Kotlin*, sendo que *Kotlin* é uma linguagem mais recente comparativamente com a linguagem *Java*. Contudo, para este Projeto foi utilizado o *Java*, pois apresenta mais suporte o que é fundamental para a resolução de pequenos problemas que poderão aparecer ao longo do desenvolvimento da aplicação. [49].

3.6.6.1 Estrutura de uma aplicação *Android*

O *Android Studio* é um *software* ideal para construir de raiz uma aplicação móvel, reunindo as características e ferramentas de apoio para a criação de aplicações para os dispositivos móveis *Android*. A sua estrutura é algo complexa, podendo ser dividida em vários módulos mais relevantes:

- ***AndroidManifest***

É um ficheiro XML que é responsável por apresentar informações essenciais e importantes sobre a aplicação ao sistema operativo *Android*. Essas informações são utilizadas para que o sistema operativo execute a aplicação com as suas devidas permissões e configurações.

Além disso, tem outras funções como:

- Configurar outros componentes/elementos da aplicação como *Activities*, *Services*, *Content Providers*, etc;
- Declarar permissões para que a aplicação funcione corretamente no dispositivo do utilizador (leitura de contatos, acesso à *internet*, acesso à câmara, entre outras);
- Apresentar a versão da aplicação, assim como a menor versão de *Android* suportada.

Estas são algumas das diversas funções deste ficheiro, sendo importante mencionar que todas as *Activities* devem ser adicionadas neste, pois não serão encontradas pelo sistema caso não tenham sido incluídas.

- ***Activities***

Activity, em português atividade, é um componente crucial para uma aplicação *Android* que normalmente está relacionada diretamente com uma interface, também conhecida como *layout*, e suas funcionalidades correspondentes, ou seja, apresenta toda a lógica por detrás de uma interface, respondendo a todas as interações que o utilizador realizar ao navegar nesta.

As atividades apresentam um ciclo de vida que é composto por vários estados internos, pois estas são criadas, iniciadas, pausadas, reiniciadas e destruídas ao longo da navegação pelo *layout* e pela aplicação. Para cada um dos seis estados existentes, existe um método de *callback* específico: *onCreate()*, *onStart()*, *onResume()*, *onPause()*, *onStop()* e *onDestroy()*. Conforme a atividade entra em um novo estado, o sistema invoca o método de *callback* correspondente. A Figura 3.7 demonstra a representação visual do comportamento do seu ciclo de vida.

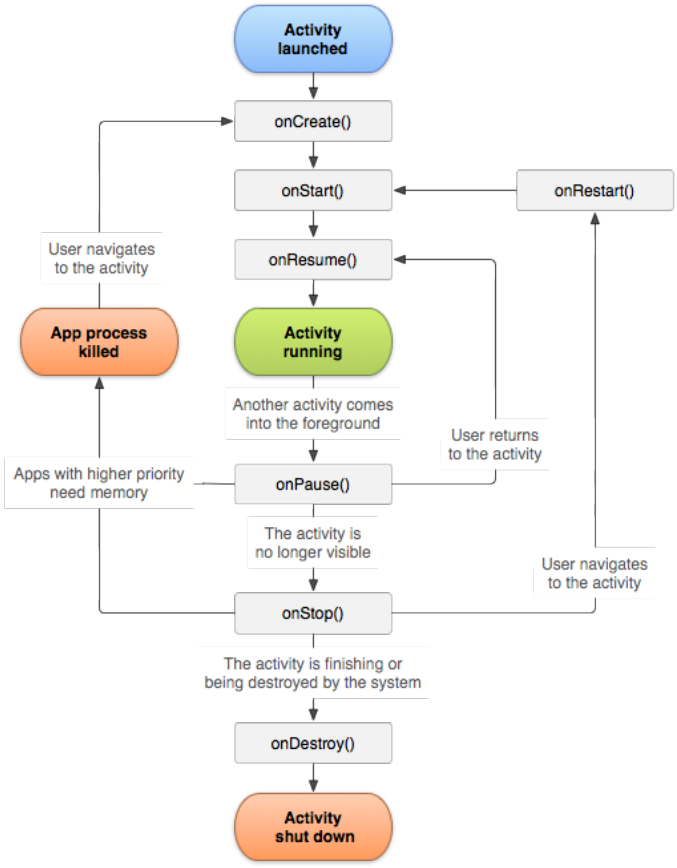


Figura 3.7: Ciclo de vida de uma atividade [15]

É importante ter em mente este ciclo no desenvolvimento de uma aplicação, uma vez que cada ponto/estado fornece uma oportunidade de realizar uma determinada ação. Ou seja, cada *callback* permite que seja realizado um procedimento específico adequado a determinada mudança de estado. Efetuar a ação acertada no momento apropriado e gerir as transições da maneira correta faz com que a aplicação seja mais robusta e apresente melhor desempenho.

- **Recursos**

Os recursos do projeto encontram-se numa pasta específica, onde cada tipo de recurso está localizado numa subpasta:

- *anim*: integra ficheiros XML que definem animações, como: transações entre interfaces e fragmentos, movimentos de botões, entre outros;
- *drawable*: contém os ficheiros Bitmap (.png, .jpg, .gif), i.e, todas as imagens utilizadas;
- *layout*: engloba todos os ficheiros XML que definem os *layouts* de interface com o utilizador;
- *menu*: abrange os ficheiros XML que definem os menus no projeto, como menus de opções, menus de contexto ou submenus.
- *values*: inclui ficheiros XML que contêm valores simples, como texto (aplicados nos *layouts*), cores, dimensões e "*styles*" que definem o tema do projeto, i.e., as cores padrões, entre outros.

- ***Gradle***

O *Android Studio* usa o *Gradle* que é um kit de ferramentas de compilação avançado para automatizar e gerir o processo de compilação, permitindo que seja definido configurações personalizadas e flexíveis. Este facilita a inclusão de binários externos ou outros módulos de biblioteca ao incluí-los como dependências no ficheiro *build.gradle*. Estas dependências podem estar localizadas na máquina pessoal do programador ou em um repositório remoto (p.e. GitHub).

Capítulo 4

Implementação

Após o estudo teórico bem como a estruturação que envolve os vários ramos do presente Projeto, neste capítulo será abordado a implementação do sistema desenvolvido. São descritos os esquemas elétricos bem como o *software* desenvolvido.

4.1 Instalação das dependências no Raspberry Pi

Objetivando o desenvolvimento de um sistema capaz de capturar, detetar/localizar e identificar códigos de barras e ainda enviar esses dados para um servidor recorreu-se à instalação de dependências como a biblioteca OpenCV, Zbar bem como a ferramenta de desenvolvimento Python na placa de desenvolvimento Raspberry Pi. Para tal, começou-se pela instalação do sistema operativo cuja função prende-se na gerência dos recursos do sistema, fornecendo uma interface entre o programador e o computador. Foi optado pelo sistema operativo Raspberry Pi OS, anteriormente denominado de Raspbian, sendo este o sistema operativo oficial da placa Raspberry Pi que se encontra num ficheiro zip no *website* oficial [50]. Com o auxílio do *software Imager*, este foi instalado dentro de um cartão SD que será posteriormente colocado no microcomputador.

Terminada a instalação do sistema operativo e após permitir o acesso ao Raspberry Pi através do *software* de acesso gráfico remoto VNC, procedeu-se à instalação da Livraria OpenCV. Inicialmente, foi instalado diversos pacotes essenciais para poder de seguida executar o OpenCV com sucesso. Com o auxílio do comando *sudo apt install*, foram instalados pacotes que contêm as ferramentas necessárias para compilar o OpenCV, tais como pacotes que adicionam suporte para diferentes formatos de imagem e vídeo, etc. Entre estes, foi instalado ainda a ferramenta Python.

Concluindo a instalação dos pacotes, foi realizado o *download* de dois ficheiros OpenCV disponíveis no repositório GitHub no *website* [51], que procura a versão mais recente disponível. Procede-se então à compilação da biblioteca recorrendo ao comando *cmake* que irá criar um *makefile*. Uma vez concluída a geração do ficheiro *make* prosseguiu-se finalmente à compilação executando o comando *make -j$(nproc)*. Quando o processo terminou, procedeu-se à instalação propriamente dita do OpenCV, através do comando *sudo make install*. Neste Projeto, a versão instalada foi a versão 4.4.0.

Depois de Python e OpenCV, efetuou-se a instalação da biblioteca ZBar, que tem como função ler e descodificar os códigos de barras em conjunto com o OpenCV. Mediante a execução do comando *sudo apt-get install libzbar0* e do comando *pip install pyzbar*, esta biblioteca foi instalada com sucesso.

4.2 Leitura de códigos de barras em tempo real

Concluídas as instalações das dependências no Raspberry Pi, procedeu-se a um pequeno ensaio de modo a verificar o correto funcionamento destas. O ensaio passou por identificar códigos de barras em imagens (Anexo A). Após certificar o correto funcionamento da biblioteca OpenCV e ZBar, iniciou-se o desenvolvimento da leitura de códigos de barras em tempo real.

O código desenvolvido para a deteção e identificação de códigos de barras encontra-se no Anexo B e, tal como se pode analisar neste, em apenas 66 linhas foi possível identificá-los em tempo real, bem como acionar o aviso sonoro e enviar o código lido para o servidor. É importante mencionar que o sistema consegue detetar mais do que um código num só *frame*/imagem. Com o intuito de percecionar o seu funcionamento é apresentado o seu fluxograma na Figura 4.1.

Primeiramente, importa-se os pacotes necessários, começando com o *VideoStream* que é utilizado para lidar com a captura de *frames*/imagens de forma eficiente e segmentada. De seguida, importa-se a livraria Zbar e ainda *imutils* que abrange um conjunto de funções básicas de processamento de imagem. Após importar todos os pacotes fundamentais, começa-se por iniciar a câmara de vídeo, esperando um período de dois segundos de modo a permitir que a câmara inicie.

Para que o sistema se encontre sempre pronto a ler produtos, é fulcral que este entre num ciclo infinito, tal como sucede na linha 30 do Anexo B. Ao entrar no ciclo, o sistema captura a imagem redimensionando-a para uma largura de 600 pixéis, que é uma dimensão que permite visualizar nitidamente a imagem

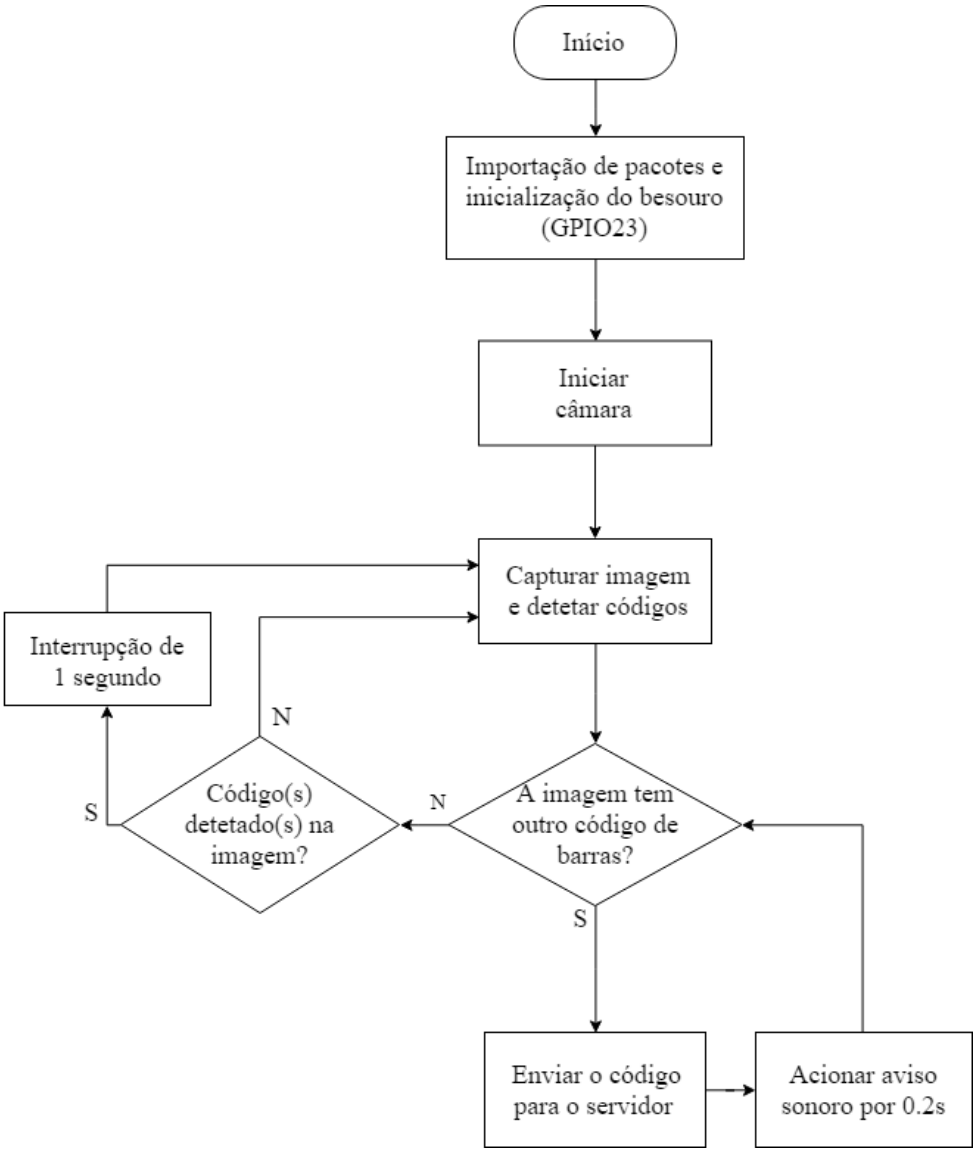


Figura 4.1: Fluxograma Sistema de captura de produtos

capturada pela câmara, que para a fase de testes foi essencial para perceber se os códigos de barras estavam a ser detetados pelo sistema.

De seguida, a imagem capturada será analisada com o auxílio da biblioteca ZBar, usando a função *decode()*. Ao detetar um ou mais, o sistema irá criar um retângulo delimitador em torno de cada código identificado na imagem, assim como o seu tipo e o código propriamente dito (linha 41 a 50).

A Figura 4.2 exemplifica como são identificados os códigos numa imagem captada pelo sistema.

Figura 4.2: Comportamento do algoritmo na deteção de um produto

O código de barras será então armazenado numa variável (*barcodeData*) que irá ser enviado para o servidor (linha 54) em JSON através do método HTTP POST. Para realizar o pedido ao servidor foi utilizado o *requests.post()* que provém da biblioteca *requests* importada no início do ficheiro Python. Este tem como argumentos o URL e os dados a enviar, sendo que o primeiro se encontra direcionado para o *localhost*, pois o servidor encontra-se na mesma unidade de processamento que interpreta as imagens e retira os códigos detetados destas.

Após encaminhar todos os códigos de barras que se encontram na imagem capturada para o servidor, o sistema é suspenso durante 1 segundo (linha 63), não permitindo detetar mais nenhuma imagem e, consequentemente, mais nenhum código nesse intervalo de tempo. Estes períodos são aplicados através do método *sleep()* da biblioteca *time* importada no início do documento, sendo que este suspende a execução por um determinado intervalo de tempo. Este segundo é aplicado sempre que é detetado um ou mais códigos na imagem capturada (linha 61), de forma que o sistema não capture os mesmos produtos diversas vezes seguidas quando se pretende que seja lido somente uma única vez. Com este intervalo, o utilizador terá tempo para se aperceber que os produtos foram lidos com sucesso através do aviso sonoro, indicando que estes podem ser colocados no lixo.

O ciclo infinito permite ao sistema estar constantemente à escuta de novos produtos, detetando-os e identificando-os de forma a atualizar a lista de compras do utilizador, indicando que produtos estão em falta na despensa.

4.3 Aviso Sonoro

No que concerne em alertar o utilizador que o produto foi lido com sucesso, optou-se por utilizar um besouro como aviso sonoro. O besouro escolhido tem uma tensão operacional de 3 a 28 Volts, de frequência de 4.6kHz e um consumo de 10mA. Este apresenta um oscilador interno que produz uma intensidade sonora de 90 decibéis [14].

De modo a ligá-lo foram utilizados os pinos de entrada e saída disponibilizados pelo Raspberry Pi (GPIO). Estas entradas e saídas digitais funcionam a 3.3 Volts, encontrando-se dentro da tensão de funcionamento do besouro. Estes também apresentam um consumo máximo de 16mA que é superior ao consumo do besouro (10mA), dando a estabilidade necessária para uma ligação Besouro – Raspberry Pi direta, tendo somente de escolher um pino dentro dos 26 disponíveis na placa. O GPIO escolhido para o efeito foi o GPIO23.

Na Figura 4.3 é possível visualizar a ligação do besouro ao *Raspberry Pi 4 Model B*.

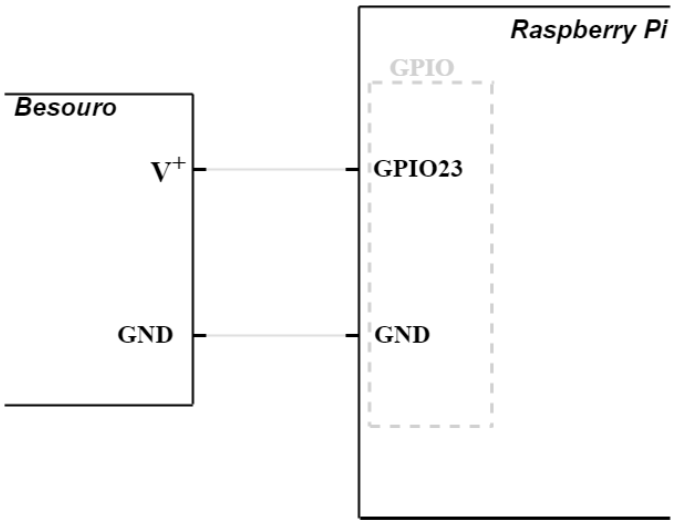


Figura 4.3: Esquema Elétrico da ligação Besouro - Raspberry Pi

Visando a ligação entre o besouro e a placa seguiu-se para o desenvolvimento do código que o ativa sempre que detetar um produto nas imagens capturadas pela câmara. Inicia-se por importar o pacote *RPi.GPIO* que fornece suporte para controlar as entradas e saídas GPIO do Raspberry Pi. Avançou-se então para a configuração do pino escolhido, começando por desativar os avisos (linha 15 do Anexo B) e estabelecer o modo de configuração da numeração dos GPIO's, sendo o escolhido o *GPIO.BCM* que se refere aos pinos pelo número

Broadcom. Ao definir o pino GPIO23 como pino de saída, este foi colocado com o nível lógico baixo (linha 21 do Anexo B), desativando o besouro. Ao detetar um código de barras, o GPIO23 é colocado a nível lógico alto, ativando-o por 0.2 segundos, sendo de seguida desativado após o respetivo período de tempo terminar.

Este processo ocorre sempre que é detetado um ou mais códigos de barras nas imagens capturadas pela câmara que, por sua vez, está ligada ao microcomputador.

4.4 Base de dados

Antes de partir para o desenvolvimento da aplicação móvel, é indispensável perceber que dados devem ser armazenados, para isso é importante compreender como estruturar a aplicação e identificar que elementos são fundamentais.

Os produtos devem apresentar informações como nome, marca, categoria e código de barras. A categoria escolhida para o produto deve pertencer a uma outra categoria principal, p.e., "Abóbora" deverá pertencer à categoria principal "Fruta & legumes", podendo ser denominada por subcategoria. O utilizador deverá poder criar listas nomeando-as, tal como adicionar, editar e remover produtos/subcategorias da lista de compras.

Os produtos lidos pelo sistema de captura de produtos devem de ser adicionados a uma lista específica que engloba todos os produtos lidos pelo sistema, que serão posteriormente atribuídos a uma lista.

Considerados todos os elementos que têm de ser armazenados, criou-se então uma base de dados em *TypeORM* juntamente com *TypeScript*. Atendendo aos elementos apresentados, foram criadas as várias tabelas com as respetivas colunas fundamentais ao Projeto. A Figura 4.4 apresenta as tabelas, respetivas colunas e ainda as interligações entre tabelas.

No total foram criadas seis tabelas, sendo que a tabela *product* armazena os dados correspondentes aos produtos, i.e., o nome, código de barras, marca e categoria. Tanto a marca como a categoria apresentam uma tabela em separado onde armazenam o nome, contudo a tabela *category* revela mais uma coluna denominada de *categoryId* que, caso se trate de uma subcategoria, recebe o valor do ID da categoria principal a que esta pertence.

A tabela *to_add_list* refere-se aos produtos lidos pelo sistema de captura de produtos que ainda não foram atribuídos a uma lista. A coluna *productId* indica o ID do produto detetado pelo sistema e a coluna *quantity* indica a quantidade de vezes que o respetivo produto foi capturado.

Relativamente à tabela *shopping_list_item*, esta armazena todos os produtos e categorias que se encontram associadas a uma lista de compras. Posto isto,

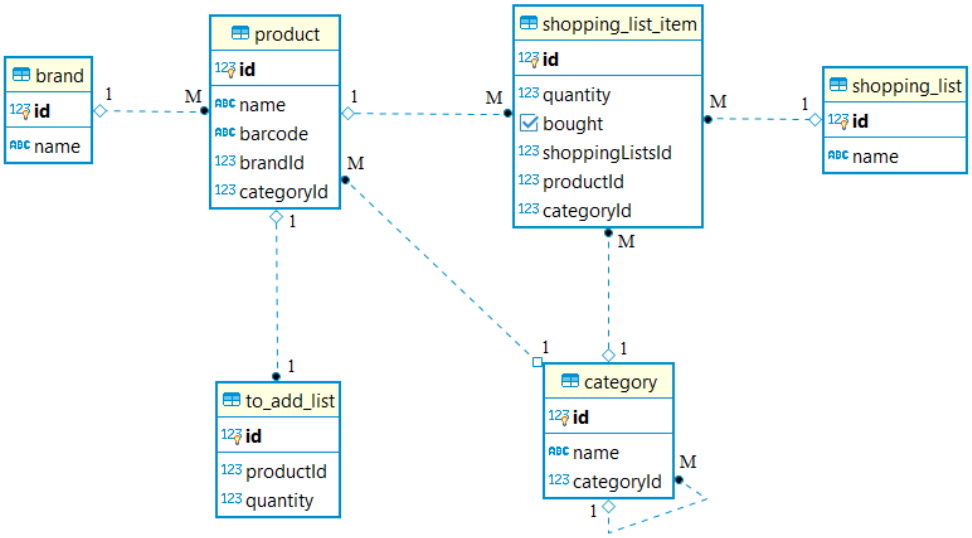


Figura 4.4: Esquema da Base de dados

somente uma das colunas, *productId* ou *categoryId*, pode conter o ID do produto/categoria correspondente. A coluna *quantity* indica a quantidade que é necessário comprar, enquanto que a coluna *bought* aponta se foi comprado ou não a totalidade do item. A *shopping_list_id* possui o ID da lista de compra que está associado, existindo igualmente uma tabela para as listas de compras de forma a armazenar o nome de cada uma delas.

De forma a interligar estas tabelas foi utilizada relações entre tabelas. Em *TypeORM*, existem vários tipos de relações: *One-to-One*, *Many-to-One*, *One-to-Many* e *Many-to-Many*, sendo que para esta base de dados foi somente recorrida à relação *One-to-One*, *One-to-Many* e *Many-to-One*.

Numa relação *One-to-One*, um registo de uma tabela só poderá ser associado a um único registo em uma outra tabela e vice-versa. Por exemplo, nesta base de dados um elemento da lista de produtos lidos pelo caixote do lixo (tabela *to-add-list*) só poderá estar associado a um só produto, assim como um produto só poderá se encontrar num só elemento da lista de produtos lidos pelo caixote.

Numa relação *One-to-Many*, o registo em uma tabela pode ser associado a um ou mais registos numa outra. A relação *Many-to-One* é o inverso da relação *One-to-Many*, sendo que, p.e., um produto só pode estar associado a uma única categoria, no entanto uma categoria pode estar ligada a vários produtos distintos. Dito isto, a categoria deverá adotar a relação *One-to-Many* em relação ao produto e o produto deverá aplicar a relação *Many-to-One* relativamente à categoria.

4.5 Aplicação *Never Forget*

A aplicação móvel desenvolvida, denominada de *Never Forget*, permite ao utilizador visualizar que produtos faltam na despensa e consequentemente que produtos devem de ser comprados. Esta interface gráfica é de extrema importância para que o utilizador no momento de compra adquire essencialmente os produtos que tem em falta, poupando tempo neste processo e eventualmente reduzir as idas ao supermercado.

A aplicação *Never Forget* foi desenvolvida através da plataforma de desenvolvimento *Android Studio*, sendo auxiliado pela documentação disponibilizada pelo *website* que o *software Android Studio* oferece aos seus utilizadores [49], tornando-se a principal fonte de informação. Para apoiar a elaboração da aplicação, foi muitas vezes recorrido ao *debug* disponibilizado na plataforma. O *debug* permite que seja percorrida cada linha de código, avaliando as variáveis, os métodos e o modo como o código se comporta, corrigindo rapidamente pequenos erros e *bugs*.

Apesar do sistema de captura de produtos, situado no caixote do lixo pessoal do utilizador, permitir muito facilmente determinar que produtos estão ausentes, existem produtos que não contêm códigos de barras (p.e. frutas e legumes) não sendo possível serem detetados pelo sistema. Esses produtos podem ser adicionados como subcategorias ou até mesmo como produtos caso estes apresentem uma marca e um nome. Todos os produtos, com ou sem código de barras, e subcategorias podem ser adicionados pelo utilizador através da aplicação, o que torna todo o processo de inserção de itens mais flexível.

Com a aplicação *Never Forget* instalada no *smartphone*, o utilizador consegue:

- Criar diversas listas de compras, proporcionando ao utilizador uma melhor organização;
- Criar subcategorias se, eventualmente, nenhuma das existentes identifiquem a classe de produtos que se pretende comprar. Por exemplo, caso não exista a subcategoria "Pasta de dentes", esta pode ser criada e associada à categoria principal "Higiene & Saúde", sendo que dentro desta subcategoria podem ser adicionados produtos;
- Criar produtos, sendo estes caracterizados pelo nome, marca, subcategoria que estão associados e o código de barras caso possua;
- Adicionar subcategorias e produtos às listas de compras criadas pelo utilizador;
- Visualizar os itens a comprar assim como as quantidades correspondentes;

- Consultar os produtos lidos pelo caixote do lixo, e posteriormente adicioná-los às listas que o utilizador considerar mais apropriadas;
- Indicar a quantidade adquirida no momento de compra, atualizando a lista;
- Capturar produtos através da câmara do *smartphone* lendo o seu código de barras;
- Pesquisar subcategorias segundo a categoria principal a que estão associadas ou através de um motor de pesquisa.

Delineadas as principais funcionalidades da aplicação móvel, é imprescindível abordar a função e importância que tem o servidor para o bom funcionamento desta aplicação. Como descrito no Capítulo 3, este gere e transmite informação, sendo que, neste caso, irá enviar todos os dados que serão visualizados pelo utilizador ao navegar pela aplicação. Este também irá receber elementos vindos tanto da aplicação como do sistema de captura de produtos, quando este detetar produtos.

Após determinar que dados guardar na base de dados é fundamental definir que pedidos devem ser solicitados pela aplicação ao servidor. Este deve estar então preparado para receber os pedidos e responder de forma simplificada para facilitar a interpretação dos dados pela aplicação.

Com as funcionalidades planificadas, é possível perceber que os métodos HTTP GET, POST, PATCH e DELETE são vitais para satisfazer os pontos projetados. O método GET solicita a representação de um recurso específico, devolvendo apenas dados, o que será muitas vezes aplicado para preencher campos nas diversas interfaces com o utilizador. O método POST é utilizado para submeter dados, adicionando elementos às tabelas da base de dados, enquanto que o método PATCH aplica modificações parciais em um recurso/elemento, atualizando os campos deste. Por fim, o método DELETE remove um recurso/elemento, utilizado, p.e., para remover um produto de uma lista de compras.

4.5.1 Solicitações HTTP ao servidor

No que concerne à comunicação entre a aplicação e o servidor recorreu-se à biblioteca *Volley* que é uma biblioteca HTTP que facilita a criação de redes para aplicações *Android*. Esta está disponível no GitHub ([52]) e pode ser adicionada ao projeto ao acrescentar a respetiva dependência ao *Gradle*.

A utilização desta biblioteca oferece os seguintes benefícios:

- Programação automática de solicitações de rede;

- Várias conexões de rede simultâneas;
- Compatibilidade com priorização de solicitações;
- Possibilidade de cancelamento de solicitações.

Estas são algumas das vantagens que a *Volley* proporciona, sendo que esta se destaca em operações do tipo "Chamada de procedimento remoto" (em inglês *Remote Procedure Calls* (RPC)) usadas para preencher um *layout*, como é pretendido em grande parte desta aplicação. Integra-se facilmente a qualquer protocolo e encontra-se preparada para responder a solicitações por meio de *strings*, imagens (Bitmap) ou até mesmo em JSON.

Para além de incluir a biblioteca ao projeto é necessário adicionar também a permissão para aceder à *internet* (*android.permission.INTERNET*) ao *AndroidManifest*. Sem esta permissão, a aplicação não se poderá conectar à rede.

A fim de realizar solicitações HTTP, foi aplicada a classe *RequestQueue* disponível na *Volley* que tem como função gerir linhas de execução de modo a executar as operações de rede essenciais à comunicação com o servidor, como ler e escrever no *cache* e analisar respostas. Segundo a documentação oficial do *software* [53], se a aplicação faz o uso constante de solicitações ao servidor é recomendado implementar um documento que engloba a *RequestQueue* e outros recursos do *Volley*, ou seja, um ficheiro que configure uma única vez o *RequestQueue*, evitando código repetido ao longo da aplicação. Na aplicação *Never Forget*, esse ficheiro, disponível em [53], foi denominado *VolleySingleton*.

Sempre que é iniciada uma atividade que necessite fazer pedidos ao servidor, é chamada a função *setup()* (do ficheiro *API.java*) que inicializa os recursos da biblioteca *Volley* (linha 24), permitindo que as respostas obtidas nos pedidos sejam direcionadas à atividade atual (Figura 4.5).

```
22  public static void setup(Context context, Callback callback) {
23      callbackClass = callback;
24      requestQueue = VolleySingleton.getInstance(context).
25              getRequestQueue();
26  }
```

Figura 4.5: Função *setup()* do ficheiro *API*

Foram criados mais quatro ficheiros em *Java*, cada um com um papel imprescindível para a comunicação entre a aplicação e o servidor. Estes são *CallbackTypes*, *Service*, *API* e *Callback* que serão descritos de seguida com o auxílio de excertos de código dos ficheiros:

- ***CallbackTypes***

O documento *CallbackTypes* contêm uma lista com valores do tipo *string* que representam cada pedido ao servidor, independentemente se se trata de uma solicitação GET, POST, PATCH ou DELETE. Por exemplo, para determinar se um certo código de barras se encontra associado a um produto é feito um pedido GET ao servidor, neste caso, o valor *CallbackTypes* atribuído a este pedido é "*GET_PRODUCT_BY_BARCODE*".

Estes *CallbackTypes* são muito úteis principalmente quando numa mesma atividade são realizados vários pedidos distintos, assim, quando a atividade receber as respostas, esta saberá encaminhá-las para as respetivas ações a realizar consoante o *CallbackTypes* a que a resposta está associada.

- ***Service***

O ficheiro *Service* é responsável por construir o URL e ainda associar um *CallbackType* a cada pedido. Maioritariamente, as atividades enviam informações específicas relativas ao pedido, de modo a que o URL seja preenchido com esses parâmetros para que a resposta devolva o pretendido. Quando se pretende criar ou atualizar um elemento na base de dados (PATCH e POST), este ficheiro tem a função de reencaminhar os dados a criar/alterar para o ficheiro API, assim como o URL e o *CallbackType*.

- ***API***

O ficheiro *API* é um dos mais importantes no que toca à interligação Aplicação – Servidor, pois é neste que são realizados os pedidos propriamente ditos. Como já foi referido, a biblioteca *Volley* está preparada para receber respostas do tipo JSON, sendo que oferece as seguintes classes baseadas na classe *JsonRequest*:

- *JsonArrayRequest*: devolve uma resposta do tipo *JSONArray*;
- *JsonObjectRequest*: devolve uma resposta do tipo *JSONObject*.

Baseando o desenvolvimento do código na documentação *Android* relativa à biblioteca [54], foi criado um ficheiro capaz de realizar qualquer pedido necessário à aplicação. Neste foi implementado uma função para cada método HTTP, como se pode averiguar na Figura 4.6.

Todas as funções recebem como argumento o URL em formato de *string* vindo do *Service*, assim como o *CallbackTypes* que identifica o tipo de pedido.

```
28  public static void get(String url, final CallbackTypes type, boolean multi) {
29      if (multi) {
30          JsonArrayRequest jsonRequest = new JsonArrayRequest(Request.Method.GET, url, jsonRequest: null, (response) → {
33                  createCallbackResponse(type, response);
34          }, (error) → { createCallbackResponse(type, erro); });
41          requestQueue.add(jsonRequest);
42      } else {
43          JsonObjectRequest jsonRequest = new JsonObjectRequest(Request.Method.GET, url, jsonRequest: null, (response) → {
46                  createCallbackResponse(type, response);
47          }, (error) → { createCallbackResponse(type, erro); });
54          requestQueue.add(jsonRequest);
55      }
56  }
57  public static void post(String url, JSONObject data, final CallbackTypes type) {...}
71  public static void patch(String url, JSONObject data, final CallbackTypes type){...}
85  public static void delete(String url, final CallbackTypes type){...}
```

Figura 4.6: Pedidos HTTP presentes no ficheiro *API*

O argumento *data*, nas funções *post()* e *patch()*, armazena informação a ser adicionada ou atualizada, respetivamente. O método GET apresenta um argumento *boolean* denominado *multi* que indica se se espera uma resposta do tipo JSON ou um *array* de JSON's. Este argumento recebe o valor *true* ou *false* conforme o pedido a efetuar, sendo este definido no *Service*.

Em todos os métodos HTTP foi aplicado uma das duas classes (*JsonObjectRequest* e *JsonArrayRequest*) para realizar solicitações ao servidor. Na classe são enviados o método HTTP correspondente à solicitação a efetuar, o URL em formato *string* e os dados a enviar em JSONObject. É aplicado ainda um *listener Response.Listener<JSONObject>()* na classe cujo método *onResponse(JSONObject response)* é chamado após a conclusão bem-sucedida da solicitação, armazenando neste o resultado do pedido em JSON na variável *response*. Por fim, é ainda empregue *Response.ErrorListener()* cujo *onErrorResponse(VolleyError e)* é chamado quando a solicitação for malsucedida. No fim de cada função e após construir a solicitação é necessário adicioná-la à variável *requestQueue*, definida na função *setup()*, com a função *add()*. Ao chamá-la, o *Volley* efetua uma linha de execução de processamento de *cache* e um *pool* de linhas de execução de envio de rede. Quando a solicitação é adicionada à fila, ela é capturada pela linha de execução do *cache* e passa por uma triagem:

- Se puder ser atendida no *cache*: a resposta será analisada na linha de execução correspondente, sendo posteriormente entregue na linha de execução principal;
- Se não puder ser atendida no *cache*: a solicitação será colocada em fila da rede. A primeira linha de execução de rede disponível recebe a solicitação da fila, realiza a transação HTTP, analisa a resposta, gravando-a no *cache* e de seguida transmite a resposta analisada de volta à linha de execução principal.

A Figura 4.7 ilustra a vida útil de uma solicitação e as diferentes etapas que esta percorre.

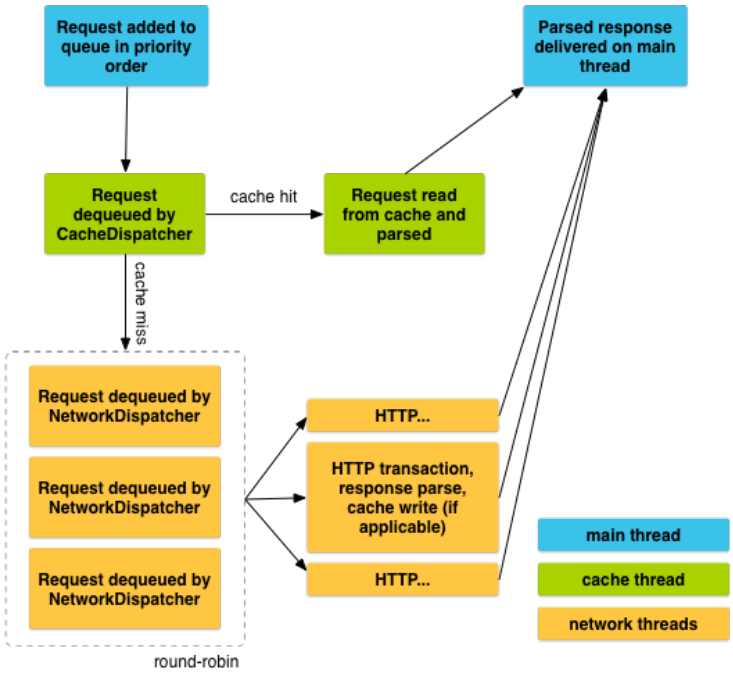


Figura 4.7: Ciclo de vida de uma solicitação [16]

Ao receber a resposta através da classe *JsonObjectRequest / JsonArrayRequest*, é chamada uma outra função denominada de *createCallbackResponse* que trata de devolver à atividade que fez o pedido o tipo de pedido (*CallbackTypes*) e a resposta obtida, como ilustra a Figura 4.8.

```
100  private static void createCallbackResponse(CallbackTypes type, Object jsonData){
101      JSONObject jsonObject = new JSONObject();
102      try {
103          jsonObject.put( name: "data", jsonData);
104      } catch (JSONException e) {
105          e.printStackTrace();
106      }
107      callbackClass.callback(type, jsonObject);
108  }
```

Figura 4.8: Função *createCallbackResponse* do ficheiro *API*

Esta função coloca o valor da resposta recebida (*jsonData*) na chave *"data"* (linha 103) no *jsonObject* criado previamente (linha 101), que será enviado como

parâmetro no método *callback* juntamente com o tipo de pedido efetuado (linha 107).

- ***Callback***

De modo a que a resposta seja devolvida à atividade que a invocou, foi criada uma interface denominada de *Callback*, que permite interligar o ficheiro API com todas as atividades que a implementarem (Figura 4.9).

```
5   public interface Callback {
6       void callback(CallbackTypes type, JSONObject data);
7   }
```

Figura 4.9: Interface *Callback*

Após implementar a interface *Callback* e o respetivo método à atividade, é possível definir que ações devem ser realizadas de acordo com o pedido realizado. A Figura 4.10 expõe um excerto do método *callback()* na *MainActivity* onde é possível verificar que consoante o tipo de pedido (*type*) a atuação perante a resposta (*data*) variará.

```
882     @Override
883     public void callback(CallbackTypes type, JSONObject data) {
884         switch (type) {
885             case GET_CATEGORY_BY_ID:
886                 try {...} catch (JSONException e) {
912                     e.printStackTrace();
913                 }
914                 break;
915             case GET_PRODUCT_BY_BARCODE:
916                 try {...} catch (JSONException e) {
956                     e.printStackTrace();
957                 }
958                 break;
```

Figura 4.10: Método *callback()* na *MainActivity*

Atendendo ao que foi explanado e a título de exemplo, a Figura 4.11 demostra como é feito um pedido HTTP pela aplicação. Neste caso é feito uma solicitação do tipo GET para determinar se o código de barras enviado está associado a algum produto da base de dados.

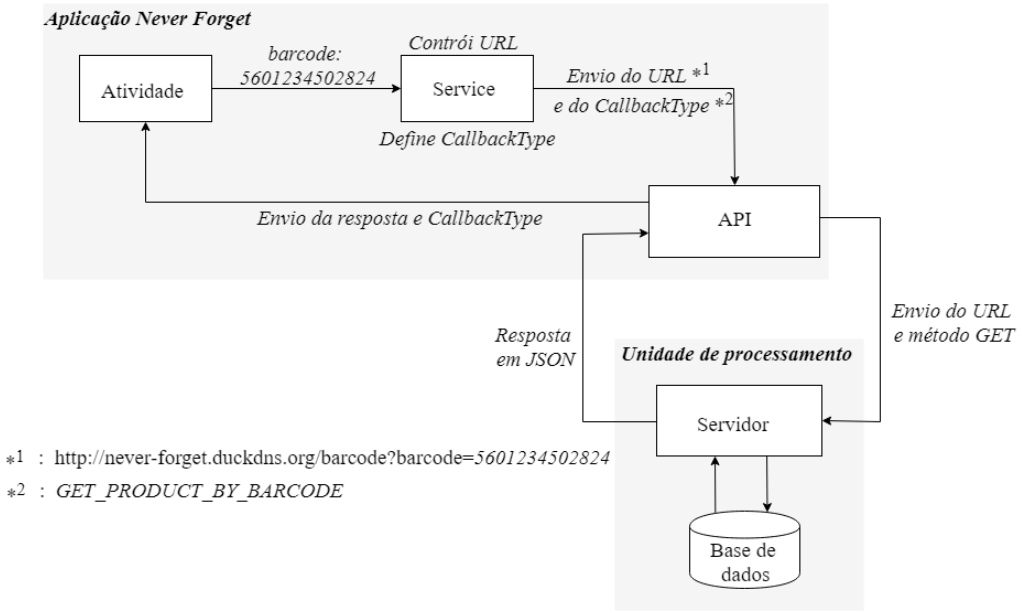


Figura 4.11: Demonstração de um pedido GET na aplicação

Na Figura 4.11 é feito um pedido GET para determinar se um certo código de barras se encontra associado a algum produto definido na base de dados. Inicialmente, a atividade faz o pedido enviando o código de barras (5601234502824) em uma *string* na solicitação em questão. Ao chegar ao *Service*, este constrói o URL com o código de barras recebido, pois este é aplicado no corpo do URL (*http://never-forget.duckdns.org/barcode?barcode=5601234502824*). De seguida, é definido o tipo de pedido (*GET_PRODUCT_BY_BARCODE*) que é enviado em conjunto com o URL para o ficheiro API, onde será realizada a solicitação propriamente dita. Neste pedido são enviados o URL e o método HTTP GET que, por sua vez, será recebido pelo servidor que irá recorrer à base de dados para verificar se existe um produto com o código transmitido. Caso exista, o servidor irá devolver o produto em JSON como resposta à solicitação realizada no API, no entanto caso não exista, será devolvido um erro a indicar que não existe nenhum produto com o código. O ficheiro API ao obter uma resposta reencaminha-a para a atividade juntamente com o *CallbackType* que permitirá à atividade direcionar a resposta para a ação que lhe compreende.

4.5.2 Leitura de códigos pela câmara do *smartphone*

Como já foi mencionado, o utilizador pode inserir produtos por meio da câmara do *smartphone*. Existem várias bibliotecas destinadas à leitura de códigos via câmara, no entanto foi optado pela biblioteca *Zxing* que é bastante com-

pleta para o efeito. Esta é muito utilizada pela comunidade *Android* e, por isso mesmo, encontra-se em constante atualização, proporcionando bom suporte aos seus utilizadores. O repositório desta biblioteca *open-source* pode ser encontrado no GitHub [55], que, conforme indica o próprio repositório, é necessário adicionar a biblioteca às dependências do projeto. Após sincronizar o ficheiro *Gradle*, a instalação da biblioteca fica concluída e pronta para ser utilizada. Como a câmara será recorrida para a leitura de códigos, a permissão deste *hardware* (*android.permission.CAMERA*) também deverá ser acrescentada ao ficheiro *AndroidManifest*.

Concluída a inserção das dependências e permissões necessárias, foi criada uma nova atividade denominada *CaptureAct*. Esta será a atividade que irá capturar os códigos através da câmara e como qualquer outra atividade, esta deverá ser inserida no *AndroidManifest*. Na Figura 4.12 é possível visualizar como se encontra definida a atividade no ficheiro.

```
36    <activity android:name=".CaptureAct"
37        android:theme="@android:style/Theme.DeviceDefault.NoActionBar.Fullscreen"
38        android:screenOrientation="portrait"
39        android:stateNotNeeded="true"
40        android:windowSoftInputMode="stateAlwaysHidden">
41    </activity>
```

Figura 4.12: Definição da *CaptureAct* no *AndroidManifest*

Ao analisar a Figura 4.12, é possível verificar que a atividade apresenta mais quatro parâmetros para além do *name* que é obrigatório quando é definida uma atividade neste documento:

- *theme* (linha 37): define o tema geral da atividade. Foi optado pelo *NoActionBar.Fullscreen* que coloca o *layout* a preencher todo o ecrã, tornando a interface e a leitura dos códigos mais limpa;
- *screenOrientation* (linha 38): estabelece a orientação da interface referente à atividade, que, neste caso, assume o valor *portrait* que coloca o *layout* na orientação de retrato;
- *stateNotNeeded* (linha 39): ao receber o valor *true*, esta configuração garante que a atividade quando é eliminada e reinicializada não necessita de guardar o estado desta, reiniciando a atividade como pela primeira vez;
- *windowSoftInputMode* (linha 40): a definição deste atributo afeta o estado do teclado de *software* (se está oculto ou visível) e o ajuste feito à interface

(se é redimensionada de modo a criar espaço para o teclado ou se o conteúdo do *layout* se desloca para tornar visível o foco atual quando parte é coberto pelo teclado). Ao definir o valor *stateAlwaysHidden*, o teclado estará sempre oculto nesta atividade.

Após definir a atividade, esta foi empregue na atividade *MainActivity* de forma a iniciá-la e, por sua vez, detetar os produtos através da câmara. Na *MainActivity* foi utilizado então a *IntentIntegrator* que é uma classe particular da biblioteca *Zxing* que, depois de customizada e associada à atividade *CaptureAct*, irá desencadear a inicialização da actividade e, consequentemente, a leitura dos códigos. A Figura 4.13 mostra como está configurada o *IntentIntegrator* na *MainActivity* e como são obtidos e tratados os resultados da leitura da atividade *CaptureAct*.

```
594   private void scanCode(){
595       Variables.flag_scan = true;
596       IntentIntegrator integrator = new IntentIntegrator( activity: this);
597       integrator.setCaptureActivity(CaptureAct.class);
598       integrator.setOrientationLocked(true);
599       integrator.setDesiredBarcodeFormats(IntentIntegrator.PRODUCT_CODE_TYPES);
600       integrator.initiateScan();
601   }
602
603   @Override
604   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
605       IntentResult result = IntentIntegrator.parseActivityResult(requestCode, resultCode, data);
606       if (result != null){
607           if (result.getContents() != null) {
608               barcode = result.getContents();
609               getProductByBarcode(barcode);
610           }else{
611               Toast.makeText( context: this,  text: "No results", Toast.LENGTH_SHORT).show();
612           }
613       }
614       super.onActivityResult(requestCode, resultCode, data);
615   }
```

Figura 4.13: Código de inicialização da *CaptureAct* e tratamento do resultado obtido

A função *scanCode()* é invocada quando o utilizador pretende ler um produto com a câmara do *smartphone*. Esta cria um *IntentIntegrator* e estabelece a *CaptureAct* como a atividade de irá realizar a captura de produtos (linha 597). Nesta é colocada ainda a orientação do *layout* bloqueada (linha 598) e definido que formatos de códigos é que a atividade pode capturar (linha 599), recebendo o valor *"PRODUCT_CODE_TYPES"* que engloba os códigos UPC-A, UPC-E, EAN-8, EAN-13 e RSS 14. Por fim, é iniciado a atividade (linha 600) bem como a leitura e ao detetar um código, a *CaptureAct* é destruída, devolvendo o valor do código

à *MainActivity* que será posteriormente interpretado pelo método *onActivityResult()*. Caso seja lido um produto, o código de barras deste será guardado na variável *barcode* do tipo *string* (linha 608).

De seguida é verificado através de um pedido GET ao servidor (linha 609) se o produto a que o código corresponde já se encontra na base de dados, pois caso seja identificado, este pode ser inserido diretamente à lista atual, contudo se não for reconhecido, o produto deve ser adicionado e posteriormente acrescentado à lista de compras.

4.5.3 Barra de pesquisa

A barra de pesquisa permite ao utilizador pesquisar uma subcategoria através da categoria principal a que está associada ou pelo nome desta, tal como foi delineado nas funcionalidades da aplicação. A nível de *layout*, a barra trata-se de um *widget*[1] *EditText* que não é mais que um campo de texto editável para a entrada de dados. Este ao ser selecionado, ou seja, pressionado, exibe uma lista das categorias principais existentes que ao serem selecionadas apresentam as subcategorias correspondentes (Figura 4.14 a)). No entanto, o utilizador ao pesquisar pelo nome da subcategoria será exposto a todos os resultados que coincidem com a busca, em forma de lista (Figura 4.14 b)).

Figura 4.14: Barra de pesquisa: por categoria principal *a)* e pelo nome *b)*

De forma a determinar o que exibir na interface foi implementada o método *setOnFocusChangeListener* ao *EditText* que é acionado sempre que haja uma mudança de estado quanto à sua seleção ou foco, proporcionando um controlo

[1] *widget*: é um elemento de interação numa interface gráfica, como, p.e., um botão.

em relação ao que apresentar nesta. A pesquisa por categoria principal é realizada através de um outro método denominado *setOnGroupClickListener* que é invocado sempre que é pressionada uma das categorias principais. Quando a categoria selecionada não se encontra expandida, i.e., a exibir as subcategorias, é realizado um pedido GET ao servidor que devolverá todas as subcategorias que estão associadas ao ID da categoria principal selecionada. Ao receber a resposta, estas serão colocadas em formato de lista abaixo da categoria principal a que estão relacionadas, tal como se pode visualizar na Figura 4.14 a).

Quanto à pesquisa mediante o nome, foi implementado o método *addTextChangedListener* que é acionado sempre que o texto no *EditText* for alterado, permitindo que seja realizado um pedido GET em cada modificação realizada, de forma a que seja devolvido as subcategorias correspondentes à pesquisa.

4.5.4 Listas

Ao longo do desenvolvimento da aplicação, foi necessário o uso de várias listas de modo a apresentar informações ao utilizador, como a lista de compras, entre outros. *ListView*, *RecyclerView* e *ExpandableView* foram os *widgets* que melhor se adaptaram à exibição de dados em formato de lista na aplicação.

Nas aplicações *Android*, as listas permitem tratar eventos (de clique, de seleção, entre outros), mas podem ser utilizadas somente para exibir dados (neste caso, não é necessário o tratamento de eventos). Contudo, todas as listas criadas na aplicação *Never Forget* possuem vários eventos que proporcionam mais interatividade, tornando-a mais intuitiva às suas funcionalidades.

A *ListView*, *RecyclerView* e *ExpandableView* são apenas um componente visual que irá conter a lista, no entanto são necessários mais alguns elementos essenciais para que a lista mostre a informação necessária e responda aos eventos pretendidos. A Figura 4.15 mostra um esquema onde são apresentados os componentes e a ligação entre eles. Atendendo a esta, segue uma breve descrição de cada um dos componentes:

- *Data List*: é a lista que armazena todos os dados a apresentar na interface;
- *View*: é a referência para a *view* (*layout*) que não é mais que o visual de cada linha da lista, que será replicado para todos os elementos;
- *Adapter*: é a classe responsável por associar cada elemento da lista de conteúdo (*Data list*) a uma *view*, servindo de ponte entre a *widget* e os dados a visualizar. Nesta classe é possível também definir o tratamento de eventos, ou seja, estabelecer ações para cada evento que seja realizado a uma *view* da lista;
- *Widget*: componente visual que permite observar a lista na interface.

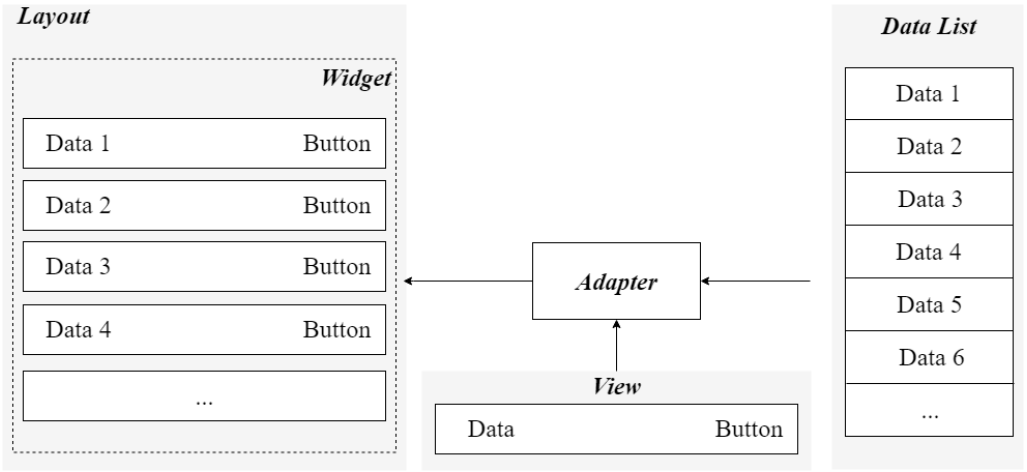


Figura 4.15: Esquema da interligação dos componentes que compõem uma lista

Os métodos implementados no *adapter* diferenciam consoante os *widget's* a aplicar, sendo estes os que permitem delinear como será apresentada a lista e que eventos terá.

A título de exemplo, ir-se-á analisar a lista utilizada para a pesquisa de subcategorias pelo seu nome, pois esta, como muitas outras listas na aplicação, utiliza o *widget RecyclerView*. Depois de acrescentar a biblioteca correspondente ao *widget* e de o adicionar ao *layout* correspondente à atividade *MainActivity*, foi criado um outro *layout* que é usado para cada linha da lista. Este é exibido em cada linha da Figura 4.14 b) e apesar deste apresentar apenas *TextView's*, que têm como objetivo exibir texto, o "+" e "-" funcionam como botões que irão atuar no *TextView* que se encontra entre eles, definindo a quantidade a adicionar à lista de compras.

No desenvolvimento do *adapter*, denominado *MyCategoryAdapter*, foi necessário expandir a classe *RecyclerView.Adapter<ViewHolder>* e implementar os métodos nele obrigatório:

- *onCreateViewHolder()*: método que define o *layout* que é usado para exibir o conteúdo em cada linha, devolvendo-o como *view* à função *MyViewHolder()* (que descreve um *layout* e armazena informações relativas à sua posição no *RecyclerView*);
- *onBindViewHolder()*: método que coloca os dados relativos à posição recebida nos respetivos campos do *layout*;
- *getItemCount()*: método que devolve quantos itens há na lista a exibir.

É importante mencionar que, inicialmente, os dois primeiros métodos não são chamados para todos os itens da lista, eles são chamados apenas para os itens que são visíveis ao utilizador. São chamados novamente quando o utilizador subir/descer na lista, associando a *view* ao conteúdo da posição que será agora visível.

Para que seja possível enviar os dados da lista que se pretende expor para o *adapter* é necessário criar uma função, para além dos métodos que já estão incluídos. Neste caso específico, a função denominada *MyCategoryAdapter*, tal como o *adapter*, recebe como argumentos o contexto que armazena a atividade relativa ao *layout* que possui o *RecyclerView* (*MainActivity*), a lista em *JSONArray* e uma interface chamada *CategoryCallback*, que possui três funções diferentes, sendo que cada uma corresponde a um evento diferente, como se pode averiguar na Figura 4.16.

```
19      JSONArray list;
20      Context context;
21      CategoryCallback callback;
22
23      public MyCategoryAdapter(Context ct, JSONArray list, CategoryCallback callback){
24          this.context = ct;
25          this.list = list;
26          this.callback = callback;
27      }
```

Figura 4.16: Função *MyCategoryAdapter* no *adapter MyCategoryAdapter*

Foram criados três eventos para interagir com os diversos elementos da lista, sendo que estes permitem aumentar (ao pressionar "+") ou diminuir (ao pressionar "-") a quantidade do elemento a ser adicionado à lista. Possibilita ainda apresentar todos os produtos que estão associados à subcategoria (ao pressionar no *layout* da linha correspondente).

A interface *CategoryCallback* declara uma função/método para cada evento, como demonstrado na Figura 4.17.

```
5   public interface CategoryCallback {
6       void onCategoryClick(int pos, TextView category, boolean category_recycler);
7       void onCategoryAdd(int pos, boolean recycler);
8       void onCategoryRemove(int pos, boolean recycler);
9   }
```

Figura 4.17: Interface *CategoryCallback*

A função *MyViewHolder()* no *adapter*, que apesar de não ser incluída nos métodos exigidos da classe *RecyclerView.Adapter<ViewHolder>*, apresenta um papel importante para criar eventos em cada linha da lista. Esta, juntamente com a interface *CategoryCallback*, permite interligar o *adapter* e a atividade, como se pode averiguar nas linhas 66, 72 e 82 da Figura 4.18.

```
63  itemView.setOnClickListener((v) → {
66          callback.onCategoryClick(getAdapterPosition(), myText1,  category_recycler: true);
67  });
69  myButton1.setOnClickListener((v) → {
72          callback.onCategoryAdd(getAdapterPosition(),  recycler: true);
73          try { myText2.setText(String.valueOf(list.getJSONObject(getAdapterPosition()).getInt( name: "quantity")));
74          } catch (JSONException e) {...}
77  });
79  myButton2.setOnClickListener((v) → {
82          callback.onCategoryRemove(getAdapterPosition(), recycler: true);
83          try { myText2.setText(String.valueOf(list.getJSONObject(getAdapterPosition()).getInt( name: "quantity")));
84          } catch (JSONException e) {...}
87  });
```

Figura 4.18: Tratamento dos eventos no *adapter CategoryCallback*

Depois de implementar a interface na *MainActivity*, procedeu-se ao tratamento propriamente dito dos eventos. Ao pressionar no *layout* de um elemento da lista (linha 63 da Figura 4.18) é invocado o método *onCategoryClick* na *MainActivity* (linha 619 da Figura 4.19) que fará exibir a lista de produtos associados à subcategoria selecionada.

Ao pressionar no botão "+" e "-" (linha 69 e 79 da Figura 4.18) serão invocados *onCategoryAdd* e *onCategoryRemove* respetivamente, que irão alterar a quantidade de itens a adicionar à lista (linha 639 e 681 da Figura 4.19).

```
618     @Override
619     public void onCategoryClick(int pos, TextView category, boolean category_recycler) {...}
637
638     @Override
639     public void onCategoryAdd(int pos, boolean recycler) {...}
679
680     @Override
681     public void onCategoryRemove(int pos, boolean recycler) {...}
```

Figura 4.19: Tratamento dos eventos na *MainActivity*

A estrutura dos *adapters* associados aos outros *widgets* não apresentam diferenças significativas comparativamente ao *adapter* explanado acima, seguindo um raciocínio bastante semelhante ao exposto neste exemplo prático aplicado no desenvolvimento da aplicação *Never Forget*.

4.6 Servidor

Ao construir a base de dados e projetar que funcionalidades deve apresentar a aplicação a desenvolver, passou-se à criação das solicitações HTTP necessárias para responder com sucesso ao que é pretendido.

Ao longo da navegação pela aplicação *Never Forget*, existem alguns pedidos mais frequentes que outros, no entanto todos são igualmente importantes para o bom funcionamento da aplicação, proporcionando todas as condições para que o utilizador tenha uma experiência agradável ao usufruir desta. Alguns desses métodos serão aprofundados nesta seção, dando a conhecer como funciona a estrutura da *framework* e a relação entre os diferentes documentos (controlador, serviço e repositório). Documentos esses que possibilitam a deteção do pedido, a recolha dos dados da base de dados e o envio desses dados ao cliente.

4.6.1 getCategoriesByName()

Este pedido GET tem como objetivo procurar categorias segundo o seu nome, reunindo as características apresentadas nos parâmetros enviados, sendo estes que irão delinear o que se pretende que seja devolvido. A Figura 4.20 mostra como está estruturado o pedido no controlador do servidor.

```
29   @Get()
30   async getCategoriesByName(
31     @Query('name') name: string,
32     @Query('categoriesFather', ParseBoolPipe) categoriesFather: boolean,
33     @Query('organized', ParseBoolPipe) organized: boolean,
34   ): Promise<Category[]> {
35     const teste = await this.categoriesService.getCategoriesByName(
36       name,
37       categoriesFather,
38       organized,
39     );
40     return teste;
41   }
```

Figura 4.20: Pedido GET *getCategoriesByName* no Controlador

O decorador *@Query* indica que o parâmetro será enviado através do URL utilizado para realizar o pedido GET. O parâmetro "*name*" do tipo *string* (linha 31) recebe o nome a procurar, podendo se tratar somente do início do nome da categoria, porém o segundo parâmetro, "*categoriesFatherOnly*" do tipo *boolean* (linha 32), indica se a procura se dirige apenas a categorias principais ou a sub-categorias, recebendo o valor *true* ou *false*, respetivamente. O último parâmetro "*organized*" do tipo *boolean* (linha 33) permite que as categorias que provêm da

resposta ao pedido se encontrem por ordem alfabética, recebendo o valor *true* caso se tencione tal.

```
6   async getCategoriesByName(
7     name: string,
8     categoriesFatherOnly: boolean,
9     organized: boolean,
10  ): Promise<Category[]> {
11    const queryBuilder = this.createQueryBuilder('category');
12
13    const queryInput = { name: `${name}%` };
14    queryBuilder.where('category.name ILIKE :name', queryInput);
15
16    if (categoriesFatherOnly) {
17      queryBuilder.andWhere('category.categoryId IS NULL');
18    } else {
19      queryBuilder.andWhere('category.categoryId IS NOT NULL');
20    }
21
22    if (organized) {
23      return await queryBuilder.orderBy('name', 'ASC').getMany();
24    } else {
25      return await queryBuilder.getMany();
26    }
27  }
```

Figura 4.21: Pedido GET *getCategoriesByName* no Repositório

Após o servidor receber o pedido, este será reencaminhado do controlador para o serviço, e de seguida para o repositório, onde os parâmetros recebidos serão aplicados para a recolha dos dados (Figura 4.21). Neste inicia-se por criar um *QueryBuilder* que é um recurso muito utilizado do *TypeORM* que permite criar consultas SQL usando uma sintaxe mais prática e conveniente. Na linha 11 criou-se então um *QueryBuilder* associado à tabela "*category*" (*queryBuilder*).

Na linha 14, a cláusula "*where*" irá filtrar os registos da tabela "*category*", ou seja, extrairá apenas os elementos que atendem à condição que se encontra dentro desta. O operador lógico "*ILIKE*" permite ignorar maiúsculas e minúsculas na consulta, o que não é um fator importante para a pesquisa de categorias na aplicação. O "*$name*" permite injetar o argumento *name* na *string* criada na linha 13 que será posteriormente adotada na consulta efetuada na linha seguinte. O "%" utilizado na linha 13 que, por sua vez, será aplicado na filtragem realizada na linha 14, representa qualquer caractere ou conjunto de carateres. Dito isto, a condição apresentada na linha 14 pesquisa todas as categorias em que o nome inicie com o parâmetro *name* enviado no pedido. Por exemplo, se este argumento receber o valor "ba" os resultados serão "Bananas", "Bacalhau", en-

tre outros, que sem o "%" seriam devolvidas as categorias "Abacate", "Robalo" juntamente com todas as outras que apresentassem "ba" algures no seu nome.

De seguida é realizada a verificação e seleção das categorias principais (linha 17) ou subcategorias (linha 19), conforme o valor de *categoriesFatherOnly*. Esta condição será acrescentada à anterior (linha 14), sendo que as principais apresentam a coluna *categoryId* a nulo (*null*).

Por fim, é retornado o conjunto de categorias que resultaram da filtragem efetuada previamente, no entanto de acordo com o valor do argumento *organized*, os resultados poderão ser dispostos alfabeticamente. Com o auxílio da cláusula "*orderBy*", que ao receber "*name*" como primeiro argumento e "*ASC*" de ascendente como segundo, as categorias a enviar são organizadas por ordem alfabética (linha 23).

Ao longo da aplicação, esta solicitação é empregue em diversas circunstâncias, como a pesquisa de subcategorias, a criação de uma nova categoria e até mesmo de um novo produto, entre outros.

4.6.2 getCategoriesById()

Este pedido devolve a categoria cujo ID corresponde ao enviado para o servidor (Figura 4.22). O parâmetro "*subCategories*" (linha 24) indica se se pretende que retorne as subcategorias associadas à categoria com o correspondente ID enviado (linha 23), i.e, se se tratar de uma categoria principal este devolverá as subcategorias associadas a esta. Este método é aplicado principalmente quando se expende cada uma das categorias principais na procura de uma subcategoria.

```
21  @Get(':id')
22  getCategoryById(
23    @Param('id', ParseIntPipe) id: number,
24    @Query('subCategories', ParseBoolPipe) subCategories: boolean,
25  ): Promise<Category> {
26    return this.categoriesService.getCategoryById(id, subCategories);
27  }
```

Figura 4.22: Pedido GET *getCategoriesById* no Controlador

4.6.3 getProductByBarcode()

É uma solicitação GET que devolve o produto (nome, marca, subcategoria e código de barras) que tem como código de barras o enviado no pedido (Figura 4.23). Caso não exista, é retornado um JSON vazio. Para que isto aconteça é aplicada a função *findOne()* do repositório enviando o código de barras como

argumento (linha 45). Este pedido é empregue somente quando o utilizador captura produtos com a câmara do *smarthphone*.

```
44  async getProductByBarcode(barcode: string): Promise<Product> {
45    const found = await this.productRepository.findOne({ barcode });
46    if (!found) {
47      throw new NotFoundException(
48        `Product with barcode "${barcode}" not found`,
49      );
50    }
51    return found;
52  }
```

Figura 4.23: Pedido GET *getProductByBarcode* no Serviço

4.6.4 getShoppingListById()

Ao enviar o ID da lista de compras, o servidor devolve a lista incluindo o seu nome e os itens comprados e que faltam comprar assim como as quantidades correspondentes, sendo que os itens podem se tratar de produtos como de subcategorias. Este pedido é executado sempre que se pretende exibir os itens da lista, assim como as suas quantidades a comprar.

4.6.5 addItemToShoppingList()

É a solicitação POST que adiciona um produto ou subcategoria a uma lista de compras (Figura 4.24). Ao enviar o ID da lista, o ID do produto/subcategoria e a quantidade a inserir, o servidor adiciona o elemento na lista e de seguida devolve a lista atualizada.

O decorador *@Body* indica que o ID do produto ou subcategoria e a quantidade a inserir vêm no corpo do pedido que, por sua vez, guarda estes dois dados dentro de um *Data Transfer Object* (DTO) denominado *AddItemShoppingDto* (linha 42). Um DTO é um objeto que define os dados enviados assim como algumas condições específicas, como, p.e, a quantidade deverá ser um número e caso não o seja deverá ser devolvida uma mensagem de erro ao cliente a informar a incorreta inserção da quantidade.

Sempre que o utilizador pretende adicionar um produto ou categoria nova a uma lista, esta será a solicitação realizada ao servidor, sendo que poderá provir de diferentes áreas na aplicação.

```
39  @Post(':id/add-item')
40  addItemToShoppingList(
41    @Param('id', ParseIntPipe) id: number,
42    @Body() addItemShoppingListDto: AddItemShoppingListDto,
43  ): Promise<ShoppingList> {
44    return this.shoppingListsService.addItemToShoppingList(
45      id,
46      addItemShoppingListDto,
47    );
48  }
```

Figura 4.24: Pedido POST *addItemToShoppingList* no Controlador

4.6.6 updateItemQuantity()

Atualiza a quantidade de um item em específico existente na lista, ao enviar o ID da lista de compras, o ID do item e a quantidade a subtrair ou a adicionar (Figura 4.25). Este método PATCH retorna a lista de compras atualizada.

```
51  async updateItemQuantity(
52    id: number,
53    idItem: number,
54    quantity: number,
55  ): Promise<ShoppingList> {
56    const shoppingList = await this.getShoppingListById(id);
57    const item = shoppingList.shoppingListItems.find(
58      (shoppingListItems: ShoppingListItem) => shoppingListItems.id == idItem,
59    );
60    if (item !== undefined) {
61      item.quantity += quantity;
62      item.bought = item.quantity === 0;
63      await item.save();
64    } else {
65      throw new NotFoundException(`Item with ID "${idItem}" not found`);
66    }
67    return shoppingList;
68  }
```

Figura 4.25: Pedido PATCH *updateItemQuantity* no Repositório

Inicialmente começa-se por obter a lista de compras por completo (linha 56) e com esta adquirir o item que se pretende alterar a quantidade (linha 57). Ao obtê-lo, a quantidade enviada será adicionada à atual do respetivo elemento (linha 61), sendo que a quantia enviada poderá ser negativa caso se pretenda reduzir o montante a comprar. Caso ao atualizar a quantidade, esta for nula significa que o item foi comprado na totalidade (linha 63).

4.6.7 deleteItemFromShoppingList()

Este pedido DELETE remove um item (produto ou subcategoria) de uma lista de compras, ao enviar o ID da lista e do item a ser eliminado. A Figura 4.26 mostra o procedimento aplicado no repositório para esta solicitação.

```
70  async deleteItemFromShoppingList(id: number, idItem: number): Promise<void> {
71    const shoppingList = await this.getShoppingListById(id);
72    const item = shoppingList.shoppingListItems.find(
73      (shoppingListItems: ShoppingListItem) => shoppingListItems.id === idItem,
74    );
75    if (item !== undefined) {
76      item.remove();
77    } else {
78      throw new NotFoundException(`Item with ID "${idItem}" not found`);
79    }
80  }
```

Figura 4.26: Pedido DELETE *deleteItemFromShoppingList* no Repositório

4.6.8 addOrCreateProductByBarcode()

Este pedido POST é responsável por interligar o caixote do lixo pessoal do consumidor com o servidor. Ao detetar um produto no seu campo de visão, o caixote envia o código detetado através deste pedido HTTP ao servidor (linha 54 do Anexo B). Ao receber o código, o servidor procura se este está associado a algum produto presente na base de dados (linha 86 da Figura 4.27). Em caso negativo, será então criado um produto somente com o campo código de barras preenchido (linha 88 da Figura 4.27).

```
82  async addOrCreateProductByBarcode(
83    createAnyProductDto: CreateAnyProductDto,
84  ): Promise<ToAddList> {
85    const barcode = createAnyProductDto.product.barcode;
86    let found = await this.productRepository.findOne({ barcode });
87    if (!found) {
88      found = await this.createProduct(createAnyProductDto);
89    }
90    this.eventsGateway.sendEvent();
91
92    return this.toAddListService.addProductToList(found.id);
93  }
```

Figura 4.27: Pedido POST *addOrCreateProductByBarcode* no Serviço

De forma a que o utilizador saiba que foi lido um produto pelo sistema, foi adotado a tecnologia *WebSocket* ao servidor que permite gerar uma sessão de comunicação interativa bidirecional entre o servidor e a aplicação. Com esta tecnologia é possível enviar mensagens para o servidor e receber respostas orientadas por eventos sem ter de sondar o servidor para obter uma resposta.

Neste Projeto foi criado um *WebSocket* de modo a notificar o consumidor que foi captado um produto pelo caixote do lixo, mediante uma notificação transmitida pela aplicação *Never Forget*.

Na linha 90 da Figura 4.27 é chamado o método *sendEvent* que se encontra definido no ficheiro *events.gateway.ts* como mostra a Figura 4.28. Este emite um evento denominado de "*raspberrypi*" (linha 11 da Figura 4.28) que será encaminhado ao cliente. Por fim, na linha 92 da Figura 4.27 o produto detetado é adicionado à lista de produtos lidos pelo caixote do lixo.

```
4   @WebSocketGateway({ namespace: 'events' })
5   export class EventsGateway {
6     @WebSocketServer()
7     server: Server;
8
9     async sendEvent(
10    ): Promise<void> {
11      this.server.emit('raspberrypi',);
12    }
13  }
```

Figura 4.28: *WebSocket*

Para que a aplicação consiga escutar os eventos enviados pelo servidor, foi utilizado uma biblioteca disponível no GitHub [56]. Após iniciar uma nova instância de Socket.IO que tem como argumento o URL onde são transmitidos os eventos (*http://never-forget.duckdns.org/events*), esta foi colocada a escutar o evento "*raspberrypi*", iniciando de seguida a conexão. Ao obter uma resposta do evento, esta será transmitida para uma função onde é criada a notificação a informar que foi lido um novo produto. Ao pressionar na notificação, o utilizador é reencaminhado para o *layout* onde são apresentados todos os produtos detetados pelo caixote e que ainda não foram inseridos em nenhuma lista.

Capítulo 5

Resultados

A presente secção tem como objetivo expor a solução implementada para o problema proposto no primeiro capítulo.

Neste capítulo serão expostos os *layouts* criados na aplicação *Never Forget*, assim como todas as funcionalidades que cada um destes apresenta. No Anexo C é exibido um esquema da aplicação desenvolvida, onde é possível perceber as rotas entre as diferentes interfaces.

Ao iniciar a aplicação é exibida uma interface de apresentação a esta, ou seja, o seu logotipo onde está inserido a sua denominação, *Never Forget*. A Figura 5.1 apresenta o *layout* de apresentação da aplicação.

Figura 5.1: Interface de apresentação

Ao arrancar a aplicação pela primeira vez, o utilizador é obrigado a criar uma lista de compras para poder começar a navegar pela aplicação adicionando e removendo itens a esta. O utilizador será forçado a criar uma lista ao iniciar a aplicação, sempre que a base de dados não apresentar nenhuma.

Depois de criar uma lista, o utilizador terá acesso à interface principal apresentada na Figura 5.2, onde poderá realizar as mais diversas ações.

Figura 5.2: Interface principal

O *layout* apresentado na Figura 5.2 é considerado a interface principal da aplicação, pois pode-se:

- Visualizar todos os itens a comprar, assim como a quantidade que está em falta;

- Visualizar que itens já foram comprados;
- Pesquisar subcategorias, e posteriormente produtos, de acordo com a categoria principal associada ou segundo a barra de pesquisa;
- Adicionar produtos ou subcategorias à lista selecionada de forma intuitiva e simples;
- Encaminhar o utilizador para as interfaces onde este pode criar subcategorias, produtos e ainda alterar a lista de compras selecionada;
- Encaminhar para o *layout* onde se pode observar que produtos foram lidos pelo caixote do lixo;
- Adicionar produtos através câmara de vídeo do *smartphone* mediante a leitura do seu código de barras.

Atendendo à Figura 5.2, é possível observar que a interface principal apresenta vários itens que estão inseridos na lista "Festa de anos". Os produtos são identificados com o seu nome seguido pela marca, tal como está exibido na lista, mais precisamente, o produto "Sumo de Laranja - Sumol", contudo, as subcategorias são representadas apenas pelo nome. Em ambos são exibidas as quantidades a comprar pelo utilizador, guiando-o no momento de compra.

No canto inferior esquerdo é indicado o total dos diversos produtos a comprar na lista em questão. Os produtos/subcategorias que possuem uma quantidade nula revelam que esse item foi comprado na totalidade, indicado igualmente mediante a seleção da *checkbox*.

Ao pressionar na área de sinalização “1” da Figura 5.2, o utilizador entrará numa nova interface, onde poderá trocar de lista, editar o nome da mesma como também removê-la. Nesta é ainda indicado o número de itens a comprar, tal como a interface principal. Na Figura 5.3 a) é possível visualizar o *layout* referente à troca de lista.

Para inserir um novo produto, o utilizador deverá clicar no campo sinalizado com o número 2 na Figura 5.2. Ao entrar no *layout* apresentado na Figura 5.3 b), o utilizador poderá inserir o nome, a marca, a subcategoria a que está associado e o código de barras caso o produto apresente um, por forma a criar um novo produto. Os três primeiros campos são obrigatórios, sendo que o produto só pode ser gerado se todos os campos estiverem preenchidos, exceto o código de barras que não é exigido.

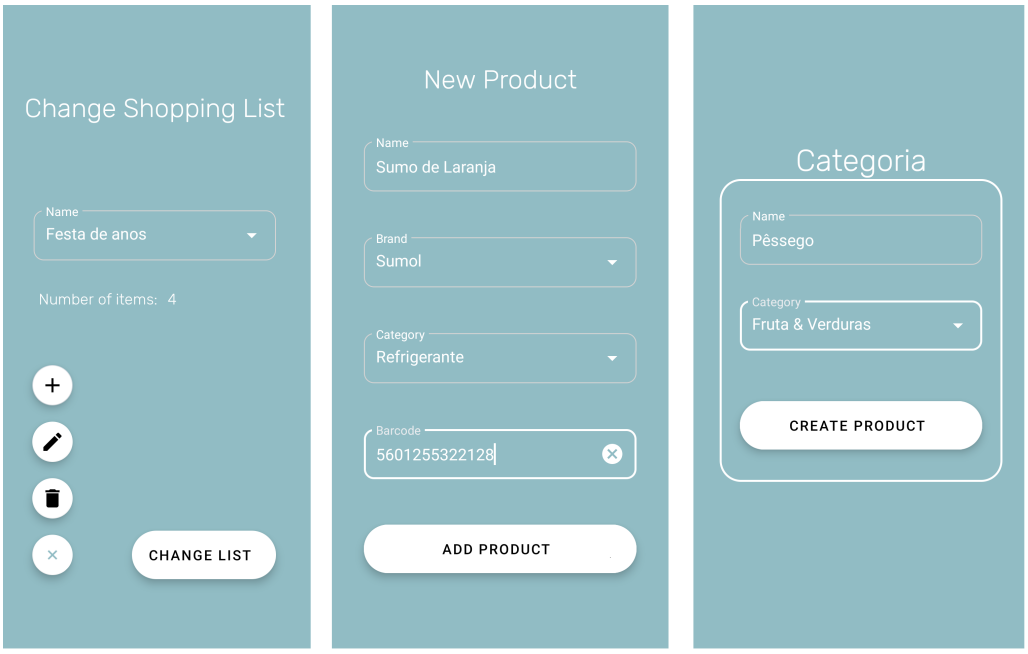
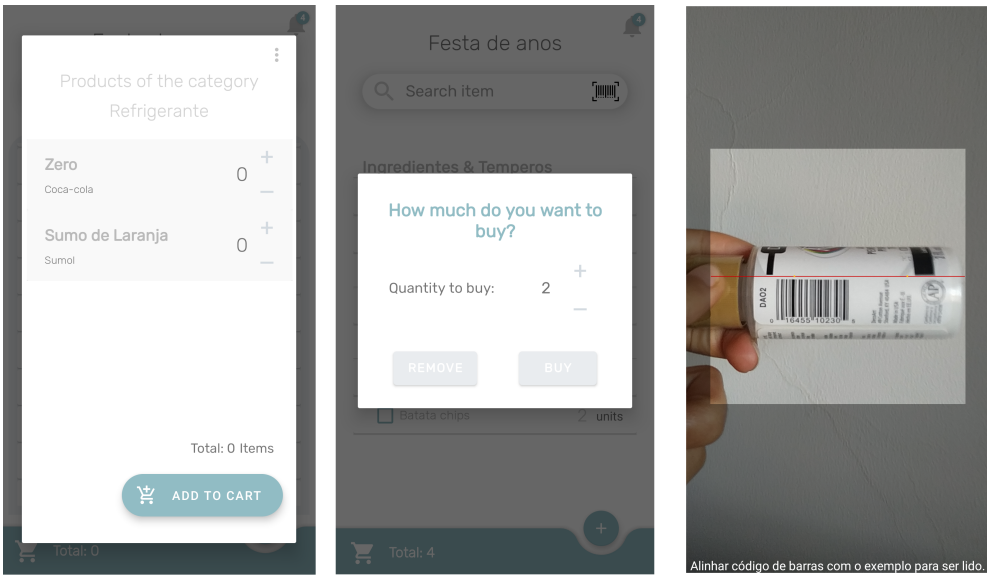


Figura 5.3: Interface para troca de lista a); para criar novo produto b); e para criar nova subcategoria c)

Ao pressionar na área marcada com o número 3, o utilizador terá acesso ao *layout* apresentado na Figura 5.3 c), onde poderá criar uma subcategoria. Nesta interface, o utilizador tem apenas de introduzir o nome que quer adicionar (campo *Name*) e selecionar a categoria principal a que esta se melhor insere (campo *Category*).

Como já foi referido na seção 4.5.3, a barra de pesquisa (sinalização 4 da Figura 5.2) permite fazer dois tipos de pesquisa de subcategorias, no entanto ao pressionar numa subcategoria surgirá uma janela à frente da interface principal, onde serão apresentados todos os produtos associados a esta. A Figura 5.4 a) mostra a janela com a lista de produtos associados à subcategoria "Refrigerante".

Nesta janela, o utilizador poderá inserir os produtos e quantidades que entender, assim como criar produtos novos relativos a esta subcategoria, sendo que o campo *Category* será automaticamente preenchido ao criar o produto. O utilizador poderá ainda editar os produtos existentes deslizando-os para a direita, e removê-los ao deslizá-los para a esquerda.



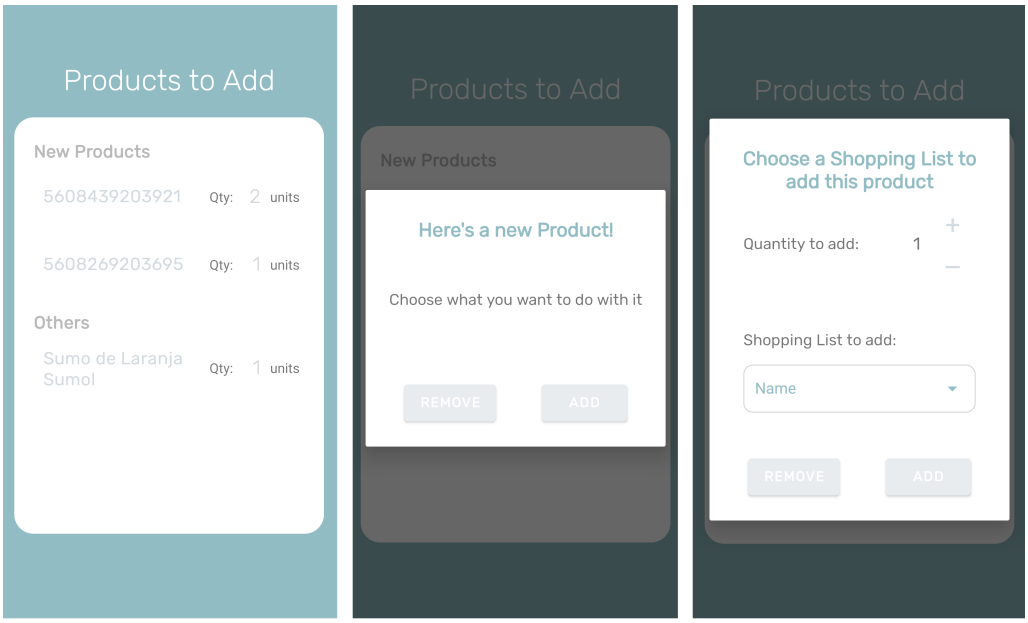
a) b) c)

Figura 5.4: Janela com lista de produtos da subcategoria "Refrigerante"a); Janela para comprar/remover itens de uma lista b); Captura de produtos mediante a câmara do *smartphone* c)

Ao selecionar um item que se encontra em falta na despensa (sinalização 5 da Figura 5.2) surgirá uma janela (Figura 5.4 b)) onde o utilizador poderá indicar a quantidade que tenciona comprar ou então poderá removê-lo da lista.

Ao pressionar no botão com um formato de código de barras que se encontra no fim da barra de pesquisa (sinalização 6 da Figura 5.2), o utilizador conseguirá capturar produtos através da câmara, tal como se pode visualizar na Figura 5.4 c).

Ao pressionar no sino que se encontra no canto superior direito do *layout* principal (sinalização 7 da Figura 5.2), o utilizador tem acesso aos produtos lidos pelo sistema de captura que se encontra no caixote do lixo pessoal. Nesta interface (Figura 5.5 a)) é possível observar os produtos novos que não são conhecidos pelo sistema, encontrando-se na seção "*New products*" e ao pressionar num deles surgirá uma outra janela (Figura 5.5 b)) que o permite remover da lista ou adicioná-lo. Caso seja adicionado, este passa a situar-se na parcela "*Others*" onde surgirá uma janela (Figura 5.5 c)) que permite inserir o produto à lista que o utilizador considerar mais adequada.



a) b) c)

Figura 5.5: Lista de produtos detetados pelo caixote do lixo a); Janela para adicionar/remover novo produto b); Janela para inserir produto a uma lista c)

A lista propriamente dita é visualizada na interface principal onde é possível observar que produtos são necessários comprar assim como as respetivas quantidades (Figura 5.6). Geralmente, nos supermercados os produtos encontram-se repartidos por seções, tal como a lista gerada. Os itens nesta estão agrupados consoante a categoria principal a que estão associados, orientando o consumidor para que este adquira os produtos por categorias, pois estes deverão se encontrar na mesma seção, reduzindo o tempo despendido no ato de recolher os produtos na unidade comercial.

Ao longo do ato de compra, o consumidor poderá indicar a quantidade adquirida de cada produto (Figura 5.4 b)), o que, por sua vez, permitirá indicar a quantia que faltará comprar ou se o produto se encontra adquirido na totalidade, tal como se encontra o elemento "*Donuts*" na Figura 5.6.

Esta interface em conjunto com as outras anteriormente apresentadas vêm responder de forma satisfatória aos requisitos e objetivos projetados para esta aplicação e sistema, dando toda a informação fundamental que o consumidor necessita para que esta tarefa se torne mais organizada e controlada.

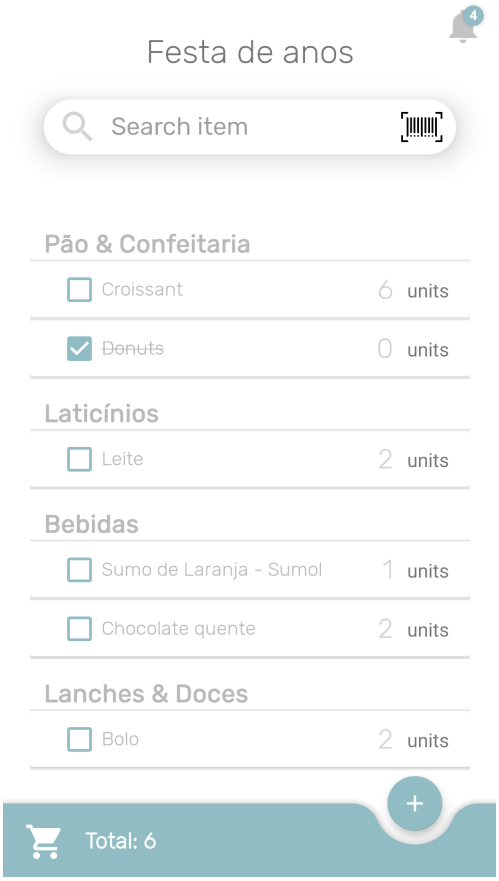


Figura 5.6: Interface da lista de compras

Capítulo 6

Conclusão

No presente capítulo são recapitulados os objetivos propostos neste Projeto e sintetizadas as principais contribuições, detalhando como os objetivos foram alcançados. Por fim, são expostas algumas observações sobre futuros desenvolvimentos, com base no trabalho desenvolvido neste Projeto.

O desenvolvimento deste Projeto foi motivado principalmente pelo tempo despendido pelos consumidores no ato de compra, assim como pelos produtos adquiridos de forma desnecessária. Estas compras impulsivas ou não planeadas aumentam o orçamento gasto e o tempo empregue neste processo rotineiro. Ao planear previamente o que comprar, o consumidor tende a adquirir somente o que necessita, reduzindo a duração do momento de compra. Em situações de pandemia como a vivída no ano 2020 em que o distanciamento social é aplicado no dia-a-dia do ser humano é importante que cada indivíduo realize as suas compras de forma rápida e eficaz, sem perder muito tempo a refletir o que está em falta na despensa, adquirindo todos os produtos sem esquecer nenhuma intenção de compra.

O presente Projeto assiste o consumidor de modo a que este compre somente o que tem em falta na sua despensa, ou seja, apenas o que necessita, reduzindo o tempo gasto nesta atividade e diminuindo o orçamento despendido. A principal finalidade deste Projeto passa por guiar o consumidor, dando-lhe toda a informação fundamental de forma a facilitar o processo de compra.

Neste sentido foi proposta a criação de um sistema para capturar produtos situado no caixote do lixo pessoal do consumidor. A localização do sistema foi escolhida estrategicamente de modo a que os produtos acabados de ser consumidos sejam lidos antes da embalagem ser colocada no lixo, permitindo saber exatamente o que falta na despensa, criando assim uma lista. Para que se tenha

acesso à informação relativa aos produtos a comprar, foi elaborada uma aplicação intuitiva que assiste o utilizador transmitindo-lhe a lista juntamente com todos os dados importantes para que o processo de compra seja uma prática mais agradável e menos fatigante.

Para o seu desenvolvimento, foi necessário o estudo da literatura disponível relativamente à área de Visão Computacional. A popularidade deste âmbito de investigação é bastante significativa e a quantidade de informação disponível é imensa, possibilitando um estudo vasto da área o que permitiu também conhecer as tecnologias existentes nesta. Foi realizado um estudo do mercado visando identificar e percecionar que soluções já existem e que tecnologias nestas emergem.

O estudo realizado previamente assim como a definição e análise dos requisitos para a realização do Projeto, permitiram estruturar da melhor forma o sistema por completo. A implementação dos algoritmos começou pela identificação de códigos de barras em imagens passando depois para a construção do algoritmo de deteção em tempo real. Estes algoritmos permitiram a aprendizagem da ferramenta Python que tornaram a leitura de produtos possível.

A implementação do servidor, responsável por gerir a informação do sistema, foi também concluída com sucesso, possibilitando o conhecimento de uma nova *framework*, mais precisamente *NestJS*, que concedeu todo o suporte para a construção do servidor e da base de dados adotada para o Projeto. O desenvolvimento da aplicação *Android* proporcionou um maior aprofundamento da linguagem *Java*, criando uma interface clara e acessível ao utilizador.

De um modo geral, os objetivos inicialmente propostos foram concluídos com sucesso, pois a interface gráfica transmite toda a informação que o consumidor necessita para uma compra organizada e controlada, relembrando-o todos os itens a adquirir. O sistema de captura de produtos deteta os produtos de forma rápida sem consumir muito tempo ao utilizador, adicionando-os aos produtos necessários a comprar, oferecendo todo um conjunto de benefícios à utilização deste sistema.

6.1 Considerações Futuras

Sugere-se, para desenvolvimentos futuros, que a aplicação permita a possibilidade de fazer *login* para os vários membros da família o que faria com que cada elemento soubesse exatamente o que se encontra na lista de compras, podendo adicionar algum produto que considere necessário. De modo a melhorar a experiência, propõe-se que exista a possibilidade de criar listas públicas e privadas, sendo as públicas acedidas por todos os membros do agregado familiar.

Propõem-se ainda, para futuros trabalhos, que os utilizadores tenham conhecimento através de um mapa geográfico (ou de uma lista) na aplicação de quantos utilizadores da aplicação *Never Forget* se encontram em cada supermercado em tempo real, permitindo obter uma estimativa do movimento nos estabelecimentos. Esta funcionalidade permitiria auxiliar o utilizador na decisão de qual o melhor momento e estabelecimento para realizar as suas compras. Contudo só seria viável apenas com uma elevada adesão a esta aplicação e sistema.

Bibliografia

[1] "Códigos de Barras - Perguntas e Respostas." https://invisibleflamelight.wordpress.com/2014/01/19/cdigos-de-barras-perguntas-e-respostas/. Acedido: 2020-04-07. [Quoted on p. v, 6, 11, 12, 15]

[2] C. Takahashi, "A Matemática dos Códigos de Barras." https://repositorio.unesp.br/bitstream/handle/11449/94272/takahashi_crs_me_sjrp.pdf?sequence=1&isAllowed=y. Acedido: 2020-04-05. [Quoted on p. v, 5, 6, 7]

[3] "Códigos de Barras UPC-A." https://www.cognex.com/pt-br/resources/symbologies/1-d-linear-barcodes/upc-a-barcodes. Acedido: 2020-04-08. [Quoted on p. v, 7]

[4] "Relatório de Atividades 2018 - GS1 Portugal." https://www.gs1pt.org/wp-content/uploads/2019/04/Relatorio-Atividades-2018.pdf. Acedido: 2020-04-13. [Quoted on p. v, 9]

[5] R. Bueno, "Detecção de Contornos em Imagens de Padrões de Escoamento Bifásico com Alta Fração de Vazio em Experimentos de Circulação Natural com o uso de Processamento Inteligente." https://www.teses.usp.br/teses/disponiveis/85/85133/tde-22042016-130130/publico/2016BuenoDeteccao.pdf. Acedido: 2020-04-26. [Quoted on p. v, vii, 19, 20, 21, 25, 26, 27, 28]

[6] M. de Albuquerque, E. Caner, A. Mello, and M. de Albuquerque, "Análise de Imagens e Visão Computacional." http://mesonpi.cat.cbpf.br/e2012/arquivos/g06/CursoE2012-PI.pdf. Acedido: 2020-04-28. [Quoted on p. v, 22, 23]

[7] M. Grassi, "Desenvolvimento e aplicação de um sistema de visão para robô industrial de manipulação." https://www.lume.ufrgs.br/bitstream/

handle/10183/6202/000482391.pdf?sequence=1&isAllowed=y. Acedido: 2020-04-28. [Quoted on p. v, 21, 23, 25]

[8] P. Maturana, "Algoritmos de Detecção de Bordas implementados em FPGA." https://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/pos-graduacao/273-dissertacao_patricia.pdf. Acedido: 2020-05-08. [Quoted on p. v, 28]

[9] "Continente Siga." https://play.google.com/store/apps/details?id=pt.continente.ContinenteSiga&hl=pt_PT. Acedido: 2020-05-27. [Quoted on p. v, 33]

[10] "Auchan." https://play.google.com/store/apps/details?id=pt.auchan.ecommerce&hl=pt_PT. Acedido: 2020-05-28. [Quoted on p. v, 35]

[11] "Bring! Lista de Compras." https://play.google.com/store/apps/details?id=ch.publisheria.bring&hl=pt_PT. Acedido: 2020-06-03. [Quoted on p. v, 36]

[12] "Raspberry Pi 4." https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. Acedido: 2020-05-16. [Quoted on p. v, 29, 41]

[13] "*Camera EPL 180degree*." http://www.webcamerausb.com/elp-180degree-fisheye-lens-1080p-wide-angle-web-usb-camerausb-camera-module-cmos-
html?fbclid=IwAR0dJELF1EN30PYkPY3Fl4ow3B2gIhAS7ZMTPnfn5U0ljYezcBWv377TTUs. Acedido: 2020-10-08. [Quoted on p. v, 42]

[14] "Besouro 12 V DC (3-28 V DC)." https://www.velleman.eu/products/view/?id=16109&country=be&lang=pt. Acedido: 2020-09-09. [Quoted on p. v, 42, 43, 55]

[15] "*Android Studio* - ciclo de vida de uma atividade." https://developer.android.com/guide/components/activities/activity-lifecycle?hl=pt-br. Acedido: 2020-10-06. [Quoted on p. v, 48]

[16] "*Android Studio - Send a simple request*." https://developer.android.com/training/volley/simple. Acedido: 2020-10-08. [Quoted on p. vi, 63]

[17] "Country codes of the EAN-13, GTIN-13." https://www.activebarcode.com/codes/ean13_laenderpraefixe.html. Acedido: 2020-04-16. [Quoted on p. vii, 11]

[18] J. Esquinca, "Aritmética: Códigos de Barras e outras aplicações de congruência." https://repositorio.ufms.br:8443/jspui/bitstream/123456789/1746/1/Josiane%20Colombo%20Pedrini%20Esquinca%281%29.pdf. Acedido: 2020-04-17. [Quoted on p. vii, 12, 13, 15]

[19] S. Magalhães and T. Mendonça, "Leitura de Códigos de Barras usando o MatLab." https://web.fe.up.pt/~up201304932/hugo/files/pt/feup/sbvi/report.pdf. Acedido: 2020-04-17. [Quoted on p. vii, 14]

[20] F. Santos, "Hábitos de compras e uso de lista de compras." http://www.scielo.mec.pt/scielo.php?script=sci_arttext&pid=S1645-44642009000100008. Acedido: 2020-05-24. [Quoted on p. 2]

[21] J. Phaniteja and P. Tom, "Evolution of Barcode." https://docplayer.net/9353550-Evolution-of-barcode.html. Acedido: 2020-04-09. [Quoted on p. 6]

[22] "Livro Comemorativo 30 anos GS1 Portugal." https://www.gs1pt.org/boxes/livro-comemorativo-30-anos-gs1-portugal/. Acedido: 2020-04-09. [Quoted on p. 7, 9, 10]

[23] "GS1 GEPIR Factsheet." https://www.gs1.org/docs/gepir/GEPIR_Factsheet.pdf. Acedido: 2020-04-12. [Quoted on p. 8]

[24] "GS1 Portugal - Ação em Portugal." https://www.gs1pt.org/acao-em-portugal/. Acedido: 2020-04-14. [Quoted on p. 9]

[25] R. Soares, "Estudo de Código de Barras por Análise de Imagens." http://repositorio.unicamp.br/bitstream/REPOSIP/259806/1/Soares_RicardoCorreia_M.pdf. Acedido: 2020-04-14. [Quoted on p. 10, 11, 12]

[26] T. Serrilho, "Identificação de código de barras em documentos de tramitação interna da UNEMAT com utilização de *webcam* para coleta de dados." https://www.passeidireto.com/arquivo/22767235/identificacao-de-codigo-de-barras-em-documentos-de-tramitacao-interna-da-unemat- Acedido: 2020-04-22. [Quoted on p. 17]

[27] L. Goulart, "Como escolher leitor de código de barras: muito além do tipo de leitor." https://www.promtec.com.br/leitor-de-codigo-de-barras/. Acedido: 2020-04-19. [Quoted on p. 17]

[28] "Quatro tipos de scanners de código de barras." https://www.universeoptics.com/barcode-scanners/. Acedido: 2020-04-22. [Quoted on p. 18]

[29] "Barcode Detection in Complex Environments." https://www.rsipvision.com/barcode-detection-in-complex-scenes/. Acedido: 2020-04-23. [Quoted on p. 18]

[30] D. Carvalho, "Processamento de Imagem num simulador de armazenamento automático." https://repositorio.ipl.pt/bitstream/10400.21/7050/1/Disserta%C3%A7%C3%A3o.pdf. Acedido: 2020-04-23. [Quoted on p. 18]

[31] M. de Albuquerque and M. de Albuquerque, "Processamento de Imagens: Métodos e Análises." http://www.cbpf.br/cat/pdsi/pdf/ProcessamentoImagens.PDF. Acedido: 2020-04-24. [Quoted on p. 18]

[32] "Processamento de imagens e visão computacional." http://foxfly.com.br/processamento-de-imagens-e-visao-computacional-saiba-a-diferenca/. Acedido: 2020-04-25. [Quoted on p. 19]

[33] J. Felix, "Sistema de Visão Computacional para Detecção e Quantificação de Enfisema Pulmonar." http://repositorio.ufc.br/bitstream/riufc/16081/1/2007_dis_jhsfelix.pdf. Acedido: 2020-04-26. [Quoted on p. 20]

[34] S. Neta, L. Dutra, and G. Erthal, "Limiarização Automática em Histogramas Multimodais." https://www.academia.edu/17854491/Limiariza%C3%A7%C3%A3o_Autom%C3%A1tica_em_Histogramas_Multimodais. Acedido: 2020-04-30. [Quoted on p. 22, 23]

[35] L. Martins, "Uma Abordagem de Reconhecimento de Objetos com uso da Projeção por Histograma voltada para Robótica Móvel." http://sistemaolimpo.org/midias/uploads/cffed09da2143b554d4391a8e3fc4e4c.pdf. Acedido: 2020-05-02. [Quoted on p. 22, 23, 24]

[36] B. Araújo, "Sistema de Visão de Máquina para Detecção e Localização Automático de Peças Utilizando Raspberry Pi." https://repositorio.ifpb.edu.br/xmlui/handle/177683/897. Acedido: 2020-05-16. [Quoted on p. 29]

[37] G. van Rossum, "Tutorial Python." https://dcc.ufrj.br/~fabiom/mab225/tutorialpython.pdf. Acedido: 2020-05-16. [Quoted on p. 30]

[38] "OpenCV." https://opencv.org/about/. Acedido: 2020-05-17. [Quoted on p. 31]

[39] A. Rosebrock, "An OpenCV barcode and QR code scanner with ZBar." https://www.pyimagesearch.com/2018/05/21/an-opencv-barcode-and-qr-code-scanner-with-zbar/. Acedido: 2020-05-20. [Quoted on p. 31]

[40] "Zbar bar code reader." http://zbar.sourceforge.net/index.html. Acedido: 2020-05-20. [Quoted on p. 31]

[41] V. Jorge, "Transações com app Continente Siga sobem 80%." https://www.distribuicaohoje.com/retalho/transacoes-com-app-continente-siga-sobem-80/. Acedido: 2020-05-27. [Quoted on p. 33]

[42] A. R. Costa, "Auchan lança app para gestão e registo de compras." https://www.distribuicaohoje.com/retalho/auchan-lanca-app-para-gestao-e-registo-de-compras/. Acedido: 2020-05-28. [Quoted on p. 34]

[43] J. Fernandes, "Auchan lança App Jumbo." https://wintech.pt/w-business/25337-auchan-lanca-app-jumbo. Acedido: 2020-05-28. [Quoted on p. 34]

[44] "NestJS." https://nestjs.com/. Acedido: 2020-08-28. [Quoted on p. 43]

[45] "Base de Dados." http://aprendis.gim.med.up.pt/index.php/Bases_de_Dados. Acedido: 2020-09-02. [Quoted on p. 45]

[46] "TypeScript." https://www.typescriptlang.org/. Acedido: 2020-08-30. [Quoted on p. 45]

[47] "TypeORM." https://typeorm.io/. Acedido: 2020-08-31. [Quoted on p. 45]

[48] "O que é o Postman?." https://enotas.com.br/blog/postman/. Acedido: 2020-08-31. [Quoted on p. 46]

[49] "*Android Studio*." https://developer.android.com/guide?hl=pt-BR. Acedido: 2020-09-12. [Quoted on p. 47, 58]

[50] "Raspberry pi os." https://www.raspberrypi.org/downloads/raspberry-pi-os/. Acedido: 2020-09-11. [Quoted on p. 51]

[51] "Github opencv." https://github.com/opencv/opencv. Acedido: 2020-09-12. [Quoted on p. 52]

[52] "Github - biblioteca volley." https://github.com/google/volley. Acedido: 2020-10-01. [Quoted on p. 59]

[53] "*Android Studio* - configurar requestqueue." https://developer.android.com/training/volley/requestqueue. Acedido: 2020-10-02. [Quoted on p. 60]

[54] "*Android Studio* - fazer uma solicitação." https://developer.android.com/training/volley/request. Acedido: 2020-10-02. [Quoted on p. 61]

[55] "Github - *ZXing Android Embedded*." https://github.com/journeyapps/zxing-android-embedded. Acedido: 2020-10-06. [Quoted on p. 66]

[56] "*Native Socket.IO and Android*." https://socket.io/blog/native-socket-io-and-android/. Acedido: 2020-10-08. [Quoted on p. 79]

Apêndice A

Identificação de Códigos de Barras numa Imagem

```
#import the necessary packages
from pyzbar import pyzbar
import argparse
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
  help="path to input image")
args = vars(ap.parse_args())

# load the input image
image = cv2.imread(args["image"])
# find the barcodes in the image and decode each of the barcodes
barcodes = pyzbar.decode(image)

# loop over the detected barcodes
for barcode in barcodes:
  # extract the bounding box location of the barcode and draw the
  # bounding box surrounding the barcode on the image
  (x, y, w, h) = barcode.rect
  cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
  # the barcode data is a bytes object so if we want to draw it on
  # our output image we need to convert it to a string first
  barcodeData = barcode.data.decode("utf-8")
  barcodeType = barcode.type
  # draw the barcode data and barcode type on the image
  text = "{} ({})".format(barcodeData, barcodeType)
  cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
    0.5, (0, 0, 255), 2)
```

```
31   # print the barcode type and data to the terminal
32   print("[INFO] Found {} barcode: {}".format(barcodeType,
       barcodeData))
33 # show the output image
34 cv2.imshow("Image", image)
35 cv2.waitKey(0)
```

Apêndice B

Sistema de Captura de Produtos em tempo real

```
# import the necessary packages
from imutils.video import VideoStream
from pyzbar import pyzbar

import argparse
import datetime
import imutils
import time
import cv2
import requests
import RPi.GPIO as GPIO

# BUZZER
# Disable warnings (optional)
GPIO.setwarnings(False)
# Select GPIO mode
GPIO.setmode(GPIO.BCM)
# Set buzzer - pin 23 as output
buzzer = 23
GPIO.setup(buzzer, GPIO.OUT)
GPIO.output(buzzer, GPIO.LOW)

# initialize the video stream and allow the camera sensor to warm
    up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
#vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

# loop over the frames from the video stream
```

```
while True:
  # grab the frame from the threaded video stream and resize it to
  # have a maximum width of 600 pixels
  frame = vs.read()
  frame = imutils.resize(frame, width=600)
  # find the barcodes in the frame and decode each of the barcodes
  barcodes = pyzbar.decode(frame)
    # loop over the detected barcodes
  for barcode in barcodes:
    # extract the bounding box location of the barcode and draw
    # the bounding box surrounding the barcode on the image
    (x, y, w, h) = barcode.rect
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
    # the barcode data is a bytes object so if we want to draw it
    # on our output image we need to convert it to a string first
    barcodeData = barcode.data.decode("utf-8")
    barcodeType = barcode.type
    # draw the barcode data and barcode type on the image
    text = "{} ({})".format(barcodeData, barcodeType)
    cv2.putText(frame, text, (x, y - 10),
      cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
    # show the barcode
    print(barcodeData)
    # Post request
    r = requests.post('http://localhost/products/barcode',data = {"
   barcode": barcodeData})
    # Turn on buzzer
    GPIO.output(buzzer,GPIO.HIGH)
    # Wait 0.2sec
    time.sleep(0.2)
    # Turn off buzzer
    GPIO.output(buzzer,GPIO.LOW)
  if len(barcodes)>=1:
      # Wait 1sec when detected barcodes
      time.sleep(1)
  # show the output frame
  cv2.imshow("Barcode Scanner", frame)
  key = cv2.waitKey(1) & 0xFF
```

Apêndice C

Esquema da aplicação *Never Forget*