UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DOUGLAS WENDER LOPES DE OLIVEIRA

UTILIZAÇÃO DE API PARA GEOLOCALIZAÇÃO EM TEMPO REAL DE ENCOMENDAS PEDIDAS POR MEIO DE UM APLICATIVO MOBILE MULTIPLATAFORMA

DOIS VIZINHOS

2022

DOUGLAS WENDER LOPES DE OLIVEIRA

UTILIZAÇÃO DE API PARA GEOLOCALIZAÇÃO EM TEMPO REAL DE ENCOMENDAS PEDIDAS POR MEIO DE UM APLICATIVO MOBILE MULTIPLATAFORMA

Use of API for realtime geolocation of orders ordered through a multiplatform mobile application

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Software do Curso de Bacharelado em Engenharia de Software da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Marlon Marcon

DOIS VIZINHOS
2022

DOUGLAS WENDER LOPES DE OLIVEIRA

UTILIZAÇÃO DE API PARA GEOLOCALIZAÇÃO EM TEMPO REAL DE ENCOMENDAS PEDIDAS POR MEIO DE UM APLICATIVO MOBILE MULTIPLATAFORMA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Software do Curso de Bacharelado em Engenharia de Software da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 24/junho/2022

Marlon Marcon
doutorado
Universidade Tecnológica Federal do Paraná

Evandro Miguel Kuszera
doutorado
Universidade Tecnológica Federal do Paraná

Newton Carlos Will
doutorado
Universidade Tecnológica Federal do Paraná

DOIS VIZINHOS
2022

Pelo carinho, afeto e dedicação que meus pais
me deram durante toda a minha existência, eu
dedico este trabalho a eles com muita gratidão.

AGRADECIMENTOS

Agradeço a Deus, por ter permitido que tivesse saúde e determinação para não desanimar durante a realização deste trabalho e também fez com que meus objetivos fossem alcançados durante todos os meus anos de estudos, também agradeço e a todos os demais que me auxiliaram nesta jornada, em especial para o meu orientador pela paciência e conhecimento passado para mim.

“Não importa o que aconteça, continue a nadar”.
(WALTERS, GRAHAM; PROCURANDO NEMO, 2003).

RESUMO

Em virtude da pandemia do novo coronavírus, o e-commerce tem se intensificado cada vez mais, esse fenômeno pode ser observado principalmente em pequenos e médios negócios. Como resultado desse aumento, os serviços de entregas têm se mostrado cada vez mais requisitados pela sua praticidade e qualidade. Atualmente o Correios é um dos maiores serviços de entregas brasileiro, porém seu alto custo e longo prazo de envio é pouco atrativo, por esse motivo a busca por métodos alternativos de entrega tem crescido cada vez mais em plataformas como Facebook e Whatsapp. Visando os pontos negativos dos métodos de envio tradicional a procura por "caronas de entrega" entre cidades próximas ou até mesmo dentro de grandes centros urbanos, se tornou um método alternativo popular. Tendo isto em mente, o presente trabalho propõe um aplicativo multi-plataforma utilizando a Framework Flutter juntamente com a API de geolocalização do Google, que inclui os mapas e as rotas, para publicação, seleção, envio e rastreio de encomendas, conectando desta forma motoristas, remetentes e destinatários; com o objetivo de concluir com rapidez e segurança envios e entregas.

Palavras-chave: desenvolvimento mobile; flutter; geolocalização.

ABSTRACT

Due to the pandemic of the new coronavirus, e-commerce has intensified more and more. This phenomenon can be observed mainly in small and medium businesses. As a result, due to practicality and quality, we follow a demand increase for delivery services. Currently, Correios is one of the most extensive Brazilian delivery services. However, its high cost and delivery time shipping are not very attractive. For this reason, the search for alternative delivery methods has grown increasingly on platforms such as Facebook and Whatsapp. Aiming at the negative points of traditional shipping methods, searching for "delivery hitchhikers"between nearby cities or even large urban centers has become a popular alternative method. With this in mind, the present work proposes a multi-platform application using the Framework Flutter jointly with the Google geolocation API. This API includes maps and routes for publishing, selecting, sending, and tracking orders, thus connecting drivers, senders, and recipients; to quickly and securely complete shipments and deliveries.

Keywords: mobile development; flutter; geolocation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Em virtude da pandemia mundial do novo coronavírus, o comércio eletrônico tem se intensificado, principalmente relacionado a pequenos e médios negócios online. Segundo a Associação Brasileira de Comércio Eletrônico (ABCOMM, 2021) tal número aumentou 400% em meados de 2020. Relacionado ao volume de vendas, também segundo a ABComm o aumento foi de 56% em relação à 2019. Além disso, as pessoas têm se tornado cada vez mais ocupadas em suas tarefas diárias e o tempo cada vez mais escasso para realizá-las. A distribuição de encomendas entre cidades vizinhas, seja no modelo B2C (*Business to Consumer*, do inglês do Negócio para o consumidor), ou C2C (*Consumer to Consumer*, do inglês do Consumidor para o Consumidor), é realizado normalmente por meios tradicionais, tais como transportadoras ou os Correios.

Muitas pessoas, se deslocam diariamente entre cidades, e comumente utilizam redes sociais, tais como Whatsapp e Facebook, para solicitar carona ou até mesmo para transporte de objetos pessoais ou encomendas entre cidades. Além disso, existem pessoas que sempre estão dirigindo pelo centro da cidade ou pelas rodovias do Brasil, que podem realizar tais tarefas gerando ganho financeiro por esse transporte. Em complemento, existem diversos aplicativos para caronas atualmente, funcionando em grandes cidades, podendo-se citar o Uber e 99 Táxi, além de outros serviços mais comuns em centros menores, como o Garupa ou Urbano Norte. Entretanto, tais serviços não englobam entregas de encomendas para uma cidade próxima. Há também aplicações que trabalham com entregas de encomendas como por exemplo o Uber Flash, Eu entrego e o Loggi, cada um com suas peculiaridades e especificações.

Esta proposta vislumbra a criação de um sistema, para rezalizar o levantamento de demanda e a alocação de transportadores independentes para mercadorias e passageiros, por meio de um aplicativo que utiliza geolocalização para rastreamento dos usuários. Levando em conta vários fatores como: a grande utilização de dispositivos móveis, como smartphones, e a acessibilidade da internet móvel de maneira geral, o foco da solução está no desenvolvimento de um aplicativo móvel que utiliza comunicação em tempo real via internet. Considerando o contexto apresentado, esta proposta visa desenvolver uma aplicação móvel nativa utilizando o framework Flutter, que utiliza o mesmo código para desenvolver aplicações aos dois principais sistemas operacionais móveis atualmente, o Android e o iOS.

Como requisitos da solução espera-se o desenvolvimento de uma arquitetura bem estruturada e uma lógica de negócio para anunciar e rastrear as encomendas pedidas pelo cliente através do smartphone, para um motorista ou entregador. Além disso, estão previstas atividades como a verificação de encomenda ou usuários, o traçado de rota para o motorista realizar a entrega e o rastreamento em tempo real da posição do motorista, responsável pela encomenda. Com base nos fatores previamente citados, a seguir apresentam-se os objetivos geral e específicos da proposta de Trabalho de Conclusão de Curso.

1.1 Objetivo Geral

O objetivo geral é desenvolver um sistema para dispositivos móveis multiplataforma, fundamentado no uso da geolocalização, para possibilitar a comunicação e acompanhamento de entregas com transportadores independentes.

1.2 Objetivos Específicos

Baseando-se no objetivo geral foram definidos os seguintes objetivos específicos:

1. Estudar e realizar o levantamento dos requisitos;
2. Analisar as características e capacidades do framework Flutter;
3. Integrar a API de Geolocalização;
4. Criar um protótipo inicial no produto;
5. Desenvolver e testar a solução proposta;

2 ESTADO DA ARTE

Nesta seção são apresentados sistemas que propõe tarefa similar à proposta neste trabalho. Na sequência, serão apresentados aspectos relacionados ao desenvolvimento de aplicações focadas em dispositivos móveis, relacionadas ao modelo de desenvolvimento e as principais tecnologias e arquiteturas utilizadas atualmente para tal.

2.1 Concorrentes

Atualmente no Brasil existem alguns aplicativos de caronas, que funcionam em grandes centros, sendo os mais famosos o Uber e o 99 Táxi, além de outros serviços mais comuns em cidades menores, como o Garupa e Urbano Norte. Entretando, tais serviços não englobam entregas de encomendas para uma cidade próxima.

Há também aplicações que trabalham com entregas de encomendas com segmentos e nichos diferentes do que o proposto, como por exemplo o Uber Flash, Eu entrego e Loggi, cada um com suas peculiaridades e especificações conforme a Tabela 1.

Tabela 1 – Comparativo entre os possíveis concorrentes do projeto.

	Prós	Contras
Uber Flash	Seguro, Referência em aplicativos, Atende as demandas	Só atende grandes centros (SP, RJ, BH), Somente bagagem de 10kg
Loggi	Funciona para transportadoras, atende outras regiões	Altos preços, aplicativo com alta incidência de travamentos,
Eu entrego	Seguro, conecta entregadores a uma rede de entregas, atende outras regiões	Sem suporte, pagamentos a cada 15 dias

Fonte: Autoria própria.

As avaliações desses aplicativos (exceto Uber já que ele é bem avaliado pelo transporte de pessoas também) são no geral ruins, a a avaliação do Loggi (LOGGI, 2021) mostra a média do aplicativo em 3,3.

O aplicativo Eu Entrego ainda tem uma média pior que a aplicação anterior, mantendo uma média de 2,0 nas suas avalições, contendo vários feedbacks negativos em comentários na loja da Google (ENTREGO, 2021).

2.2 Fundamentos

O projeto foi estruturado usando algumas práticas e paradigmas que auxiliam no desenvolvimento de uma aplicação consistente, de fácil manutenabilidade e entendimento. Se apro-

veitando de conceitos de programação orientata a objetos e de estruturas de armazenamento de dados na nuvem com a tecnologia NoSQL, sendo responável por isso o Firebase assim como o projeto irá utilizar a API de Geolocalização, ambos fornecidos pelo Google. Essa API fornece mapas e rotas para o aplicativo. Os capítulos abaixo tratam destes fundamentos que serão utilizados para o desenvolvimento do projeto.

2.2.1 Programação Orientada a Objetos

O paradigma de Programação Orientada a Objetos é uma abordagem que permite enxergar as estruturas básicas do sistema como objetos, podendo ser listas destes mesmos objetos, contendo métodos e atributos distintos. Ela permite ao sistema melhorar o reuso de códigos e extensibilidade.

Alguns dos princípios da orientação a objetos que são implementados no paradigma são: relacionamento, hierarquização, abstração, encapsulamento.

Essa proposta permite representar o código de maneira mais fiel possível ao mundo real, este diferencial permite a definição de estruturas e operações, dentre elas a herança, mecanismo que permite que as definições do objeto sejam facilmente estendidas. Juntamente com a herança, o polimorfismo tem sua importância, permitindo selecionar funcionalidades que um objeto irá utilizar de forma dinâmica em tempo de execução.

2.2.2 NoSQL

Será utilizado no projeto o banco de dados NoSQL (Not Only SQL), o qual fornecem maneiras eficientes de armazenamento de grandes volumes de dados, e auxiliam em pesquisas que necessitam de baixa latência, fatores importantes que são considerados durante a escolha para o determinado projeto.

Os bancos de dados relacionais nem sempre são as melhores escolhas, o NoSQL fornece várias maneiras para trabalhar com a estrutura de dados, uma delas é a orientada a documentos, onde seu formato é semelhante ao JSON (Javascript Object Notation). Outro formato que pode ser utilizado é o tipo a chave-valor, onde cada chave (key ou identificador), possui um ou mais atributos de valores. Porém essas estruturas são bastante flexíveis e permitem a escalabilidade da aplicação, e por isso é altamente recomendada para o projeto.

2.2.3 Arquitetura Limpa

A Arquitetura Limpa ou Clean Architecture, é um método de estruturação da aplicação de forma que resolva o gerenciamento de estado separando das páginas ou views e controlado-

res. Seu objetivo fundamental é separar os interesses de cada módulo da aplicação e permitir a escalabilidade do aplicativo.

Ao dividir o sistema em camadas que separam a lógica de negócios da implementação específica da plataforma, três princípios da Engenharia de Software são mantidos (código agnóstico a framework, testabilidade e isolamento de módulos externos). O padrão de desenvolvimento engloba três principais camadas, sendo elas:

- **Domínio** que é onde se encontram os casos de uso são as funções responsáveis pela comunicação da visão do aplicativo com os dados, as entidades que são modelos que extendem as funções dos models da camada de dados, e os repositórios que são a assinatura para a camada de dados utilizar como padrão.
- **Dados**, onde se encontram os modelos, sua definição dos campos e métodos utilizados como *toJson* ou *fromJson*, datasources que é de onde os dados vão vir, como API ou como no caso da aplicação, do Firebase. E repositório que é a "função"que vai chamar o datasource.
- **Apresentação**, que onde se encontram a parte visual do aplicativo e se comunicam com os dados (camada *data*) através do caso de uso (da camada de domínio).

Conforme a Figura 1, é possível visualizar a estrutura da arquitetura limpa. As seções subsequentes descrevem de maneira mais detalhada cada príncípio citado anteriomente.

2.2.3.1 Código agnóstico a framework

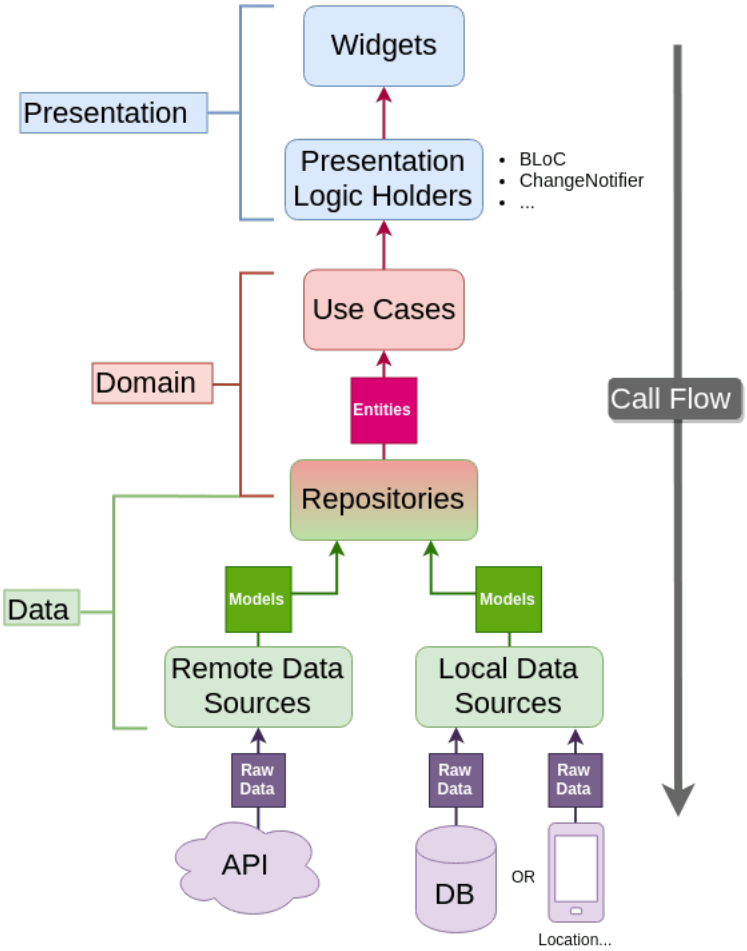
Com o modelo de arquitetura em camadas, o aplicativo desenvolvido pode ser independente de estrutura. A lógica de negócios não depende de bibliotecas ou estruturas específicas. No Flutter, as camadas de negócios do aplicativo são escritos apenas em Dart, sem o conhecimento da existência dessa lógica dentro de um app do Flutter.

Dessa maneira, com a lógica independente do Framework, a arquitetura do software melhora sua portabilidade, pois o código pode ser reutilizado em uma aplicação web por exemplo, ou até mesmo ser reescrito para outra linguagem como o JavaScript. Essa estrutura mantém a lógica de negócios limpa e permite que utilize do código e do fluxo do aplicativo para uma possível reestruturação do aplicativo.

2.2.3.2 Testabilidade

O Clean Arch no Flutter é altamente testável, pois é importante destacar a possibilidade de escrever testes mais eficientes (ARAUJO, 2019). Devido a separação completa da lógica de negócios da interface do usuário (do inglês User Interface, ou UI), banco de dados e outros

Figura 1 – Demonstração da arquitetura limpa no Flutter



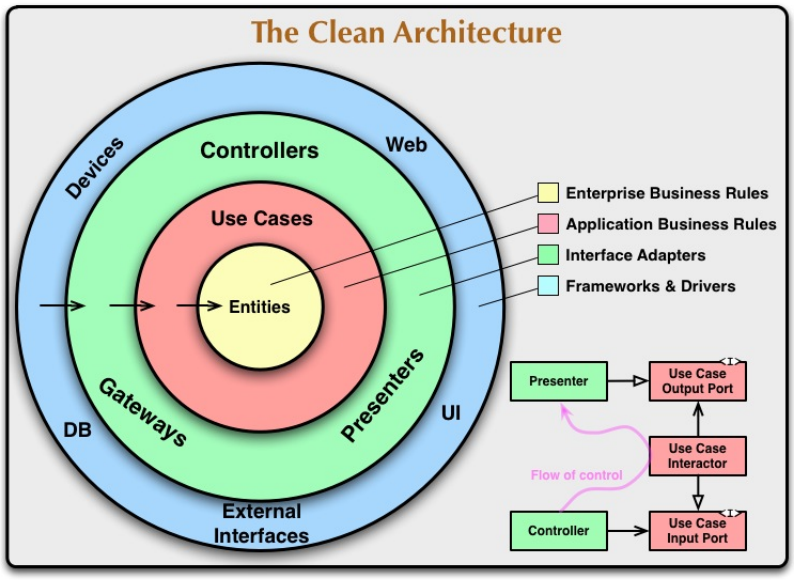
Fonte: Rešetár (2019).

elementos (Widgets), o teste se torna fácil pois as regras de negócios e a UI podem ser testadas independentemente. Ao testar as camadas internas, as dependências podem ser facilmente simuladas herdando das classes abstratas que são injetadas nas camadas internas. Isso deixa o teste isolado e confiável, independementemente da implementação.

2.2.3.3 Isolamento de módulos externos

Na arquitetura limpa, a camada de negócios (representada pela camada de domínio, a qual possui os casos de uso e as entidades) está no centro da arquitetura, e é onde está o "coração"da aplicação, totalmente isolada de implementações. Logo, a interface, banco de dados, e outros elementos, se tornam módulos externos do app e não devem afetar a lógica do negócio (o caso de uso).

Figura 2 – Camadas da arquitetura limpa segundo o Blog Clean Code

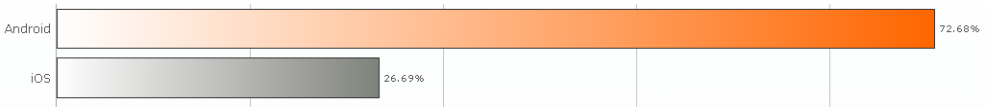


Fonte: Martin (2012).

2.3 Desenvolvimento para dispositivos móveis

Smartphones e tablets são itens que passaram a ter total relevância no dia a dia das pessoas, cada vez mais potentes e versáteis, estes pequenos dispositivos tem dominado o mercado de desenvolvimento de software. Esta oportunidade vem sendo abraçada por grandes empresas como Google, Amazon e Apple, para desbravar a grandeza dessa oportunidade de mercado. Porém quando se trata de desenvolvimento móvel, existem 2 principais sistemas operacionais em destaque: Android e iOS, que até março de 2021, possuiam 72,68% e 26,69% do mercado respectivamente (STATCOUNTER, 2021), como mostra a Figura 3.

Figura 3 – Participação dos sistemas Android e iOS no mercado de sistemas operacionais móveis do mundo (Março 2020 - Março de 2021).



Fonte: StatCounter (2021).

Quando se fala em desenvolvimento para dispositivos móveis, existem diversas maneiras e tecnologias que permitem que o desenvolvimento de aplicações, cada um com suas vantagens e desvantagens seja possível por exemplo fazer com que o mesmo código seja utilizado para as duas principais plataformas do mercado (MARKETEAM, 2020). A Tabela 2, mostra uma tabela comparativa dos principais tipos de tecnologias que são utilizadas no desenvolvimento móvel atualmente.

Mediante a isso, o melhor tipo de desenvolvimento para o projeto é o híbrido, pois necessitamos utilizar recursos do dispositivo, como GPS e possivelmente câmera em próximos

Tabela 2 – Comparativo quanto aos tipos de desenvolvimento mais utilizados atualmente.

	Nativo	Web App	Híbrido
Tempo	O projeto envolve a criação de mais de um código e por isso se torna um pouco mais longo.	Tem um projeto mais simples e por isso mais rápido	Semelhante ao web app
Preço	Pela complexidade e a necessidade de ser escrito em duas linguagens, se torna bem mais caro.	É relativamente mais barato por usar apenas um código e/ou layout	É mais barato por usar um código único.
Acesso a recursos	O desenvolvimento nativo conta com todos os recursos do dispositivo	Demanda mais tempo para configuração, além de necessitar de ser bem desenvolvido por questões de segurança	Semelhante ao app nativo;
Atualização	O usuário atualiza pela loja do dispositivo, precisa de download para ser executado	Dependendo de como for estruturado o aplicativo, na próxima vez que o usuário abrir ou recarregar o aplicativo ele será atualizado sem a necessidade de um novo download	O usuário precisa esperar o lançamento da atualização, semelhante ao nativo.
Experiência de usuário	A experiência é personalizada para cada sistema operacional (Android e iOS)	Devido a lentidões ou falhas no layout (Como é renderizado um site pode existir a "quebra" do layout padrão), trazendo uma péssima experiência do usuário	O código híbrido faz com que o aplicativo consiga trazer todas as particularidades (em comum) dos sistemas operacionais

Fonte: Autoria própria.

lançamentos, outro fator relevante também se deve ao curto prazo de tempo para entregar o aplicativo. Dois parâmetros que são importantes e relevantes para utilização da tecnologia.

2.3.1 Tipos de desenvolvimento móvel

Ao pensar em desenvolver um software para dispositivos móveis, algumas perguntas devem ser respondiadas. Entre elas está a pergunta: qual o melhor tipo de desenvolvimento para a minha aplicação? Um aplicativo nativo? Um Wep App? ou um aplicativo híbrido?

Para a responder essa pergunta, alguns fatores devem ser considerados. Cada método de desenvolvimento é diferente, e, é importante considerar quais recursos o aplicativo fornecerá aos usuários, isto porque esse fator afeta diretamente o tipo de desenvolvimento. Uma característica fundamental é que o aplicativo deve sempre fornecer uma boa experiência para o usuário, e neste caso, o tipo de desenvolvimento desempenha um papel de grande importância.

Essas três formas possuem tempos de desenvolvimento diferentes, logo deve-se analisar se o tempo de desenvolvimento é o adequado para o lançamento do seu aplicativo e se o orçamento está dentro do planejado. (MARKETEAM, 2020)

A seguir são apresentados detalhes acerca dos três modelos abordados neste trabalho: o desenvolvimento nativo, o desenvolvimento por meio de web app (também conhecido como SPA) e o desenvolvimento híbrido.

2.3.1.1 Desenvolvimento Nativo

Ao falar sobre aplicativos, a abordagem nativa é a mais tradicional. Este tipo é o mais comum nas lojas de aplicativos. São construídos em uma linguagem de programação específica para um determinado sistema operacional.

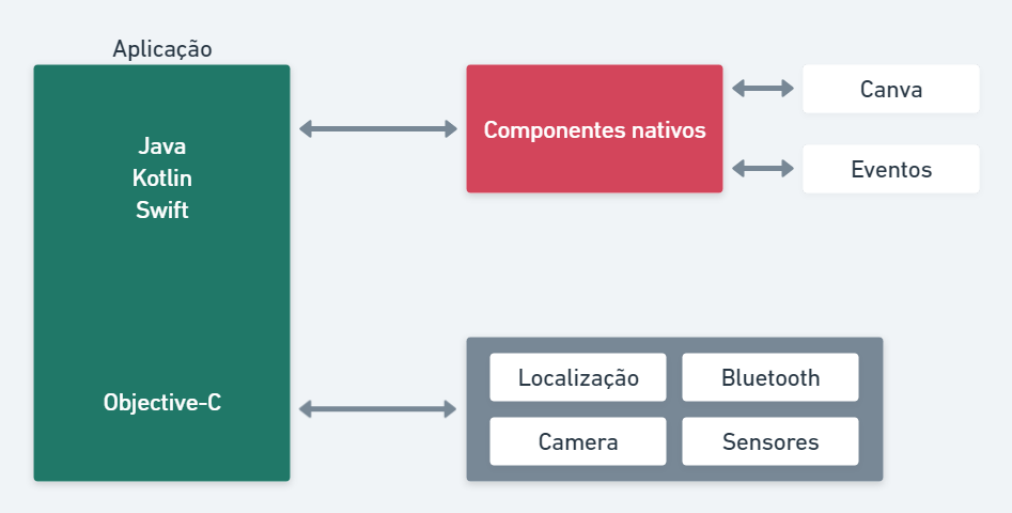
Como dito anteriormente, existem dois sistemas operacionais principais em smartphones: Android e iOS. A diferença entre eles não é apenas estética, pois um produto desenvolvido para um sistema não é adequado para o outro. Afinal, cada plataforma possui suas prórias ferramentas e elementos de interface. Os aplicativos nativos são programados nas linguages de seus respectivos sistemas, como Java e Kotlin no Android e Objective-C e Swift no iOS. Mas cada sistema também possui outras linguagens que podem ser utilizadas.

Por serem escritos especificamente para o sistema operacional, os aplicativos nativos são mais rápidos e confiáveis(MARKETEAM, 2020) do que outros aplicativos. Isso porque a abordagem nativa proporciona aos usuários uma experiência melhor quando eles necessitam usar as funcionalidades de um smartphone (como câmera, GPS ou notificações).

Ao programar aplicativos nativos, os desenvolvedores devem seguir padrões de *design*. Essas diretrizes são fornecidas para cada sistema operacional, que contém as melhores práticas que podem fornecer uma melhor experiência do usuário. Como exemplos de aplicativos móveis que utilizam a abordagem nativa tem-se o Whatsapp, Facebook Messenger, Waze e Uber.

Quanto a arquitetura de como funciona um aplicativo desenvolvido de forma nativa, a Figura 4 retrata como é que o programa se relaciona com a visão do aplicativo e acesso aos recursos do dispositivo. Cada aplicação desenvolvida em sua própria linguagem de programação, se comunica de maneira única com recursos internos dos dispositivos (chamados de serviços) e desenham os seus componentes visuais(widgets) de acordo com a plataforma do sistema operacional.

Figura 4 – Como a aplicação nativa utiliza os recursos visuais e acessa os recursos do dispositivo



Fonte: Singh (2018).

Como dito anteriormente, o aplicativo nativo apenas funciona na plataforma que ele foi desenvolvido. Caso a empresa opte pelo aplicativo em mais de uma plataforma, será necessário optar por um desenvolvimento que englobe tanto o Objective-C quanto o Java por exemplo. Como as tecnologias e peculiaridades de cada sistema operacional são diferentes, os custos para escolha desta abordagem tendem a ser mais elevados, visto que é necessário manter os aplicativos com duas equipes de desenvolvimento.

2.3.1.2 Desenvolvimento Web App ou SPA (Single Page Application)

O uso de dispositivos móveis se tornou um hábito e uma poderosa ferramenta em nossas vidas. Estima-se que o tempo médio de uso seja de quatro horas diárias usando as mais populares aplicações (Facebook, Instagram, Whatsapp, etc) disponíveis nas lojas de aplicativos ou navegando pela web (KHALAF; KESIRAJU, 2017). O Web App veio como uma maneira de ter a usabilidade para o usuário onde ele deixe de depender de um sistema operacional do aparelho e possa utiliza em qualquer navegador que estiver disponível, tanto no aparelho telefônico ou em um computador.

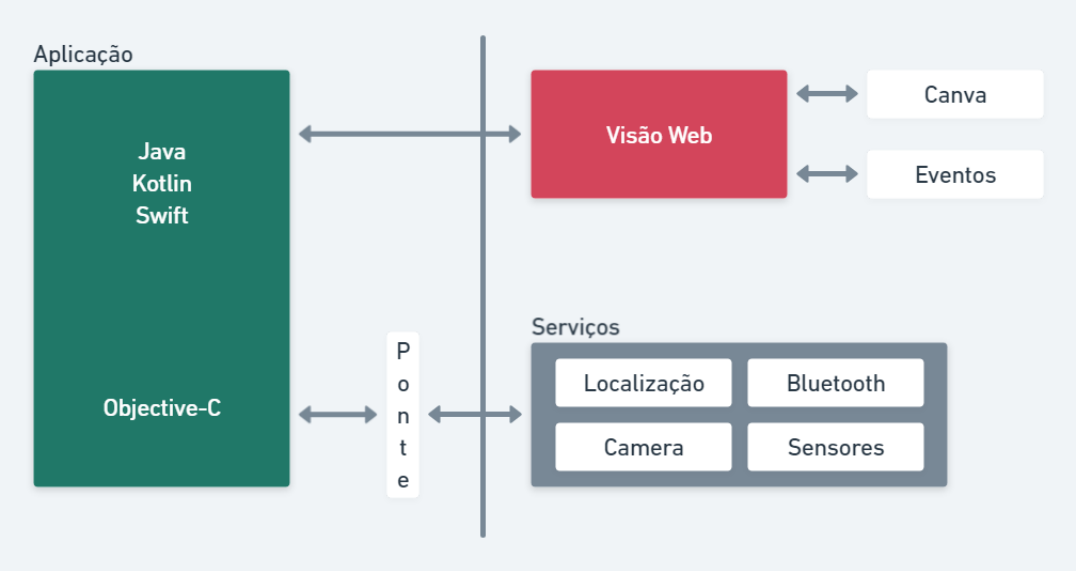
Um Web App não é uma aplicação desenvolvida com o propósito de ser móvel, e sim uma visão de navegador (web view), que transforma uma página web em um aplicativo, sendo esta desenvolvida utilizando o princípio *mobile first*, ou seja, com o foco principal nas aplicações móveis (que tenham telas em dimensões menores) e não web (aplicações para monitores maiores como por exemplo 1920x1080 pixels). Quando a ideia é apenas apresentar conteúdo ou ter um dispositivo móvel online, a abordagem por meio de Web App se torna uma opção mais barata, que não demanda treinamento de uma equipe específica para o desenvolvimento móvel, a equipe de desenvolvimento web desenvolve o aplicativo por completo, utilizando tecnolgias como HTML5, CSS e Javascript.

É importante considerar que o Web App requer uma conexão com internet constante para funcionar. Além disso, algumas funções nativas do dispositivo móvel podem apresentar lentidão ou não funcionar (como localização, câmera, gravação de áudio, etc).

A arquitetura de web app se apropria de uma Web View para mostrar os componentes para o usuário, e quanto aos recursos físicos do dispositivo ele utiliza uma “ponte” para fazer o acesso. Por este motivo, o aplicativo costuma ter lentidões excessivas. A Figura 5 apresenta um diagrama que descreve a arquitetura baseada em Web App.

Este tipo de arquitetura traz aplicativos mais lentos que os nativos, porque não estão integrados ao sistema operacional. Como exemplo de um Web App pode-se citar o aplicativo para suporte à aprendizagem de instrumentos musicais CifraClub.

Figura 5 – Como uma aplicação desenvolvida com JavaScript, HTML e CSS acessa os recursos em dispositivos móveis.



Fonte: Singh (2018).

2.3.1.3 Desenvolvimento Híbrido

Os aplicativos desenvolvidos com tecnologia híbrida são desenvolvido por meio de linguagens e ferramentas que permitem que o mesmo código seja usado no iOS e Android. Dessa forma, é possível aumentar a produtividade e ter uma curva de aprendizado muito menor em relação ao desenvolvimento nativo.

Inicialmente os aplicativos desenvolvidos com tecnologias híbridas não apresentavam um desempenho compatível com os aplicativos nativos, e isso se devia ao fato de que a tecnologia estava em experimentação, e aplicativos desenvolvidos com tal tecnologia apresentavam problemas de uso (MALINOSQUI, 2019). Muitas funcionalidades nativas por exemplo não funcionavam e a performance era infinitamente inferior.

Essa mudança no desenvolvimento híbrido se deve ao fato de que a tecnologia *webview* começou a ter uma performance melhor nos dispositivos móveis. Porém as tecnologias de desenvolvimento híbrido hoje não funcionam apenas baseados em *webview*. Frameworks como ReactNative, NativeScript e Flutter nos permitem utilizar o mesmo código para ambas as plataformas, gerando os componentes como nativos,o que nos leva a uma experiência semelhante a obtida com o desenvolvimento nativo. (MALINOSQUI, 2019)

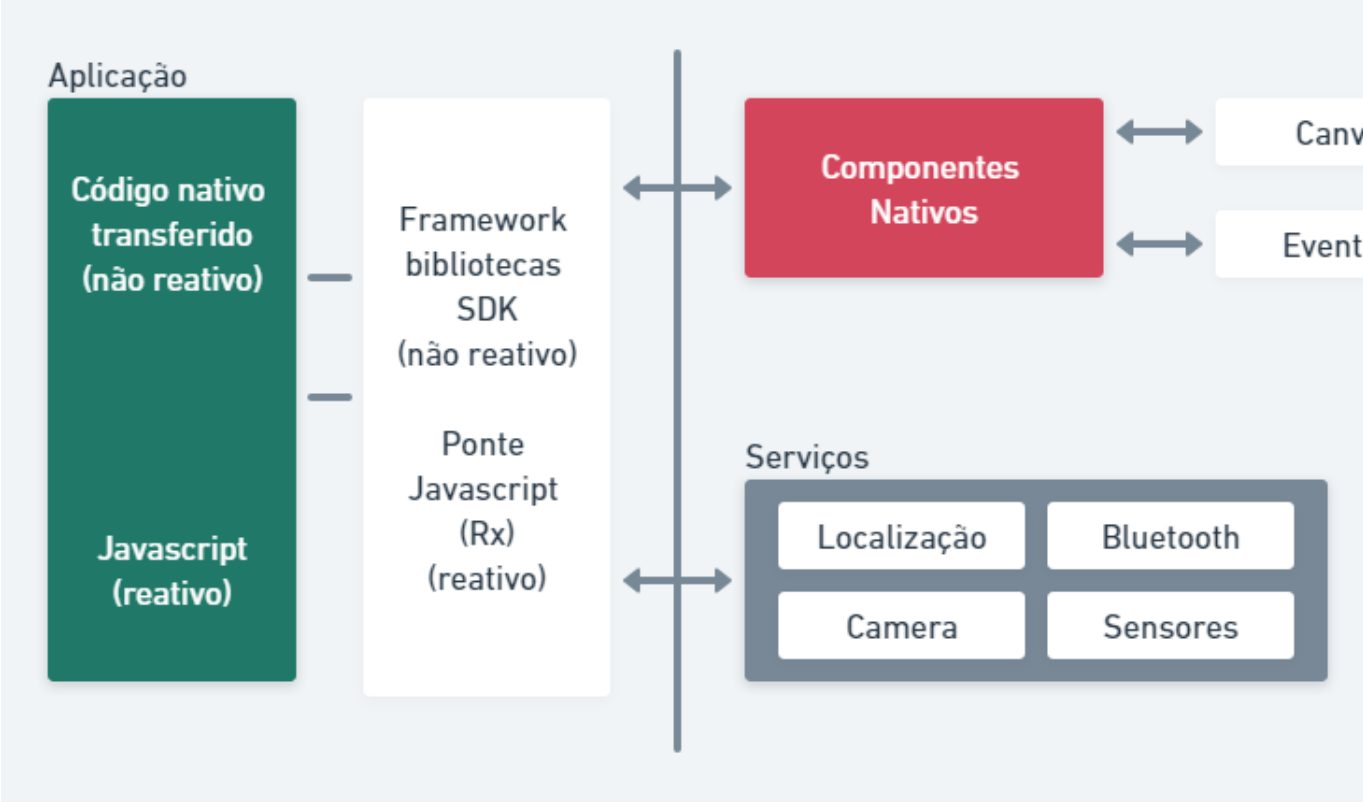
A partir dessa evolução, a tecnologia híbrida começou a ser utilizada por grandes empresas no desenvolvimento de suas plataformas. Um exemplo é o *Nubank* que deixou totalmente suas duas aplicações nativas para uma única aplicação híbrida.

2.3.2 Frameworks Híbridas

Em 2019 o mercado de frameworks que possibilitavam o desenvolvimento híbrido era grande (GALANTE, 2019), e as opções variavam de acordo com os gostos e necessidades do desenvolvedor e do projeto. Para este projeto, foram escolhidas 3 frameworks relevantes, elas são: React Native, NativeScript e Flutter.

Na Figura 6 demonstra como um desenvolvimento híbrido multiplataforma funciona, com exceção da Framework Flutter.

Figura 6 – Como funciona uma aplicação desenvolvida de maneira híbrida em dispositivos móveis.



Fonte: Singh (2018).

Conforme exposto anteriormente, o resultado obtido a partir do uso desse tipo de desenvolvimento se assemelha bastante ao nativo. O framework híbrido realiza chamadas aos componetes nativos (Widgets), e a renderização destes componenetes em tela segue o padrão do sistema operacionais de execução. A seguir são apresentados os frameworks ReactNative, NativeScript e Flutter, que utilizam a abordagem de desenvolvimento híbrida.

2.3.2.1 React Native

É um dos frameworks multiplataforma mais populares, desenvolvido e mantido pelo Facebook, atualmente a maior empresa com foco em redes sociais, sendo proprietário, além do

Facebook, do Instagram e Whatsapp. Embora tenha sido criado em 2013, tornou-se recentemente uma das opções preferidas para desenvolvedores de aplicativos móveis. React Native é basicamente uma estrutura de código aberto que fornece um amplo suporte para IDEs e outras ferramentas de desenvolvimento de aplicativos. É uma das frameworks Javascript para criar aplicativos nativos para plataformas Android e iOS, e você ainda pode reaproveitar muito código utilizando o React para desenvolver o layout da sua aplicação na web. Exemplos de aplicações que utilizam React Native são: Facebook, Skype e o Airbnb

2.3.2.2 NativeScript

É um framework open source que utiliza preferencialmente a linguagem Javascript para desenvolver multiplataforma. Além disso, você pode utilizar CSS para a interface do aplicativo. Foi criado em 2014 pela Progress, hoje é conhecida como Telerik AD. Os aplicativos mais conhecidos da framework são: Raiffeisen Bank (instituição financeira européia) e Puma.

2.3.2.3 Flutter

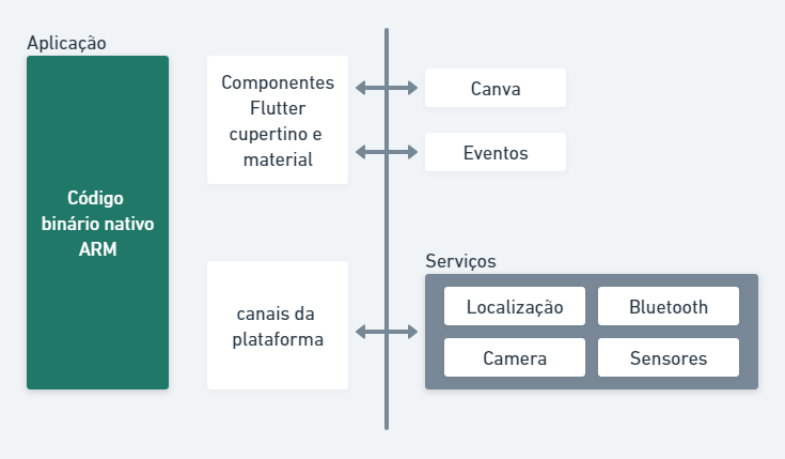
O Flutter foi lançado em 2017 pela Google e é um SDK de código aberto para o desenvolvimento de aplicativos escrito na linguagem Dart. Quando se trata de escolha de Framework para dispositivos multiplataforma, este é o mais adequado, pois além de Android e iOS, também é possível desenvolver para web, desktop (Windows, Linux e Mac), e embarcados (FLUTTER, 2021).

O framework faz o uso do mecanismo de renderização 2D chamado Skia, que utiliza o potencial do processador para renderizar os componentes nativos de maneira mais rápida ao criar e descartar os widgets(FLUTTER, 2019), algo que não existe "embarcado"nas outras frameworks, sendo necessário uma implementação de um mecanismo próprio para liberar os recursos visuais. Outro fator que facilita muito no desenvolvimento e na curva de aprendizado comparando o Flutter com as outras tecnologias é o fator de não precisar compilar toda a aplicação a qualquer alteração no código, pois ele conta com o hot reload (implementado desde a versão beta), que consiste em: quando o aplicativo é iniciado para depuração, toda alteração no arquivo é renderizada para a aplicação em tempo real, permitindo assim um desenvolvimento mais rápido e controlado.

Quanto a arquitetura do Flutter, embora seja um tipo de desenvolvimento híbrido, ele utiliza diretamente o Canvas para desenhar os seus Widgets, fazendo com que ganhe desempenho comparado a um tipo de desenvolvimento híbrido "padrão".

Com isso em mente, conclui-se que há espaço para a criação de um novo aplicativo para o mercado de encomendas utilizando as tecnologias: Firebase no backend e Flutter no Frontend para atender a demanda de dispositivos Android e iOS, utilizando a arquitetura limpa para a

Figura 7 – Imagem demonstrando com uma aplicação desenvolvida em Flutter funciona em dispositivos móveis.



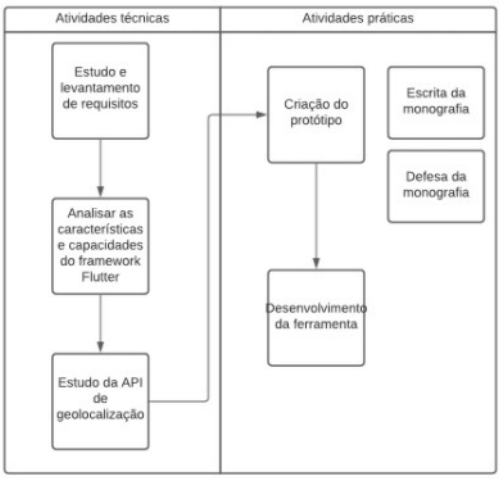
Fonte: Singh (2018).

estrutura da aplicação ficar com baixo nível de acoplação, permitindo assim a capacidade de escalabilidade da aplicação.

3 METODOLOGIA

O projeto foi desenvolvido com cinco atividades de desenvolvimento e estudo, e duas atividades de apresentação do trabalho desenvolvido, cada atividade está contida em duas categorias, sendo uma delas técnicas e a outra prática conforme a metodologia apresentada na Figura 8.

Figura 8 – Imagem que demonstra o fluxo de trabalho para o desenvolvimento do projeto.



Fonte: Autoria própria.

As atividades seguiram a ordem da imagem para obter um trabalho de qualidade e garantia de que todos os processos necessários para o desenvolvimento sejam realizados.

3.1 Tecnologias

Definida a estrutura e as ferramentas, o projeto foi desenvolvido usando como linguagem de programação o Dart (que é a linguagem utilizada pelo Flutter), utilizando os fundamentos explicados anteriormente, como a programação orientada a objetos e a arquitetura limpa. Quanto ao armazenamento do aplicativo, será utilizado o Firebase, que é uma plataforma de *Baas* que fornece outras funções como login, analitycs e monetização através de anúncios.

3.1.1 Flutter

O Flutter foi escolhido para o desenvolvimento por vários motivos, entre eles a capacidade de desenvolver para multiplataformas, podendo escalar o mesmo código para outros tipos de dispositivos e plataformas como web e desktop. Devido a versão 2, lançada no ano de 2021, ela permite a criação de aplicativos nativos para até 5 sistemas operacionais: iOS, Android, Windows, MacOS e Linux. Além de a possibilidade de ser utilizado na web para navegadores como Chrome, Firefox, Safari e até mesmo o Edge. Nesta versão também é possível desenvol-

ver para embarcados, como por exemplo a Toyota que esta utilizando da arquitetura do Flutter pois facilita a incorporação em seu ambiente de destino ao compilar o mecanismo e envolvê-lo em um embarcado. O objetivo da Toyota é ter um fluxo de trabalho que permita que as ferramentas de design gerem código para executar e validar o software imediatamente (GOOGLE, 2021). O Framework utiliza a linguagem Dart, que foi desenvolvida pelo Google, orientada a objetos e pode ser compliado para código nativo ou Javascript (BRACHA, 2015). A sintaxe se assemelha bastante ao Javascript, porém podendo ser utilizado com classes e outros princípios da orientação a objetos, como interfaces e classes abstratas. No ano de 2022, a ferramenta chegou a versão 3.0 totalmente estável e testada, novas bibliotecas precisaram de atualização, pois essa versão permite o desenvolvimento para dispositivos dobráveis (foldable) e alguns padrões de desenvolvimento novos chegaram também no Dart, como enum com parâmetros.

3.1.2 Firebase

O Firebase é uma plataforma de BaaS, que permite que aplicações possam ser desenvolvidas de maneira centralizada, como login e armazenamento em um lugar só. Embora ele seja um serviço pago, o projeto utilizará a versão gratuita para os testes e desenvolvimentos iniciais. O Flutter já possui pacotes que permitem a integração do aplicativo com o Firebase de maneira rápida e consolidada e agora com a versão 3.0, ficou ainda mais fácil. O Firebase também permite a atualização em tempo real da aplicação a partir de qualquer alteração no banco de dados.

O armazenamento do Flutter foi realizado utilizando-se do Realtime Database, que organiza os dados em uma árvore como um arquivo JSON. A estruturação definida é apresentada a seguir e mais detalhes são apresentados na Seção 4.3.

- **delivery**, as entregas cadastradas;
- **drivers**, motoristas cadastrados e também definem o tipo do usuário;
- **users**, usuários genéricos da plataforma que enviam a encomenda;
- **drivers_working**, esta funciona como uma "tabela intermediária"onde ficam as localizações instantâneas dos motoristas que pegaram a encomenda para envio.

3.1.3 API de Geolocalização

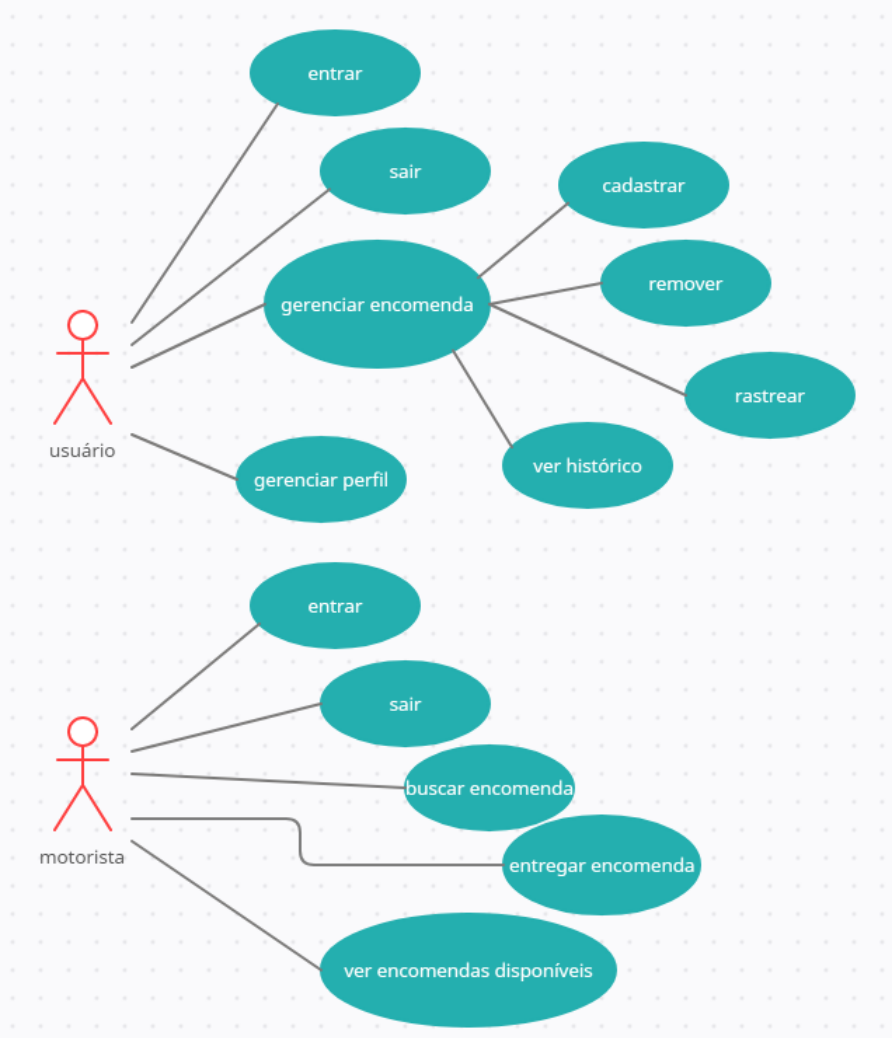
Para utilizarmos a API de geolocalização, é necessário a criação da conta no Google Cloud Platform (GCP), para liberar uma chave de acesso para as requisições do aplicativo para essa ferramenta. Assim como o Firebase, a API tem uma versão gratuita que permite até 1000 solicitações mensais. Ela é dividida em duas partes, sendo a primeira a da renderização dos

mapas, contendo endereços e lugares, e a segunda sendo as rotas, que contém a previsão de horário de chegada, e qual a melhor rota para seguir que também calcula a distância e o tempo necessário, essa API é atualizada e gerenciada pelo próprio Google.

3.2 Diagrama de caso de uso

Como a arquitetura do nosso aplicativo é o Clean Arch, o melhor diagrama para representar o software é o de caso de uso, já que muitas das vezes as classes englobam as entidades e modelos da nossa arquitetura e geraria uma certa complicação e dificuldade. O diagrama de caso de uso da Figura 9 descreve as atividades a realizadas pelos dois usuários do sistema, definidos como usuário, aqueles que enviam as encomendas, e motorista, aquele que realiza a entrega dessa encomenda enviada.

Figura 9 – Diagrama de caso de uso para o projeto.



Fonte: Autoria própria.

Para o usuário as funções são as seguintes:

- Entrar, ou seja, realizar o login na plataforma, também foi definido o login automático para a plataforma.
- Sair, realizar a saída do sistema, removendo os dados salvos como email e senha.
- Gerenciar a encomenda, que da uma certa autonomia para o usuário para a encomenda dele, onde ele pode visualizar o seu histórico de encomendas, cadastrar uma encomenda nova, remover uma encomenda caso o motorista não selecionou ela ainda, e rastrear a localização dessa encomenda (ou a última localização registrada).
- Gerenciar perfil, funcionalidade básica para identificação, contato e foto do usuário cadastrado.

Do mesmo modo, para o motorista, as funções exercidas por ele:

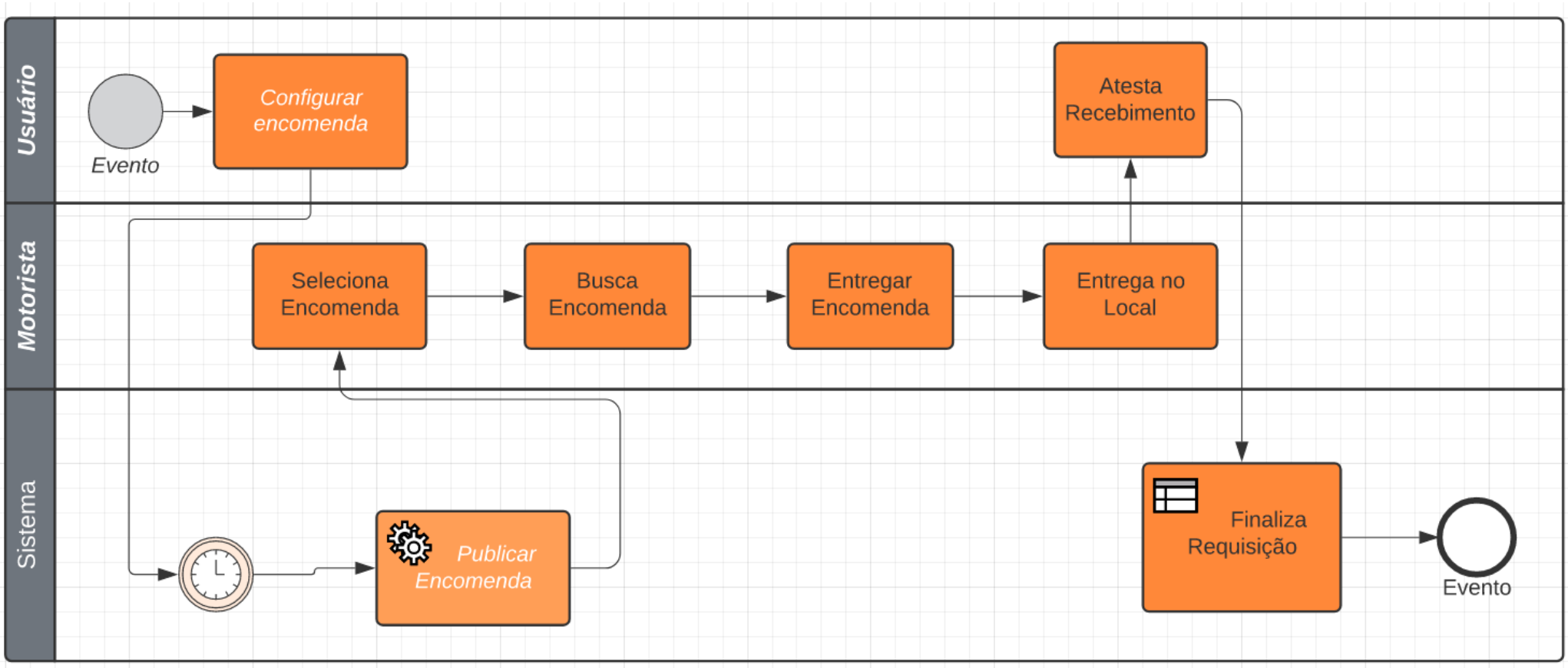
- Entrar, ou seja, realizar o login na plataforma, também foi definido o login automático para a plataforma.
- Sair, realizar a saída do sistema, removendo os dados salvos como email e senha.
- Gerenciar a encomenda, que da autonomia para o motorista para selecionar a encomenda dele, levar e entregar, onde ele pode visualizar o seu histórico de encomendas entregues e disponíveis, selecionar a encomenda que deseja enviar, buscar a encomendae entregar.

3.3 Diagrama de Fluxo

O diagrama de fluxo da Figura 10 representa o fluxo desde o momento que o usuário publica a encomenda até a entrega. Neste diagrama existem 4 atores (raias) que são definidas em Usuário 1 (o remetente), aquele que configura o pacote a ser enviado e efetua o pagamento da entrega. O Usuário 2 (o destinatário), que pode ou não existir durante o processo padrão de entrega, é o personagem que vai receber a encomenda. O motorista (entregador), é o personagem que leva o pacote até o destino. E por último o sistema, que gerencia o processo de disponibilização dos pacotes e a finalização do processo de entrega.

Configurar encomenda é o processo inicial do envio realizado pelo usuário 1 aonde ele define as opções da encomenda, como endereço onde buscar, para onde levar, os dias e horários disponíveis para buscar e caso queira pode escolher um motorista específico e até mesmo um usuário a quem deve receber o pacote.

Figura 10 – Diagrama de fluxo para o projeto



Fonte: Autoria própria.

Envio para outro usuário é uma decisão do usuário 1 caso ele queira enviar sua encomenda para um outro usuário que esteja cadastrado no aplicativo. Caso sim, ele seleciona o seu usuário destinatário, caso contrário, ele informa o nome e o local para envio.

Aceita encomenda, quando o usuário 1 envia sua encomenda para outro usuário, o usuário 2 deve aceitar a encomenda para que essa publicação seja publicada pelo sistema para ser visualizada pelos motoristas da plataforma.

Publicar encomenda para motoristas, nesse processo, o sistema obtém as informações dos usuários e publica essas informações para que os motoristas cadastrados possam ver (caso a encomenda não possua um motorista determinado), e também permite que os motoristas escolham seus envios.

Seleciona encomenda, quando o sistema publica a encomenda, isso permite que o motorista possa escolher uma encomenda de acordo com a sua preferência.

Valida motorista, após a seleção do motorista, o aplicativo deve notificar o usuário 1, para que ele visualize o perfil do motorista e realize o aceite do mesmo. Caso ele aceite, o motorista está apto para buscar e enviar a encomenda, do contrário o sistema deve realizar a publicação da encomenda novamente.

Busca encomenda, após a aprovação do usuário, o motorista está apto a realizar a busca e entrega da encomenda.

Entrega da encomenda, atividade realizada após a busca, monitorada pelo aplicativo trazendo dados como tempo previsto para entrega e o valor.

Destinatário definido esta decisão é realizada pelo motorista para realizar a entrega. Caso o pacote tenha um usuário definido, ele necessita de uma validação do usuário para realizar a entrega, se não, ele deve entregar no local registrando com uma assinatura da pessoa que recebeu.

Entregar para o usuário, conforme dito anteriormente, é um processo de entrega para o usuário 2 para que ele possa validar e liberar o motorista.

Entregar no local, essa atividade é realizada pelo motorista quando não possui um destinatário como um usuário do aplicativo. Ele necessida de uma assinatura da pessoa que recebeu a encomenda.

Atesta recebimento, caso o motorista entregue para um usuário definido, o usuário deve atestar o recebimento para liberar o motorista (Não finalizada).

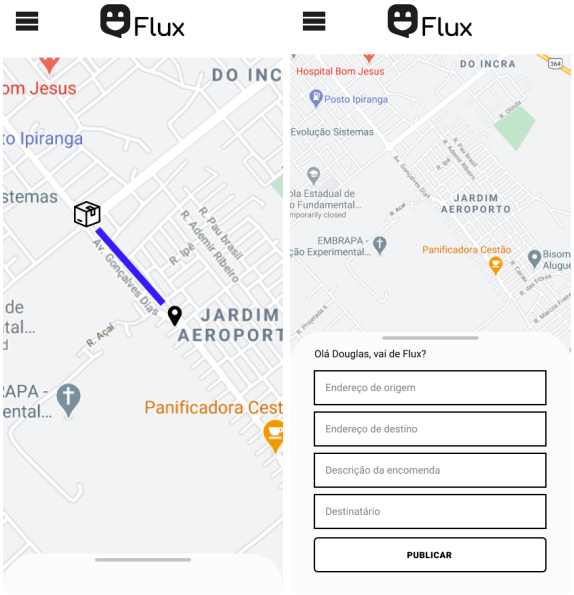
Finaliza requisição é o processo final, realizado pelo sistema após a confirmação de entrega (Não finalizada).

Avalia motorista é um processo realizado pelos 2 usuários e garante uma média de nota mais alta para o motorista. Não é uma atividade obrigatória (Não finalizada).

3.4 Prototipagem

O desenvolvimento do visual do protótipo, feito no Figma (software de prototipagem), foi desenvolvido para ter uma boa usabilidade e design agradável, na Figura 11 mostra uma tela inicial da aplicação, mostrando o mapa e se tiver uma encomenda em curso, ele mostra o trajeto da encomenda. e também a opção de configurar e publicar uma nova encomenda.

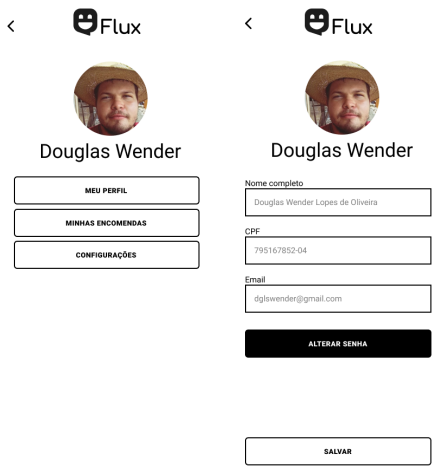
Figura 11 – Protótipo das telas iniciais do projeto.



Fonte: Autoria própria.

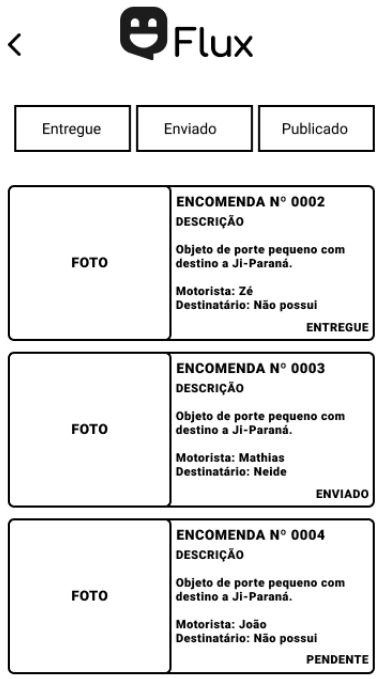
Foi definido também padrões de botões, campos de texto e cores para o projeto, focado na clareza e no objetivo de cada ferramenta visual, conforme as imagens das Figuras 12 e 13.

Figura 12 – Protótipo dos menus do projeto



Fonte: Autoria própria.

Figura 13 – Protótipo da lista de envios do projeto.



Fonte: Autoria própria.

4 RESULTADOS

Como resultado do projeto foram desenvolvidos dois aplicativos, sendo um para os motoristas e outro para os usuários que vão enviar as encomendas.

4.1 Flux Client - App para os usuários

Este é o aplicativo para os usuários de "mar aberto", que podem baixar, instalar e utilizar, é o principal aplicativo do nosso sistema e é responsável por configurar a localização de origem e destino da encomenda, realizar o cálculo do valor do serviço e salvar a encomenda publicada.

4.1.1 Login

Realizado com o *Firebase Authentication*, que é um sistema próprio da Google para gerenciar a entrada e sessões diretamente pelo Firebase. Nessa tela do aplicativo, confirme a Figura 14(a), ela contém dois campos de texto (email e senha) e dois botões (login e cadastro)

4.1.2 Registro

Também utiliza a API do Firebase de Autenticação, permitindo também os registros, porém foi adaptada para que na hora de registrar pelo app, gere também um registro no banco de dados com os dados do usuário, como nome e telefone. A Figura 14(b) demonstra como ficou o formulário de registro.

4.1.3 Home, publicar encomenda e procurar endereços

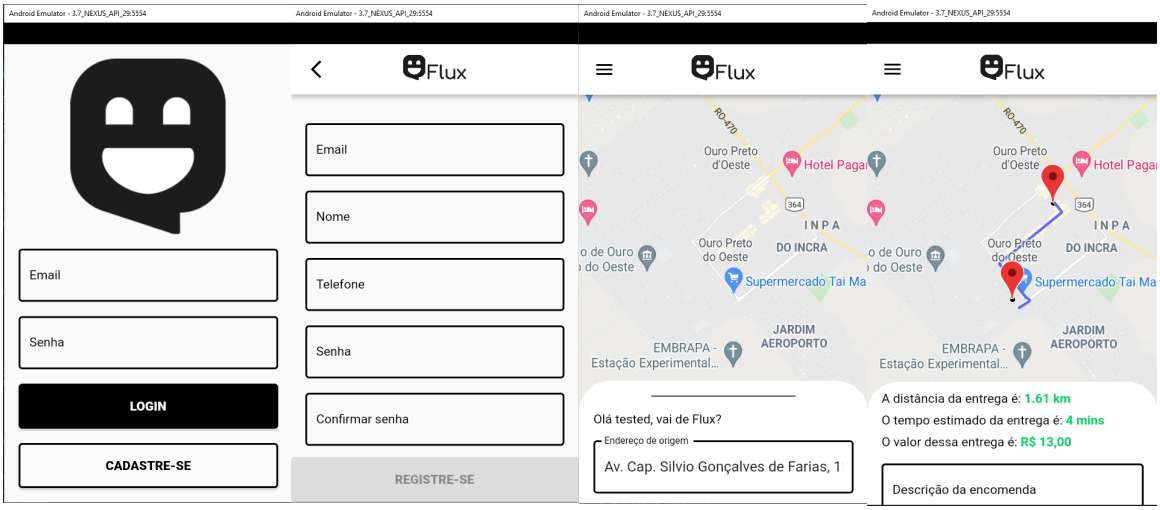
Parte principal do aplicativo, que inicializa com a localização de origem onde o dispositivo se encontra no momento. Junto a essa tela da Figura 14(c), há um formulário para enviar uma nova encomenda, e após selecionar os dois endereços (origem e destino) um demonstrativo do valor da encomenda, distância e tempo estimado conforme apresenta a Figura 14(d) . Ao tocar nos endereços tanto de origem como de destino, o aplicativo redireciona para outra tela, que é a de pesquisa de locais, onde ele auxilia no endereçamento para garantir o ponto de latitude e longitude com precisão disponibilizada pelo Google.

4.1.4 Menu e configurações

Uma tela onde ficam as opções do menu principal, como acessar as configurações da conta e as encomendas publicadas. Na Figura 15(a) pode-se observar a imagem de perfil, e dois

Figura 14 – Telas da aplicação Flux Client.

(a) Login **(b) Registro** **(c) Início** **(d) Valores**



Fonte: Autoria própria.

botões, um das configurações que leva para a tela da Figura 15(b), onde se apresentam os dados da conta e a opção de "sair"que remove o acesso a conta naquele dispositivo fazendo necessário realizar o acesso novamente.

4.1.5 Lista de encomendas e encomenda

Uma tela que traz todas as encomendas publicadas e o histórico de encomendas já entregues para o usuário logado, a Figura 15(c) mostra como é essa lista, com uma imagem, o seu código, quem é o motorista responsável e o nome do destinatário. Ao selecionar a encomenda desejada, o aplicativo exibe a tela da Figura 15(d) que tem mais informações referentes a encomenda selecionada como o destino, o valor da corrida e o endereço de origem.

4.2 Flux Rider - App para os motoristas

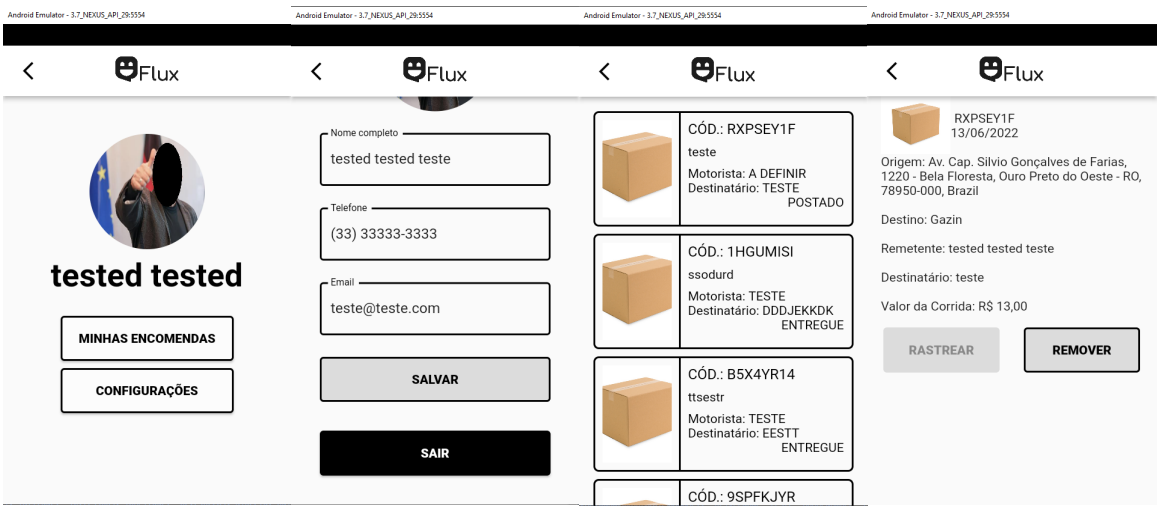
Para o sistema estar em total funcionamento, se faz necessário uma aplicação a parte para enviar a localização da encomenda e permitir também que o motorista escolha a sua entrega para realizar. Dito isto, as subseções a seguir demonstram as telas do aplicativo Flux Rider, o aplicativo para os entregadores.

4.2.1 Login

Da mesma maneira que o login para os usuários padrões, os motoristas também possuem seus logins na mesma base do Firebase Authentication, porém dentro do Firebase Database,

Figura 15 – Telas da aplicação Flux Client.

(a) Menu **(b) Configurações** **(c) Lista de Encomendas** **(d) Publicação**



Fonte: Autoria própria.

onde é armazenado os dados, é uma coleção diferente para possuir essa diferença entre os usuários e não permitir que o usuário normal tente entrar no aplicativo do motorista. Na Figura 16(a) pode-se observar a semelhança com a tela de login do outro aplicativo.

4.2.2 Registro

Diferente do registro da aplicação principal, para o motorista se faz necessário mais informações como modelo do carro, a cor, placa e seu número de telefone para o seu cadastro é um formulário extenso, conforme a Figura 16(b), que também exige passar por uma avaliação (por fotos) do veículo e documentos do usuário para a liberação do usuário.

4.2.3 Home

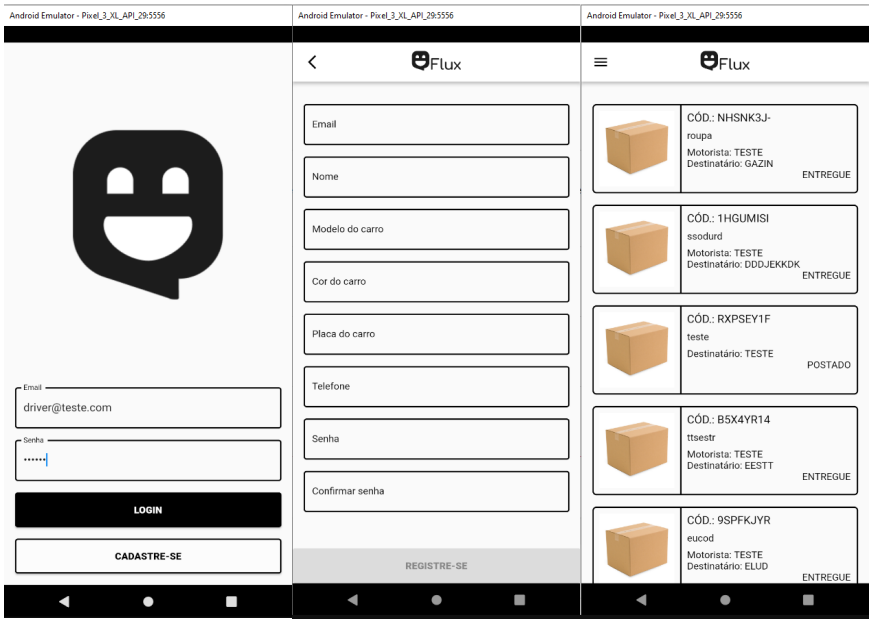
Diferente da home do aplicativo para os usuários, o do motorista não necessita de um mapa na parte principal do aplicativo, apenas a lista com as encomendas disponíveis e a lista com encomendas já entregues. A Figura 16(c) demonstra a lista de encomendas que já foram entregues.

4.2.4 Encomenda

É onde se encontra os detalhes da encomenda, como endereços e valores. Como demonstrados na Figura 17(a) são utilizados botões para buscar, onde abre o mapa instalado (preferencialmente do Google) para criar a melhor rota do local atual do motorista para o endereço

Figura 16 – Telas da aplicação Flux Rider.

(a) Login (b) Registro (c) Home



Fonte: Autoria própria.

de origem, e o botão levar, que traça a melhor rota para o endereço de destino. Durante o mapa aberto, a aplicação não pode ser finalizada de maneira "natural"por limpeza de memória por exemplo, apenas forçando a finalização. Quando ele volta do mapa do botão "levar"a aplicação pergunta se a entrega foi finalizada (Figura 17(b)) e caso sim, há alteração no banco de dados e ela sai do status de "a caminho"para "entregue". Vale lembrar que depois que a encomenda foi buscada, ela não pode ser cancelada.

4.2.5 Menu

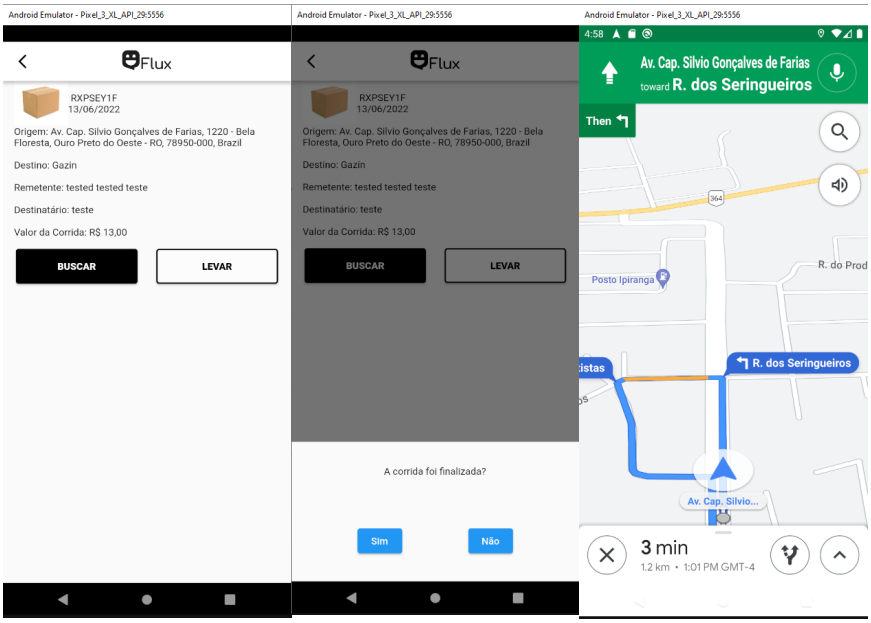
Esse menu só fica uma imagem (que seria mais apropriado ser a do veículo ou do motorista), e o botão sair para encerrar a atividade da conta naquele dispositivo.

4.3 Banco de Dados

Conforme planejado, o armazenamento de dados foi feito totalmente no Firebase Database, que é um banco de dados NoSQL que nos permite escalar a aplicação facilmente e também tem um tempo de resposta rápido para aplicações que usam dados móveis. A estrutura ficou definida como um campo create que permite a criação de novos registros no banco, uma coleção chamada de delivery que é onde se encontram as encomendas agrupadas pelo usuário, outra coleção chamada drivers que são os registros de motoristas, a terceira é a de *"drivers_working"*

Figura 17 – Telas da aplicação Flux Rider.

(a) Entrega (b) Verificação (c) Rota



Fonte: Autoria própria.

que mostra quais motoristas estão trabalhando e a sua localização em tempo real, e a coleção users que é o armazenamento dos dados dos usuários cadastrados no sistema.

4.3.1 Usuário

O modelo do usuário ficou bem simples, dado que também foi utilizado parte do login o Firebase Authentication para armazenamento e gerência de senhas. A Figura 20 representa um modelo de como ficam armazenados os usuários no Firebase Database.

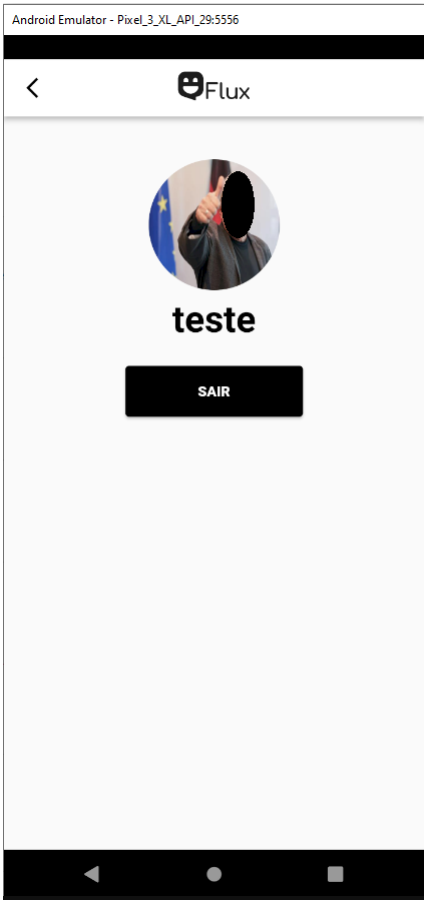
4.3.2 Motorista

No modelo do motorista ficam armazenados juntamente os dados do veículo do mesmo, como cor do carro, placa e modelo. Também foi utilizado o login do Firebase Authentication então não se faz necessário o armazenamento da senha. A Figura 21 representa um modelo de como ficam armazenados os motoristas no Firebase Database.

4.3.3 Entrega

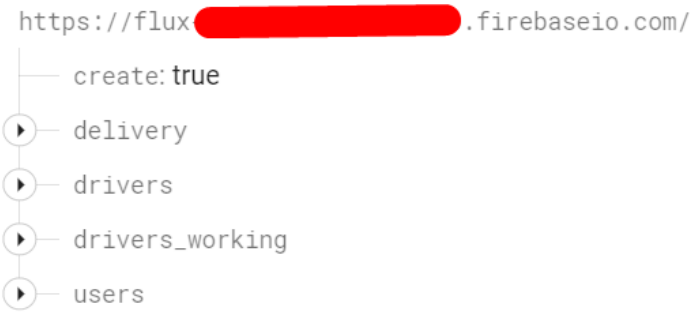
Este é o modelo central da aplicação, onde ficam armazenados os dados da encomenda publicada pelo usuário e também o motorista responsável (caso ela já foi aceita por algum). A Figura 22 demonstra todos os campos preenchidos do modelo, o primeiro é "created_at"que é o

Figura 18 – Menu da aplicação Flux Rider



Fonte: Autoria própria.

Figura 19 – Arquitetura da aplicação Flux



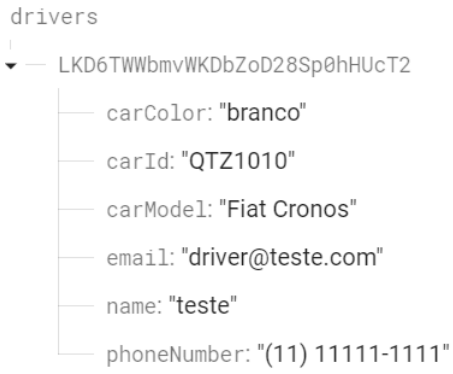
Fonte: Autoria própria.

horário que o registro é inserido no banco de dados, os próximos três campos com "delivery"no início dizem respeito a encomenda, o que ela é, um documento da pessoa que está enviando a encomenda e o nome ou email do usuário que vai receber essa encomenda. Os dois campos de "destination"dizem respeito ao destino da encomenda, o endereço descrito, e a localização que é o ponto dele no mapa com latitude e longitude. Os campos que tem "driver"são os campos que ligam essa encomenda a um motorista específico, caso estes campos estejam em brancos quer dizer que a encomenda ainda não foi selecionada para "buscar"e no caso ainda pode ser

Figura 20 – Modelo de usuário da aplicação Flux

Fonte: Autoria própria.

Figura 21 – Modelo do motorista da aplicação Flux



Fonte: Autoria própria.

cancelada. Os campos *"origin"* seguem o mesmo padrão de *"destination"*. O *"payment_method"* é um campo que ainda não foi implementado, pois na aplicação ainda não é possível utilizar o cartão de crédito para realizar o pagamento. Os campos *"request"* são referentes ao usuário que está requerindo a entrega, e são em ordem: o número de telefone, o código de identificação do usuário e o seu nome de usuário. Por fim os últimos campos são o status, que hoje são 3: *waiting*, que é o de espera e pode ser excluído, *on_road* que é quando o motorista está indo buscar a encomenda, *on_delivery* quando o motorista inicia a entrega e por fim *finished*, quando a encomenda chega ao destino final. E o *"value_of_run"* que representa o valor da corrida em centavos.

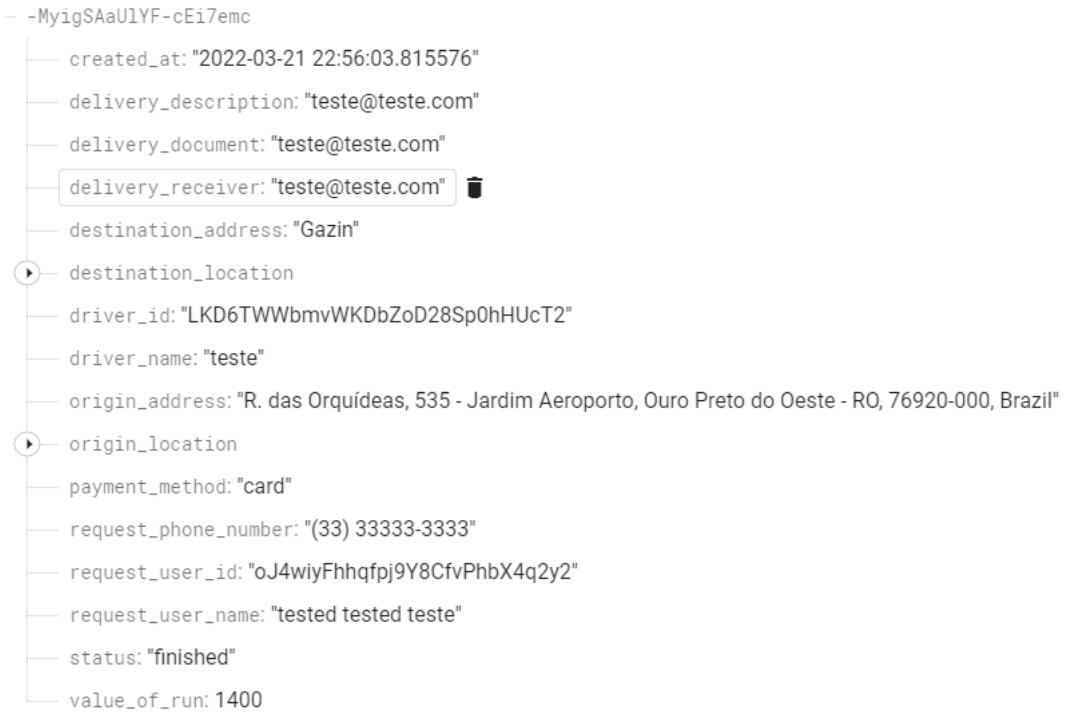
4.3.4 Motoristas entregando

Esta é uma coleção a parte para ter menos dados e economizar os dados na hora de fazer as requisições, e ela tem apenas um código de localizador chamado "g", e sua latitude e longitude. Esse padrão é o convencional do próprio Google Geofire, que faz a atualização sempre que o dispositivo se movimenta.

4.4 Repositório

Todo o código do projeto, ou seja, das duas aplicações, é público e está disponível no Github através do link: https://github.com/douglaswender/flux, para executar o projeto é ne-

Figura 22 – Modelo da entrega da aplicação Flux



Fonte: Autoria própria.

Figura 23 – Modelo da posição do entregador da aplicação Flux

Fonte: Autoria própria.

cessário a configuração de algumas chaves privadas que são disponibilizadas pelo Google, em breve serão disponibilizados atualizações neste repositório com a documentações para realizar essa configuração inicial do projeto.

4.5 Vídeo Demonstrativo

Também foi disponibilizado um vídeo onde é demonstrado a execução da funcionalidade principal da aplicação, que é o rastreio em tempo real do motorista responsável pela encomenda. Todo o processo foi executado através de emuladores e com rotas simuladas pelos mesmos. O vídeo está disponível em: https://youtu.be/IAQjlO0-B70

5 CONCLUSÃO

Analisando os resultados obtidos até aqui, o desenvolvimento completo de uma aplicação, com todas as funcionalidades e testes em produção ainda não foram realizados, mas já é o suficiente para saber que é possível desenvolver e utilizar as tecnologias alavancadas para o projeto. O início da prototipagem do design, definindo o fluxo principal da aplicação e definindo alguns requisitos iniciais foram essenciais para o desenvolvimento do aplicativo.

Também foram realizadas pesquisas quanto ao Flutter para entender as suas capacidades quanto a utilização do GPS para a localização em tempo real, também foi preciso analisar a viabilidade do uso da API de Geolocalização, pois essa versão inicial não terá verbas financeiras para o desenvolvimento.

A definição da Framework Flutter para o desenvolvimento da aplicação multiplataforma e a arquitetura, foram de extrema importância, pois permite a escalabilidade da aplicação caso ela venha a ser disponibilizada para mais usuários, visando a evolução do projeto para outros municípios e regiões.

A escolha para armazenamento de dados foi o Firebase, pois sua velocidade é muito importante na aplicação e pôde ser utilizado para servir tanto como um sistema de *login* quanto para o armazenamento das requisições dos processos de entrega.

O projeto também visou atingir o mercado local das cidades onde forem escolhidas para funcionarem, pois, necessita de uma pesquisa para conhecer o mercado local, saber se eles utilizam o transporte para as cidades vizinhas com uma certa frequência, e também por isso foi estimado uma distância máxima de 80 (oitenta) quilômetros de raio para realizar a entrega.

Por fim, conclui-se que o projeto pode ser totalmente desenvolvido em Flutter e não existem limitações atualmente para que uma aplicação não possa ser desenvolvida para produção com essas ferramentas escolhidas.

REFERÊNCIAS

ABCOMM. **Commerce Brasileiro: acompanhe as tendências e aproveite as ações para o seu segmento**. 2021. Disponível em: https://abcomm.org/noticias/commerce-brasileiro-acompanhe-as-tendencias-e-aproveite-as-acoes-para-o-seu-segmento/. Acesso em: 3 de março de 2021.

ARAUJO, R. de. **Clean Architecture: the solution to have a reusable, flexible and testable code**. 2019. Disponível em: https://rayanedearaujo.medium.com/clean-architecture-the-solution-to-have-a-reusable-flexible-and-testable-code-d8cdba74ab9f. Acesso em: 27 de abril de 2021.

BRACHA, G. **The Dart programming language**. [*S.l.*]: Addison-Wesley Professional, 2015.

ENTREGO, E. **Eu Entrego**. 2021. Disponível em: https://play.google.com/store/apps/details?id=com.euentrego.entregador. Acesso em: 01 de maio de 2021.

FLUTTER. **Improving rendering performance**. 2019. Disponível em: https://flutter.dev/docs/perf/rendering. Acesso em: 03 de maio de 2021.

FLUTTER. **Multi-Platform**. 2021. Disponível em: https://flutter.dev/multi-platform. Acesso em: 27 de junho de 2022.

GALANTE, V. R. **Melhores Frameworks para o desenvolvimento de aplicativos**. 2019. Disponível em: https://usemobile.com.br/framework-desenvolvimento-aplicativos-2019/. Acesso em: 01 de maio de 2021.

GOOGLE. **Seamless multi-platform app development with Flutter**. 2021. Disponível em: https://medium.com/googleplaydev/seamless-multi-platform-app-development-with-flutter-ea0e8003b0f9. Acesso em: 01 de maio de 2021.

KHALAF, S.; KESIRAJU, L. **US Mobile Daily Time Spent**. 2017. Disponível em: https://www.flurry.com/blog/us-consumers-time-spent-on-mobile-crosses-5/. Acesso em: 10 de abril de 2021.

LOGGI. **Loggi**. 2021. Disponível em: https://play.google.com/store/apps/details?id=com.loggi.client. Acesso em: 01 de maio de 2021.

MALINOSQUI, G. **Aplicativo híbrido: O que é e porque você deveria conhecer**. 2019. Disponível em: https://ezdevs.com.br/aplicativo-hibrido-porque-voce-deveria-conhecer/. Acesso em: 02 de maio de 2021.

MARKETEAM. **Aplicativo nativo, web App ou aplicativo híbrido?** 2020. Disponível em: https://usemobile.com.br/aplicativo-nativo-web-hibrido/. Acesso em: 02 de maio de 2021.

MARTIN, R. C. **The Clean Architecture**. 2012. Disponível em: http://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. Acesso em: 27 de abril de 2021.

REšETáR, M. **Flutter TDD Clean Architecture Course [1] – Explanation Project Structure**. 2019. Disponível em: https://resocoder.com/2019/08/27/flutter-tdd-clean-architecture-course-1-explanation-project-structure/. Acesso em: 27 de abril de 2021.

SINGH, R. **Mobile Development Approaches and Flutter Architecture: FLUTTER PART-I**. 2018. Disponível em: https://medium.com/gradeup/mobile-development-approaches-and-flutter-architecture-flutter-part-i-a7e08838c97a. Acesso em: 18 de abril de 2021.

STATCOUNTER. **Stat Counter**. 2021. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide. Acesso em: 04 de abril de 2021.