INSTITUTO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DA PARAÍBA
DIRETORIA DE DESENVOLVIMENTO DE ENSINO
COORDENAÇÃO DO CURSO SUPERIOR DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

GUILHERME ESDRAS SILVA E SOUZA
JOSÉ HENRIQUE AZEVEDO DE BRITO

FACILIFARMA – UM SISTEMA DE CONTROLE DE VENDAS DE MEDICAMENTOS

CAMPINA GRANDE - PB
2023

GUILHERME ESDRAS SILVA E SOUZA
JOSÉ HENRIQUE AZEVEDO DE BRITO

FACILIFARMA – UM SISTEMA DE CONTROLE DE VENDAS DE MEDICAMENTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação, do Instituto Federal da Paraíba – *Campus* Campina Grande, em cumprimento às exigências parciais para a obtenção do título bacharel em Engenharia de Computação.

ORIENTADOR(A):
HENRIQUE DO NASCIMENTO CUNHA

CAMPINA GRANDE - PB
2023

S729f Souza, Guilherme Esdras Silva e.
Facilifarma : um sistema de controle de vendas e medicamentos / Guilherme Esdras Silva e Souza, José Henrique Azevedo de Brito. - Campina Grande, 2023.
31 f.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia de computação) - Instituto Federal da Paraíba, 2023.
Orientador: Prof. Me. Henrique do Nascimento Cunha.

1. Engenharia de computação - desenvolvimento de software 2. Desenvolvimento de sistemas - controle de medicamentos 3. Medicamento controlado - venda e controle I. Brito, José Henrique Azevedo de. II.Cunha, Henrique do Nascimento. II Título.

CDU 004.438

GUILHERME ESDRAS SILVA E SOUZA
JOSÉ HENRIQUE AZEVEDO DE BRITO

FACILIFARMA – UM SISTEMA DE CONTROLE DE VENDAS DE MEDICAMENTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação, do Instituto Federal da Paraíba – *Campus* Campina Grande, em cumprimento às exigências parciais para a obtenção do título bacharel em Engenharia de Computação.

Aprovada em ____ / ____ / _____

Banca Examinadora

_____
Prof(a). Henrique do Nascimento Cunha, DSc. - IFPB
Orientador (IFPB)

_____
Prof(a). Cesar Rocha Vasconcelos, DSc - IFPB
Examinador

_____
Prof(a). Mirna Carelli Oliveira Maia, DSc. - IFPB
Examinador

Abstract. *This project aims to make it easier for patients to buy controlled medication by connecting the medical/patient/pharmacy triangle in a practical way. The goal is to provide customers with a convenient and reliable app to purchase medication from pharmacies located nearby, and to ensure that pharmacies have a complete and intuitive platform to manage their sales through the app and connect with their customers. The app will allow users to search for pharmacies in their area, view products, and place orders. For pharmacies, the dashboard will enable the visualization of statistics and generation of reports in various formats, containing charts and tables for total control of the number of sales, customers reached, products sold, and various other useful data for understanding their reach. For doctors, there will be a prescription signing app so that patients can purchase their medications at pharmacies with their prescription.*
Keywords: *Controlled medication, purchasing app, pharmacies, dashboard, prescriptions, and patients.*

Resumo. *Este projeto visa facilitar a vida dos pacientes na hora de comprar medicamentos controlados, conectando o triângulo médico/paciente/farmácia de forma prática. O objetivo é fornecer aos clientes um aplicativo conveniente e confiável para compra de medicamentos em farmácias localizadas próximas a eles, e garantir que as farmácias tenham uma plataforma completa e intuitiva para administrar suas vendas pelo aplicativo e contato com os seus clientes. O aplicativo permitirá aos usuários pesquisar farmácias em sua área, visualizar produtos, fazer pedidos. Para as farmácias, a dashboard possibilitará a visualização de estatísticas e geração de relatórios em diversos formatos, contendo gráficos e tabelas para total controle do número de vendas, clientes alcançados, produtos vendidos e diversos outros dados úteis para noção de alcance. Para os médicos, um app de assinatura de prescrições, para que pacientes consigam comprar seus medicamentos nas farmácias com a receita.*
Palavras chaves: *medicamentos controlados, aplicativo de compra, farmácias, dashboard, prescrições e pacientes.*

1. Introdução

A pandemia de COVID-19, iniciada em 2020, foi um marco histórico que impôs novas formas de viver, trabalhar e consumir. Ela trouxe diversos desafios para a sociedade em todo o mundo, incluindo a necessidade de adaptação das atividades cotidianas para garantir a segurança e saúde da população. As medidas de isolamento social e a necessidade de evitar aglomerações mudaram completamente a dinâmica do mercado, principalmente no setor farmacêutico. Nesse contexto, a aquisição de medicamentos e produtos farmacêuticos tornou-se uma preocupação ainda maior, visto que as pessoas precisavam evitar sair de casa e se expor ao vírus.

Essa situação exigiu o desenvolvimento de soluções alternativas para a compra e venda de produtos farmacêuticos, que permitissem aos clientes adquirir os medicamentos de forma conveniente e segura, sem precisar sair de casa. Em resposta a essa necessidade, foi criada um sistema completo de compra e venda de medicações e produtos farmacêuticos, que visa oferecer praticidade e segurança aos clientes, além de possibilitar um melhor relacionamento entre empresas e clientes.

Por meio dessa plataforma, é possível realizar compras de medicamentos e produtos farmacêuticos diretamente pelo *smartphone*, de maneira prática e conveniente, sem precisar se deslocar até a farmácia física. Além disso, as empresas do setor farmacêutico podem aumentar sua

visibilidade e o retorno financeiro mesmo em épocas de crise e pandemia, atendendo às necessidades dos clientes de forma eficiente e adaptando-se ao novo contexto imposto pela pandemia.

Dessa forma, a plataforma representa uma solução alternativa para o setor farmacêutico, permitindo que as empresas ampliem seu alcance e a qualidade do serviço prestado aos clientes, ao mesmo tempo em que oferece praticidade e segurança para aquisição de medicamentos em um momento de grande desafio para a sociedade. A presente pesquisa tem como objetivo avaliar a eficácia dessa plataforma, verificando seus impactos no setor farmacêutico.

1.1. Objetivos

1.1.1. Objetivo geral

- Desenvolver um *marketplace* de medicamentos com prescrições

1.1.2. Objetivo específicos

- Permitir a compra de remédios que não necessitem de prescrições.
- Fornecer uma aplicação de *dashboard* para as farmácias terem controle sobre as vendas com estatísticas.
- Fornecer um *app* para medicos assinarem prescrições para pacientes possam comprar remédios.

1.2. Relevância

Com a adesão desse produto, os usuários finais poderão comprar remédios sem precisar sair de casa. As farmácias poderão ganhar mais notoriedade, e com isso mais vendas; poderão gerar relatórios e aumentar sua receita através de *marketing* em cima da nossa aplicação, com promoções e descontos visando atrair mais clientes.

Além disso, as farmácias terão um maior contato com seus clientes, tendo a possibilidade de verificar quais dos seus produtos são mais relevantes para o público do que outros.

2. Desenvolvimento

2.1. Sistemas presentes no mercado

Atualmente no mercado já existem algumas soluções alternativas para venda e compra de medicamentos online. Soluções estas capazes de satisfazer de forma simples e minimalista as necessidades de venda à distância para as farmácias e as de compra independentemente da localidade para os clientes. Dentre estas soluções podemos citar algumas como, QueroDelivery®, Memed®, Pague Menos®, Drogasil®, Droga Raia®, Ultrafarma®e Drogarias Pacheco®. A maioria, mas não todas, com aplicação móvel disponível, e poucas com uma *dashboard* de controle.

Porém, a solução proposta aqui, tem como finalidade, além de suprir as necessidades já preenchidas pelas soluções existentes, proporcionar um maior controle e relatório das vendas para as farmácias através de um painel de controle (*dashboard*) completo e com diversas funcionalidades úteis, e a possibilidade de maior contato com diversas farmácias através do app para clientes.

Abaixo, duas tabelas comparando a solução proposta com os sistemas já existentes, exibindo quais funcionalidades e propriedades cada um possuí. Um “X” indica que a funcionalidade em questão está presente naquele sistema e um “-” indica ausência da funcionalidade.

Tabela 1. Comparativo de funcionalidades entre sistemas farmacêuticos analisados

	Funções				
Nome	QueroDelivery	Memed	PagueMenos	Drogasil	Facilifarma
Venda de medicamentos	X	-	X	X	X
Venda de medicamento sob prescrição	-	-	X	X	X
Prescrição de profissional de saúde	-	X	-	-	X
App	X	-	X	X	X
Site	X	X	X	X	X
Controle para farmácias	-	-	-	-	X
Marketplace	-	-	-	-	X

Tabela 2. Comparativo de funcionalidades entre sistemas farmacêuticos analisados

	Funções			
Nome	Droga Raia	Ultrafarma	Drogaria Pacheco	Facilifarma
Venda de medicamentos	X	X	X	X
Venda de medicamento sob prescrição	X	-	X	X
Prescrição de profissional de saúde	-	-	-	X
App	-	X	X	X
Site	X	X	X	X
Controle para farmácias	-	-	-	X
Marketplace	-	-	-	X

2.2. *Backend*

2.2.1. Arquitetura a nível de aplicação

Nos últimos anos, houve uma ascensão da arquitetura de microserviços, um padrão arquitetural que descreve uma aplicação como um conjunto de pequenos serviços responsáveis por uma determinada função, que se comunicam por meio por meio de alguma API, geralmente uma API REST.

O desenvolvimento de aplicações com esse estilo de arquitetura, nos dá liberdade de escolher a melhor linguagem, banco de dados, API, frameworks de acordo com a funcionalidade do serviço, além de ser fácil de ser escalada, gerenciada, testada e implantada [Lopes 2023]. Também temos uma facilidade na divisão de equipes de desenvolvimento para cada pequena parte da aplicação.

O *backend* foi divido em 11 microserviços:

- ***Authentication Backend***: Gerenciamento de *login*/cadastro e operações relacionadas;
- ***Discount Coupon Backend***: CRUD de cupons de desconto;
- ***Doctor Backend***: CRUD para guardar informações de doutores;

- ***Gateway Certilion Backend***: *Backend* com integração a API de assinatura digital Certiltion. Essa API é utlizada para assinar e validar as prescrições feitas por médicos;
- ***Eureka Backend***: *Service Registry* responsável por gerir as conexões de todos os outros microserviços e suas réplicas.
- ***Gateway Backend***: Microserviço que recebe todas as requisições vindas do *Frontend*, e distribui essas requisições para seus determinados microserviços, que estão conectados ao *Eureka Backend*;
- ***Patient Backend***: CRUD para guardar informações relacionadas aos pacientes;
- ***Pharmacy Backend***: CRUD para guardar informações relacionadas as farmácias;
- ***Prescription Backend***: CRUD parar guardar prescrições assinadas pelos doutores;
- ***Purchase History Backend***: Guarda histórico das compras, vendas de remédios;
- ***Remedy Backend***: CRUD parar guardar remédios das farmácias;

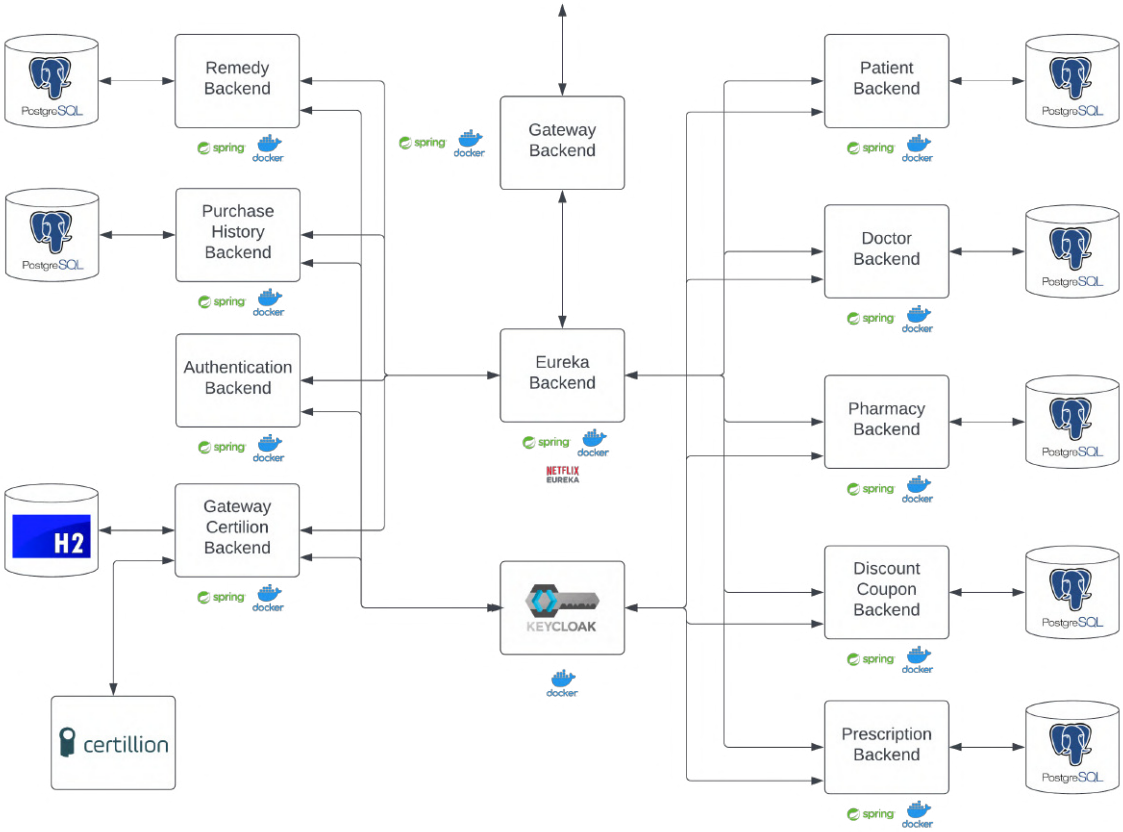

Figura 1. Arquitetura e comunição do *backend*

Alguns microserviços como o *Gateway Certilion Backend* e *Prescription Backend* se comunicam entre si, mas não foi possível ilustrar isso na figura.

2.2.2. Arquitetura a nível de projeto

O método de padronização de arquitetura utilizado no *backend*, é chamado de *Clean Architecture* e foi proposto por [Martin 2017], também conhecido como Uncle Bob. É uma arquitetura em camadas com a proposta de prover sistemas coesos, testáveis, que funcionam independentemente de qualquer tecnologia, e que favoreçam a reusabilidade de código.

Segundo Robert C. Martin, o objetivo desta arquitetura é a individualização de preocupações, organizando o software em camadas, onde regras de negócios e interfaces são separadas e suas responsabilidades divididas.

Cada camada deste modelo representa uma parte diferente do software. As camadas externas representam mecanismos, como *Frameworks Web* e Bancos de Dados. Regras e políticas de negócios estão nas camadas internas. A principal e mais fundamental regra da *Clean Architecture* é a regra de dependência, que estabelece que "...o nome de algo declarado em um círculo externo não deve ser mencionado pelo código em um círculo interno. Isso inclui funções, classes, variáveis ou qualquer outro nome ou entidade do software"

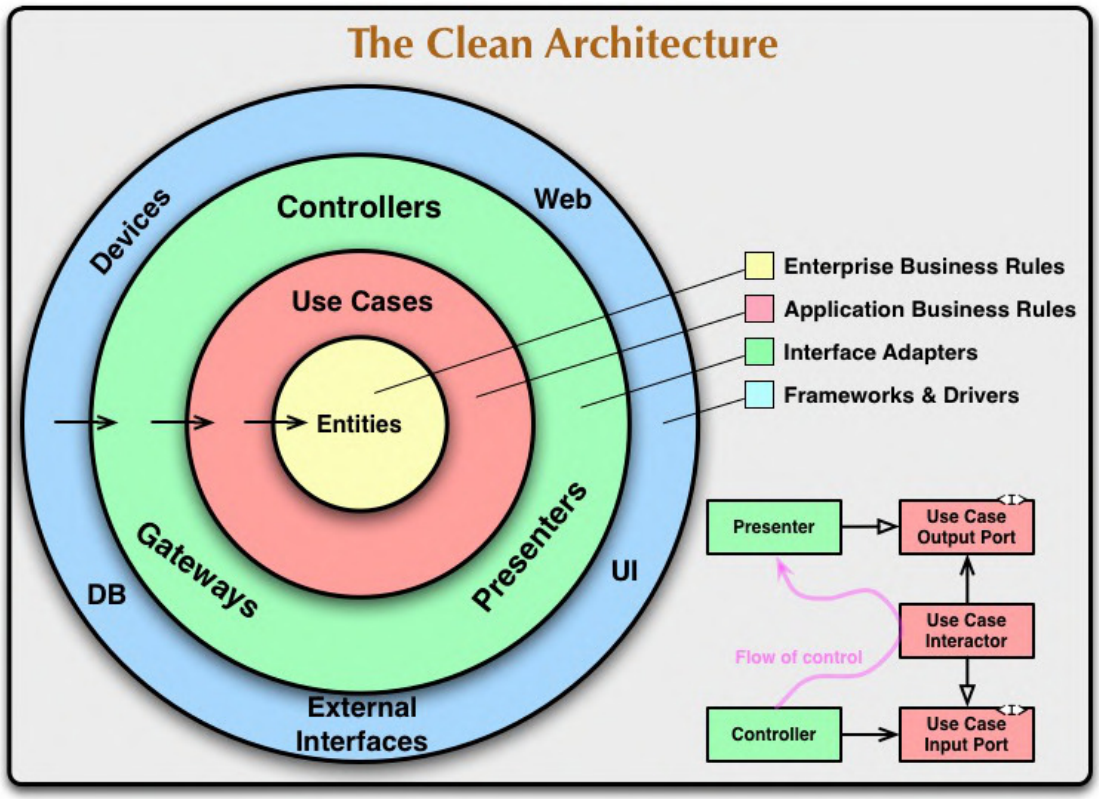

Figura 2. Camadas do *Clean Architecture*

2.2.3. Linguagem de programação

A linguagem de programação usada no *backend* foi Java, com o kit de desenvolvimento OpenJDK.

2.2.4. Ecossistema de *frameworks* Spring

Além da linguagem de programação, foram utilizados alguns *frameworks* do ecossistema Spring, segue abaixo os utilizados:

- **Spring Boot**: É um framework com intuito de facilitar a criação e desenvolvimento de aplicações Spring;

- **Spring Framework**: É o "coração" do ecossistema, é peça fundamental para que uma aplicação e seus frameworks Spring funcionem;
- **Spring Data**: É o framework usado na parte de persistência de dados;
- **Spring Cloud**: Framework essencial utilizado para construção de aplicações distribuídas, como microsserviços;
- **Spring Security**: Framework de segurança de autenticação e controle de acesso personalizável;
- **Spring for GraphQL**: Dá suporte a linguagem de consulta para APIs, GraphQL, para aplicações Spring.

2.2.5. Outros *frameworks* utilizados

- **Lombok**: Biblioteca de anotações para redução de *boilerplate* em códigos Java. Com Lombok, podemos adicionar métodos *getters*, *setters*, construtores a classes em tempo de compilação;
- **Twilio**: API para envio de mensagens;
- **JUnit**: Framework de testes de unidade para Java. Também pode ser usado com testes de integração, mas juntamente com outros *frameworks*.

2.2.6. Gerenciamento de dependências

Todas essas dependências do citadas anteriormente são geridas pelo Gradle, uma ferramenta de automação e gerenciamento de dependências bastante flexível e rápida usada para auxiliar na construção de projetos Java, C, C++ e Javascript.

2.2.7. Segurança

Na parte de segurança, está sendo utilizado o Keycloak, uma aplicação responsável por gerir credenciais de usuários e suas permissões. Todo microserviço do *backend* tem conexão com o Keycloak(feita atráves do Spring Boot e Spring Security), isso porque é necessário gerenciar o token JWT recebido pelas requisições aos serviços, para saber se ele está válido, e se o usuário desse token tem permissão para usar alguma funcionalidade. O token JWT é obtido pelo microserviço de login e cadastro, e gerado pelo próprio Keycloak.

2.2.8. API

Como em quase todo sistema, é necessário ter uma *Application Programming Interface*, a API, que é uma interface para termos acessos a serviços e dados. Algumas APIs da aplicação serão em GraphQL, que é uma linguagem de consulta, que tem como finalidade deixar as APIs mais rápidas, fluídas e intuitivas para os desenvolvedores. É uma alternativa a arquitetura REST comumente usada nas APIs com requisições HTTPs.

Em contrapartida ao REST, GraphQL tem uma alta flexibilidade, podemos indicar em uma consulta quais campos de um determinado dado são necessários no momento, e isso acaba ajudando no tráfego e posteriormente no tratamento.

APIs que não puderam serem feitas em GraphQL, como em casos de buscas mais minuciosas, precisas, foram feitas em REST. Essas buscas são feitas com filtros e ordenação, por exemplo, buscar produtos ordenados por nome, que tenham o preço entre R$ 10,00 e R$ 20,00 e descrição com a palavra "ácido".

2.2.9. Banco de dados

- **PgAdmin4**: Software open-source de administração e desenvolvimento para PostgreSQL;
- **PostgreSQL**: Também conhecido como Postgres [Oliveira 2006], é um sistema de gerenciamento de banco de dados objeto-relacional (SGBDRO), desenvolvido pela PostgreSQL Global Development Group, sobre a licença PostgreSQL License. PostgreSQL se destaca por ser robusto e possuir vários recursos. Para base de dados muito grande, complexa e que exige confiabilidade e escalabilidade vale a pena usar o PostgreSQL;
- **H2**: Banco de dado geralmente utilizado em testes de aplicações Java.

2.2.10. Modelagem de dados

Inicialmente, para termos uma boa modelagem de dados, caso ela seja relacional, precisamos passar por 3 níveis: Nível conceitual, aqui focamos apenas no que pode ser armazenado de uma entidade, sem se preocupar de que forma é armazenado. Definimos as entidades, seus atributos e relacionamentos a partir de alguma realidade vista; Nível lógico: Obtido a partir do conceitual, nesse nível, definimos a estrutura do banco, ou seja, tipos dos atributos da relações, anteriormente chamadas de entidades, criação de alguma nova relação, normalização, definições de chaves primárias e estrangeira e outros; Nível físico: Basicamente é o nível lógico definido a partir da linguagem SQL. No caso da nossa aplicação, vamos seguir outro caminho para essa modelagem. Como estamos utilizando o paradigma orientado a objetos no *backend*, precisamos utilizar alguma estratégia para transformar, converter os dados nessa paradigma para o relacional, e para fazer isso, é necessário usar um framework ORM, onde geralmente é utilizado o Hibernate. Esses frameworks abstraem o banco, ele acaba se comportando como um banco orientado a objetos. Com isso, foram feitos UMLs das classes de cada microserviço do *backend*. O uso do Hibernate é feito pelo Spring Data JPA.

2.2.11. Entidades

As entidades e suas UMls seram mostradas de acordo com cada microserviço.

Patient Backend

Patient

A entidade *Patient* representa um paciente/usuário do sistema que realiza as compras. Possuí os campos: cpf, name, lastName, birthDate, email, phoneNumber, gender, address e creditCards. Campo cpf é utilizado como Id.

Address

A entidade *Address* representa o endereço usado por um paciente para receber remédio após a compra. Possuí os campos: id, street, streetNumber, district, city, state, country, complement e cep.

CreditCard

A entidade *CreditCard* representa o cartão que o paciente pode usar para pagar uma compra. Possuí os campos: number, name, expirationDate, securityCode e patient.

Gender

O *enum Gender* representa o gênero do paciente. Possuí as constantes: MALE, FEMALE e OTHER.

Doctor Backend

Doctor

A entidade *Doctor* representa um doutor que assinará prescrições. Possuí os campos: crm, cpf, name, lastName, birthDate, email, phoneNumber, gender, address e creditCards. O campo crm é usado como Id. CRM indica o registro de um médico.

Address

A entidade *Address* representa o endereço usado por um doutor. Possuí os campos: id, street, streetNumber, district, city, state, country, complement e cep.

Gender

O *enum Gender* representa o gênero do doutor. Possuí as constantes: MALE, FEMALE e OTHER.

Pharmacy Backend

Pharmacy

A entidade *Pharmacy* representa uma farmácia que vende remédios. Possuí os campos: cpnj, name, description, openingHours, email, cnpj, phoneNumber e address.

Address

A entidade *Address* representa o endereço usado por uma farmácia. Possuí os campos: id, street, streetNumber, district, city, state, country, complement e cep.

OpeningHours

A entidade *OpeningHours* representa o horário de funcionamento de algum dia da farmácia. Possuí os campos id, openingTime, closingTime, workingDay e pharmacy.

Day

O *enum Day* representa o os dias da semana, usado na entidade *OpeningHours*. Possuí as constantes: MONDAY, TUESDAY, WEDNESDAY, THURDAY, FRIDAY, SATURDAY e SUNDAY.

Discount Coupon Backend

DiscountCoupon

A entidade *DiscountCoupon* representa um cupom de desconto que pode ser usado na compra por algum paciente. Possuí os campos: code, description, pharmacyCnpj, discount, validFrom, validUntil, isSingleUse, minimumPurchasesAmount, active, createdAt e updateAt.

Prescription Backend

Prescription

A entidade *Prescription* representa uma prescrição. Possuí os campos: documentHash, comment, date, signature, patientCpf, pharmacyCnpj, doctorCrm, productId, isValid. O campo documentHash representa o Id da prescrição, e também é o Id na base de dados da Certilion. O campo signature representa a assinatura da prescrição, usada para fazer validações com a API da Certilion.

Remedy Backend

Product

A entidade *Product* representa um produto de farmácia. Possuí os campos: id, ean, pharmacyCnpj, name, description, imageUrl, weight, unit, status, amount, brand, manufacturer, price, stock e category.

Category

A entidade *Category* representa uma categoria para algum produto. Possuí os campos: id, name, description e products.

Status

O *enum Status* representa o status de um produto. Possuí as constantes: ACTIVE e DESACTIVE.

Unit

O *enum Unit* representa o o tipo de unidade de medida de peso de um produto. Possuí as constantes: MG, G e KG.

Purchase History Backend

PurchaseHistory

A entidade *PurchaseHistory* representa o histórico/pedido feito por um paciente a uma farmácia. Possuí os campos: id, timestamp, shippingPrice, patientCpf, pharmacyCnpj, status, pay-

mentMethod e orderItens.

OrdeItem

A entidade *OrdeItem* representa item ou itens do pedido. Possuí os campos: id, productId, prescriptionId, discountCouponId, amount, totalValue e purchaseHistory.

PaymentMethod

O *enum PaymentMethod* representa o tipo de pagamento usado pelo paciente na compra. Possuí as constantes: CARTAO_CREDITO, CARTAO_DEBITO e PIX.

Status

O *enum Status* representa o status do pedido. Possuí as constantes: ABERTO, ANALISE E FECHADO.

Gateway Certilion Backend

Code

A entidade *Code* representa o código necessário para assinar uma prescrição. Possuí os campos: code e doctorCrm.

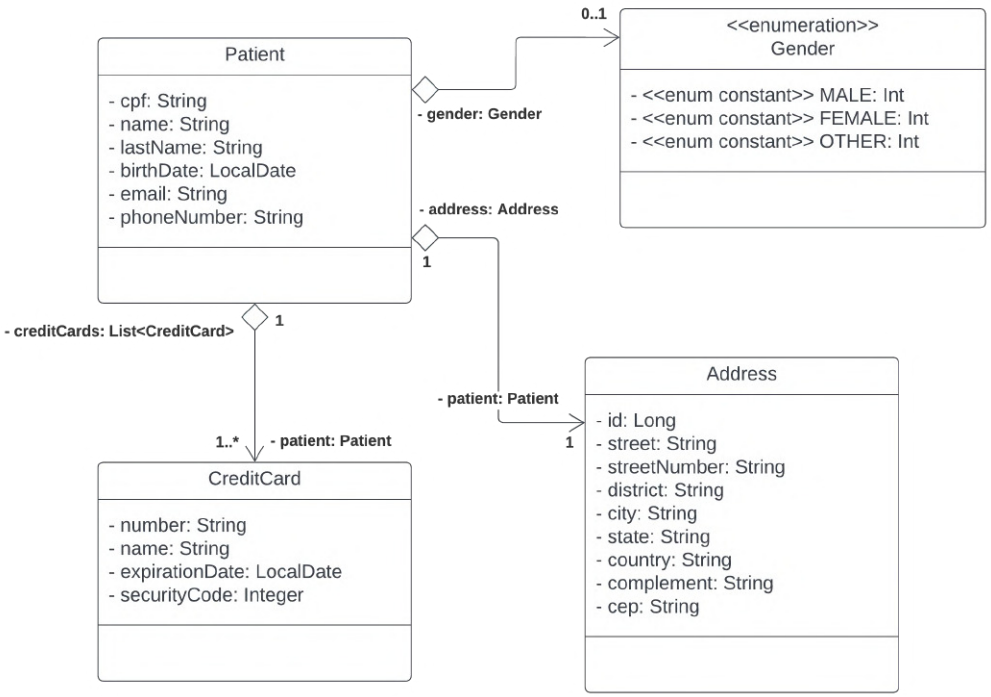

Figura 3. Diagrama do *Patient Backend*

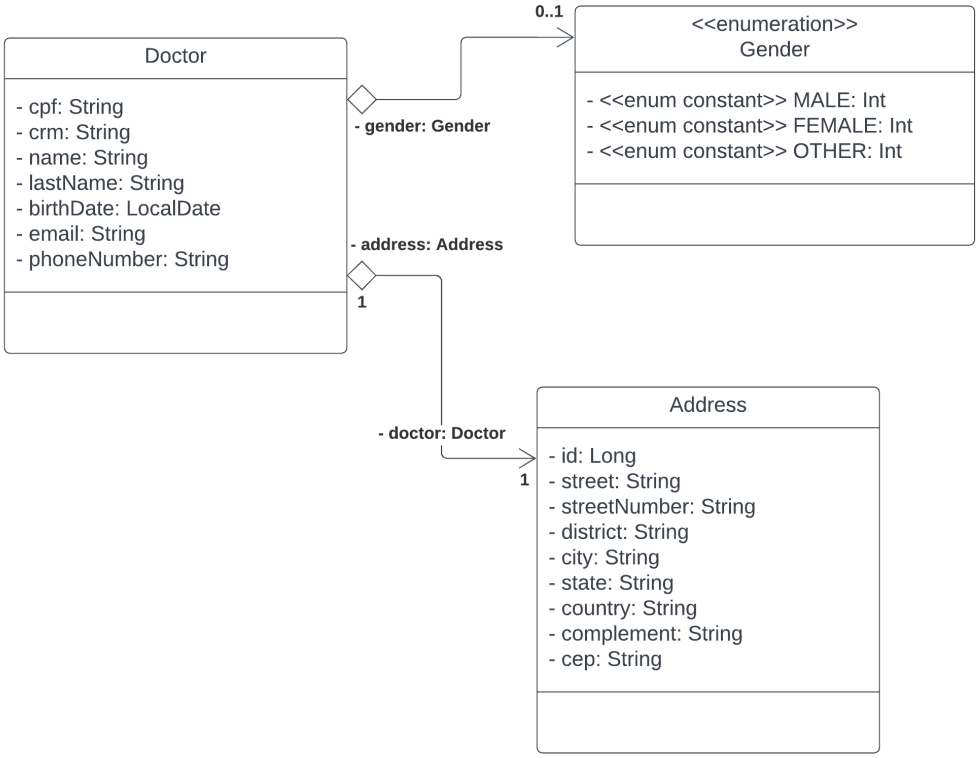

Figura 4. Diagrama do *Doctor Backend*

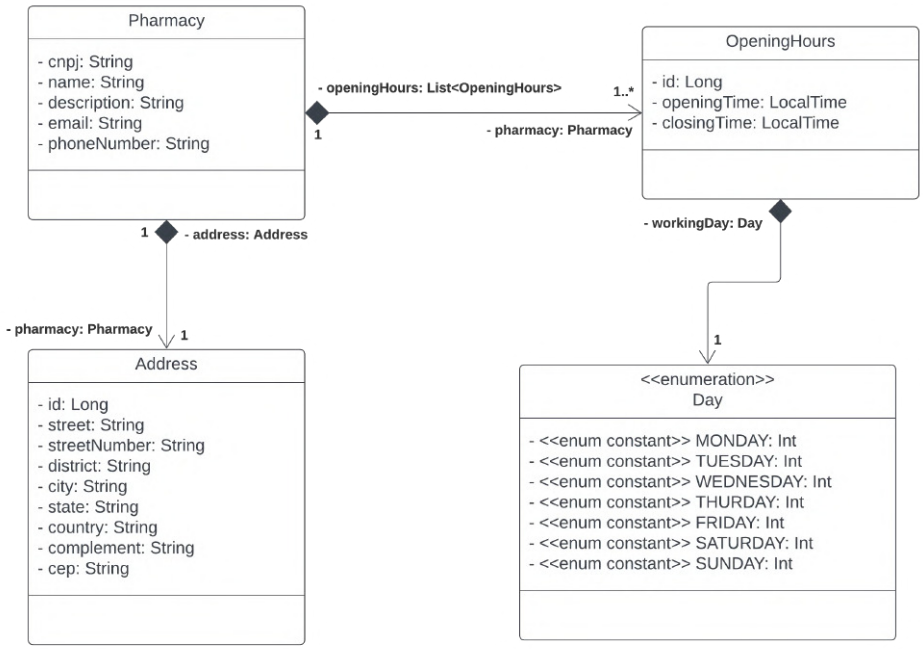

Figura 5. Diagrama do *Pharmacy Backend*

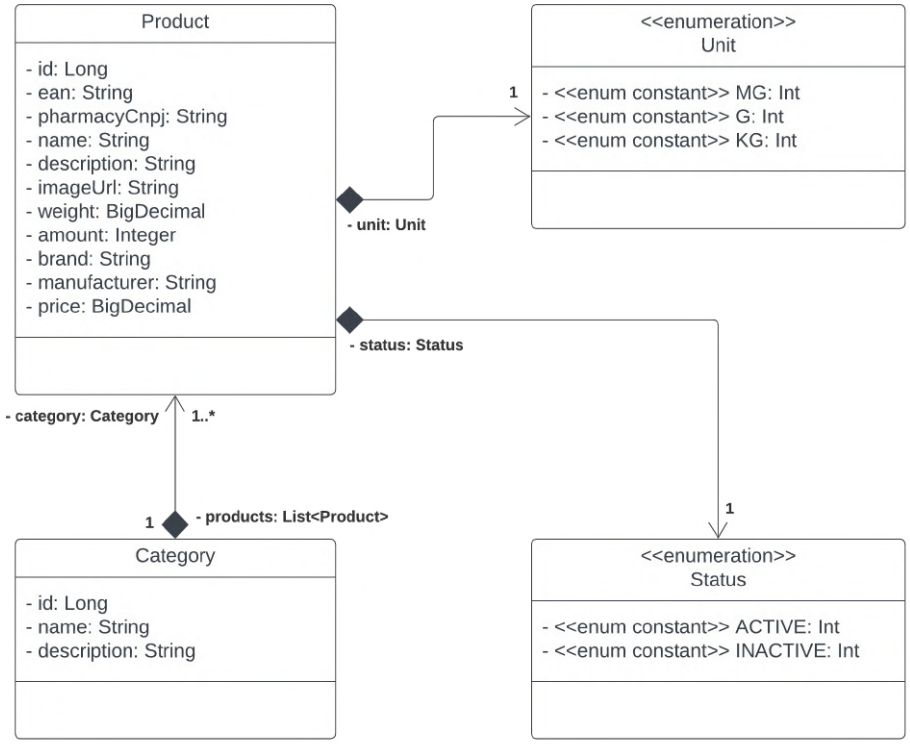

Figura 6. Diagrama do *Remedy Backend*

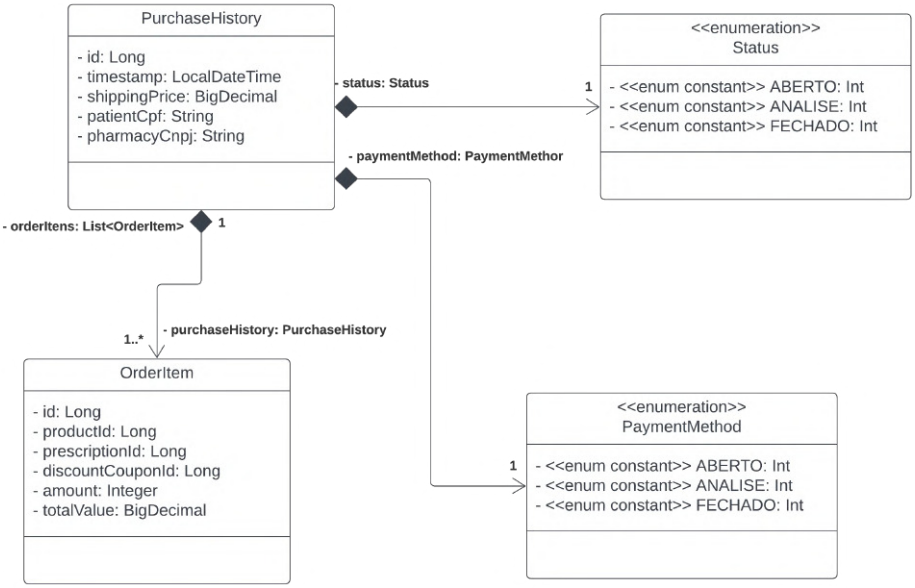

Figura 7. Diagrama do *Purchase History Backend*

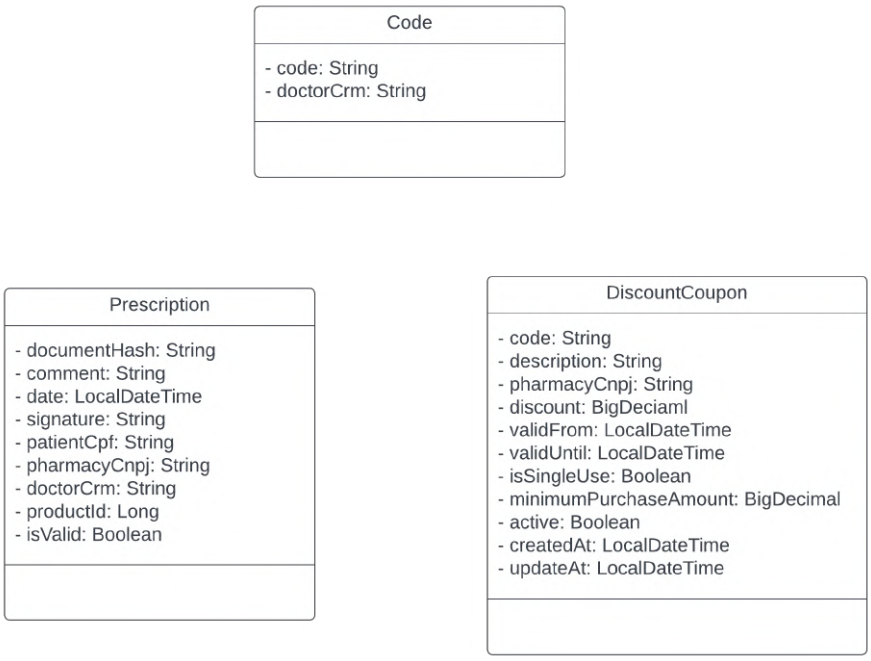

Figura 8. Diagrama do *Gateway Certilion Backend, Prescription Backend e Discount Coupon Backend*

2.2.12. Testes de integração

O teste de integração é o teste entre diferentes módulos em um sistema, aqui eu cito o teste de requisições HTTP, o mais comum para servidores. Nesse teste, você verifica o resultado de uma requisição completa, analisando o formato de resposta, o código de status na resposta HTTP, o formato de dados e validações. Esse teste é muito importante, e facilita muito em futuras manutenções, onde a criação de novas funcionalidades ou alterações em antigas funcionalidades podem modificar o comportamento de outras partes do sistema [Fernandes 2018].

No sistema, foram feitos testes de integração das APIs em GraphQL e REST.

2.3. *Frontend*

2.3.1. Linguagens de programação

As linguagems de programação usadas no *frontend* foram JavaScript e TypeScript.

2.3.2. Frameworks e ferramentas

- **Vue.js**: É um framework JavaScript progressivo usado para construir interfaces de usuário.
- **Vite**: É um compilador de desenvolvimento ultrarrápido e configurável para aplicações JavaScript.
- **Vuetify**: É um framework de componentes UI para Vue.js que fornece estilos e elementos de interface prontos para uso.

- **Pinia**: É um gerenciador de estados para Vue.js que permite uma gestão eficiente do estado da aplicação.
- **React Native**: É um framework de desenvolvimento de aplicativos móveis que permite a criação de aplicativos nativos para iOS e Android usando JavaScript e React.
- **Native Base**: É um conjunto de componentes de interface de usuário para React Native que oferece estilos e elementos prontos para uso.
- **Expo**: É uma plataforma de desenvolvimento de aplicativos móveis que simplifica o processo de criação, teste e implantação de aplicativos usando React Native.
- **MobX**: É uma biblioteca de gerenciamento de estado para React que permite um controle eficiente do estado da aplicação.
- **Jest**: É uma estrutura de teste de código aberto amplamente utilizada para JavaScript e React que oferece suporte para testes unitários, integração e snapshot.
- **Vitest**: É um ambiente de testes para Vue.js que permite escrever e executar testes unitários e de integração para os componentes da aplicação.
- **Apollo Client**: É uma biblioteca de gerenciamento de estado de cliente GraphQL para aplicações JavaScript.
- **Storybook**: É uma ferramenta de desenvolvimento para UI componentes, permitindo a visualização e documentação dos componentes individualmente.

2.4. Ferramentas auxiliares para o desenvolvimento geral

- **Keycloak**: Keycloak é uma ferramenta criada pela empresa Red Hat que faz gerenciamento de credenciais de usuários e de suas permissões. Pode ser usada como repositório oficial de usuários de uma companhia, seja através de um cadastro, ou de um vínculo com uma base de dados já existente, como ldap, por exemplo.;
- **Git**: Sistema de controle de versão de códigos;
- **Github**: Servidor de armazenamento de projetos que utiliza Git;
- **Docker**: Tecnologia de conteinerização open-source baseada em Linux, que tem a finalidade de facilitar a criação e administração de ambientes isolados;
- **Notion**: Plataforma de produtividade e gerenciamento de projetos tudo-em-um que combina as funcionalidades de aplicativos como bloco de notas, gerenciador de tarefas, wiki e muito mais;
- **Figma**: Editor online é focado em trabalho colaborativo, permite criar interfaces e protótipos;
- **Lucidchart**: Site para criação de diagramas e wireframes.
- **Discord**: Plataforma de comunicação com chamadas de vídeos, aúdios, trocas de mensagens em grupos, comunidades ou de formar particular. Foi usada para comunicação dos desenvolvedores.
- **Google Meet**: Plataforma para videoconferência. Usada nos *Plannings* semanais

3. Metodologia

3.1. Métodos Ágeis

A construção do projeto da Facilifarma foi realizada seguindo a metodologia ágil de desenvolvimento de software com encontros semanais chamados de *planning*, que é conhecida por proporcionar maior flexibilidade e adaptabilidade às mudanças e demandas que possam surgir ao longo do processo de desenvolvimento. O uso desta metodologia permitiu uma maior colaboração entre a equipe de desenvolvimento, possibilitando a definição de metas e entregas mais precisas e efetivas.

O desenvolvimento do projeto foi dividido em *sprints*, que são ciclos de tempo pré-definidos, nos quais a equipe trabalha na entrega de uma parte específica do projeto. Para o gerenciamento do projeto, foi utilizado o Notion, uma ferramenta de gestão de projetos online que permite a criação de quadros, listas e cartões para o acompanhamento das tarefas e atividades realizadas pela equipe.

3.2. Tecnologias utilizadas no projeto

3.2.1. *Frontend*

A construção do *Frontend* foi realizada utilizando as seguintes tecnologias: Vite, Vue.js, TypeScript, Vuetify, Pinia, Apollo Client (para requisições GraphQL), Vitest (para testes unitários) e Storybook (para documentação de componentes). Essas tecnologias foram escolhidas por serem reconhecidas por sua eficiência, flexibilidade e facilidade de uso.

Por fim, foi adotado um processo de revisão de código, em que cada alteração realizada no código-fonte passou por uma revisão por um ou mais membros da equipe antes de ser integrada ao código principal. Isso permitiu uma maior segurança e qualidade do código produzido, bem como a identificação e correção de erros e problemas com maior agilidade.

3.2.2. *Backend* e banco de dados

Para o desenvolvimento do *backend* do sistema, foi utilizado a linguagem de programação Java com o OpenJDK, juntamente com os frameworks Spring Boot, Spring Framework, Spring Data, Spring Cloud, Spring Security, Spring for GraphQL, Lombok, Twilio, Junit.

A nível de projeto, o *backend* foi feito usando *Clean Architecture*, arquitetura responsável por prover sistemas coesos, testáveis, que funcionam independentemente de qualquer tecnologia. A nível de aplicação, foi utilizado microservices, um padrão arquitetural que descreve uma aplicação como um conjunto de pequenos serviços responsáveis por uma determinada função, que se comunicam por meio de alguma API.

Todos esses frameworks são geridos pelo Gradle, ferramenta de automação e gerenciamento de dependências bastante flexível e rápida usada para auxiliar na construção de projetos Java, C, C++ e Javascript.

A segurança do sistema é feito por meios de tokens JWT, geridos e criados pela aplicação Keycloak, que tem o intuito de gerir credenciais de usuários e suas permissões.

A comunicação com *backend* é feita por meios de APIs GraphQL e REST. GraphQL é uma linguagem de consulta, que tem como finalidade deixar as APIs mais rápidas fluídas e intuitivas para os desenvolvedores, é uma alternartiva a arquitetura REST com requisições HTTP.

Os testes seram de integração, foram feitos nas APIs GraphQL e REST, usando JUnit, juntamente com Spring Boot e HttpClient, biblioteca de cliente HTTP própria do Java.

Para persistência, foi utilizado PostgreSQL com PgAdmin4 e H2.

O Docker foi utilizado para ter uma facilidade na execução do sistema, além de isola-lo do SO da máquina. Todos os microserviços e banco de dados rodam através do Docker em forma de containers, criados com Dockerfile e Docker Compose. Uso de *containers* tras a vantagem como de ter alta resiliência, performance, portabilidade e escalabilidade.

3.3. Requisitos Funcionais

Os requisitos funcionais desempenham um papel crucial no desenvolvimento de qualquer sistema, fornecendo uma descrição clara e detalhada das funcionalidades esperadas. No contexto do projeto Facilifarma, foram identificados uma série de requisitos funcionais que são essenciais para garantir o bom funcionamento da plataforma. Esses requisitos são responsáveis por definir as principais funcionalidades que a aplicação deve oferecer aos usuários.

A tabela de requisitos funcionais apresentada na Tabela 3 lista os requisitos identificados, cada um identificado por um ID e acompanhado de uma descrição concisa.

Esses requisitos funcionais foram identificados com base nas necessidades dos usuários, bem como nas metas e objetivos estabelecidos para o Facilifarma. A sua implementação é fundamental para proporcionar uma experiência eficiente, segura e satisfatória aos usuários, permitindo-lhes realizar compras de medicamentos de forma conveniente e acessível.

Ao cumprir esses requisitos funcionais, a equipe de desenvolvimento do Facilifarma busca atender às expectativas dos usuários, oferecendo uma plataforma completa e intuitiva que atenda às suas necessidades e supere as limitações do processo tradicional de compra de medicamentos em farmácias físicas.

A seguir, serão detalhados os requisitos funcionais em termos de suas especificações técnicas e sua importância para o sucesso e eficácia do sistema Facilifarma.

Tabela 3. Requisitos Funcionais

ID	Requisito
RF1	O sistema deve permitir o cadastro de usuários
RF2	Os usuários devem poder fazer login e autenticar-se no sistema
RF3	A aplicação deve exibir a lista de produtos disponíveis na farmácia
RF4	Os usuários devem poder adicionar produtos ao carrinho de compras
RF5	O sistema deve permitir que os usuários realizem o pagamento online
RF6	Os usuários devem receber notificações sobre o status do pedido
RF7	As farmácias devem ter acesso a uma dashboard de controle para gerenciar suas vendas e se comunicar com os clientes
RF8	A dashboard deve fornecer estatísticas e relatórios sobre o número de vendas, clientes alcançados, produtos vendidos, entre outros dados
RF9	As farmácias devem receber suporte técnico eficiente e rápido para resolver problemas e responder às perguntas dos usuários

3.4. Fluxogramas de Navegação

Os fluxogramas de navegação de telas desempenham um papel fundamental no desenvolvimento do frontend, proporcionando uma representação visual clara e organizada do fluxo de navegação de uma aplicação. Esses fluxogramas ajudam os designers e desenvolvedores a entenderem como os usuários podem interagir com a aplicação, quais são as diferentes telas ou páginas disponíveis e como elas estão conectadas.

Ao criar um fluxograma de navegação, é possível identificar os caminhos possíveis que um usuário pode percorrer ao utilizar a aplicação, bem como as ações que podem ser realizadas em cada tela. Isso auxilia na definição da arquitetura de informação e na determinação da hierarquia das telas, facilitando a criação de uma experiência de usuário intuitiva e coerente.

Por exemplo, um fluxograma de navegação de um aplicativo de compras online pode mostrar que a partir da tela inicial, o usuário pode navegar para a tela de categorias de produtos, escolher uma categoria específica, visualizar os produtos disponíveis, adicionar itens ao carrinho e, em seguida, prosseguir para a tela de finalização da compra. Esse fluxograma permite que os desenvolvedores compreendam a sequência de telas e a lógica de transição entre elas, facilitando o desenvolvimento da interface e a implementação das funcionalidades necessárias.

Em resumo, os fluxogramas de navegação de telas são uma ferramenta essencial no processo de desenvolvimento do frontend, ajudando a visualizar e planejar o fluxo de interação do usuário, melhorar a usabilidade da aplicação e alinhar as expectativas entre os membros da equipe de desenvolvimento e os stakeholders.

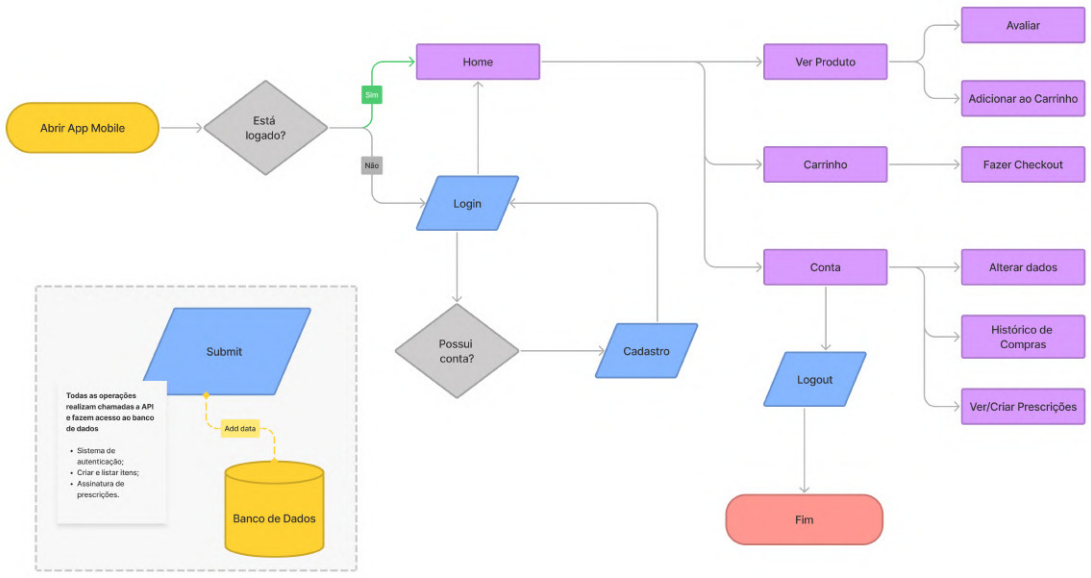

Figura 9. ***Fluxograma do Aplicativo***

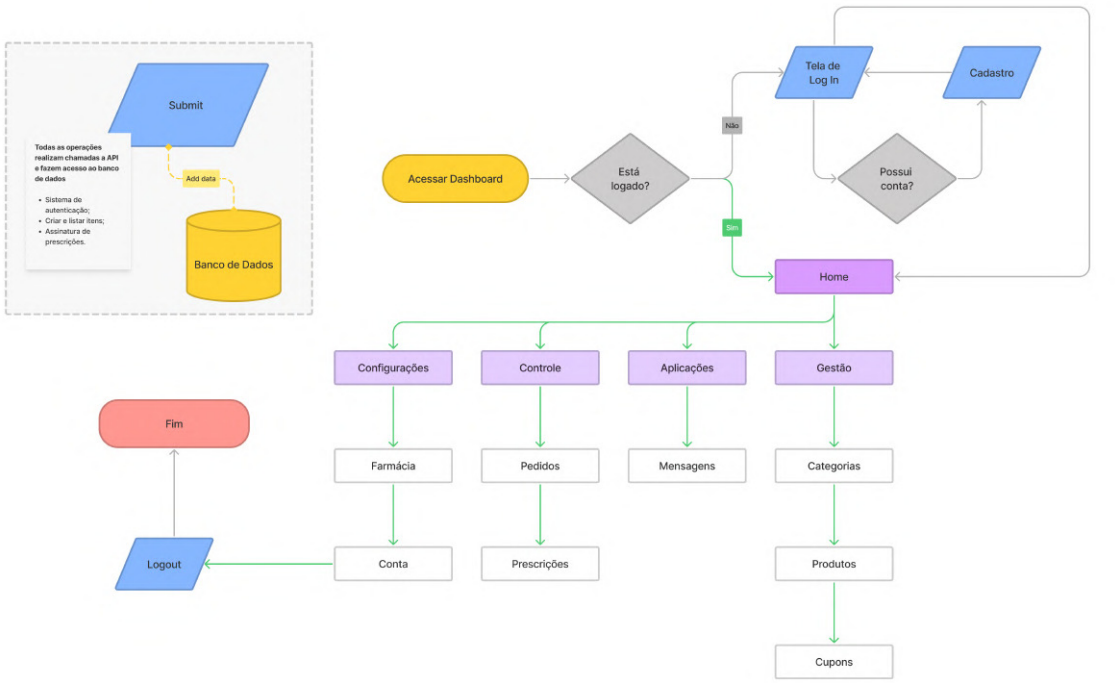

Figura 10. ***Fluxograma da Dashboard***

4. Resultados

Os resultados obtidos foram satisfatórios e cumprem os objetivos técnicos e específicos almejados. As farmácias podem utilizar a *dashboard* para criar uma conta utilizando seu CNPJ e criando uma senha. Os clientes e médicos podem criar uma conta no aplicativo utilizando seu CPF e CRM (no caso de médicos). O sistema de compra e venda de medicamentos conta com a opção de adicionar cupons de desconto feitas pela farmácia vendendo o produto, e também de anexar prescrições feitas pelo médico diretamente pelo aplicativo na seção de prescrições localizadas na seção de configurações da conta.

4.1. Telas da *Dashboard*

As telas da *Dashboard* podem ser organizadas e classificadas da seguinte forma:

Landing Page Tela inicial.
Auth Telas relacionadas a autenticação e cadastro de usuários.
Main As telas principais internas da *Dashboard* que só podem ser acessadas por usuários autenticados.
Error Telas de *callback* de erros e eventos inesperados para os usuários serem redirecionados quando algo não sair conforme esperado durante a navegação pelas rotas da aplicação.

Dentro da *Dashboard* por sua vez, as telas estão agrupadas conforme mostrado à seguir, e esse agrupamento pode ser visto na barra lateral de navegação da aplicação, como poderá ser observado nas *screenshots* mais adiante.

Home Tela inicial com sumários e gráficos.

Controle Telas relacionadas a controle de dados da farmácia.
Aplicações Aplicações extras para a farmácia.
Gestão Telas de gestão de conteúdo da farmácia.
Configurações Telas de configuração da conta autenticada e da farmácia.

4.1.1. *Landing Page*

Para fins de *marketing* e atração de novos clientes, a aplicação conta com uma tela inicial chamada de "*Landing Page*", conforme mostra a figura 11, onde novos visitantes podem obter um resumo do que se trata o produto e suas principais funcionalidades e características. Além disso, serve como ponto de partida para acessar a *Dashboard*.

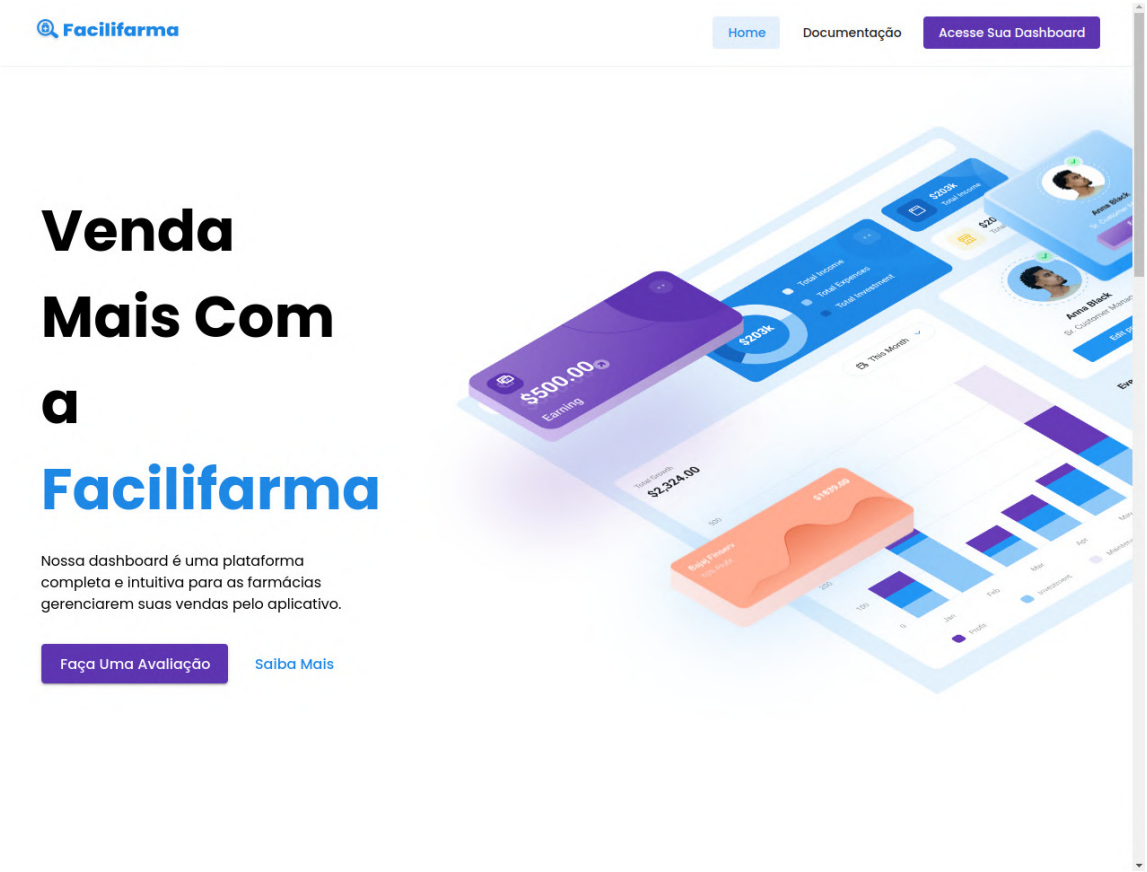

Figura 11. ***Landing Page Hero***

4.1.2. Autenticação

Para realizar a autenticação na *Dashboard* o usuário deve realizar o login através da tela específica para tal, explicitada na figura 12, usando o CNPJ da farmácia cadastrada bem como sua senha.

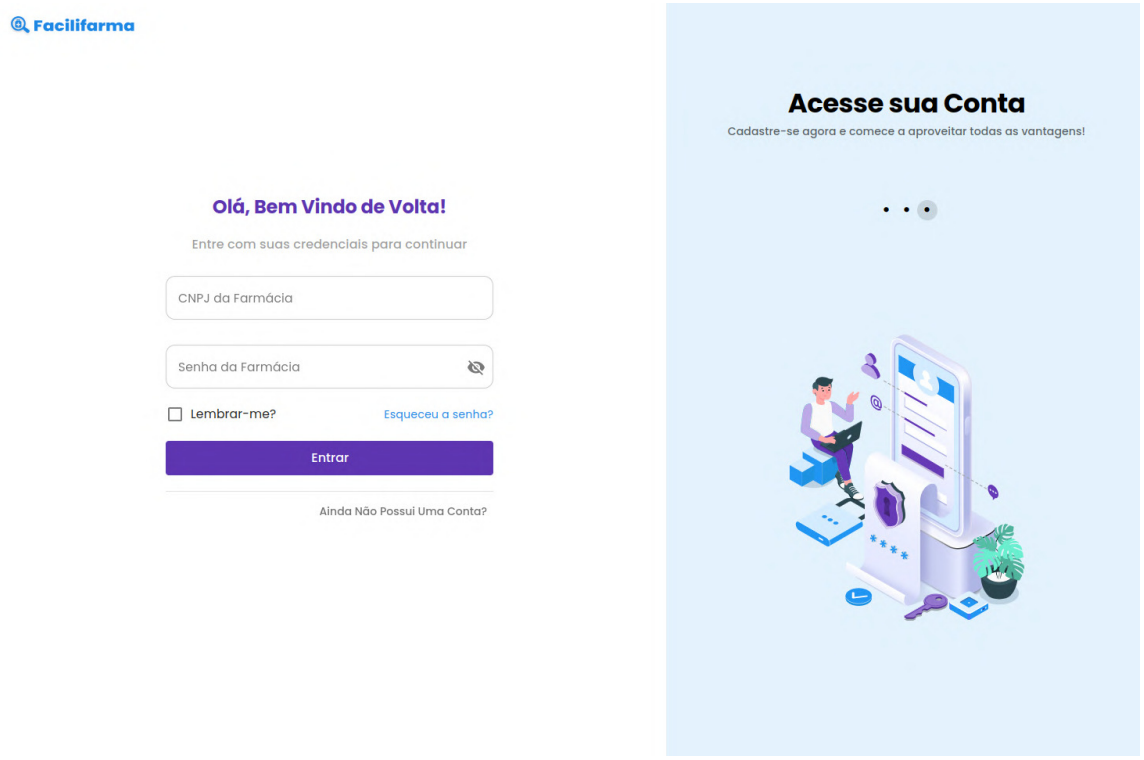

Figura 12. Página de *Login*

4.1.3. Cadastro

Para realizar o cadastro de novas farmácias, uma tela de registro é disponibilizada. Conforme mostrado na figura , nela os usuários podem informar os dados da farmácia à ser cadastrada em um "*multi-step form*"(formulário com múltiplos passos). Ao final, um código de verificação é enviado para o número de celular informado durante o processo de cadastro para o mesmo poder ativar a conta.

4.1.4. *Home*

A tela de *Home* fica responsável por exibir um sumário de vendas e gráficos importantes para visualização de dados para o administrador da farmácia.

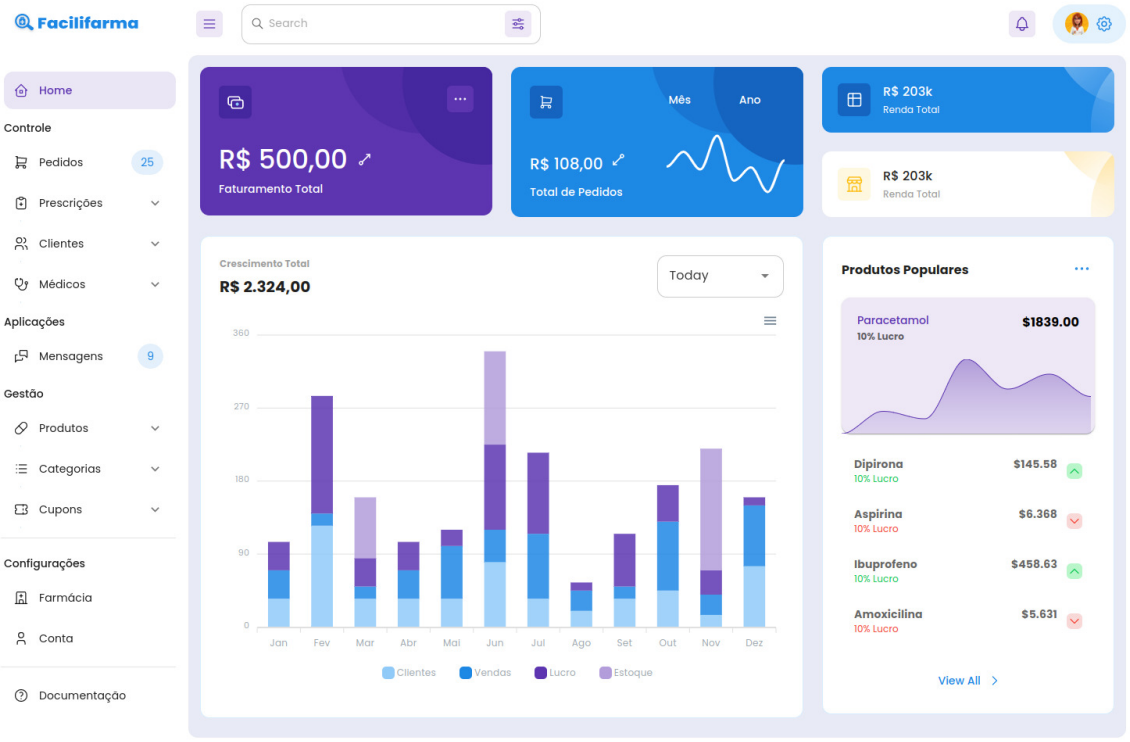

Figura 13. Tela de *Home* da *Dashboard*

4.1.5. Controle

A seção de controle da *dashboard* conta com páginas relacionadas ao controle de entrada e saída de mercadoria da farmácia, bem como prescrições. As telas de Pedidos, Prescrições, Clientes e Médicos apresentam tabelas com resumos.

Cada vez que um usuário realiza uma interação com a farmácia, isto é, comprando um medicamento, ou assinando uma prescrição (no caso de usuários médicos), os seus dados vão para uma dessas tabelas para a farmácia ter seu controle.

4.1.6. Aplicações

Na seção de aplicações podem ser encontradas aplicações adicionais para ajudar na dinâmica da farmácia. Um exemplo disso, é o chat, que poda ajudar a farmácia a manter o contato com seus clientes.

4.1.7. Gestão

Na seção de gestão a farmácia pode gerir seus conteúdos, ou seja, adicionar categorias, produtos e cupons de descontos, ativando-os e os tornando visíveis no aplicativo para serem visualizados e comprados pelos usuários.

4.1.8. Configurações

Na parte de configurações, o administrador da farmácia pode alterar o perfil da farmácia, como, por exemplo, seu nome de exibição, slogan, biografia, etc. E também, ajustar as preferências de navegação da *dashboard* para melhor experiência, como tema e *layout*, e da conta, como senha.

4.2. Telas do Aplicativo

As telas do aplicativo podem ser divididos em duas partes:

Autenticação Telas de autenticação para usuários não autenticados.
Principal Telas principais da aplicação para usuários autenticados.

Dentro do aplicativo, por sua vez, as telas estão agrupadas conforme mostrado à seguir, e esse agrupamento pode ser visto na barra inferior de navegação da aplicação, como poderá ser observado nas *screenshots* mais adiante.

Home Tela inicial com os produtos cadastrados e disponibilizados para compra pelas farmácias.
Carrinho Telas relacionadas ao pedido à ser realizado pelo usuário.
Configurações Telas de configuração da conta autenticada.

4.2.1. Cadastro

A tela de cadastro pode ser acessada atráves da tela de *Welcome*, e pode ser de duas maneiras: cadastro para Pacientes e Médicos. A diferença entre os dois é que um deve obrigatoriamente informar o seu CRM (o médico).

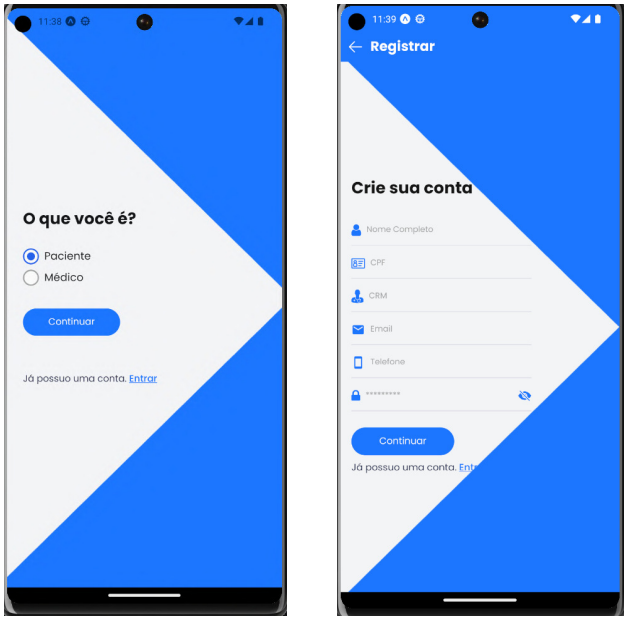

Figura 14. Tela de escolha de tipo de conta para cadastro e cadastro, respectivamente

4.2.2. *Home*

Essa é a tela principal, onde os usuários podem adicionar os produtos disponibilizados pelas farmácias em seu carrinho para, posteriormente, efetuar o pedido de compra.

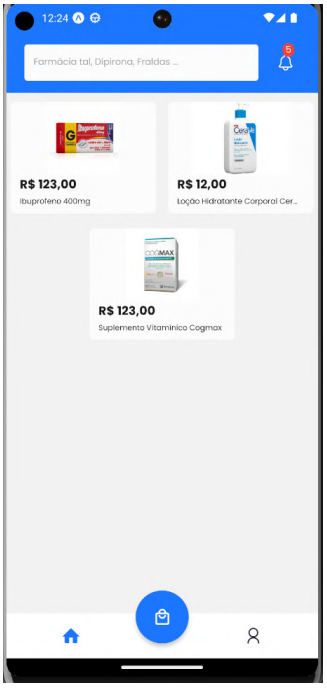

Figura 15. Tela de Home do Aplicativo com alguns produtos

4.2.3. Carrinho e *Checkout*

Nesta tela ficam localizados todos os produtos que o cliente deseja comprar. Além disso, é à partir dela que ele pode seguir para o processo de *Checkout*, adicionando um endereço de entrega, selecionando o meio de pagamento e anexando prescrições e cupons de desconto ao seu pedido.

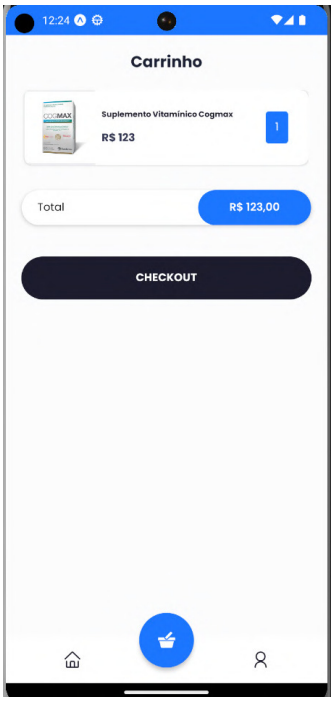

Figura 16. Tela de Carrinho do Aplicativo com produto de exemplo

4.2.4. Configurações

Aqui os usuários podem alterar seus dados, ver o histórico de pedido e também suas prescrições. No caso de médicos, a criação e assinatura de prescrições também é feita por essa tela, na aba de "Prescrições"

Figura 17. Tela de Perfil do Aplicativo na aba de prescrições

4.3. Processo de Criação e Assinatura de Prescrições pelo App

Primeiramente, para um médico assinar uma prescrição, é necessário criar uma solicitação de autorização contendo os parâmetros necessários para o PSC (empresa/servidor que guardará o certificado digital do médico. Usado pela Certilion para pode assinar algum documento) identificar a aplicação cliente(no caso, a Certilion) e solicitar autorização ao usuário conforme a permissão de uso requisitada. Após isso, o médico preenche as campos das prescrições e as envia para o *Gateway Certilion Backend*, que fara o upload da junção dessas informações para a Certilion, e, também um upload de informações, com algumas vindas da Certilion como o documentHash, para o *Prescription Backend*. O documentHash é necessário para assinar a prescrição e validar.

Após essa etapa, o médico assina a prescrição, novamente enviando uma requisição para o *Gateway Certilion Backend*, que se comunicará com a Certilion e depois o *Prescription Backend*. Nessa etapa, é salvo o signature no *Prescription Backend*, que é realmente de fato a assinatura da prescrição. Ela precisa ser salva também para uma possível validação. Por fim, o paciente tem a prescrição assinada pelo médico, e com isso, ele pode fazer a compra de algum medicamento.

Para realizar uma compra, o cliente deve abrir a página do medicamento e adicioná-lo ao carrinho. A tela principal, apelidada de "Home"contém uma gama de medicamentos criados e disponibilizados pelas farmácias. As farmácias podem criar seus medicamentos através da *dashboard* na página de cadastro de produtos.

Na tela de *Checkout*, o cliente pode anexar uma de suas prescrições na compra, ao finalizar a compra, a farmácia receberá o pedido e poderá vê-lo na listagem de pedidos da *dashboard*. O mesmo ocorre com as prescrições. A farmácia possui uma tela de listagem específica para prescrições. Toda vez que um novo pedido com prescrição é recebido pela farmácia, a listagem de prescrições é atualizada com a nova prescrição recebida e o detalhe do número do pedido assim como o CPF do cliente à qual a prescrição pertence.

Nessas telas a farmácia pode ver os detalhes e invalidar a prescrição, garantindo que a mesma não possa ser utilizada em outras compras.

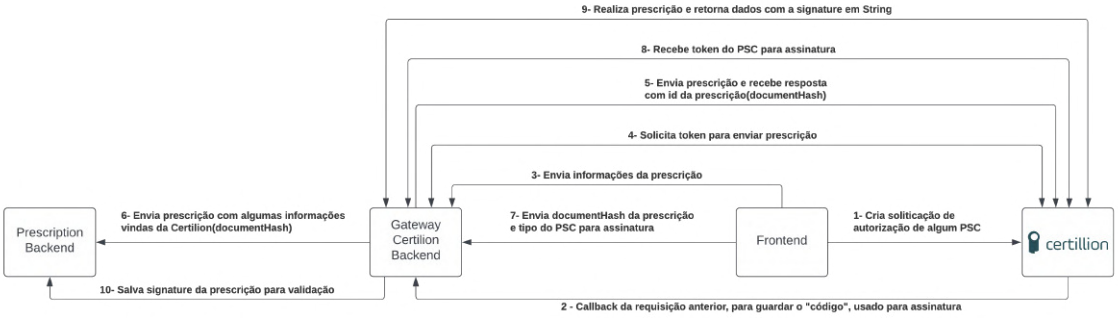

Figura 18. Fluxo de assinatura de prescrição

5. Ameaças à validade

Infelizmente, não foram feitos testes de usabilidade, nem a solução foi testada com usuários, farmácias e médicos reais, em um ambiente real. Foram feitos apenas testes de integração.

6. Limitações

As limitações deste trabalho foram o curto espaço de tempo, juntamente com algumas escolhas, para desenvolvimento, pois o sistema completo requer um tempo maior. Escolhas feitas como o uso de *Clean Architecture* e GraphQL tiveram um grande impacto.

O *Clean Architecture*, como dito, tem a finalidade de fazer com que o projeto seja testável e independente de qualquer tecnologia, seja de uma API, ou conexão de banco de dados por exemplo. Mas, é necessário criar muitos arquivos, desdes classes, interfaces, separadas em pacotes específicos, e isso em um desenvolvimento à curto prazo, acaba dificultando muito o desenvolvedor e atrasando o projeto. O *frontend* do sistema acabou deixando de lado essa arquitetura para usar uma arquitetura de camada simples, já o *backend*, continuou usando o *Clean Architecture*, mas, tendo que desrespeitar algumas "regras"dessa arquitetura para ter seu funcionamento.

GraphQL, é uma linguagem de API fácil, rápida e intuitíva para o desenvolvedor, mas, para serviços mais específicos como um upload de arquivos, ou uma busca mais minuciosa com filtragem, acaba perdendo para o REST. Além disso, entra a comunicação entre microservices, já que o *backend* do sistema foi divido assim. O GraphQL também não se dá com esse tipo de aplicação, já que não existe alguma biblioteca específica ou framework específico para isso, e isso, no mundo de aplicações Java e Spring Boot.

7. Conclusão

Este trabalho teve como principal objetivo desenvolver um *marketplace* de medicamentos com prescrições, juntamente com uma *dashboard* de controle de vendas para as farmácias e um app de assinatura de prescrições para os médicos. Os objetivos foram atingidos, focando-se inicialmente nas funcionalidades diferenciais do sistema.

Com base nos resultados apresentados neste artigo científico, podemos concluir que o projeto Facilifarma é uma solução alternativa e promissora para a área de vendas e administração de estoques de farmácias. Através da integração do aplicativo e da *dashboard*, o projeto apresenta uma interface intuitiva e completa tanto para os usuários quanto para as farmácias, melhorando a experiência de compra de medicamentos e facilitando a administração de estoques.

Esse projeto pode servir como uma base para futuros desenvolvimentos e projetos na área de saúde e farmacêutica, fornecendo *insights* e soluções para os desafios enfrentados no setor. Esperamos que essa solução ajude a melhorar a qualidade de vida dos pacientes, simplificando o processo de compra de medicamentos e fortalecendo a relação entre médicos, pacientes e farmácias.

Além disso, uma das funcionalidades mais inovadoras do projeto Facilifarma é o sistema de prescrições médicas, que permite aos usuários, e especialmente aos médicos, criar, assinar e enviar prescrições diretamente pelo aplicativo. Essa funcionalidade pode melhorar significativamente a eficiência e segurança do processo de prescrição, eliminando erros manuais e reduzindo a necessidade de visitas presenciais ao médico para obtenção de receitas médicas.

Esse sistema de prescrições também representa uma importante contribuição para a área de saúde e pode abrir caminhos para novas tecnologias e soluções na área médica e farmacêutica. Combinado com a interface intuitiva do aplicativo e da *dashboard*, essa funcionalidade reforça o compromisso do projeto Facilifarma em fornecer soluções de alta qualidade para a área de saúde, visando sempre o bem-estar dos pacientes.

7.1. Sugestões para trabalhos futuros

Uma sugestão para trabalhos futuros seria a implementação de uma funcionalidade de inteligência artificial para fornecer recomendações personalizadas aos usuários com base em seu histórico de compras e necessidades médicas. Além disso, seria interessante explorar a possibilidade de integração com prontuários eletrônicos de hospitais e clínicas, permitindo que médicos possam visualizar o histórico médico de seus pacientes diretamente no aplicativo e fornecer prescrições mais precisas e adequadas. Isso certamente contribuiria para a melhoria da qualidade do atendimento médico prestado aos pacientes.

Existem inúmeras outras sugestões específicas e mais acessíveis para melhoria deste trabalho. Dentre elas, podemos citar:

- Evolução da interface do App de médico e marketplace;
- Integração com a alguma API Stripe de pagamentos;
- Evolução da arquitetura para uma mais simples no *backend*;
- Adicioncar mais integrações entre os microserviços, para validação de dados;
- Criação de um sistema baseado em *blockchain* para melhor gerenciamento de prescrições médicas digitais;
- Adição de um sistema de *delivery*;
- e Sistema de avaliações e de mensagens (que já está com uma prévia da interface gráfica concluída, mas que necessita de uma lógica funcional).

Referências

Apollo GraphQL (2023). Apollo Client. Disponível em: https://www.apollographql.com/docs/react/. Acesso em: 03 maio 2023.

Beck, K. (2003). *Test-Driven Development: A Practical Guide*. Addison-Wesley Professional.

Becker, L. (2021). O que é React Native? Orgânica. Disponível em: https://www.organicadigital.com/blog/o-que-e-react-native/. Acesso em: 20 dez 2022.

Chroma Software (2023). Storybook. Disponível em: https://storybook.js.org/. Acesso em: 03 maio 2023.

Discord Inc. (2023). Discord. Disponível em: https://discord.com/. Acesso em: 04 maio 2023.

Docker Inc. (2023). Docker. Disponível em: https://www.docker.com/. Acesso em: 03 maio 2023.

Droga Raia (2022). Droga Raia®. Disponível em: https://www.drogaraia.com.br/. Acesso em: 2 dez 2022.

Drogarias Pacheco (2022). Drogarias Pacheco®. Disponível em: https://www.drogariaspacheco.com.br/. Acesso em: 3 dez 2022.

Drogasil (2021). Drogasil®. Disponível em: https://www.drogasil.com.br/. Acesso em: 2 dez 2022.

Expo Inc. (2023). Expo. Disponível em: https://expo.dev/. Acesso em: 03 maio 2023.

Fernandes, M. (2018). Teste de Unidade e Teste de Integração: O que são? Medium .Disponível em: https://medium.com/@mateus1198/teste-de-unidade-e-teste-de-integra%C3%A7%C3%A3o-o-que-s%C3%A3o-de58d7a3d3d2. Acesso em: 01 maio 2023.

Frost, B. (2016). *Atomic Design*. Atomic Design, Pittsburgh, PA, 1st edition.

Google (2023). Google Meet. Disponível em: https://meet.google.com/. Acesso em: 04 maio 2023.

Gradle (2023). Gradle Build Tool. Disponível em: https://gradle.org/. Acesso em: 01 maio 2023.

H2 Database (2023). H2 Database Engine. Disponível em: https://www.h2database.com/html/main.html. Acesso em: 01 maio 2023.

Linus Torvalds, J. H. (2023). Git. Disponível em: https://git-scm.com/. Acesso em: 03 maio 2023.

Lombok, P. (2023). Lombok. Disponível em: https://projectlombok.org/. Acesso em: 01 maio 2023.

Lopes, L. (2023). Monolíticos x Microsserviços: uma abordagem técnica. Medium .Disponível em: https://medium.com/@vinciusdichter/monol%C3%ADticos-x-microsservi%C3%A7os-uma-abordagem-t%C3%A9cnica-ef376e5f8204. Acesso em: 13 jun 2023.

Martin, R. C. (2017). Clean architecture: A craftsman's guide to software structure and design. *Prentice Hall Professional*.

Martins, R. Lucidchart: crie fluxogramas e wireframes no Google Docs. QiNetwork. Disponível em: https://blog.qinetwork.com.br/lucidchart/. Acesso em: 01 maio 2023.

MDN (2022). JavaScript. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 20 dez 2022.

Memed (2021). Memed®. Disponível em: https://memed.com.br/. Acesso em: 1 dez 2022.

Meta Platforms (2023). Jest.js. Disponível em: https://mobx.js.org/README.html. Acesso em: 03 maio 2023.

Microsoft®(2023). TypeScript. Disponível em: https://www.typescriptlang.org/. Acesso em: 03 maio 2023.

MobX Inc. (2023). MobX. Disponível em: https://mobx.js.org/README.html. Acesso em: 03 maio 2023.

Native Base Inc. (2023). Native Base. Disponível em: https://vitejs.dev/https://nativebase.io/. Acesso em: 03 maio 2023.

Notion Labs Inc. (2023). Notion. Disponível em: https://www.notion.so/product. Acesso em: 04 maio 2023.

Oliveira, K. A. (2006). PostgreSQL x MySQL. Qual Escolher? Mozilla. Disponível em: https://www.devmedia.com.br/postgresql-x-mysql-qual-escolher/3923. Acesso em: 20 dez 2022.

Oliveira, M. (2022). KEYCLOAK, o que você deveria saber sobre o assunto. Verity .Disponível em: https://www.verity.com.br/post/keycloak-o-que-voc%C3%AA-deveria-saber-sobre-o-assunto. Acesso em: 20 dez 2022.

Oracle. O que é tecnologias Java e por que preciso dela? Disponível em: https://www.java.com/pt-BR/download/help/whatis_java.html. Acesso em: 20 dez 2022.

Pague Menos (2021). Pague Menos®. Disponível em: https://www.paguemenos.com.br/. Acesso em: 1 dez 2022.

QueroDelivery (2021). QueroDelivery®. Disponível em: https://querodelivery.com/. Acesso em: 1 dez 2022.

Sakurai, R. (2014). *TDD: Desenvolvimento Guiado por Testes*. Novatec Editora.

Souza, I. d. (2020). O que é Yarn e como funciona seu gerenciamento de pacotes. Rockcontent. Disponível em: https://rockcontent.com/br/blog/yarn/. Acesso em: 20 dez 2022.

Spring® by VMware Tanzu (2023a). Spring Boot. Disponível em: https://spring.io/projects/spring-cloud. Acesso em: 01 maio 2023.

Spring® by VMware Tanzu (2023b). Spring Cloud. Disponível em: https://spring.io/projects/spring-boot. Acesso em: 01 maio 2023.

Spring® by VMware Tanzu (2023c). Spring Data. Disponível em: https://spring.io/projects/spring-data. Acesso em: 01 maio 2023.

Spring® by VMware Tanzu (2023d). Spring for GraphQL. Disponível em: https://spring.io/projects/spring-graphql. Acesso em: 01 maio 2023.

Spring® by VMware Tanzu (2023e). Spring Framework. Disponível em: https://spring.io/projects/spring-framework. Acesso em: 01 maio 2023.

Spring® by VMware Tanzu (2023f). Spring Security. Disponível em: https://spring.io/projects/spring-security. Acesso em: 01 maio 2023.

Stripe Inc. (2022). Stripe®. Disponível em: https://stripe.com/. Acesso em: 1 dez 2022.

The GraphQL Foundation (2023). GraphQL. Disponível em: https://graphql.org/. Acesso em: 01 maio 2023.

The JUnit Team (2023). JUnit 5. Disponível em: https://junit.org/junit5/. Acesso em: 01 maio 2023.

Tom Preston-Werner, Chris Wanstrath, S. C. P. J. H. M. (2023). Github. Disponível em: https://github.com/. Acesso em: 03 maio 2023.

Twilio (2023). Twilio. Disponível em: https://www.twilio.com/en-us. Acesso em: 01 maio 2023.

Ultrafarma (2022). Ultrafarma®. Disponível em: https://www.ultrafarma.com.br/. Acesso em: 3 dez 2022.

You, E. (2023a). Vite.js. Disponível em: https://vitejs.dev/. Acesso em: 03 maio 2023.

You, E. (2023b). Vue.js. Disponível em: https://vuejs.org/. Acesso em: 03 maio 2023.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Campina Grande
R. Tranqüilino Coelho Lemos, 671, Dinamérica, CEP 58432-300, Campina Grande (PB)
CNPJ: 10.783.898/0003-37 - Telefone: (83) 2102.6200

Documento Digitalizado Ostensivo (Público)

TCC(artigo) corrigido com ficha catalográfica

Assunto: TCC(artigo) corrigido com ficha catalográfica
Assinado por: Jose Henrique
Tipo do Documento: Anexo
Situação: Finalizado
Nível de Acesso: Ostensivo (Público)
Tipo do Conferência: Cópia Simples

Documento assinado eletronicamente por:

- **José Henrique Azevedo de Brito**, **ALUNO (201821250018) DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO - CAMPINA GRANDE**, em 02/07/2023 17:04:47.

Este documento foi armazenado no SUAP em 02/07/2023. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 867205
Código de Autenticação: 5f6ebe1eec