

**UNIVERSIDADE PAULISTA
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO
TRABALHO DE CURSO**

**VISÃO COMPUTACIONAL PARA RECONHECIMENTO DE OBJETOS:
MONTAGEM DE LISTAS**

**ERICK TAKESHI HAYAKAWA
JOÃO PAULO FRANCISCO TIMÓTEO
LEANDRO SATOSHI TANAKA SAKAMOTO
LUCAS SERPA DA SILVA
NICOLAS ROCHA PEREIRA
PAULO HENRIQUE PRIMON DA SILVA**

SÃO JOSÉ DOS CAMPOS

2022

**ERICK TAKESHI HAYAKAWA
JOÃO PAULO FRANCISCO TIMÓTEO
LEANDRO SATOSHI TANAKA SAKAMOTO
LUCAS SERPA DA SILVA
NICOLAS ROCHA PEREIRA
PAULO HENRIQUE PRIMON DA SILVA**

**VISÃO COMPUTACIONAL PARA RECONHECIMENTO DE OBJETOS:
MONTAGEM DE LISTAS**

Trabalho de Curso apresentado ao Instituto de Ciências Exatas e Tecnologia da Universidade Paulista, como parte dos requisitos necessários para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Fernando Rodrigues de Sá

SÃO JOSÉ DOS CAMPOS

2022

UNIVERSIDADE PAULISTA
CURSO DE CIÊNCIA DA COMPUTAÇÃO
SÃO JOSÉ DOS CAMPOS - 2022

ERICK TAKESHI HAYAKAWA
JOÃO PAULO FRANCISCO TIMÓTEO
LEANDRO SATOSHI TANAKA SAKAMOTO
LUCAS SERPA DA SILVA
NICOLAS ROCHA PEREIRA
PAULO HENRIQUE PRIMON DA SILVA

**VISÃO COMPUTACIONAL PARA RECONHECIMENTO DE OBJETOS:
MONTAGEM DE LISTAS**

APROVADO EM ____/____/____

BANCA EXAMINADORA

NOME DO PROFESSOR – ORIENTADOR E PRESIDENTE DA BANCA

NOME DO PROFESSOR – EXAMINADOR

NOME DO PROFESSOR – EXAMINADOR

Dedicamos este trabalho de conclusão de curso a todos os nossos familiares que nos apoiaram durante essa trajetória acadêmica, ao orientador, que nos instruiu no decorrer deste trabalho e a todos os professores que nos lecionaram durante a realização do bacharelado em Ciência da Computação.

AGRADECIMENTOS

Agradecemos aos colegas, familiares e professores, que nos deram suporte às dificuldades e barreiras que encontramos durante esta etapa acadêmica que estamos prestes a finalizar, com todo o conhecimento que nos foi proporcionado, além dos ensinamentos que levaremos para o nosso dia a dia.

Obrigado Doutor Fernando de Sá, instrutor e conselheiro de nosso trabalho de conclusão de curso. Agradecemos sua confiança e dedicação em nosso grupo. Nunca desistindo de nosso esforço, sempre nos guiando e aconselhando com dedicação sobre o que fazer para nos impedir de cometer erros durante os momentos mais complicados e desafiadores do nosso projeto.

“Tente uma, duas, três vezes e se possível tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas primeiras tentativas, a persistência é amiga da conquista. Se você quer chegar aonde a maioria não chega, faça o que a maioria não faz”.

Bill Gates

RESUMO

Este trabalho de conclusão de curso tem como objetivo o auxílio às pessoas na realização de compras. Para atingir essa meta, foi desenvolvido um aplicativo para dispositivos móveis capaz de agilizar a criação de listas a elas associadas a partir do uso da visão computacional, implementada neste projeto para realizar o reconhecimento de produtos. Esse processo possui uma grande importância, pois ele será crucial para a criação da lista inteligente dos itens que devem ser reabastecidos numa possível compra futura, usando como base a última lista gerada. Levando em conta que o público-alvo deste projeto abrange desde jovens, adultos até pessoas da terceira idade, utilizou-se um modelo simples e intuitivo, prezando pela usabilidade e facilidade de uso. Vale ressaltar que foi realizada uma pesquisa quantitativa para se obter o nível de aceitação do produto e foi possível observar que a grande maioria faria uso da aplicação caso existisse um aplicativo como esse. Desse modo, é possível concluir que através desse *software*, será possível evitar a necessidade de uma criação manual de uma lista de produtos, reduzindo drasticamente o tempo gasto com esse tipo de trabalho.

Palavras-chave: Visão Computacional; Reconhecimento; Lista Inteligente; Facilidade de Uso.

ABSTRACT

This final paper aims to assist people when shopping. To achieve this goal, an application for mobile devices was developed to speed up the creation of shopping lists using computer vision, implemented in this project to perform product recognition. This process is of great importance, because it will be crucial for the creation of the smart list of items that should be replenished in a possible future purchase, based on the last generated list. Considering that the target audience of this project ranges from young people, adults to elderly people, a simple and intuitive template was used, praising usability and ease of use. It is worth mentioning that a quantitative survey was carried out to obtain the acceptance level of the product and it was possible to observe that the great majority would use the application if such application existed. Therefore, it is a possible conclusion that through this software, it will be possible to avoid the necessity of manual creation on shopping lists, drastically reducing the time spent on this kind of work.

Keywords: Computer Vision; Recognition; Smart list; Ease of use.

LISTA DE ILUSTRAÇÕES

Figura 1 – Gráfico de Regiões do Brasil	19
Figura 2 – Gráfico de periodicidade de compras	20
Figura 3 – Gráfico de aceitação de uso	20
Figura 4 – Modelo de identificação de classe	24
Figura 5 – Modelo de identificação de classe	25
Figura 6 – Geração de uma pontuação	27
Figura 7 – Quatro passos do YOLO	30
Figura 8 – Diagrama com todas as funcionalidades do Sistema	35
Figura 9 – Diagrama de caso de uso “Criar Lista de Compras”	36
Figura 10 – Diagrama de caso de uso “Listar Lista de Compra”	37
Figura 11 – Diagrama de caso de uso “Finalizar Lista de Compras”	38
Figura 12 – Diagrama de caso de uso “Criar Histórico de Compras”	40
Figura 13 – Diagrama de caso de uso “Buscar Histórico de Compras”	41
Figura 14 – Diagrama de caso de uso “Criar Conta”	42
Figura 15 – Modelo Conceitual	43
Figura 16 – Modelo Lógico	45
Figura 17 – Logotipo do aplicativo	46
Figura 18 – Paleta de cores	47
Figura 19 – Tela de login	48
Figura 20 – Tela de cadastro	49
Figura 21 – Telas de lista de compra e compras realizadas	50
Figura 22 – Modal de adicionar lista	50
Figura 23 – Tela da câmera	51
Figura 24 – Modal de visualização	52

Figura 25 – Tela de adicionar produtos	53
Figura 26 – Tela de checar produtos	54
Figura 27 – Estrutura de pastas	55
Figura 28 – Navegação Stack	56
Figura 29 – Navegação Tab	57
Figura 30 – Hooks	57
Figura 31 – Componentes nativos	59
Figura 32 – Componentes personalizados	60
Figura 33 – Arquivo de estilização.....	61
Figura 34 – Funções.....	62
Figura 35 – Ciclo de dados com a context API.....	63
Figura 36 – Context API	63
Figura 37 – Exemplo de requisição	64
Figura 38 – Requisição usuário	66
Figura 39 – Resposta usuário.....	66
Figura 40 – Requisição login	67
Figura 41 – Resposta login.....	67
Figura 42 – Requisição autenticação	68
Figura 43 – Resposta autenticação	69
Figura 44 – Requisição lista de compras.....	69
Figura 45 – Resposta lista de compras	69
Figura 46 – Requisição inserir produto.....	70
Figura 47 – Resposta inserir produto.....	71
Figura 48 – Requisição busca de listas	72
Figura 49 – Resposta busca de listas.....	72

Figura 50 – Requisição de busca de lista	73
Figura 51 – Resposta da busca lista	74
Figura 52 – Requisição inserção de produto	74
Figura 53 – Resposta inserção de produto.....	75
Figura 54 – Requisição busca de compras.....	76
Figura 55 – Requisição de busca de compras por ID.....	76
Figura 56 – Requisição para criação de lista inteligente.....	77
Figura 57 – Resposta para criação de lista inteligente	78
Figura 58 – Importação das bibliotecas	79
Figura 59 – Comunicação com o terminal	80
Figura 60 – Comando para o terminal	80
Figura 61 – Comandos para percorrer todo o arquivo	81
Figura 62 – Armazenamento dos resultados	81
Figura 63 – Retorno da lista de índices	82
Figura 64 – Construção da imagem final	83
Figura 65 – Função draw_prediction	83
Figura 66 – Métodos de imshow.....	84
Figura 67 – Criação de JSON.....	84
Figura 68 – Imagem input.....	85
Figura 69 – Imagem com reconhecimento	85
Figura 70 – Retorno do JSON	86
Figura 71 – Imagem input.....	86
Figura 72 – Retorno do JSON	87
Figura 73 – Retorno do JSON	87
Figura 74 – Métodos utilizados para criar lista de compras.....	88

Figura 75 – Tempo gasto para criação de lista de compras	89
Figura 76 – Porcentagem de pessoas que criam lista de compras	89
Figura 77 – Periodicidade nas compras	90
Figura 78 – Gráfico de aceitação de uso	90
Figura 79 – Reconhecimento de produtos	91

LISTA DE TABELAS

Tabela 1 – Caso de Uso “Criar Lista de Compras”	36
Tabela 2 – Caso de Uso “Listar Lista de Compras”	37
Tabela 3 – Caso de Uso “Finalizar Lista de Compras”	38
Tabela 4 – Caso de Uso “Criar Histórico de Compras”	40
Tabela 5 – Caso de Uso “Buscar Histórico de Compras”	41
Tabela 6 – Caso de Uso “Criar Conta”	42

LISTA DE ABREVIATURAS E SIGLAS

APIs	<i>Application Programming Interface</i> (Interface de Programação de Aplicativos)
CNN	Rede Neural Convolucional
CPU	<i>Central Processing Unit</i> (Unidade Central de Processamento)
CSS	<i>Cascading Style Sheets</i> (Folhas de Estilo em Cascata)
CUDA	<i>Compute Unified Device Architecture</i> (Arquitetura de Dispositivo de Computação Unificada)
ENEM	Exame Nacional do Ensino Médio
FGV	Fundação Getúlio Vargas
GPU	<i>Graphics Processing Unit</i> (Unidade de Processamento Gráfico)
HTML	<i>HyperText Markup Language</i> (Linguagem de Marcação de HiperTexto)
HTTP	<i>HyperText Transfer Protocol</i> (Protocolo de Transferência de Hipertexto)
IA	Inteligência Artificial
ID	Identificador Único
IOS	<i>iPhone Operating System</i> (Sistema Operacional do iPhone)
JS	<i>JavaScript</i>
JSON	<i>JavaScript Object Notation</i> (Notação de Objetos JavaScript)
JWT	JSON Web Token
mAP	<i>Mean Average Precision</i> (Média das Precisas Médias)
MER	Modelo Entidade Relacionamento
ORM	<i>Object-relational mapping</i> (Mapeamento objeto-relacional)
PHP	<i>PHP: Hypertext Preprocessor</i> (Pré-processador de Hipertexto PHP)
RNN	Rede Neural Recorrente
SQL	<i>Structure Query Language</i> (Linguagem de Consulta Estruturada)
SGBD	Sistema de Gerenciamento de Banco de Dados
URL	<i>Uniform Resource Locator</i> (Localizador Uniforme de Recursos)
UUID	<i>Universally Unique Identifier</i> (Identificador Único Universal)
YOLO	<i>You Only Look Once</i> (Você só olha uma vez)

SUMÁRIO

1 INTRODUÇÃO	17
1.1 Objetivo geral	18
1.2 Objetivos específicos	18
1.3 Procedimentos Metodológicos	18
1.3.1 Pesquisas e Análise dos Resultados.....	19
1.4 Estrutura do trabalho	21
2 FUNDAMENTAÇÃO TEÓRICA	22
2.1 Inteligência Artificial	23
2.2 Visão Computacional	23
2.2.1 Classificação de imagem e Detecção de objetos	24
2.2.2 Deep Learning	26
2.2.3 Rede Neural Convolucional (CNN)	26
2.3 Python	28
2.4 Darknet e YOLO	29
2.4.1 Funcionamento do YOLO	30
2.5 JavaScript	30
2.6 Node.js	31
2.7 React Native	31
2.8 PostgreSQL.....	32
3 O APLICATIVO LISTA FÁCIL	33
3.1 Requisitos	33
3.1.1 Requisitos funcionais.....	33
3.1.2 Requisitos não funcionais.....	34

3.2 MODELAGEM.....	34
3.2.1 Requisitos do Usuário	35
3.2.2 Diagrama de caso de uso.....	35
3.2.3 Modelo Conceitual.....	43
3.2.4 Modelo Lógico	44
3.3 FRONT-END	46
3.3.1 Logotipo.....	46
3.3.2 Cores.....	47
3.3.3 Telas.....	47
3.3.4 Login.....	48
3.3.5 Cadastro.....	48
3.3.6 Lista de compra e compras realizadas	50
3.3.7 Câmera.....	51
3.3.8 Adicionar produtos à compra.....	53
3.3.9 Checagem de produtos da lista	53
3.3.10 Desenvolvimento	54
3.3.10.1 Estrutura de arquivos	55
3.3.10.2 Códigos	56
3.3.10.3 Sistema de navegação	56
3.3.10.4 Hooks	57
3.3.10.5 Componentes nativos.....	58
3.3.10.6 Componentes personalizados	59
3.3.10.7 Estilização	60
3.3.10.8 Funções.....	61
3.3.10.9 Contextos	62

3.3.10.10 APIs.....	64
3.4 Back-end	65
3.4.1 Autenticação.....	65
3.4.2 Rotas sem autenticação	66
3.4.3 Rota para criação de Usuário.....	66
3.4.4 Rota para login	67
3.4.5 Rotas Autenticadas	68
3.4.6 Rota de Criação de Lista de Compras.....	69
3.4.7 Rota para inserção produtos na Lista de Compras	70
3.4.8 Rota para buscar todas as listas de compras do usuário	71
3.4.9 Rota para buscar uma lista de compra por ID	73
3.4.10 Rota para a criação de Compras.....	73
3.4.11 Rota para inserção de produtos na compra	74
3.4.12 Rota para buscar todas as compras do usuário	75
3.4.13 Rota para buscar uma compra específica por ID	76
3.4.14 Rota para a criação da lista inteligente.....	77
3.5 PROCESSO PARA O RECONHECIMENTO DE ITENS	78
3.5.1 Configurações necessárias	79
3.5.2 Explicação do Script	79
3.5.3 Apresentação de resultados do Script.....	84
4 RESULTADOS.....	88
5 CONSIDERAÇÕES FINAIS	93
REFERÊNCIAS.....	94

1 INTRODUÇÃO

O presente Trabalho de Conclusão de Curso tem como o tema a visão computacional para reconhecimento de objetos, delimitando-se ao reconhecimento de produtos encontrados em supermercados.

A evolução na capacidade de processamento de imagens vem contribuindo no auxílio das tarefas do dia a dia, através de *softwares* e aplicativos capazes de substituir trabalhos manuais por processos automatizados.

Sendo assim, escolheu-se o desenvolvimento do Lista Fácil, um aplicativo de gerenciamento de compras que visa criar uma lista de compras de produtos encontrados em supermercados, usando como base fotos capturadas pelo usuário, eliminando, assim, a necessidade de uma criação manual de uma lista.

O processamento de imagens é um estágio crucial para o desenvolvimento do aplicativo, pois é nele que ocorre a detecção dos objetos, permitindo a obtenção das informações desejadas. Isso ajudará os usuários a realizar o reconhecimento dos produtos de forma automatizada, que é um processo extremamente complicado e custoso.

Tendo em vista que o dia a dia das pessoas é conturbado, surgiu-se a ideia de se desenvolver uma aplicação capaz de facilitar e agilizar a montagem da listagem de itens a serem comprados, que é uma tarefa corriqueira para grande parte das pessoas. Dentre os processos mais cansativos da compra está a criação de uma lista de compras, pois dependendo do tamanho da família ou da quantidade de produtos que necessitam ser comprados para reabastecer o estoque, pode-se demorar várias horas para realizar tal processo.

De acordo com um estudo da FGV (Fundação Getúlio Vargas), há 242 milhões de telefones inteligentes em uso no Brasil, isto significa que, teoricamente, cada brasileiro possui ou porta mais de um *smartphone*. Tendo isso em vista, o desenvolvimento de uma aplicação *mobile* foi escolhido, levando em conta a alta disponibilidade de celulares.

Inicialmente, será necessário que o usuário tire uma foto nítida de sua dispensa com todos os produtos adquiridos na compra inicial, com a finalidade de identificar e registrar cada produto detectado em sua compra na base de dados da aplicação.

Na próxima compra, uma outra foto será necessária para a montagem da lista, realizando-se a comparação das 2 imagens e a identificação das mercadorias que

foram consumidas durante todo o mês, a fim de gerar uma lista pronta para que o cliente possa efetuar a compra dos itens ausentes que não foram detectados.

Por fim, vale ressaltar que o usuário poderá criar uma conta no aplicativo, pois isso permitirá que ele seja capaz de criar um histórico de compras e cadastrar uma lista de compras para acessá-los posteriormente quando for necessário.

A seguir serão apresentados o objetivo geral, os objetivos específicos, os procedimentos metodológicos e a estrutura deste trabalho.

1.1 Objetivo geral

O objetivo geral consiste na utilização da visão computacional para a criação de um aplicativo, baseado em reconhecimento de objetos, auxiliando a comunidade no processo de criação de listas de compras.

1.2 Objetivos específicos

- a. Realizar a pesquisa sobre o Elemento Computacional;
- b. Desenvolver um sistema que utilize a Visão Computacional;
- c. Realizar o treinamento da Rede Neural para reconhecer objetos específicos;
- d. Listagem de itens reconhecidos pela rede neural na análise da imagem enviada;
- e. Criar a lista com os produtos reconhecidos; e
- f. Finalização da lista e *checklist* de compras.

1.3 Procedimentos Metodológicos

Para este trabalho, o tipo de pesquisa utilizado foi a descritiva, já que se permite uma nova visão com relação ao tema, visando uma aplicação na qual possa ajudar as pessoas em atividades comumente realizadas no dia a dia.

A aplicação da coleta de dados foi feita de forma online, através de um questionário em que os usuários respondem perguntas sobre frequência e tempo gasto para realizar a ação. Os entrevistados utilizados foram pessoas de diversas

idades e classes sociais para a maior abrangência de dados. A pesquisa ocorreu em diversas cidades, sendo aplicada em mais de 256 pessoas aqui no Brasil.

Ao longo do trabalho, foi realizada a obtenção dos dados utilizados na fundamentação teórica e para entendimento dos princípios iniciais do desenvolvimento. Em seguida, foram definidos os requisitos e modelos que deveriam ser aplicados para o desenvolvimento do aplicativo e do reconhecimento dos objetos.

Para a coleta de dados foi realizada por meio de entrevistas contendo uma série de perguntas relacionadas ao tema, a fim de apresentar a frequência e ações dos entrevistados sobre a relevância do tema apresentado.

Os dados coletados através das entrevistas foram apresentados em gráficos para melhor visualização e entendimento dos resultados obtidos. Podendo assim ser analisado de forma quantitativa, para melhorar a análise das respostas, mas podendo ser analisado de forma qualitativa também, melhorando a interpretação de algumas respostas.

1.3.1 Pesquisas e Análise dos Resultados

Com a realização de uma coleta de dados online, foi possível verificar a relevância de uma aplicação na qual auxilie o usuário a realizar uma listagem inteligente de sua lista de compras.

Tendo 256 respostas de diversas pessoas do Brasil, sendo elas 74,2% da região sudeste do país, como mostrado na Figura 1.

Figura 1 – Gráfico de Regiões do Brasil



Fonte: Autoria Própria, 2022.

Analisando os resultados obtidos verificamos que a frequência mais comum da realização de compras é semanalmente com um total de 45,7% das pessoas, como demonstrado na Figura 2.

Figura 2 – Gráfico de periodicidade de compras



Fonte: Autoria Própria, 2022.

Como finalização do questionário foi perguntado se as pessoas consideram a ideia de uma aplicação que auxilia na criação de uma lista de compra, uma ideia válida e se elas utilizariam esse aplicativo para a realização da lista de compra, como mostra a Figura 3.

Figura 3 – Gráfico de aceitação de uso



Fonte: Autoria Própria, 2022.

Com esse último questionamento pode-se concluir que a criação de um aplicativo como o Lista Fácil é relevante para uma parte das pessoas, na qual desejam um auxílio para facilitar e agilizar a criação de uma lista de compra.

1.4 Estrutura do trabalho

Este TCC terá a seguinte estrutura de capítulos:

O capítulo 2, a seguir, trata-se do referencial teórico, onde estão inseridos os principais módulos e conceitos utilizados como base na criação deste trabalho, constando com tópicos de visão computacional, inteligência artificial entre outros.

O Capítulo 3 apresenta os requisitos funcionais e não funcionais definidos para o trabalho e aplicativo que serão desenvolvidos.

No capítulo 4 é apresentada a modelagem onde pode-se verificar os casos de uso, modelo entidade relacionamento, Modelo lógico e conceitual, na qual mostra o fluxo na qual o usuário irá percorrer, além das ações que o sistema deve executar.

O capítulo 5 representa a parte do *front-end* do sistema desenvolvido, contendo a explicação das telas desenvolvidas e suas características. Além de possuir suas principais funções retratadas em linhas de código.

O capítulo 6 apresenta os códigos e métodos utilizados para a realização da programação no *back-end* em Node.js, com exemplos de requisições e retornos das APIs utilizadas para a comunicação do *front-end* entre o *back-end*.

O capítulo 7 contém os códigos representantes da parte de reconhecimento de objetos, contendo exemplos dos códigos além das explicações de como é feita a análise e como é retornado os resultados das imagens para o *front*.

No capítulo 8 é apresentado os resultados adquiridos com toda a análise e pesquisa sobre o assunto abordado, visando os testes e aplicações do sistema que ocorreram durante o desenvolvimento deste trabalho.

Por fim, no capítulo 9, procura-se concluir todo o processo realizado para a pesquisa, construção e desenvolvimento deste sistema, juntamente com os resultados adquiridos de pesquisas e entrevistas.

2 FUNDAMENTAÇÃO TEÓRICA

Com o objetivo de facilitar e agilizar o processo de criação e gerenciamento de uma lista de compra, será desenvolvido uma aplicação *mobile* onde seja possível criar uma lista de itens a serem comprados, além do gerenciamento de sua compra, utilizando principalmente o processamento de imagem para reconhecimento dos produtos.

Sabendo que grande parte da sociedade brasileira já possui um smartphone atualmente, além de visar a mobilidade e facilidade de utilização em qualquer local ou hora. Foi escolhido desenvolver uma aplicação *mobile* onde pudesse influenciar no dia a dia do máximo de pessoas.

De início, será preciso uma imagem dos itens comprados para ser feita a listagem de itens em estoque, assim que for listado os itens de estoque o usuário poderá verificar quais itens foram identificados e validar se está tudo correto ou se deseja adicionar mais algum item.

Após o tempo de consumo dos produtos, o cliente terá a possibilidade de montar sua lista de compra, utilizando o processamento de uma imagem nos itens que restaram em estoque. Comparando as duas listas é possível identificar e montar a lista com os itens em falta.

Com os itens faltantes listados, o usuário poderá adicionar ou remover os itens que desejar para que a lista seja o mais flexível possível e ajustável aos diversos usuários do sistema. Com a lista montada, o usuário pode ir ao mercado para realizar as compras.

No mercado, o usuário terá um *checklist* para verificar quais itens ainda estão pendentes em sua lista e quais já estão em seu carrinho. Ao finalizar o *checklist*, o usuário finaliza a compra, caso tenha comprado algum item a mais ele poderá adicioná-lo na listagem de itens comprados.

Utilizando a visão computacional como elemento computacional principal, permite reconhecer os produtos e realizar a lista de forma rápida. A linguagem Python foi escolhida como principal ferramenta para essa função do sistema devido a facilidade de uso e a simplicidade de aplicação.

Já que o sistema será desenvolvido para dispositivos *mobile*, será utilizado Node.js e React Native, respectivamente, para o desenvolvimento do *back-end* e do *front-end* da aplicação, possibilitando assim a comunicação entre a aplicação e o

processamento das imagens. Já para armazenar a informações, será utilizado o PostgreSQL como Sistema de Gerenciamento de Banco de Dados Relacional com isso. A seguir serão explicadas todas as tecnologias utilizadas para desenvolver o *software*.

2.1 Inteligência Artificial

Inteligência Artificial (IA) pode ser explicada como ciência e engenharia de computadores inteligentes, relacionadas a tarefas para entender a inteligência humana, mas sem a limitação de fatores biológicos.

Em 1950 foi publicado por Alan Turing, conhecido como “*pai da ciência da computação*”, o primeiro artigo que deu início à conversa sobre Inteligência Artificial, onde ocorreu o surgimento da seguinte pergunta: “As máquinas podem pensar?”.

Juntamente com essa pergunta Turing lançou um teste, chamado de “Teste de Turing”, na qual tem como objetivo distinguir respostas escritas por computadores ou dada por humanos, onde uma pessoa realizaria as perguntas para a máquina ou o humano, a partir dessa série de respostas o examinador deve julgar se as respostas surgiram de uma IA ou um humano.

A inteligência artificial pode ser classificada em 2 tipos, a primeira delas é a Inteligência Artificial fraca na qual tem como objetivo geral realizar uma tarefa em específico. Já a segunda é a Inteligência Artificial Forte é uma forma de IA que teria uma inteligência igualável a inteligência humana, tendo a capacidade de resolver problemas, aprender e planejar o futuro pensando como se fosse um humano.

2.2 Visão Computacional

Visão computacional é uma área da Inteligência Artificial que atua como um sistema que envia informações extraídas de imagens digitais, como vídeos, fotos e outros tipos de imagens digitais.

Enquanto uma Inteligência Artificial é o cérebro de um computador a visão computacional são seus olhos, funcionando praticamente como os olhos humanos, mas com a desvantagem de ter a experiência de vida para diferenciar os objetos.

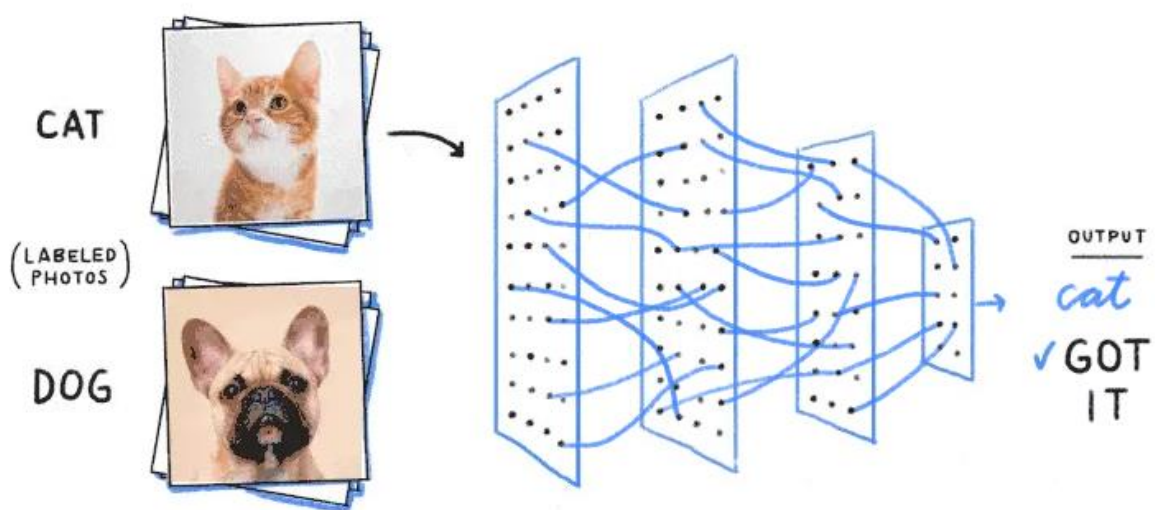
Para que a máquina perca esta desvantagem é preciso realizar um treinamento, podendo assim realizar a diferenciação de objetos. Sendo uma máquina este treinamento é realizado com um tempo muito menor do que humanos precisam, já que em um minuto um computador pode realizar centenas de processos além de uma precisão e percepção muito maior que a humana.

2.2.1 Classificação de imagem e Detecção de objetos

Dentro da visão computacional existem duas áreas que são capazes de identificar um objeto dentro de uma imagem, a classificação de objetos e a detecção de objetos.

O sistema de classificação de objetos busca apenas identificar uma classe de objeto dentro da imagem fornecida, nesse caso pode-se utilizar uma imagem de um gato como exemplo, mostrado na Figura 4, onde o sistema de classificação só retornará a classe do objeto identificado, nesse caso gato.

Figura 4 – Modelo de identificação de classe



Fonte: Becoming Human, 2017.

Já em um sistema de detecção de objetos é necessário identificar qual classe de objeto está sendo mostrado e a sua localização dentro da imagem, sendo assim necessário retornar com o resultado da classe identificada e posição além de mais um parâmetro de confiança, que representa o grau de certeza da predição, como mostrado na Figura 5.

Figura 5 – Modelo de identificação de classe



Fonte: iaexpert.academy, 2020.

Para que um objeto seja reconhecido por um computador utilizando a visão computacional é preciso analisar inúmeras vezes imagens do objeto e suas partes, isso para apenas para reconhecer e diferenciar um objeto com precisão, com diversos objetos seria preciso um tempo muito maior para que o treinamento seja perfeito.

O treinamento da máquina utiliza duas tecnologias principais, o *Deep Learning* e a Rede Neural Convolucional (CNN), onde o *Machine Learning* permite o computador a se auto ensinar os dados extraídos das imagens, fazendo um programa autossuficiente sem precisar que alguém o programe para reconhecer algo.

A Rede Neural Convolucional divide a imagem em pixels e camadas que auxiliam no aprendizado da máquina, realizando convoluções, operação utilizada em duas funções matemáticas para criar uma terceira, nas camadas, gerando previsões do que ele analisou. A Rede Neural realiza uma série de convoluções e verifica a precisão de suas análises até que suas previsões sejam compatíveis com o resultado desejado.

2.2.2 Deep Learning

Deep Learning faz parte do *Machine Learning*, que tem como objetivo fornecer dados e informações utilizando a análise de diferentes tipos de dados. Sendo criada para funcionar como um cérebro humano trocando informações entre neurônios, neste caso criando uma rede neural.

Nesse método as informações são processadas em camadas, sendo a primeira camada a inserção dos dados para a análise e a última os resultados adquiridos no processamento. Podendo existir diversos números de camadas entre elas variando de acordo com a operação.

Ao receber as informações é feita a distinção detalhada dos dados, que serão enviadas para os neurônios realizarem análises matemáticas atribuindo um peso a cada detalhe enviado, criando uma listagem com pesos para que no final o sistema consiga criar uma relação entre os detalhes adquiridos e o resultado esperado.

Como o *Deep Learning* exige uma grande quantidade de dados para seu treinamento, a abundância de dados e cálculos necessários para a análise seria demasiada para uma máquina convencional processar, já que é necessário uma alta capacidade de processamento da GPU (*Graphics Processing Unit*).

2.2.3 Rede Neural Convolucional (CNN)

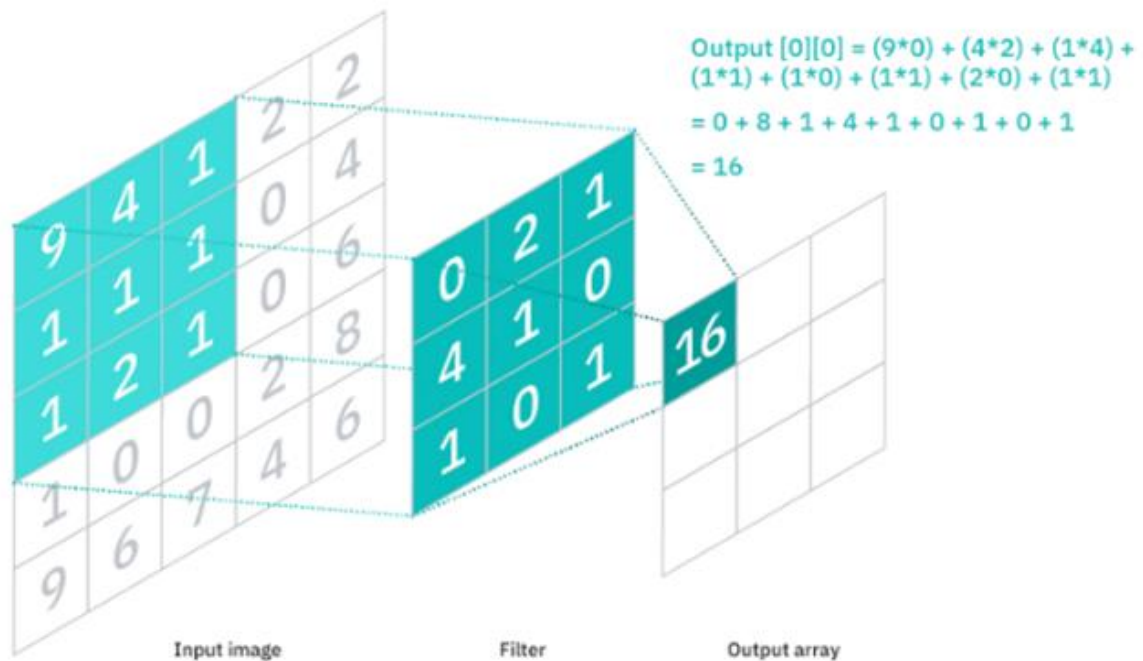
Rede Neural é uma subárea do *Machine Learning*, sendo o componente principal do *Deep Learning*. Uma Rede Neural Convolucional ou CNN é notada pela sua superioridade no processamento de imagens, falas ou áudios. Consistindo em 3 camadas principais: Camada Convolucional, Camada de polimento e por último a camada de Conexão Total.

A primeira camada, camada convolucional, realiza a grande parte da computação que ocorre no processamento de uma CNN. Onde é preciso alguns componentes, os dados enviados, um filtro, e por fim um mapa de processos.

Com a análise de uma imagem colorida, podem-se ter 3 dimensões diferentes para analisar com um o detector conhecido como *kernel* que se moverá pelos pixels, este processo é conhecido como processo convolucional.

O *kernel* é aplicado em uma área da imagem produzindo uma pontuação, calculada de acordo com os valores adquiridos dos pixels e o filtro, após isso o filtro se move para análise uma nova série de pixels gerando assim outra pontuação desses novos pixels. Ao final de todo o processo de análise da imagem é gerado uma série de pontuações que são chamados de mapa convolucional, representado na Figura 6.

Figura 6 – Geração de uma pontuação



Fonte: IBM, 2020.

A segunda camada de um CNN é a Camada de Polimento, nesta camada é realizada uma redução de dimensionamento, assim reduzindo o número de dados enviados. Colocando um filtro em toda a imagem aplicando uma função de agregação aos valores gerando as pontuações necessárias.

Tendo dois tipos de polimento é possível classificar esta camada em Polimento Maximo, onde o filtro se move através dos dados enviados selecionando o pixel com o maior valor como pontuação. Polimento Médio é a segunda classificação de polimento desta camada, com o filtro se movendo através dos dados enviados ele calcula a média dos valores adquiridos para gerar sua pontuação.

Camada de Conexão Total é a última camada de uma CNN, tendo por sua vez a função de classificar os mapas convolucionais gerados pelas duas camadas anteriores utilizando uma função de ativação softmax para classificar apropriadamente os dados enviados, produzindo uma probabilidade de 0 a 1.

2.3 Python

Python é uma linguagem de programação de alto nível, possuindo uma sintaxe que seja mais simplista próxima a linguagem humana, publicada em 1991. Linguagem orientada a objetos altamente utilizada em áreas relacionadas à análise de dados, *machine learning*, IA e desenvolvimento *web*. Em áreas como a ciência, o Python é utilizado principalmente pelas suas bibliotecas matemáticas, NumPy e SciPy.

Tendo sua criação nos anos 90 pelo Guido Van Rossum, programador e matemático, a linguagem Python possui como objetivo, para aplicações simples ou até mesmo para aplicações complexas, a facilitação da escrita e compreensão com a utilização de um código limpo, simples e legível que priorize e facilite o trabalho do desenvolvedor.

Python usufrui de recursos como tipagem dinâmica e forte, orientação a objetos e multiparadigmas. Tendo seu código aberto e a disponibilidade gratuita de sua utilização para os desenvolvedores, podendo rodar em grande parte dos sistemas operacionais atualmente. Possuindo tal disponibilidade e código aberto acaba nos oferecendo uma grande e poderosa quantidade de bibliotecas padrões e frameworks que foram desenvolvidos pela própria comunidade.

Contendo uma sintaxe simples com uma exigência menor de elementos gramaticais como parêntese em alguns casos e a falta de ponto e vírgula no final de cada linha, se comparadas às linguagens como C++ ou PHP, sendo estruturado com base em espaços em branco. Possuindo uma versionabilidade que possibilita de ser utilizada como uma linguagem de script que executa ao lado de um servidor para aplicações web ou podendo ser uma ferramenta de plugins na qual pode expandir o funcionamento de outros programas.

Ela será usada no projeto com a finalidade de realizar o processamento das imagens capturadas pelo usuário e retornar um objeto JSON (*JavaScript Object Notation*) contendo os itens julgados como em falta na lista de compras.

Atualmente, o Python é uma linguagem baseada em versões, na qual a organização Python Software Foundation proporciona as novas versões, sendo a versão atual a 3.9, mas a fundação continua a dar suporte às versões antigas. Por

causa de todo esse suporte e foco no trabalho realizado pelos desenvolvedores a comunidade tem colocado o Python uma das linguagens mais populares dentro da comunidade de desenvolvedores.

Esta linguagem foi utilizada para realizar a Visão Computacional e *Machine Learning*, duas das áreas que o Python tem uma grande influência na Tecnologia da Informação atualmente, sendo utilizado juntamente com o YOLO (*You Only Look Once*), para detecção de objetos, e o *Darknet*, para o treinamento das redes neurais.

2.4 Darknet e YOLO

Darknet é um *framework* de rede neural desenvolvido com a linguagem C juntamente com CUDA (*Compute Unified Device Architecture*), que pode ser integrado com CPUs (*Central Process Unit*) e GPUs. Podendo ser utilizado em implementações de rede neural incluindo YOLO para detecção de imagens em tempo real, ImageNet Classification, RNNs entre outros.

YOLO (*You only look once*) é um sistema de detecção de objeto *single pass*, na qual utiliza uma rede neural convolucional para extrair as características das imagens, que pode processar imagens em até 30 FPS tendo uma mAP de 57,9%. mAP é uma métrica muito utilizada para avaliar a precisão de um sistema de detecção de objetos, analisando o quão preciso é suas previsões e quão positivo são suas previsões.

O funcionamento do Darknet integrado com o YOLO funciona aplicando uma única rede neural a imagem, a rede divide a imagem em regiões e prevê os limites das caixas de detecção e probabilidade para cada região. Nesse caso, as caixas possuem pesos de acordo com sua probabilidade prevista.

Pelo jeito que é realizado a análise das imagens, acaba gerando uma vantagem sobre o outros sistemas de detecção de imagem, já que olhando para a imagem inteira nos testes suas previsões são informadas utilizando um contexto global, além de fazer previsões com uma única avaliação da rede diferente de sistemas como R-CNN e Fast R-CNN fazendo com que tenha uma velocidade de análise muito maior.

2.4.1 Funcionamento do YOLO

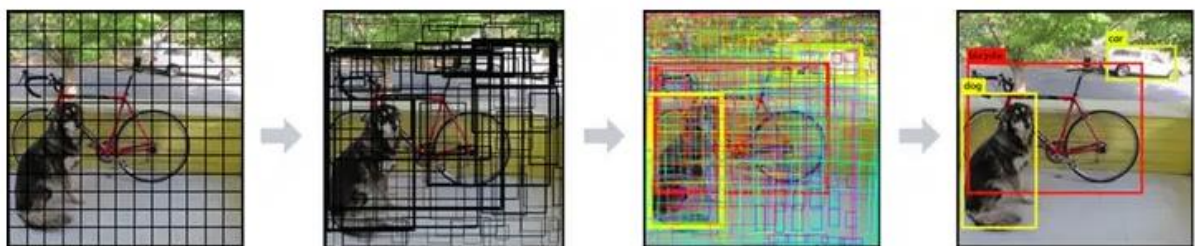
O tratamento de imagem implementado pelo YOLO é um problema de regressão. Podendo ser divididos em 4 passos até que a análise da imagem seja completada e os objetos sejam todos classificados e localizados.

Primeiramente, o sistema precisa dividir a imagem em um *grid* de $N \times N$ quadrantes, normalmente sendo dividida em 19×19 ou 13×13 quadrantes. Seguindo para o segundo passo onde os quadrantes criados são responsáveis pela predição de cinco *boundingbox*, caixas de delimitação que representam a localização e tamanho do objeto, juntamente com a confiança de se ter um objeto dentro do *boundingbox*.

Com os *boundingbox*'s realizados entra-se na fase três onde os quadrantes realizam a classificação, obtendo as classificações dos objetos encontrados dentro de suas *boundingbox*'s correspondentes a sua confiança.

Possuindo as classes é necessário realizar um filtro de confiabilidade nos *boundingbox*'s criados, já que a maioria deve corresponder com 30% ou menos de confiança. Nesse caso o valor de filtro, chamado de *threshold*, deve ser definido de acordo com o propósito do trabalho, neste trabalho utilizou-se 80% de *threshold* para a eliminação de *boundingbox*'s indevidos, como mostrado na Figura 7.

Figura 7 – Quatro passos do YOLO



Fonte: iaexpert.academy, 2020.

2.5 JavaScript

JavaScript, também conhecida como JS, é uma linguagem de programação interpretada usada no desenvolvimento *web* e fora do navegador em ambientes como o Node.js. Junto com o HTML (*HyperText Markup Language*) e o CSS (*Cascading Style Sheets*), ela é uma das 3 linguagens indispensáveis para se desenvolver uma página *web*. Utiliza-se o JavaScript para a programação *front-end*, permitindo que a interação com o usuário seja mais dinâmica.

2.6 Node.js

Quando é citado em Node.js, automaticamente é pensado em JavaScript. O JavaScript nasceu para atender as demandas de front, porém com o passar do tempo, foi-se tendo um crescimento tecnológico exponencial, fazendo com que surgisse a ideia de se ter uma mesma linguagem, porém para o lado do cliente e do servidor.

O Node é um ambiente de execução assíncrono, que se dá muito bem com servidores “*Single Threaded*”. Ele utiliza o interpretador JavaScript V8 usado no navegador Google Chrome, que é responsável pela tradução do código JavaScript em instruções de máquina. A partir dele, conseguem-se desenvolver aplicações web, APIs (*Application Programming Interface*) e até mesmo microsserviços. Com o uso do Node, pode-se ter uma única linguagem de programação para tratar requisições entre cliente e servidor.

Em meio a tantas vantagens, tendo algumas características específicas dele, dentre as principais podem-se colocar: Multiplataforma, permitindo criar aplicativos móveis, aplicativos para desktop etc. *Open Source*, código aberto, fazendo com que a pessoa tenha total acesso ao código fonte, permitindo-a realizar customizações e até contribuir com a comunidade. Escalável, desenvolvido para desenvolver aplicações escaláveis. Multi-Paradigma, permitindo ao usuário programar em diferentes paradigmas como, orientado a objetos, imperativo, funcional, dirigido a eventos etc.

2.7 React Native

Foi lançado em 2015 pelo Facebook sendo um projeto de código aberto. Com o passar dos tempos, foi se tornando e é atualmente uma das maiores soluções desenvolvidas para o desenvolvimento mobile. O React Native é um framework em JavaScript, como descrito anteriormente, utilizado especificamente para o desenvolvimento de aplicativos para celulares.

Dentre tantas vantagens, uma das maiores se não a maior vantagem de se usar esse *framework* é que ele é multiplataforma, ou seja, viabiliza que o código seja reaproveitado, eliminando-se a necessidade de desenvolver uma aplicação para cada um dos sistemas operacionais móveis mais usados na atualidade, o Android e o IOS. Pode-se dizer que, o React Native proporciona uma enorme economia de tempo e,

recursos, sendo assim, ela é considerada uma linguagem “Híbrida”, pois um código desenvolvido uma única vez, se adequa a várias plataformas.

Uma curiosidade muito importante sobre o React Native, é que ele foi desenvolvido com base no próprio React, uma biblioteca JavaScript que já era muito conhecida quando o framework mobile foi lançado. Resumindo, o React Native veio para revolucionar a vida dos *front-ends*, pois antigamente só podiam trabalhar com tecnologias baseadas na web, sendo assim, capacitando os mesmos para a elaboração de aplicativos robustos para plataformas móveis.

2.8 PostgreSQL

PostgreSQL é um SGBD (Sistema de Gerenciamento de Banco de Dados) relacional de objetos baseado na linguagem SQL (*Structured Query Language*) e é considerado um dos SGBDs *open source* mais robustos do mercado. Ele é compatível em diversos sistemas operacionais, como o Windows, Linux, Unix, FreeBSD, OpenBSD, NetBSD, Mac OS e Solaris. Um software de *open source* ou código aberto significa que qualquer usuário pode utilizá-lo, modificá-lo e distribuí-lo de acordo com as suas necessidades.

Entre suas características estão, por exemplo, consultas complexas, fácil acesso, SGBD que apresenta alta qualidade e robustez, chaves estrangeiras, indexação por texto, suporte ao modelo híbrido objeto relacional, entre outros (CAVALCANTE, 2021).

3 O aplicativo Lista Fácil

3.1 Requisitos

A definição dos requisitos de *software* tem um papel crucial dentro de um projeto, pois eles correspondem às funcionalidades, propriedades e restrições que são necessárias para alcançar o objetivo de um *software* (ROSSETTI, 2021).

Um dos principais motivos de falha ou reajuste em um software pode ser causada pela má definição dos requisitos, tanto funcionais quanto não funcionais de um sistema. Por isso é preciso ter um cuidado maior ao listar quais são os requisitos necessários para que um software seja desenvolvido sem precisar de ajustes futuros.

A seguir serão demonstrados os requisitos funcionais e não funcionais do sistema apresentado neste trabalho de conclusão de curso.

3.1.1 Requisitos funcionais

Ao definir os requisitos funcionais, deve-se levar em consideração algumas questões, como, por exemplo, o requisito deve ser específico sobre o que o sistema deve realizar, deve ser mensurável, deve ser atingível dentro do prazo definido, deve ter relevância para os objetivos de negócio e deve ser limitado no tempo para que seu acompanhamento de progresso seja realizável (VISURE, 2022).

A seguir serão apresentados alguns requisitos funcionais do aplicativo a ser desenvolvido.

- a) O aplicativo deve permitir o cadastro e exclusão de usuário, além da alteração de dados cadastrais;
- b) O aplicativo precisa realizar uma verificação de login e senha para acessar os dados do usuário e as funcionalidades do sistema.
- c) O aplicativo deve permitir a criação de uma lista de compra;
- d) O aplicativo deve identificar os objetos de uma foto enviada pelo usuário.
- e) O aplicativo deve permitir a finalização de uma lista de compra;
- f) O aplicativo deve permitir a consulta de listas de compras finalizadas anteriormente;

- g) O aplicativo deve criar um histórico de compras realizadas; e
- h) O aplicativo deve permitir a visualização do histórico de uma compra realizada.

3.1.2 Requisitos não funcionais

Requisitos não funcionais são aqueles que não estão relacionados diretamente com o desenvolvimento do sistema, eles ditam o comportamento do sistema em determinadas situações (NOLETO, 2020). A seguir serão apresentados alguns requisitos não funcionais do aplicativo a ser desenvolvido.

- a) O aplicativo deverá ser desenvolvido na linguagem de programação Python;
- b) O aplicativo deverá ser desenvolvido com as tecnologias de React Native;
- c) O aplicativo deverá ser desenvolvido com as tecnologias do Node.js;
- d) O aplicativo deverá se comunicar com o Sistema de Gerenciamento de Banco de Dados PostgreSQL;
- e) O aplicativo deverá ser multiplataforma, podendo ser executado tanto na plataforma Android quanto no iOS;
- f) O aplicativo deverá ter alta disponibilidade, funcionando 24 horas por dia, 7 dias por semana;
- g) O aplicativo deverá apresentar facilidade de uso; e
- h) A interface do aplicativo deve ser intuitiva, oferecendo facilidade de entendimento e manuseio.

3.2 MODELAGEM

Neste tópico serão abordados sobre o funcionamento do aplicativo e todas as interações que ocorrerão entre o sistema e o usuário para atingir determinados objetivos por meio do uso de casos de uso.

3.2.1 Requisitos do Usuário

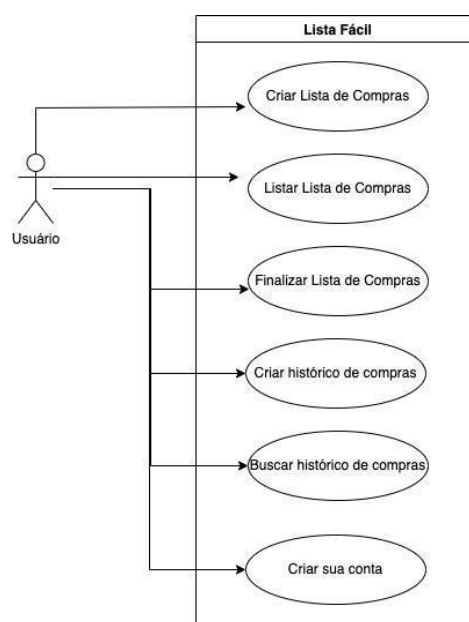
Para usufruir do sistema o usuário deve possuir um dispositivo de acesso para o sistema, sendo ele celular ou outro dispositivo. Também é necessário que o usuário possua um login e senha cadastrada no sistema para que possa ter acesso a todas as funcionalidades. Caso ele não possua, é possível que ele realize um cadastro para criar tais acessos.

3.2.2 Diagrama de caso de uso

Os casos de uso têm como objetivo facilitar o fluxo de funcionalidades de uma aplicação. Cada funcionalidade do sistema é representada por um caso de uso, dessa forma é possível criar uma abstração da implementação do sistema, conhecer cada cenário e cada fluxo de comportamento da aplicação. Para a criação dos casos de uso é necessário estabelecer os atores das funcionalidades, os atores podem ser departamentos, pessoas e até mesmo outro serviço externo que vá interagir diretamente com o sistema, é válido lembrar que um ator representa um papel e não um usuário específico do sistema.

A Figura 8 a seguir apresenta todas as funcionalidades que um usuário pode executar dentro da aplicação.

Figura 8 – Diagrama com todas as funcionalidades do Sistema



Fonte: Autoria Própria, 2022.

A Figura 9 a seguir representa o diagrama de caso de uso “Criar Lista de Compras”, que ilustra resumidamente as partes envolvidas no processo de criação de uma lista de compras.

Figura 9 – Diagrama de caso de uso “Criar Lista de Compras”



Fonte: Autoria Própria, 2022.

A Tabela 1 a seguir mostra o caso de uso “Criar Lista de Compras”, explicando detalhadamente a maneira como ocorre o processo de criação da lista de compras.

Tabela 1 – Caso de Uso “Criar Lista de Compras”

Nome do Caso de Uso	Criar Lista de Compras
Ator Principal	Usuário
Atores Secundários	Sistema
Resumo	Este caso de uso descreve as etapas necessárias para criar uma lista de compras
Pré-Condições	O usuário deve ter uma conta no Sistema
Pós-Condições	Nenhuma
Fluxo Básico	<ol style="list-style-type: none"> 1. Solicitar a Criação de uma Nova Lista de Compras 2. Selecionar uma Compra anterior 3. Enviar uma imagem dos itens que ele ainda possui na sua dispensa 4. Confirmar a lista de compras gerada pelo sistema
Fluxo Alternativo	<p>Sem Compra Anterior</p> <p>2.1 Se o usuário não tiver uma compra anterior cadastrada ele deverá digitar a sua lista de compra no aplicativo, se ele quiser Fim do Fluxo</p>

Restrições e validações	<ol style="list-style-type: none"> 1. A Lista de Compra só pode ser criada se o usuário estiver conectado no sistema 2. O usuário só pode buscar as listas de compras criadas por ele
-------------------------	---

Fonte: Autoria Própria, 2022.

A Figura 10 a seguir representa o diagrama de caso de uso “Listar Lista de Compra”, que tem a função de demonstrar quem são as partes envolvidas no processo de listagem de listas de compras.

Figura 10 – Diagrama de caso de uso “Listar Lista de Compra”



Fonte: Autoria Própria, 2022.

A Tabela 2 a seguir representa o caso de uso “Listar Lista de Compras”, explicando detalhadamente a maneira como ocorre o processo de listagem de lista de compras.

Tabela 2 – Caso de Uso “Listar Lista de Compras”

Nome do Caso de Uso	Listar Lista de Compras
Ator Principal	Usuário
Atores Secundários	Sistema
Resumo	Este caso de uso descreve as etapas necessárias para listar todas as listas de compras do usuário
Pré-Condições	O usuário deve ter uma conta no Sistema
Pós-Condições	Nenhuma
Fluxo Básico	<ol style="list-style-type: none"> 1. Solicitar a Listagem de Todas as Listas de Compras

	2. Retornar uma Listagem com todas as listas de compras, possuindo nome da lista, data de criação e quantidade de produtos
Fluxo Alternativo	Sem Lista de Compra 2.1 Se o usuário não possuir uma lista de compras deve ser retornado a seguinte mensagem: “Você não possui uma Lista de Compras”
Restrições e validações	1. O usuário só pode buscar as listas de compras criadas por ele

Fonte: Autoria Própria, 2022.

A Figura 11 a seguir representa o diagrama de caso de uso “Finalizar Lista de Compras”, que tem a função de demonstrar quem são as partes envolvidas no processo de finalização de lista de compras.

Figura 11 – Diagrama de caso de uso “Finalizar Lista de Compras”



Fonte: Autoria Própria, 2022.

A Tabela 3 a seguir representa o caso de uso “Finalizar Lista de Compras”, explicando detalhadamente a maneira como ocorre o processo de finalização de lista de compras.

Tabela 3 – Caso de Uso “Finalizar Lista de Compras”

Nome do Caso de Uso	Finalizar Lista de Compras
Ator Principal	Usuário
Atores Secundários	Sistema
Resumo	Este caso de uso descreve as etapas necessárias para o usuário finalizar uma lista de compras

Pré-Condições	O usuário deve ter uma conta no Sistema Todos os itens da lista devem estar selecionados
Pós-Condições	Nenhuma
Fluxo Básico	<ol style="list-style-type: none"> 1. Solicitar a Listagem de Todas as Listas de Compras 2. Retornar uma Listagem com todas as listas de compras, possuindo nome da lista, data de criação e quantidade de produtos 3. Escolher uma Lista de Compra 4. Selecionar todos os itens da Lista de Compra 5. Selecionar a opção de finalizar Lista de Compra
Fluxo Alternativo	<p>Sem Lista de Compra</p> <p>2.1 Se o usuário não possuir uma lista de compras deve ser retornado a seguinte mensagem: “Você não possui uma Lista de Compras”</p>
Fluxo Alternativo	<p>Não selecionar todos os itens</p> <p>4.1 Caso o usuário não tenha selecionado todos os itens, ele pode finalizar a lista de compra, mas será retornado a seguinte mensagem: “Alguns itens não foram selecionados, deseja finalizar esta lista de compras?”</p>
Restrições e validações	<ol style="list-style-type: none"> 1. O usuário só pode buscar as listas de compras criadas por ele

Fonte: Autoria Própria, 2022.

A Figura 12 a seguir representa o diagrama de caso de uso “Criar Histórico de Compras”, que tem a função de demonstrar quem são as partes envolvidas no processo de criação de histórico de compras.

Figura 12 – Diagrama de caso de uso “Criar Histórico de Compras”

Fonte: Autoria Própria, 2022.

A Tabela 4 a seguir representa o caso de uso “Criar Histórico de Compras”, explicando detalhadamente a maneira como ocorre o processo de criação de histórico de compras.

Tabela 4 – Caso de Uso “Criar Histórico de Compras”

Nome do Caso de Uso	Criar Histórico de Compra
Ator Principal	Usuário
Atores Secundários	Sistema
Resumo	Este caso de uso descreve as etapas necessárias para o usuário criar um histórico de compra
Pré-Condições	O usuário deve ter uma conta no Sistema
Pós-Condições	Nenhuma
Fluxo Básico	<ol style="list-style-type: none"> 1. Solicitar a Finalização de uma Compra 2. Enviar uma ou várias fotos dos produtos comprados 3. Confirmar se todos os itens estão listados 4. Enviar Lista para ser salvo como uma compra
Fluxo Alternativo	<p>Itens Faltando na lista</p> <p>3.1 Caso a aplicação não tenha reconhecido todos os itens da compra, é possível do usuário adicionar na lista de forma manual</p>

Restrições e validações	2. O usuário precisa enviar uma foto de suas compras para que o aplicativo possa realizar o reconhecimento de imagem
-------------------------	--

Fonte: Autoria Própria, 2022.

A Figura 13 a seguir representa o diagrama de caso de uso “Buscar Histórico de Compras”, que tem a função de demonstrar quem são as partes envolvidas no processo de busca de histórico de compras.

Figura 13 – Diagrama de caso de uso “Buscar Histórico de Compras”



Fonte: Autoria Própria, 2022.

A Tabela 5 a seguir representa o caso de uso “Buscar Histórico de Compras”, explicando detalhadamente a maneira como ocorre o processo de busca de histórico de compras.

Tabela 5 – Caso de Uso “Buscar Histórico de Compras”

Nome do Caso de Uso	Buscar Histórico de Compras
Ator Principal	Usuário
Atores Secundários	Sistema
Resumo	Este caso de uso descreve as etapas necessárias para buscas todas as compras do usuário
Pré-Condições	O usuário deve ter uma conta no Sistema O usuário deve ter cadastrado uma compra
Pós-Condições	Nenhuma
Fluxo Básico	1. Solicitar a Listagem de Todas as Compras

	2. Retornar uma Listagem com todas as compras, possuindo nome, data da compra e quantidade de produtos
Fluxo Alternativo	Sem Lista de Compra 2.1 Se o usuário não possuir uma compra deve ser retornado a seguinte mensagem: “Você não possui Compras”
Restrições e validações	1. O usuário só pode buscar as compras criadas por ele

Fonte: Autoria Própria, 2022.

A Figura 14 a seguir representa o diagrama de caso de uso “Criar Conta”, que tem a função de demonstrar quem são as partes envolvidas no processo de criação de uma conta.

Figura 14 – Diagrama de caso de uso “Criar Conta”



Fonte: Autoria Própria, 2022.

A Tabela 6 a seguir representa o caso de uso “Criar Conta”, explicando detalhadamente a maneira como ocorre o processo de criação de uma conta.

Tabela 6 – Caso de Uso “Criar Conta”

Nome do Caso de Uso	Criar conta
Ator Principal	Usuário
Atores Secundários	Sistema
Resumo	Este caso de uso descreve as etapas necessárias para usuário criar sua conta no sistema
Pré-Condições	O usuário deve ter instalado o aplicativo

Pós-Condições	Nenhuma
Fluxo Básico	1. Preencher o formulário de cadastro 2. Clicar no link de confirmação enviado por e-mail
Fluxo Alternativo	Nenhum
Restrições e validações	Nenhuma

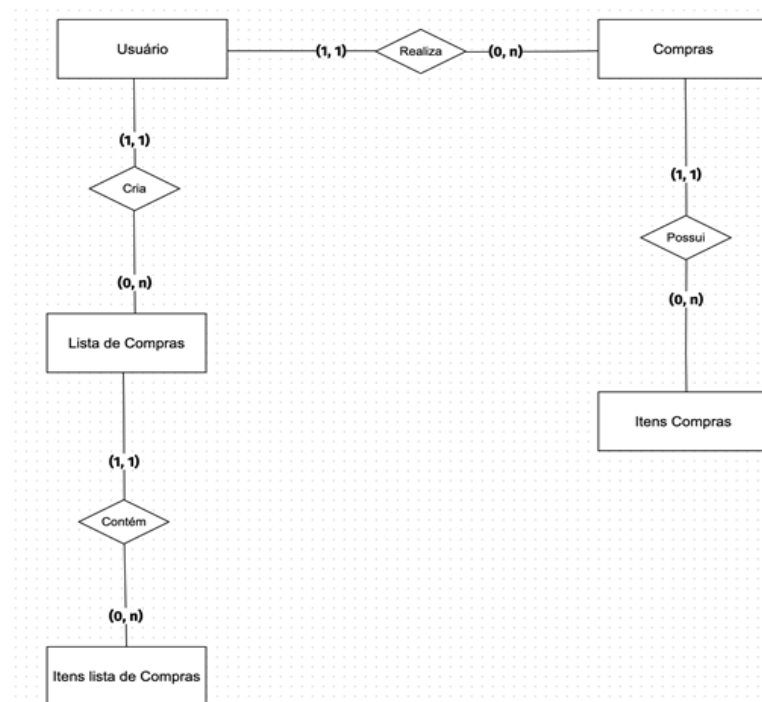
Fonte: Autoria Própria, 2022.

3.2.3 Modelo Conceitual

O Modelo Entidade Relacionamento (MER) possui dois tipos de modelagem, sendo o modelo conceitual e o modelo lógico.

O modelo conceitual tem como principal objetivo apresentar de forma abstrata o fluxo de relações das entidades apresentadas do projeto para que a equipe consiga levantar alguns comportamentos e requisitos da aplicação. Abaixo está o Modelo Conceitual da Lista Fácil, como demonstrado na Figura 15.

Figura 15 – Modelo Conceitual



Fonte: Autoria Própria, 2022.

Conforme o diagrama supracitado, as definições das relações da entidade são:

- a. Usuário REALIZA Compra: Um usuário cadastrado na plataforma pode realizar um ou várias compras no sistema.
- b. Compra POSSUI Itens da Compra: Uma compra criada na plataforma pode possuir 1 ou vários itens.
- c. Usuário CRIA Lista de Compras: Um usuário pode criar uma ou diversas listas de compras dentro da plataforma.
- d. Lista de Compras CONTÉM Itens Lista de Compras: Uma lista de compras pode conter 1 ou diversos itens

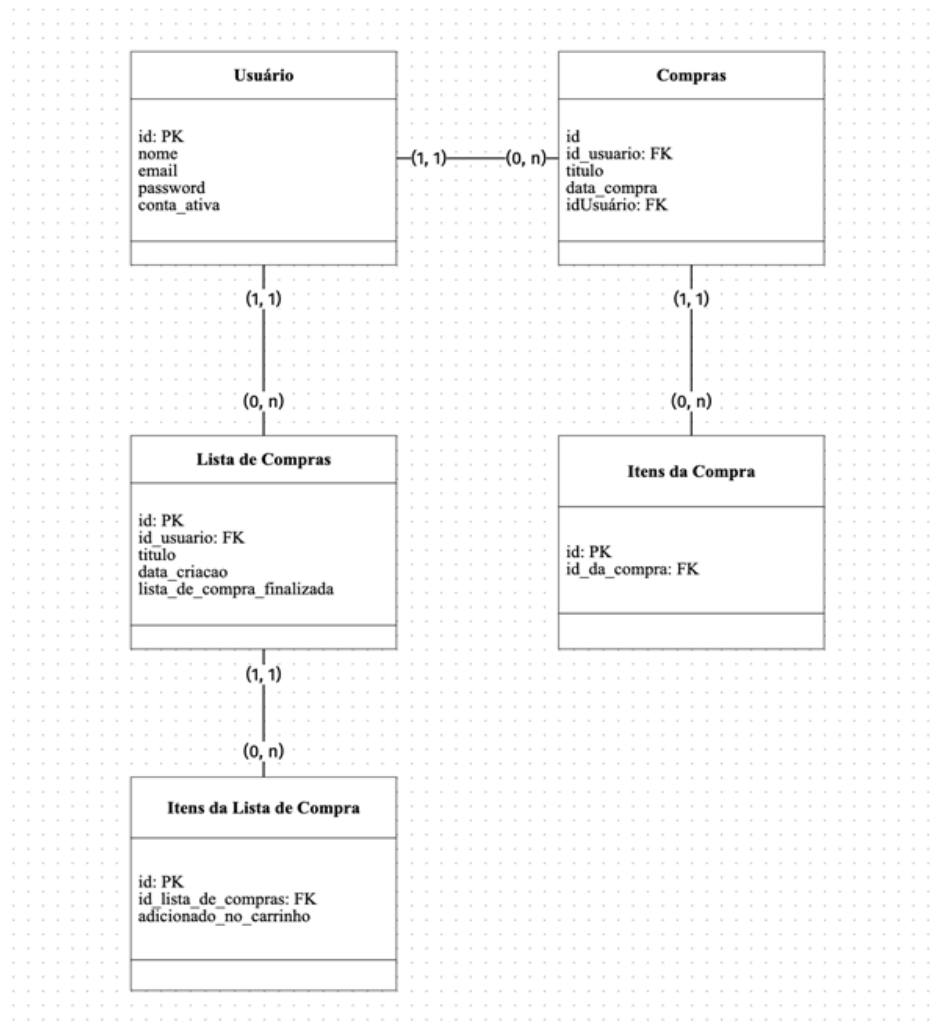
3.2.4 Modelo Lógico

O Modelo Lógico é muito parecido com o Modelo Conceitual, ambos apresentam as entidades e as suas relações, porém, o modelo lógico conta com os atributos já tipados.

Com o modelo lógico é possível definir se os campos serão do tipo CHAR, VARCHAR, INT e BOOLEAN. Neste modelo também é possível definir quais serão as Chaves Primárias (*Primary Keys*) que são representadas abaixo como PK e Chaves Estrangeiras (*Foreign Keys*).

Uma vantagem de ter o modelo lógico bem escrito é que a maioria das ferramentas possuem suporte para que seja gerado o código SQL do modelo, dessa forma é possível realizar a criação do seu banco de dados a partir do modelo lógico. Segue abaixo o diagrama do modelo lógico da lista fácil, como mostrado na Figura 16.

Figura 16 – Modelo Lógico



Fonte: Autoria Própria, 2022.

3.3 FRONT-END

Tendo em vista que o público-alvo do aplicativo abrange desde jovens, adultos até pessoas da terceira idade, a construção das telas do aplicativo levou em consideração, como fator principal, a experiência do usuário, implementando-se uma interface simples e agradável, prezando pela fácil usabilidade e entendimento do sistema.

Visando esse objetivo, o sistema foi desenvolvido com poucas informações na tela, evitando a poluição visual e confusão entre os utilizadores, prezando por uma navegação mais intuitiva e eficaz entre as páginas do aplicativo, permitindo a extração de todas as funcionalidades do sistema com o mínimo de interações possíveis.

3.3.1 Logotipo

Com o objetivo do aplicativo a assistência e facilitação tanto da montagem quanto da realização de listas de compras foi feita a escolha de uma logo na qual lembrasse o usuário do objetivo desse aplicativo, conforme apresentado na Figura 17, sendo escolhido um carrinho de compras com traços simplistas.

Visto que a aplicação tem o objetivo de facilitar tanto a criação quanto a realização de listas de compras, escolheu-se um logotipo com traços minimalistas, visando fazer o usuário assimilá-lo com o objetivo do aplicativo.

Figura 17 – Logotipo do aplicativo



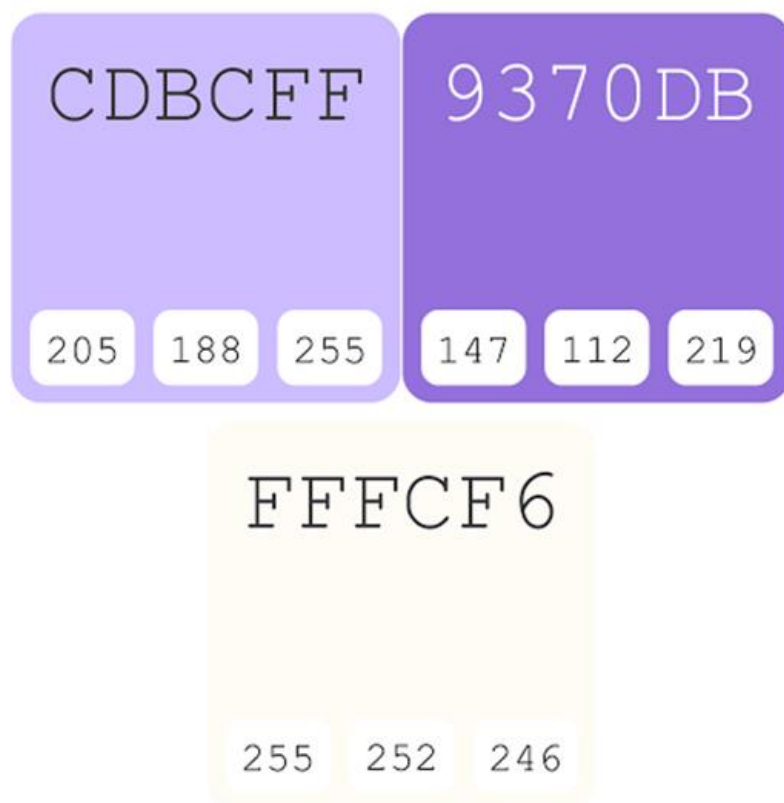
Fonte: pngwing¹.

¹ Disponível em <<https://www.pngwing.com/pt/free-png-sccrq>>. Acesso em 03 de setembro de 2022.

3.3.2 Cores

Selecionado uma palheta de cores claras, trazendo uma tonalidade mais sutil para o sistema, foi escolhido tons de violeta mais claros na qual traz um sentimento de prosperidade e relaxamento, combinado com um fundo bege claro seguindo com o propósito de extrair um sentimento de tranquilidade, conforme na Figura 18, com isso criou-se telas que sejam amigáveis com a vista e agradáveis ao usuário.

Figura 18 – Paleta de cores



Fonte: Autoria Própria, 2022.

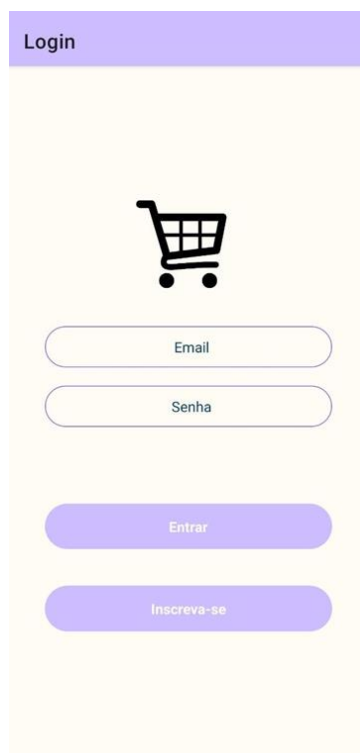
3.3.3 Telas

Nesta seção serão apresentadas todas as telas em que o usuário poderá navegar dentro do *software* desenvolvido. Podem-se citar, por exemplo, a tela de login, de cadastro, de lista de compras e compras realizadas, de câmera e assim por diante.

3.3.4 Login

A tela de login do aplicativo, representado na Figura 19, segue o padrão da maioria das telas de login de outros sistemas, contendo dois campos principais: e-mail e senha na qual o usuário deve informar suas credenciais.

Figura 19 – Tela de login

A imagem mostra a interface de login de um aplicativo. No topo, há uma barra de cabeçalho roxa com o texto "Login" em branco. Abaixo, no centro, há um ícone de carrinho de compras preto. Seguem-se dois campos de entrada de texto brancos com bordas arredondadas e cinzas: o primeiro rotulado "Email" e o segundo rotulado "Senha". Abaixo desses campos, há dois botões de ação roxos com bordas arredondadas e texto branco: o primeiro rotulado "Entrar" e o segundo rotulado "Inscreva-se". O fundo da tela é amarelo claro.

Fonte: Autoria Própria, 2022.

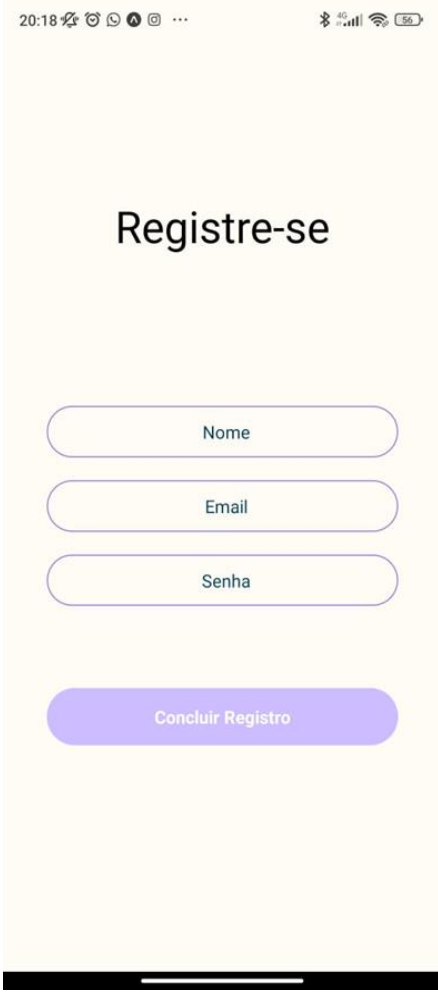
Abaixo é possível visualizar mais dois botões, “Entrar” e “Inscreva-se”, onde o botão de entrar realiza a busca no banco de dados para finalizar a validação da senha e após a autenticação é gerado um token para o intervalo de uma hora para manipulação de informações no ambiente interno da plataforma. E o botão de inscreva-se faz o direcionamento para a tela de cadastro de novos usuários para a plataforma.

3.3.5 Cadastro

Desenvolvido para cadastro do sistema, foi criado a tela de registro da plataforma, representado na Figura 20, assim trazendo o primeiro contato do usuário

com a plataforma. Composta por 3 campos de texto, onde o usuário necessita informar para realizar o cadastro na plataforma.

Figura 20 – Tela de cadastro

A imagem mostra a interface de uma tela de cadastro em um aplicativo móvel. No topo, há uma barra de status com o horário 20:18, ícones de notificação e status de conexão (Bluetooth, 4G, Wi-Fi, bateria). O título centralizado é "Registre-se". Abaixo dele, há três campos de entrada de texto, cada um com uma borda arredondada e um rótulo interno: "Nome", "Email" e "Senha". Abaixo dos campos, há um botão de ação arredondado com o texto "Concluir Registro". A interface tem um fundo amarelo claro e uma barra de navegação preta no rodapé.

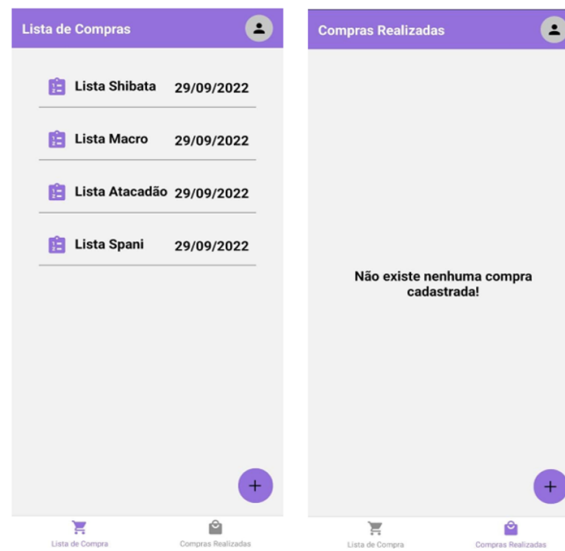
Fonte: Autoria Própria, 2022.

Os campos são compostos por 3 informações, sendo elas nome, e-mail, e senha, onde é feita a validação do e-mail, verificando se existe o "@" na informação passada pelo usuário e a validação da senha, não podendo ter mais que 8 caracteres. Abaixo é possível encontrar o botão de "Concluir Registro", onde é feita a validação e envio das informações para o *back end*.

3.3.6 Lista de compra e compras realizadas

Desenvolvido para apresentar as listas de compras e as compras realizadas cadastradas pelo usuário, a tela representada na Figura 21, traz como informações nome e a data de cadastro e lista em ordem de cadastro decrescente.

Figura 21 – Telas de lista de compra e compras realizadas



Fonte: Autoria Própria, 2022.

No topo da tela é possível encontrar o nome da tela e um botão para acessar o perfil de usuário. Abaixo, é listado todas as listas cadastradas pelo usuário na plataforma, mostrando o nome da lista e a data de cadastro. Na parte de baixo da tela, contém um TabMenu, que dá acesso às duas telas, tanto lista de compra, como compras realizadas e o botão de adicionar, onde abre um modal para o cadastro de uma nova lista de compras, representado na Figura 22.

Figura 22 – Modal de adicionar lista



Fonte: Autoria Própria, 2022.

É trazido um nome padrão para criação da lista, no qual é permitido ser alterado pelo usuário, caso seja desejado. Abaixo do campo de texto do nome da lista, existe um *checkbox*, que por padrão é selecionado. Nele é definido se o usuário deseja criar uma lista a partir do reconhecimento de imagem de uma foto que será tirada por ele, caso ele queira incluir todos os produtos manualmente, basta ele não selecionar que será encaminhado para uma tela de cadastro de produtos.

Por fim, no modal contém um botão de cancelar, para não criar a lista de compra ou adicionar para ser encaminhado para a tela de cadastro ou tela da câmera para tirar uma foto.

3.3.7 Câmera

Desenvolvida para permitir que o usuário faça o enquadramento e fotografe os produtos ao quais ele deseja registrar na lista de compra ou nas compras realizadas, dentro da plataforma, representada na Figura 23. Após realizar o primeiro acesso no aplicativo, é solicitado a permissão para acessar a câmera do dispositivo, caso seja recusado, a tela em questão não poderá ser utilizada.

Figura 23 – Tela da câmera



Fonte: Autoria Própria, 2022.

A tela contém o enquadramento da foto em resolução 3 x 4 para visualização do usuário, seguido de 3 botões de ação. O primeiro botão realiza a troca da câmera traseira pela câmera frontal do dispositivo, permitindo alterá-la quando necessário. O segundo botão seria o de captura, onde quando apertado ele registra a foto que está sendo enquadrada naquele determinado momento, abrindo um modal de visualização, conforme a Figura 24. Já o terceiro botão é utilizado para acender o flash da câmera para registrar fotos com melhor visualização.

Figura 24 – Modal de visualização



Fonte: Autoria Própria, 2022.

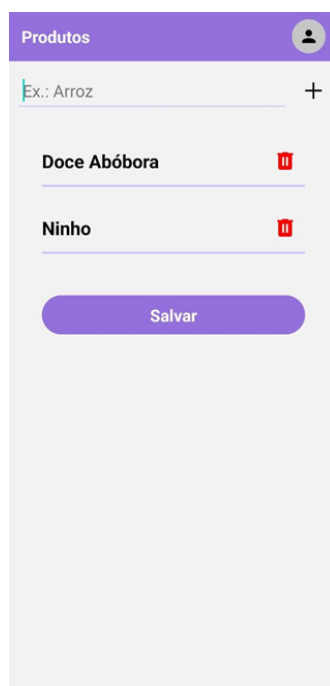
No modal é possível visualizar a imagem registrada pela tela de câmera ao usuário realizar a fotografia. No topo da tela, existe um botão para fechar o modal e retornar à tela da câmera, sendo assim, cancelada a foto registrada. No centro, é possível enxergar a foto registrada pelo usuário. E na parte de baixo da tela, contém o botão de confirmação, para enviar a foto para processamento e retorno do reconhecimento dos itens que existem nela.

Após a confirmação do usuário, a fotografia é enviada ao *back-end* para processamento e retorna uma lista com todos os produtos identificados. Após o sucesso do processamento, o usuário é encaminhado para a tela de adicionar produtos à compra.

3.3.8 Adicionar produtos à compra

Desenvolvida para adicionar os produtos à lista de compra ou em compras realizadas, ela tem como objetivo, mostrar os produtos reconhecidos pelo sistema e permitir removê-los ou adicionar novos produtos a lista apresentada, conforme representada na Figura 25.

Figura 25 – Tela de adicionar produtos



Fonte: Autoria Própria, 2022.

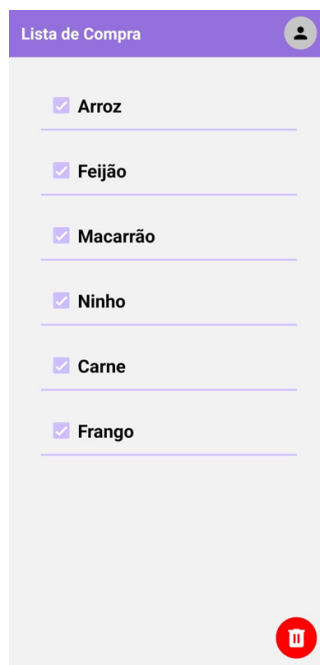
Nessa tela, é possível digitar o nome do produto que deseja adicionar à lista no campo de texto localizado na parte superior da tela e pressionar o botão de “+” para incluí-lo aos demais produtos. Abaixo é possível visualizar todos os nomes dos produtos presentes na lista acompanhado de um botão vermelho com o ícone de lixeira, que permite a remoção do item da lista. Após o fim da lista, contém o botão de salvar, onde é enviado e registrado no banco de dados da aplicação.

3.3.9 Checagem de produtos da lista

Desenvolvida para permitir que o usuário registre quais produtos já foram pegos durante o seu tempo de compras. Permitindo que o usuário possa marcar com

um *check* quais produtos já foram pegos e verificar quais itens ainda faltam ser comprados, conforme representado na Figura 26.

Figura 26 – Tela de checar produtos



Fonte: Autoria Própria, 2022.

Nessa tela, é listado todos os produtos que foram cadastrados na lista de compras selecionada pelo usuário na tela de listas de compras. No centro da tela contém todos os produtos acompanhados de um *checkbox* para permitir que o usuário marque ou desmarque, informando se o produto foi ou não pego durante o período de compras. No fim da tela, contém também um botão vermelho para exclusão da lista, caso o usuário deseje remover a lista completa da plataforma.

3.3.10 Desenvolvimento

Pensando em um desenvolvimento rápido e com maior abrangência de público-alvo, foi escolhido desenvolver o *front-end* em JavaScript, utilizando o *framework* React Native, devido a facilidade de poder implementar o mesmo código tanto para plataformas Android e IOS. Outra vantagem desse framework é a facilidade e rapidez de codificação, também por ser muito famoso e utilizado nas áreas empresariais,

fazendo com que tenha um suporte e apoio muito grande por parte da comunidade de desenvolvedores.

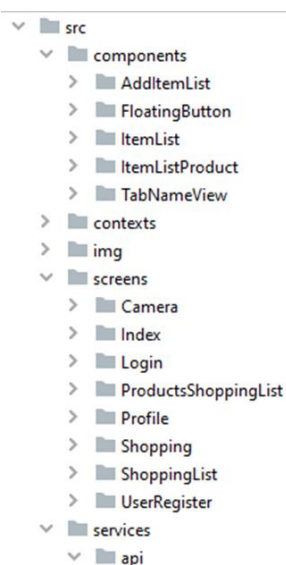
3.3.10.1 Estrutura de arquivos

O React Native é baseado em uma estrutura de aproveitamento de códigos através de componentes, sendo assim, é possível criar parte do código como componente e chamá-lo em qualquer tela pertencente ao seu aplicativo.

Com isso, a estruturação de pastas foi dividida em 5 partições, conforme representado na Figura 27, sendo elas:

- Components: Nela contém todos os componentes criados e utilizados dentro do sistema;
- Contexts: Nela contém todas as classes criadas para gerir as informações gerada pelo usuário ao utilizar a plataforma;
- Img: Armazena todas as imagens padrões que são utilizadas na aplicação;
- Screens: Nela contém todas as telas existentes na aplicação; e
- Services: Nela contém todos os scripts de consulta e busca externa, tais como chamadas de APIs.

Figura 27 – Estrutura de pastas



Fonte: Autoria Própria, 2022.

Cada componente ou tela possui um arquivo de estilização, onde é possível encontrar todos os atributos de estilos utilizados na tela que ele é chamado. E também possui um arquivo de codificação, onde se encontram os componentes chamados para compor a tela que será visualizada, acompanhado das variáveis e as funções utilizadas nessa mesma tela.

3.3.10.2 Códigos

Neste tópico serão apresentados os códigos do projeto desenvolvido.

3.3.10.3 Sistema de navegação

Para se navegar entre as telas, foi adotado a navegação em Stack, representada na Figura 28, na qual permite sobrepor uma tela filha na frente da tela atual, permitindo que o usuário consiga voltar à tela anterior sem perder os dados inseridos.

Figura 28 – Navegação Stack

```
<Stack.Navigator initialRouteName="TabMenu">
  <Stack.Screen
    name="Perfil"
    component={Profile}
    options={{
      headerShown: false,
    }}
  />
  <Stack.Screen
    name="Camera"
    component={CameraView}
    options={{
      headerShown: false,
    }}
  />
  <Stack.Screen
    name="ProductsShoppingList"
    component={ProductsShoppingList}
    options={{
      headerShown: false,
    }}
  />
  <Stack.Screen
    name="TabMenu"
    component={Tabs}
    options={{
      headerShown: false,
    }}
  />
</Stack.Navigator>
```

Fonte: Autoria Própria, 2022.

Também foi adotado a navegação em Tab, representada na Figura 29, onde é criado um menu na parte inferior da tela em forma de aba permitindo o usuário navegar entre os módulos existentes dentro da plataforma.

Figura 29 – Navegação Tab

```
<Tab.Navigator initialRouteName="Lista de Compra">
  <Tab.Screen
    name="Lista de Compra"
    component={ShoppingList}
    options={{
      tabBarActiveTintColor: "#9370DB",
      tabBarIcon: ({size : number , color : string }) => (<MaterialCommunityIcons name="cart" color={color} size={size} />),
      headerShown: false,
    }}
  />
  <Tab.Screen
    name="Compras Realizadas"
    component={Shopping}
    color
    options={{
      tabBarActiveTintColor: "#9370DB",
      tabBarIcon: ({size : number , color : string }) => (<MaterialCommunityIcons name="shopping" color={color} size={size} />),
      headerShown: false,
    }}
  />
</Tab.Navigator>
```

Fonte: Autoria Própria, 2022.

3.3.10.4 Hooks

São funções internas que permitem utilizar métodos dentro dos componentes do React Native, podendo ser rodados através de uma mudança de estado ou ação do usuário, fazendo com que diminuísse a complexidade de desenvolvimento do código, representados na Figura 30.

Figura 30 – Hooks

```
const [hasPermission, setHasPermission] = useState( initialState: null);

useEffect( effect: () => {
  (async () => {
    const { status } = await Camera.requestCameraPermissionsAsync();
    setHasPermission( value: status === "granted");
  })();
}, deps: []);
```

Fonte: Autoria Própria, 2022.

Dentre todos os *hooks* que existem, foram utilizados 4 *hooks* principais na aplicação, sendo eles:

- *useState*: Permite rastrear o estado de uma propriedade dentro de um componente na aplicação. Assim recarregando todas as partes do código que está referenciada a ele.
- *useContext*: É utilizado para rastrear o estado de uma variável ou função globalmente dentro da aplicação. Permitindo que o desenvolvedor consiga chamar as variáveis ou funções apresentadas a ele em qualquer parte do código.
- *useRef*: Permite que o desenvolvedor inclua uma referência a um componente. Assim conseguindo chamá-lo com maior facilidade.
- *useEffect*: É um gancho onde permite que o desenvolvedor execute funções ou blocos de código a partir de uma alteração de estado de uma determinada propriedade.

3.3.10.5 Componentes nativos

São componentes padrões do React Native no qual permite renderizar um elemento dentro da tela. O React Native disponibiliza muitos componentes por padrão em sua biblioteca, dentre eles, é possível citar alguns que foram utilizados na aplicação desenvolvida, conforme mostrado na Figura 31 abaixo.

Figura 31 – Componentes nativos

```

<Modal
  animationType={"slide"}
  transparent={true}
  visible={openModal}
  style={styles.screenModal}
>
  <View style={styles.screenModal}>
    <View style={styles.contentModal}>
      <Image
        style={styles.imgPhoto}
        source={{uri: capturedFoto}}
      />
    </View>
    <TouchableOpacity
      style={styles.closeButton}
      onPress={() => {setOpenModal( value: false)}}
    >
      <FontAwesome name="close" size={50} color={"#ffffff"} />
    </TouchableOpacity>
    <TouchableOpacity
      style={styles.nextButton}
      onPress={() => {console.log("Envia a img para o python")}}
    >
      <FontAwesome name="check" size={50} color={"#ffffff"} />
    </TouchableOpacity>
  </View>
</Modal>

```

Fonte: Autoria Própria, 2022.

É possível observar que é disposto dentro do componente *Modal*, um componente *Image* no qual renderiza uma imagem passada como parâmetro e seguido de dois componentes *TouchableOpacity*, no qual representa um botão para representar uma ação passada como propriedade para ele. E por fim, o componente *View*, que permite uma melhor distribuição dos itens na tela, permitindo que possa encaixá-lo em qualquer parte da tela.

3.3.10.6 Componentes personalizados

Os componentes personalizados, representado na Figura 32, é um conjunto de componentes nativos ou personalizados, utilizados para reaproveitamento de código. Podem ser chamados por outros componentes personalizados ou por uma própria tela, fazendo com que o código fique mais limpo e mais organizado.

Figura 32 – Componentes personalizados

```

export default function TabNameView(props){
  const {navigation} = props;
  return (
    <View style={styles.content}>
      <Text style={styles.textNameView}>{props.nameView}</Text>
      {typeof navigation == "undefined" ? (
        <Text></Text>
      ) : (
        <TouchableOpacity
          style={styles.contentProfile}
          onPress={() => {
            navigation.navigate( args: "Perfil");
          }}
        >
          <MaterialCommunityIcons name={"account"} color={"#000000"} size={24} />
        </TouchableOpacity>
      )}
    </View>
  )
}

```

Fonte: Autoria Própria, 2022.

3.3.10.7 Estilização

Todas as telas e componentes possuem seu próprio arquivo de estilização, representado na Figura 33, onde nele é inserido os códigos referentes ao estilo do elemento declarado no arquivo de codificação.

Figura 33 – Arquivo de estilização

```
import { StyleSheet } from "react-native";

const styles = StyleSheet.create({
  content: {
    paddingTop: 10,
    height: 60,
    width: "100%",
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center"
  },
  textInput: {
    fontSize: 18,
    padding: 5,
    borderBottomColor: "#CDBCFE",
    borderBottomWidth: 1,
    marginHorizontal: "3%",
    width: "81%",
  },
  addButton: {
    padding: 5,
    width: "10%",
    marginRight: "3%",
  },
});

export default styles
```

Fonte: Autoria Própria, 2022.

O arquivo é importado e inicializado como propriedade e inserido ao seu respectivo elemento para que sejam aplicados os atributos de estilização.

3.3.10.8 Funções

As funções, representada na Figura 34, são blocos de códigos definidos para executar uma ação ou retornar uma informação ou resultado.

Figura 34 – Funções

```

const create = async () => {
  let result = await addShoppingList(token, title);
  if (typeof result["id"] != "undefined") {
    createList(result["id"]);
    if (isIntelligentList) {
      navigation.navigate( args: "Camera");
      setOpenModal( value: false);
    } else {
      navigation.navigate( args: "ProductsShoppingList");
      setOpenModal( value: false);
    }
  } else {
    ToastAndroid.show( message: "Não foi possível criar a lista de compra!", ToastAndroid.SHORT);
  }
}

```

Fonte: Autoria Própria, 2022.

Nessa função é possível observar que realiza o cadastramento da lista de compra e encaminha o usuário para a tela de lista de produtos, caso o *check* não esteja selecionado ou para a tela de fotografia, para o usuário conseguir retirar as fotos dos produtos.

3.3.10.9 Contextos

As *Context* API, representada na Figura 35, é uma facilitadora de manipulação de dados dentro do React Native, onde ela permite que armazene as informações em uma variável definida previamente nela e permite acessá-la em outras partes do código, sem que seja passado por *props*.

Figura 35 – Ciclo de dados com a *context* API

Fonte: DevMedia, 2021.

Supondo que existam 3 componentes, onde o primeiro chame o segundo e o segundo chame o terceiro, e que o desenvolvedor precise acessar um dado que foi gerado no primeiro componente e mostrá-lo no terceiro componente, ao invés de passar esse dado como *props* para o segundo e repassar para o terceiro, ele apenas inclui esse dado que foi declarado inicialmente na *context* API, conforme representado na Figura 36, e acessá-lo no terceiro componente.

Figura 36 – Context API

```
export const AuthContext = createContext( {defaultValue: {}});

function AuthProvider({children}) {

  const [tokenId, setTokenId] = useState( initialState: "");

  function sessionLogin(token) {
    if (token !== '') {
      setTokenId(token);
    }
  }

  function sessionLogout(token) {
    if (token !== '' && token === tokenId) {
      setTokenId( value: "");
    }
  }

  return(
    <AuthContext.Provider value={{token: tokenId, sessionLogin, sessionLogout}}>
      {children}
    </AuthContext.Provider>
  )
}

export default AuthProvider;
```

Fonte: Autoria Própria, 2022.

Na figura acima, está sendo utilizado a *context* API para gerir as informações de login do usuário. Após o usuário efetuar o login na plataforma, é enviado o token gerado pelo *back-end* e é salvo dentro da classe acima. Podendo ser consultada a qualquer momento na aplicação para efetuar buscas e cadastros no sistema.

3.3.10.10 APIs

São utilizadas para realizar a integração entre o *front-end* com o *back-end*, seja por consulta, cadastro, edição ou deleção de informações ao banco de dados. Para isso, é feito uma requisição através de uma URL (*Uniform Resource Locator*), passando os parâmetros necessários para efetuar aquela determinada ação, conforme representada na Figura 37.

A Figura 37 demonstra uma consulta à API de *login*, onde o *e-mail* e senha fornecidos pelo usuário são enviados como parâmetros para a URL de requisição e caso as informações estejam corretas, um valor de *token* é retornado.

Figura 37 – Exemplo de requisição

```
const baseUrl = url + "login";

export async function Login(email, password) {
  const data = {
    email: email.trim(),
    password: password.trim()
  }
  const options = {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(data),
  };
  let response = await fetch(baseUrl, options);
  let resp = await response.json();
  let error = '';
  if (typeof resp["errors"] !== "undefined") {
    let errors = resp["errors"];
    error = errors[0]["msg"];
  } else if (typeof resp["message"] !== "undefined") {
    error = resp["message"];
  }
  if (error == '') {
    return resp;
  } else {
    return {message: error};
  }
}
```

Fonte: Autoria Própria, 2022.

3.4 Back-end

O back-end do projeto “Lista Fácil” foi desenvolvido utilizando Node.JS e tendo como principais *frameworks* e ferramentas auxiliares:

- Express: Micro *Framework* JavaScript responsável por gerenciar rotas, *middlewares* e exceções.
- Prisma: Esta ferramenta é um ORM (*Object-Relational Mapping*), feito para aplicações JavaScript que tem como responsabilidade facilitar e abstrair as operações com banco de dados.

A aplicação *back-end* se comunica com a aplicação *mobile*, através do protocolo HTTP (*HyperText Transfer Protocol*), este é um protocolo muito utilizado na Web, sendo o seu tipo cliente-servidor, com o auxílio deste protocolo foi construído uma API (*Application Programming Interface*) que transporta as informações entre as duas aplicações através de JSON (*Javascript Object Notation*), que é um formato baseado em texto para representar dados estruturas baseado na sintaxe do JavaScript.

3.4.1 Autenticação

A aplicação *back-end* conta com a autenticação utilizando o sistema JWT (*JSON Web Token*), este é um padrão utilizado na indústria de desenvolvimento de software definida pela RFC 7159, que tem como principal missão armazenar de forma segura e compacta objetos no formato JSON entre diferentes aplicações. Para a utilização do JWT no *back-end* foi utilizado uma biblioteca de código aberto e pública para a implementação na aplicação.

Um token JWT possui a seguinte estrutura:

- *Header*: O cabeçalho contém duas informações, sendo elas, o tipo do *token* e qual o algoritmo de assinatura que está sendo utilizado, nesta aplicação foi adotado o SHA256.
- *Payload*: Neste atributo fica todas as informações que são trafegadas entre a aplicação

- *Signature*: Esta é a assinatura do *Token*, este atributo serve para validar se o token é válido ou não.

3.4.2 Rotas sem autenticação

As seguintes rotas não precisam da autenticação JWT para serem requisitadas, as rotas são relacionadas ao fluxo de criação de conta na aplicação e login.

3.4.3 Rota para criação de Usuário

Para a criação do usuário a aplicação deve enviar uma requisição HTTP do tipo POST para o *endpoint/user* com o *payload* representado na Figura 38, abaixo.

Figura 38 – Requisição usuário

```
{
  "name": "usuário",
  "email": "email@gmail.com",
  "password": "senha123"
}
```

Fonte: Autoria Própria, 2022.

Após o envio dessa requisição, a aplicação retorna os seguintes dados, conforme Figura 39:

Figura 39 – Resposta usuário

```
{
  "id": "d84f28f0-a844-4762-a3f9-a1fe539d6510",
  "name": "usuário"
}
```

Fonte: Autoria Própria, 2022.

Os dados retornados são o ID do usuário no formato UUID (*Universally Unique Identifier*) e o nome do usuário através do atributo *name*.

3.4.4 Rota para login

Após ter criado sua conta na plataforma, o usuário deve *logar* no aplicativo. O aplicativo envia uma requisição POST para a rota “/login” enviando os seguintes dados, conforme apresentado na Figura 40.

Figura 40 – Requisição *login*

```
{  
  "email": "email@gmail.com",  
  "password": "senha123"  
}
```

Fonte: Autoria Própria, 2022.

Após validar as informações os dados retornados pelo *back-end* são os apresentados na Figura 41.

Figura 41 – Resposta login

```
{  
  "user": {  
    "id": "d84f28f0-a844-4762-a3f9-a1fe539d6510",  
    "name": "usuário",  
    "email": "email@gmail.com",  
    "active_account": true  
  },  
  "token":  
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2NjUwMzM5MjQsInVzZXIiOiJnsiaWQiOiJkODRmMjhmMC1hODQ0LTQ3NjItYTNmOS1hMWZlNTM5ZDY1MTAiLCJuYV1lIjoiaWZlbnR1bW50Ij09eyJpYXQiOiIyMDMwMzI0fQ.34Q7XwZ8w0XCWt85GAKHQHLLaygm-HQzQmE6q4Qvg-8"  
}
```

Fonte: Autoria Própria, 2022.

Os dados retornados são um objeto *user* que contém o ID, nome, *email* e se a conta do usuário está ativa e um outro objeto denominado *token* que possui o token JWT do usuário, este é o token que valida que o usuário está cadastrado na plataforma e que permite ele realizar operações na aplicação.

3.4.5 Rotas Autenticadas

Para acessar as seguintes rotas é necessário que o usuário esteja autenticado na plataforma, deste modo, o aplicativo deve enviar no cabeçalho da requisição HTTP o atributo *Authorization* contendo o *token* JWT retornado na requisição de login. Ao realizar uma requisição para as rotas autenticadas é acionado um *Middleware* para realizar a validação deste *token*, o código de implementação do *middleware* está representando na Figura 42 abaixo.

Figura 42 – Requisição autenticação

```
import { validateToken } from "../auth/validateToken.js";
import { StatusCodes } from "http-status-codes";

const authMiddleware = (req, res, next) => {
  const token = req.headers["authorization"];
  if (!token) {
    return res
      .status(StatusCodes.UNAUTHORIZED)
      .json({ error: true, message: "Token de autenticação não fornecido" });
  }

  const result = validateToken(token);

  if (result.messageError) {
    return res
      .status(StatusCodes.UNAUTHORIZED)
      .json({ error: true, message: result.messageError });
  }

  req.userId = result.user.id;
  req.userName = result.user.name;

  next();
};

export { authMiddleware };
```

Fonte: Autoria Própria, 2022.

A mensagem de erro fornecida para o aplicativo é representada na Figura 43.

Figura 43 – Resposta autenticação

```
{
  "error": true,
  "message": "Token de autenticação não é válido"
}
```

Fonte: Autoria Própria, 2022.

3.4.6 Rota de Criação de Lista de Compras

O primeiro passo do usuário na aplicação é realizar a criação da lista, o aplicativo deve enviar uma requisição POST para a rota “/shopping-lists” com o *payload* representado na Figura 44, ilustrada abaixo.

Figura 44 – Requisição lista de compras

```
{
  "title": "Nome da Lista"
}
```

Fonte: Autoria Própria, 2022.

Após o envio da requisição, o retorno será o *payload* abaixo conforme a Figura 45.

Figura 45 – Resposta lista de compras

```
{
  "id": 124,
  "title": "Nome da Lista"
}
```

Fonte: Autoria Própria, 2022.

O retorno apresenta o ID da lista, que é gerado através de autoincremento e o título da lista que o usuário informou, com essas informações o aplicativo consegue montar a listagem de listas de compras de forma ordenada.

3.4.7 Rota para inserção produtos na Lista de Compras

Após a criação da lista, o aplicativo deve informar os produtos que aquela lista contém, este processo acontece após o reconhecimento dos produtos que foram processados pela Inteligência Artificial. O aplicativo deve enviar uma requisição do tipo POST para a rota “shopping-lists/{ListaID}/products”, como parâmetro nesta rota o aplicativo deve enviar o ID da lista que ele deseja que os produtos sejam adicionados. Para realizar esta operação o aplicativo deve enviar um *payload* com o atributo *products* que é uma estrutura de dados do tipo *array* que contém o atributo *name* e o valor do produto, representado conforme a Figura 46:

Figura 46 – Requisição inserir produto

```
{
  "products": [
    {
      "name": "arroz"
    },
    {
      "name": "feijão"
    },
    {
      "name": "tapioca"
    }
  ]
}
```

Fonte: Autoria Própria, 2022.

Após o envio da requisição, o *back-end* retorna para o aplicativo o retorno representado na Figura 47.

Figura 47 – Resposta inserir produto

```
{
  "shoppingList": {
    "id": 124,
    "title": "Nome da Lista",
    "ShoppingListProducts": [
      {
        "id": 32,
        "name": "arroz"
      },
      {
        "id": 33,
        "name": "feijão"
      },
      {
        "id": 34,
        "name": "tapioca"
      }
    ]
  }
}
```

Fonte: Autoria Própria, 2022.

O *Payload* de retorno conta com um objeto denominado *shoppingList*, que é referente a lista de compras que os produtos foram vinculados e um outro objeto chamado *ShoppingListProducts*, que contém as informações referentes ao produto cadastrado naquela lista de compras, que possui ID e o nome do produto.

3.4.8 Rota para buscar todas as listas de compras do usuário

A aplicação fornece o recurso para listagem de todas as listas de compras de um usuário autenticado, para realizar esta operação o aplicativo deve enviar uma requisição do tipo GET para a rota `/users/me/shopping-lists`. O retorno desta requisição está representado na Figura 48.

Figura 48 – Requisição busca de listas

```
{
  "products": [
    {
      "name": "arroz"
    },
    {
      "name": "feijão"
    },
    {
      "name": "tapioca"
    }
  ]
}
```

Fonte: Autoria Própria, 2022.

Após o envio da requisição, o *back-end* retorna para o aplicativo o retorno representado na Figura 49.

Figura 49 – Resposta busca de listas

```
[
  {
    "id": 125,
    "title": "Nome da Lista 2",
    "createdAt": "2022-10-07T23:46:51.060Z"
  },
  {
    "id": 124,
    "title": "Nome da Lista",
    "createdAt": "2022-10-07T18:49:08.706Z"
  }
]
```

Fonte: Autoria Própria, 2022.

O *Payload* de retorno conta com um *array* de objetos que possui o ID, título e a data de criação da Lista de Compras no formato *timestamp*. O retorno dessa listagem é no formato decrescente, deste modo ordenando das listas de compras mais recentes para a menos recente.

3.4.9 Rota para buscar uma lista de compra por ID

Para acessar os detalhes de uma lista de compras, é possível que seja buscado através de seu identificador único, para a realização dessa operação o aplicativo deve enviar uma requisição GET para a rota `"/shopping-list/{ListaID}"`, contendo o ID da lista de compras na URL. O retorno dessa busca está representando na Figura 50.

Figura 50 – Requisição de busca de lista

```
{
  "shoppingList": {
    "id": 124,
    "title": "Nome da Lista",
    "ShoppingListProducts": [
      {
        "id": 32,
        "name": "arroz"
      },
      {
        "id": 33,
        "name": "feijão"
      },
      {
        "id": 34,
        "name": "tapioca"
      }
    ]
  }
}
```

Fonte: Autoria Própria, 2022.

As informações retornadas são o objeto *shoppingList* que contém as informações sobre a lista, como ID e título e também o objeto *ShoppingListProducts*, que é do tipo *array* e contém as informações dos produtos daquela lista, retornando o ID e nome do produto.

3.4.10 Rota para a criação de Compras

Após a realização de todo fluxo de cadastro de Lista de Compra e seus respectivos produtos, o usuário pode realizar o cadastro de suas compras, para que

no processo de criação da lista de compras seguinte, este processo seja totalmente automatizado. Para a realização deste processo, o aplicativo deve enviar uma requisição do tipo POST para a rota “/purchases” com o *payload* representado na Figura 51 a seguir.

Figura 51 – Resposta da busca lista

```
{  
  "title": "lista de compras"  
}
```

Fonte: Autoria Própria, 2022.

3.4.11 Rota para inserção de produtos na compra

No fluxo de compras, o usuário deve cadastrar os produtos vinculados da sua compra, com isso o aplicativo deve enviar uma requisição POST para a rota “/purchases/{{CompralD}}/products” com o conteúdo apresentado na Figura 52.

Figura 52 – Requisição inserção de produto

```
{  
  "products": [  
    {  
      "name": "arroz"  
    },  
    {  
      "name": "feijão"  
    }  
  ]  
}
```

Fonte: Autoria Própria, 2022.

O retorno para esta requisição é apresentado conforme a Figura 53.

Figura 53 – Resposta inserção de produto

```
{
  "purchase": {
    "id": 1,
    "title": "lista de compras",
    "createdAt": "2022-10-08T00:26:07.086Z",
    "PurchasedProducts": [
      {
        "id": 1,
        "name": "arroz"
      },
      {
        "id": 2,
        "name": "feijão"
      }
    ]
  }
}
```

Fonte: Autoria Própria, 2022.

Onde é retornado um objeto denominado *purchase*, que contém as informações da compra ID, título da compra e data de criação, também possui as informações dos produtos cadastrados no objeto *PurchasedProduct*, com o ID e nome cadastrados respectivamente.

3.4.12 Rota para buscar todas as compras do usuário

Para que o usuário tenha conhecimento de todas as suas compras, a aplicação fornece o método de buscar as compras realizadas por um usuário autenticado. Para realizar esta operação a aplicação deve enviar uma requisição do tipo GET para a rota `/users/me/purchases`, com o sucesso da requisição o aplicativo recebe o seguinte retorno de dados representado na Figura 54.

Figura 54 – Requisição busca de compras

```
[
  {
    "id": 2,
    "title": "lista de compras 2",
    "createdAt": "2022-10-08T01:46:35.228Z"
  },
  {
    "id": 1,
    "title": "lista de compras",
    "createdAt": "2022-10-08T00:26:07.086Z"
  }
]
```

Fonte: Autoria Própria, 2022.

O retorno contém um *array* populado com as informações das listas do usuário que contém o ID, título da lista e a data de criação no formato *timestamp*.

3.4.13 Rota para buscar uma compra específica por ID

Para que a aplicação possa encontrar os detalhes da lista, é necessária uma rota para buscar essas informações através do identificador único. Para a realização desta operação deve ser enviado uma requisição do tipo GET para a rota `"/purchases/{{CompralD}}"`, a resposta para essa requisição está representada na Figura 55.

Figura 55 – Requisição de busca de compras por ID

```
{
  "id": 1,
  "title": "lista de compras",
  "createdAt": "2022-10-08T00:26:07.086Z",
  "PurchasedProducts": [
    {
      "id": 1,
      "name": "arroz"
    },
    {
      "id": 2,
      "name": "feijão"
    }
  ]
}
```

Fonte: Autoria Própria, 2022.

As informações retornadas são o objeto *shoppingList* que contém as informações sobre a lista, como ID e título e também o objeto *ShoppingListProducts*, que é do tipo *array* e contém as informações dos produtos daquela lista, retornando o ID e nome do produto.

3.4.14 Rota para a criação da lista inteligente.

Para o fluxo completo da aplicação é necessário criar a Lista de compra inteligente, deve ser enviado o *payload* enviado, representado na Figura 56, pelo aplicativo em uma requisição POST para a rota “/smart-list”,

Figura 56 – Requisição para criação de lista inteligente

```
{
  "products": [
    {
      "name": "batata"
    },
    {
      "name": "maçã"
    }
  ]
}
```

Fonte: Autoria Própria, 2022.

No *payload* acima, deve ser enviado apenas os produtos que o usuário possui em sua casa, produtos estes que foram processados pela inteligência artificial na aplicação *front-end*, a aplicação *front-end* envia esses dados para o *back-end*, onde por sua vez busca a última compra do usuário e verifica se os itens enviados no *payload* constam na última compra, caso os itens constam eles são removidos da listagem e é criado uma nova lista de compras de forma automática e inteligente com todos os produtos da última compra cadastrada pelo usuário exceto pelos produtos que ele já possui em sua casa. O retorno dessa requisição está representado na Figura 57.

Figura 57 – Resposta para criação de lista inteligente

```
{
  "shoppingList": {
    "id": 125,
    "title": "Lista Inteligente - Gerada Automaticamente",
    "ShoppingListProducts": [
      {
        "id": 35,
        "name": "arroz"
      },
      {
        "id": 36,
        "name": "feijão"
      },
      {
        "id": 37,
        "name": "tapioca"
      }
    ]
  }
}
```

Fonte: Autoria Própria, 2022.

Nesta resposta a aplicação recebe todas as informações de uma Lista de compras, porém de gerado de automático, o conteúdo possui um objeto denominado *ShoppingList* que possui o ID e título da Lista de Compras geradas e um objeto *ShoppingListProducts* que possui todos os produtos gerados automaticamente para a sua lista de compra inteligente.

3.5 PROCESSO PARA O RECONHECIMENTO DE ITENS

Para realizar o reconhecimento de imagens enviadas pelo aplicativo foi desenvolvido um código em Python, utilizando tecnologias como YOLO e Darknet para a criação da base de dados das imagens que foram aprendidas através de *Deep Learning*.

As bibliotecas e pacotes externos utilizados neste código foram:

- Django-Rest-API: Um *framework Web* Python para realizar a API de processamento da imagem e retornar em formato JSON.
- OpenCV-Python: Biblioteca de Python para fazer processamento de imagens e interagir com o YOLO e Darknet.

3.5.1 Configurações necessárias

Para a execução do reconhecimento de imagens são necessários alguns arquivos como:

- Imagem: A imagem a qual está sendo processada para reconhecer os itens.
- *Config*: Arquivo de configuração do YOLO para conseguir realizar o vínculo da imagem e dos binários.
- *Weights*: Este arquivo possui todos os binários processados pelo YOLO e *Darknet* no processo de *Deep Learning*.
- Classes: Este arquivo contém os nomes dos itens que foram utilizados para o aprendizado.

3.5.2 Explicação do Script

Realizamos os testes primeiramente de forma local, com a criação de um *script* em Python.

Na Figura 58 é apresentado o início do *script* com as importações das bibliotecas e módulos padrões do Python necessários.

Figura 58 – Importação das bibliotecas

```
import cv2
import argparse
import numpy as np
import json
import array as arr
```

Fonte: Autoria Própria, 2022.

A próxima etapa do script, é utilizar a biblioteca Numpy, para conseguir que a aplicação se comunique com o terminal do computador que está executando o código, representado na Figura 59.

Figura 59 – Comunicação com o terminal

```
ap = argparse.ArgumentParser()  
ap.add_argument('-i', '--image', required=True,  
                help = 'path to input image')  
ap.add_argument('-c', '--config', required=True,  
                help = 'path to yolo config file')  
ap.add_argument('-w', '--weights', required=True,  
                help = 'path to yolo pre-trained weights')  
ap.add_argument('-cl', '--classes', required=True,  
                help = 'path to text file containing class names')  
args = ap.parse_args()  
  
produtos = []
```

Fonte: Autoria Própria, 2022.

No trecho acima, a aplicação recebe o parâmetro de imagem, que deve ser processada, o parâmetro de *config* referente ao arquivo do YOLO, em seguida o parâmetro *weights*, que são as informações binárias processadas pelo *Deep Learning* e por fim o arquivo classes que contém os nomes dos itens utilizados no aprendizado. A variável produto é iniciada de forma global na aplicação é do tipo *array* e é utilizada no método *create_json*.

Na figura 60 está representado o comando que deve ser enviado para a execução do código no terminal.

Figura 60 – Comando para o terminal

```
python3 yolo_opencv.py --image 1.jpg --config yolov3.cfg --weights yolov3.weights --classes yolov3.txt
```

Fonte: Autoria Própria, 2022.

A primeira etapa de execução do script é encontrar o arquivo de classes e percorrer todo o arquivo com as informações, conforme Figura 61.

Figura 61 – Comandos para percorrer todo o arquivo

```

with open(args.classes, 'r') as f:
    classes = [line.strip() for line in f.readlines()]

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

net.setInput(blob)

outs = net.forward(get_output_layers(net))

class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4

```

Fonte: Autoria Própria, 2022.

Nesta parte do código definimos também quais serão as cores de forma aleatória das caixas que vão identificar os objetos. Durante a leitura do arquivo realizamos a chamada do método *readnet* do *darknet* que recebe como parâmetro os arquivos *Weights* e de configuração. Este método é responsável por buscar as informações da imagem e criar o vínculo entre as informações que foram processadas pelo *Deep Learning* e pelo arquivo de classes, Em seguida os resultados são armazenados na variável *outs*. A Figura 62 representa o armazenamento dos resultados,

Figura 62 – Armazenamento dos resultados

```

net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

net.setInput(blob)

outs = net.forward(get_output_layers(net))

```

Fonte: Autoria Própria, 2022.

Com o processamento já feito pelo Darknet são declaradas as seguintes variáveis:

- *class_ids*: é do tipo *array* e é responsável por armazenar os IDs dos tipos de itens encontrados.
- *confidences*: é do tipo *array* e é responsável por armazenar a porcentagem de acerto dos itens encontrados.
- *boxes*: é do tipo *array* e tem a responsabilidade de armazenar os pontos na imagem onde serão desenhadas as caixas mostrando os itens reconhecidos.
- *conf_threshold*: Essa variável é referente ao limite mínimo de confiabilidade das imagens e tem como valor padrão 0.5.
- *nms_threshold*: Essa variável é referente ao limite máximo de supressão das boxes e tem como valor padrão 0.4.

Após essas configurações o script percorre as saídas processadas pelo Darknet, verificando se a confiabilidade do reconhecimento da imagem é maior que 50%, caso positivo ele começa a construção da imagem de respostas, definindo onde as boxes ficarão posicionadas, quais são os IDs dos itens encontrados e que foram armazenados na variável *class_id* retornando deste modo uma lista de índices, conforme a Figura 63.

Figura 63 – Retorno da lista de índices

```
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
```

Fonte: Autoria Própria, 2022.

Para desenhar a imagem e construir o resultado final com a imagem original o script percorre todos os índices, processa a informação de boxes e envia para o método *draw_prediction*, demonstrado na Figura 64.

Figura 64 – Construção da imagem final

```

for i in indices:
    box = boxes[i]
    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]
    draw_prediction(image, class_ids[i], confidences[i], round(x), round(y), round(x+w), round(y+h))

```

Fonte: Autoria Própria, 2022.

A função *draw_prediction* recebe como parâmetro a imagem enviada pelo usuário, o valor do índice do *class_id* do item, o índice confiabilidade do item armazenado na variável *confidences*, arredondamento do valor X, Y, arredondamento da soma das variáveis X e W e como último parâmetro a o arredondamento das somas das variáveis Y e H, segue o método *draw_prediction* na Figura 65.

Figura 65 – Função *draw_prediction*

```

def draw_prediction(img, class_id, confidence, x, y, x_plus_w, y_plus_h):

    confidencePorcent = confidence * 100
    confidenceFormatted = round(confidencePorcent, 2)
    label = str(classes[class_id]) + "=" + str(confidenceFormatted) + "%"
    color = COLORS[class_id]

    cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)

    cv2.putText(img, label, (x-10,y-50), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

```

Fonte: Autoria Própria, 2022.

Na execução deste método é calculado a confiabilidade da imagem em porcentagem e em seguida este valor é formatado para possuir duas casas decimais. Em seguida é definido a variável *label* que recebe o nome do item identificado e seu valor de reconhecimento, em seguida é criado o valor da cor da box que vai apresentar as informações do item. Por fim é criado um retângulo e inserido o texto no retângulo que são as informações do item.

Em nossos testes, para validar a eficiência do código utilizamos a interface visual para verificar o comportamento do código, para isso foi utilizado os métodos *imshow*, para criar uma janela que espera que o usuário pressione qualquer tecla para o encerramento da aplicação, segue trecho da implementação na Figura 66.

Figura 66 – Métodos de *imshow*

```
cv2.imshow("object detection", image)
cv2.waitKey()
cv2.imwrite("object-detection.jpg", image)
cv2.destroyAllWindows()
```

Fonte: Autoria Própria, 2022.

Como será utilizado o padrão web, na aplicação apresentada neste trabalho substituímos o método *draw_prediction* pelo método *create_json*, este método recebe como parâmetro apenas as variáveis *class_id* e o *confidence*, conforme ilustrado abaixo na Figura 67.

Figura 67 – Criação de JSON

```
def create_json(class_id, confidence):
    confidencePorcent = confidence * 100
    confidenceFormatted = round(confidencePorcent, 2)
    text = str(classes[class_id]) + "-" + str(confidenceFormatted) + "%"
    produtos.append(text)
```

Fonte: Autoria Própria, 2022.

A função *create_json* realiza as mesmas operações do método *draw_prediction*, porém ele adiciona o valor da variável *text* na variável global *produtos* que é retornado para o usuário no final do processo um JSON com todos os itens encontrados no reconhecimento.

3.5.3 Apresentação de resultados do Script

Neste tópico é apresentado o resultado do *Script*, foram utilizadas duas imagens para realizar o teste, a primeira imagem é a representada na Figura 68.

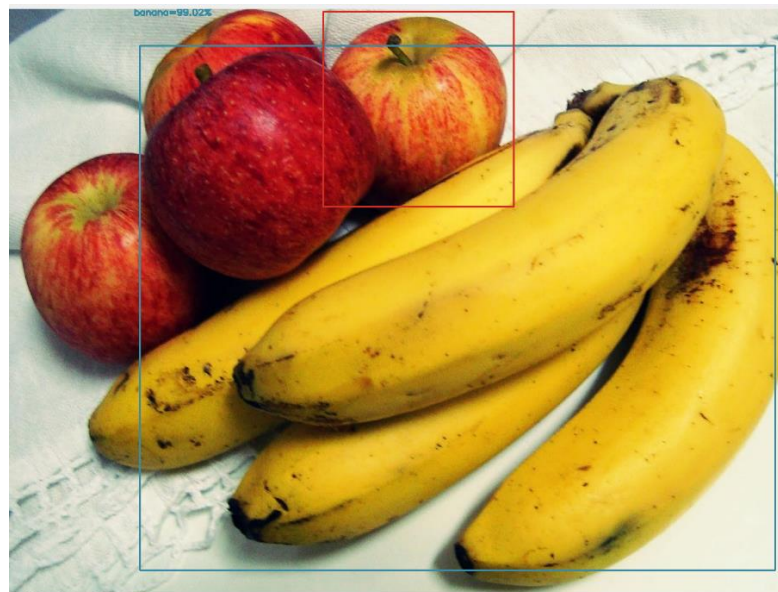
Figura 68 – Imagem *input*



Fonte: Associação Brasileira de Produtos de Maçã².

Após o script ter sido executado na linha de comando, é apresentado a seguinte imagem para o usuário, conforme Figura 69.

Figura 69 – Imagem com reconhecimento



Fonte: Autoria Própria, 2022.

O retorno do reconhecimento de imagem também é apresentado em formato JSON, conforme Figura 70.

² Disponível em <<https://www.abpm.org.br/maca-e-tudo-de-bom/maca-e-banana-estao-entre-as-frutas>>. Acesso em 25 de setembro de 2022.

Figura 70 – Retorno do JSON

```
{  
  [  
    "banana-99.02%",  
    "apple-95.96%"  
  ]  
}
```

Fonte: Autoria Própria, 2022.

O segundo teste foi realizado com a imagem de uma laranja, representada na Figura 71.

Figura 71 – Imagem *input*

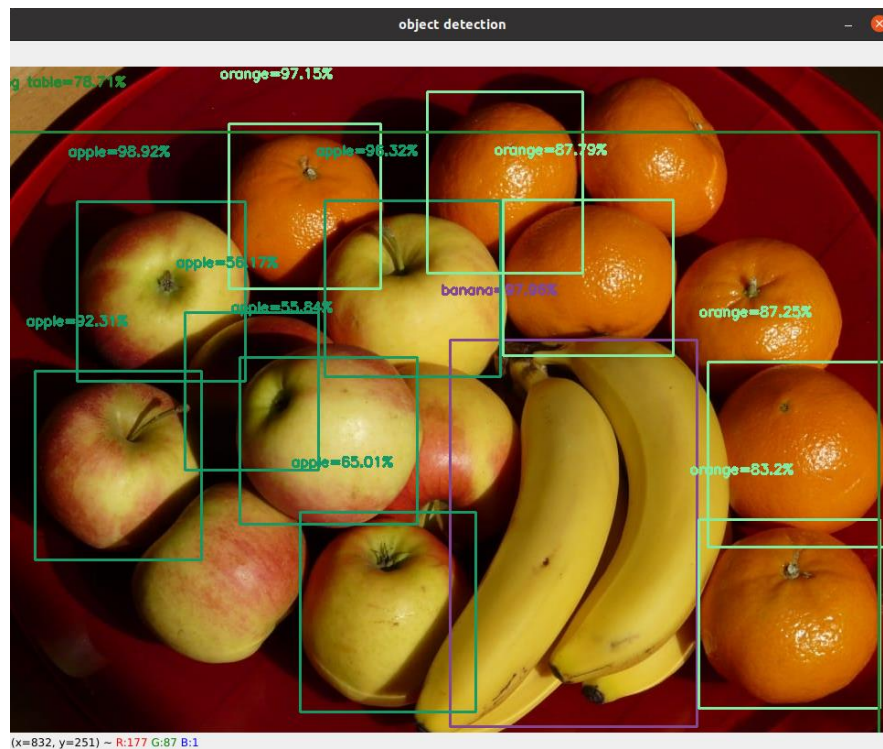


Fonte: PeakPX³.

O resultado do reconhecimento da imagem é apresentado na Figura 72.

³ Disponível em <<https://www.peakpx.com/513186/banana-oranges-and-apples-lot>>. Acesso em 25 de setembro de 2022.

Figura 72 – Retorno do JSON



Fonte: Autoria Própria, 2022.

E o resultado em formato JSON está representado na Figura 73.

Figura 73 – Retorno do JSON

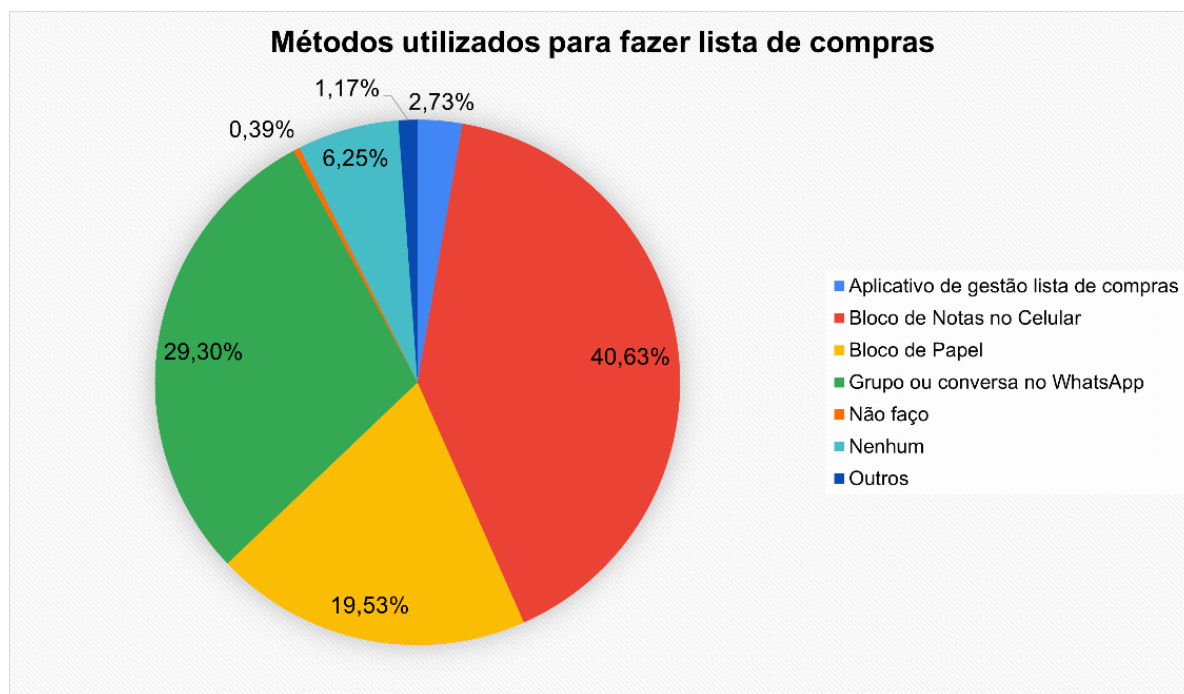
```
{
  [
    "apple-98.92%",
    "banana-97.96%",
    "orange-97.15%",
    "apple-96.32%",
    "apple-92.31%",
    "orange-87.79%",
    "orange-87.25%",
    "orange-83.2%",
    "orange-82.74%",
    "apple-65.01%",
    "apple-56.17%",
    "apple-55.84%"
  ]
}
```

Fonte: PeakPX, 2022.

4 RESULTADOS

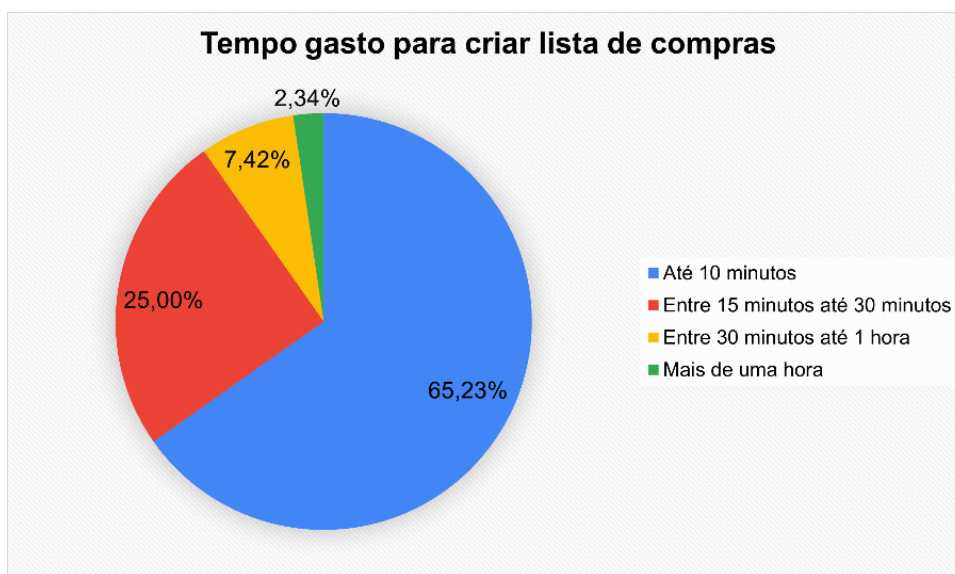
Diante dos resultados obtidos a partir da pesquisa quantitativa, foi possível observar que o desenvolvimento de um aplicativo de criação de lista de compras seria viável e útil. Relacionaram-se 4 parâmetros para avaliar a viabilidade de tal aplicativo, os métodos utilizados para a criação de listas, o tempo gasto, a frequência de compras e a aceitação de uso de um aplicativo capaz de automatizar este processo. Segundo a Figura 74 é possível notar que aproximadamente 60% dos entrevistados ainda realizam a listagem manual seja no papel ou no celular e 29,30% utilizam o aplicativo de mensagens WhatsApp como um bloco de anotações para realizar tal tarefa.

Figura 74 – Métodos utilizados para criar lista de compras



Fonte: Autoria Própria, 2022.

De acordo com a Figura 75, observa-se que 65,23% dos entrevistados gastam até 10 minutos para criar uma lista de compras, enquanto 34,77% passam mais de 15 minutos para fazer a mesma tarefa.

Figura 75 – Tempo gasto para criação de lista de compras

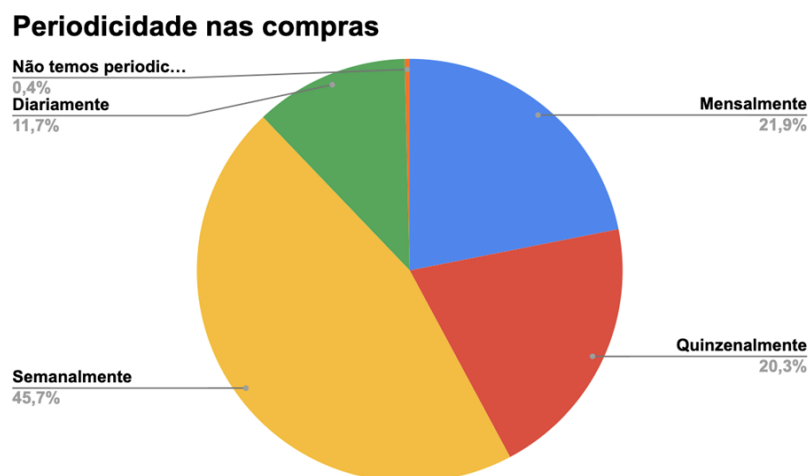
Fonte: Autoria Própria, 2022.

Em relação à criação de uma lista de compras, os resultados mostram que 48% das pessoas criam lista de compras antes de ir em algum mercado e 36% criam de vez em quando, como demonstrado na Figura 76.

Figura 76 – Porcentagem de pessoas que criam lista de compras

Fonte: Autoria Própria, 2022.

Quanto maior o intervalo entre as compras, a lista de compras tende a ficar mais extensa, virando um processo maçante. Segundo a Figura 77, 20,3% dos entrevistados fazem compras quinzenalmente, 21,9% fazem mensalmente e 45,7% fazem semanalmente.

Figura 77 – Periodicidade nas compras

Fonte: Autoria Própria, 2022.

A última pergunta realizada foi sobre a aceitação de uso, caso um aplicativo que auxiliasse os usuários na criação de lista de compras existisse. Conforme Figura 78, é possível observar que 91,4% fariam uso do Lista Fácil.

Figura 78 – Gráfico de aceitação de uso

Fonte: Autoria Própria, 2022.

A partir dos dados coletados nessa pesquisa, pode-se concluir que o desenvolvimento do aplicativo Lista Fácil seria de grande utilidade para grande parcela dos entrevistados, tendo em vista que o processo de criação da lista de compras ainda é realizado manualmente, o que acaba consumindo bastante tempo dos respondentes. Apesar de existir um intervalo pequeno entre as compras realizadas e o tempo gasto para listar os itens necessários para compra seja pequeno, através desse aplicativo será possível diminuir ainda mais o tempo necessário para realizar tal tarefa.

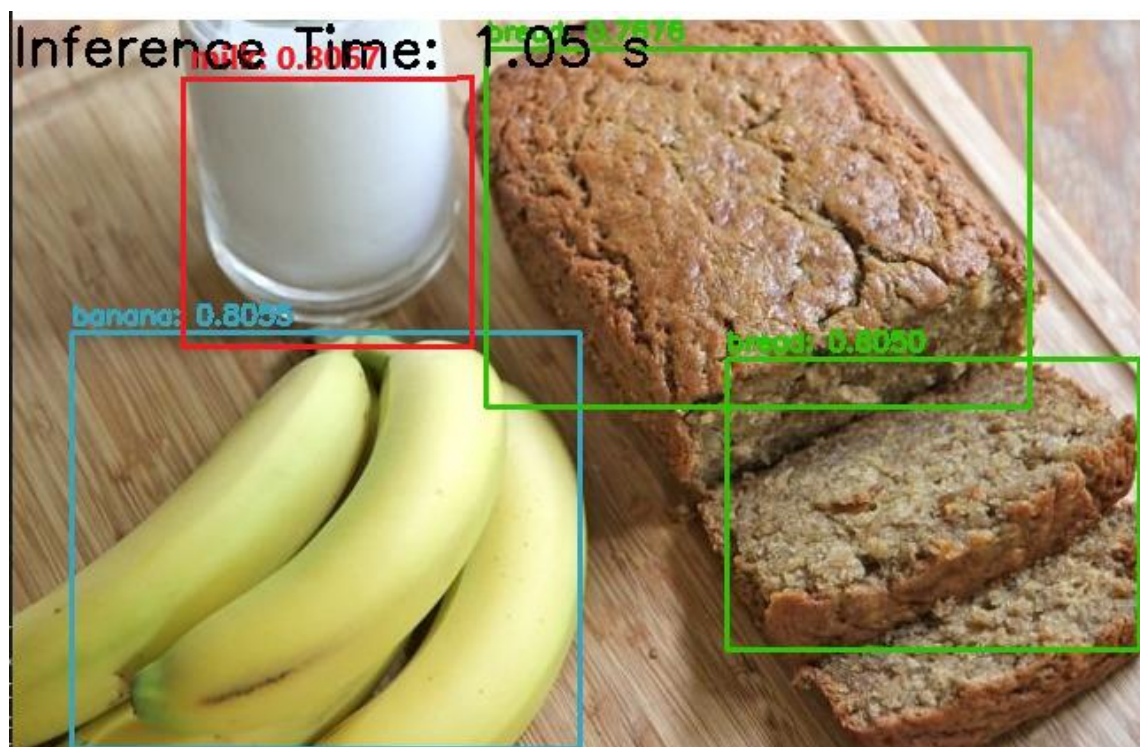
Utilizando o YOLO juntamente com o Darknet é possível realizar o treinamento de uma rede neural para o reconhecimento de um objeto em específico, porém é necessária uma grande amostra de dados para que esse treinamento seja feito com 100% de precisão em suas análises, sendo necessário mais de 500 imagens de treino juntamente com seus arquivos de *boundingbox*'s.

Além da necessidade de uma grande massa de dados para o treinamento, é necessária uma máquina capaz de suportar o treinamento, já que o treinamento consome muito de um computador, com cada objeto a ser aprendido pela rede neural tendo que ser treinado por pelo menos 2000 gerações da inteligência artificial para se obter uma precisão de quase 100%.

Como base nos testes realizados para a verificação e validação do método de visão computacional para reconhecimento dos objetos foi utilizado um *dataset* com 3 classes sendo elas: Banana, Garrafa e Pão. Cada classe contou com 500 imagens e *boundingbox*'s para o treinamento.

Com 500 imagens de cada classe, treinou-se a Inteligência Artificial por 2000 eras por um período de 6 horas, o que resultou em um treinamento de 80% a 90% de precisão nas identificações de Banana e Garrafa, mas com uma precisão de 70% a 80% na identificação da classe de Pão, como mostrado na Figura 79.

Figura 79 – Reconhecimento de produtos



Fonte: Autoria Própria, 2022.

Sendo assim, pode-se deduzir que para retornar uma precisão aceitável, acima de 70%, é necessário no mínimo o treinamento de 700 eras para cada classe. Caso desejado a precisão de aproximadamente 100% com poucas classes, é possível atingir esse resultado com cerca de 1000 eras para cada classe.

À medida que se aumenta o número de classes é necessário aumentar também o número de eras que a Inteligência Artificial irá treinar, já que quanto maior o número de classes necessárias para distinguir, maior é a dificuldade da IA para analisar a imagem e diferenciar um objeto de outro, pois suas escolhas de classes aumentaram.

Como apresentado no trabalho, é necessário realizar a criação e treinamento de uma grande massa de dados para reconhecimento de diversos itens, principalmente voltados para produtos de supermercado, visto que na maioria dos arquivos disponibilizados na Internet, todos eles são apenas de frutas, alimentos, móveis e objetos genéricos.

Em função dessa dificuldade encontrada, é recomendado a criação de novos estudos voltados para visão computacional baseada em *Deep Learning* para os mais diversos cenários de itens e objetos, expandindo, por exemplo, para a indústria automotiva como o reconhecimento de peças de uma frota específica de carros.

Outros trabalhos podem abordar a temática do uso do reconhecimento de imagens e *Deep Learning* para o uso na educação, como alunos presentes na sala de aula, processamento de imagens para correções de provas e gabaritos em larga escala como no caso do ENEM (Exame Nacional do Ensino Médio).

Com a finalização do desenvolvimento do protótipo do aplicativo *mobile* capaz de criar uma lista de compras, pode-se concluir que os objetivos levantados ao início do Trabalho de Conclusão de Curso foram todos atingidos.

5 CONSIDERAÇÕES FINAIS

Utilizando o poder do processamento de imagem do Python é possível reconhecer os produtos comprados pelo usuário, sendo assim possível criar uma lista comparando os itens que ele possui em estoque e os que ele comumente compra.

Com essa aplicação, o usuário não precisa realizar uma verificação e listagem manual dos produtos que estão faltantes no estoque e ele pode realizar de forma simples e rápida, apenas tirando uma foto dos itens que possui em estoque.

Com a lista já montada, ele pode ir ao mercado com um *checklist* de itens a serem comprados, assim facilitando o rastreio de produtos que já foram colocados no carrinho e itens que ainda faltam ser adicionados.

Este trabalho de conclusão de curso pretendeu facilitar o processo de criação de listas de compras, uma tarefa corriqueira que consome bastante tempo e, na maior parte dos casos, é maçante. Implementou-se a visão computacional com o intuito de desenvolver o aplicativo que seria capaz de gerar uma lista de compras de maneira inteligente.

A fim de compreender se o projeto proposto seria viável, realizou-se uma pesquisa de campo *online*, através da plataforma Google Forms. Foi possível observar que a maioria das respostas foi positiva e que os respondentes fariam uso do *software*, visto que esse poderia diminuir o tempo a ser gasto com a criação da lista caso ela fosse realizada manualmente, seja através do auxílio de um pedaço de papel ou através de um aplicativo de anotações no celular.

A média de criação de uma lista de compras é 10 minutos, tendo cerca de 65,2% dos entrevistados seguido de 25% que utiliza 15 a 30 minutos para montar uma lista. Já com a utilização da aplicação é possível montar uma lista em cerca de 5 minutos, diminuindo o tempo total em 50% ou mais. Além de evitar erros humanos com o esquecimento de algum item.

Como questão final para a pesquisa de campo, foi questionado se a pessoa gostaria e utilizaria a aplicação Lista Fácil para realizar a criação de suas listas de compras, tendo o resultado de 91,4% de aceitação dos entrevistados, gerando um total de 231 pessoas que utilizariam o aplicativo Lista Fácil.

REFERÊNCIAS

BECKER, Lauro. Orgânica Digital. **O que é React Native?**, 2021. Disponível em: <https://www.organicadigital.com/blog/o-que-e-react-native/>. Acesso em: 17 de abril de 2022.

CANGUÇU, Raphael. Codificar. **O que são Requisitos Funcionais e Requisitos Não Funcionais?**, 2021. Disponível em: <https://codificar.com.br/requisitosfuncionais-nao-funcionais/>. Acesso em: 24 de abril de 2022.

CAVALCANTE, Pablo H. A. GeekHunter. **Tutorial PostgreSQL**: introdução prática ao serviço, 2021. Disponível em: <https://blog.geekhunter.com.br/tutorial-postgresql-introducao-pratica-ao-servico/>. Acesso em: 10 de setembro de 2022.

CUNHA, André. Alura. **React Native**: o que é e tudo sobre o framework, 2022. Disponível em: <https://www.alura.com.br/artigos/react-native>. Acesso em: 18 de setembro de 2022.

DEV MEDIA. **Tecnologia Node.js**. Disponível em: <https://www.devmedia.com.br/guia/node-js/40312>. Acesso em: 16 de abril de 2022.

DEV MEDIA. **Tecnologia PostgreSQL**. Disponível em: <https://www.devmedia.com.br/guia/tecnologia-postgresql/34328>. Acesso em: 10 de setembro de 2022.

DEV MEDIA. **MER e DER**: Modelagem de Bancos de Dados, 2014. Disponível em: <https://www.devmedia.com.br/mer-e-der-modelagem-de-bancos-de-dados/14332>. Acesso em: 7 de maio de 2022.

HIGOR. DevMedia. **Introdução a Requisitos de Software**, 2013. Disponível em: <https://www.devmedia.com.br/introducao-a-requisitos-de-software/29580>. Acesso em: 24 de abril de 2022.

IBM. **Deep learning**. Disponível em: <https://www.ibm.com/br-pt/cloud/deep-learning>. Acesso em: 10 de setembro de 2022.

IBM. **What is computer vision?** Disponível em: <https://www.ibm.com/topics/computer-vision>. Acesso em: 10 de setembro de 2022.

IBM Cloud Education. **Convolutional Neural Networks**, 2020. Disponível em: <https://www.ibm.com/cloud/learn/convolutional-neural-networks>. Acesso em: 10 de Setembro de 2022.

MEIRELLES, Fernando S. FGV. **Panorama do Uso de TI no Brasil - 2022**, 2022. Disponível em: <https://portal.fgv.br/artigos/panorama-uso-ti-brasil-2022>. Acesso em: 05 de agosto de 2022.

MELO, Diego. Tecnoblog. **O que é Node.js? [Guia para iniciantes]**. Disponível em: <https://tecnoblog.net/responde/o-que-e-node-js-guia-para-iniciantes/>. Acesso em: 16 de abril de 2022.

NOLETO, Cairo. Trybe. **Requisitos não funcionais: o guia completo!**, 2020. Disponível em: <https://blog.betrybe.com/tecnologia/requisitos-nao-funcionais>. Acesso em: 24 de abril de 2022.

NOLETO, Cairo. Trybe. **Javascript: o que é, aplicação e como aprender a linguagem JS**, 2022. Disponível em: <https://blog.betrybe.com/javascript/>. Acesso em: 17 de abril de 2022.

PESSÔA, Camila. Alura. **Node.JS: definição, características, vantagens e usos possíveis**, 2022. Disponível em: <https://www.alura.com.br/artigos/node-js-definicao-caracteristicas-vantagens-usos>. Acesso em: 18 de setembro de 2022.

POSTGRESQL. **Supported Platforms**. Disponível em: <https://www.postgresql.org/docs/current/supported-platforms.html>. Acesso em: 10 de setembro de 2022.

REDMON, Joseph. **YOLO: Real-Time Object Detection**. Disponível em: <https://pjreddie.com/darknet/yolo/>. Acesso em: 18 de setembro de 2022.

ROMULO. DevMedia. **React.JS: Passando dados com Context API**, 2021. Disponível em: <https://www.devmedia.com.br/react-js-passando-dados-com-context-api/42904>. Acesso em: 18 de setembro de 2022.

ROSSETTI, Micaela L. SoftDesign. **REQUISITOS DE SOFTWARE: FUNCIONAIS E NÃO-FUNCIONAIS**, 2021. Disponível em: <https://softdesign.com.br/blog/requisitosde-software-funcionais-e-nao-funcionais>. Acesso em: 24 de abril de 2022.

VISURE Solutions. **O que são Requisitos Funcionais: Exemplos, Definição, Guia Completo**. Disponível em: <https://visuresolutions.com/pt/blog/functional-requirements/>. Acesso em: 17 de setembro de 2022.

W3SCHOOLS. **React useRef Hook**. Disponível em: https://www.w3schools.com/react/react_useeffect.asp. Acesso em: 24 de setembro de 2022.

W3SCHOOLS. **React** **useRef** **Hook.** Disponível em:
https://www.w3schools.com/react/react_useref.asp. Acesso em: 24 de setembro de 2022.

W3SCHOOLS. **React** **useContext** **Hook.** Disponível em:
https://www.w3schools.com/react/react_usecontext.asp. Acesso em: 24 de setembro de 2022.

W3SCHOOLS. **React** **useState** **Hook.** Disponível em:
https://www.w3schools.com/react/react_ustate.asp. Acesso em: 24 de setembro de 2022.