PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E ARTES
ENGENHARIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE APLICATIVO MÓVEL - (TASTEQUICKLY)

WATYSON GUIMARÃES SOARES

GOIÂNIA
2023

WATYSON GUIMARÃES SOARES

DESENVOLVIMENTO DE APLICATIVO MÓVEL - (TASTEQUICKLY)

Trabalho de Conclusão de Curso apresentado à Escola de Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para obtenção do título de Bacharel em Engenharia da Computação.

Orientador:

Prof. Me. Eugênio Júlio Messala Cândido Carvalho

Banca examinadora:

Prof. Me. Olegário Correa da Silva Neto

Prof. Dr. Leonardo Guerra de Rezende Guedes

GOIÂNIA
2023

WATYSON GUIMARÃES SOARES

DESENVOLVIMENTO DE APLICATIVO MÓVEL - (TASTEQUICKLY)

Este trabalho de Conclusão de Curso julgado adequado para obtenção do título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, em ____/____/_____.

_____
Prof. Coordenador de TCC
Coordenador(a) de Trabalho de Conclusão de Curso

Banca Examinadora:

_____
Orientador: Prof. Me. Eugênio Júlio Messala Cândido Carvalho

_____
Prof. Me. Olegário Correa da Silva Neto

_____
Prof. Dr. Leonardo Guerra de Rezende Guedes

GOIÂNIA
2023

AGRADECIMENTOS

Agradeço a todos que me apoiaram durante a realização desse trabalho acadêmico. Sem o apoio e encorajamento de vocês, eu não teria conseguido chegar até aqui. Gostaria de expressar minha gratidão a:

Deus por ter me dado saúde e força para superar as dificuldades.

Meu orientador acadêmico, professor Me. Eugênio Júlio Messala Cândido Carvalho, pela orientação e auxílio ao longo de todo o processo.

Meus pais, que me deram o apoio necessário para a conclusão de mais esse ciclo. Sou grato pelo amor, suporte e incentivo incondicionais que recebi de vocês. Vocês sempre estiveram ao meu lado, me encorajando e acreditando em mim.

Ao Professor Alexandre Ribeiro, pelo auxílio no início do curso e por me encorajar a continuar, quando estava doente e prestes a desistir.

Amigos e colegas de turma que me acompanharam nessa jornada, em especial Fernando Gomes de Souza Junior, Gabriel da Costa Diniz, Guilherme Henrique Mendonça Nascente, Gustavo Miranda, Marcio Antusa da Costa.

Aos professores e membros da banca, agradeço pelo tempo e dedicação em avaliar meu trabalho e contribuir com sugestões valiosas. Suas críticas construtivas e feedback foram fundamentais para o aprimoramento deste projeto.

Por fim, agradeço a mim mesmo pelos esforços, persistência e dedicação ao longo desse processo. Acreditar em si mesmo(a) é essencial para superar os desafios e alcançar os objetivos. Estou orgulhoso(a) do meu trabalho e dos resultados alcançados.

A todos vocês, meu sincero agradecimento. Vocês foram peças-chave na realização deste trabalho e contribuíram significativamente para o meu crescimento acadêmico e pessoal. Obrigado!

RESUMO

Este trabalho tem como objetivo criar e documentar um aplicativo de cardápio digital com serviço de delivery para um restaurante. Foi desenvolvido nas linguagens de programação Rust e Dart e utilizando as ferramentas Actix e Flutter. O aplicativo se baseou em três conceitos de arquitetura, Model-View-ViewModel (MVVM), cliente-servidor e camadas. A arquitetura MVVM foi aplicada nos clientes para separar a interface dos modelos utilizados. A arquitetura em camadas foi usada no servidor para separar e organizar as diferentes responsabilidades de cada parte. A arquitetura cliente-servidor foi adotada para permitir a comunicação entre a interface, seja ela o aplicativo móvel ou a web, e o servidor. A utilização dessas arquiteturas possibilitou uma estrutura mais organizada e modular o que facilitou o desenvolvimento do protótipo do aplicativo com cada componente tendo sua responsabilidade bem definida.

Palavras chave: Actix, Dart, Flutter, Rust, MVVM, Cliente-Servidor, Camadas.

ABSTRACT

This work aims to create and document a prototype of a digital menu application with a delivery service for a restaurant that was developed in the Rust and Dart programming languages and using the Actix and Flutter tools. The application was based on three architecture concepts, Model-View-ViewModel (MVVM), client-server and layers. The MVVM architecture was applied to the clients to separate the interface from the models used. The layered architecture was used on the server to separate and organize the different responsibilities of each part. The client-server architecture was adopted to allow communication between the interface, be it the mobile application or the web, and the server. The use of these architectures enabled a more organized and modular structure, which facilitated the development of the application prototype with each component having its responsibility well defined.

Keywords*: Actix, Dart, Flutter, Rust, MVVM, Client-Server, Layers.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

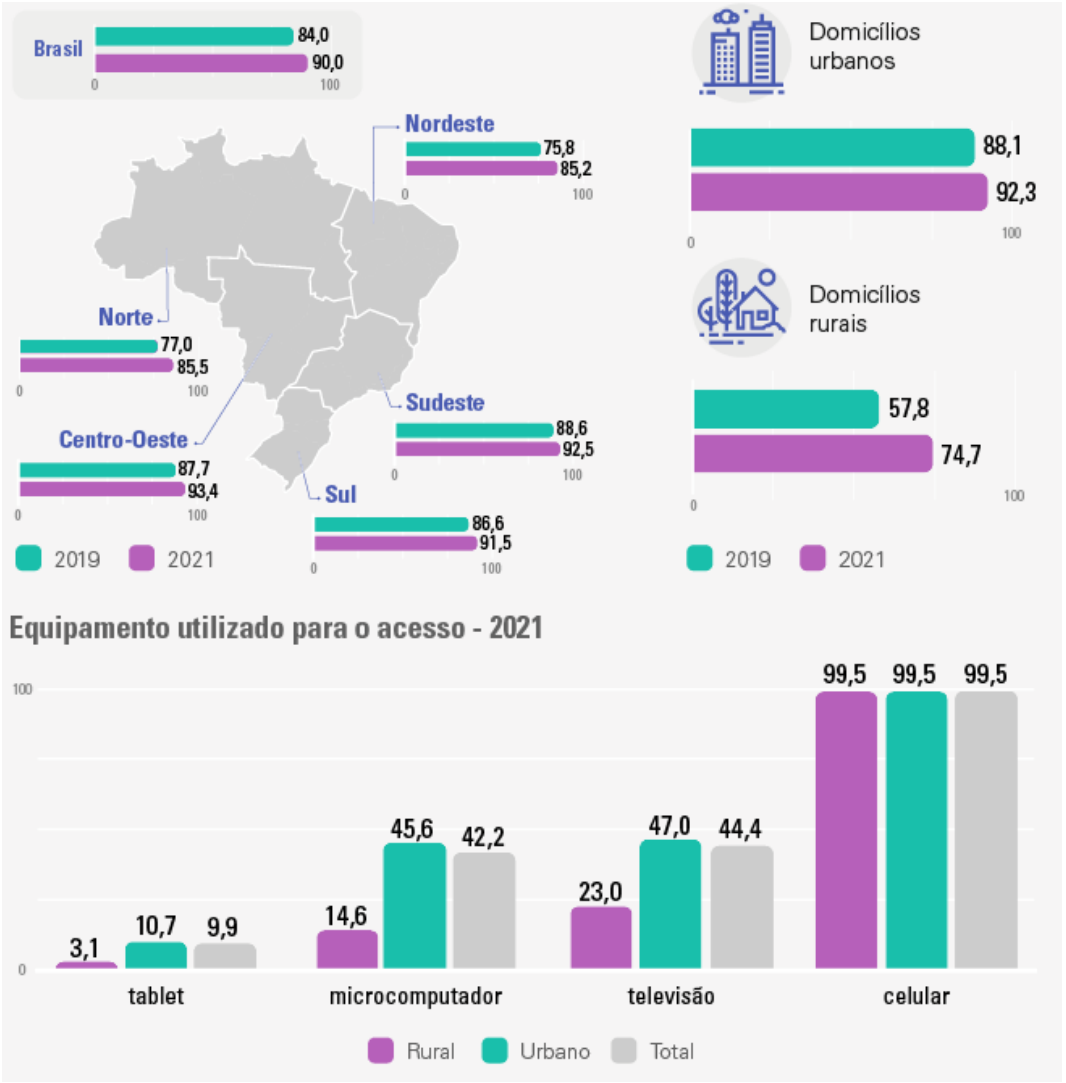
ADK	- *Android Development Kit* (Kit de desenvolvimento Android)
AOT	- *Ahead of time* (Antes do tempo)
API	- *Application Programming Interface* (Interface de Programação de Aplicativos)
GUI	- *Graphical User Interface* (Interface Gráfica do Usuário)
HTTP	- *Hyper Text Transfer Protocol* (Protocolo de Transferência de Hipertexto)
IDE	- *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
RAM	- *Random Access memory* (Memória de acesso aleatório)
REST	- *Representational State Transfer* (Transferência de Estado Representacional)
RF	- Requisito funcional
RN	- Regras de negócio
RNF	- Requisito não funcional
SDK	- *Software development kit* (Kit de desenvolvimento de software)
SGBD	- Sistemas de Gestão de Bases de Dados
VS Code	- Visual Studio Code
GPS	- *Global Positioning System* (Sistema de posicionamento global)

SUMÁRIO

1 INTRODUÇÃO

Em 2021, a internet alcançou 90% dos domicílios no país, com um aumento de 6 pontos percentuais em relação a 2019. O destaque vai para o telefone celular, que permaneceu como o principal dispositivo de acesso à internet em 99,5% dos domicílios. A internet era utilizada em 90% das casas do país em 2021, sendo que esse número chegou a 92% nas áreas urbanas, como podemos ver na ilustração 1. Esses números reforçam a importância da conectividade na vida das pessoas, especialmente nas cidades, onde a infraestrutura digital está mais consolidada. (BRITTO; NERY, 2022)

Ilustração 1 - Panorama do uso da Internet no país (%)



Fonte: Jéssica Cândido (2022).

A digitalização das empresas foi muito impulsionada pela Covid-19, o que transformou as compras online no Brasil. Um em cada quatro brasileiros pretende continuar realizando compras online mesmo após a pandemia. No Brasil, as pessoas estão satisfeitas com a experiência de compra online e a consideram fácil e conveniente. Os aplicativos de delivery estão sendo amplamente usados para alimentos e restaurantes, supermercados, farmácias, entretenimento e combustível. (PAYPAL BRASIL, 2021).

O cardápio digital é uma ferramenta para restaurantes que permite aos clientes realizarem pedidos por meio de dispositivos móveis. Existem diversos tipos de cardápios digitais, como os virtuais, aplicativos e QR Codes. A escolha desse tipo de ferramenta traz vantagens, como agilidade na escolha dos clientes, redução de erros nos pedidos, flexibilidade na atualização do cardápio e facilidade de acesso para os clientes. Em resumo, o cardápio digital melhora a experiência do cliente, a organização do restaurante e proporciona vantagem competitiva. (SCUADRA, 2023).

Diante do crescimento que os clientes têm nas compras online e sua crescente familiaridade com dispositivos móveis, despertou em mim o interesse por explorar aplicativos que facilitem a comunicação entre o cliente e o vendedor.

1.1 Objetivos Gerais

Este trabalho tem como objetivo estudar as linguagens de programação Rust e Dart nas ferramentas Actix e Flutter para o desenvolvimento de um aplicativo a fim de demonstrar o estudo realizado. O desenvolvimento do aplicativo envolve um software móvel para um restaurante, fornecendo um cardápio digital e o serviço de entregas delivery.

1.2 Objetivos Específicos

- Documentar as ferramentas estudadas;
- Documentar as necessidades do aplicativo;
- Modelagem de software: Diagrama de casos de uso;
- Desenvolvimento do *backend* do aplicativo usando o *Framework* Actix;
- Desenvolvimento do *frontend* mobile do aplicativo usando o *Framework* Flutter;
- Desenvolvimento do banco de dados usando o PostgreSQL;

- Análise do desenvolvimento.

1.3 Resultados Esperados

Com este trabalho, espera-se obter os seguintes resultados:

- Desenvolvimento de um protótipo de sistema utilizando as linguagens de programação Rust e Flutter.

1.4 Metodologia do Trabalho

A metodologia usada no desenvolvimento deste trabalho foi:

- Pesquisa bibliográfica das ferramentas, linguagens e frameworks utilizados.
- Desenvolvimento de uma aplicação.
- Descrição da aplicação

1.5 Organização do Trabalho

Este trabalho está divido em 6 partes, sendo elas:

1. Introdução ao trabalho: Este capítulo aborda os objetivos, resultados, metodologia e organização deste trabalho.
2. Ferramentas: Este capítulo apresenta o conjunto de ferramentas utilizadas durante o desenvolvimento do sistema.
3. Descrição global do aplicativo: Neste capítulo são apresentadas as interfaces do aplicativo, as funções do sistema e suas restrições.
4. Requisitos específicos do aplicativo: Este capítulo engloba os requisitos de software, descrição dos dados do sistema, diagrama de casos de uso e diagrama de domínio, além dos casos de uso descritivos.
5. Protótipo: Neste capítulo estão disponíveis: As imagens dos protótipos dos aplicativos mobile e web.
6. Considerações finais: Este capítulo contém as considerações finais do trabalho, bem como sugestões para trabalhos futuros.

2. FERRAMENTAS E TÉCNICAS

Este capítulo apresenta as ferramentas e tecnologias usadas no desenvolvimento do sistema.

2.1. Visual Studio Code (VSCode)

Desde o lançamento em 2015, o Visual Studio Code (VSCode), desenvolvido pela empresa Microsoft, consolidou-se como um dos recursos mais eficientes para o desenvolvimento de aplicativos no mercado. De acordo com a pesquisa *Stack Overflow Developer Survey* de 2021, o VSCode é o editor de código mais popular entre os desenvolvedores, sendo utilizado por mais de 70% dos entrevistados. (STACK OVERFLOW, 2022)

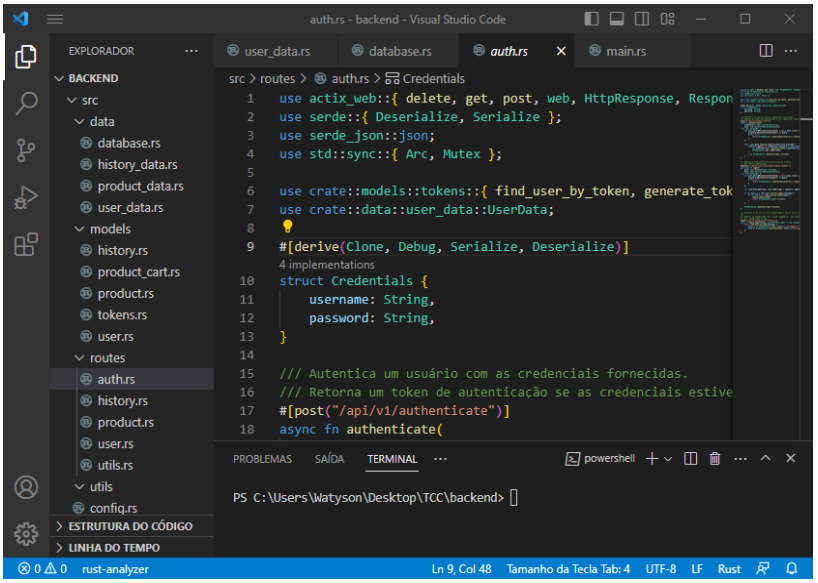
Entre suas principais vantagens está a capacidade de suportar diversas linguagens, desde as mais comuns até as menos conhecidas. Além disso, o VSCode oferece suporte a várias linguagens de programação e é altamente configurável, permitindo que os desenvolvedores personalizem a interface e incluam funcionalidades de acordo com suas preferências. Outra característica que o torna tão popular é sua integração com diversas ferramentas e extensões. Desde *linters* e *debuggers* até *plugins* para trabalhar com bancos de dados ou *frameworks* específicos, existe uma enorme quantidade de recursos para serem adicionados ao editor de código para aprimorar e otimizar a experiência de desenvolvimento. O VSCode está em constante evolução e se tornando cada vez mais poderoso e flexível, graças à sua equipe de desenvolvimento ativa e sua comunidade de usuários. Ele também é um software de código aberto que fomenta uma cultura colaborativa e incentiva o compartilhamento de ideias e soluções. Em resumo, o VSCode é um programa para editar código-fonte completo, flexível e altamente personalizável, que tem suporte a uma grande variedade de linguagens para programação e integração com diversas ferramentas e extensões. (MICROSOFT, 2022)

O principal fator de escolha desta ferramenta é o suporte a múltiplas linguagens, pois permite que apenas uma ferramenta tenha que ser utilizada para escrever os códigos. Junto a isso, ele é um *software* gratuito e suas extensões

também facilitam em muito a análise e correção do código nas linguagens utilizadas. A ilustração 2 mostra como é a tela de edição (Interface) do Visual Studio Code.

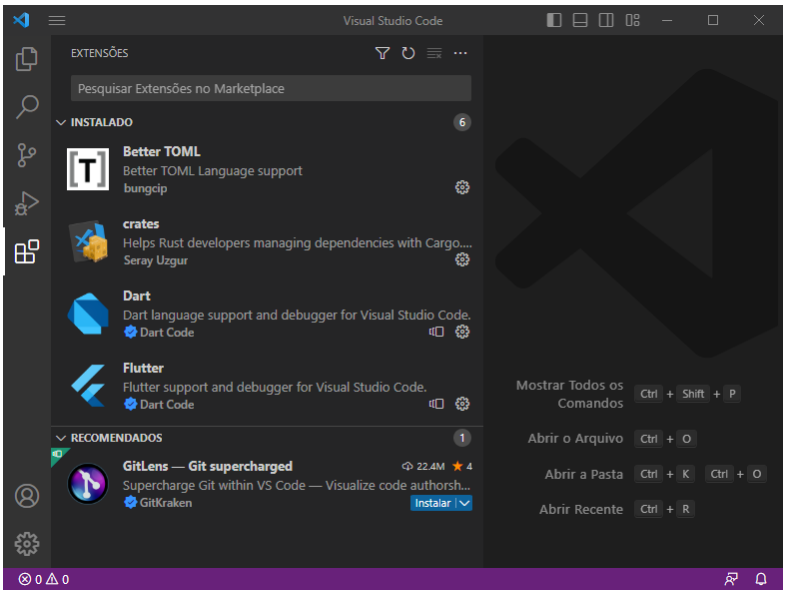
Na tela de extensões do VSCode, na ilustração 3, é possível visualizar as extensões instaladas quanto as recomendadas, localizadas no canto esquerdo da tela.

Ilustração 2 - Interface (VSCode)



Fonte: Autoria própria.

Ilustração 3 - Interface de extensões (VSCode)



Fonte: Autoria própria.

2.2. Android Studio

O *Android Studio* está entre as principais ferramentas para o desenvolvimento de aplicações móveis que usem o sistema operacional Android, desenvolvido e lançado em maio de 2013 pela Google. Sendo uma plataforma *Integrated Development Environment* (IDE), ele fornece um ambiente completo para o desenvolvimento de aplicativos, incluindo recursos avançados de depuração, ferramentas de construção e um emulador de dispositivos. Ele também possibilita que desenvolvedores criem aplicativos para dispositivos Android usando linguagens suportadas, como C++ e Flutter. (GOOGLE, 2022)

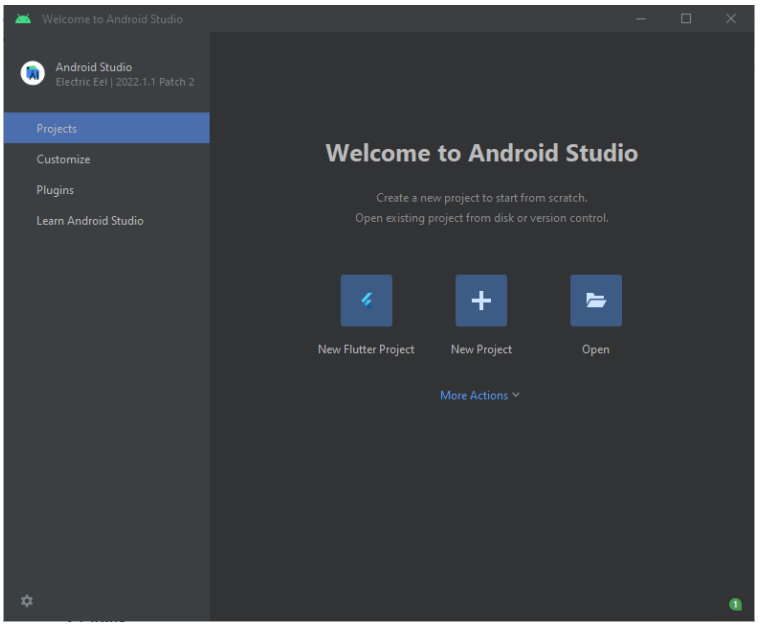
Um dos recursos mais úteis do *Android Studio* é seu emulador, que permite aos desenvolvedores testarem seus aplicativos em uma variedade de dispositivos Android virtuais, sem precisar de um *hardware* físico. O emulador incluído no Android Studio possui a capacidade de emular uma grande variedade de dispositivos Android, com diferentes resoluções, especificações de *hardware* e versões do Android. Ele é um componente precioso no processo de desenvolvimento para aplicativos para Android, pois permite que desenvolvedores testem seus aplicativos, além de simular condições de rede diferentes tem a capacidade garantir que os aplicativos funcionem corretamente em diferentes ambientes. (GOOGLE, 2022)

O Android Studio é amplamente utilizado no Brasil pelos desenvolvedores de aplicativos móveis para Android, em substituição a antiga IDE baseado em Eclipse e ao *Android Development Kit* (ADK) do Google. Uma das principais vantagens do *Android Studio* é o gerenciador de dependências Gradle, que oferece diversas opções ao desenvolvedor na hora de compilar. (ALLAN, 2016)

O principal fator de escolha do *Android Studio* para desenvolvimento de aplicativos para Android foi a possibilidade de integração de seu emulador com a ferramenta VSCode. Essa integração permite que o código escrito no VSCode possa ser testado no emulador com atualizações em tempo real bastando salvar o arquivo alterado, isso facilita o processo de desenvolvimento e garante que os aplicativos funcionem corretamente em diferentes configurações de *hardware* e *software*, além de ser um aplicativo gratuito. A ilustração 4 mostra a interface da ferramenta *Android*

Studio, nela podemos ver suas opções de *plugins* e projetos. A ilustração 5 mostra o emulador já configurado e em funcionamento.

Ilustração 4 - Interface (Android Studio)



Fonte: Autoria própria.

Ilustração 5 - Funcionamento do emulador (Android Studio)

Fonte: Autoria própria.

2.3. Insomnia REST

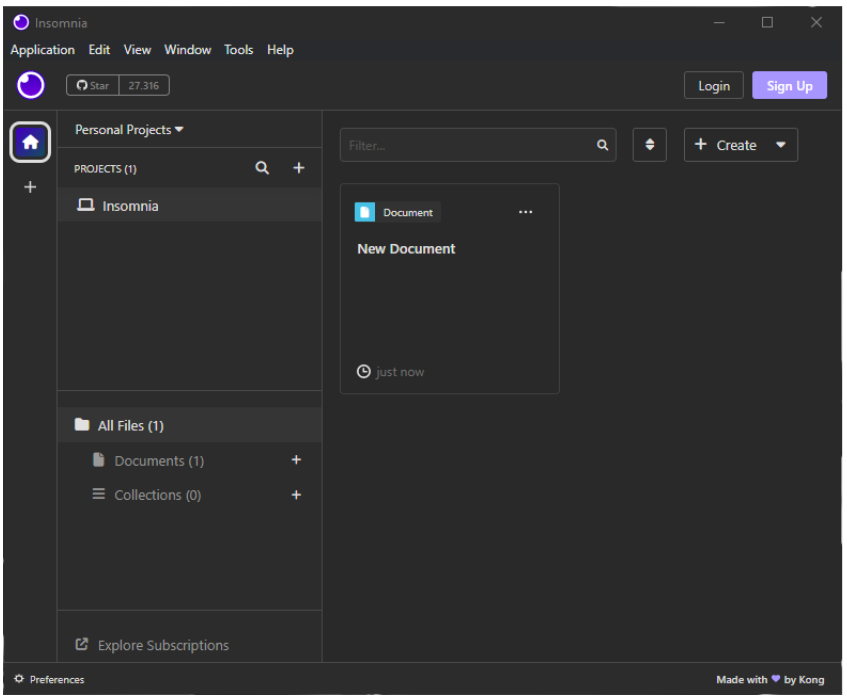
O Insomnia é um recurso útil para desenvolvedores de *software* que desejam testar e validar APIs de forma rápida e fácil. Desenvolvido pela Kong e lançado em 2015, o Insomnia é um cliente *Representational State Transfer* (REST) com código aberto e suporte para diversos sistemas operacionais. A interface de usuário do Insomnia é amigável e intuitiva, facilitando ainda mais os testes. Os recursos do Insomnia são sofisticados e completos, incluindo suporte para executar diversos tipos de solicitações permitindo armazenar, organizar e executar solicitações em espaços de trabalho e grupos. O construtor de parâmetros de string de consulta é outro recurso que merece destaque, pois permite criar parâmetros de consulta complexos com facilidade. É possível exportar e compartilhar espaços de trabalho, e a ferramenta também oferece suporte para solicitações encadeadas. Além disso, ele oferece recursos avançados, como auxiliares de segurança, criação de código e variáveis de ambiente. Esses recursos são muito úteis para o controle de cookies, variáveis de ambiente, geração de código e autenticação. (KANJILAL, 2022)

O teste de *Application Programming Interface* (API) tem muitos benefícios, incluindo teste inicial, independência de *Graphical User Interface* (GUI), independência de idioma, cobertura de teste aprimorada e lançamentos mais rápidos. O teste de API permite validar a lógica de negócios antes mesmo de o aplicativo ser totalmente criado, ajudando a encontrar mais bugs em menos tempo. Ele também permite testar a funcionalidade principal do aplicativo que está sendo construído sem a necessidade de utilizar sua interface gráfica. (KANJILAL, 2022)

O Insomnia foi escolhido devido a sua facilidade de uso e as facilidades que ele proporciona quanto aos testes, pois permite testar as funcionalidades da aplicação sem a necessidade de utilizar a interface gráfica e por também ser um *software* gratuito. Na ilustração 6, podemos ver a interface do aplicativo Insomnia, nela podemos notar a separação de projetos em arquivos que facilitam a organização para os testes.

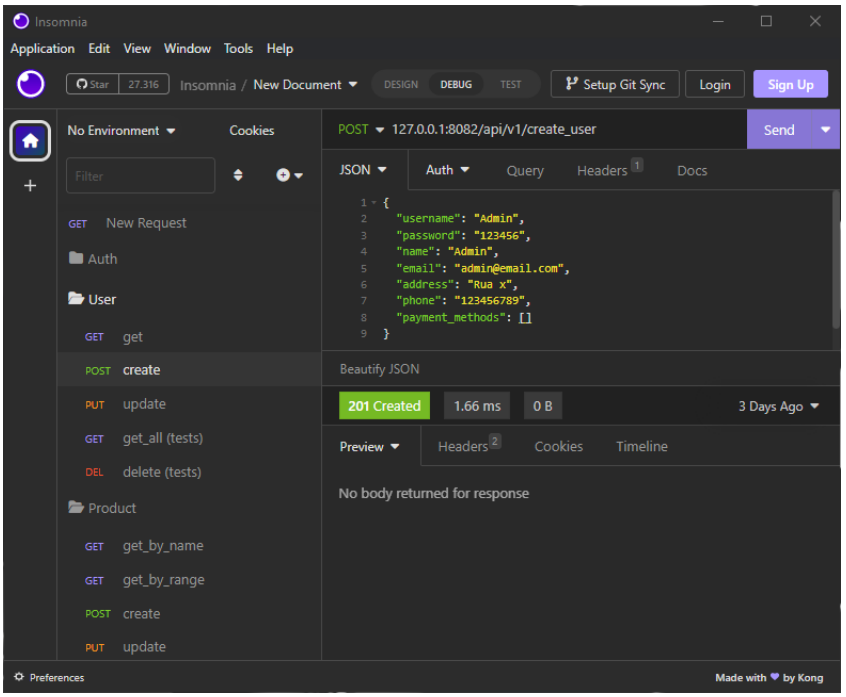
Através da ilustração 7, é possível visualizar alguns dos dados de teste utilizados pelo aplicativo, além da separação dos testes em grupos.

Ilustração 6 - Interface (Insomnia)



Fonte: Autoria própria.

Ilustração 7 - Interface de debug (Insomnia)



Fonte: Autoria própria.

2.4. PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados relacional com código aberto que foi criado em 1986 na Universidade da Califórnia em Berkeley. Com recursos avançados de integridade de dados, suporte a transações, recuperação de falhas e escalabilidade, o PostgreSQL é amplamente utilizado em aplicações de grande escala, sendo uma alternativa gratuita e robusta aos sistemas de banco de dados comerciais. Além disso, ele oferece uma ampla gama de tipos de dados, suporte para várias linguagens de programação populares e extensibilidade através de *plugins* e módulos personalizados. (POSTGRESQL, 2023)

Os Sistemas de Gestão de Bases de Dados (SGBD) são programas de computador essenciais para o armazenamento, organização e manipulação de dados em sistemas informáticos. Eles garantem a segurança, integridade, concorrência e recuperação de informações em um banco de informações. Para garantir a segurança, os SGBDs utilizam acesso autorizado, regras para usuários e operações, além de procedimentos de backup e restauração de dados. A integridade dos dados é mantida devido a verificações de restrições e gestão de transações. Eles também garantem a concorrência na atualização de dados em bases de dados multiusuário, utilizando subsistemas que trabalham juntos para garantir o bom funcionamento e manutenção do banco de dados. Os subsistemas incluem a definição de dados, manipulação de dados, administração de dados e geração de aplicações. Essas atividades são realizadas de maneira organizada e eficiente, garantindo a disponibilidade segura dos dados para consulta e atualização. (JULIANO, 2014)

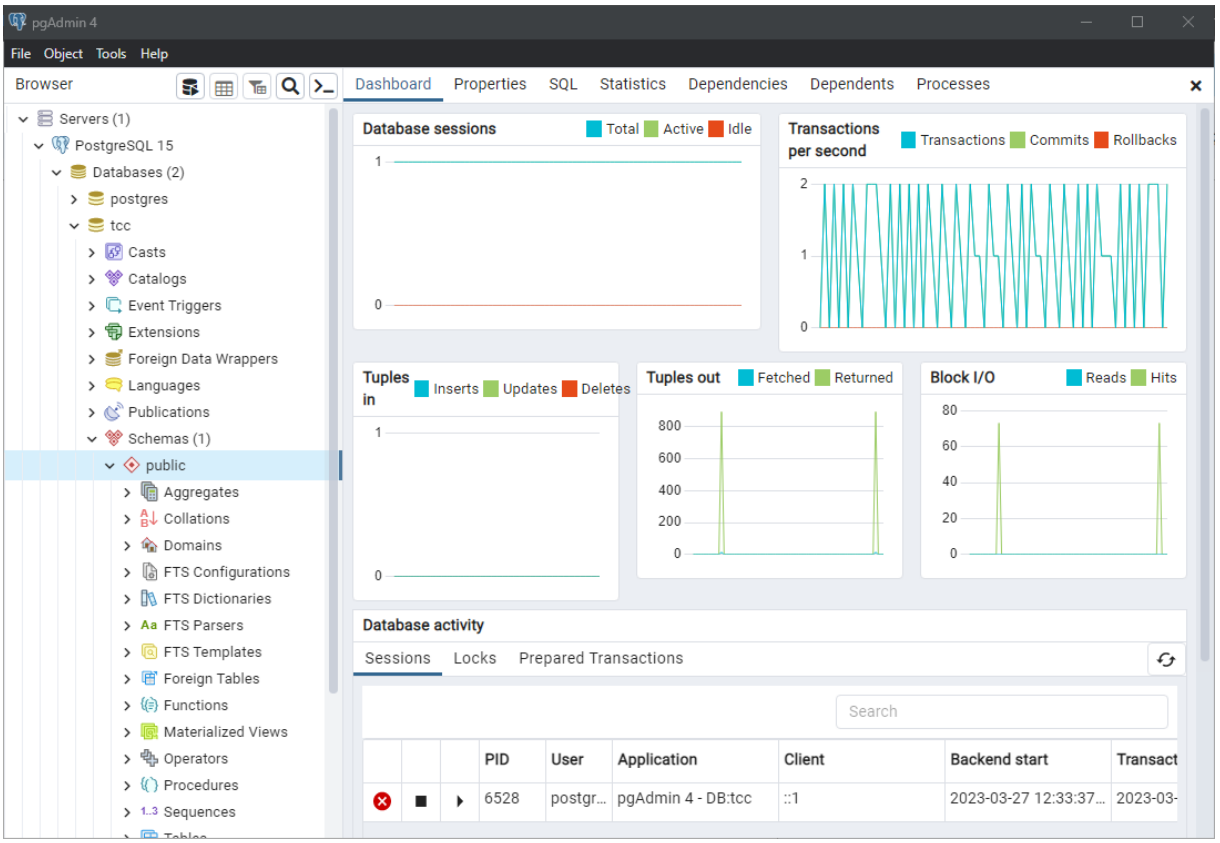
Modelagem Relacional é a representação de dados em forma de tabelas, onde cada tabela representa uma entidade e as linhas da tabela representam instâncias da entidade. Ele é baseado na teoria de conjuntos e álgebra relacional, onde o foco são os dados e não no armazenamento. O Modelo entidade relacionamento proposto por Peter P. Chen é baseado na teoria da Lei do Mundo, que afirma que o mundo está cheio de coisas que possuem características próprias e se relacionam entre si. As entidades são a representação genérica de um componente do mundo real que é significativo para a organização e sobre o qual

desejamos armazenar informações. As entidades podem ser coisas tangíveis, funções ou eventos/ocorrências. (SOUZA, 2023)

O PostgreSQL foi escolhido devido a sua licença gratuita e seu fácil aprendizado.

A ilustração 8 exibe a interface gráfica do programa, a qual permite visualizar o comportamento do banco de dados em específico, vendo suas transações, seções e outras informações. Além disso, é possível executá-lo usando sua interface de linhas de comando.

Ilustração 8 - Interface (PostgreSQL)



Fonte: Autoria própria.

2.5. Framework Actix

O Actix Web é um *micro-framework* desenvolvido em Rust. O objetivo do *framework* é criar serviços *Hyper Text Transfer Protocol* (HTTP) com facilidade sem exigir que seja necessário um servidor adicional, tornando-o uma opção viável para criação de serviços prontos para produção. O Actix Web tem a capacidade de expor um servidor contido em um executável nativo e suporta diferentes versões do protocolo HTTP, permitindo criar serviços seguros e de alto desempenho para diversos tipos de aplicativos. Embora o uso de atores ainda seja necessário para *endpoints* e *WebSocket*, a utilidade geral do ator está diminuindo à medida que o ecossistema de futuros e *async/await* amadurece, o que significa que o Actix Web está evoluindo para se tornar uma ferramenta mais geral e útil para criar serviços HTTP em Rust. (ACTIX, 2023)

A arquitetura do Actix é baseada em atores e mensagens, um paradigma de programação que torna o processamento de informações independente e escalável. Essa característica faz com que o Actix seja muito utilizado para aplicativos de rede e sistemas distribuídos. Nele cada conexão de *socket* é tratada como um ator, o que permite que o sistema seja altamente escalável e eficiente em lidar com múltiplas conexões simultâneas. A arquitetura de atores do Actix é uma abordagem moderna e eficaz para lidar com problemas de concorrência e escalabilidade em sistemas distribuídos. (OLEINIK, 2020)

O Rust é uma linguagem de programação conhecida por sua concorrência sem disputa de dados, permitindo maior segurança e baixo *overhead* na memória, ou seja um alto consumo de memória de forma desnecessária. O compilador dele é capaz de gerar um código nativo inteligente de desalocação, evitando problemas como *memory leaks*, ou seja partes da memória que não são deletadas após o uso, e *overheads* de processamento, que seriam usos desnecessários do processador. Com essas vantagens, o Rust se torna uma linguagem eficiente e segura para programação concorrente, oferecendo uma alternativa para programação de sistemas com múltiplos threads. (JÚNIOR, 2019)

O Rust foi projetado para ser rápido e eficiente, sem a necessidade de coletor de lixo. Ela tem a capacidade de alimentar serviços de desempenho crítico e pode ser integrada a outras linguagens. O controle de memória único do Rust, que utiliza

o conceito de "propriedade" de memória, permite que a linguagem rastreie quem pode ler e gravar na memória, liberando-a imediatamente quando não é mais necessária e tornando virtualmente impossível ter bugs de memória em tempo de execução. Nele, o compilador cuida do gerenciamento de memória, isso significa que você não precisa controlá-lo manualmente. Isso torna o desenvolvimento de aplicativos mais seguro e eficiente. (HOWARTH, 2020)

O *framework* Actix foi escolhido por priorizar o desempenho e simplificar a escrita quando comparado a outros e por utilizar a linguagem Rust que prioriza a segurança e o desempenho e é capaz de realizar um controle da memória sem o uso do coletor de lixo. A principal desvantagem do Actix é a curva de aprendizado.

Na ilustração 9a, há um código mínimo para uma aplicação web usando o framework Actix-Web. Ele define uma rota "/" que responde ao método GET com a mensagem "Hello, world!". Em seguida, configura um servidor local que executa a aplicação.

Ilustração 9a - Código de aplicação web mínima escrita em rust (Actix-Web).

```rust
use actix_web::{get, App, HttpResponse, HttpServer, Responder};

#[get("/")]
async fn index() -> impl Responder {
    HttpResponse::Ok().body("Hello, world!")
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {App::new().service(index)})
    .bind("127.0.0.1:8080")?
    .run()
    .await
}
```

Fonte: Autoria própria.

A ilustração 9b apresenta um código que conecta o banco de dados PostgreSQL ao Actix-Web. Ele verifica as credenciais do usuário, autentica o login e retorna um token em JSON. A função *connect* estabelece a conexão com o banco de dados. A função *authenticate* verifica as credenciais no banco de dados. O *handler login* recebe as credenciais, autentica e retorna um *token* JSON. A função *main* configura e inicia o servidor HTTP.

Ilustração 9b - Código de conexão com o POSTGRES (Actix-Web).

```
// Dependências no arquivo Cargo.toml
[dependencies]
actix-web = "4.3.1"
tokio = { version = "1.26.0", features = ["full"] }
tokio-postgres = "0.7.8"

// Arquivo main.rs
use actix_web::{get, web, App, HttpResponse, HttpServer, Responder};
use tokio_postgres::{Client, NoTls};

struct Credentials {
    username: String,
    password: String,
}

async fn connect() -> Result<Client, tokio_postgres::Error> {
    let config = "postgres://NOMEUSUARIO:SENHA@HOST:PORTA/NOMEBANCO";
    let (client, connection) = tokio_postgres::connect(&config, NoTls).await?;
    tokio::spawn(async move {
        if let Err(e) = connection.await {
            eprintln!("connection error: {}", e);
        }
    });
    Ok(client)
}

async fn authenticate(credentials: Credentials) -> Result<i32, String> {
    let client = connect().await.map_err(|e| format!("Failed to connect to
database: {}", e))?;
    let query = "SELECT id FROM tbl_user WHERE username = $1 AND password = $2";
    let row = client.query_opt(query, &[&credentials.username,
&credentials.password]).await
        .map_err(|e| format!("Failed to execute query: {}", e))?;
    row.map(|row| row.get(0)).ok_or_else(|| "Invalid username or
password".to_string())
}

#[post("/authenticate/login")]
async fn login(credentials: web::Json<Credentials>) -> impl Responder {
    match authenticate(credentials.into_inner()).await {
        Ok(id) => HttpResponse::Ok().json(LoginResponse::new(id,
generate_token(id).unwrap())),
        Err(_) => HttpResponse::Unauthorized().finish(),
    }
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| App::new().service(login))
        .bind("127.0.0.1:8080")?
        .run()
        .await
}
```

Fonte: Autoria própria.

2.6. Framework Flutter

Flutter é um *Software development kit* (SDK) de código aberto desenvolvido pelo Google, foi lançado em 2015 para criar aplicativos rapidamente para iOS e Android. Ele usa a linguagem de programação Dart, que é fácil de aprender se você já estiver familiarizado com Java, JS ou C#. Diferente de outras tecnologias, como o React Native, o Flutter desenha a interface do usuário do zero por meio de uma biblioteca de gráficos C++ 2D chamada Skia. Ele também é baseado em uma arquitetura de fluxo de dados unidirecional, tornando a programação reativa além de ser extensível com plugins de terceiros que adicionam novos componentes de interface do usuário personalizados ou envolvem recursos específicos da plataforma ainda não cobertos pelas classes integradas. (BELLINASO, 2018)

O Flutter tem se tornado cada vez mais popular entre os desenvolvedores de aplicativos móveis devido às suas vantagens em relação a outros *frameworks*. Uma das principais vantagens é a utilização da compilação *ahead of time* (AOT), o que permite que o Flutter compile todo o código antes da execução, melhorando significativamente o desempenho e a fluidez das aplicações. Além disso, a linguagem de programação Dart, utilizada pelo Flutter, oferece muitos recursos modernos para facilitar o desenvolvimento e a manutenção de aplicativos móveis. Ele usa seus próprios *Widgets* para criar interfaces de usuário e eles oferecem temas para o *Material Design* e o Cupertino, permitindo que os desenvolvedores personalizem a aparência de suas aplicações de acordo com suas preferências. Além disso, como os *Widgets* também são responsáveis pelo *layout* da aplicação, eles podem ser invisíveis, proporcionando uma experiência de usuário mais agradável. Em resumo, o Flutter é uma excelente opção para desenvolvedores de aplicativos móveis que procuram uma solução moderna, rápida e eficiente. Com suas características inovadoras e recursos avançados, o Flutter pode ajudar os desenvolvedores a criar aplicativos móveis de alta qualidade, personalizados e com excelente desempenho. (CAVALCANTI, 2018)

O Flutter foi escolhido por ser multiplataforma, no caso, por ser capaz de produzir um mesmo aplicativo tanto para web quanto *mobile* sem a necessidade de refazer todo o código. Além de ter *hot reload*, que permite ao desenvolvedor visualizar toda alteração realizada quase que instantaneamente. Sua principal

desvantagem é o custo de memória, o flutter acaba por consumir muita memória com coisas simples.

A ilustração 10 é um exemplo de código mínimo de um aplicativo Flutter que exibe a mensagem "Olá, mundo!" em uma tela. Ele define a classe MyApp como um *widget* de estado imutável que retorna um *widget* MaterialApp. O aplicativo é iniciado com o método runApp() e utiliza o *widget Scaffold* para definir a estrutura básica do layout, contendo um AppBar e um *widget Center* com um *Text* que exibe a mensagem *"Hello, world!".*

Ilustração 10 - Código de aplicação mínima escrita em dart (Flutter).

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hello, world!',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello, world!'),
        ),
        body: Center(
          child: Text('Hello, world!'),
        ),
      ),
    );
  }
}
```

Fonte: Autoria própria.

2.7 Arquiteturas de software

Este subcapítulo, apresenta as arquiteturas que serão utilizadas na construção do protótipo do sistema.

2.7.1 Arquitetura Cliente-Servidor

A comunicação cliente-servidor é essencial para o funcionamento de diversos sistemas que são utilizados. Ela funciona através do protocolo HTTP, onde o cliente especifica o tipo de ação que deseja realizar e envia uma requisição para servidor, essa requisição é processada e então respondida. (MENDES, 2021)

De acordo com Caio Mendes (2021), os principais métodos de requisição que permitem que o cliente especifique o tipo de ação que deseja realizar estão descritos na tabela 1.

Tabela 1 - Métodos de requisição

Métodos	**Responsabilidade**
GET	Retorna um dado do servidor.
POST	Alterar um dado no servidor.
PUT	Envia um dado para o servidor.
DELETE	Deleta um dado no servidor.

Fonte: Caio Mendes (2021).

De acordo com Caio Mendes (2021), quando o servidor retorna uma resposta, ele o faz com um *status*, esse status define o resultado da requisição realizada. Sendo os principais *status* descritos na tabela 2.

Tabela 2 - Códigos de status HTTP

Status	**Tipo de resposta**
1XX	Informação
2XX	Sucesso
3XX	Redirecionamento

4XX	Erro no cliente
5XX	Erro no servidor

Fonte: Caio Mendes (2021).

2.7.2 Arquitetura de Camadas

A arquitetura em camadas é um estilo de arquitetura que separa a aplicação em camadas lógicas. Essas camadas devem ser organizadas com uma hierarquia e são capazes de trabalhar juntas para fornecer uma funcionalidade completa, embora o número de camadas possa variar a topologia mais comum consiste em quatro camadas: apresentação, negócio, persistência e banco de dados. Cada camada possui uma função e uma responsabilidade específica dentro da arquitetura. Usando a topologia citada anteriormente a camada de apresentação deveria lidar com a interface do usuário, a camada de negócios executa operações de negócios específicos, a camada de persistência deve ser responsável por recuperar informações que foram salvas e a camada de banco de dados deve ser responsável por manter os dados salvos. Devem existir barreiras entre as camadas, portanto uma camada não deve saber como a outra funciona. As características arquiteturas da arquitetura em camadas valorizam a modularidade, reutilização e facilidade de manutenção porém deve-se tomar cuidado com a sobrecarga na comunicação entre camadas o que pode afetar negativamente a performance da aplicação, além disso a escalabilidade também pode se tornar um desafio já que adicionar ou remover camadas é difícil e às vezes impossível. (CASTIGLIONI, 2022)

2.7.3 Arquitetura Model-View-ViewModel (MVVM)

O MVVM (*Model-View-ViewModel*) é um padrão de arquitetura amplamente utilizado no desenvolvimento mobile. Ele separa a lógica de negócios da interface do usuário, com a camada View-Model coordenando as operações. O MVVM oferece benefícios como facilidade de aprendizado, suporte ao desenvolvimento mobile, simplificação de testes e facilidade na manutenção do código. É compatível com diversas linguagens e frameworks, incluindo Swift, Java, Dart (com Flutter) e frameworks JavaScript. (SILVA, 2022)

3 DESCRIÇÃO GLOBAL DO APLICATIVO

Este capítulo apresenta uma descrição global do aplicativo em relação às suas interfaces, funções e restrições.

O aplicativo foi desenvolvido utilizando o *framework* Actix para o desenvolvimento do servidor e o *framework* Flutter para o desenvolvimento dos clientes. Suas principais funções incluem fornecer um cardápio para restaurantes e auxiliar na gestão dos pedidos pelo administrador.

3.1 Interfaces

Esta seção apresenta uma descrição geral das interfaces do aplicativo. Sendo elas a interface do sistema (mobile e web), interface de *hardware* e interface de comunicação.

3.1.1 Interfaces do Sistema

Login: Permite ao usuário fazer *login* utilizando o nome de usuário e senha previamente criados na tela de cadastro. Deve incluir uma opção para recuperação de senha por email e um meio para cadastro de novos clientes.

Cadastro: O cadastro deve solicitar o nome de usuário, senha (com confirmação), nome, email, telefone, endereço. Por meio dela também deve ser possível retornar para a tela de login.

Recuperar Senha: Deve requisitar o email cadastrado para que seja possível enviar uma mensagem que possibilite o usuário a trocar a senha por uma nova, deve permitir retornar para a tela de *login*.

Cardápio: Deve conter uma barra de aplicativos que inclua um ícone para acessar os dados do usuário e um ícone para acessar o carrinho de compras. Deve permitir visualizar o cardápio, onde os produtos serão listados cada um contendo uma foto, o nome e o preço. Ao selecionar o produto o usuário deve ir para uma tela de confirmação de pedido.

Confirmação de pedido: Permite que o usuário visualize detalhes do pedido, como ingredientes e preço. O usuário deve escolher colocar alguma observação e a

quantidade desejada do produto, limitada a 10 por pedido. Caso seja confirmado o produto, o mesmo deve ser adicionado no carrinho para que o usuário finalize a compra. Deve permitir voltar para o cardápio.

Carrinho: Conter o endereço de entrega, o troco desejado e o preço total da compra, além de uma lista de todos os pedidos realizados com as seguintes informações: preço, quantidade e deve permitir excluir um determinado produto da lista. Oferece a opção de confirmar os pedidos e visualizar o histórico de pedidos realizados anteriormente. Deve permitir retornar ao cardápio.

Histórico: Contém uma lista com todos os pedidos realizados e seu status, se está em análise, preparação, a caminho, entregue e cancelado.

Perfil: Deve conter os seguintes dados do usuário: nome, email, telefone e endereço. Deve permitir a edição dos seguintes dados: email, endereço, telefone e senha. Deve permitir retornar para o cardápio.

Quanto às interfaces do aplicativo web que devem ser utilizadas pelo administrador do sistema, temos:

Login: Permite fazer *login* utilizando o nome de usuário e senha do admin.

Gerência: Uma tela que deve conter o que o administrador pode utilizar para gerenciar o cardápio, os pedidos e seu perfil de administrador.

Gerência do Cardápio: Deve permitir realizar as funções de criar, ler, atualizar e deletar, com relação aos produtos mostrados no cardápio, nas alterações deve ser capaz de permitir que um produto seja retirado do cardápio sem que o mesmo deva ser excluído.

Gerência de Pedidos: Deve conter uma lista de todos os pedidos realizados, esses pedidos devem aparecer de forma resumida apenas com o número, a data e o estado do pedido. Quando selecionados devem exibir todos os dados do pedido e permitirem alterar o estado do mesmo.

3.1.2 Interfaces do Usuário e Administrador

Quanto às interfaces do usuário, o sistema deve ser desenvolvido para ser utilizado em smartphones que possuam o sistema operacional Android, com a versão a partir de 13, e acesso à internet.

Quanto às interfaces do administrador, o sistema deve ser desenvolvido para ser utilizado em navegadores web.

3.1.3 Interface de Hardware

Requisitos mínimos mobile:

Processador: Quad Core (1.3GHz);

Random Access memory (RAM): 2 Gigabytes;

Armazenamento livre: 100 Megabytes.

Requisitos mínimos web:

Processador: Dual Core (1.3GHz);

RAM: 2 Gigabytes;

Armazenamento livre: 100 Megabytes.

3.1.4 Interface de Comunicação

O sistema vai usar o Postgres para realizar o armazenamento de dados dos produtos, encomendas, usuários.

3.2 Funções do Sistema

- Gerenciar perfil próprio;
- Ver Cardápio;
- Encomendar pedidos;
- Visualizar o histórico de pedidos;
- Gerenciar cardápio;
- Gerenciar pedidos;

3.3 Restrições

1. O aplicativo só pode ser acessado quando houver uma conexão com a internet.
2. Caso o usuário não esteja logado, ele verá a tela de login.
3. A tela de gerenciamento de pedidos só pode ser acessada e usada pelo administrador.
4. A tela de gerenciamento de cardápio só pode ser acessada e usada pelo administrador.
5. O aplicativo deve limitar a quantidade de cada pedido da encomenda em 10.

4 REQUISITOS ESPECÍFICOS

Este capítulo apresenta os requisitos do sistema, sendo eles requisitos funcionais (RF), requisitos não funcionais (RNF) e regras de negócio (RN), descrição de dados do sistema, diagrama de casos de uso, casos de uso descritivo e diagrama de domínio.

4.1 Requisitos Funcionais

Tabela 3 - Requisitos funcionais

Id.	Descrição
RF 01	O usuário deve ser capaz de realizar *login* usando um nome de usuário e senha válidos.
RF 02	O usuário deve ser capaz de recuperar uma senha perdida.
RF 03	O usuário deve ser capaz de criar seu meio de acesso (*login*) com os seguintes dados devidamente validados: nome de usuário, senha, nome, email, telefone e endereço.
RF 04	O usuário deve visualizar a tela de *login* ao acessar o aplicativo.
RF 05	O usuário deve ser capaz de selecionar seus pedidos e enviá-los ao carrinho.
RF 06	O usuário deve ser capaz de escolher a quantidade de pedidos que deseja.
RF 07	O usuário deve ser capaz de enviar observações sobre o pedido durante a compra.
RF 08	O usuário deve selecionar a quantia necessária de troco.
RF 09	O usuário deve ser capaz de visualizar seu histórico de pedidos.
RF 10	O usuário deve ser capaz de visualizar o *status* do pedido.
RF 11	O administrador deve ser o único capaz de gerenciar o cardápio e os pedidos.
RF 12	O administrador deve ser capaz de escolher se o produto deve ou não aparecer no cardápio.

RF 13	O usuário deve ser capaz de editar as seguintes informações de seu perfil: email, endereço, telefone e senha.
RF14	O administrador deve ser capaz de alterar o estado atual de entrega dos pedidos.
RF 15	O pedido deve conter os seguintes status: Em análise, Em preparação, A caminho, Entregue e Cancelado.

Fonte: Autoria própria.

Tabela 4 - RF 01: Acesso ao sistema

Identificador	**Nome**
RF 01	Acesso ao sistema

Caso de uso	**Autor**
CSU 03	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de realizar *login* usando um nome de usuário e senha válidos.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 5 - RF 02: Recuperar senha

Identificador	**Nome**
RF 02	Recuperar senha
Caso de uso	**Autor**
CSU 03	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de recuperar uma senha perdida.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 6 - RF 03: Cadastro usuário

Identificador	**Nome**
RF 03	Cadastro usuário
Caso de uso	**Autor**
CSU 03	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de criar seu meio de acesso (*login*) com os seguintes dados devidamente validados: nome de usuário, senha, nome, email, telefone e endereço.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 7 - RF 04: Visualizar *login* ao acessar aplicativo

Identificador	**Nome**
RF 04	Visualizar *login* ao acessar aplicativo
Caso de uso	**Autor**
CSU 03	Watyson Guimarães Soares
Descrição	
O usuário deve visualizar a tela de *login* ao acessar o aplicativo.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 8 - RF 05: Selecionar pedidos

Identificador	**Nome**
RF 05	Selecionar pedidos
Caso de uso	**Autor**
CSU 04	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de selecionar seus pedidos e enviá-los ao carrinho.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 9 - RF 06: Escolher quantidade de produtos

Identificador	**Nome**
RF 06	Escolher quantidade de produtos
Caso de uso	**Autor**
CSU 04	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de escolher a quantidade de pedidos que deseja.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 10 - RF 07: Enviar observações do pedido

Identificador	**Nome**
RF 07	Enviar observações do pedido
Caso de uso	**Autor**
CSU 04	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de enviar observações sobre o pedido durante a compra.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 11 - RF 08: Selecionar quantia de troco

Identificador	**Nome**
RF 08	Selecionar quantia de troco
Caso de uso	**Autor**
CSU 05	Watyson Guimarães Soares
Descrição	
O usuário deve selecionar a quantia necessária de troco.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 12 - RF 09: Visualizar histórico

Identificador	**Nome**
RF 09	Visualizar histórico
Caso de uso	**Autor**
CSU 06	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de visualizar seu histórico de pedidos.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 13 - RF 10: Visualizar *status* do pedido

Identificador	**Nome**
RF 10	Visualizar *status* do pedido
Caso de uso	**Autor**
CSU 06	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de visualizar o *status* do pedido.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 14 - RF 11: Gerência do cardápio e pedidos

Identificador	**Nome**
RF 11	Gerência do cardápio e pedidos
Caso de uso	**Autor**
CSU 01, CSU 02	Watyson Guimarães Soares
Descrição	
O administrador deve ser o único capaz de gerenciar o cardápio e os pedidos.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 15 - RF 12: Organizar cardápio

Identificador	**Nome**
RF 12	Organizar cardápio
Caso de uso	**Autor**
CSU 02	Watyson Guimarães Soares
Descrição	
O administrador deve ser capaz de escolher se o produto deve ou não aparecer no cardápio.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 16 - RF 13: Gerenciar próprio perfil

Identificador	**Nome**
RF 13	Gerenciar próprio perfil
Caso de uso	**Autor**
CSU 03	Watyson Guimarães Soares
Descrição	
O usuário deve ser capaz de editar as seguintes informações de seu perfil: email, endereço, telefone e senha.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 17 - RF 14: Alterar estado atual dos pedidos

Identificador	**Nome**
RF 14	Alterar estado atual dos pedidos
Caso de uso	**Autor**
CSU 01	Watyson Guimarães Soares
Descrição	
O administrador deve ser capaz de alterar o estado atual de entrega dos pedidos.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 18 - RF 15: Definição de status do pedido

Identificador	**Nome**
RF 15	Definição de *status* do pedido
Caso de uso	**Autor**
CSU 01, CSU 06	Watyson Guimarães Soares
Descrição	
O pedido deve conter os seguintes *status*: Em análise, Em preparação, A caminho, Entregue e Cancelado.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

4.2 Requisitos Não Funcionais

Tabela 19 - Requisitos não funcionais

Id.	Descrição
RNF 01	As informações de usuários são restritas e só podem ser acessadas/visualizadas pelo usuário.
RNF 02	Aplicativo deve ser intuitivo e responsivo.
RNF 03	O servidor deve ser escrito na linguagem RUST e usar o *framework* Actix.
RNF 04	Os aplicativos devem ser escritos na linguagem DART e usar o *framework* Flutter.
RNF 05	O servidor deve ser capaz de suportar mais de um usuário simultaneamente.

Fonte: Autoria própria.

Tabela 20 - RNF 01: Restrição de informações pessoais

Identificador	**Nome**
RNF 01	Restrição de informações pessoais

Caso de uso	**Autor**
CSU 03	Watyson Guimarães Soares
Descrição	
As informações de usuários são restritas e só podem ser acessadas/visualizadas pelo usuário.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 21 - RNF 02: Usabilidade do aplicativo

Identificador	**Nome**
RNF 02	Usabilidade do aplicativo
Caso de uso	**Autor**
	Watyson Guimarães Soares
Descrição	
Aplicativo deve ser intuitivo e responsivo.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 22 - RNF 03: Linguagem do servidor

Identificador	**Nome**
RNF 03	Linguagem do servidor
Caso de uso	**Autor**
	Watyson Guimarães Soares
Descrição	
O servidor deve ser escrito na linguagem RUST e usar o *framework* Actix.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

Tabela 23 - RNF 04: Linguagem do aplicativo

Identificador	**Nome**
RNF 04	Linguagem do aplicativo
Caso de uso	**Autor**
	Watyson Guimarães Soares
Descrição	
Os aplicativos devem ser escritos na linguagem DART e usar o *framework* Flutter.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria do próprio autor

Tabela 24 - RNF 05: Usuários simultaneamente

Identificador	**Nome**
RNF 05	Usuários simultaneamente
Caso de uso	**Autor**
	Watyson Guimarães Soares
Descrição	
O servidor deve ser capaz de suportar mais de um usuário simultaneamente.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

4.3 Regras de Negócio

Tabela 25 - Regras de negócio

Id.	Descrição
RN 01	O aplicativo deve limitar a quantidade de pedidos em uma única encomenda em 10.

Fonte: Autoria do próprio autor

Tabela 26 - RN 01: Limite por encomenda

Identificador	**Nome**
RN 01	Limite por encomenda

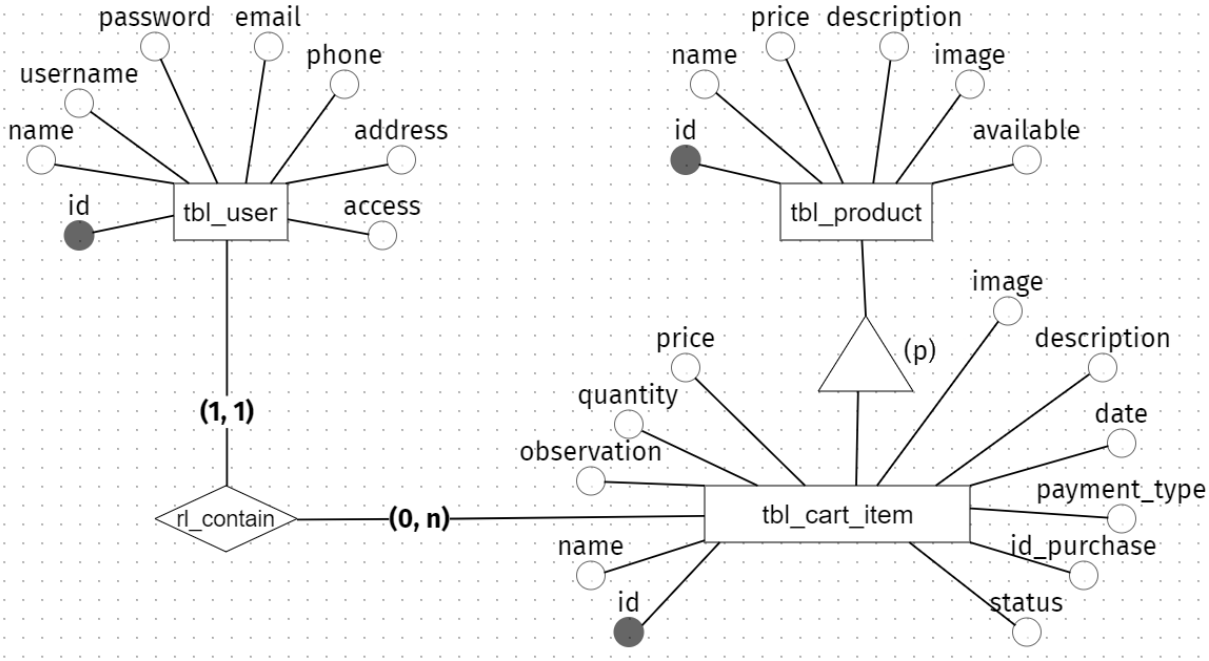
Caso de uso	**Autor**
	Watyson Guimarães Soares
Descrição	
O aplicativo deve limitar a quantidade de pedidos em uma única encomenda em 10.	
Dependência	**Prioridade**
	Essencial

Fonte: Autoria própria.

4.4 Descrição de Dados do Sistema

A ilustração 11 mostra o modelo de entidade relacionamento e nas tabelas 27, 28 e 29 tem uma descrição dos dados de cada entidade.

Ilustração 11 - Modelo entidade relacionamento (MER)



Fonte: Autoria própria.

Tabela 27 - Dados do sistema - Item do carrinho

Id.	Tipo	Tamanho	Descrição
id	Inteiro	-	Identificador.
name	Texto	255	Nome do produto.
observation	Texto	255	Observação do usuário na compra.
quantity	Real	-	Quantidade escolhida pelo usuário na compra.
price	Real	-	Preço do produto.
description	Texto	510	Descrição do produto.
image	Texto	255	Nome do arquivo imagem no servidor.
idtbl_user	Inteiro	-	Identificador do usuário que realizou a compra do produto.
date	Texto	255	Data da compra do produto
payment_type	Texto	255	Tipo de pagamento
id_purchase	Inteiro	-	Número aleatório que identifica a compra.
status	Inteiro	-	Número que determina o status e varia de 0 a 4, sendo 0

Fonte: Autoria própria.

Tabela 28 - Dados do sistema - Produto

Id.	Tipo	Tamanho	Descrição
id	Inteiro	-	Identificador.
name	Texto	255	Nome do produto.
price	Real	-	Preço do produto.
description	Texto	510	Descrição do produto.
image	Texto	255	Nome do arquivo imagem no servidor.
available	Bool	-	Identifique se o produto deve ser mostrado no cardápio.

Fonte: Autoria própria.

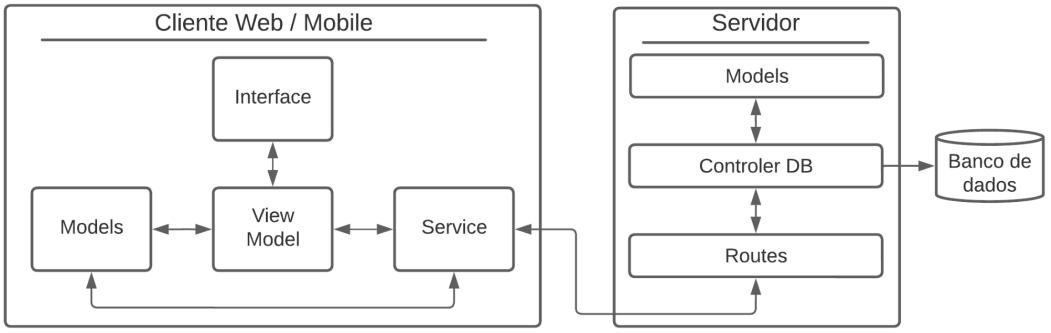
Tabela 29 - Dados do sistema - Usuário

Id.	Tipo	Tamanho	Descrição
id	Inteiro	-	Identificador.
name	Texto	255	Nome do usuário.
username	Texto	255	Nome de usuário usado para realizar o login.
password	Texto	255	Senha de usuário usada para realizar o login.
email	Texto	255	Email do usuário.
phone	Texto	255	Telefone do usuário.
address	Texto	255	Endereço do usuário.
access	Inteiro	-	Determina o nível de acesso do usuário, varia de 0 a 2.

Fonte: Autoria própria.

4.5 Modelo da Arquitetura

Ilustração 12 - Modelo da Arquitetura



Fonte: Autoria própria.

O software foi projetado utilizando três ideias de arquiteturas diferentes. Primeiro, a arquitetura cliente-servidor possibilitou a divisão do software em duas partes, sendo uma responsável pela interface do programa e a outra responsável pelo armazenamento e manutenção dos dados, podemos ver ambas separadas na ilustração 12.

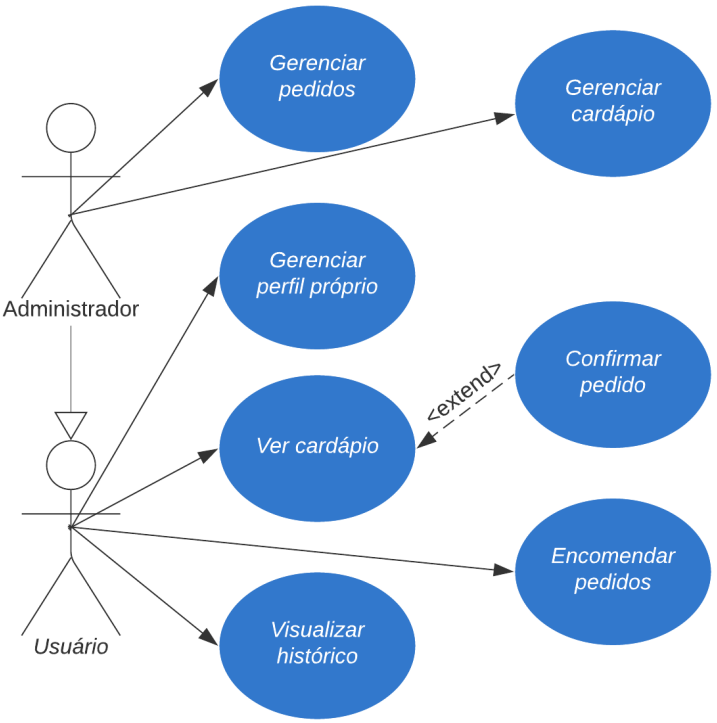
A segunda arquitetura usada foi a arquitetura MVVM no cliente web/mobile, que possibilitou a separação da interface gráfica das operações realizadas em cada interface, o que permitiu uma maior manutenção e legibilidade do programa, estão representadas na ilustração 12 na parte Cliente Web/Mobile.

A terceira arquitetura usada foi a arquitetura em camadas no servidor, o que permitiu separar cada ação do servidor em uma camada responsável por ela, facilitando a manutenção e o reuso de código, estão representadas na ilustração 12 na parte Servidor.

4.6 Diagrama de Casos de Uso

A Ilustração 13 apresenta o diagrama de casos de uso do sistema.

Ilustração 13 - Diagrama de caso de uso



Fonte: Autoria própria.

4.7 Casos de Uso Descritivos

Tabela 30 - CSU 01: Gerenciar pedidos

Identificador: CSU 01

Nome: Gerenciar Perfil

Responsável: Watyson Guimarães Soares

Requisitos Relacionados: RF 11, RF 15, RF 16

Descrição/Resumo: Permite realizar a gerência dos pedidos.

Atores: Administrador

Pré-condições: Os atores devem estar conectados à internet e ter acessado o sistema.

Pós-condições: O sistema deve salvar as alterações no banco de dados.

Cenário Principal:

1. O sistema apresenta uma lista com todos os pedidos.
2. O ator seleciona um pedido.
3. Sistema mostra os dados do pedido (data, descrição, observações, quantidade, preço e estado atual).
4. Ator define um novo estado atual.
5. Ator clica no botão atualizar.
6. Sistema atualiza os dados no banco de dados.
7. O sistema mostra a mensagem "Dados atualizados com sucesso.".
8. Caso de uso encerrado.

Cenários Alternativos:

1a - Fluxo alternativo - Ator visualiza pedido

1. O sistema apresenta uma lista com todos os pedidos.
2. O ator seleciona um pedido.
3. Sistema mostra os dados do pedido (data, descrição, observações, quantidade, preço e estado atual).
4. Ator clica no botão Voltar.
5. O sistema apresenta uma lista com todos os pedidos.
6. Caso de uso encerrado.

Cenários de Exceção:

1.1. O sistema está fora de acesso.

1.1.1. O sistema mostra a mensagem "Erro ao tentar se conectar com o servidor.".

Fonte: Elaborado pelo autor.

Tabela 31 - CSU 02: Gerenciar cardápio

Identificador: CSU 02

Nome: Gerenciar cardápio

Responsável: Watyson Guimarães Soares

Requisitos Relacionados: RF 11, RF 13

Descrição/Resumo: Possibilita que os atores gerenciem o cardápio.

Atores: Administrador

Pré-condições: Os atores devem estar conectados à internet e ter acessado o sistema.

Pós-condições: O sistema deve salvar as alterações no banco de dados.

Cenário Principal:

1. O sistema apresenta uma lista com todos os produtos.
2. Ator seleciona um dos produtos da lista.
3. Sistema mostra a tela com os dados do produto (imagem, nome, preço, descrição, disponível no cardápio).
4. Ator realiza as devidas alterações nos dados do produto.
5. Ator clica no botão atualizar.
6. Sistema atualiza os dados no banco de dados.
7. O sistema mostra a mensagem "Dados atualizados com sucesso.".
8. Caso de uso encerrado.

Cenários Alternativos:

1a - Fluxo alternativo - Ator visualiza produto

1. O sistema apresenta uma lista com todos os produtos.
2. Ator seleciona um dos produtos da lista.
3. Sistema mostra a tela com os dados do produto (imagem, nome, preço, descrição, disponível no cardápio).

4. Ator clica no botão voltar.
5. O sistema apresenta uma lista com todos os produtos.
6. Caso de uso encerrado.

1b - Fluxo alternativo - Criar novo produto

1. O sistema apresenta uma lista com todos os produtos.
2. Ator seleciona o botão "+" no canto superior direito da tela.
3. O sistema mostra a tela com todos os campos para criar um novo produto.
4. Ator preenche os campos com os dados do novo produto.
5. Ator clica no botão criar.
6. O sistema cria os dados no banco de dados.
7. O sistema mostra a mensagem "Produto criado com sucesso.".
8. Caso de uso encerrado.

1c - Fluxo alternativo - Ator apenas visualizar cardápio

1. O sistema apresenta uma lista com todos os produtos.
2. Ator visualiza o cardápio.
3. Ator clica no ícone "←", no canto superior esquerdo, para retornar ao menu de gerência.
4. Caso de uso encerrado.

1d - Fluxo alternativo - Ator deleta produto (sim)

1. O sistema apresenta uma lista com todos os produtos.
2. Ator seleciona o botão com ícone de lixeira de um produto.
3. O sistema confirma se deseja deletar o produto.
4. Ator seleciona "Sim".
5. O sistema deleta o produto.
6. O sistema deleta os dados no banco de dados.
7. O sistema mostra a mensagem "Dados deletados com sucesso.".
8. Caso de uso encerrado.

1e - Fluxo alternativo - Ator deleta produto (não)

1. O sistema apresenta uma lista com todos os produtos.
2. Ator seleciona o botão com ícone de lixeira de um produto.
3. O sistema confirma se deseja deletar o produto.

4. Ator seleciona “Não”.
5. O sistema retorna para a tela de gerência do cardápio.
6. Caso de uso encerrado.

Cenários de Exceção:

1.1. O sistema está fora de acesso.

1.1.1. O sistema mostra a mensagem "Erro ao tentar se conectar com o servidor.".

2.1. Ator deixa algum campo em branco na criação de um novo produto;

2.1.1. O sistema mostra a mensagem abaixo dos campos “Valor não pode ser vazio”.

Fonte: Elaborado pelo autor.

Tabela 32 - CSU 03: Gerenciar perfil próprio

Identificador: CSU 03

Nome: Gerenciar perfil próprio

Responsável: Watyson Guimarães Soares

Requisitos Relacionados: RF 01, RF 02, RF 03, RF 14

Descrição/Resumo: Possibilita aos atores, administrarem seus próprios dados pessoais (email, endereço, telefone, senha) no sistema.

Atores: Usuário

Pré-condições: Os atores devem estar conectados à internet e ter acessado o sistema.

Pós-condições: O sistema deve salvar as alterações no banco de dados.

Cenário Principal:

1. O sistema apresenta a tela de perfil do usuário.
2. Ator altera seus dados pessoais (email, endereço, telefone, senha).
3. Ator clica no botão atualizar.
4. O sistema atualiza os dados no banco de dados.
5. O sistema mostra a mensagem "Dados atualizados com sucesso.".
6. Caso de uso encerrado.

Cenários Alternativos:

1a - Fluxo alternativo - Ator visualiza seus dados pessoais

1. O sistema apresenta a tela de perfil do usuário.
2. Ator altera seus dados pessoais (email, endereço, telefone, senha).
3. Ator clica no botão voltar.
4. Caso de uso encerrado.

Cenários de Exceção:

1.1. O sistema está fora de acesso.

1.1.1. O sistema mostra a mensagem "Servidor fora do ar, tente novamente mais tarde.".

2.1. Ator deixa algum campo em branco;

2.1.1. O sistema mostra a mensagem abaixo dos campos “Todos os campos estão vazios ou nulos.”.

Fonte: Elaborado pelo autor.

Tabela 33 - CSU 04: Ver cardápio

Identificador: CSU 04

Nome: Ver cardápio

Responsável: Watyson Guimarães Soares

Requisitos Relacionados: RF 05

Descrição/Resumo: Possibilita que o ator veja o cardápio e realize o pedido de algum produto.

Atores: Usuário

Pré-condições: Os atores devem estar conectados à internet e ter acessado o sistema.

Pós-condições: O sistema deve enviar os dados do produto desejado para o carrinho do ator.

Cenário Principal:

1. O sistema apresenta uma lista com todos os produtos disponíveis.

2. Ator seleciona um produto.
3. O sistema mostra uma tela com os dados do produto (imagem, nome, preço, descrição).
4. Ator seleciona a quantidade do produto que deseja.
5. Ator escreve alguma observação quanto aos pedidos (Opcional).
6. Ator clica em confirmar.
7. O sistema adiciona o item desejado na lista pessoal de compras do ator (carrinho).
8. O sistema apresenta a mensagem "Item adicionado ao carrinho com sucesso.".
9. O sistema retorna para a tela do cardápio.
10. Caso de uso encerrado.

Cenários Alternativos:

1a - Fluxo alternativo - Ator visualiza os dados do produto

1. O sistema apresenta uma lista com todos os produtos disponíveis.
2. Ator seleciona um produto.
3. O sistema mostra uma tela com os dados do produto (imagem, nome, preço, descrição).
4. Ator visualiza os dados do produto.
5. Ator clica em voltar.
6. O sistema retorna para a tela do cardápio.
7. Caso de uso encerrado.

Cenários de Exceção:

1.1. O sistema está fora de acesso.

1.1.1. O sistema mostra a mensagem "Erro ao tentar se conectar com o servidor.".

Fonte: Elaborado pelo autor.

Tabela 34 - CSU 05: Encomendar pedidos

Identificador: CSU 05

Nome: Encomendar pedidos

Responsável: Watyson Guimarães Soares

Requisitos Relacionados: RF 05, RF 06, RF 07, RF 08

Descrição/Resumo: Permite ao ator, escolher se realiza ou não a encomenda do produto.

Atores: Usuário

Pré-condições: Os atores devem estar conectados à internet e ter acessado o sistema.

Pós-condições: O sistema deve salvar os dados do pedido no banco de dados.

Cenário Principal:

1. Ator seleciona o carrinho.
2. Sistema mostra os dados do carrinho (endereço, troco, lista de produtos).
3. Ator verifica e atualiza os dados de endereço e troco.
4. Ator confirma os produtos na lista de produtos.
5. Ator clica no botão confirmar.
6. O sistema salva os dados no banco de dados.
7. O sistema limpa a lista de produtos do carrinho.
8. O sistema mostra a mensagem "Item adicionado aos pedidos com sucesso."
9. Caso de uso se encerra.

Cenários Alternativos:

1a - Fluxo alternativo - Ator retira produto da lista de produtos.

1. Ator seleciona o carrinho.
2. Sistema mostra os dados do carrinho (endereço, troco, lista de produtos).
3. Ator verifica e atualiza os dados de endereço e troco.
4. Ator clica no ícone de lixeira.
5. O sistema retira o produto da lista de produtos.
6. Caso de uso se encerra.

1b - Fluxo alternativo - Ator visualiza a lista de produtos.

1. Ator seleciona o carrinho.
2. Sistema mostra os dados do carrinho (endereço, troco, lista de produtos).
3. Ator visualiza os dados do carrinho
4. Ator clica no botão voltar.
5. O sistema mostra a tela de cardápio.
6. Caso de uso se encerra.

Cenários de Exceção:

1.1. O sistema está fora de acesso.

1.1.1. O sistema mostra a mensagem "Erro ao tentar se conectar com o servidor.".

2.1. Ator confirma a compra sem produtos na lista.

2.1.1. O sistema mostra a mensagem abaixo dos campos “Sem itens para confirmar a compra”.

Fonte: Elaborado pelo autor.

Tabela 35 - CSU 06: Visualizar histórico

Identificador: CSU 06

Nome: Visualizar histórico

Responsável: Watyson Guimarães Soares

Requisitos Relacionados: RF 09, RF 10

Descrição/Resumo: Permite ao ator visualizar o histórico pessoal.

Atores: Usuário

Pré-condições: Os atores devem estar conectados à internet e ter acessado o sistema.

Pós-condições: N/A

Cenário Principal:

1. Ator seleciona o histórico.
2. O sistema mostra os dados do histórico do mais recente para o mais antigo.
3. Ator visualiza os dados.

4. Ator clica em voltar.
5. Caso de uso se encerra.

Cenários Alternativos:

Cenários de Exceção:

1.1. O sistema está fora de acesso.

1.1.1. O sistema mostra a mensagem "Erro ao tentar se conectar com o servidor.".

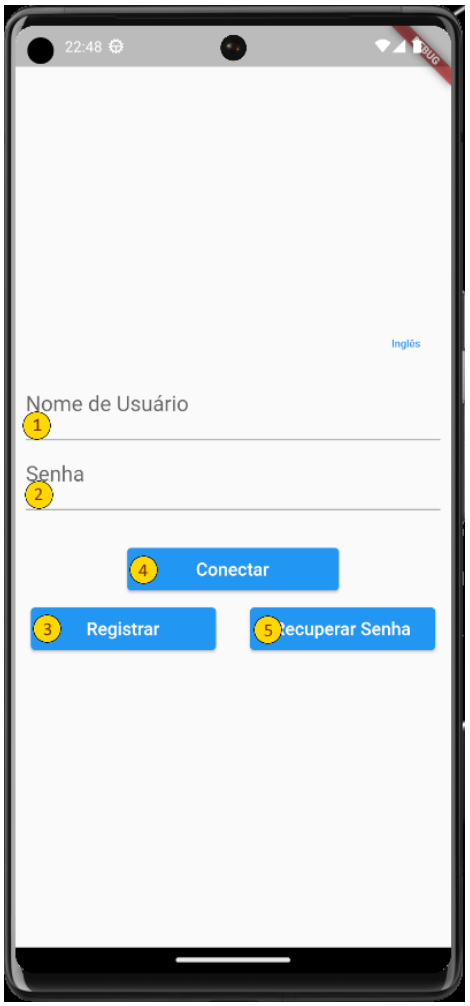
Fonte: Elaborado pelo autor.

5 PROTÓTIPO

Neste capítulo, serão mostrados os protótipos dos aplicativos, juntamente com uma descrição de suas funcionalidades.

5.1 Aplicativo Mobile

Ilustração 14 - Tela de login



Fonte: Autoria própria.

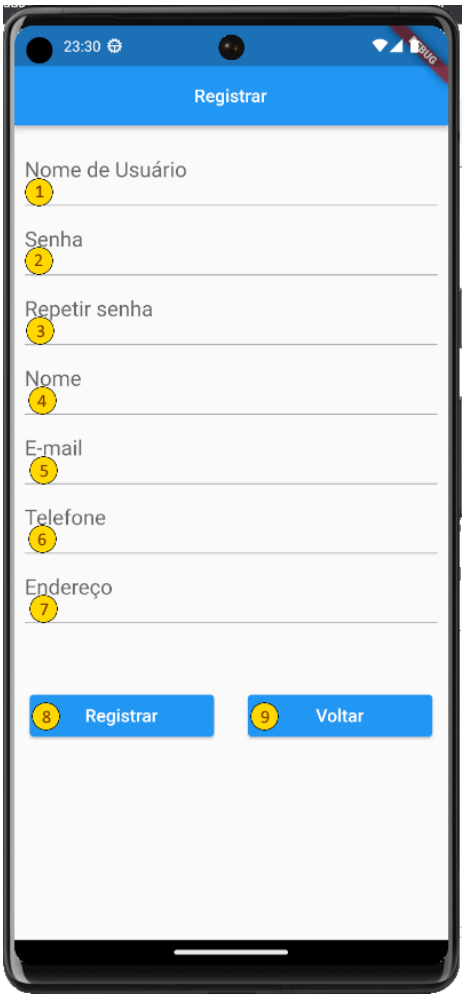
A tela aparece primeiro assim que o usuário entra no aplicativo e permite que ele faça login no sistema, se registre ou recupere sua conta.

Descrição dos componentes:

- **Pontos 1 e 2:** Campos onde o usuário deve colocar seus dados de acesso previamente cadastrados.

- **Ponto 3:** Um botão onde o usuário deve pressionar após preencher os campos nome de usuário e senha para efetuar a entrada no aplicativo.
- **Ponto 4:** Um botão onde o usuário será redirecionado para a tela de cadastro.
- **Ponto 5:** Um botão onde o usuário será redirecionado para a tela de recuperar senha.

Ilustração 15 - Tela de cadastro



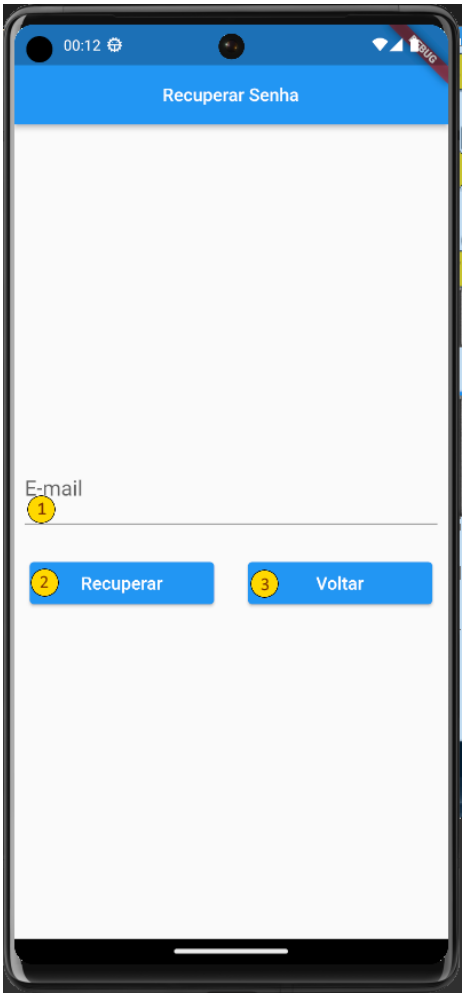
Fonte: Autoria própria.

A tela onde o usuário deve usar para se registrar no sistema, pode ser acessada por meio da tela de login ao pressionar o botão “Cadastrar”.

Descrição dos componentes:

- **Pontos 1, 2, 3, 4, 5, 6, 7:** Campos onde o usuário deve preencher com seus respectivos dados.
- **Ponto 8:** Após preencher os campos anteriores o usuário deve clicar aqui para registrar os dados.
- **Ponto 9:** Um botão onde o usuário será redirecionado para a tela de login.

Ilustração 16 - Recuperação de conta



Fonte: Autoria própria.

A tela para recuperar a senha pode ser acessada por meio da tela de login ao pressionar o botão 'Recuperar Senha'.

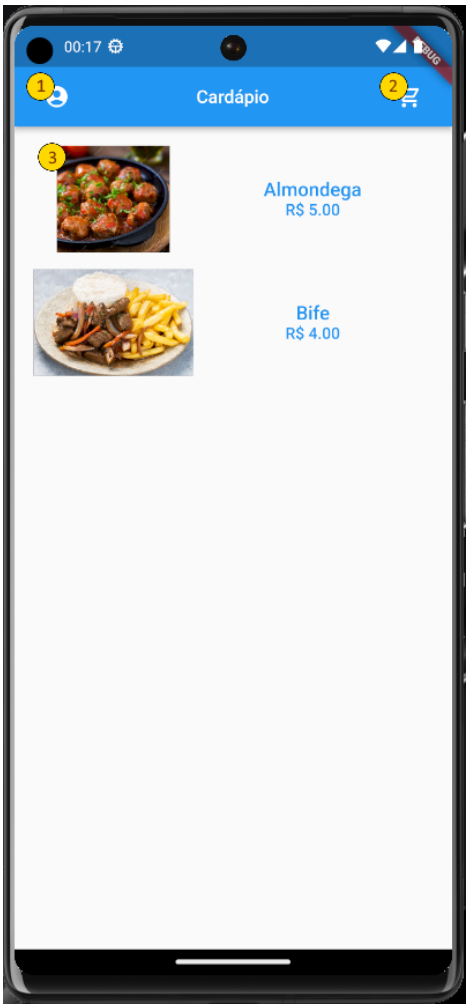
Descrição dos componentes:

Ponto 1: Campo onde o usuário já cadastrado coloca seu email para recuperar a senha.

Ponto 2: Um botão que realiza a recuperação de senha.

Ponto 3: Um botão onde o usuário será redirecionado para a tela de login.

Ilustração 17 - Cardápio



Fonte: Autoria própria.

A tela onde o usuário pode ver os produtos disponíveis no cardápio é acessada logo após realizar o login.

Descrição dos componentes:

Ponto 1: Botão que redireciona o usuário para a tela de perfil.

Ponto 2: Botão que redireciona o usuário para a tela do carrinho de compras.

Ponto 3: Lista com as opções que o usuário pode escolher para fazer o pedido.

Ilustração 18 - Confirmação de pedido

Fonte: Autoria própria.

A tela que permite ao usuário escolher os produtos para realizar o seu pedido. Após o usuário realizar a escolha, o produto será adicionado ao carrinho, e essa tela pode ser acessada ao selecionar algum item na lista do cardápio.

Descrição dos componentes:

Ponto 1: Botões onde o usuário pode escolher a quantidade do produto desejada, limitados em 10.

Ponto 2: Campo onde o usuário pode inserir alguma observação relacionada ao pedido.

Ponto 3: Botão onde o usuário pode confirmar o pedido, vai enviar o pedido para o carrinho de compras do usuário.

Ponto 4: Botão que permite ao usuário retornar para a tela do cardápio.

Ilustração 19 - Carrinho

Fonte: Autoria própria.

A tela do carrinho onde o usuário pode confirmar o seu pedido e pode ser acessada por meio da tela do cardápio.

Descrição dos componentes:

Ponto 1: Campo onde o usuário insere o endereço para entrega.

Ponto 2: Campo onde o usuário insere caso precise de troco.

Ponto 3: Botão onde o usuário pode excluir um produto da lista do carrinho.

Ponto 4: Botão onde permite ao usuário finalizar a compra e fazer o pedido.

Ponto 5: Botão que permite ao usuário retornar para a tela do cardápio.

Ponto 6: Botão que permite ao usuário ir para a tela do histórico.

Ilustração 20 - Histórico de pedidos

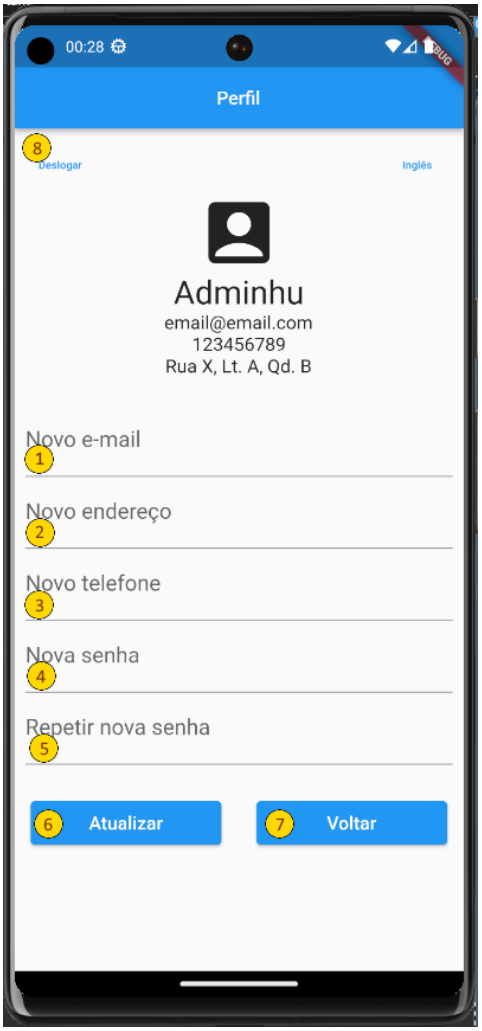
Fonte: Autoria própria.

A tela do histórico é onde o usuário pode visualizar todos os pedidos realizados e pode ser acessada por meio da tela do carrinho.

Descrição dos componentes:

Ponto 1: Botão que permite ao usuário retornar para a tela do carrinho.

Ilustração 21 - Perfil



Fonte: Autoria própria.

A tela do perfil pessoal do usuário permite visualizar e atualizar seus dados pessoais, e pode ser acessada por meio da tela do cardápio.

Descrição dos componentes:

Pontos 1, 2, 3, 4, 5: Campos onde o usuário insere seus respectivos dados que deseja atualizar.

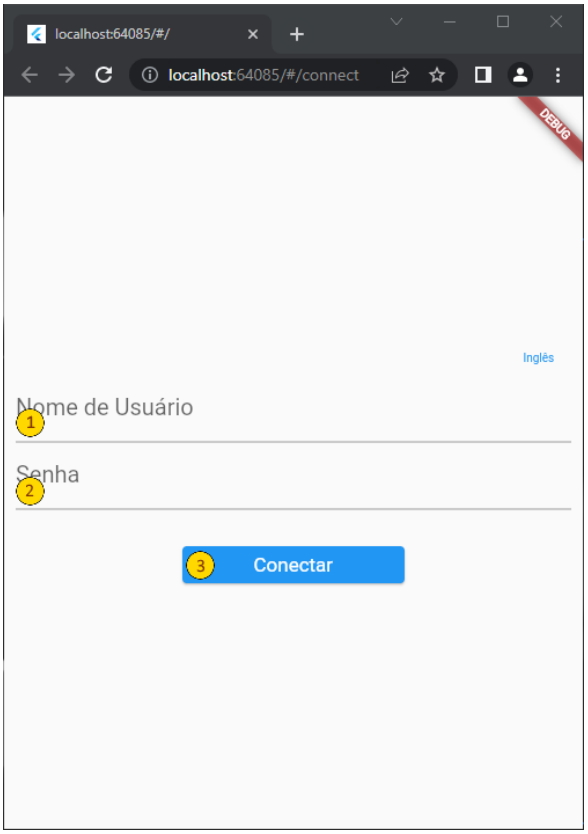
Ponto 6: Botão que atualiza os dados do usuário, sendo necessário preencher pelo menos um dos campos.

Ponto 7: Botão que redireciona o usuário para a tela do cardápio.

Ponto 8: Realiza a função de sair da conta do usuário e o envia para a tela de login.

5.2 Aplicativo Web

Ilustração 22 - Login



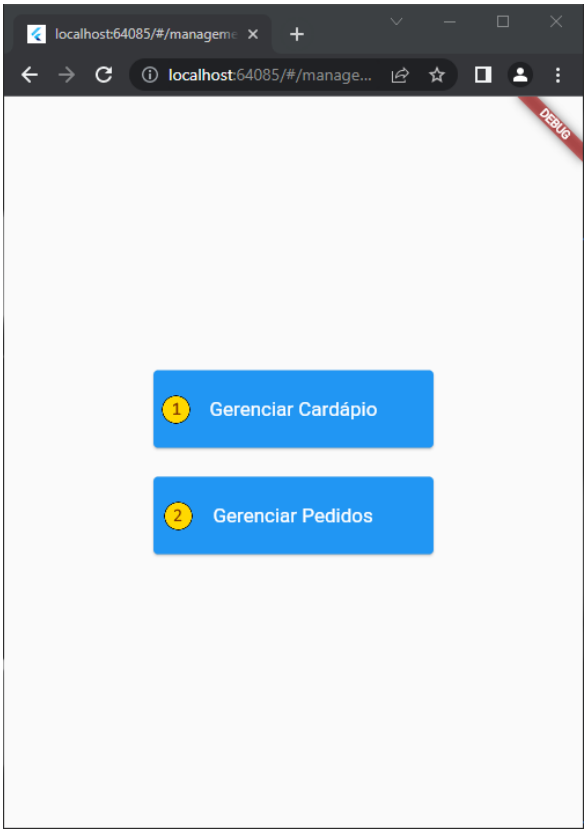
Fonte: Autoria própria.

A tela que aparece assim que o usuário entra no aplicativo é a tela de login, onde ele pode realizar o login no sistema.

Descrição dos componentes:

- **Pontos 1 e 2:** Campos onde o usuário deve inserir seus dados de acesso previamente cadastrados.
- **Ponto 3:** Um botão onde o usuário deve pressionar após preencher os campos nome de usuário e senha para efetuar a entrada no aplicativo.

Ilustração 23 - Gerência



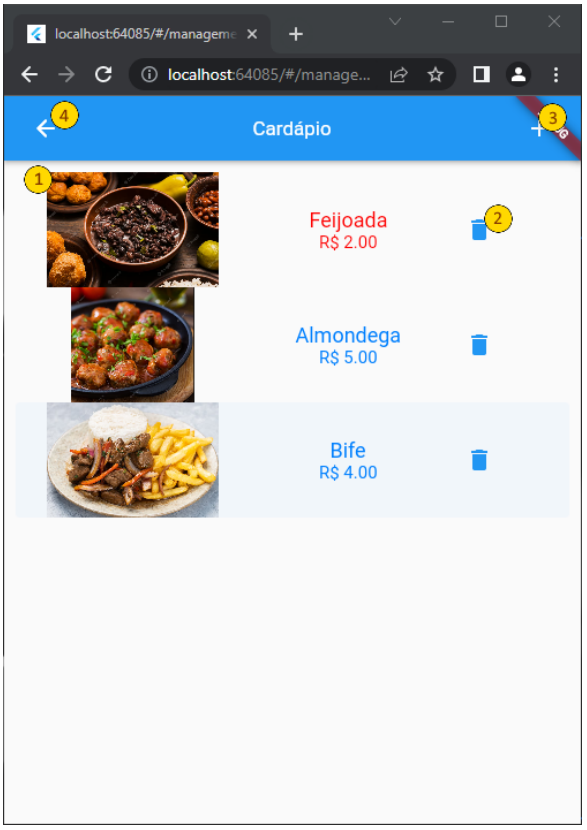
Fonte: Autoria própria.

Tela de menu que aparece após realizar o acesso no sistema e apresenta opções para o administrador realizar a gerência.

Descrição dos componentes:

- **Ponto 1:** Botão que direciona o usuário para a tela de gerência do cardápio.
- **Ponto 2:** Botão que direciona o usuário para a tela de gerência dos pedidos.

Ilustração 24 - Gerenciar cardápio



Fonte: Autoria própria.

A tela de gerência do cardápio permite ao usuário controlar os dados que devem aparecer no cardápio para os usuários e pode ser acessada selecionando "Gerenciar Cardápio" no menu de gerência.

Descrição dos componentes:

- **Ponto 1:** Lista com todos os produtos disponíveis. Os que aparecem com nome vermelho estão cadastrados, mas não são exibidos para o usuário no cardápio.
- **Ponto 2:** Botão que exclui os dados daquele produto do sistema.
- **Ponto 3:** Botão que direciona o usuário para a tela de cadastro de um novo produto.
- **Ponto 4:** Botão que direciona o usuário de volta para a tela do menu de gerência.

Ilustração 25 - Edição de pedido

Fonte: Autoria própria.

A tela de edição dos produtos do cardápio permite ao usuário editar os dados dos produtos e pode ser acessada ao selecionar um produto na tela de gerenciar o cardápio.

Descrição dos componentes:

Pontos 1, 2, 3, 4: Campos onde o usuário deve preencher com os respectivos dados do produto que deseja adicionar

Ponto 5: Indica se o produto deve aparecer no cardápio ou não.

Ponto 6: Um botão onde o usuário salva as alterações dos dados do produto.

Ponto 7: Um botão que direciona o usuário para a tela de gerenciar o cardápio.

Ilustração 26 - Criar novo pedido

Fonte: Autoria própria.

A tela de criação dos produtos do cardápio permite ao usuário criar novos produtos e pode ser acessada ao selecionar o botão de “+” no canto superior direito na tela gerenciar cardápio.

Descrição dos componentes:

Pontos 1, 2, 3, 4: Campos onde o usuário deve preencher com os respectivos dados do novo produto.

Ponto 5: Indica se o produto deve aparecer no cardápio ou não.

Ponto 6: Um botão onde o usuário salva os dados do novo produto.

Pontos 7, 8: Botão que direciona o usuário para a tela de gerenciar o cardápio.

Ilustração 27 - Gerenciar pedidos

Fonte: Autoria própria.

A tela de gerenciamento dos pedidos permite ao usuário controlar os pedidos realizados e pode ser acessada ao selecionar o botão "Gerenciar Pedidos" no menu de gerência.

Descrição dos componentes:

Ponto 1: Botão que direciona o usuário de volta para a tela do menu de gerência.

Ponto 2: Filtro de estado que controla quais os tipos de pedidos que devem ser exibidos.

Ponto 3: Lista com todos os pedidos realizados filtrados pelo estado selecionado no filtro do ponto 2, ordenados do mais antigo ao mais recente.

Ilustração 28 - Atualização de venda

Fonte: Autoria própria.

A tela de edição do estado das encomendas permite ao usuário atualizar o estado das encomendas e pode ser acessada ao selecionar um pedido na tela "Gerenciar Pedidos".

Descrição dos componentes:

Ponto 1: Uma lista com todos os pedidos realizados.

Ponto 2: Botão que atualiza o estado da encomenda no banco de dados.

Ponto 4: O estado atual daqueles pedidos encomendados.

Pontos 3, 5: Botão que direciona o usuário de volta para a tela de gerenciar pedidos.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, o objetivo foi estudar as linguagens de programação Rust e Dart, juntamente com as ferramentas Actix e Flutter, para o desenvolvimento de um aplicativo. Durante o desenvolvimento do projeto foi possível explorar diversas características únicas de cada linguagem e ferramenta.

O Rust se destacou devido a sua segurança e desempenho, mas também apresentou um desafio quanto ao seu aprendizado, ele se mostrou um pouco complexo de se aprender.

O Dart em conjunto ao Flutter foi muito eficiente no desenvolvimento de aplicativos móveis, permitindo criar telas muito responsivas rapidamente. O aplicativo final demonstra que o uso dessas ferramentas juntas foi suficiente para o desenvolvimento da aplicação proposta.

6.1 Sugestões para trabalhos futuros

- Implementar suporte a múltiplas linguagens no banco de dados;
- Adicionar mais opções de pagamento;
- Implementar uma lista de possíveis endereços para o usuário;
- Implementar a opção de capturar o endereço do usuário via *Global Positioning System* (GPS) quando o mesmo for cadastrar um endereço;
- Implementar a visualização do trajeto do produto até o endereço solicitado.

REFERÊNCIAS

ACTIX. **What is Actix Web?**. 2023. Disponível em: https://actix.rs/docs/whatis. Acesso em: 28 mar. 2023.

ALLAN. **Tutorial de Android Studio: Introdução completa ao Android Studio**. 2016. Disponível em: https://www.devmedia.com.br/tutorial-de-android-studio/34003. Acesso em: 27 mar. 2023.

BELLINASO, Marco. **Flutter: the good, the bad and the ugly**. 2018. Disponível em: https://medium.com/asos-techblog/flutter-vs-react-native-for-ios-android-app-development-c41b4e038db9. Acesso em: 30 mar. 2023.

BRITTO, Vinícius; NERY, Carmen. **Internet já é acessível em 90,0% dos domicílios do país em 2021**. 2022. Disponível em: https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/34954-internet-ja-e-acessivel-em-90-0-dos-domicilios-do-pais-em-2021. Acesso em: 26 out. 2022.

CÂNDIDO, Jessica. **Panorama do uso da Internet no país (%)**. 2022. 1 fotografia. 751x851 pixels. Disponível em: https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/34954-internet-ja-e-acessivel-em-90-0-dos-domicilios-do-pais-em-2021. Acesso em: 26 out. 2022.

CASTIGLIONI, Matheus. **Arquitetura em Camadas**. 2022. Disponível em: https://imasters.com.br/arquitetura-da-informacao/arquitetura-em-camadas. Acesso em: 03 maio 2023.

CAVALCANTI, Eric. **Flutter — uma breve introdução**. 2018. Disponível em: https://medium.com/@ecavalcanti/flutter-uma-breve-introdu%C3%A7%C3%A3o-d9071fcb8474. Acesso em: 30 mar. 2023.

GOOGLE. **Conhecer o Android Studio**. 2022. Disponível em: https://developer.android.com/studio/intro?hl=pt-br. Acesso em: 27 mar. 2023.

HOWARTH, Jesse. **Why Discord is switching from Go to Rust**. 2020. Disponível em: https://medium.com/discord-engineering/why-discord-is-switching-from-go-to-rust-a190bbca2b1f. Acesso em: 30 mar. 2023.

JULIANO. **Gerenciamento de Banco de Dados: Análise Comparativa de SGBD'S**. 2014. Disponível em: https://www.devmedia.com.br/gerenciamento-de-banco-de-dados-analise-comparativa-de-sgbd-s/30788. Acesso em: 28 mar. 2023.

JÚNIOR, Elemar. **Como a linguagem Rust resolve o desafio de desalocação de objetos (com ownership, sem memory-leaks e sem GC)**. 2019.

https://eximia.co/como-a-linguagem-rust-resolve-o-desafio-de-desalocacao-de-objetos-com-ownership-sem-memory-leaks-e-sem-gc. Acesso em: 24 ago. 2022.

KANJILAL, Joydip. **Test Your REST APIs Using Insomnia REST Client**. 2022. Disponível em: https://www.codemag.com/Article/2107051/Test-Your-REST-APIs-Using-Insomnia-REST-Client. Acesso em: 27 mar. 2023.

MENDES, Caio. **Arquitetura Cliente-Servidor**. 2021. Disponível em: https://medium.com/@caiomay.mendes/arquitetura-cliente-servidor-b2adeeb3632e. Acesso em: 03 maio 2023.

MICROSOFT. **Visual Studio Code Documentation**. 2022. Disponível em: https://code.visualstudio.com/docs. Acesso em: 24 mar. 2023.

OLEINIK, Anthony. **WebSockets in Actix Web Full Tutorial — WebSockets & Actors**. 2020. Disponível em: https://levelup.gitconnected.com/websockets-in-actix-web-full-tutorial-websockets-actors-f7f9484f5086. Acesso em: 30 mar. 2023.

PAYPAL BRASIL. **1 em cada 4 brasileiros pretende continuar fazendo compras online diariamente após a pandemia**. 2021. Disponível em: https://newsroom.br.paypal-corp.com/consumo-online-no-brasil. Acesso em: 07 nov. 2022.

POSTGRESQL. **About**. 2023. Disponível em: https://www.postgresql.org/about/. Acesso em: 27 mar. 2023.

SCUADRA. **O que é e quais as vantagens do cardápio digital para o restaurante?**. 2023. Disponível em: https://www.scuadra.com.br/blog/o-que-e-e-quais-as-vantagens-do-cardapio-digital-para-o-restaurante. Acesso em: 24 maio 2023.

SILVA, Gisele. **O que é arquitetura MVVM?**. 2022. Disponível em: https://coodesh.com/blog/dicionario/o-que-e-arquitetura-mvvm. Acesso em 26 maio 2023.

SOUZA, Alex. **Modelagem Relacional (uma visão geral)**. 2023. Disponível em: https://medium.com/blog-do-zouza/modelagem-relacional-uma-vis%C3%A3o-geral-44cd8807fc87. Acesso em: 25 abr. 2023.

STACK OVERFLOW. **2022 Developer Survey**. 2022. Disponível em: https://survey.stackoverflow.co/2022/. Acesso em: 24 mar. 2023.

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR

Av. Universitária, 1069 ● Setor Universitário
Caixa Postal 86 ● CEP 74605-010
Goiânia ● Goiás ● Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br ● reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante **WATYSON GUIMARÃES SOARES** do Curso de **ENGENHARIA DA COMPUTAÇÃO**, matrícula **2018200330049-8**, telefone: **(62) 9 9487-6758,** e-mail **watysonsoares@hotmail.com**, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado **DESENVOLVIMENTO DE APLICATIVO MÓVEL - (TASTEQUICKLY)**, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, **17** de **JUNHO** de **2023**.

Assinatura do autor: Watyson G. Soares

Nome completo do autor: Watyson Guimarães Soares

Assinatura do professor–orientador:

Nome completo do professor-orientador: