UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS
Faculdade de Engenharia da Computação
Bacharelado em Engenharia da Computação

Proposta de Projeto Final de Curso

APLICAÇÃO MOBILE PARA GESTÃO DE INVENTÁRIO COMERCIAL

Lucas Vinicius Silva Idelfonso

Marabá-PA
2023

Lucas Vinicius Silva Idelfonso

APLICAÇÃO MOBILE PARA GESTÃO DE INVENTÁRIO COMERCIAL

Projeto de Conclusão de Curso, apresentado à Universidade Federal do Sul e Sudeste do Pará, como parte dos requisitos necessários para obtenção do Título de Bacharel em Engenharia da Computação.

Orientador:
Prof. Dr. João Victor Costa Carmona

Marabá-PA
2023

Lucas Vinicius Silva Idelfonso

APLICAÇÃO MOBILE PARA GESTÃO DE INVENTÁRIO COMERCIAL

Projeto de Conclusão de Curso, apresentado à Universidade Federal do Sul e Sudeste do Pará, como parte dos requisitos necessários para obtenção do Título de Bacharel em Engenharia da Computação.

Marabá: 10 de Agosto de 2023

BANCA QUALIFICADORA:

Prof. Dr. João Victor Costa Carmona
(Orientador - UNIFESSPA)

Prof. Dr. Diego de Azevedo Gomes
(Membro da Banca - UNIFESSPA)

Prof. Me. Claudio Maciel de Souza Coutinho
(Membro da Banca - UNIFESSPA)

Marabá-PA
2023

AGRADECIMENTOS

Gostaria de agradecer e dedicar este trabalho as seguintes pessoas:

Primeiramente a minha família que me incentivaram nos momentos difíceis e compreenderam a minha ausência enquanto eu me dedicava à realização deste trabalho, um muito obrigado especial a minha mãe Rayldhalia Martins da Silva que proporcionou toda a minha educação e ao meu irmão Gabryel Vitor da Silva Correia.

Aos professores, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional ao longo do curso. Principalmente ao meu orientador João Vitor Costa Carmona e a minha banca qualificadora formada pelos professores Claudio de Castro Coutinho Filho e Diego de Azevedo Gomes.

Aos amigos, que sempre estiveram ao meu lado, pela amizade incondicional e pelo apoio demonstrado ao longo de todo o período de tempo em que me dediquei a este trabalho. Em especial a Amanda Fiel Savino, Beatriz de Jesus Silva Cavalcante, Bruno Borges Guerra, Bryan Franklin Sena de Sousa, Jefferson Yure Silva Pereira e Lucas Keley Sousa de Jesus

RESUMO

O presente trabalho tem como objetivo desenvolver uma aplicação mobile para gestão de inventário comercial que atenda às necessidades dos microempresários e autônomos que gerenciam seus empreendimentos sem muitos recursos de infraestrutura tecnológica. Com o desenvolvimento de uma aplicação mobile com funcionamento totalmente online, a proposta visa trazer dinamismo e praticidade para ser utilizado a qualquer momento e em qualquer lugar, sem a necessidade de um servidor local. Além disso, o trabalho apresenta a metodologia utilizada para o desenvolvimento da aplicação, envolvendo o uso do framework Django para a criação de uma API RESTful para comunicação com o aplicativo mobile. A interface do usuário foi desenvolvida com o framework Flutter, proporcionando uma experiência de usuário agradável e intuitiva. No que se refere à segurança da aplicação, a autenticação do usuário é realizada por meio de tokens JWT, garantindo a privacidade dos usuários e a segurança dos dados armazenados. O banco de dados utilizado é o PostgreSQL, que é um sistema gerenciador de banco de dados relacional de código aberto. O trabalho é fundamentado em conceitos importantes como Clean Code, SOLID e Clean Architecture, que buscam apresentar uma solução eficiente e segura para a gestão de inventário comercial. A aplicação desenvolvida é uma excelente opção para os microempresários e autônomos que buscam uma solução tecnológica que seja fácil de usar e segura para a gestão de seus negócios.

Palavras-chave: Django, Flutter, Python, Dart, API Rest, microempresários, autônomos.

ABSTRACT

The present work aims to develop a mobile application for commercial inventory management that meets the needs of microentrepreneurs and self-employed people who manage their enterprises without many technological infrastructure resources. With the development of a mobile application with fully online operation, the proposal aims to bring dynamism and practicality to be used anytime and anywhere, without the need for a local server. In addition, the work presents the methodology used for the development of the application, involving the use of the Django framework to create a RESTful API for communication with the mobile application. The user interface was developed with the Flutter framework, providing a pleasant and intuitive user experience. Regarding the security of the application, user authentication is performed through JWT tokens, ensuring the privacy of users and the security of stored data. The database used is PostgreSQL, which is an open source relational database management system. The work is based on important concepts such as Clean Code, SOLID and Clean Architecture, which seek to present an efficient and secure solution for commercial inventory management. The application developed is an excellent option for microentrepreneurs and self-employed people who are looking for a technological solution that is easy to use and secure for the management of their business.

Keywords: Django, Flutter, Python, Dart, API Rest, micro-entrepreneurs, self-employed..

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
DRF	*Django Rest Framework*
EPP	*Empresa de Pequeno Porte*
ERP	*Enterprise Resource Plannin*
HTTP	*Hypertext Transfer Protoco*
JSON	*JavaScript Object Notation*
ME	*Micro Empreendedor*
MEI	*Micro Empreendedor Individual*
MVP	*Minimum Viable Product*
MVC	*Model View Controller*
ORM	*Object-relational mapping*
PIB	*Produto Interno Bruto*
REST	*Representational State Transfer*
SEBRAE	*Serviço Brasileiro de Apoio às Micro e Pequenas Empresas*
SOAP	*Simple Object Access Protocol*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifiere*
VM	*Virtual Machine*
WWW	*World Wide Web*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Em 2006, entrou em vigor a Lei nº. 123/2006, conhecida como a Lei Geral da Micro e Pequena Empresa. Essa lei estabelece um regime tributário específico para pequenos negócios, permitindo a segmentação de tipos de pequenas empresas no Brasil com base em seu faturamento anual e número máximo de funcionários contratados. As empresas com receita anual de até 360 mil são classificadas como microempresas (ME), enquanto as empresas com até 4,8 milhões em receita anual são classificadas como empresas de pequeno porte (EPP). Além disso, a partir de 2008 ao ser criada a Lei complementar n° 128/2008, foi posta em vigor a modalidade de MEI (Microempreendedor individual), que visa estabelecer um regime jurídico específico para aqueles profissionais autônomos que desejam ser reconhecidos como pequenos empresários. Os MEIs estão limitados a um faturamento anual de até 81 mil reais e podem ter até um funcionário contratado em seu nome. Como empresas, esses pequenos empreendedores podem usufruir dos benefícios criados para incentivar a abertura de negócios e o aumento de empregos no Brasil.

Atualmente, em 2022, possuímos 18,5 milhões de microempreendedores atuando em nosso mercado nacional, o que representa 99% do total de empresas, 72% dos empregos e 30% do PIB do país (SEBRAE, 2022). Considerando esses números, podemos analisar que os pequenos empreendedores alavancaram nosso país nos últimos anos, especialmente nos últimos dois anos, após a pandemia, registrando um crescimento de 7% desde 2020. Logo, podemos observar que o instinto empreendedor está cada vez mais presente na economia brasileira, ampliando cada vez mais os segmentos de mercados em nosso comércio, onde a cada dia ideias e serviços são criados em cima das crescentes necessidades.

Analisando essa conjuntura, e observando o crescente avanço de inovações tecnológicas em nosso mercado, inúmeros serviços e produtos estão cada vez mais presentes no segmento de mercado via internet. Desse modo, tornando-se uma tendência quase que obrigatória a se seguir, onde se faz cada vez mais presente o uso de ferramentas tecnológicas para atender essa crescente demanda, como os sistemas de Gestão Empresarial, os famosos ERP(Enterprise Resource Planning), que além de atenderem essa demanda de gerir os recursos das empresas, possibilitam a centralização das informações, dessa forma, facilitando a gestão para os empreendedores.

O objetivo deste trabalho é a construção de um software de Gestão de Inventario Comercial, que irá consistir na criação de um software que irá auxiliar na gestão de pequenos negócios, focando nos varejistas que trabalham com comércio online. Dessa forma, o uso do projeto será centralizado em ajudar no planejamento financeiro dos empreendimentos de seus usuários, desse modo, contará com um módulo focado em estoque dos produtos junto com o cadastros das remessas dos mesmos, logo facilitando a leitura dos gastos.

O software desenvolvido foi estruturado em duas partes fundamentais: uma API

(Interface de Programação de Aplicativos) em Python, responsável pela lógica da solução. Essa API possibilita o acesso seguro e online ao banco de dados e ao sistema de autenticação. Na segunda parte, temos uma aplicação móvel que disponibiliza todas as funcionalidades do sistema por meio de uma interface de usuário intuitiva. Para criar a aplicação móvel, optamos por utilizar o Flutter, uma plataforma de desenvolvimento móvel desenvolvida pelo Google. Essa escolha se dá devido à sua capacidade de compilação nativa, o que resulta em um desempenho otimizado e uma experiência do usuário mais fluida.

1.1 Justificativa

Hoje segundo a pesquisa realizada pelo SEBRAE, disponível no material "O Impacto da pandemia de coronavírus nos Pequenos Negócios", 84% das microempresas entrevistadas utilizam o Whatsapp como ferramenta para suas vendas, e dentre o total analisado, em torno de 50% também utilizam do Facebook e Instagram como seu principal canal de vendas, ou seja, a maioria das empresas hoje necessitam desses serviços para manter as divulgações de seus produtos e a comunicação com seus clientes. Nesse contexto, à medida que novos canais e meios de comunicação vão se destacando no mercado, o empreendedor necessita estar ciente e operando no mesmo para se manter relevante no cenário.

Nesse sentido, é notório o uso e aumento do número de ferramentas que são necessárias para se poder gerir um empreendimento, e no cenário dos microempreendedores isso acaba se tornando um empecilho, pois segundo os critérios para o cadastro do MEI, o número de contratos de funcionários é limitado somente a uma pessoa, assim, deixando exposto que a maioria das tarefas de gestão estão centralizadas em um número pequeno de pessoas, o que nos direciona para a problemática da necessidade de um software mais versátil que possa ser acessado em qualquer localidade, facilitando para esse microempreendedor conseguir administrar seu empreendimento até mesmo sozinho e com mais eficiência.

1.2 Objetivo Geral

Desenvolver um de software para a gestão de inventario comercial de micro-empreendedores, através da construção de um aplicativo Mobile junto com uma API Rest.

1.3 Objetivos Específicos

Nos objetivos específicos serão retratados os seguintes pontos com o objetivo de agregar e alcançar o objetivo geral:

- Desenvolver os softwares utilizando as stacks em tendência no mundo da programação;

- Desenvolver o projeto utilizando as boas práticas do Clean Code, incorporando as metodologias do SOLID e implementando os padrões de desenvolvimento mais comumente utilizados;
- Criação da API Rest;
- Criação do Aplicativo Mobile;

2 TRABALHOS CORRELATOS

2.1 Controle Universitário: Um Aplicativo de controle dos estudos e do andamento do graduando ao longo do curso superior

O trabalho escrito por Anna (2021) é apresentado o desenvolvimento de um aplicativo móvel com a intenção de ajudar os alunos no gerenciamento das suas disciplinas e tarefas durante o ensino superior. Dessa forma, permitindo aos alunos a cada semestre verificar suas notas, faltas e atividades a serem entregues. O processo de desenvolvimento realizado pelo autor, apresenta o uso da linguagem Java que é caracterizada por ser uma das linguagens nativas para aplicativos Android. Além disso, ele utiliza o Android Studio, como ambiente de desenvolvimento e o firebase para hospedagem e armazenamento dos dados, ambas ferramentas do Google.

Para a criação da interface gráfica da aplicação, foi utilizado o Material Design, que se caracteriza como um conjunto de diretrizes criadas pela Google para o design de telas de aplicações móveis e sistemas web. Esta biblioteca disponibiliza um conjunto de componentes que auxiliam na padronização das telas e no desenvolvimento mais rápido e pratico das interfaces.

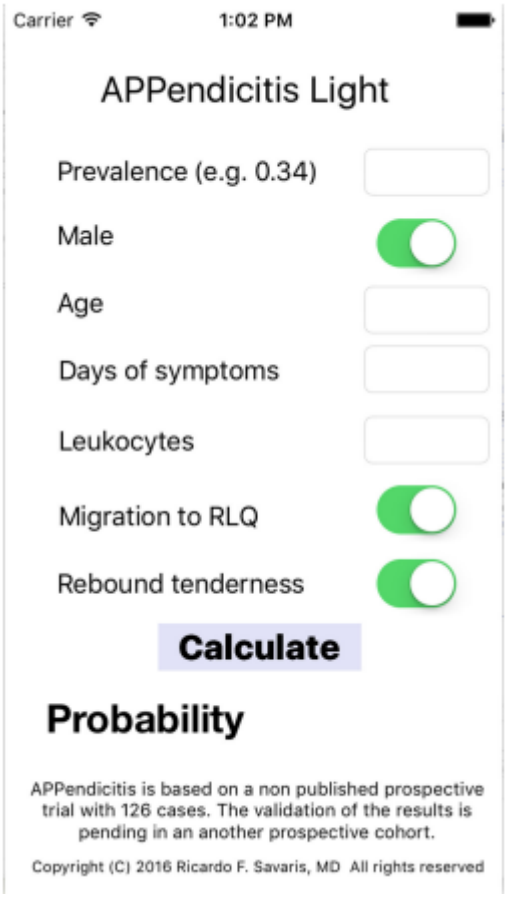
O projeto foi escolhido por propor o desenvolvimento de uma aplicação mobile como forma de resolução do problema, utilizando o desenvolvimento mobile nativo do Android, assim podendo servir como analise e comparação. Além disso, no trabalho da Anna (2021) é utilizado o Material Design que também é utilizado pelo Flutter.

2.2 Um aplicativo multiplataforma desenvolvido com Flutter e NoSQL para o cálculo da probabilidade de apendicite

”O presente trabalho consiste na elaboração de um aplicativo móvel para Android e iOS para calcular a probabilidade de apendicite, com o objetivo de auxiliar médicos no diagnóstico da doença. Ao longo do trabalho são exploradas as ferramentas utilizadas no desenvolvimento, como o framework Flutter para o desenvolvimento multiplataforma e o Cloud Firestore, um banco de dados NoSQL na nuvem para o armazenamento de dados.”(CORAZZA, 2018)

Este trabalho foi escolhido por trazer uma proposta de desenvolvimento de um aplicativo multiplataforma utilizando o framework de desenvolvimento Flutter. A proposta do trabalho envolvia criar uma solução que trouxesse melhorias ao trabalho já existente, visto na figura 1 . Dessa forma, além de trazer melhorias visando superar as limitações como a simplicidade da interface, o aplicativo trouxe novos recursos como um sistema de login e armazenamento de dados na nuvem. O desenvolvimento do projeto se iniciou com a capacitação na tecnologia escolhida para o desenvolvimento, além do estudo nas

Figura 1 – Aplicativo Desenvolvido pelo Coorientador



Fonte: (CORAZZA, 2018)

documentações do banco de dados escolhido, visando facilitações no momento da implementação. Em seguida, o desenvolvimento pratico do aplicativo foi destrinchado utilizando a metodologia Scrum para dividir o trabalho e os estágios do mesmo em ”Sprints”, trazendo organização para o processo.

2.3 Aplicativo para dispositivo móvel android para suporte ao processo de inventariamento de estoque

O projeto do Costa (2015) teve o objetivo de desenvolver um aplicativo móvel que irá automatizar o processo de organização de estoque para os clientes da empresa de desenvolvimento de software Dream Sistemas. Dessa forma, o aplicativo permite que funcionários utilizem seus smartphones para identificar os produtos nas prateleiras (usando a câmera destes dispositivos para scanear a imagem do código de barras), enviando estes dados para um servidor de aplicação, automatizando o processo de organização do inventario. Dentre os objetivos específicos, o projeto propõe os seguintes:

- Definir layout do arquivo a ser criado pelo software da Pharma Manager para

transferência dos itens para o sistema proposto;

- Utilizar biblioteca Zxing para capturar código de barras;
- Criar layout para o arquivo gerado pelo sistema proposto para
- importação no sistema da farmácia;
- Desenvolver aplicativo móvel;
- Integrar aplicativo móvel ao software Pharma Manager.

No projeto ele apresenta os resultados do desenvolvimento do sistema proposto, incluindo a aplicação dos métodos informados no capítulo de materiais e métodos para o desenvolvimento do sistema. A sessão de desenvolvimento também apresenta uma breve análise das principais funcionalidades do aplicativo, dividida em requisitos funcionais, não funcionais e o diagrama UML de implementação. Dentre os resultados obtidos, temos a construção do aplicativo móvel visto na figura 2 No momento que o aplicativo é executado, ele carrega a tela de apresentação e em seguida são demonstrados os itens do inventario adicionados pelo usuário do software. Como visto na figura 2, a tela de inventario mostra o código de barras de cada item, a descrição e a quantidade de itens em estoque.

Figura 2 – Tela Itens do Inventario

Fonte: (COSTA, 2015)

O trabalho foi escolhido para ilustrar um exemplo de projeto de desenvolvimento de software que utiliza a tecnologia de aplicativos móveis para automatizar processos e melhorar a eficiência empresarial. O projeto de Costa (2015) mostra como o uso de smartphones pode ser uma ferramenta valiosa para organizar o inventário de uma empresa, simplificando o processo de identificação e registro de produtos. Além disso, o trabalho apresenta uma análise completa das funcionalidades do aplicativo, bem como dos resultados obtidos. É um exemplo interessante de como a tecnologia pode ser aplicada para otimizar processos de negócios e melhorar a eficiência operacional.

2.4 Desenvolvimento de um aplicativo para adoção de pets utilizando Flutter e Firebase

O objetivo do trabalho é desenvolver um aplicativo multi-plataforma para ajudar pessoas a adotarem e doarem animais de estimação. O aplicativo foi criado utilizando duas tecnologias disponibilizadas pelo Google, o framework Flutter para construção do aplicativo móvel e o Firebase como back-end. (PORDEUS, 2021).

Figura 3 – Protótipo Tela Inicial

Fonte: (PORDEUS, 2021)

A etapa de desenvolvimento foi dividida em três etapas: instalação do ambiente de desenvolvimento, coleta de requisitos junto da prototipação do aplicativo e apresentação do aplicativo desenvolvido. Ele detalha o processo de desenvolvimento do aplicativo AdotaPet, desde a configuração do ambiente de desenvolvimento até a criação do protótipo e a apresentação do aplicativo final. Nesse sentido, foi utilizado o método das historias de usuário(User Storys) para obtenção dos requisitos e como visto na figura 3 foi utilizada

técnica de prototipação de baixa fidelidade para demonstrar visualmente as telas que seriam construídas na aplicação. Já na figura 4 podemos visualizar a mesma tela da figura 3, mas já finalizada e construída com o framework Flutter.

Figura 4 – Tela Inicial

Fonte: (PORDEUS, 2021)

O trabalho de Pordeus (2021) foi escolhido como um dos correlatos por apresenta um exemplo prático de como é possível utilizar tecnologias modernas para criar um aplicativo móvel multi-plataforma. O estudo detalha o processo de desenvolvimento do aplicativo AdotaPet, desde a coleta de requisitos até a apresentação do aplicativo final. O uso do framework Flutter e do Firebase como back-end permitiu que o aplicativo fosse criado de forma rápida e eficiente, sem a necessidade de desenvolver diferentes versões para diferentes plataformas. Além disso, a técnica de prototipação de baixa fidelidade demonstrada no estudo é uma abordagem útil para garantir que os requisitos do usuário sejam atendidos antes de começar a construir o aplicativo. Em resumo, o trabalho de Pordeus (2021) fornece um bom exemplo de como tecnologias modernas podem ser utilizadas para criar aplicativos móveis eficientes e multi-plataforma.

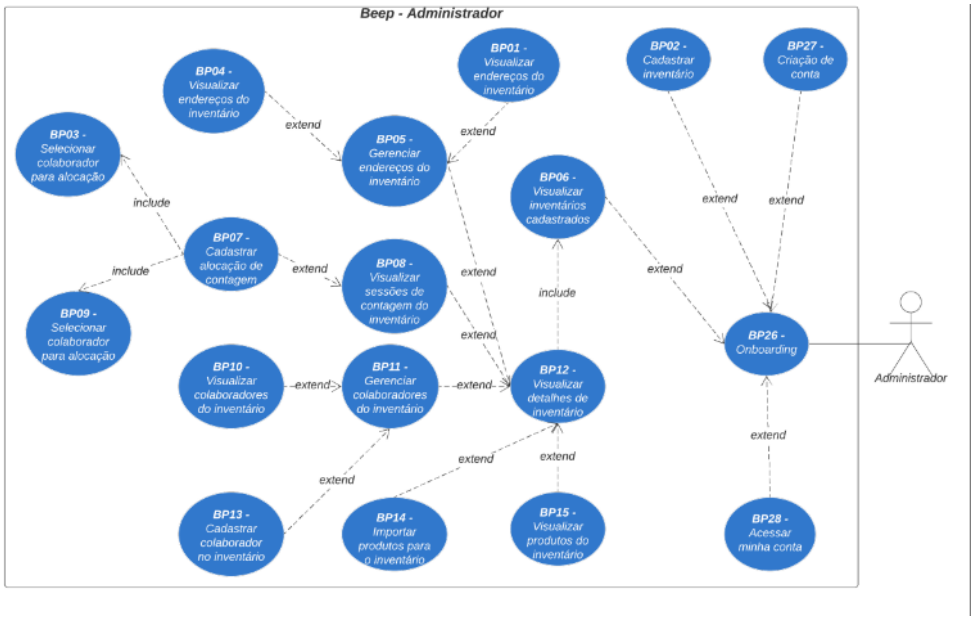
2.5 Desenvolvimento de uma aplicação para realização de inventários utilizando dispositivos móveis

O objetivo deste trabalho consiste em desenvolver um aplicativo móvel MVP, com o intuito de automatizar e aprimorar a eficiência na execução de inventários, oferecendo uma alternativa aos coletores de dados tradicionais. Além disso, busca-se apoiar o processo

de realização de inventários por meio dessa solução proposta. Dessa forma, o projeto do Gomes (2021), tem como escopo desenvolver, implantar e testar um aplicativo para a realização de inventários através de celulares e tablets em um supermercado, utilizando-se da metodologia de desenvolvimento de software Kanban.

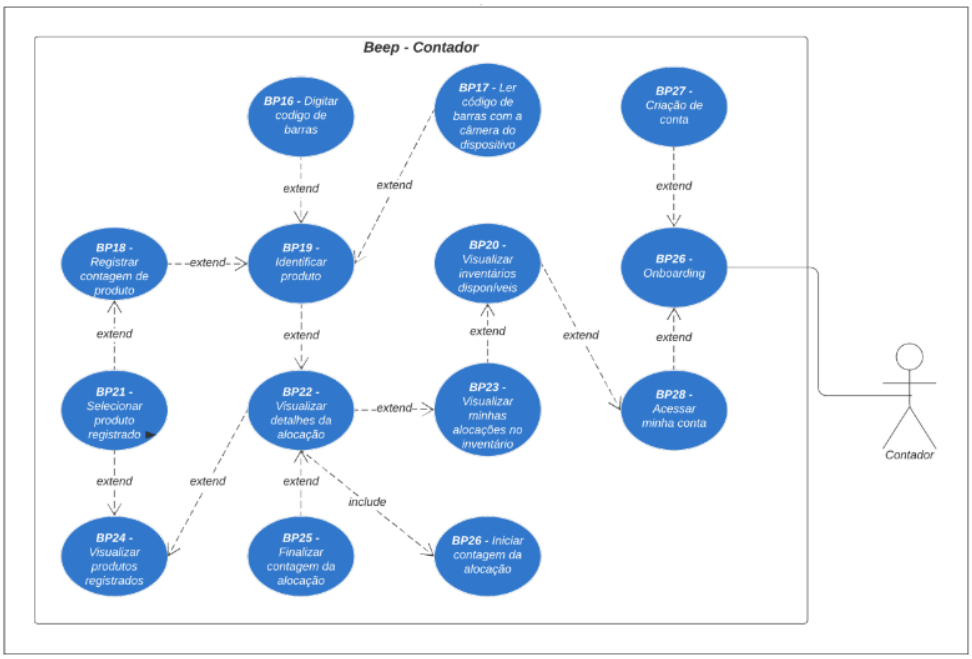
A solução proposta tem como foco o gerenciamento, realização e análise de inventários, e será desenvolvida como um Minimum Viable Product (MVP) que contém as regras de negócio mais impactantes para o funcionamento da solução. Nesse sentido, o apresenta o uso da metodologia Kanban no desenvolvimento de software, que busca organizar o trabalho em estágios sequenciais para facilitar a visualização do progresso das tarefas. Além disso, para obtenção dos requisitos ele utiliza dois diagramas de casos de uso visto na figura 5 e 6, onde cada um fica responsavel por apresentar um dos atores do sistema - Administrador e Contador. O caso de uso do Administrador permite a criação e configuração de inventários, enquanto o caso de uso do Contador permite a contagem de produtos nos inventários em que participa.

Figura 5 – Casos de Uso Administrador



Fonte: (GOMES, 2021)

Figura 6 – Casos de Uso Contador



Fonte: (GOMES, 2021)

O projeto de Gomes (2021) apresenta uma análise detalhada do processo de desenvolvimento do aplicativo, desde a definição dos requisitos até a implantação e testes da solução. Para obtenção dos requisitos, foi utilizado diagramas de casos de uso para apresentar visualmente as funcionalidades do aplicativo e as interações dos usuários com a solução.

Em resumo, o trabalho de Gomes (2021) oferece um exemplo prático de como a tecnologia móvel pode ser aplicada para melhorar a eficiência e a produtividade em operações empresariais. A utilização da metodologia Kanban no processo de desenvolvimento do software permitiu uma abordagem mais ágil e eficiente, enquanto a tecnologia de aplicativos móveis ofereceu uma solução prática e acessível para a realização de inventários em tempo real. Por esses motivos, o projeto de Gomes (2021) é uma escolha relevante como correlato em uma monografia sobre desenvolvimento mobile.

2.6 Correlação entre os trabalhos

O primeiro critério para escolha dos trabalhos foi a separação pela categoria, onde foi buscado somente aplicações que atuavam em dispositivos mobile. Ademais, como segundo critério foi escolhido trabalhos que atuaram com a tecnologia Flutter que seria utilizada neste trabalho, ou com a linguagem Java por ser considerada a linguagem de desenvolvimento nativo do Android. Dessa forma, durante a etapa de estudo dos trabalhos, foi avaliado os critérios de planejamento de desenvolvimento de cada projeto, listados logo abaixo:

- Todos os trabalhos apresentam o uso de tecnologias modernas, com destaque para o Flutter e o Firebase, para o desenvolvimento de aplicativos móveis eficientes e multi-plataforma;
- Cada trabalho apresenta uma metodologia de desenvolvimento de software específica, como User Storys e Kanban, para garantir a eficiência e a agilidade no desenvolvimento do aplicativo;
- Todos os trabalhos têm como objetivo melhorar a eficiência e a produtividade em operações empresariais, seja no gerenciamento de inventários, na adoção de animais de estimação ou na organização de estoques de produtos.

Tabela 1 – Tabela Trabalhos Correlatos

Trabalhos	**Ano**	**Tecnologia**	**Categoria**	**Objetivo**	**Backend**
Trabalho 2.1	2021	Java	Aplicativo Móvel	Gestão Acadêmica	Firebase
Trabalho 2.2	2018	Flutter	Aplicativo Móvel	Saúde	Firebase
Trabalho 2.3	2015	Java	Aplicativo Móvel	Inventario Comercial	Servidor Próprio
Trabalho 2.4	2021	Flutter	Aplicativo Móvel	Adoção de Animais	Firebase
Trabalho 2.5	2021	Flutter	Aplicativo Móvel	Inventario Comercial	Firebase
Trabalho Autor	2023	Flutter	Aplicativo Móvel	Inventario Comercial	API Rest

Fonte: Autor

Durante a análise minuciosa de cada trabalho, foi verificado especificamente qual tecnologia foi utilizada no desenvolvimento do backend de cada aplicação. A tabela 1 mostra que a maioria dos trabalhos utilizou o Flutter como tecnologia de desenvolvimento e o Firebase como solução para a construção do backend. No entanto, mesmo com a prevalência dessas tecnologias, existem outras opções disponíveis que podem ser mais adequadas para certos projetos.

Com base nessas descobertas, este trabalho tem como objetivo trazer uma forma inovadora de implementação, criando uma API personalizada para o backend. Isso permitirá uma maior flexibilidade e controle sobre o funcionamento do sistema, além de possibilitar o uso de outras tecnologias que possam ser mais adequadas para a solução de problemas específicos. Portanto, o desenvolvimento de uma API própria é uma etapa crucial para o sucesso deste trabalho, uma vez que permitirá a criação de um sistema mais personalizado e adaptado às necessidades do projeto em questão

3 FUNDAMENTAÇÃO TEÓRICA

Neste capitulo será abordado conceitos de fundamentações necessárias para compreensão das tecnologias e praticas utilizadas para a construção deste projeto. Nesse sentido, serão apresentados os protocolos HTTP e JSON como forma de transferência de dados, a utilização de bancos de dados para armazenamento, a adoção do padrão de arquitetura MVC, além das boas práticas de programação, como o Clean Code e o SOLID. Com essa fundamentação teórica, busca-se garantir uma base sólida para compreender a implementação do aplicativo e contribuir para a área de estudo.

3.1 Backend e Frontend

Backend e frontend são dois conceitos amplamente utilizados no mundo do desenvolvimento de software, principalmente no contexto da Web. Assim, o frontend é caracterizado como a parte visível da aplicação web, com a qual o usuário interage, sendo composto por tudo o que aparece na interface do usuário, como botões, menus, formulários e outros elementos gráficos. Já o backend, por outro lado, é a parte invisível da aplicação web, que lida com a lógica de negócios e o gerenciamento de dados. É responsável por processar solicitações do usuário, acessar o banco de dados, executar cálculos complexos e retornar os resultados para o frontend.

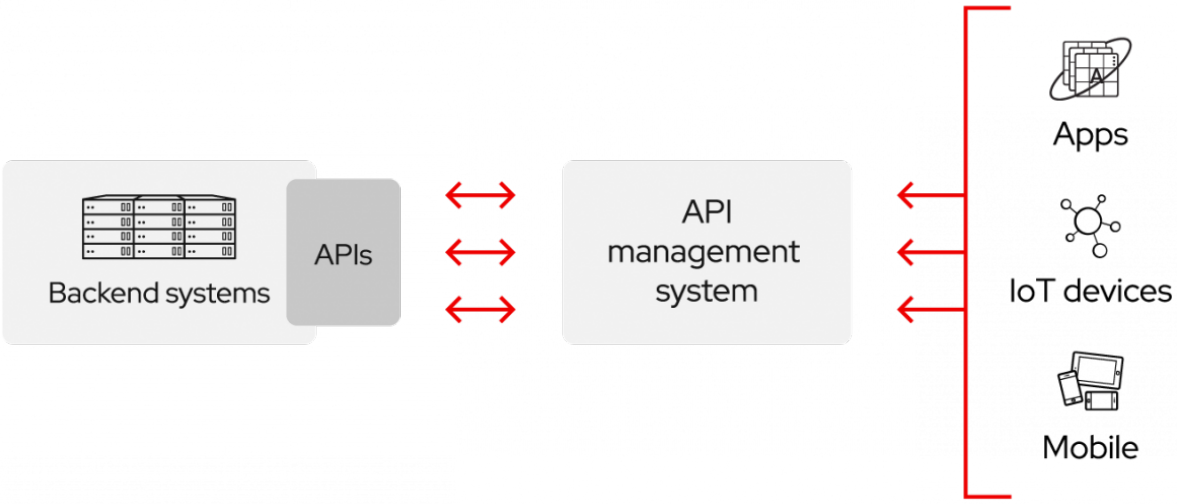
3.2 API

Uma API (Application Programming Interface) é um conjunto de rotinas, protocolos e ferramentas para construir software e aplicações. Ela define a maneira pela qual os componentes de software interagem entre si. APIs geralmente são usadas para permitir que diferentes sistemas se comuniquem uns com os outros, permitindo o compartilhamento de dados e informações. As APIs podem ser públicas ou privadas, dependendo do uso para o qual foram projetadas.

As APIs são amplamente utilizadas em sistemas de integração, como visto na figura 7 a API fornece comunicação entre os sistemas que atuam no backend presentes nos servidores Web com os sistemas finais que atuam no frontend, presentes em diversos dispositivos até mesmo em dispositivos mobile e sistemas de IoT(Internet das Coisas). Logo, as APIs permitem que diferentes sistemas possam se comunicar de forma eficiente e segura, possibilitando a criação de soluções mais robustas e escaláveis.

Uma das principais vantagens de se utilizar uma API é a possibilidade de reaproveitar funcionalidades e dados de outros sistemas, sem a necessidade de desenvolver tudo do zero. Além disso, as APIs permitem que diferentes plataformas possam se comunicar, o que é essencial em soluções que envolvem múltiplos dispositivos e sistemas.

Figura 7 – Fluxo da comunicação via API



Fonte: (REDHAT, 2023)

3.2.1 API Rest

Com a popularidade do uso das APIs para o desenvolvimento de soluções, foi criado um protocolo para ajudar a padronizar a troca de informações, o protocolo SOAP (Simple Object Access Protocol). Nesse sentido, as APIs desenvolvidas com o protocolo SOAP utilizam o XML (Extensible Markup Language) que permite definir e armazenar dados de maneira compartilhável, como formato no envio e recebimento de mensagens, além receberem as solicitações através dos métodos HTTP(Hypertext Transfer Protocol), que serão vistos na seção 3.3. Além disso, foi criada a arquitetura Representational State Transfer (REST), onde diferente do protocolo SOAP a arquitetura Rest não possui padronização oficial, sendo assim elas seguem práticas para serem consideradas RESTful. Dessa forma, segundo o Fielding (2000), as API Rest seguem essas 6 praticas a seguir:

- Cliente Servidor: este principio significa que o cliente não precisa saber como os dados são armazenados ou manipulados pelo servidor permitindo que os dois evoluam independentemente um do outro.
- Sem Estado: cada solicitação do cliente deve conter todas as informações necessárias para entender a solicitação e não pode tirar proveito de qualquer contexto armazenado no servidor. Isso aumenta a visibilidade, confiabilidade e escalabilidade.
- Restrições de cache: as respostas devem ser definidas como cacheable ou não cacheable para evitar que os clientes usem dados desatualizados ou inapropriados em resposta a outras solicitações.
- Interface Uniforme: uma interface uniforme simplifica e desacopla a arquitetura, o que permite que cada parte evolua independentemente. Isso é alcançado aplicando

quatro restrições: identificação de recursos, manipulação de recursos por meio de representações, mensagens auto-descritivas e hipermedia como o motor do estado da aplicação.

- Sistema em Camadas: um sistema em camadas permite que uma arquitetura seja composta por camadas hierárquicas, restringindo o comportamento dos componentes de tal forma que cada componente não possa "ver" além da camada imediata com a qual está interagindo.

3.3 Protocolos HTTP

Sendo o principal método de envio de mensagens entre o cliente(o aplicativo) e o servidor(API) da aplicação criada neste projeto, o protocolo HTTP (Hypertext Transfer Protocol) se caracteriza por ser o protocolo de comunicação utilizado na World Wide Web (WWW) para transferir dados na web. Assim, o HTTP é responsável por permitir que as páginas da web sejam carregadas em um navegador a partir de um servidor web remoto. Dessa forma, o HTTP é um protocolo sem estado, ou seja, cada solicitação feita por um cliente ao servidor é independente e não depende de solicitações anteriores. Isso significa que cada solicitação é tratada pelo servidor como uma nova solicitação, sem levar em consideração solicitações anteriores. Além disso, existem vários tipos de solicitações que um cliente pode fazer ao servidor usando o HTTP, e os principais são:

- GET: o método GET solicita uma representação do recurso identificado pelo URI (Uniform Resource Identifier) especificado na solicitação e retorna uma visualização de acordo com o que foi especificado.
- POST: o POST tem a função de enviar dados para o servidor para que sejam processados, podendo estar acompanhado de um corpo(body) que fica responsável por guardar o conteúdo que será manipulado.
- PUT: método que tem a responsabilidade de atualizar o recurso identificado pelo URI especificado na solicitação.
- DELETE: responsável por remover o recurso identificado pelo URI especificado na solicitação.

3.4 JSON

Presente no corpo das trocas de mensagens entre as aplicações cliente e servidor, o JSON se caracteriza por ser uma formatação leve de troca de dados, gerando um arquivo de fácil leitura para o desenvolvedor e de fácil criação para as aplicações. Nesse sentido, o JSON é em formato texto e completamente independente de qualquer linguagem, por

utilizar de convenções que são familiares para as linguagens, como C, C++, C#, Java, JavaScript, Perl, Python e muitas outras. Por este motivo, o JSON se tornou a principal linguagem de troca de dados na web.

3.5 Banco de Dados

O banco de dados é um sistema que permite o armazenamento e a recuperação de informações de maneira estruturada, organizada e eficiente, sendo uma tecnologia fundamental para muitas aplicações modernas, desde pequenos aplicativos até grandes sistemas empresariais. Dessa forma, um banco de dados é pode ser composto por uma ou mais tabelas, cada uma contendo registros (ou linhas) e campos (ou colunas) que definem os tipos de dados que serão armazenados. Os registros representam as informações a serem armazenadas, enquanto os campos definem os tipos de dados que podem ser armazenados em cada registro.

3.5.1 Banco de Dados Relacional

Existem diversos tipos de banco de dados, como os banco de dados relacionais que são popularmente mais comuns, e os banco não convencionais que trazem um abordagem mais diferentes, como o NoSQL e o de memoria. Nesse sentido, para a abordagem idealizada neste projeto foi necessária a utilização do banco de dados relacional, por ser basear no modelo relacional, que organiza as informações em tabelas com relações predefinidas entre elas, assim sendo o tipo de banco mais compatível com as tecnologia que serão citadas no próximo capitulo

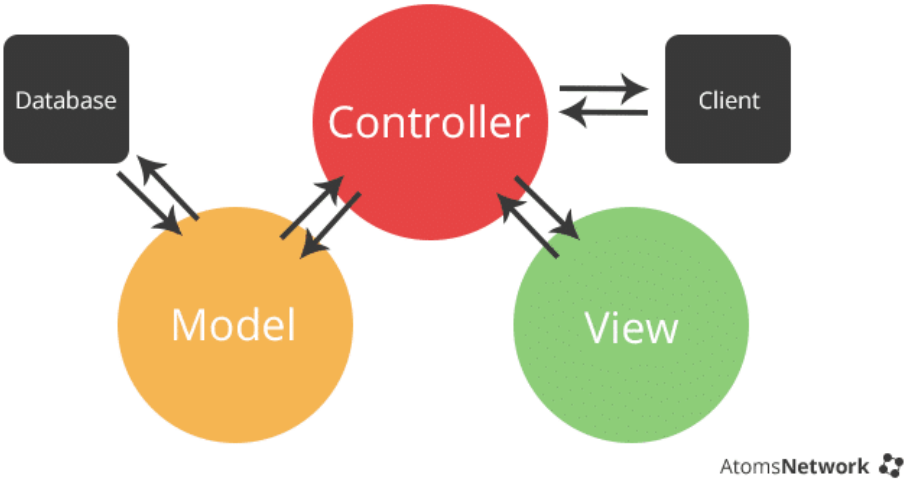
3.6 Framework

Segundo HostGator (2020), um framework é um conjunto de bibliotecas, ferramentas e padrões que fornecem uma estrutura para organizar e desenvolver aplicativos. Ele agiliza o processo de desenvolvimento, permitindo que os desenvolvedores se concentrem nas etapas mais importantes do projeto. É comum que projetos de média e alta complexidade sejam construídos com o auxílio de algum framework popular da linguagem escolhida para o projeto. Para este trabalho, foram utilizados o Django e o Django Rest Framework para a criação do backend, além do Flutter e do MobX para o desenvolvimento do frontend.

3.7 MVC

O MVC (Model-View-Controller) é uma arquitetura de desenvolvimento de software amplamente utilizada na indústria, criada para organizar em camadas as responsabilidades envolvidas dentro de uma aplicação, permitindo uma divisão clara de áreas responsáveis pelas regras de negócios e áreas responsáveis pelas Views.

Figura 8 – Modelo fluxo MVC



Fonte: (FILHO, 2017)

Podemos verificar na figura 8 um fluxograma que ilustra a comunicação das camadas que envolvem o MVC, onde o Model representa a camada de dados, responsável pela manipulação e gerenciamento dos dados da aplicação, o View representa a camada de apresentação, que é responsável por exibir as informações para o usuário final e o Controller representa a camada de controle, que recebe as requisições do usuário e controla a interação entre o Model e o View.

Essa arquitetura permite que as equipes de desenvolvimento trabalhem em paralelo, com os desenvolvedores do backend focados na lógica de negócios e os desenvolvedores do frontend focados na interação com o usuário final. Como resultado, o MVC é uma abordagem eficaz para projetos de desenvolvimento de software de larga escala e é amplamente utilizada em várias tecnologias e linguagens de programação.

3.7.1 Model

O Model é responsável por representar as entidades da aplicação, traves dessa camada será feito o acesso e a manipulação dos dados.

3.7.2 View

A View é responsável por dividir o frontend da aplicação, camada responsável por exibir e consumir os dados manipulados pelo backend. Logo, ela apresentará a parte visual da aplicação, onde será acessível a interação com o usuário final.

3.7.3 Controller

A camada de controle foi criada com o intuito de intermediar a interação entre as duas camadas anteriores, tornando possível a passagem e a manipulação dos dados

enviados entre o backend e o frontend das aplicações.

3.8 Clean Code

Um dos maiores desafios ao se desenvolver uma solução através de construção de um software é manter a sua manutenibilidade, devido a uma serie de fatores durante seu processo de planejamento e principalmente no processo de desenvolvimento do código. Nesse sentido, é importante que o código mantenha sua legibilidade, assim facilitando a compreensão de todos envolvidos principalmente em momentos futuros, logo, é necessário a presença de boas praticas de escrita de código, nomeado atualmente como "Clean Code", ou código limpo. Nesse sentido, segundo o Martin (2008) em seu livro, a proporção de leitura para escrita do código é de 10:1, ou seja, os programadores passam mais parte do tempo lendo seus códigos do os escrevendo em si, logo, conceitos como a nomeação de variáveis e funções de forma clara e concisa, além da separação do código em funções e módulos que desempenham tarefas específicas, podem tornar o código mais conciso e fácil de entender, dessa forma, facilitando a vida útil da aplicação.

3.9 SOLID

Já com o SOLID nós temos um conjunto de princípios de programação orientada a objetos que ajudam a desenvolver software mais robusto e fácil de manter. O acrônimo SOLID representa cinco princípios diferentes, cada um dos quais se concentra em um aspecto específico da programação orientada a objetos:

- S - Single Responsibility Principle (Princípio da Responsabilidade Única): cada classe deve ter uma única responsabilidade clara e bem definida.
- O - Open/Closed Principle (Princípio Aberto/Fechado): as classes devem ser abertas para extensão, mas fechadas para modificação.
- L - Liskov Substitution Principle (Princípio da Substituição de Liskov): as classes derivadas devem ser substituíveis pelas classes base sem afetar o comportamento do programa.
- I - Interface Segregation Principle (Princípio da Segregação de Interfaces): as interfaces devem ser pequenas e focadas em uma única responsabilidade.
- D - Dependency Inversion Principle (Princípio da Inversão de Dependência): as classes de alto nível não devem depender das classes de baixo nível. Em vez disso, as classes de alto nível devem depender de abstrações que podem ser implementadas pelas classes de baixo nível.

3.10 Clean Architecture

A Clean Architecture é um estilo arquitetural de software que busca separar as preocupações de negócio da implementação técnica. A ideia central por trás da Clean Architecture é que o código deve ser organizado em camadas, cada uma com uma responsabilidade clara e bem definida. Essas camadas devem ser independentes e não devem depender de detalhes de implementação específicos, como bancos de dados ou frameworks.

A arquitetura limpa é composta por várias camadas principais, incluindo a camada de Entidades, a camada de Casos de Uso, a camada de Interfaces de Usuário e a camada de Infraestrutura. Cada camada possui uma responsabilidade específica e se comunica com as outras camadas por meio de interfaces bem definidas.

Algumas das vantagens da Clean Architecture incluem a facilidade de manter e modificar o código, a facilidade de testar o código e a facilidade de entender como o código funciona. Ao separar as preocupações de negócio da implementação técnica, a Clean Architecture permite que os desenvolvedores se concentrem na lógica de negócio em vez de se preocupar com detalhes de implementação.

No contexto do projeto de desenvolvimento de uma aplicação mobile para gestão de inventário comercial, a Clean Architecture pode ser útil para garantir que o código seja organizado de forma clara e fácil de entender. Com a arquitetura limpa, é possível separar as preocupações de negócio da implementação técnica e garantir que cada camada do aplicativo mobile tenha uma responsabilidade clara e bem definida.

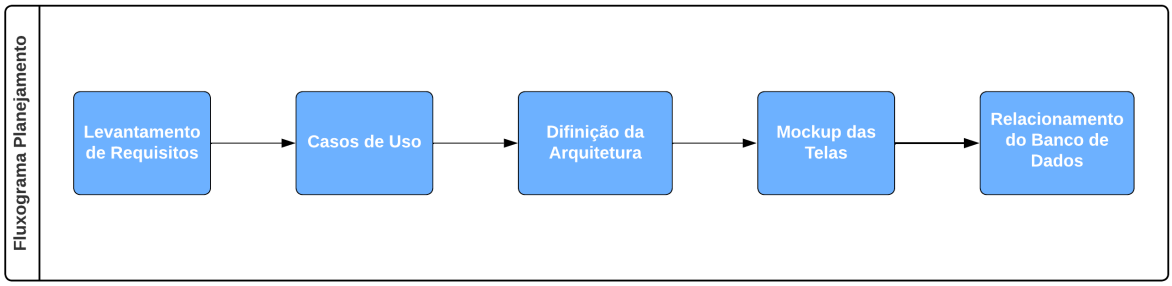
Em resumo, a Clean Architecture é uma abordagem valiosa para o desenvolvimento de aplicativos mobile, pois ajuda a garantir que o código seja organizado de forma clara e fácil de entender. Ao separar as preocupações de negócio da implementação técnica, a Clean Architecture permite que os desenvolvedores se concentrem na lógica de negócio e criem aplicativos mobile eficientes e fáceis de manter.

4 METODOLOGIA E FERRAMENTAS

A seção de Metodologia e Ferramentas descreve as abordagens e ferramentas utilizadas no desenvolvimento do projeto. São apresentadas as metodologias utilizadas para o desenvolvimento, bem como as ferramentas e tecnologias empregadas na sua implementação. A seção aborda desde a definição dos requisitos do projeto, passando pela escolha das ferramentas e tecnologias utilizadas, até a descrição da arquitetura do aplicativo. Entre as tecnologias apresentadas estão o Django, o JWT, o Flutter e o PostgreSQL. Além disso, também são abordadas as metodologias de desenvolvimento de software utilizadas no projeto, como a UML e o Kanban.

4.1 Planejamento do Projeto

Figura 9 – Fluxograma Fase de Planejamento



Fonte: Autor

A fase de planejamento é uma etapa fundamental para o sucesso de um projeto. Durante essa fase, é importante definir claramente os objetivos, as metas e as estratégias do projeto, bem como identificar os recursos necessários para a sua implementação. O planejamento também permite que os desenvolvedores identifiquem os riscos e as dificuldades que podem surgir ao longo do projeto e que desenvolvam estratégias para lidar com esses problemas. Dessa forma, na figura 9 podemos observar o fluxograma das etapas trabalhadas durante esta fase, iniciada com o levantamento dos requisitos do projeto, em seguida com os Casos de Uso, definição da arquitetura, mockup das telas e finalizando com o relacionamento do banco de dados utilizado.

4.1.1 Requisitos do Sistema

A etapa de obtenção de requisitos é um recurso valioso dentro do planejamento do projeto pois descrevem as funcionalidades e características que o software deve ter e são a base para todo o processo de desenvolvimento. Dessa forma, a especificação dos requisitos funcionais e não funcionais do projeto que definem as funcionalidades e características da aplicação.

4.1.1.1 Requisitos Funcionais

Os requisitos funcionais são aqueles que descrevem as funcionalidades e comportamentos esperados do sistema. Nesse sentido, podemos observar na tabela 2 a seguir os requisitos funcionais deste projeto.

Tabela 2 – Tabela Requisitos Funcionais

RF1:	Cadastro de produtos: O software deve permitir que o usuário cadastre novos produtos no sistema, incluindo informações como nome, descrição, preço, categoria e marca.
RF2:	Atualização dos Produtos: O software deve permitir que o usuário atualize as informações ou exclua os produtos do sistema.
RF3:	Controle de estoque: O software deve permitir que o usuário visualize o estoque atual de cada produto.
RF4:	Controle das Remessas: O software deve permitir que o usuário cadastre e visualize todas as remessas relacionadas aquele produto.
RF5:	Atualização das remessas: O software deve permitir que o usuário atualize e exclua as remessas do sistema.
RF6:	Pesquisa de Produtos: O software deve permitir que os usuários possam pesquisar e filtrar os tipos de produtos usando as categoria.

Fonte: Autor

4.1.1.2 Requisitos Não Funcionais

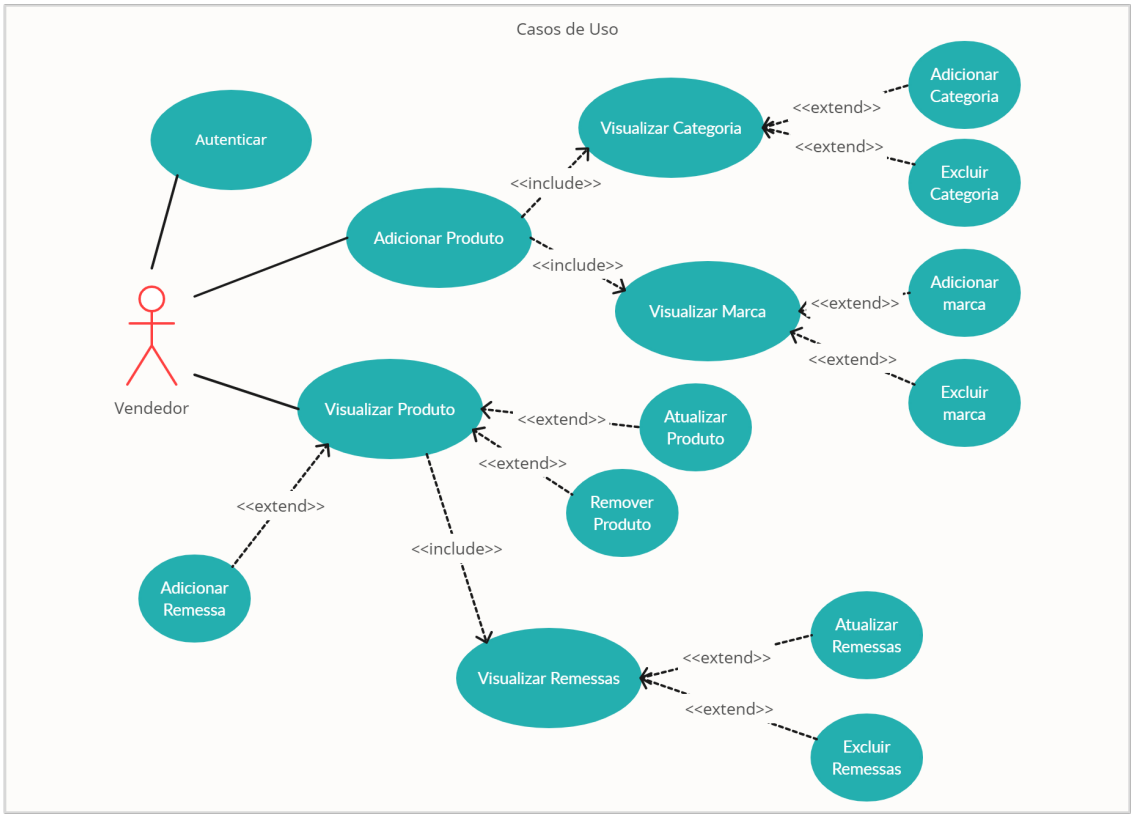
Os requisitos não funcionais são aqueles que descrevem qualidades e características do sistema que não estão relacionadas diretamente com suas funcionalidades. Podemos observar os requisitos não funcionais deste projeto na tabela 3

Tabela 3 – Tabela Requisitos Não Funcionais

RNF1:	Segurança: garantir que o sistema seja seguro e que as informações dos usuários e dos produtos sejam protegidas.
RNF2:	Usabilidade: garantir que o sistema seja fácil de usar e que os usuários possam navegar pelas funcionalidades de forma intuitiva.
RNF3:	Desempenho: garantir que o sistema seja rápido e responsivo, mesmo quando há muitos usuários utilizando o sistema simultaneamente.
RNF4:	Escalabilidade: garantir que o sistema possa suportar um grande número de usuários e que possa ser facilmente escalado para atender às demandas crescentes.
RNF5:	Confiabilidade: garantir que o sistema seja confiável e que possa ser utilizado continuamente sem interrupções ou falhas.
RNF6:	Manutenibilidade: garantir que o sistema seja fácil de manter e que possa ser atualizado facilmente quando necessário.

Fonte: Autor

Figura 10 – Casos de Uso do Projeto



Fonte: Autor

4.1.2 Casos de Uso

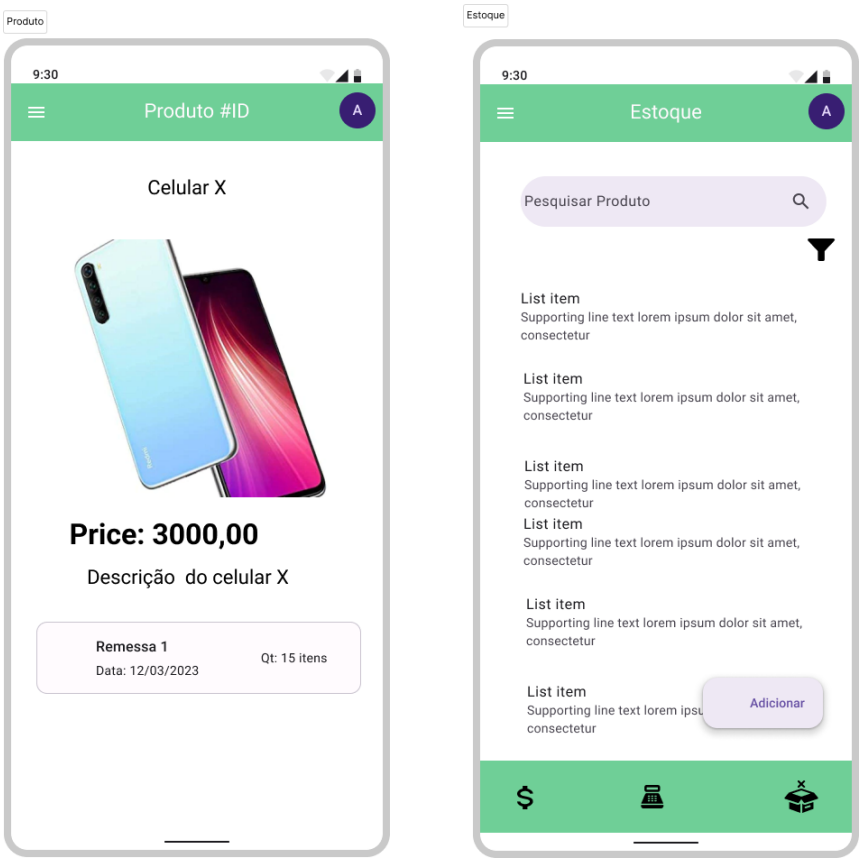
O diagrama de casos de uso faz parte das ferramentas utilizadas para a modelagem de um software, onde utilizamos os artefatos apresentados pela linguagem UML(Unified Modeling Language). Dessa forma, seu uso se faz muito importante, pois possibilita que consigamos organizar de forma mais clara os atores que irão interagir com o software que será desenvolvido, além de especificar as suas possíveis ações dentro do sistema.(CREATELY, 2021) Logo, com a utilização dessa ferramenta podemos verificar o nosso sistema do ponto de vista de seus futuros usuários, assim facilitando a comunicação entre os desenvolvedores que possuem um ponto de vista técnico com os clientes que possuem uma visão mais prática da ferramenta que está sendo criada com o software.

Na figura 10 podemos ver o fluxograma de uso do aplicativo, onde o usuário ”Vendedor”, tem seu primeiro acesso a aplicação através do modulo de autenticação, após isso ele é direcionado para o ”modulo de Stock”, onde ele consegue acesso as funções e entidades que envolvem os produtos.

4.1.3 Mockup das Telas

O mockup das telas de um aplicativo é uma representação visual estática ou interativa do seu design e layout. Ele permite visualizar como será a aparência e o fluxo de interação do aplicativo antes mesmo de ser desenvolvido. O objetivo principal de um mockup é validar a ideia do projeto, obter feedback dos usuários e colaboradores e facilitar a comunicação entre as partes interessadas. Diante do exposto, podemos observar na figura 11 dois exemplos dos mockups criados para as telas Estoque e Produtos, onde temos uma visualização inicial de como será construída as telas do aplicativo mobile.

Figura 11 – Mockup das Telas Produto e Estoque



Fonte: Autor

4.1.4 Arquitetura

A figura 12 abaixo demonstra o frontend da aplicação que consiste em uma aplicação mobile desenvolvida em Flutter utilizando a arquitetura modular. Dessa forma, com a presença dos módulos de Autenticação, Inventario e Gestão de Produto, a aplicação permite a gestão de estoque de forma simples e intuitiva, com telas para login, gerenciamento de produtos, categorias e lotes. Diante do exposto, para o desenvolvimento do frontend, foram utilizadas diversas ferramentas e metodologias, como a criação de mockups de telas, a definição dos requisitos do projeto e a criação de casos de uso. Além disso, foi dada

atenção especial ao design e à usabilidade da aplicação, com o objetivo de proporcionar uma experiência agradável e intuitiva para o usuário.

Figura 12 – Diagrama da arquitetura do frontend do Projeto

Fonte: Autor

Na figura 13 temos a ilustração de como ficará o backend do projeto, demonstrando a sua criação através do framework Django, onde será criada as APIs e as Regras de Negócios da nossa aplicação, além das chamadas e requisições com o banco de dados que será o PostgreSQL. Dessa forma, a API se caracteriza como uma parte importante do projeto, que fornece os dados necessários para a aplicação mobile. Para garantir a segurança dos dados, a API utiliza o método de autenticação JWT (JSON Web Token), que permite que os usuários se autentiquem no sistema de forma segura. Além disso, foram implementadas permissões de usuário para controlar o acesso aos recursos da API. Por fim, a API foi implantada em um servidor na nuvem, utilizando o serviço Google Cloud Platform

Figura 13 – Diagrama da arquitetura do Backend do Projeto

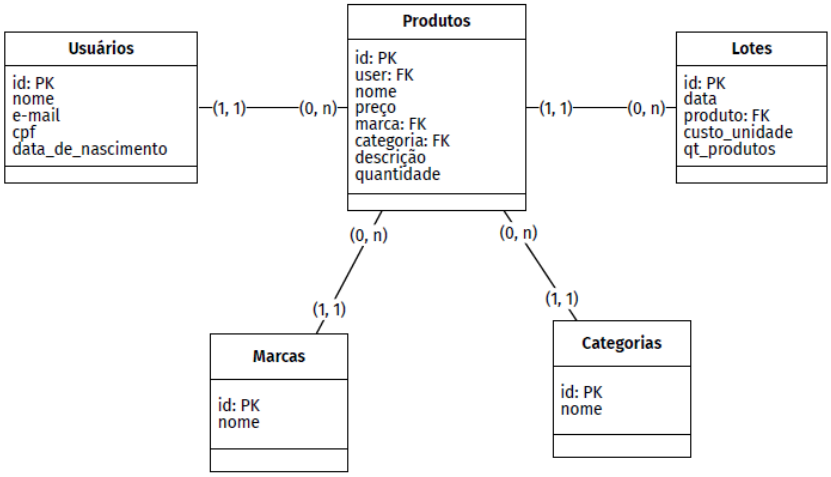
Fonte: Autor

4.1.5 Relacionamento do Banco de Dados

O relacionamento de um banco de dados refere-se à forma como as diferentes entidades ou tabelas do sistema se relacionam entre si para armazenar e recuperar informações. Esses dados são geralmente organizados em tabelas, que consistem em linhas e colunas. Ademais, este relacionamento entre as tabelas é estabelecido através de chaves, que são colunas que identificam exclusivamente cada registro em uma tabela. Existem diferentes tipos de relacionamentos entre tabelas de dados, sendo eles listados abaixo:

- Relacionamento Um-para-Um(1:1): Para este tipo de relacionamento, um registro em uma tabela está associado a apenas um registro em outra tabela e vice-versa.
- Relacionamento Um-para-Muitos(1:N): Nesse tipo de relacionamento, um registro em uma tabela pode estar associado a vários registros em outra tabela, mas cada registro nessa outra tabela está associado a apenas um registro na primeira tabela.
- Relacionamento Muitos-para-Muitos (N:N): Nesse tipo de relacionamento, vários registros em uma tabela podem estar associados a vários registros em outra tabela. Para modelar esse tipo de relacionamento, é necessário criar uma tabela intermediária que mapeie as associações entre os registros das duas tabelas.

Figura 14 – Modelo Logico Banco de Dados



Fonte: Autor

Na figura 14, podemos observar o relacionamento estabelecido para o projeto, no qual temos as tabelas Usuário, Produto, Categoria, Marca e Lote interligadas. Essas conexões representam um relacionamento de 1:N entre a tabela Usuários e a tabela Produtos, simbolizando ”1”Usuário para ”N”Produtos. De maneira semelhante, as tabelas Marcas e Categorias também se relacionam com a tabela Produtos de forma 1:N. Por fim, temos o mesmo relacionamento com a tabela de Lotes, simbolizando N Lotes para 1 Produto .

4.2 Ferramentas Utilizadas

4.2.1 Notion

O Notion é uma ferramenta de produtividade que pode ser utilizada para auxiliar no desenvolvimento de projetos. Ele permite a criação de documentos, notas, listas de tarefas, bancos de dados e muito mais, tudo em um único lugar. Com o Notion, é possível organizar e gerenciar todas as informações relacionadas ao projeto de forma eficiente e colaborativa. Uma das principais vantagens do Notion é a sua flexibilidade. A ferramenta permite que os usuários criem documentos e bancos de dados personalizados, adaptados às suas necessidades específicas. Isso significa que é possível criar fluxogramas, diagramas, listas de tarefas e outros tipos de conteúdo diretamente no Notion.(NOTION, 2019)

No contexto deste projeto, o Notion foi utilizado para gerenciar as tarefas e informações relacionadas ao projeto. Dessa forma, foi possível estruturar as informações de cada uma das etapas, facilitando cada processo e possibilitando a organização de cada fase do processo.

Em resumo, o Notion é uma ferramenta de produtividade flexível e colaborativa que pode ser utilizada para auxiliar no desenvolvimento de projetos. Com recursos como a

criação de documentos personalizados, compartilhamento de arquivos e visualização de calendário, ele pode ser uma ferramenta valiosa para gerenciar as informações e tarefas relacionadas a um projeto.

4.2.2 JWT

JWT (JSON Web Token) é um padrão aberto que define um formato compacto e independente de plataforma para transmitir com segurança informações entre partes como um objeto JSON. O objetivo principal do JWT é garantir que as informações transmitidas sejam confiáveis e seguras, sem a necessidade de transmissão de dados sensíveis pelo cabeçalho da requisição.(OKTA, 2023)

Um JWT é composto por três partes: o cabeçalho, o payload e a assinatura, como vistos na figura 15. O cabeçalho contém informações sobre o tipo de token e o algoritmo de criptografia utilizado para assinar o token. O payload contém as informações que serão transmitidas, como o nome do usuário ou a data de expiração do token. A assinatura é gerada a partir da combinação do cabeçalho e do payload, e é usada para garantir a autenticidade do token.

Os tokens JWT são amplamente utilizados em aplicações web e mobile para autenticação e autorização de usuários. Eles são uma alternativa mais segura do que as tradicionais cookies de sessão, que podem ser vulneráveis a ataques de CSRF (Cross-Site Request Forgery). Além disso, os tokens JWT podem ser usados em diferentes plataformas, o que os torna uma opção conveniente para aplicações que utilizam múltiplos dispositivos ou serviços.

4.2.3 Insomnia

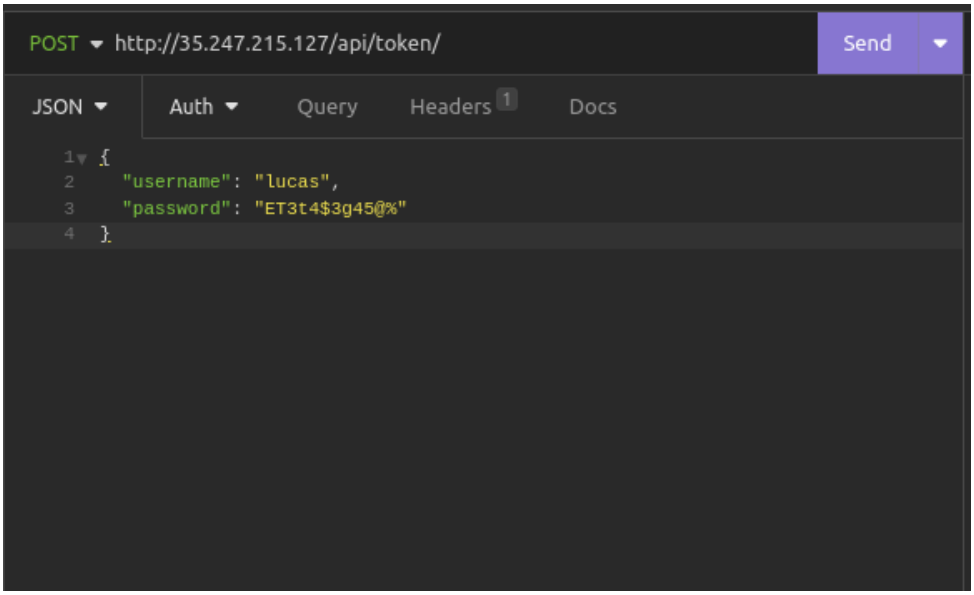
O Insomnia é um cliente RESTful que permite a criação e execução de requisições HTTP e HTTPS. Ele é uma ferramenta muito útil para desenvolvedores que precisam testar APIs e serviços web durante o processo de desenvolvimento. Dessa forma, com o Insomnia, foi possível criar e salvar coleções de requisições, que podem ser organizadas por categoria, visto na figura 16. Além disso, foi possível salvar as respostas das requisições para referência futura.

A utilização do Insomnia facilitou o teste das requisições HTTP realizadas entre o cliente e a API, permitindo verificar visualmente o funcionamento do servidor e se as respostas estavam de acordo com o esperado de cada solicitação. Dessa forma, testes como a autenticação (Figura 17) ou uma simples requisição GET (Figura 18) foram realizados com facilidade graças à ferramenta.

Figura 15 – Estrutura do JWT

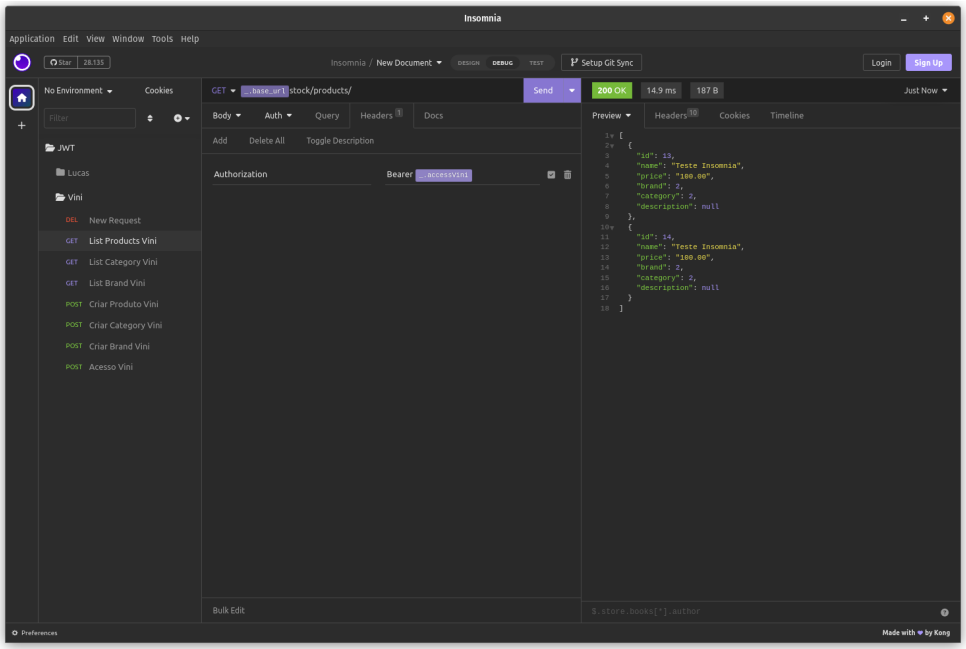
Fonte: (OKTA, 2023)

Figura 17 – Teste de Autenticação do Insomnia em ambiente de Desenvolvimento



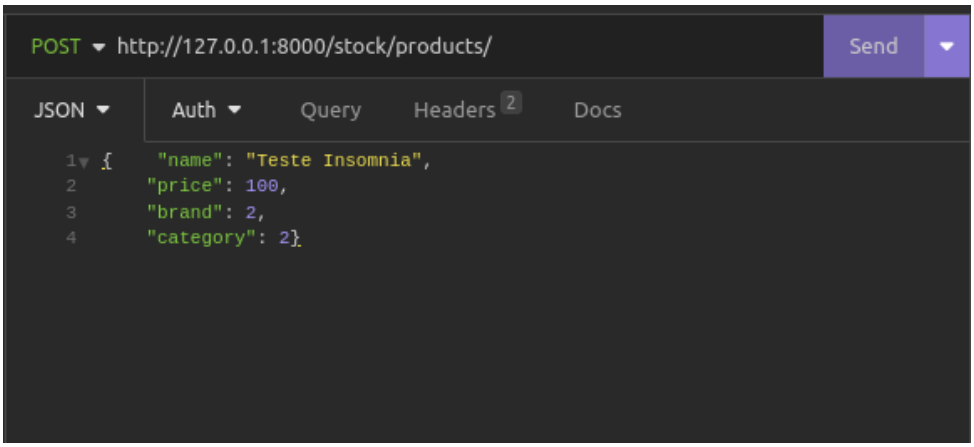
Fonte: Autor

Figura 16 – Ambiente do Insomnia



Fonte: Autor

Figura 18 – Requisição POST do Insomnia em ambiente de Desenvolvimento



Fonte: Autor

O Insomnia é uma ferramenta gratuita e de código aberto, disponível para Windows, Mac e Linux. Logo, tornando possível o uso no Sistema Operacional Linux em que esse projeto foi construído. Em resumo, ele é uma das melhores opções para desenvolvedores que precisam testar APIs e serviços web de forma rápida e eficiente.

4.2.4 PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto. Ele é um dos bancos de dados mais avançados e populares do mercado, com

recursos avançados de segurança, escalabilidade e desempenho. O PostgreSQL é conhecido por sua alta confiabilidade e integridade de dados, o que o torna ideal para aplicações críticas e alta escalabilidade.

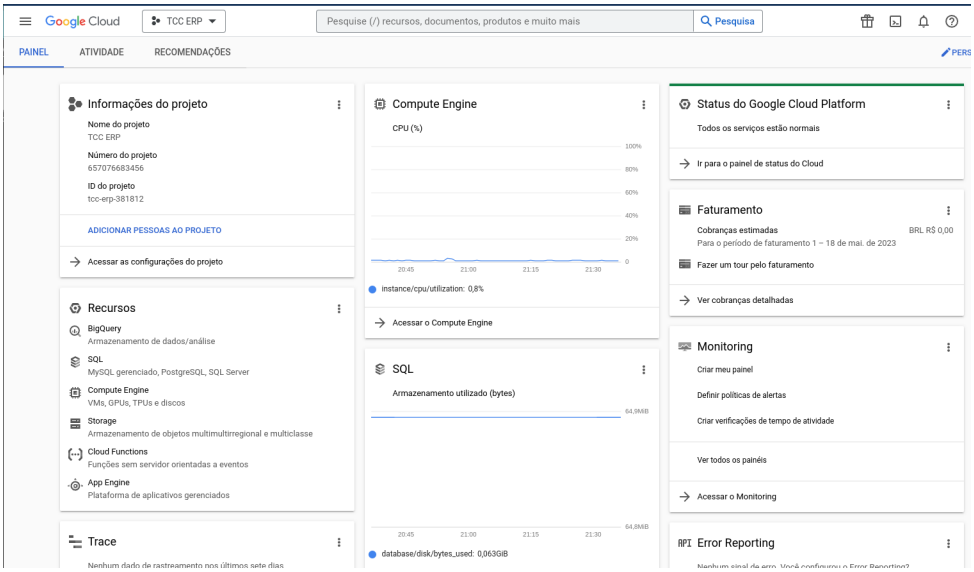
Entre os recursos do PostgreSQL, incluem-se suporte a transações ACID que é uma estrutura de banco de dados que garante seu correto funcionamento e impede que dados sejam corrompidos ou perdidos no processamento de transações, assim garantindo que as transações sejam concluídas com sucesso ou revertidas em caso de falha; suporte a várias linguagens de programação, incluindo C, Java, Python e Ruby; e suporte a vários formatos de dados, incluindo XML e JSON.

O PostgreSQL também é altamente escalável, o que significa que ele pode lidar com grandes volumes de dados e usuários simultâneos. Ele oferece recursos de replicação e clusterização, permitindo que vários servidores do PostgreSQL trabalhem juntos para fornecer serviços de banco de dados altamente disponíveis e redundantes. Para desenvolvedores, o PostgreSQL oferece uma ampla gama de ferramentas e recursos, incluindo suporte a SQL avançado, gatilhos, procedimentos armazenados, funções e muito mais. Ele também é compatível com muitos frameworks e plataformas de desenvolvimento, como o Django, framework utilizado neste projeto.

4.2.5 Google Cloud

O Google Cloud é um dos inúmeros serviços disponibilizados pela empresa Google. Seu foco principal esta em fornecer uma plataforma em nuvem para desenvolvimento e hospedagem de soluções em software online. Para o contexto deste projeto, foi utilizado um dos produtos fornecidos por esta plataforma que é o de criação de maquina virtual, onde foi criado um servidor operando o Sistema Operacional Ubuntu que ficou responsavel por fornecer a solução de deploy e hospedagem da API, tornando possível que a mesma operasse online.

Figura 19 – Painel do Google Cloud



Fonte: Autor

4.2.6 Python

O Python é uma linguagem de programação de alto nível lançada no ano de 1991, criada pelo matemático e programador Guido van Rossum. Atualmente a linguagem conta com o desenvolvimento comunitário mantendo seu código fonte aberto, desse modo conseguindo se tornar a linguagem com maior popularidade no mundo hoje, segundo o índice de popularidade do PYPL(Popularity of Programming Language Index), visto na figura 20, onde foi analisado a frequência em que os tutoriais e materiais responsáveis pela linguagem foram pesquisados nos sites de busca. Dessa forma, por ser uma linguagem de forte tipagem e com orientação a objetos, o Python é extremamente útil para o desenvolvimento de aplicações robustas e seguras, além disso, se caracteriza por ser uma linguagem interpretada e dinâmica, trazendo velocidade e facilidade para ser implementada e executada, logo se tornando uma ótima opção para resolução de problemas que necessitam de uma rápida implementação junto com uma fácil aplicação.

Figura 20 – Ranking das linguagens mais populares segundo o PYPL

Worldwide, Oct 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.3 %	-1.8 %
2		Java	17.2 %	-0.9 %
3		JavaScript	9.69 %	+0.4 %
4		C#	7.2 %	-0.2 %
5		C/C++	6.45 %	-0.5 %

Fonte: (WORLDWIDE, 2023)

4.2.6.1 Django

O Django é uma ferramenta de desenvolvimento web de código aberto escrita em Python, criada para facilitar o desenvolvimento de aplicações para a web, usando a arquitetura MVC. Com o Django, é possível construir o backend da aplicação com as regras de negócios desenvolvidas no Model e a interface do usuário disponível na View. Isso permite uma divisão clara de áreas responsáveis pelas regras de negócios e áreas responsáveis pela apresentação dos dados para o usuário final. O Django é uma escolha popular para projetos de desenvolvimento de software de larga escala e é amplamente utilizado na indústria.

4.2.6.2 Django Rest Framework

O Django Rest Framework (DRF) é uma extensão do Django que permite a criação de APIs RESTful em Python. Desse modo, ele fornece uma estrutura poderosa para construir APIs web, incluindo suporte para autenticação, serialização de dados e navegabilidade. O DRF segue a arquitetura do Django, usando o padrão Model-View-Controller (MVC) para organizar as responsabilidades da aplicação. Dessa forma, o DRF é uma escolha popular para projetos de desenvolvimento de software de larga escala e é amplamente utilizado na indústria.

4.2.7 Dart

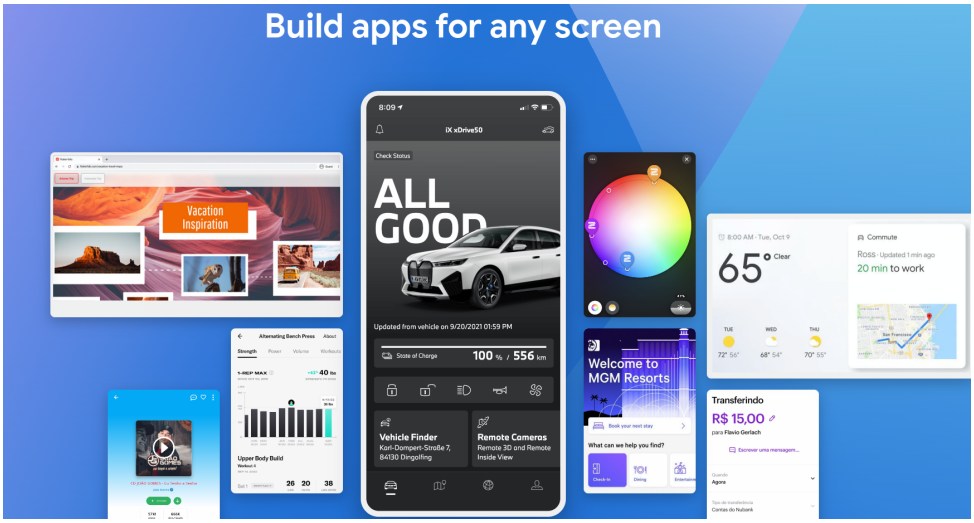
O Dart é uma linguagem de programação fortemente tipada que foi desenvolvida pela Google no ano de 2011, cujo o objetivo era substituir o JavaScript na criação de softwares para Web. No entanto, a linguagem não obteve sucesso nesse propósito, e acabou

sendo deixada de lado até 2015, quando foi anunciada como a linguagem oficial do mais novo framework do Google, o Flutter.

4.2.7.1 Flutter

O flutter é um framework de desenvolvimento, criado pela Google e implementado no ano de 2015. Inicialmente teve seu foco no desenvolvimento nativo de aplicações mobile, tanto para o sistema operacional do Google, o Android, quanto para o sistema da Apple, o IOS, mas atualmente possui implementações para outras plataformas como a Web e Desktop, assim podendo criar aplicações nativas tanto para Windows quanto para as distribuições Linux. Dessa forma, com a utilização do Flutter podemos ter a extrema facilidade de realizar o mesmo projeto e interface gráfica em todas as plataformas em que o mesmo opera, ou seja, a mesma interface criada para a aplicação mobile será utilizada para a aplicação Web.

Figura 21 – Exemplo de aplicações criadas com o Flutter



Fonte: (DEVELOPERS, 2022)

4.2.7.2 MobX

O MobX é uma biblioteca de gerenciamento de estados inicialmente desenvolvida para JavaScript, com o objetivo de gerenciar estados e manter a reatividade das aplicações web. Atualmente, é desenvolvida e aprimorada pela comunidade do Flutter, tornando-se um dos padrões de desenvolvimento mais recomendados para a criação de aplicações usando o Flutter.

O MobX oferece uma forma simples e eficiente de gerenciar o estado da aplicação, especialmente quando comparado com outras alternativas disponíveis no mercado. Ele permite que os desenvolvedores definam a lógica de negócios da aplicação de uma maneira que fique clara e fácil de entender, melhorando a manutenibilidade do código. Além disso,

o MobX oferece recursos avançados como lazy-loading, caching, e tempo de execução da ação, que podem melhorar significativamente o desempenho da aplicação.

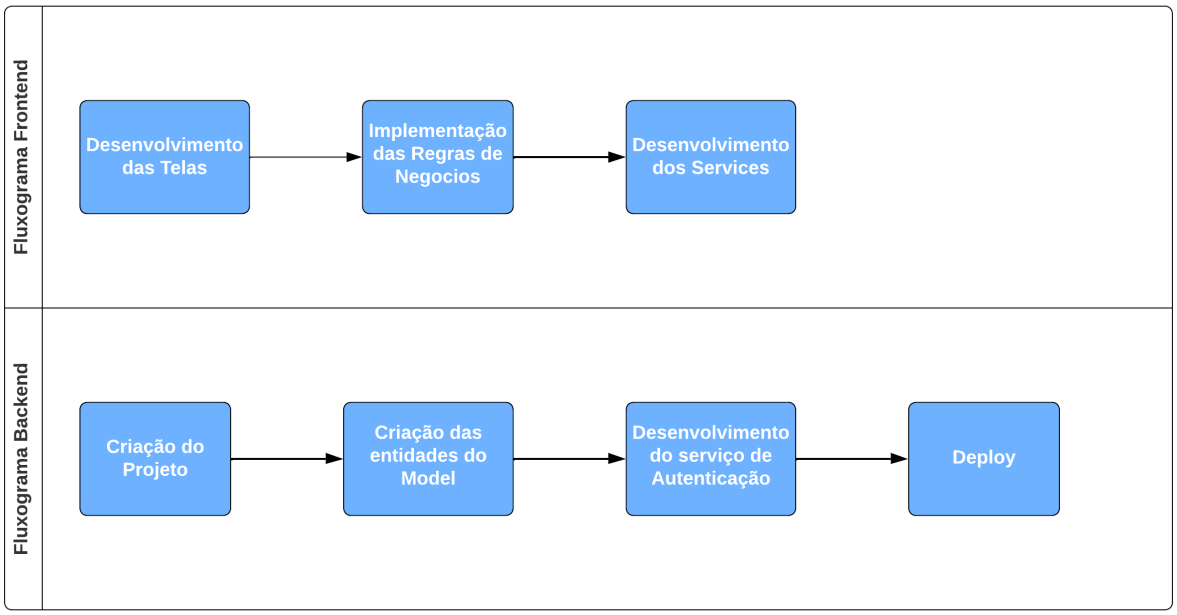
4.2.7.3 Modular

Inspirado no framework Web Angular, o modular é uma arquitetura de projeto construída para segmentar as áreas de funcionalidades de um aplicativo em módulos, dessa forma , melhorando a escalabilidade da aplicação, tornando mais fácil e prático para manutenções futuras. Dessa forma, o modular padroniza os projetos do Flutter seguindo o padrão do MVC, além de propor facilidade para injeção de dependências e criação de rotas modulares.

5 DESENVOLVIMENTO

Nesta seção, serão apresentados todos os passos do desenvolvimento do projeto proposto, com a descrição detalhada do processo de implementação da solução. Serão apresentadas as tecnologias e ferramentas utilizadas, bem como as principais decisões de design e arquitetura do sistema. Como ilustrado na figura 22 o desenvolvimento do projeto foi realizado em diversas etapas, com a finalidade de garantir a qualidade e eficiência do produto final. Primeiramente, foi feita a especificação do setup, que incluiu a escolha das tecnologias e ferramentas a serem utilizadas. Em seguida, foram realizados o desenvolvimento do backend e frontend. O backend foi construído utilizando Django e Django Rest Framework, com o objetivo de criar uma API que fornecesse dados para a aplicação mobile. Já o frontend foi desenvolvido utilizando Flutter e MobX, com o objetivo de criar uma aplicação mobile que permitisse a gestão de estoque de forma simples e intuitiva.

Figura 22 – Fluxograma Desenvolvimento



Fonte: Autor

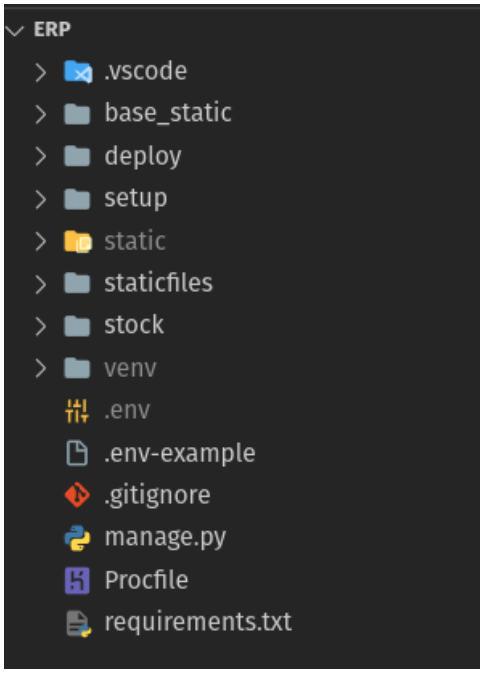
5.1 Desenvolvimento do Backend

Para iniciar o desenvolvimento do backend, foi necessário escolher uma linguagem de programação e um framework para a criação da API. Como visto no capitulo anterior, neste trabalho, foi escolhido o framework Django em conjunto com o Django Rest Framework para criar a API que fornecerá os dados para a aplicação mobile. Em seguida, ilustrado na figura 23 foi definido a estrutura de pastas do projeto e criar classes para cada

entidade do sistema, como Produtos, Categorias e Lotes, cada uma com suas respectivas views, serializers e rotas. Também foi necessário definir como seria feita a autenticação dos usuários, que neste caso será realizada utilizando JSON Web Tokens (JWT).

5.1.1 Estrutura de Pastas

Figura 23 – Estrutura de Pastas Frontend



Fonte: Autor

A figura 23 demonstra a ilustração da estrutura de pastas definida para o backend deste projeto, onde podemos observar as pastas 'Setup' e 'Stock', responsáveis por armazenar os principais arquivos do projeto. Diante do exposto, ao executar o comando de inicialização do Projeto no Django foi criada a pasta Setup com os arquivos principais para a execução do Django, dentro da pasta Setup temos o arquivo 'settings.py' que contem as configurações necessárias para executar o framework com as adaptações próprias para cada projeto. Em seguida, foi criada a pasta Stock, que ficou responsável por manter a estrutura do Modulo de Estoque, contendo os arquivos de model, views, serializers e de rotas.

Na imagem também observamos a presenças de outras pastas como a 'venv', que esta abrigando o ambiente virtual criado para este projeto com uma versão própria da linguagem Python, junto com os pacotes escolhidos para a construção da API. Além disso, temos as pastas 'static', 'staticfiles' e 'base_static' que estão abrigando os arquivos estáticos do projeto, como as imagens, arquivos '.css' e '.js'. Por ultimo, temos a pasta '.vscode' que só está presente no ambiente de desenvolvimento, por conter arquivos de configurações que o editor de texto Visual Studio Code utiliza.

5.1.2 Definição dos Models

Como podemos observar na seção 4.1.5 no nosso projeto temos o relacionamento das entidades, Usuário, Produto, Categoria, Marca e Lote. Dessa forma, através do ORM(Mapeamento objeto-relacional) presente no Django, é possível ordenar, consultar e construir a relação do banco de dados com certo nível de facilidade, graças a abstração criada pelo framework. É possível dizer em outras palavras, que no lugar de realizar uma ação direta no banco de dados com código SQL por exemplo, utilizamos o ORM como ponte de comunicação entre o banco e a aplicação. Na figura 24 é visível a utilização do ORM na construção das entidades Category(Categoria) e Brand(Marca), onde escrevemos os models utilizando a sintaxe padrão do Python na estrutura requisitada pelo Django, adicionando os atributos presentes no diagrama da figura 14 importando as configurações do pacote models fornecido pelo próprio Django nativamente.

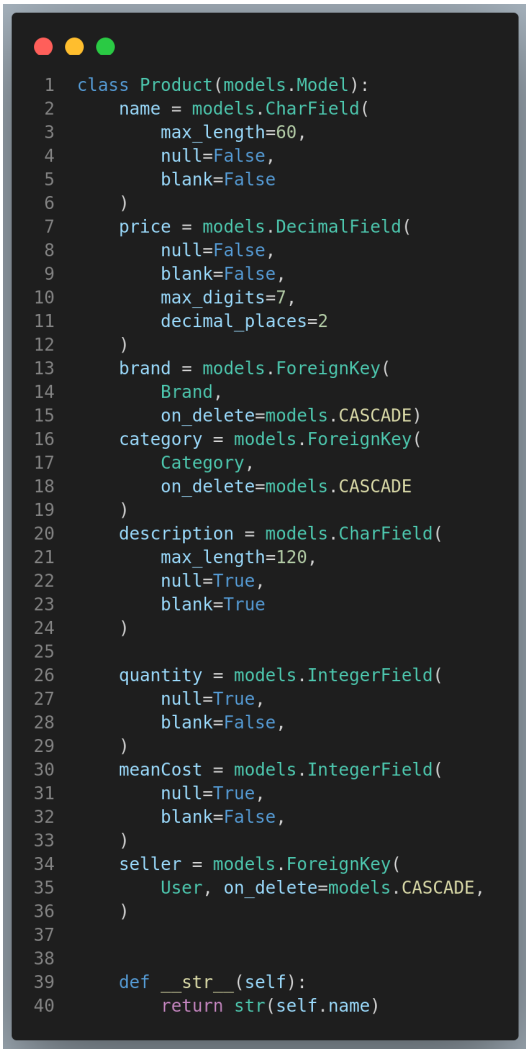
Figura 24 – Classes Category e Brand

```python
from django.db import models
from rest_framework.authtoken.models import Token
from django.contrib.auth.models import User

class Category(models.Model):
    name = models.CharField(max_length=65)
    seller = models.ForeignKey(
        User, on_delete=models.CASCADE,
    )

    def __str__(self):
        return self.name

class Brand(models.Model):
    name = models.CharField(max_length=65)
    seller = models.ForeignKey(
        User, on_delete=models.CASCADE,
    )

    def __str__(self):
        return self.name
```

Fonte: Autor

Na figura 35 podemos notar a presença do relacionamento das entidades Usuário, Brand(Marca) e Category(Categoria) com o nosso objeto Produto. Novamente vemos a presença e facilidade do ORM do Django, que nos permite criar o relacionamento N:1(N Produtos para 1 objeto X), dentro do próprio objeto Produto, onde criamos os parâmetros do tipo Objeto que no banco de dados irá armazenar o identificador do objeto relacionado.

Figura 25 – Classe Product



```python
class Product(models.Model):
    name = models.CharField(
        max_length=60,
        null=False,
        blank=False
    )
    price = models.DecimalField(
        null=False,
        blank=False,
        max_digits=7,
        decimal_places=2
    )
    brand = models.ForeignKey(
        Brand,
        on_delete=models.CASCADE)
    category = models.ForeignKey(
        Category,
        on_delete=models.CASCADE
    )
    description = models.CharField(
        max_length=120,
        null=True,
        blank=True
    )

    quantity = models.IntegerField(
        null=True,
        blank=False,
    )
    meanCost = models.IntegerField(
        null=True,
        blank=False,
    )
    seller = models.ForeignKey(
        User, on_delete=models.CASCADE,
    )


    def __str__(self):
        return str(self.name)
```

Fonte: Autor

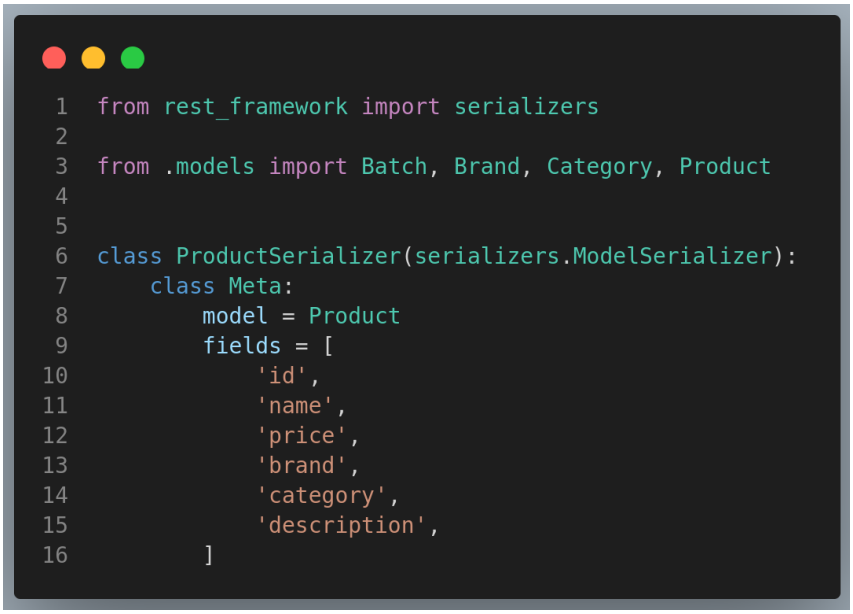
5.1.3 Configurando os Serializers

O Django Rest Framework traz diversas melhorias à estrutura principal do Django, permitindo uma integração mais eficiente e o funcionamento adequado da API desenvolvida. Um dos recursos notáveis é a adição da classe Serializers, uma ferramenta poderosa para a conversão de tipos de dados complexos, como instâncias do modelo Django, em tipos de dados Python que podem ser facilmente renderizados em JSON, XML ou outros formatos.

Os Serializers oferecem um controle preciso sobre a saída dos dados serializados, além de possibilitar a validação dos dados de entrada. Uma vantagem adicional é a geração automática de campos, com base nos campos do modelo, o que simplifica bastante o processo de criação da API. Além disso, são fornecidas implementações padrão para os métodos create() e update(), tornando o desenvolvimento mais ágil e eficiente. Na figura 26 é possível observar a utilização do Serializer no projeto, onde através da classe ProductSerializer foi possível definir os atributos que seriam serializados e possibilitados

de serem utilizados nas requisições da API.

Figura 26 – Classe Serializer



```python
from rest_framework import serializers

from .models import Batch, Brand, Category, Product


class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = [
            'id',
            'name',
            'price',
            'brand',
            'category',
            'description',
        ]
```

Fonte: Autor

5.1.4 Definindo as Views

As Views no Django Rest Framework são responsáveis por definir a lógica de negócios da API e determinar como as solicitações HTTP serão tratadas. Elas são o ponto de entrada para o processamento de solicitações recebidas pela API e podem ser usadas para retornar dados ao cliente, processar dados de entrada e realizar outras operações relevantes à aplicação. Dessa forma, as views no Django Rest Framework são altamente personalizáveis e podem ser estendidas para atender às necessidades específicas de uma aplicação. Elas podem ser usadas para implementar autenticação de usuários, processar solicitações de entrada e saída, validar dados e realizar outras operações relevantes à aplicação.

Como visto na figura 27, nós temos a classe ProductViewSet, que é responsável por processar os dados relacionado aos Produtos, nela podemos observar parâmetros como o 'queryset' que organiza os produtos em ordem decrescente, o parâmetro 'search_fields' que organiza pelo nome no momento da busca e o campo 'fielterset_fields' que organiza de acordo com os filtros de categoria e marca. Junto a isso, temos o permission_classes que atribui bloqueio a esta classe, dando permissão de uso para apenas os usuários logados no sistema.

Figura 27 – Classe ProductViewSet

```python
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.order_by('-id')
    serializer_class = ProductSerializer
    filter_backends = [DjangoFilterBackend,filters.OrderingFilter,filters.SearchFilter]
    ordering_fields = ['name']
    search_fields = ['name']
    filterset_fields = ['category','brand']
    permission_classes = [IsAuthenticated]
```

Fonte: Autor

5.1.5 Criando o sistema de Rotas

As URLs no Django são responsáveis por mapear as solicitações HTTP recebidas pela aplicação para as views correspondentes. Elas são definidas em um arquivo de configuração chamado urls.py, visto na figura 23, que é um dos principais arquivos de configuração de uma aplicação Django. O urls.py contém uma lista de padrões de URL, cada um dos quais é associado a uma view específica. Conforme visto na figura 28, foi criadas as rotas de interação com todas entidades criadas para a API, junto com a definição do nome do modulo e por fim, o parâmetro 'urlpatterns' que define as rotas que irão ser usadas na aplicação.

Figura 28 – Arquivo urls.py da pasta Stock

```python
router = DefaultRouter()
router.register(r'products', ProductViewSet,basename='Products')
router.register(r'categorys', CategoryViewSet,basename='Categorys')
router.register(r'brands', BrandViewSet,basename='Brand')
router.register(r'batchs', BatchViewSet,basename='Batchs')

app_name = 'stock'

urlpatterns =[
    path('',include(router.urls)),
    path('product/<int:pk>/batchs/',ListBatch.as_view()),
]
```

Fonte: Autor

Ademais, como pode ser visto na figura 29 temos um segundo arquivo urls.py, que está armazenado na pasta Setup, que contem parâmetros principais que devem ser herdados pelos outros módulos que podem ser acrescentados na API, como as rotas de autenticação e documentação por exemplo.

Figura 29 – Parâmetro urlpatterns do arquivo urls.py da pasta Setup

```python
urlpatterns = [
    path('admin/',admin.site.urls),
    path('stock/',include('stock.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/token/verify/', TokenVerifyView.as_view(), name='token_verify'),
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),

]+ static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Fonte: Autor

5.1.6 Implementando a autenticação

Para realizar a autenticação no sistema, foi utilizada a tecnologia de JSON Web Tokens (JWT). Como visto na seção 4.2.2 , ao fazer login, o backend do sistema gera um JWT que é enviado para o frontend e armazenado no dispositivo do usuário. Nesse sentido, para a implementação do JWT, foi necessária a instalação do pacote 'djangorestframework-jwt', esse pacote inclui uma view que pode ser utilizada para gerar o token e uma classe de autenticação que foi adicionada ao arquivo 'settings.py' do projeto, visto na figura 30. Nesse sentido, podemos observar na imagem a presença dos atributos 'ACCESS_TOKEN_LIFETIME' que determina o tempo de vida do Token de acesso, junto com 'REFRESH_TOKEN_LIFETIME' que determina o tempo de vida do Token de refresh. Dessa forma. por estar presente em todas as solicitações feitas com a API, o Token de acesso possui um tempo de vida muito menor e necessita do Refresh Token para ser atualizado sempre que o tempo expirar. Em seguida após o tempo do Refresh Token acabar também, o usuário é obrigado a realizar novamente a autenticação no sistema, assim garantindo uma melhor segurança ao utilizar a API.

Figura 30 – Configuração do JWT no arquivo settings.py

```python
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ]
}
SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=360),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "BLACKLIST_AFTER_ROTATION": False,
    "SIGNING_KEY": os.environ.get('SECRET_KEY_JWT','INSECURE'),
    "AUTH_HEADER_TYPES": ("Bearer",),
}
```

Fonte: Autor

5.2 Deploy da Aplicação

Para executar a etapa de deploy da aplicação foi necessário a criação de uma conta no serviço Google Cloud, visto na seção 4.2.5. Além disso, foi utilizado a função de criação de Maquina Virtual no proprio serviço, onde foi configurado o ambiente virtual que executaria o backend da aplicação no servidor do Google. Dessa forma, foram instalados o Gunicorn e o Nginx. O Gunicorn é um servidor web HTTP para aplicativos Python, enquanto o Nginx é um servidor web de alto desempenho que pode ser usado como proxy reverso para aplicações web.

5.2.1 Criando Conta do Google Cloud

Primeiramente, foi criada uma conta no Google Cloud, utilizando minha conta universitária do Google, após isso foi habilitado o período de testes gratuito do serviço proporcionado pelo Google. Em seguida, foi criado o projeto na plataforma, que permite dar prosseguimento com as demais ferramentas, como a maquina virtual por exemplo.

5.2.2 Criando Maquina Virtual

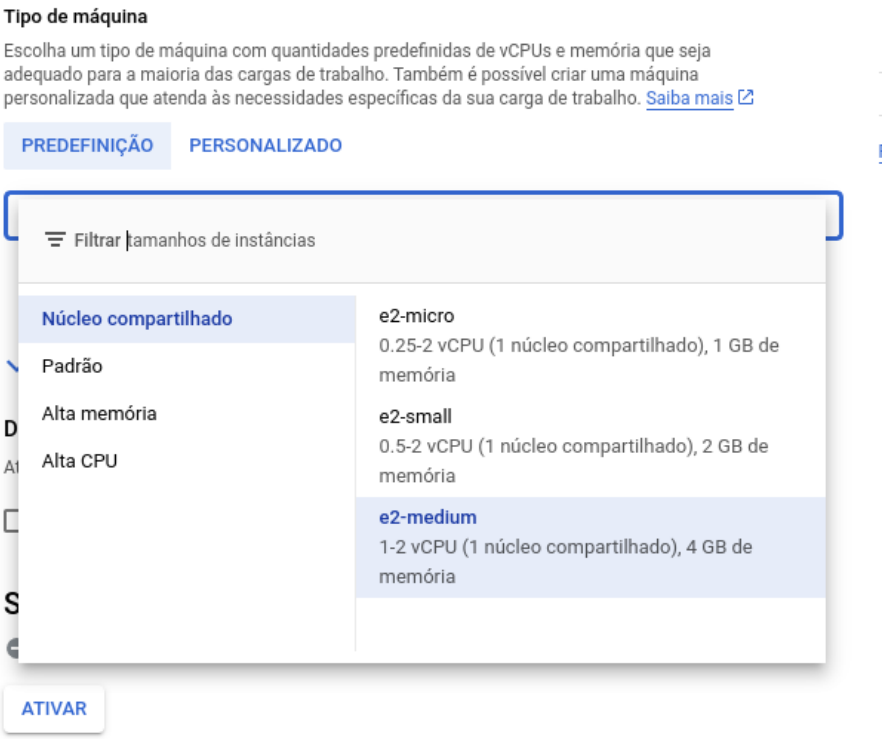
Após a fase de criação da conta e do projeto, foi utilizado a ferramenta Computer Engine, que permite o usuário a criação de maquinas virtuais que irão operar nos servidores da plataforma. Nesse sentido, foi criado uma maquina virtual com uma das opções de hardware disponibilizada previamente pela ferramenta, a e2-medium visto na figura 31, que disponibiliza 4 GB de memoria RAM junto com 1-2 vCPU para processamento. Está escolha foi atribuída visando um bom funcionamento do servidor dentro do orçamento possibilitado pelo plano gratuito. Ademais, para funcionamento da maquina virtual foi instalado o sistema operacional Ubuntu 22.04 LTS, visto na figura 32, que foi escolhido por ser uma das distros Linux mais comercializadas atualmente, trazendo uma versão estável de fácil utilização e de bom desempenho.

5.2.3 Configurando Ambiente Virtual

Para configurar o ambiente virtual na VM (Virtual Machine), procedeu-se com a atualização dos pacotes do sistema operacional e a instalação de duas tecnologias fundamentais para o funcionamento do projeto: o Python 3.10 e o PostgreSQL. Além disso, como ilustrado na figura 33, foram instaladas as dependências do projeto, bem como os pacotes Gunicorn e Nginx. O Gunicorn é um servidor web HTTP para aplicativos Python, enquanto o Nginx é um servidor web de alto desempenho que atua como proxy reverso para aplicações web.

Uma vez que o Gunicorn e o Nginx foram instalados, o arquivo de configuração do Nginx foi ajustado para direcionar as solicitações do servidor para o Gunicorn. Por fim,

Figura 31 – Opções de Hardware Disponibilizadas no Google Cloud



Fonte: Autor

Figura 32 – Sistema Operacional instalado no Servidor da API

Fonte: Autor

o código do projeto foi enviado para o servidor utilizando o Git, e o serviço foi iniciado por meio do comando do Gunicorn.

Figura 33 – Comandos executados para configuração do ambiente virtual

```
sudo apt update -y
sudo apt upgrade -y
sudo apt autoremove -y
sudo apt install build-essential -y
sudo apt install python3.10 python3.10-venv python3.10-dev -y
sudo apt install nginx -y
sudo apt install certbot python3-certbot-nginx -y
sudo apt install postgresql postgresql-contrib -y
sudo apt install libpq-dev -y
sudo apt install git
```

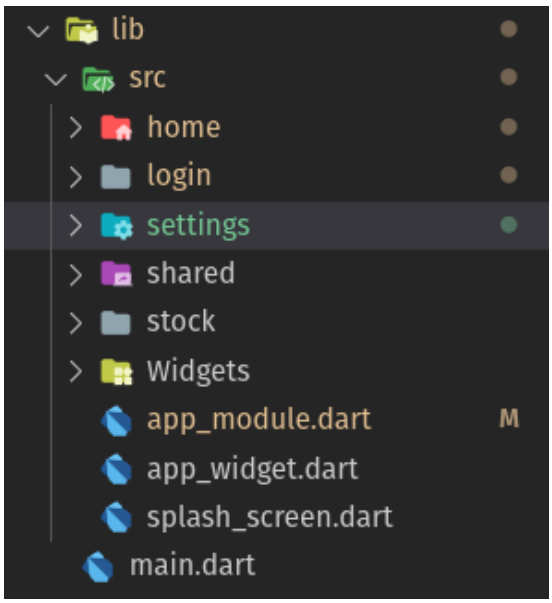
Fonte: Autor

5.3 Desenvolvimento do Frontend

Na etapa de desenvolvimento do frontend da aplicação foi definida a linguagem de programação Dart utilizada no framework Flutter, explicadas na seção 4.2.7.1. Nesse sentido como visto na figura 34 a estrutura do projeto se seguiu utilizando os padrões de desenvolvimento modular que permite a separação das funcionalidades em módulos independentes, facilitando a manutenção e evolução da aplicação. Além disso, o gerenciador de estado MobX foi utilizado para gerenciar as informações da aplicação de forma reativa, permitindo a atualização automática das telas sempre que ocorrem mudanças nos dados.

5.3.1 Organização das Pastas do Projeto

Figura 34 – Estrutura de Pastas Frontend



Fonte: Autor

As pastas exibidas na figura 34 demonstram a estrutura formada para o projeto, separando as atribuições em módulos quase que independentes. As funções das pastas estão definida abaixo:

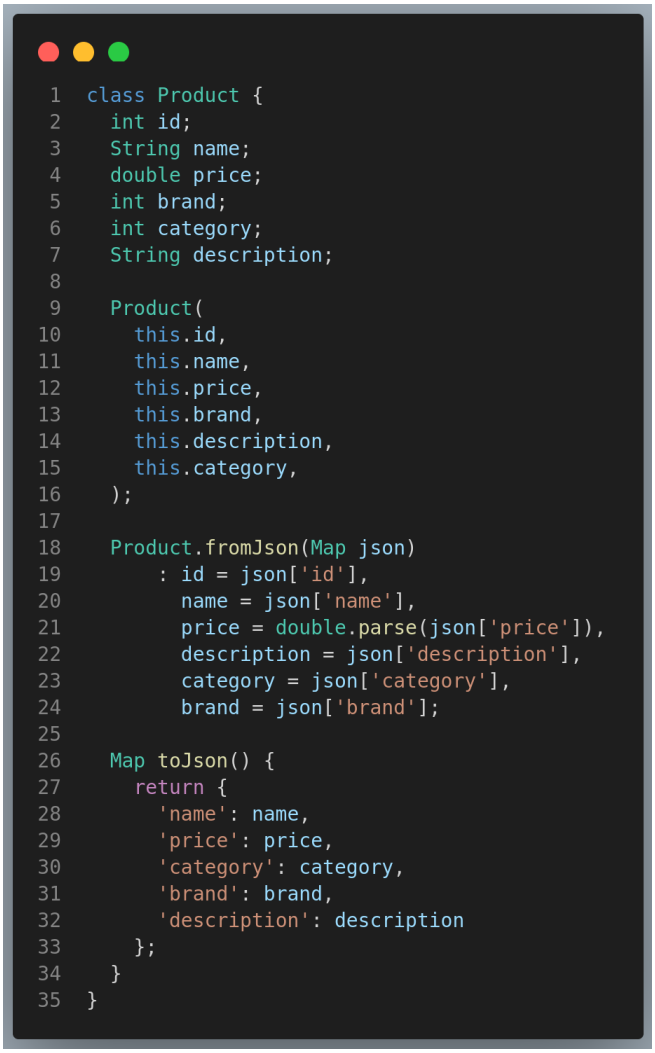
- home: o modulo 'home' que possui os arquivos que formam a tela inicial do aplicativo
- login: A pasta de login ficou responsavel por armazenar os arquivos e o redirecionamento da função de login
- settings: A pasta de 'settings' contem os arquivos da tela de configurações do aplicativo
- shared: Estão os arquivos que serão compartilhados entre os módulos, como os models, services e arquivos do tema do aplicativo.
- stock: É o modulo principal do aplicativo, contem todas as telas referentes as funções de inventario e cadastro dos produtos.
- Widgets: Na pasta 'widgets' os arquivos que formavam os componentes usados no aplicativo, como botões, caixa de textos e etc.

5.3.2 Classe Model

Na etapa do frontend, as classes models desempenharam um papel fundamental na composição da estrutura das regras de negócios estabelecidas para o aplicativo. Isso possibilitou o tratamento dos dados obtidos da API com o suporte à orientação a objetos fornecida pela linguagem Dart.

Conforme ilustrado na figura 35, a classe Product possui não apenas os mesmos atributos da classe no backend da API, mas também funções que possibilitaram converter e tratar os dados recebidos pelo backend, transformando-os em objetos e em formato JSON, sempre que necessário. Essa abordagem permitiu uma integração mais eficiente e uma manipulação mais estruturada dos dados em todo o aplicativo.

Figura 35 – Classe Product



```dart
class Product {
  int id;
  String name;
  double price;
  int brand;
  int category;
  String description;

  Product(
    this.id,
    this.name,
    this.price,
    this.brand,
    this.description,
    this.category,
  );

  Product.fromJson(Map json)
      : id = json['id'],
        name = json['name'],
        price = double.parse(json['price']),
        description = json['description'],
        category = json['category'],
        brand = json['brand'];

  Map toJson() {
    return {
      'name': name,
      'price': price,
      'category': category,
      'brand': brand,
      'description': description
    };
  }
}
```

Fonte: Autor

5.3.3 Classe Views

Diferente do Django as classes views no Flutter seguem o esquema explicado na seção 3.7, onde ficam responsáveis por definir a estrutura visual das telas do aplicativo. Dessa forma, as Views são definidas como classes que herdam da classe StatefulWidget ou StatelessWidget. As classes que herdam de StatefulWidget podem ter um estado interno que pode mudar ao longo do tempo, enquanto as classes que herdam de StatelessWidget não têm estado interno e são apenas renderizadas uma vez.

Figura 36 – Exemplo de trecho de código da tela Produto

```
Widget build(BuildContext context) {
    return Scaffold(
      floatingActionButton: FloatingActionButton.extended(
        label: const Text("Adicionar Remesssa"),
        icon: const Icon(Icons.add),
        onPressed: () {
          Modular.to.pushNamed('./addBatch', arguments: widget.product);
        },
      ),
      appBar: AppBar(title: const Text('Produto')),
      body: Padding(
        padding: const EdgeInsets.only(left: 15, right: 10, bottom: 50),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.only(top: 40.0, bottom: 15),
              child: Center(
                child: Card(
                  child: Row(
                    //crossAxisAlignment: CrossAxisAlignment.center,
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      Image.network(
                          'https://m.media-amazon.com/images/I/51iJx7YWDOL._AC_SY445_SX342_QL70_ML2_.jpg',
                          cacheWidth: 150),
                      const SizedBox(
                        width: 20,
                      ),
                      Flexible(
                        child: Column(
                          children: [
                            Text('Produto: ${widget.product.name}'),
                            Text('Id do Produto: #${widget.product.id}'),
                            Text('Preço: R\$ ${widget.product.price}0'),
                          ],
                        ),
                      )
                    ],
                  ),
                ),
              ),
            ),
```

Fonte: Autor

5.3.4 Classe Controllers

Nas classes de controller, utilizou-se a ferramenta Mobx para o gerenciamento do estado das views e o controle do fluxo das informações enviadas e recebidas pelos usuários. A figura 38 ilustra o funcionamento dessa ferramenta, onde as 'Actions' são responsáveis por alterar e gerir as informações armazenadas nos 'Observable', os quais notificam as alterações na view, gerando as 'Reactions' para os usuários.

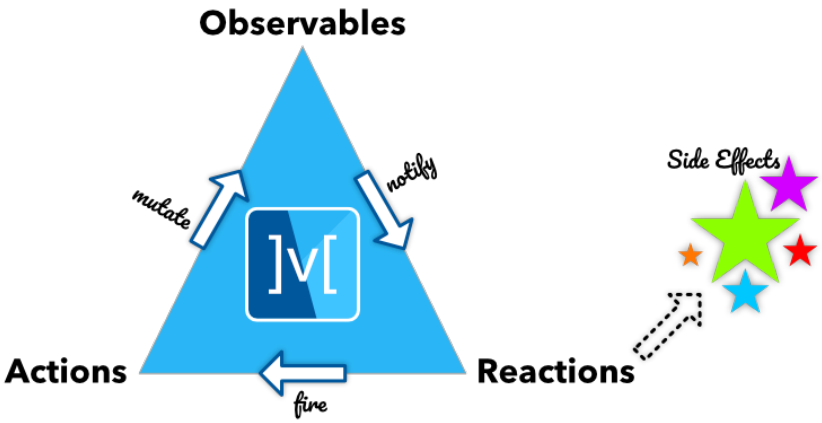
No exemplo da classe ProductStore, apresentado na figura 37, demonstra-se um trecho de código responsável pelo fluxo de informações que transitam entre a view e as classes de serviços. O nome 'Store' foi adicionado seguindo o padrão de usabilidade do MobX, que foi utilizado para criar o fluxo de gerenciamento de estados.

Figura 37 – Classe ProductStore

```dart
abstract class _ProductStore with Store {

  final service = ProductJsonServices();
  final storeBatch = Modular.get<BatchStore>();
  final stockStore = Modular.get<StockStore>();


  @observable
  Product product = Product(1, 'name', 1, 1, 'description', 1);

  @action
  Future<Product> getProductJson(Response response) async {
    product = Product.fromJson(jsonDecode(response.body));
    return product;
  }
```

Fonte: Autor

Figura 38 – Fluxograma do funcionamento do Mobx



Fonte: (PIXELINGENE, 2023)

5.3.5 Classe Services

Nas classes de serviços, são atribuídas as responsabilidades de consumo da API. Dessa forma, foram criadas funções que realizam as requisições HTTP, de acordo com as informações necessárias para cada trecho específico do aplicativo. Em conjunto, utilizam-se as classes models para converter as informações em objetos ou listas de objetos. Posteriormente, os dados são enviados para as classes de controllers, onde as informações são transmitidas ao usuário. Na figura 46, podemos analisar um trecho da classe ProductJsonServices, no qual é apresentada uma de suas principais funções: getProducts. Essa função

desempenha um papel crucial ao fazer uma requisição GET à API, com o objetivo de obter uma lista de todos os produtos cadastrados.

Figura 39 – Classe Service Product

```dart
class ProductJsonServices extends JsonServices {
  // final Future<SharedPreferences> prefs = SharedPreferences.getInstance();
  final secureStorage = SecureStorage();

  late String urlProduct;

  ProductJsonServices() {
    urlProduct = '${url}products/';
  }

  Future<List<Product>> getProducts() async {
    List<Product> products = [];
    final String? access = await secureStorage.getAccess();
    if (access != null) {
      Response response = await http.get(
        Uri.parse(urlProduct),
        headers: <String, String>{'Authorization': 'Bearer $access'},
      );
      final data = utf8.decode(response.bodyBytes);
      Iterable body = json.decode(data);
      if (response.statusCode == 401) {
        Modular.to.navigate('/home/login');
      }
      try {
        products = body.map((model) => Product.fromJson(model)).toList();
      } catch (e) {
        print(e);
      }
    }
    if (access == null) {
      Modular.to.navigate('/home/login');
    }

    return products;
  }

```

Fonte: Autor

5.4 Telas do Aplicativo

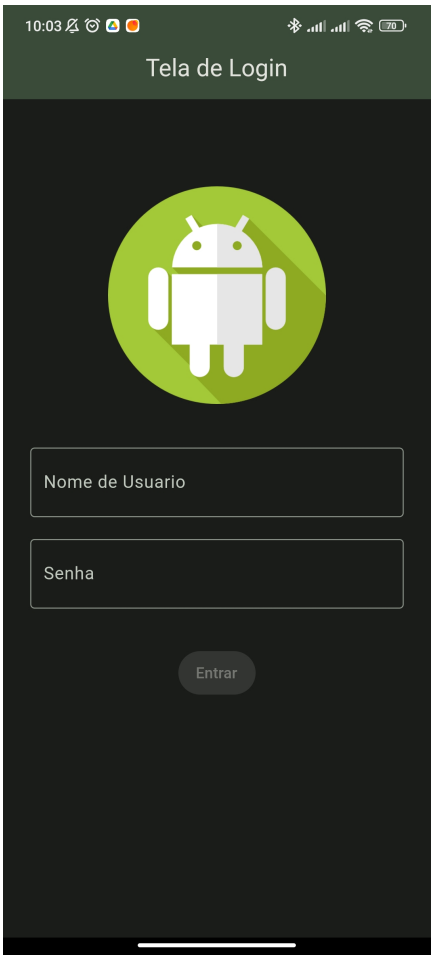
Esta seção apresenta as principais interfaces de usuário da aplicação mobile desenvolvida no projeto. Foram utilizadas as diretrizes de design do Material Design para criar telas simples, intuitivas e visualmente agradáveis, que permitem aos usuários gerenciar o estoque de forma eficiente. Cada tela é acompanhada por uma breve descrição de suas funcionalidades e características

5.4.1 Tela Login

A tela de login é a primeira tela que o usuário visualiza ao acessar o sistema. Ela é responsável por garantir a segurança do sistema, permitindo que apenas usuários

autorizados acessem as informações do estoque e finanças. Para isso, a tela de login deve solicitar credenciais de autenticação, como nome de usuário e senha, e verificar se as informações fornecidas são válidas antes de permitir o acesso ao sistema.

Figura 40 – Tela de Login

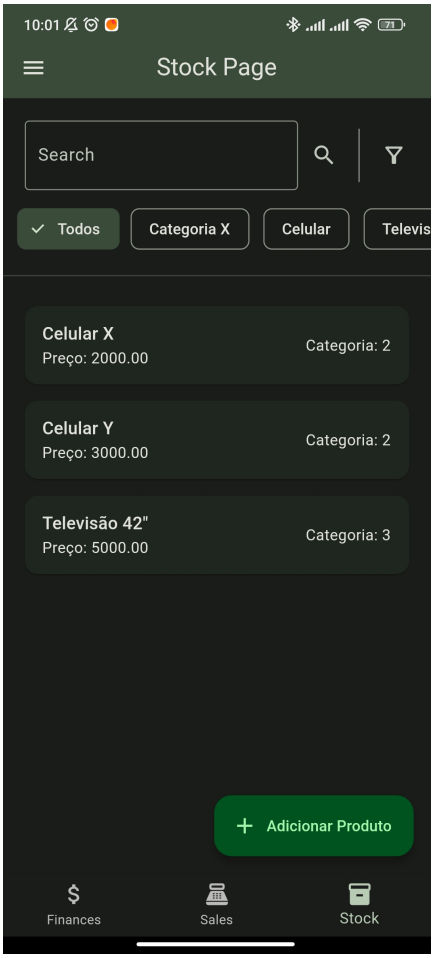


Fonte: Autor

5.4.2 Tela de Estoque

A tela Estoque é responsável por exibir os últimos produtos cadastrados, permitir a pesquisa de produtos já cadastrados, permitir o acesso aos detalhes de cada produto e direcionar para a tela de adicionar produto. É uma tela importante para gerenciamento do estoque, pois permite que o usuário visualize rapidamente os produtos disponíveis e possa acessá-los para realizar qualquer alteração necessária. Como visto na figura 41, permite que o usuário adicione novos produtos ao estoque por meio do botão que redireciona a tela de adicionar produto.

Figura 41 – Tela Inicial com o estoque dos Produtos



Fonte: Autor

5.4.3 Tela Adicionar Produto

A tela Adicionar Produto tem como responsabilidade permitir que o usuário cadastre um novo produto no estoque. Para isso, como exibido na figura 42 ela deve permitir que o usuário preencha as informações do produto, como nome, código, categoria e marca. Além disso, ela deve fornecer um botão para enviar o produto cadastrado para a tela de adicionar remessa, para que seja possível cadastrar uma nova remessa do produto. A tela Adicionar Produto é fundamental para o gerenciamento do estoque, pois permite que novos produtos sejam adicionados e cadastrados de forma organizada e eficiente.

Figura 42 – Tela Adicionar Produto

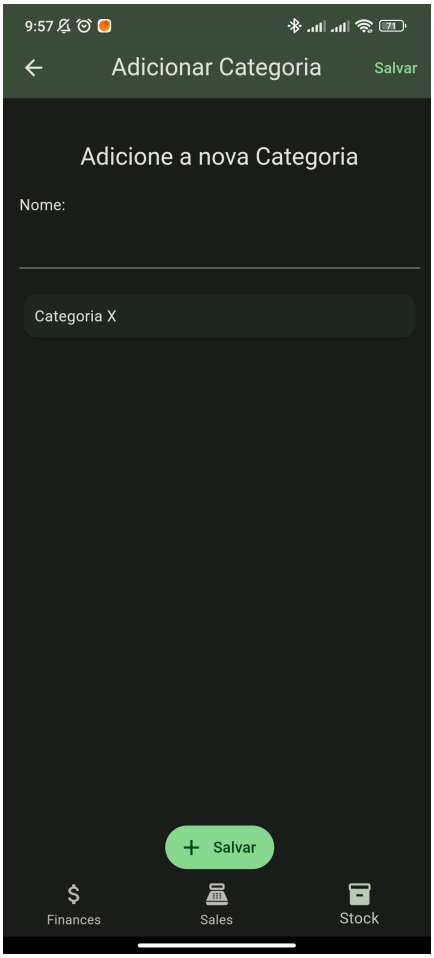
Ad

Fonte: Autor

5.4.4 Tela Adicionar Categoria

A tela Adicionar Categoria é responsável por permitir que o usuário adicione uma nova categoria ao sistema, para que possa ser utilizada na classificação dos produtos. Ela deve permitir que o usuário preencha as informações da categoria, como nome e descrição, e fornecer um botão para enviar a categoria cadastrada para o sistema. A tela Adicionar Categoria é importante para o gerenciamento do estoque, pois permite que novas categorias sejam adicionadas de forma organizada e eficiente, facilitando a classificação e organização dos produtos no sistema.

Figura 43 – Tela Inicial com o estoque dos Produtos



Fonte: Autor

5.4.5 Tela Adicionar Remessa

A tela Adicionar Remessa é responsável por permitir que o usuário cadastre uma nova remessa do produto já existente no estoque. Ela deve exibir as informações do produto e permitir que o usuário preencha as informações da remessa, como quantidade, data de entrada, data de validade, etc. Além disso, ela deve fornecer um botão para enviar a remessa cadastrada para a tela de adicionar lançamento, para que seja possível cadastrar um novo lançamento financeiro referente a essa remessa. A tela Adicionar Remessa é fundamental para o gerenciamento do estoque, pois permite que novas remessas sejam adicionadas e cadastradas de forma organizada e eficiente.

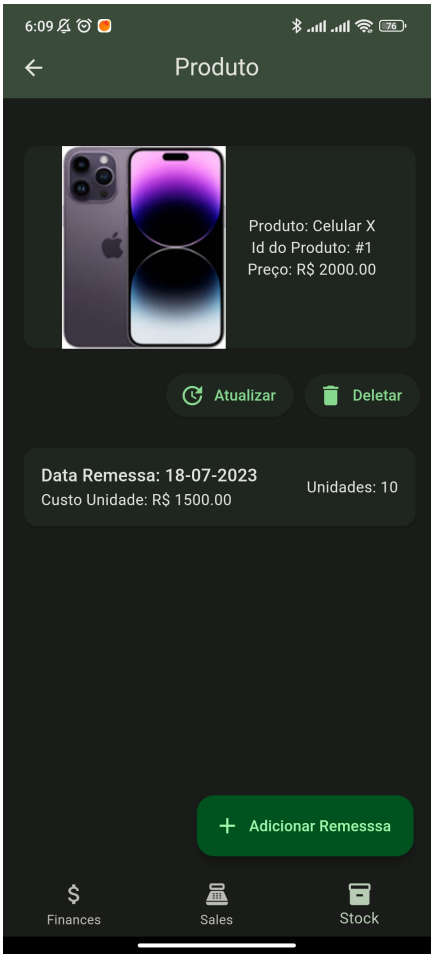
Figura 44 – Tela Inicial com o estoque dos Produtos

Fonte: Autor

5.4.6 Tela Visualizar Produto

A tela de visualizar produto é responsável por exibir as informações detalhadas de um determinado produto do estoque. Ela deve receber as informações do produto selecionado na tela Stock e exibi-las de forma clara e organizada, incluindo nome, código, categoria, marca, quantidade em estoque e outras informações relevantes. Além disso, a tela deve permitir que o usuário acesse informações sobre as remessas do produto, como a quantidade recebida, a data de entrada e a data de validade, e permitir que o usuário adicione uma nova remessa do produto por meio da tela de adicionar remessa. A tela de visualizar produto é uma parte importante do gerenciamento do estoque, pois permite que o usuário visualize as informações detalhadas de um produto específico e faça alterações ou adições conforme necessário.

Figura 45 – Tela Visualizar Produto

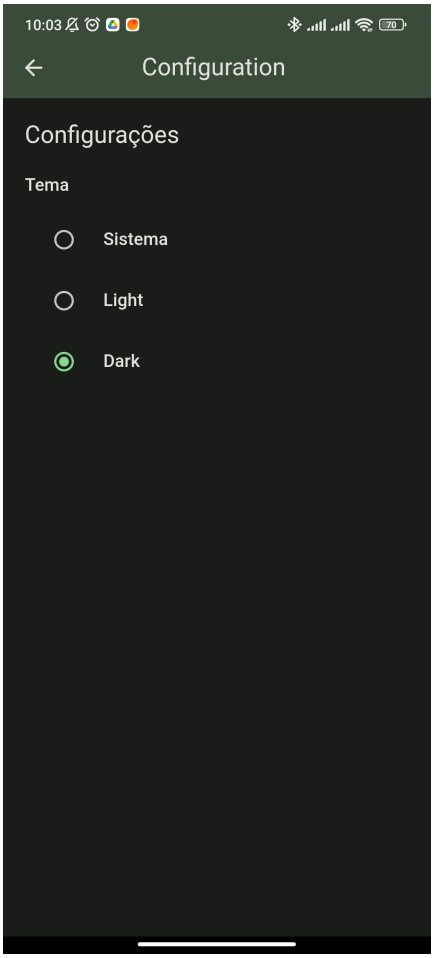


Fonte: Autor

5.4.7 Tela de Configurações

A tela de configurações é responsável por permitir que o usuário personalize o aplicativo de gerenciamento de estoque e finanças de acordo com suas preferências. Entre as opções disponíveis, está a opção de mudar o tema do aplicativo. Essa personalização pode tornar o aplicativo mais agradável e fácil de usar para o usuário, além de permitir que ele se adapte melhor às suas necessidades e preferências individuais. A tela de configurações é uma parte importante do aplicativo, pois permite que o usuário personalize a experiência de uso e torne o aplicativo mais adequado às suas necessidades pessoais.

Figura 46 – Tela de Configurações



Fonte: Autor

6 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Após o desenvolvimento da aplicação mobile e da API, foi realizado um teste com um grupo de usuários para avaliar a usabilidade e eficiência do sistema. O teste consistiu em uma pesquisa de satisfação e na realização de tarefas específicas dentro da aplicação.

6.1 Testes com usuários

Os resultados da pesquisa de satisfação foram bastante positivos, com a maioria dos usuários avaliando a aplicação como fácil de usar e intuitiva. Além disso, a maioria dos usuários afirmaram que a aplicação atendeu às suas expectativas e que eles recomendariam a aplicação para outras pessoas.

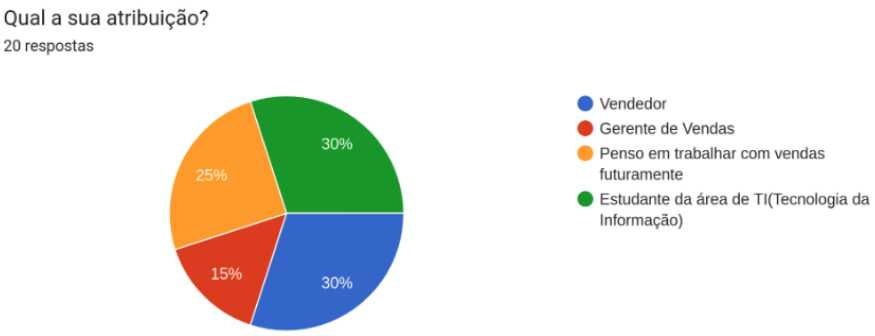
Já na realização das tarefas, os usuários conseguiram concluir as atividades de gestão de estoque de forma rápida e eficiente, sem encontrar muitos problemas ou dificuldades. Isso demonstra que a aplicação é capaz de atender às necessidades dos usuários de forma eficiente e intuitiva.

Em resumo, os resultados do teste com usuários foram bastante positivos, demonstrando que a aplicação mobile e a API desenvolvidas são capazes de atender às necessidades dos usuários de forma eficiente e intuitiva. A pesquisa de satisfação e a realização das tarefas demonstraram que os usuários estão satisfeitos com a aplicação e que ela é capaz de cumprir os seus objetivos de gerenciamento de estoque de forma eficiente.

6.2 Testes realizados e resultados obtidos

Na distribuição final, foram realizados testes de satisfação com um grupo seleto de usuários. Esse grupo foi composto por 9 pessoas que trabalhavam com vendas, 5 pessoas que tinham interesse em trabalhar com vendas no futuro e 6 estudantes da área de TI. O objetivo do primeiro segmento, relacionado às vendas, era avaliar a satisfação do público-alvo do aplicativo, com foco nos pequenos empreendedores que poderiam se beneficiar da proposta do projeto. Já o segundo grupo, formado pelos estudantes de tecnologia, foi escolhido com o propósito de aproveitar a visão técnica e o conhecimento dos participantes em relação ao desenvolvimento de softwares.

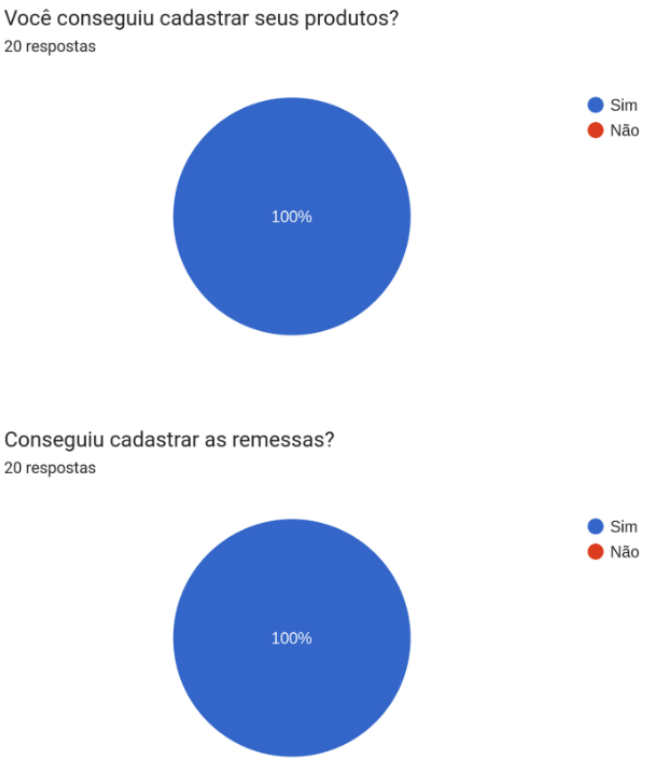
Figura 47 – Participantes da Pesquisa



Fonte: Autor

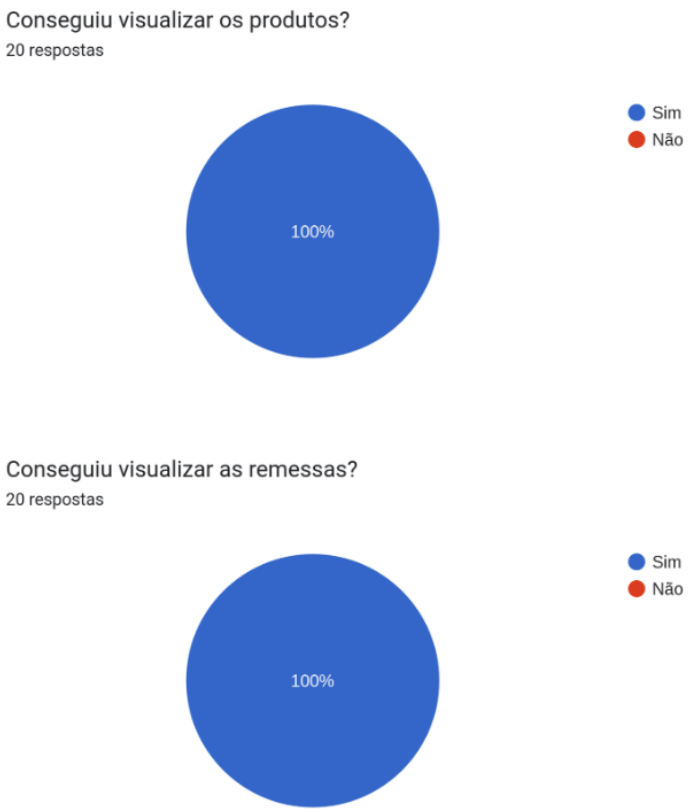
As primeiras 5 perguntas foram relacionadas as funcionalidades do sistema, para obter a informação de que os usuários conseguiram entender o funcionamento do mesmo. Como visto na imagem 48, houveram 100% de sucesso entre todos os usuários em conseguir realizar as funções principais do aplicativo, como o cadastro de produtos, remessas e categorias. Além disso, como visto na figura 49 também foi obtido 100% de resultados ao visualizar o conteúdo cadastrado no app.

Figura 48 – Participantes da Pesquisa



Fonte: Autor

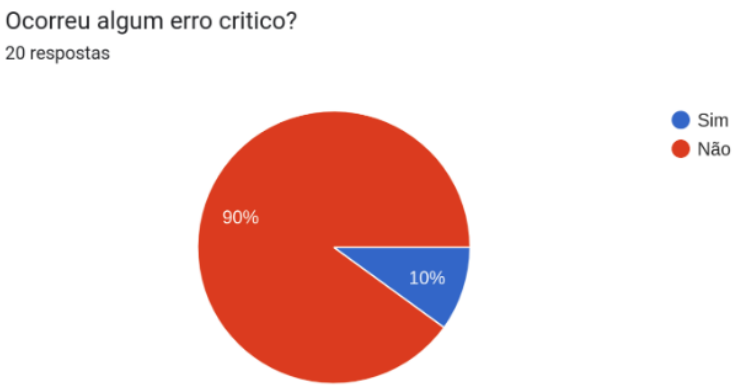
Figura 49 – Participantes da Pesquisa



Fonte: Autor

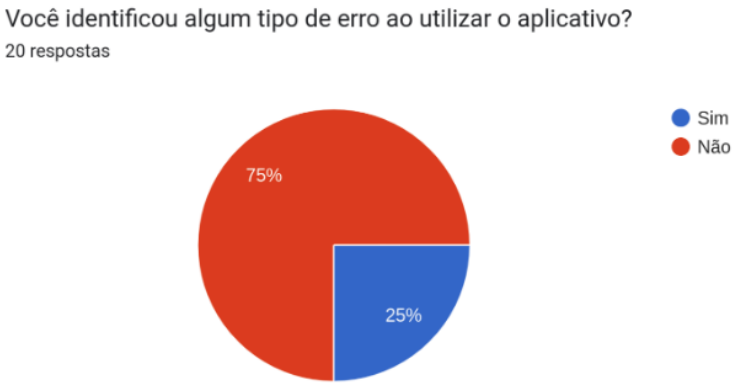
Já na categoria de erros foi verificado que 10% dos usuários verificaram erros criticos ao utilizar a aplicação e 20% verificaram erros menores, como visto nas figuras 50 e 51. Ao entrar em contato com os usuários foram relatados que os erros criticos envolviam a falta de internet no telemóvel ao tentar fazer a autenticação, evento ocorrido devido a implementação do requisito do aplicativo funcionar online. Já com os usuários que relataram os erros menores envolviam questões de desempenho do aplicativo em serviços de internet mais lentos e telemóveis mais antigos, que aconteceram devido a atrasos do servidor e adequação do Flutter a versões mais recentes do Android. Chegando a conclusão que melhorias podem ser feitas em relação a velocidade de resposta entre a API e o aplicativo, além de testes e otimizações em smartphones mais antigos visando o melhor desempenho.

Figura 50 – Participantes da Pesquisa



Fonte: Autor

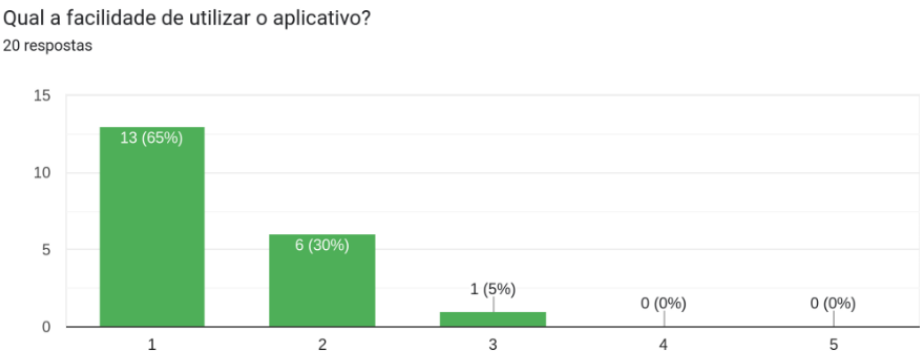
Figura 51 – Participantes da Pesquisa



Fonte: Autor

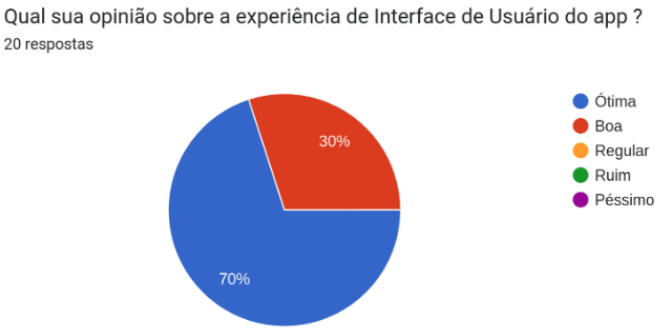
Como ultimo seguimento de perguntas foi focado na usabilidade do aplicativo, se o mesmo atendia a demanda de ser fácil e intuitivo de utilizar. Chegando ao resultado de 95% de satisfação entre os usuários que marcaram as opções 1 e 2 que se aproximavam do atributo 'Muito Fácil' nas perguntas, visto na figura 52. Além disso na figura 53, podemos ver que a experiencia de interface agradou a todos os usuários que votaram entre Boa e Ótima como resposta.

Figura 52 – Facilidade de utilizar o aplicativo



Fonte: Autor

Figura 53 – Experiencia de Usuario



Fonte: Autor

7 CONSIDERAÇÕES FINAIS

O desenvolvimento do projeto permitiu a aplicação prática de diversas tecnologias e metodologias aprendidas ao longo do curso, incluindo o desenvolvimento de APIs RESTful com Django Rest Framework, a criação de aplicativos mobile com Flutter e a utilização de metodologias ágeis de desenvolvimento de software.

A utilização dessas tecnologias e metodologias permitiu o desenvolvimento de uma aplicação mobile eficiente e intuitiva, capaz de gerenciar o estoque de forma simples e prática. Os resultados do teste com usuários demonstraram que a aplicação atendeu às necessidades dos usuários de forma eficiente e intuitiva, com uma alta taxa de satisfação.

O desenvolvimento do projeto também permitiu a aplicação prática dos conceitos de arquitetura de software, incluindo a utilização da arquitetura modular no frontend e a organização em classes na API. Esses conceitos permitiram a criação de um sistema escalável e de fácil manutenção, capaz de evoluir com o tempo e atender às necessidades dos usuários.

Em resumo, o desenvolvimento do projeto permitiu a aplicação prática de diversas tecnologias e metodologias aprendidas ao longo do curso, resultando em uma aplicação mobile eficiente e intuitiva para gerenciamento de estoque. Os resultados do teste com usuários foram positivos, demonstrando a efetividade da aplicação em atender às necessidades dos usuários. O projeto também permitiu a aplicação prática de conceitos de arquitetura de software, tornando o sistema escalável e de fácil manutenção.

7.1 Melhorias Futuras

Uma das maiores expectativas para a conclusão do projeto é que ele atendesse a demanda de ser escalável e de fácil manutenção. Dessa forma, diversas melhorias futuras podem ser implementadas no desenvolvimento tanto da API quanto do aplicativo mobile, melhorias como:

- Implementação de um modulo de Finanças, registrar as remessas dos produtos junto com gastos diversos dos vendedores
- Implementação de um modulo de Vendas, registrar as saídas dos produtos e integrar com o modulo de Finanças para o controle de ganhos e saídas
- Implementação do modulo de Usuários, incluir diferentes tipos de usuários para controle de hierarquia

No geral, há muitas possibilidades para melhorar a aplicação e torná-la ainda mais útil e eficiente para os usuários, e essas melhorias podem ser implementadas de forma

gradual ao longo do tempo para garantir que a aplicação continue a evoluir e atender às necessidades em constante mudança dos usuários.

REFERÊNCIAS

ANNA, V. G. S. **Controle Universitário: Um Aplicativo de controle dos estudos e do andamento do graduando ao longo do curso superior**. UFOP, 2021. Disponível em: <http://www.monografias.ufop.br/handle/35400000/3464>.

CORAZZA, P. V. **Um aplicativo multiplataforma desenvolvido com flutter e NoSQL para o cálculo da probabilidade de apendicite**. UFRGS, 2018. Disponível em: <http://hdl.handle.net/10183/190147>.

COSTA, D. D. P. **Proposta de app para divulgação do Curso de Gestão da Informação da Universidade Federal do Paraná**. [S.l.]: UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ, 2015.

CREATELY. Tutorial do diagrama de caso de uso (guia com exemplos). 2021. Disponível em: <https://creately.com/blog/pt/diagrama/tutorial-de-diagrama-de-caso-de-uso/>.

DEVELOPERS, G. **Crie aplicativos para qualquer tela**. Flutter.dev, 2022. Disponível em: <https://flutter.dev/>. Acesso em: 06 Set. 2022.

FIELDING, R. T. Architectural styles and the design of network-based software architectures. 2000. Disponível em: <https://www.ics.uci.edu/˜fielding/pubs/dissertation/top.htm>.

FILHO, A. S. **Adicionando um controlador a um aplicativo ASP.NET Core MVC com o Visual Studio 2017**. COREMVC, 2017. Disponível em: <https://coremvc.com.br/adicionando-um-controlador-um-aplicativo-asp-net-core-mvc-com-o-visual-studio-2017/>. Acesso em: 06 Set. 2022.

GOMES, F. M. do C. **Desenvolvimento de uma aplicação para realização de inventários utilizando dispositivos móveis**. [S.l.]: UNIVERSIDADE FEDERAL DO CEARÁ, 2021.

HOSTGATOR. **Framework: o que é, quais utilizar e como eles funcionam!** hostgator.com.b, 2020. Disponível em: <https://www.hostgator.com.br/blog/frameworks-na-programacao/>. Acesso em: 18 Abril. 2023.

MARTIN, R. C. **Clean Code: A Handbook of Agile Software Craftsmanship**. [S.l.]: Prentice Hall, 2008.

NOTION. **Uma história de ferramentas e do futuro do trabalho!** www.notion.so, 2019. Disponível em: <https://www.notion.so/pt-br/about>. Acesso em: 01 Junho. 2023.

OKTA. **JSON Web Tokens**. jwt.io, 2023. Disponível em: <https://jwt.io/>. Acesso em: 18 Abril. 2023.

PIXELINGENE. **Documentation Mobx.dart**. dart.pixelingene.com, 2023. Disponível em: <https://pub.dev/documentation/mobx/latest/>. Acesso em: 17 Abril. 2023.

PORDEUS, I. C. **Desenvolvimento de um aplicativo para adoção de pets utilizando flutter e firebase**. [S.l.]: CENTRO UNIVERSITÁRIO CHRISTUS SISTEMAS DE INFORMAÇÃO, 2021.

REDHAT. **O que é API?** redhat.com, 2023. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em: 10 Abril. 2023.

SEBRAE. **Dia da Micro e Pequena Empresa evidencia a importância dos empreendedores para o Brasil**. agenciasebrae.com.br, 2022. Disponível em: <https://agenciasebrae.com.br/brasil-empreendedor/dia-da-micro-e-pequena-empresa-evidencia-a-importancia-dos-empreendedores-para-o-brasil/>. Acesso em: 30 Set. 2022.

WORLDWIDE. **PYPL PopularitY of Programming Language**. PYPL, 2023. Disponível em: <https://pypl.github.io/PYPL.html>. Acesso em: 16 de Abril. 2023.

Apêndices

APÊNDICE A – FEEDBACK DE SATISFAÇÃO DO USO DO APLICATIVO

Formulário de Satisfação e Feedback sobre APLICAÇÃO MOBILE PARA GESTÃO DE INVENTÁRIO COMERCIAL

A.1 Qual a sua atribuição?

- Vendedor
- Gerente de Vendas
- Penso em trabalhar com vendas futuramente
- Estudante da área de TI(Tecnologia da Informação)

A.2 Você conseguiu cadastrar seus produtos?

- Sim
- Não

A.3 Conseguiu cadastrar as remessas?

- Sim
- Não

A.4 Conseguiu visualizar os produtos?

- Sim
- Não

A.5 Conseguiu visualizar as remessas?

- Sim
- Não

A.6 Conseguiu utilizar a ferramenta de busca para pesquisar os produtos cadastrados?

- Sim
- Não

A.7 Ocorreu algum erro critico?

- Sim
- Não

A.8 Você identificou algum tipo de erro ao utilizar o aplicativo?

- Sim
- Não

A.9 Qual a facilidade de utilizar essa funcionalidade?

- 1 - Muito Fácil
- 2
- 3
- 4
- 5 - Extremamente Difícil

A.10 Você utilizaria a aplicação no seu negocio?

- Sim
- Não

A.11 Você recomenda o uso do aplicativo?

- Sim
- Não
- Talvez

A.12 Qual sua opinião sobre a experiência de Interface de Usuário do app ?

- Ótima
- Boa
- Regular

- Ruim
- Péssima