UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
BACHARELADO EM ENGENHARIA ELÉTRICA ELETRÔNICA

Proposta de um aplicativo de localização e mapeamento da UFAM

Julis Figueira de Araújo

Manaus – AM
Outubro -2023

Julis Figueira de Araújo

Proposta de um aplicativo de localização e mapeamento da UFAM

Monografia de Graduação apresentada ao Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de bacharel em Engenharia Elétrica Eletrônica.

Orientador

Prof. Dr Francisco de Assis Pereira Januário

Universidade Federal do Amazonas – UFAM

Faculdade de Tecnologia - FT

Manaus-AM

Outubro - 2023

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Araújo, Julis Figueira
A663p Proposta de um aplicativo de localização e mapeamento da UFAM / Julis Figueira Araújo . 2023
89 f.: il. color; 31 cm.

Orientador: Francisco de Assis Pereira Januário
TCC de Graduação (Engenharia Elétrica - Eletrônica) - Universidade Federal do Amazonas.

1. Comunidade acadêmica. 2. Android. 3. Flutter. 4. Firebase. 5. Ufam. I. Januário, Francisco de Assis Pereira. II. Universidade Federal do Amazonas III. Título

Monografia de Graduação sob o título Proposta de um aplicativo de localização e mapeamento da UFAM apresentada por Julis Figueira de Araújo e aceita pelo Instituto de Computação da Universidade Federal do Amazonas, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Prof. Dr. Francisco de Assis Pereira Januário
Departamento de Eletrônica e Computação (DTEC)
UFAM

Prof. Dr. Thiago Brito Bezerra
Departamento de Eletrônica e Computação (DTEC)
UFAM

Profa. Dra. Fabíola Guerra Nakamura
Instituto de Computação (ICOMP)
UFAM

Manaus-AM, 20 de outubro de 2023

Agradecimentos

Agradeço aos meus pais, que de forma incessante sempre me apoiaram durante toda a minha jornada de estudos. Minhas maiores lembranças, de quando pequeno até hoje, sempre serão do incentivo para que me tornasse alguém que pudesse desenvolver todo o potencial que viam em mim.

A minha irmã e melhor amiga, Clara, agradeço por sempre rir das minhas frustações durante esse caminho. Os momentos de descontração que tivemos e temos sempre fizeram com que essa jornada fosse mais relaxante. Estarei sempre aqui para estimular todas as coisas que sonhar para sua vida.

Agradeço aos meus amigos Elbesson, Inara e Patrick por serem os melhores amigos com quem pude compartilhar grande parte dessa trilha acadêmica. Sei que no momento mais complicado perante a universidade fomos alicerces uns dos outros. Uma de minhas grandes felicidades é saber que hoje conquistamos e desfrutamos juntos de tudo aquilo que sonhávamos há três anos, quando compartilhávamos dos mesmos medos e angústias.

Aos meus amigos Fábio e Gabriel, serei sempre grato por termos dividido o final dessa jornada. Grande parte disso não haveria sido construído caso não houvesse vocês. Sei que seremos grandes profissionais e que nossa amizade se estenderá mesmo com o fim desse ciclo. A minhas amigas Ariel, Carita, Natália e Thayla, agradeço por também termos compartilhado o fim dessa jornada juntos, vocês sempre fizeram com que momentos tristes se tornassem mais relaxados e reflexivos, e momentos de alegria fossem ainda mais comemorados e agradecidos.

Ao casal Jéssica e Phelippe, meus amigos desde a infância/adolescência, obrigado por sempre se manterem presentes até mesmo quando eu estive ausente. Além de compartilharmos grandes histórias de vida, vocês também estão nesse importante capitulo.

Ao meu amigo Maurício, apesar de nos conhecermos a tanto tempo, sou grato pela reaproximação de nossa amizade nos últimos anos. Estávamos juntos quando “ser” engenheiro era apenas uma utopia e estamos juntos novamente quando isso se tornou uma realidade.

Agradeço ao meu orientador Francisco Januário por ter acreditado neste projeto desde o começo. Sei que esse é um importante passo para a modernização da universidade e seu apoio é fundamental para que tudo isso aconteça.

Por fim, agradeço aos meus cachorros, Bolt e Ana Catarina, que nos momentos em que me dedicava aos estudos além do que o necessário, sempre me pediam a atenção, mesmo que de forma involuntária, para que pudéssemos passear e assim nos animarmos em todo fim de tarde.

Proposta de um aplicativo de localização e mapeamento da UFAM

Autor: Julis Figueira de Araújo
Orientador: Prof. Dr. Francisco de Assis Pereira Januário

RESUMO

Este trabalho visa abordar os desafios enfrentados pela comunidade acadêmica da UFAM (Universidade Federal do Amazonas), que incluem dificuldades na navegação no campus, acesso a eventos e busca por informações sobre ensalamento e linhas de ônibus localizadas na estação da universidade. Diante do cenário tecnológico em constante evolução e da necessidade de facilitar a vida dos estudantes, docentes e colaboradores, surge a motivação para desenvolver um aplicativo Android abrangente. Este aplicativo, criado com o uso do framework Flutter, linguagem Dart, Firebase e a API do Google Maps, oferece uma solução inovadora que agiliza o acesso a informações essenciais, promovendo eficiência e engajamento na comunidade acadêmica da UFAM.

Palavras-chave: Comunidade acadêmica, Android, Flutter, Firebase, UFAM, Eficiência.

Proposal for a UFAM location and mapping application

Author: Julis Figueira de Araújo
Advisor: Prof. Dr. Francisco de Assis Pereira Januário

ABSTRACT

This work aims to address the challenges faced by the academic community at UFAM (Federal University of Amazonas), which include difficulties in navigating the campus, accessing events and searching for information about the distribution of classrooms and bus lines located at the university station. Given the constantly evolving technological scenario and the need to make the lives of students, teachers and employees easier, the motivation to develop a comprehensive Android application arises. This application, created using the Flutter framework, Dart language, Firebase and the Google Maps API, offers an innovative solution that speeds up access to essential information, promoting efficiency and engagement in the UFAM academic community.

Keywords: Academic community, Android, Flutter, Firebase, UFAM, Efficiency.

Lista de figuras

Lista de tabelas

Sumário

1 Introdução

Nas universidades, é comum que se tenham murais espalhados pelos corredores com os principais avisos, divulgações de eventos, ensalamentos e outras informações sobre o ambiente. Entretanto, essa forma de comunicação tem se tornado cada vez menos prática devido a vivermos em um cenário onde a tecnologia nos proporciona uma troca de informações cada vez mais rápida e dinâmica, principalmente com o uso de celulares.

A cada início de período, por exemplo, a distribuição do ensalamento se torna um desafio para os alunos, pois estes precisam constantemente consultar o local onde cada disciplina será ministrada antes de cada aula. Esse cenário gera uma preocupação adicional para os estudantes, uma vez que a busca por informações sobre as salas de aula pode ser uma tarefa demorada e complicada, comprometendo a eficiência e a organização de suas atividades acadêmicas.

Outra situação que abrange instituições de ensino superior de grande porte é a falta de sinalização e mapeamento adequado de seu *campus*. Muitos estudantes, especialmente os calouros, enfrentam problemas para encontrar suas salas em meio a um labirinto de corredores e prédios desconhecidos. Essa situação pode gerar atrasos nas aulas, estresse e um sentimento de desorientação.

Além disso, outro problema que abrange as universidades é a falta de divulgação de seus eventos e a necessidade de uma divulgação mais objetiva. A baixa participação e interesse dos estudantes, docentes e colaboradores, devido a uma divulgação pouco abrangente acaba resultando em um desperdício de recursos e esforços por parte dos organizadores. Muitas vezes, eventos valiosos e enriquecedores acabam passando despercebidos, pois as informações não chegam ao público-alvo de maneira efetiva. Em ambientes com uma ampla diversidade de atividades acadêmicas, culturais e sociais acontecendo constantemente, é crucial garantir que todos os membros da comunidade universitária estejam bem informados sobre as oportunidades disponíveis.

Dessa forma, este trabalho tem como finalidade o desenvolvimento de uma aplicação ampla e intuitiva para atender às necessidades específicas dos estudantes, docentes e colaboradores da Faculdade de Tecnologia da UFAM. O aplicativo proporciona uma plataforma integrada que abrange o mapeamento,

ensalamento e divulgação de eventos da instituição. Além disso, disponibiliza informações atualizadas sobre os horários dos ônibus que partem da estação localizada na universidade.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolver um aplicativo mobile que permita o acesso ao mapeamento, ensalamento e divulgação de eventos da Faculdade de Tecnologia e contenha as informações dos ônibus da estação localizada na UFAM, com o propósito de proporcionar uma solução integrada e conveniente para atender às necessidades da comunidade acadêmica.

1.1.2 Objetivos Específicos

- Incorporar recursos de mapeamento para ajudar os usuários a localizarem facilmente salas, laboratórios, secretarias e outras áreas importantes da Faculdade de Tecnologia.
- Permitir que os usuários acessem informações atualizadas do aplicativo através do uso de ferramentas que mantenha a base de dados sincronizada.
- Implementar recursos de busca e filtro permitindo que os usuários pesquisem eventos ou lugares específicos.
- Desenvolver uma interface atraente e fácil de usar, garantindo que os usuários possam navegar pelo mapa do *campus* da faculdade de forma rápida e intuitiva.

1.2 Organização do trabalho

No capítulo 2, são abordados os fundamentos do projeto, explorando conceitos essenciais, tecnologias e ferramentas que sustentam a pesquisa. Tópicos como mapeamento, rotas, Flutter, Firebase e API são discutidos em detalhes, fornecendo uma base sólida para a compreensão do contexto do projeto. O capítulo 3 detalha o processo de desenvolvimento do projeto. Começando pela fase de

modelagem, são apresentados os requisitos funcionais e não funcionais, juntamente com diagramas de casos de uso e de classe. Em seguida, o capítulo aborda a configuração do Firebase, o desenvolvimento do backend e a criação de interfaces de usuário. Cada etapa do desenvolvimento é discutida de maneira aprofundada. No capítulo 4, o foco se volta para os testes realizados no projeto. Diferentes tipos de testes são abrangidos, desde testes de interface do usuário e funcionalidade até testes de desempenho, segurança e conexão. A avaliação da robustez e eficácia do sistema é um aspecto crítico deste capítulo. E finalmente o capítulo 5 oferece uma análise final do projeto. Aqui, são apresentadas as conclusões obtidas ao longo da pesquisa e desenvolvimento. Além disso, são sugeridas possíveis direções para trabalhos futuros, destacando oportunidades de expansão e aprimoramento do projeto.

2 Fundamentos

2.1 Mapeamento e rotas

2.1.1 Coordenadas geográficas

O sistema de coordenadas geográficas é utilizado para definir a localização de pontos na Terra por meio do uso de linhas imaginárias traçadas sobre a superfície do globo terrestre. Esse sistema é composto por uma unidade angular de medida e um meridiano principal. Um ponto tem a referência por seus valores de latitude e longitude medidos do centro da Terra até um ponto na superfície (Maling, 2006).

A latitude é o ângulo medido em relação ao equador em direção aos pólos. É zero no equador, e aumenta em direção aos dois pólos. Possui valor máximo de 90º no Polo Norte e valor mínimo de 90º no Polo Sul. Linhas circulares de mesma latitude são chamadas de paralelo, como mostrado no globo esquerdo da Figura 1 (Maling, 2006).

A longitude é o ângulo medido em relação ao meridiano principal, que pode possuir valor máximo de 180º no sentido leste 180º no sentido oeste. Linhas em formato de elipse com a mesma longitude são chamadas são chamadas de meridianos, como apresentado no globo da direita da Figura 1(Maling, 2006).

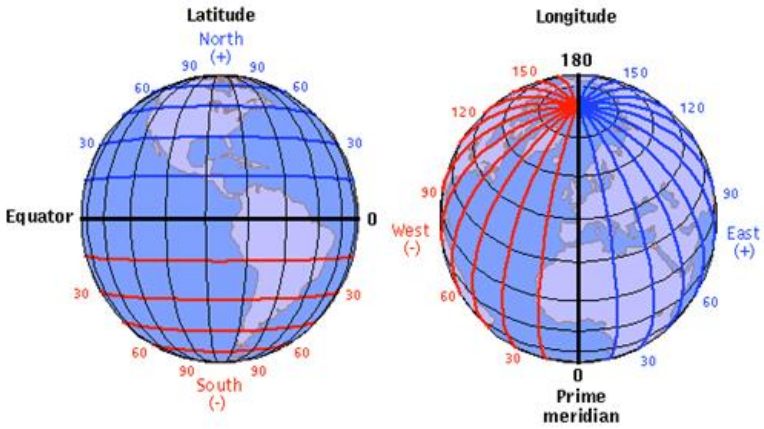


Figura 1. Linhas imaginárias de latitude e longitude. Fonte: JOURNEY NORTH, 2023.

2.1.2 GPS

O Sistema de Posicionamento Global (GPS) é uma tecnologia que se vale de satélites para realizar o rastreamento e a localização precisa. Sua metodologia fundamental envolve a mensuração das distâncias entre o receptor e múltiplos satélites que são observados simultaneamente. As posições dos satélites são previamente determinadas e transmitidas ao usuário juntamente com os sinais do GPS. Com base nessas posições conhecidas e nas distâncias medidas entre o receptor e os satélites, é possível calcular a posição do receptor. Além disso, ao analisar as mudanças na posição ao longo do tempo, é possível determinar a velocidade do receptor. O GPS desempenha um papel crucial em diversas aplicações, com destaque para o posicionamento de alta precisão e a navegação (Xu, 2007).

O Sistema de Posicionamento Global (GPS) foi desenvolvido, construído, operado e mantido pelo Departamento de Defesa dos EUA. O primeiro satélite GPS foi lançado em 1978, e o sistema alcançou sua plena operacionalidade em meados da década de 1990 (Parkinson & Spilker, 1996).

Neste trabalho, o GPS será utilizado para obter a localização exata do usuário e calcular o caminho até o destino desejado. A Figura 2 mostra o processo de como a localização exata do usuário é obtida através dos satélites.

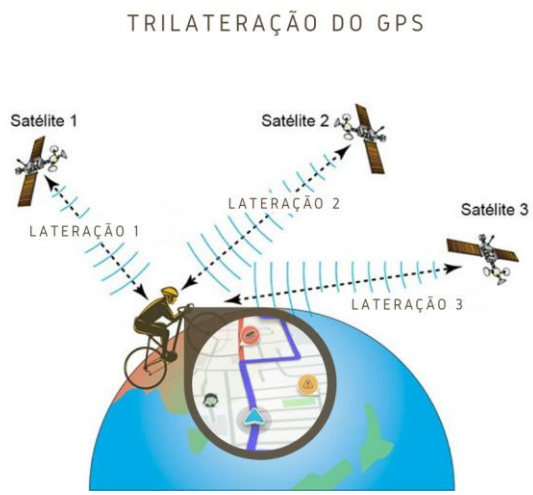


Figura 2. Rede de três satélites orbitando a terra e um receptor. Fonte: BURBULHAN, 2020.

2.1.3 Caminho mínimo

Dentro de um sistema de navegação como o Google Maps, por exemplo, onde queremos criar a menor e mais otimizada rota possível de um ponto a outro, são levados em conta vários fatores como distância, trânsito e tempo de deslocamento, por exemplo. Geralmente para encontrar essa rota dispõe-se de uma técnica utilizada em algoritmos para encontrar o caminho mais curto entre dois pontos em um grafo ou rede.

A questão do caminho mínimo, frequentemente chamada de caminho mais curto, refere-se à busca da rota ideal entre dois nós em uma rede. Em uma rede, que pode apresentar diferentes configurações, podem existir diversas rotas possíveis entre um par de nós, identificados como ponto de partida e ponto de chegada. Dentro dessas alternativas, o caminho mínimo é aquele caracterizado pelo menor "custo", o qual é determinado pela soma dos valores dos segmentos que constituem o percurso (Wilhelm, 2023).

Na Figura 3 é possível ver um exemplo de uma rede com cinco nós, A, B, C, D e E. As setas indicam o caminho feito entre cada um desses nós, junto de seus pesos ao lado.

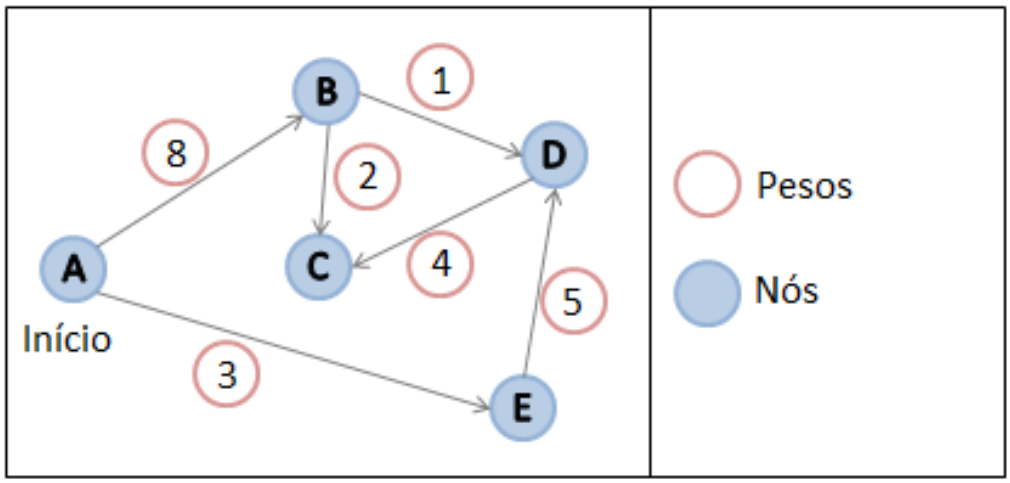


Figura 3. Rede de nós. Fonte: Autoria própria.

Na tabela 1 é possível analisar o caminho mínimo para cada um dos nós partindo do nó A. Essa tabela indica os pontos para o caminho mais curto e a soma de todos os pesos desse trajeto.

Tabela 1. Caminho mínimo para todos os nós partindo do nó A da Figura 3.

Destino	Menor Caminho	Soma dos pesos
B	(A, B)	8
C	(A, B, C)	8 + 2 = 10
D	(A, E, D)	3 + 5 = 8
E	(A, E)	3

Fonte: Autoria própria.

2.1.4 Algoritmo de *Dijkstra*

O algoritmo de Dijkstra proporciona uma resolução para o problema do caminho mais curto partindo de um único ponto de origem. É eficaz tanto em grafos direcionados quanto não direcionados, desde que todas as arestas tenham custos não negativos. Nesse sistema, o nó de partida é o ponto de entrada, e o objetivo é encontrar a menor distância desse ponto de origem para cada um dos pontos no sistema (Wilhelm, 2023).

O algoritmo tem a finalidade de calcular o custo mínimo entre o nó de origem O e todos os outros nós no grafo. Inicialmente, o conjunto S contém apenas o nó O. A cada etapa, selecionamos o nó mais próximo da origem dentre aqueles que ainda não foram processados. Posteriormente, atualizamos a distância de cada nó restante em relação à origem. Se, ao passar pelo novo nó adicionado, a distância se tornar menor em comparação à distância anterior, essa nova distância é armazenada. O algoritmo, ao escolher um nó como origem, prossegue iterativamente para calcular o custo mínimo desse nó, de modo que, em alguma iteração, todos os nós sejam alcançados a partir de todos os outros nós do grafo. Na primeira iteração, determina o nó mais próximo de O; na segunda iteração, o segundo nó mais próximo; e assim sucessivamente, até que, em algum momento, todos os nós sejam acessados (Wilhelm, 2023).

Neste aplicativo o algoritmo de Dijkstra será utilizado para obter o menor caminho entre o local atual do usuário e o destino escolhido. Para criar uma rede de nós e malhas serão utilizados os corredores da universidade previamente mapeados com base em sua latitude e longitude.

2.2 Framework e ambiente de desenvolvimento

2.2.1 *Flutter*

Desenvolvido em 2015 pela Google, o *Flutter* é um framework de desenvolvimento híbrido com compilação para mobile (*Android* e *iOS*), web, desktop (*Windows, macOS* e *Linux)* e dispositivos embarcados que tem o Dart como linguagem de programação. Apesar de ser um framework relativamente recente, o *Flutter* já vem sendo utilizado em grande escala por grandes empresas como: *Nubank*, *iFood*, *Alibaba* e *Ebay*, por exemplo.

A seguir são abordados alguns conceitos bem importantes no desenvolvimento *Flutter*.

2.2.1.1 Widgets

Para a criação de telas, o *Flutter* utiliza o conceito de *widget* que nada mais é que um componente visual para definir a interface de um aplicativo (PINHEIRO, 2020).

Dessa maneira todos os componentes de uma tela como botões, textos, imagens e campos de texto, por exemplo, são *widgets*.

Existem dois tipos de *widgets* em *Flutter*:

- *Widgets Stateless:* São *widgets que não podem mudar seu estado interno, ou seja, sem estado. Eles são construídos com base na sua herança inicial permanecendo constantes enquanto são mostrados na tela. Um exemplo desse widget é o Text onde um texto com o nome menu é exibido de forma permanente.*
- *Widgets Stateful*: Este tipo de *widget* possui estado interno que pode ser alterado enquanto o *widget* é mostrado. Quando o seu estado muda, ele é refeito e reflete as mudanças na tela. Um exemplo de *widget* com estado é o *TextField*, onde o texto é atualizado na tela a cada vez que o usuário digita.

Segundo Marcoratti (2019), o leque de uso de *widgets* no *Flutter* reside não apenas na capacidade de criar *widgets* personalizados para atender às necessidades particulares, mas também na habilidade de combiná-los a fim de obter

interfaces mais complexas. Isso culmina na criação da árvore de *widgets*. Esta árvore representa visualmente a organização hierárquica dos *widgets*, na qual *widgets*-pai englobam *widgets*-filhos, construindo uma estrutura que espelha a composição da interface do usuário, como é mostrado na Figura 4.

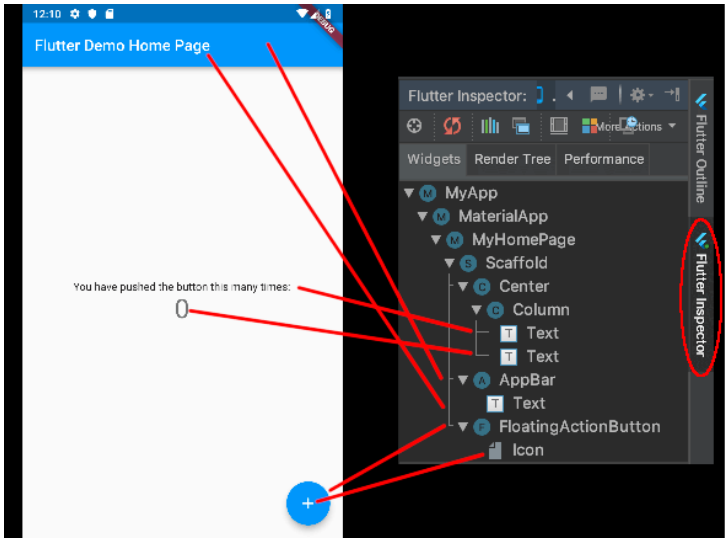


Figura 4. Arvore hierárquica de *widgets*. Fonte: MACORATTI, 2019.

2.2.1.2 *Dart*

Criada em outubro de 2011 pela *Google*, *Dart* é uma linguagem de programação fortemente tipada e multi-paradigma. Com o estilo de sintaxe fundamentado em C e com compilações de código baseada em *JavaScript*, sua missão inicial era de substitui-lo a fim de unir *frontend*, aplicações móveis e o lado do servidor, logo tentando mudar a linguagem fundamental dos navegadores (BARRO, 2023). Devido ao *Java Script* já estar consolidado, a tentativa não foi bem sucedida. Entretanto o Dart acabou se tornando bastante popular com o surgimento do *Flutter*.

De maneira geral o Dart é uma linguagem orientada a objetos que suporta conceitos como classes, herança, interfaces e encapsulamento. Além disso, oferece suporte à programação assíncrona, permitindo comunicação eficiente com servidores usando as palavras-chave "async" e "await". Essa linguagem oferece várias opções de ambiente de execução, incluindo a máquina virtual DartVM, que possui os modos JIT (Just-in-Time Compiler) e AOT (Ahead-of-Time Compiler). No modo JIT, a compilação ocorre durante a execução, permitindo desenvolvimento ágil

e testes rápidos. Por outro lado, no modo AOT, a compilação antecipada gera código nativo, resultando em desempenho mais rápido, pois não há necessidade de compilar durante a execução.

2.2.1.3 Gerenciamento de Estado

O gerenciamento de estado no *Flutter* refere-se à maneira como se controla, armazena e atualiza os dados que afetam a interface do usuário da aplicação. Em qualquer aplicativo interativo, os dados estão em constante mudança devido à interação do usuário e eventos externos, como *APIs* por exemplo. O *Flutter* oferece várias bibliotecas para gerenciar o estado, como *BLoC*, *MobX*, *GietX* e *Provider*, por exemplo. Nesta aplicação será utilizado o *GetX* devido a sua facilidade de uso e interação.

O *GetX* é uma biblioteca de gerenciamento de estados em *Flutter* reconhecida pelo seu bom desempenho, produtividade, organização e, além disso, por precisar de menos código e evitar reconstruções desnecessárias (ALMEIDA, 2023).

GetX tem 3 princípios básicos. Isso significa que são prioritários para todos os recursos da biblioteca: produtividade, desempenho e organização (Pub.Dev, 2023d).

- Produtividade: usa uma sintaxe fácil e agradável. Isso economizará horas de desenvolvimento e fornecerá o desempenho máximo para a aplicação;
- Desempenho: é focado em desempenho e consumo mínimo de recursos;
- Organização: permite o desacoplamento total da *View*, lógica de apresentação, lógica de negócio, injeção de dependência e navegação. Não é necessário um contexto para navegar entre as rotas, portanto, não depende da árvore de *widgets* (visualização) para isso.

2.2.1.4 Desenvolvimento híbrido e arquitetura

Como já citado, o *Flutter* permite criar aplicativos nativos para várias plataformas a partir de um único código base. A compilação *multiplataforma* é um processo pelo qual o código-fonte desenvolvido em *Flutter* é transformado em código nativo para a plataforma ao qual se destina, como *Android*, *iOS*, web e/ou

desktop. De maneira genérica esse processo de arquitetura segue os seguintes estágios mostrados na Figura 5:

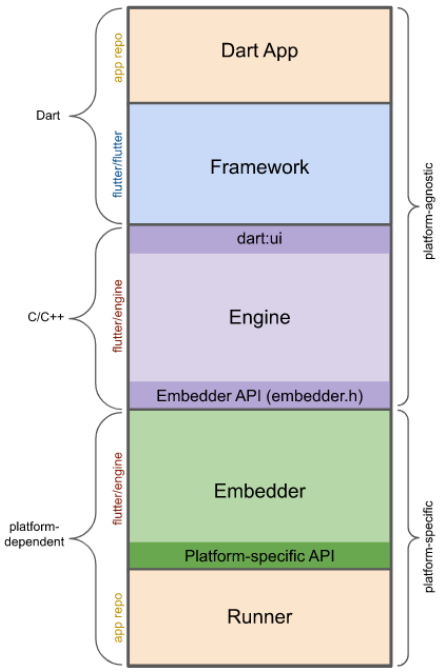


Figura 5. Estágios da arquitetura *Flutter*. Fonte: FLUTTER, 2023.

O desenvolvimento de aplicativos Dart envolve a composição de widgets na interface do usuário, a implementação da lógica de negócios e a competência do desenvolvedor. O framework Flutter oferece uma API de alto nível para criar aplicativos de alta qualidade, incluindo recursos como widgets, teste de cliques, detecção de gestos e acessibilidade. A engine é responsável por rasterizar as cenas e fornece implementações de baixo nível das principais APIs do Flutter, como gráficos e layout de texto. Ela também se integra à plataforma específica usando a API Embedder. O Embedder coordena com o sistema operacional para acessar serviços como renderização, acessibilidade e entrada, além de gerenciar o loop de eventos, expondo a API da plataforma para integração. O Runner compõe as partes expostas pela plataforma específica do Embedder em um pacote de aplicativo executável na plataforma de destino (Flutter, 2023).

Apesar da possibilidade de uso do *Flutter* em diversas plataformas, o presente trabalho visa o desenvolvimento apenas para dispositivos *Android*, visto a facilidade de desenvolvimento e publicação em sua loja de aplicativos. Essa exclusividade permite oferecer uma abordagem eficiente e econômica para atingir um amplo público de usuários. Além disso, oferece a oportunidade de agilizar o

processo de criação, reduzir a complexidade de manutenção e entrega de atualizações pós-desenvolvimento.

2.2.1.5 *Packages*

Package é um conjunto de código reutilizável com direcionamento específico, geralmente encapsulado em uma biblioteca, que é incorporado a projetos de modo especificado. Os *packages* são uma parte fundamental do ecossistema *Flutter*, pois permitem o compartilhamento de código de maneira eficiente. Podem conter *widgets*, utilitários, serviços, recursos visuais e lógica de negócios, por exemplo. Os pacotes permitem a criação de código modular que pode ser compartilhado facilmente.

Um *package* contém uma *API* escrita em código *Dart* combinada com uma ou mais implementações específicas da plataforma. Os *packages* de *plug*-in podem ser escritos para *Android* (usando *Kotlin* ou *Java*), iOS (usando *Swift* ou *Objective*-C), web, *macOS*, *Windows* ou *Linux* ou qualquer combinação deles (Flutter, 2023).

2.2.2 *Android Studio*

Quando se deseja desenvolver um software é sempre importante que se escolha uma IDE que forneça todas as ferramentas necessárias para o projeto. No desenvolvimento deste aplicativo, o ambiente escolhido foi o *Android Studio* não só pelo seu sistema de edição e compilação de código, mas também por oferecer um emulador integrado que permite simular a aplicação em diferentes dispositivos e versões, trazendo maior segurança quanto à responsividade da UI.

O *Android* Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de apps Android. De acordo com IntelliJ IDEA (2023) , este ambiente pode oferecer um sistema de build flexível baseado em Gradle e um emulador rápido com inúmeros recursos, proporcionando um ambiente unificado para o desenvolvimento em dispositivos Android. Além disso, permite a edição em tempo real para atualizar elementos combináveis em emuladores e dispositivos físicos, oferece frameworks e ferramentas de testes versáteis e inclui ferramentas de lint para detectar problemas de desempenho, usabilidade e compatibilidade com versões.

2.3 Backend e banco de dados

2.3.1 Banco de dados *NoSQL*

O banco de dados *NoSQL* é um banco cujo os dados não são armazenados no formato de tabela, ou seja, não possui esquema. Também chamado de banco de dados não relacional, este tipo de armazenamento permite o tratamento de uma grande quantidade de dados que podem ou não estar em constante mudança. Além disso, possui a grande vantagem de processamento de grandes dados que não relacionados entre si permitindo sua consulta de diversas formas. Os tipos mais comuns desse modelo de dados são (Azure, 2023):

- **Chave-valor**: Os armazenamentos de chave-valor fazem o pareamento de chaves e valores usando uma tabela de *hash*. Os tipos de chave-valor são melhores quando uma chave é conhecida e o valor associado dela é desconhecido.
- **Documento**: Os bancos de dados de documentos ampliam o conceito do banco de dados chave-valor organizando documentos inteiros em grupos chamados coleções. Eles são compatíveis com os pares chave-valor aninhados e permitem consultas em qualquer atributo em um documento.
- **Colunar**: Bancos de dados colunares, de coluna larga ou de famílias de colunas armazenam dados de modo eficiente, consultam linhas de dados esparsos e são vantajosos ao consultar em colunas específicas no banco de dados.
- **Grafo**: Os bancos de dados de grafo usam um modelo baseado em nós e bordas para representar dados interconectados, como relações entre pessoas em uma rede social, e oferecem armazenamento simplificado e navegação por meio de relações complexas.

2.3.*2 Backend as a Service – Baas*

Backend as a *Service*, ou apenas *BaaS*, caracteriza-se por um pacote de soluções hospedadas na nuvem, com um propósito final que dispõe *APIs* ou *SDKs* para estes serviços serem utilizados por outras aplicações 4 (Amorim, 2017).

Dentre suas principais ferramentas podemos destacar o gerenciamento de banco de dados, autenticação, hospedagem, notificações e armazenamento em nuvem. Este tipo de serviço de *software* vem se tornando bem popular entre os programadores visto que acaba tornando o desenvolvimento de aplicações bem mais simples permitindo que haja maior concentração no desenvolvimento front-end e maior segurança e comodidade no *backend*. Utilizando um *BaaS*, a ênfase é totalmente direcionada para o desenvolvimento *frontend* da aplicação, uma vez que muitos dos recursos são disponibilizados de forma automatizada, conforme ilustrado na Figura 6:

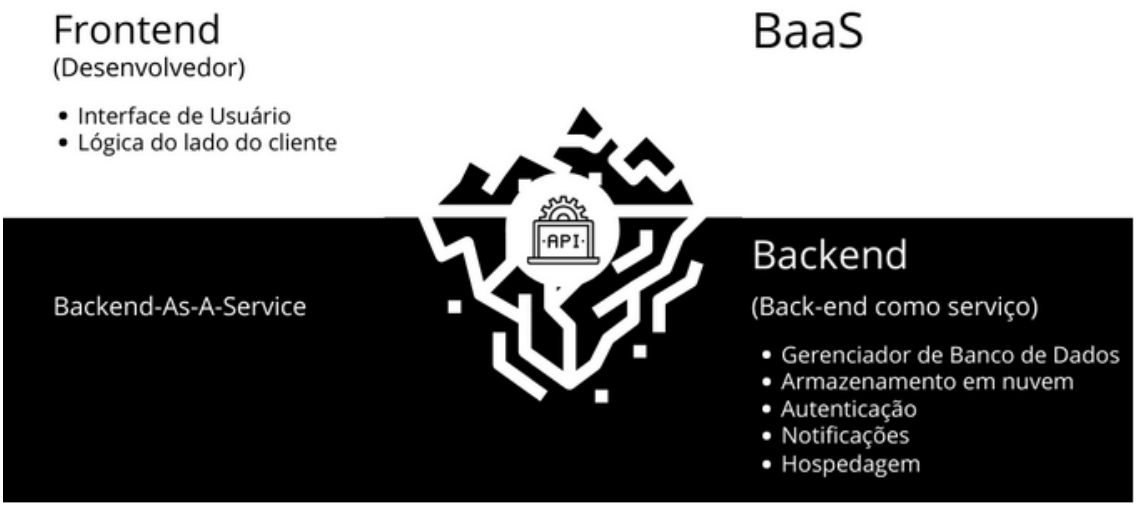


Figura 6. Integração backend e frontend utilizando um Baas. Fonte: Treina Web, 2020.

2.3.3 Firebase

Um dos serviços mais utilizados de *Baas* é o *Firebase*. Adquirido em 2014 pela *Google*, o *Firebase* fornece aos desenvolvedores uma variedade de ferramentas e serviços para ajudar no desenvolvimento de aplicativos de qualidade, aumentar sua base de usuários e ser mais lucrativo. Toda sua base é construída na infraestrutura do Google, sendo categorizado como um programa de banco de dados *NoSQL*, que armazena dados em documentos do tipo *JSON*. Além disso, suporta o desenvolvimento nas seguintes linguagens: Dart, C++, Java, *Javascript*, *Node*.js, *Objective*-C e *Swift*. (Geekhunter, 2023).

Dentro do leque de funcionalidades oferecidas pelo Firebase, este trabalho irá utilizar apenas duas: o *Firestore Database* para armazenar todo o banco de dados

das funcionalidades do aplicativo e o *Storage* para armazenar as mídias referentes à divulgação de eventos.

2.3.3.1 Firestore Database

Firestore Database é um banco de dados em nuvem *NoSQL* flexível e escalável (Firebase, 2023). Ele permite o armazenamento de dados do sistema com sincronização em tempo real com os usuários. Além disso, uma de suas vantagens é o funcionamento off-line devido ao armazenamento em cache de uma cópia dos dados da nuvem quando online.

O armazenamento dos dados é feito em documentos que contêm mapeamentos de campos para valores. Esses documentos são armazenados em coleções, que são contêineres de documentos que você pode usar para organizar dados e criar consultas. Os documentos suportam diversos tipos de dados, desde os mais simples, como *strings*, int e *doubles*, até os mais complexos como *maps* e *arrays*. Também há a possibilidade de criar subcoleções dentro dos documentos e criar estruturas de dados hierárquicas que podem ser escalonadas à medida que o banco de dados cresce e se torna mais complexo (Firebase, 2023).

Na Figura 7 é possível ver de maneira mais detalhada como funcionam as coleções no *Firebase*. No exemplo é criada uma coleção de animais junto de algumas de suas propriedades.

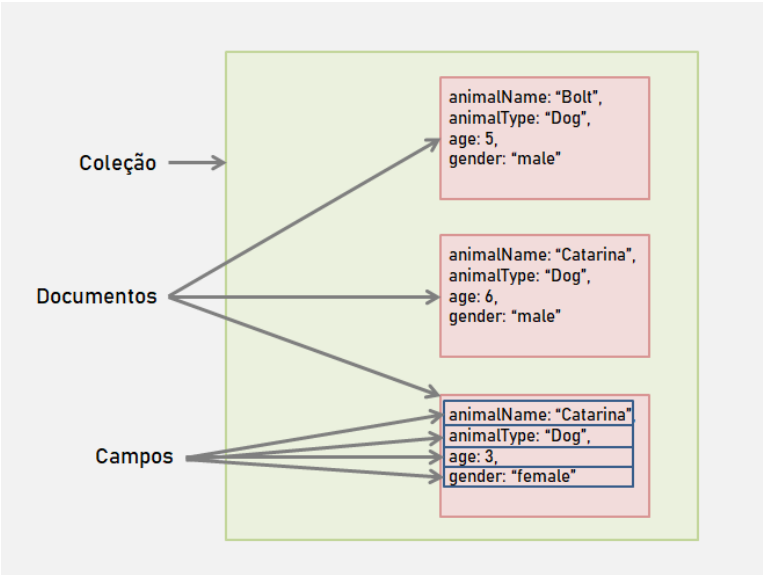


Figura 7. Modelo de uma coleção de documentos de animais. Fonte: Autoria própria.

- **Flexibilidade**: estruturas de dados hierárquicas flexíveis que armazena seus dados em documentos, organizados em coleções. Os documentos podem conter objetos aninhados complexos, além de subcoleções.
- **Consultas**: consultas para recuperar documentos individuais e específicos ou recuperar todos os documentos em uma coleção que corresponda aos parâmetros da consulta. As consultas podem incluir vários filtros em cadeia e combinar filtragem e classificação. Além disso podem ser indexadas por padrão, portanto, o desempenho da consulta é proporcional ao tamanho do conjunto de resultados, e não ao conjunto de dados.
- **Atualizações em tempo real**: sincronização de dados para atualizar dados em qualquer dispositivo conectado. No entanto, ele também há a possibilidade de fazer consultas de busca simples e únicas de maneira eficiente.

2.3.3.2 Storage

O *Storage* é um serviço de armazenamento de objetos avançado, como vídeos e fotos, por exemplo, simples e econômico criado para a escala do *Google*. Através dele é possível usar a segurança do *Google* para fazer *upload* e *download* de arquivos nos *apps* do *Firebase*, independentemente da qualidade da rede. Dentre seus principais recurso podemos citar (Firebase, 2023).

- **Operações confiáveis**: o *uploads* e *downloads* são feitos independentemente da qualidade da rede. Os uploads e downloads são mais confiáveis, o que significa que eles são retomados no ponto em que foram interrompidos, poupando tempo e largura de banda dos usuários.
- **Alta escalabilidade:** o *Storage* foi criado para escalonar em *exabytes* para quando necessário, com isso é possível evoluir facilmente do protótipo para a produção usando a mesma infraestrutura utilizada no *Spotify* e no *Google* Fotos.

2.4 API

Conforme Silva (2023), o termo API significa *Application Programming Interface* ou, em português, *Interface* de Programação de Aplicativos. Em suma, a definição de API está ligada a um conjunto de padrões de programação e rotinas que promovem a conexão de aplicativos e sistemas.

2.4.1 Google Maps API

A *API* do *Google Maps* é um recurso da *Google* que permite a utilização de serviços de mapeamento e geolocalização em plataformas digitais. A API funciona utilizando os dados contidos nos servidores da *Google*. Uma vez este serviço apto na aplicação é possível personalizar o mapa de acordo com a sua finalidade adicionando polígonos, marcadores, alterar o tipo de visualização e quantidade de detalhes quando renderizado, por exemplo.

3 Desenvolvimento

3.1 Modelagem

A modelagem de um sistema é um componente essencial do ciclo de desenvolvimento de software. Para este projeto adotou-se o padrão UML (Unified Modeling Language), uma linguagem destinada à modelagem de estruturas que compõem aplicações. Ela se baseia fortemente em conceitos de Orientação a Objetos e oferece uma variedade de notações para criar diagramas que representam vários aspectos de um software. Por meio dessa abordagem de modelagem, torna-se viável registrar de maneira clara e concisa as interações, funcionalidades e comportamentos do software. Diagramas de Casos de Uso serão utilizados para descrever as interações entre os usuários e o sistema, enquanto Diagramas de Classes definirão a estrutura das entidades envolvidas.

3.1.1 Requisitos

Requisitos funcionais e não funcionais são duas categorias distintas de requisitos que são usadas para especificar o comportamento e as características de um sistema de software.

3.1.1.1 Requisitos funcionais

Os requisitos funcionais definem as funções ou funcionalidades específicas que um sistema de software deve executar. Eles descrevem o que o sistema deve fazer em termos de ações, serviços ou operações. A seguir são apresentados os requisitos funcionais do sistema:

- O sistema deve permitir que os usuários acessem a tela de exploração a partir da tela inicial do aplicativo;
- Os usuários devem poder tocar em um local listado para visualizar informações completas, incluindo nome, descrição, localização e fotos;
- A partir da tela de detalhes do local ou evento, os usuários podem navegar para uma tela de mapa que exibe a localização exata no mapa;
- Os usuários podem criar rotas que os guiarão de um local para outro;

- O sistema deve fornecer instruções de navegação para seguir a rota;
- Os usuários devem poder navegar facilmente entre as diferentes telas do aplicativo, incluindo a tela inicial, a tela de eventos, a tela de departamentos, a tela de salas, a tela de configurações e outras telas relevantes;
- O aplicativo deve exibir uma lista de eventos, incluindo detalhes como título, data e hora, localização e descrição;
- Os usuários devem poder tocar em um evento para visualizar detalhes adicionais;
- O aplicativo deve exibir uma lista das unidades acadêmicas da UFAM contidas no banco de dados;
- O aplicativo deve permitir aos usuários visualizar uma lista de salas em uma unidade acadêmica específica;
- O aplicativo deve permitir que os usuários pesquisem eventos, unidades acadêmicas e salas;
- Os usuários devem poder acessar a tela de configurações para ajustar as preferências do aplicativo;
- O aplicativo deve oferecer suporte para os idiomas inglês e português para atender a uma base de usuários diversa.

3.1.1.2 Requisitos não funcionais

Os requisitos não funcionais especificam as características do sistema que não estão diretamente relacionadas às funcionalidades específicas, mas afetam a qualidade, desempenho, segurança e usabilidade do software. A seguir são apresentados os requisitos não funcionais do sistema:

- A versão mínima do aplicativo *Android* deve ser a 10.0;
- A versão mínima do *Flutter* para o desenvolvimento do aplicativo deve ser 3.7.12;
- A versão mínima do *Dart* para o desenvolvimento do aplicativo deve ser 2.19.6;
- A versão mínima do *Android* Studio para o desenvolvimento do aplicativo deve ser a 2022.1.1;

3.1.2 Diagrama de Casos de Uso

Um diagrama de casos de uso genérico é uma representação visual que identifica as interações entre atores e funcionalidades em um sistema. Usando elipses para representar atores e retângulos para os casos de uso, o diagrama ilustra como os usuários interagem com o sistema para alcançar objetivos.

A seguir, na Figura 8, é apresentado o diagrama de casos de uso do *software*. O sistema tem como funções fundamentais consultar ensalamento, buscar lugares, buscar eventos e consultar ônibus. A partir destes pode-se realizar diversas outras interações.

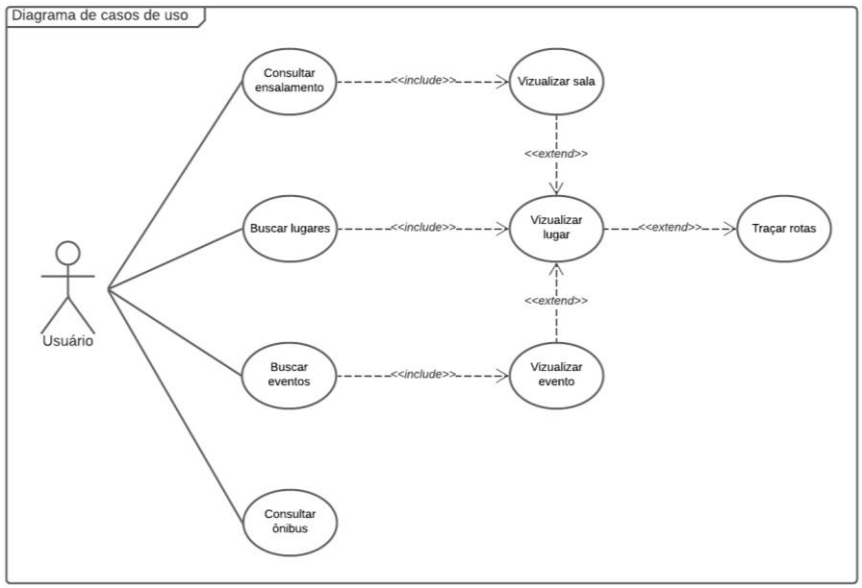


Figura 8. Diagrama de casos de uso. Fonte: Autoria própria.

Os casos de uso incluem a capacidade de consultar o ensalamento, buscar lugares, buscar eventos, consultar informações sobre um determinado ônibus, visualizar salas e traçar rotas até um ponto escolhido. Essas funcionalidades permitem que o usuário acesse informações específicas e navegue pelo sistema de forma eficiente. Além disso, os relacionamentos entre os casos de uso foram definidos, sendo *"Include"* para indicar que o caso de uso "Consultar ensalamento" inclui os casos de uso "Buscar lugares" e "Buscar eventos", garantindo que o usuário consulte informações detalhadas de um ensalamento. Enquanto isso, o relacionamento *"Extend"* estabelece que o caso de uso "Visualizar sala" estende o caso de uso "Consultar ensalamento", permitindo que os usuários obtenham informações sobre a sala, mesmo sem consultar informações sobre o ensalamento.

Essa abordagem de casos de uso e relacionamentos oferece uma estrutura organizada e flexível para a interação do usuário com o sistema.

3.1.3 Diagrama de Classes

Um diagrama de classes é uma representação visual que descreve a estrutura estática de um sistema por meio de classes, atributos e relacionamentos. Essencial para a modelagem orientada a objetos, o diagrama de classes ajuda a definir a arquitetura do software e o design, permitindo uma visão abrangente das entidades do sistema e suas interconexões.

A seguir, na Figura 9, é apresentado o diagrama de classes do software.

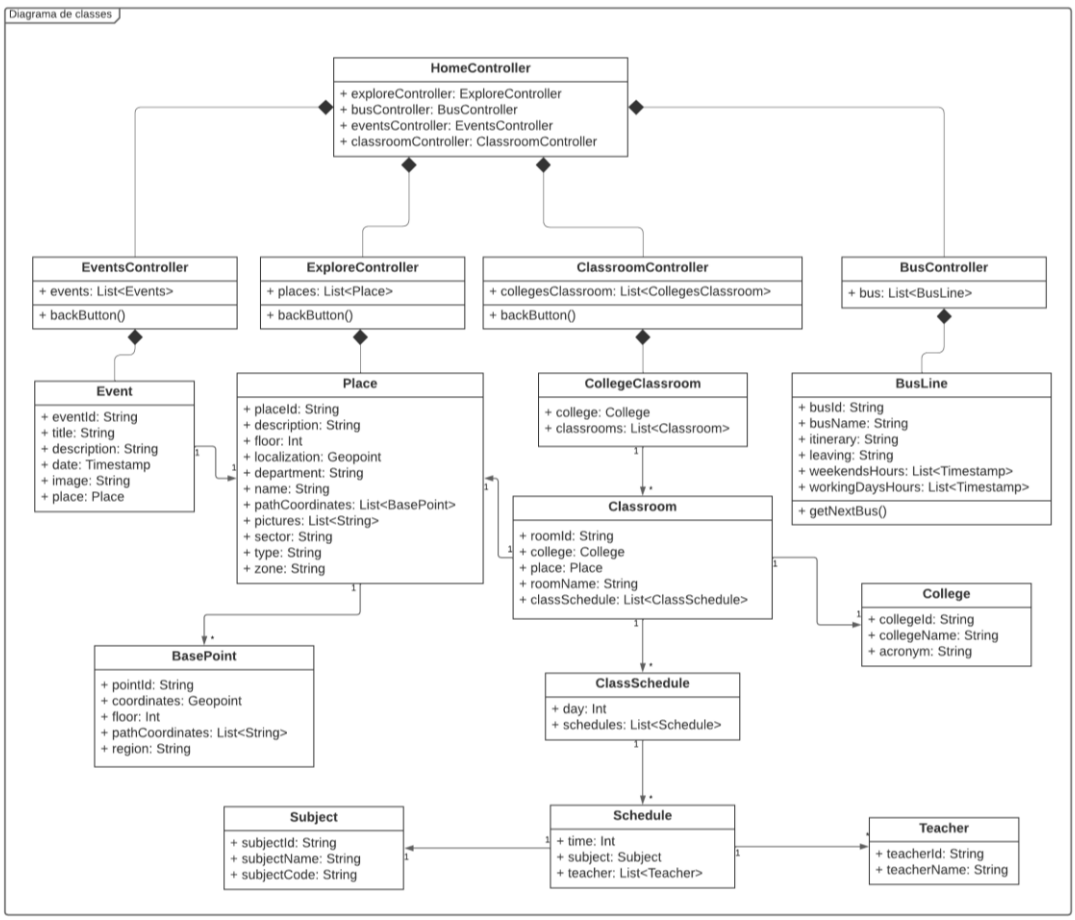


Figura 9. Diagrama de classes. Fonte: Autoria própria.

No diagrama de classes fornecido, várias classes desempenham funções específicas no sistema, incluindo a *HomeController*, responsável pelo controle da interface inicial do sistema; o *EventsController*, que gerencia a exibição e edição de

eventos; o *ExploreController*, encarregado da exibição de eventos e lugares; o *ClassroomController*, que controla a exibição e edição de salas de aula; o *BusController*, responsável pela exibição de informações sobre ônibus. Além disso, várias classes modelam entidades no sistema, como o Place, que representa locais, como ensalamentos ou pontos de ônibus; o *CollegeClassroom*, que descreve salas de aula em ensalamentos específicos; o *Event*, que representa eventos, como palestras e shows; o *BusLine*, modelando linhas de ônibus; *Subject*, que representa disciplinas acadêmicas; Schedule, que modela horários de aula; e *Teacher*, que descreve informações sobre professores. Cada uma dessas classes desempenha um papel vital no sistema, contribuindo para a funcionalidade geral e a representação dos objetos do mundo real. Na seção 3.3, cada uma dessas classes é abordada de forma individual detalhando cada um de seus objetos e funcionalidades.

3.3 Desenvolvimento do *backend*

Neste capítulo, é explorado o desenvolvimento do backend do sistema, o qual é baseado na utilização da estrutura de dados fornecida pelo Firebase,

3.3.1 *BasePoint*

As propriedades do objeto *BasePoint* são mostradas na tabela 2.

Tabela 2. Esquema de *BasePoint*.

Propriedade	**Tipo**	**Descrição**
pointId	*String*	ID do ponto
coordinates	*Geopoint*	Coordenada geográfica do ponto
floor	*Int*	Andar em que se encontra o ponto
pathCoordinates	*List<String>*	*ID's* dos outros *basePoints* com o qual o ponto pode criar um caminho.
region	*String*	Unidade acadêmica onde o ponto está localizado

Fonte: Autoria própria (2023).

A classe *basePoint* é uma das principais classes do sistema. A partir dela que é mapeada toda a universidade e são criadas as rotas para todos os lugares mapeados. Uma vez instanciada, a classe contem cinco atributos: o *pointoId* que é o ID do ponto, único para cada documento da coleção; *coordinates* que contem as informações de latitude e longitude do ponto; *floor* que representa o andar onde o ponto se encontra, *pathCoordinates* que possui os *IDs* de todos os outros ponto com o qual ele pode criar caminho *region* que guarda a informação de qual unidade acadêmica aquele ponto é pertencente.

3.3.2 *Place*

As propriedades do objeto *Place* são mostradas na tabela 3.

Tabela 3. Esquema de *Place*.

Propriedade	**Tipo**	**Descrição**
placeId	*String*	Id do lugar
description	*String*	Descrição do lugar
floor	*Int*	Andar em que se encontra o lugar
localization	*Geopoint*	Coordenada geográfica do lugar
department	*String*	Nome do prédio ou corredor que o lugar está localizado
name	*String*	Nome do lugar
pathCoordinates	*List<BasePoint>*	Informações dos pontos base que o lugar pode criar rota
pictures	*List<String>*	Fotos do lugar
sector	*String*	Setor do campus onde está o lugar
type	*String*	Indicação do tipo de lugar
zone	*String*	Nome da unidade acadêmica onde o lugar está localizado

Fonte: Autoria própria (2023).

A classe *Place* tem a finalidade de representar cada lugar mapeado na universidade. O desenvolvimento e a construção dessa classe foram feitos de modo a oferecer ao usuário a maior quantidade possível de informações do lugar, como nos atributos *description*, *floor*, *department*, *pictures*, sector, type e zone.

Um atributo bem importante dessa classe é o *pathCoordinates*. É a partir dos objetos contidos em sua lista que será criada a rota para esse lugar, visto que estes

são os pontos base de *geolocalização* com o qual esse lugar pode se relacionar para criar um caminho.

3.3.3 *Teacher*

As propriedades do objeto *Teacher* são mostradas na tabela 4.

Tabela 4. Esquema de Teacher.

Propriedade	Tipo	Descrição
teacherId	*String*	Id do professor
teacherName	*String*	Nome do professor

Fonte: Autoria própria (2023).

Esta classe é utilizada para definir o objeto professor. Como se trata de um objeto simples, conta com apenas dois construtores: teacherId, que retorna o id do professor no banco de dados, e teacherName, que guarda o nome do professor.

3.3.4 *Subject*

As propriedades do objeto *Subject* são mostradas na tabela 5.

Tabela 5. Esquema de *Subject*.

Propriedade	Tipo	Descrição
subjectId	*String*	Id da matéria
subjectName	*String*	Nome da matéria
subjectCode	*String*	Código da matéria

Fonte: Autoria própria (2023).

A classe subject define as matérias que são ministradas na sala de aula. Esse objeto conta com três construtores: subjectId, que define o id da matéria no banco de dados, subjectName, que guarda o valor do nome da matéria, e subjectCode, que guarda o código da matéria utilizado em sua descrição.

3.3.5 *Schedule*

As propriedades do objeto Schedule são mostradas na tabela 6.

Tabela 6. Esquema de Schedule.

Propriedade	Tipo	Descrição
time	*Int*	Horário de inicio da aula
subject	*Subject*	Matéria
teacher	*List<Teacher>*	Professores responsáveis por ministrar a matéria

Fonte: Autoria própria (2023).

Essa classe é utilizada para criar objetos que representam diferentes horários de aula, onde cada objeto terá um atributo do tipo int que representa o inicio da aula, um do tipo *Subject* que contem as informações da matéria que será ministrada e outra do tipo *List<Teacher>* que contem as informações dos professores responsáveis pela matéria.

3.3.6 *ClassSchedule*

As propriedades do objeto *ClassSchedule* são mostradas na tabela 7.

Tabela 7. Esquema de *ClassSchedule*.

Propriedade	Tipo	Descrição
day	*Int*	Dia da semana
schedules	*List<Schedules>*	Informações de aula

Fonte: Autoria própria (2023).

Essa classe tem a finalidade de instanciar objetos que descrevem as aulas que serão ministradas para cada dia da semana. Cada objeto incluirá dois atributos: um inteiro que representa o dia da semana e um objeto do tipo *List<Schedules>* que contém os diferentes horários de aula daquele dia.

3.3.7 *College*

As propriedades do objeto *College* são mostradas na tabela 8.

Tabela 8. Esquema de *College*.

Propriedade	Tipo	Descrição
collegeId	*String*	Id da unidade acadêmica
collegeName	*String*	Nome da unidade acadêmica
acronym	*String*	Acrônimo da unidade acadêmica

Fonte: Autoria própria (2023).

A função desta classe é criar instâncias que descrevem uma unidade acadêmica da universidade. Os atributos *collegeName* e *acronym* descrevem, respectivamente, o nome da unidade acadêmica e o seu acrônimo.

3.3.8 *Classroom*

As propriedades do objeto *Classroom* são mostradas na tabela 9.

Tabela 9. Esquema de *Classroom*.

Propriedade	Tipo	Descrição
roomId	*String*	Id do evento
college	*College*	Unidade acadêmica em que está localizada a sala
place	*Place*	Descrição do evento
roomName	*String*	Nome ou número da sala
classSchedule	*List<ClassSchedule>*	Aulas da sala

Fonte: Autoria própria (2023).

Esta classe foi projetada para criar instâncias que descrevem uma sala de aula ou laboratório da universidade. Nessa classe o atributo *classSchedule* contem as informações das aulas programadas para aquela sala durante os dias da semana. O atributo *college* contem as informações sobre a unidade acadêmica a qual a sala pertence. E o campo *place* representa o lugar onde sala está localizada e serve para o usuário criar uma rota até ele.

3.3.9 *CollegeClassroom*

As propriedades do objeto *CollegeClassroom* são mostradas na tabela 10.

Tabela 10. Esquema de *CollegeClassroom.*

Propriedade	Tipo	Descrição
college	*College*	Unidade acadêmica em que está localizada a sala
classrooms	*List<Classroom>*	Título do evento

Fonte: Autoria própria (2023).

A função principal desta classe é separar as salas e laboratórios de acordo com a unidade acadêmica a que eles pertencem. Cada instância possui dois atributos: *college*, do tipo *College*, que representa a unidade e classrooms do tipo *List<Classroom>* que representa todas as suas salas ou laboratórios.

3.3.10 *Event*

As propriedades do objeto *Event* são mostradas na tabela 11.

Tabela 11. Esquema de *Event*.

Propriedade	Tipo	Descrição
eventId	*String*	Id do evento
title	*String*	Título do evento
description	*String*	Descrição do evento
date	*Timestamp*	Date e horário do evento
image	*String*	Imagem de divulgação do evento
place	*Place*	Objeto para o lugar onde o evento ocorrerá

Fonte: Autoria própria (2023).

O campo *eventId* é a identificação de cada objeto, ou seja, é único para cada vez que a classe é instanciada. O campo “*place*” é um objeto do tipo *Place* que representa o lugar onde o evento ocorrerá e serve para o usuário criar uma rota até ele.

3.3.11 *BusLine*

As propriedades do objeto *BusLine* são mostrados na tabela 12.

Tabela 12. Esquema de *BusLine*.

Propriedade	Tipo	Descrição
busId	*String*	Id do ônibus
busName	*String*	Nome ou número do ônibus
Itinerary	*String*	Itinerário do ônibus
leaving	*String*	Local de inicio da rota do ônibus
weekendsHours	*List<Timestamp>*	Horários de partida do ônibus durante o fim de semana
workingDaysHours	*List<Timestamp>*	Horário de partida do ônibus durante os dias úteis

Fonte: Autoria própria (2023).

O campo *busId* é a identificação de cada objeto, ou seja, é único para cada vez que a classe é instanciada. Nos campos *weekendsHours* e *workingDaysHours* foi realizado um processo de ordenação crescente nas *Lists*, organizando os valores do menor ao maior.

3.3.12 Funções do *BusLine*

No objeto *BusLine* a única função criada é o *getNextBus* que retorna o tempo estimado para o próximo ônibus sair da estação e iniciar a rota. Para calcular esse tempo, são extraídos os *timestamps* do *weekendsHours* ou *workingDaysHours* formando uma List, onde cada *timestamp* representa um horário. Apenas os *timestamps* que estão agendados para depois do tempo atual são considerados nessa extração. A partir desses registros, é selecionado o menor valor de *timestamp*, indicando o horário mais próximo no futuro. Com base nesse menor *timestamp*, é calculado o intervalo de tempo restante até a chegada do próximo ônibus subtraindo-o do tempo atual.

3.3.13 Criação de rotas

Com o intuito de traçar rotas dentre os pontos mapeados na universidade, a classe *Graphs* implementa o algoritmo de *Dijkstra*, uma técnica utilizada em teoria dos grafos para encontrar o caminho mais curto entre dois vértices em um grafo não direcionado. Na classe os vértices são representados por strings e as arestas entre os vértices têm pesos associados (valores numéricos). A seguir são listados seus métodos:

- *addVertex*: permite adicionar um vértice ao grafo. Se o vértice já existir, ele não será adicionado novamente. Caso contrário, um novo vértice é criado com um mapa vazio de adjacências.
- *addEdge*: permite adicionar uma aresta entre dois vértices com um peso associado. Ele verifica se os vértices de origem e destino existem (adicionando-os, se necessário) e, em seguida, atualiza os mapas de adjacências para refletir a nova aresta entre eles. Importante notar que o grafo é não direcionado, ou seja, a aresta é adicionada tanto para o vértice de origem quanto para o vértice de destino.
- *getPoints*: calcula o caminho mais curto entre dois vértices (origem e destino) usando o algoritmo de Dijkstra. Este processo segue as seguintes etapas: inicializa um mapa de distâncias, um mapa de predecessores e uma lista de vértices não visitados. Para cada vértice no grafo, a distância é inicializada como infinito, a menos que seja o ponto de origem, que começa com distância zero. Inicializa os predecessores de todos os vértices como vazio e adiciona todos os vértices à lista de vértices não visitados. Enquanto ainda existirem vértices não visitados, o método busca o vértice não visitado com a menor distância, o que é realizado com a ajuda da função *_getShortestDistanceVertex*. Em seguida, relaxa todas as arestas do vértice atual em direção aos seus vizinhos, atualizando as distâncias e predecessores se encontrar um caminho mais curto. Quando o vértice de destino é alcançado, o caminho mais curto é reconstruído chamando a função *_getPath*, que retorna a lista de vértices que compõem o caminho mais curto.
- *getDistance*: calcula a distância entre dois vértices (origem e destino) usando o mesmo algoritmo de *Dijkstra*. Ele retorna a distância entre os dois vértices como um valor numérico.

- *_getShortestDistanceVertex*: método privado que encontra o vértice não visitado com a menor distância acumulada até o momento.
- _getPath: método privado que reconstrói o caminho percorrido a partir do vértice de destino até o vértice de origem, usando as informações de anteriores calculadas durante o algoritmo de *Dijkstra*.

Além da classe *Graphs*, foi criada a classe *CreateRoute*. Esta classe é responsável por fazer o intermédio entre o *GoogleMaps* e a classe *Graphs* na aplicação. Dessa maneira, ela visa criar e gerenciar uma rota de navegação diretamente no mapa, com base na localização atual do usuário obtida através do *geolocator* (Pub.Dev, 2023j). A seguir são listados suas principais funcionalidades e métodos:

- Construtor *CreateRoute*: O construtor da classe recebe vários parâmetros, incluindo uma lista de *polilinhas* (*polylines*), um objeto *BasePointsRequest* (*basePoints*) para obter informações sobre pontos de interesse, uma posição (position) que representa a localização atual do usuário, um ponto de partida opcional, que é utilizado quando o usuário não está dentro da área da universidade (*startPoint*) e um local de destino (*place*).
- *startNavigation*: método responsável por iniciar a navegação. Ele cria uma instância da classe *Graphs* para trabalhar com gráficos de pontos de interesse e chama métodos internos para adicionar pontos ao gráfico, calcular a rota, definir os limites do mapa e exibir a rota no mapa.
- *_addPoints*: método que adiciona pontos de interesse ao gráfico utilizado para calcular a rota. Ele cria arestas entre pontos próximos com base na distância entre eles.
- *verifyArea*: método que verifica se o usuário está dentro de uma área específica (por exemplo, "Setor Norte") com base em informações geográficas e em um polígono que delimita a área. Ele usa a biblioteca maps*_toolkit* (Pub.Dev, 2023o) para determinar se as coordenadas do usuário estão dentro do polígono delimitado pela área.
- *_calculateDistance*: método que calcula a distância entre dois pontos geográficos (representados como objetos LatLng) usando a fórmula da distância euclidiana.

- *getBasePointByPointId*: método que recebe um identificador de ponto (*pointId*) e retorna informações sobre o ponto de interesse correspondente a esse identificador a partir do objeto *basePoints*.
- *getNearestBasePoint*: este método calcula o ponto de interesse mais próximo com base na localização atual do usuário. Ele calcula as distâncias entre a posição do usuário e todos os pontos de interesse e retorna o ponto mais próximo.
- *initLocalization*: este método determina a localização inicial a ser exibida na rota com base na configuração. Se o usuário estiver dentro de uma área segura, ele usa a localização do usuário; caso contrário, usa o ponto de partida especificado.
- *finishNavigation*: este método finaliza a navegação, removendo a rota exibida no mapa.

3.3.14 Idioma

Um grande diferencial dessa aplicação é sua versatilidade em oferecer ao usuário a opção de escolha do idioma. Nesta versão do aplicativo são oferecidos os idiomas português e inglês.

Para controlar essa funcionalidade, foi criada a classe *LanguageManager*, que nada mais é que uma abstração para gerenciar as *strings* de texto em diferentes idiomas, permitindo a internacionalização do aplicativo. Ela fornece um conjunto de constantes de *string* que representam várias mensagens ou rótulos exibidos em um aplicativo.

A classe *LanguageManager* é abstrata e define um contrato que outras classes concretas devem implementar para fornecer as *strings* em um idioma específico. Ela tem uma série de campos de *string* que representam diferentes mensagens ou rótulos usados no aplicativo.

Existem duas classes concretas que implementam a classe *LanguageManager* para dois idiomas diferentes: PT_BR (Português Brasileiro) e EN_US (Inglês dos Estados Unidos). Cada uma dessas classes fornece as *strings* traduzidas para o idioma correspondente.

Por exemplo, a classe PT_BR implementa as *strings* em português brasileiro para rótulos como "Explorar", "Eventos", "*Ensalamento*", etc. Enquanto a classe EN_US implementa as mesmas *strings* em inglês, como "Explore", "*Events*", "*Classroom*", etc.

Esse padrão permite que o aplicativo seja internacionalizado para diferentes idiomas sem a necessidade de alterar o código-fonte do aplicativo. Em tempo de execução, o aplicativo pode selecionar a classe *LanguageManager* apropriada com base nas preferências de idioma do usuário e usar as *strings* correspondentes a esse idioma para exibir o conteúdo no aplicativo. Isso torna o aplicativo acessível a um público mais amplo, independentemente do idioma preferido do usuário.

Com o intuito de armazenar a linguagem escolhida pelo usuário na memória do dispositivo foi criada a classe *GetStorageManager*. Esta classe é responsável por gerenciar as preferências do aplicativo usando a biblioteca *GetStorage* (Pub. Dev, 2023) Ela fornece métodos para armazenar e recuperar informações relacionadas ao idioma, permitindo a persistência dessa configuração entre as sessões do usuário. Dentre os elementos desta classe temos:

- final *box* = *GetStorage*: A classe *GetStorage* é importada da biblioteca *get_storage* (Puv. Dev, 2023) e é usada para criar uma instância de armazenamento persistente. Essa instância, chamada de *box*, será usada para armazenar e recuperar os valores das preferências;
- *String languageKey*: Variável que armazena a chave (identificador único) para as preferências de idioma no armazenamento. Essa chave é usada para acessar o valor correspondente no armazenamento;
- *setLanguage*(String language): Este método permite definir o idioma preferido pelo usuário. Ele recebe uma *string* language como parâmetro e armazena essa informação no armazenamento box associada à chave *languageKey*.
- *getLanguage*: Este método recupera o idioma preferido do usuário. Ele verifica se há dados armazenados para a chave *languageKey* no armazenamento box usando *box*.*hasData*(*languageKey*). Se houver dados, ele retorna o idioma armazenado. Caso contrário, ele retorna "pt_BR" como valor padrão.

3.3.15 Requisições do *Firebase*

De maneira geral, todas as requisições de dados do Firebase são feitas de forma semelhante, apesar de cada requisição ser feita em uma classe diferente, de acordo com o objeto que se deseja utilizar. Cada classe possui duas variáveis e uma função declarados: uma *List* com os objetos da requisição, uma variável que indica o status da requisição e um método que faz a requisição. Na Figura 10, por exemplo, é mostrada como é feita a requisição dos objetos *BasePoint*, utilizados para criar os pontos de mapeamento da universidade.

```dart
class BasePointsRequest {

  List<BasePoint> basePointsRequest = [];
  final status = Status.none.obs;

  basePointRequest() async {
    bool connectionStatus = await InternetConnectionChecker().hasConnection;
    if (connectionStatus) {
      LogConsole().p("Internet connected", LogStatus.internetConnection);
      await FirebaseFirestore.instance
          .collection('basePoints')
          .get()
          .then((QuerySnapshot querySnapshot) {
        for (var doc in querySnapshot.docs) {
          var basePoint = BasePoint(
              coordinates: doc["coordinates"],
              floor: doc["floor"],
              pathCoordinates: doc["pathCoordinates"],
              pointId: doc["pointId"],
              region: doc["region"]);
          basePointsRequest.add(basePoint);
        }
        status.value = Status.success;
        LogConsole().p("BASE POINTS request successfully", LogStatus.firebaseFirestoreSuccess);
      }).catchError((error) {
        status.value = Status.error;
        LogConsole().p("BASE POINTS request failure. Erro: $error", LogStatus.firebaseFirestoreFailure);
      });
    } else {
      status.value = Status.internetError;
      LogConsole().p("Internet disconnected", LogStatus.internetConnection);
    }
  }
```

Figura 10. Requisições de dados de *BasePoints* do *Firebase*. Fonte: Autoria própria (2023).

Esta classe utiliza uma lista de objetos para armazenar os resultados de uma consulta ao Firestore. Além disso, possui uma variável de status do tipo enum chamada Status, que informa o estado da requisição no frontend, podendo ser "success," "error," "internetError" ou "none." A principal função assíncrona da classe começa verificando a conectividade à Internet com a biblioteca *InternetConnectionChecker*. Se houver uma conexão, ela realiza uma consulta ao *Firestore* através do método "*FirebaseFirestore.instance.collection('nome da*

coleção').get()", no caso “basePoints”, e processa os documentos resultantes, criando objetos que são adicionados à lista. O status da requisição é definido como "success" em caso de sucesso na consulta ou "error" em caso de erro. Quando não há conexão com a Internet, o status é definido como "internetError".

3.4 *Controller*

De modo a criar o gerenciamento de estado do aplicativo, foram desenvolvidas as classes *controller*. Estas classes desempenham papéis essenciais, atuando como controladores intermediários entre o backend (os dados e lógica de negócios) e o *frontend* (a interface do usuário). Elas gerenciam a lógica de apresentação, a comunicação com o *backend* e a interação do usuário.

O software apresenta 5 classes *controller*: *ExploreController*, *EventsController*, *ClassroomController*, *BusController* e *HomeController*. As 4 primeiras são responsáveis por determinar o comportamento das principais funcionalidades do aplicativo perante ao usuário e a *HomeController* é o controlador responsável por elas.

Todas estas classes utilizam a declaração *extends GetxController*, ou seja, elas estão usando a funcionalidade fornecida pela biblioteca *GetX* (Pub.Dev, 2023m) para gerenciamento de estado e gerenciamento de dependências.

A seguir é explicado o funcionamento de cada classe:

3.4.1 *ExploreController*

O *ExploreController* é responsável pelo controle das funcionalidades relacionadas à exploração do mapa e da universidade. Aqui estão suas principais responsabilidades:

- Inicialização e configuração do controlador.
- Gerenciamento do estado do aplicativo, como a mudança de modos (normal, pesquisa, navegação).
- Recuperação e gerenciamento de pontos de referência (*BasePoints*) da universidade.
- Recuperação de informações sobre locais (*Places*) da universidade.
- Gerenciamento da exibição de rotas no mapa.

- Manipulação das ações do usuário, como tocar em um local para exibir detalhes.
- Manipulação da localização do usuário e atualizações de mapa.

3.4.2 *EventsController*

O *EventsController* é responsável pelo controle das funcionalidades relacionadas a eventos na universidade. Suas principais responsabilidades incluem:

- Inicialização e configuração do controlador.
- Recuperação e gerenciamento de eventos da universidade.
- Pesquisa e filtragem de eventos.
- Manipulação das ações do usuário, como tocar em um evento para exibir detalhes.
- Gerenciamento da interface do usuário relacionada a *eventos*.

3.4.3 *ClassroomController*

O *ClassroomController* é responsável pelo controle das funcionalidades relacionadas às salas de aula e departamentos da universidade. Suas principais responsabilidades incluem:

- Inicialização e configuração do controlador.
- Recuperação e gerenciamento de informações sobre salas de aula e departamentos da universidade.
- Manipulação das ações do usuário, como tocar em uma sala de aula ou departamento para exibir detalhes.
- Gerenciamento da interface do usuário relacionada a salas de aula e departamentos.

3.4.4 BusController

O *BusController* é responsável pelo controle das funcionalidades relacionadas às linhas de ônibus da universidade. Suas principais responsabilidades incluem:

- Inicialização e configuração do controlador.

- Recuperação e gerenciamento de informações sobre as linhas de ônibus da universidade.
- Manipulação das ações do usuário, como visualizar informações sobre as linhas de ônibus.
- Gerenciamento da interface do usuário relacionada a linhas de ônibus.

3.4.5 *HomeController*

O *HomeController* atua como um controlador principal que coordena a navegação entre as diferentes partes do aplicativo. Suas principais responsabilidades incluem:

- Inicialização e configuração do controlador e das visualizações correspondentes.
- Coordenação da exibição de diferentes telas (*Explore*, Eventos, Salas de Aula, Ônibus) com base na seleção do usuário.
- Navegação entre as diferentes visualizações do aplicativo.
  Manipulação de ações específicas, como tocar em um local ou evento para abrir a visualização correspondente.
- Integração com os controladores específicos (*ExploreController, EventsController, ClassroomController, BusController*) para coordenar ações e estados.

3.5 Desenvolvimento do *frontend*

3.5.1 Disposição das funcionalidades

A disposição das funcionalidades do aplicativo é baseada no *widget BottomNavigationBar*. Este componente de UI fica situado na parte inferior da tela. Ele contém ícones que representam seções diferentes do aplicativo, permitindo que os usuários alternem facilmente entre elas. A divisão destas seções é feita com base nas funções de explorar o mapa da universidade, busca de eventos, acesso ao ensalamento e acesso as informações das linhas de ônibus, como mostrado na Figura 11.

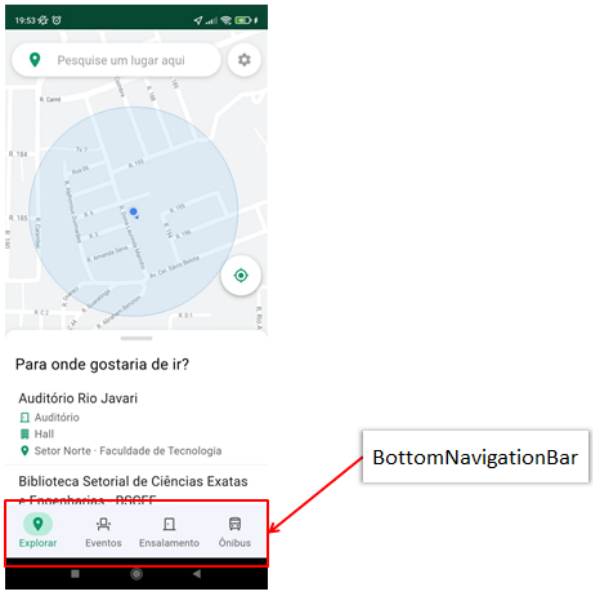


Figura 11. Disposição das principais funcionalidades da aplicação na tela. Fonte: Autoria própria (2023).

3.5.2 Telas de Explore

Essa tela conta com uma entrada de texto, onde é possível pesquisar os lugares mapeados, e um botão de acesso às configurações de idioma no topo. Centralizado pode-se ver o mapa onde são criadas as rotas e um botão que quando clicado retorna a localização atual do usuário no mapa. Na parte inferior tem-se o *widget DraggableScrollableSheet* com a lista de lugares mapeados na universidade. Este componente da tela nada mais é que um *widget* que combina a funcionalidade de um *Draggable* (permitindo que um *widget* seja arrastado) e um *CustomScrollView* (que é uma visualização rolável personalizável). Isso cria uma folha que pode ser arrastada para cima ou para baixo para revelar ou ocultar conteúdo adicional.

Nesta tela, a versatilidade é essencial, para isso são definidos diferentes modos de exibição de acordo com as necessidades específicas dos usuários. A partir disso são definidos os modos que são apresentados da seção 3.5.3 à 3.5.8.

3.5.2.1 Normal

No modo normal, apresentado na Figura 11, os usuários têm uma visão geral da tela de explorar. Eles podem explorar o mapa, pesquisar e acessar lugares específicos da universidade. Caso ele clique na entrada de texto, o modo pesquisa é ativado. Caso clique em algum lugar na lista presente no *draggable*, o modo

visualização do lugar é ativado. E caso ele arraste o draggable até o topo da tela, o modo *draggable* expandido é ativado.

3.5.2.2 Pesquisa

O modo de pesquisa é ativado quando o usuário deseja encontrar um local específico. Ele pode usar a barra de pesquisa para digitar o nome do local ou uma palavra-chave relevante. O aplicativo retornará resultados correspondentes, facilitando a localização de um destino desejado.

Como mostrado na Figura 12, neste modo o *draggable* sobe instantaneamente até o topo. Ao lado esquerdo da barra de pesquisa, um botão fica ativo que quando é clicado volta para o modo normal. Quando esta barra possui algum texto, um botão ao lado direito fica ativo para limpar a pesquisa, ilustrado na figura 13.

Figura 12. Tela explore em modo pesquisa. Fonte: Autoria própria (2023).

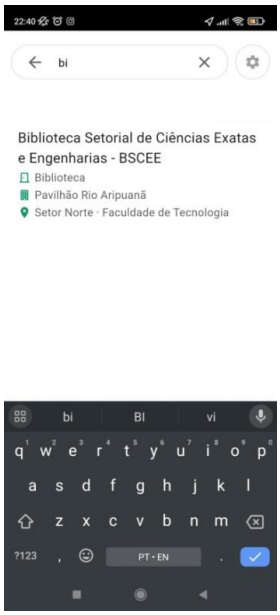


Figura 13. Tela explore em modo pesquisa quando a caixa de texto está preenchida. Fonte: Autoria própria (2023).

3.5.2.3 *Draggable* Expandido

Mostrado na Figura 14, esse modo é acionado quando o usuário deseja ver a lista de lugares mapeados de forma expandida. Ele é desativado quando o usuário arrasta a lista para baixo.

Figura 14. Tela explore em modo draggable expandido. Fonte: Autoria própria (2023).

3.5.2.4 Visualização do lugar

Ao clicar em um local de interesse, como um laboratório, sala de aula ou biblioteca, o usuário entra no modo de visualização desse local. Ele pode ver informações detalhadas, incluindo fotos, campus onde está localizado e descrições, Conforme ilustrado nas Figuras 15 e 16. Além disso, podem iniciar a navegação para o local se desejar visitá-lo.

Assim que esse modo é ativo, na parte superior da tela a barra de pesquisa dá lugar para somente para um botão ao lado esquerdo onde é possível voltar para o modo normal. Na parte inferior, o *draggable* não pode mais ser arrastado até em cima, tendo assim a sua altura fixa e a lista de lugares é alterada para as informações do lugar específico escolhido. E o mapa recebe um marcador centralizado na localização escolhida.

Ao clicar no botão rota no centro do *draggable*, é verificado se a localização do usuário está dentro da universidade ou não. Caso sim, o modo navegação se torna ativo instantaneamente e caso contrário um *dialog* é mostrado no centro da tela onde o usuário pode escolher um ponto de partida a partir das opções listadas para iniciar sua rota, como apresentado na Figura 17.

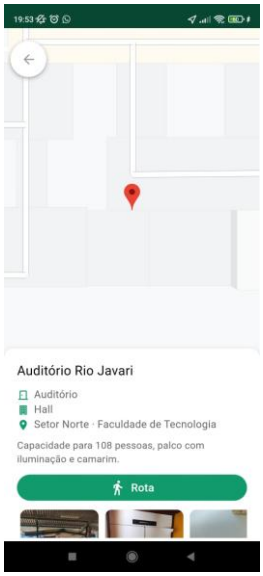


Figura 15. Tela explore em modo visualização do lugar. Fonte: Autoria própria (2023).

Figura 16. Tela explore em modo visualização do lugar com vizualização de fotos. Fonte: Autoria própria (2023).

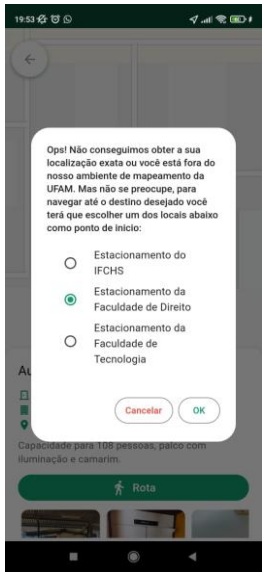


Figura 17. Dialog de escolha de um ponto de partida da rota. Fonte: Autoria própria (2023)

3.5.2.5 Navegação

Como mostrado na Figura 18, neste modo o usuário obtém a rota detalhada até o seu local de destino a partir de *polilinhas* que são desenhadas no mapa. Na parte inferior, o *draggable* tem sua altura diminuída, mostrando ao lado esquerdo um botão que quando clicado volta para o modo de visualização do lugar e no centro um texto com a distância do ponto de partida até o destino. Além disso, mostra também de onde o usuário está partindo.

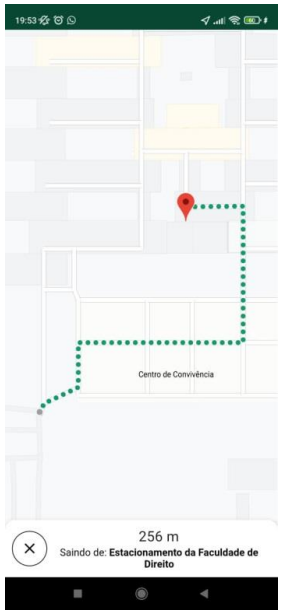


Figura 18. Tela explore em modo navegação. Fonte: Autoria própria (2023).

3.5.2.6 Sem conexão com a internet

Este modo é exibido a partir do momento que o aplicativo detecta falta de conexão com a internet, limitando o acesso a informações para o usuário. Neste modo aparece apenas um botão de tentar novamente é mostrado no centro da tela que quando clicado tenta fazer uma nova requisição de dados do mapa e de lugares ao servidor do *Firebase*, como mostrado na Figura 19.

Figura 19. Tela explore em modo sem internet. Fonte: Autoria própria (2023).

3.5.3 Telas de Eventos

No topo da tela, há uma barra de pesquisa onde os usuários podem inserir palavras-chave para procurar eventos específicos. O restante da tela é ocupado por uma lista de eventos. Cada evento é exibido como um cartão que contém informações sobre o título do evento, data, hora e local, como ilustrado na Figura 20.

Figura 20. Tela de eventos. Fonte: Autoria própria (2023).

Conforme a Figura 21, quando o usuário clica e digita na barra de pesquisa, os resultados são atualizados automaticamente na tela, caso sejam encontrados. Caso contrário, uma tela informando que não foram encontrados resultados é apresentada, como ilustrado na figura 22.

Figura 21. Tela de eventos em modo de pesquisa. Fonte: Autoria própria (2023).

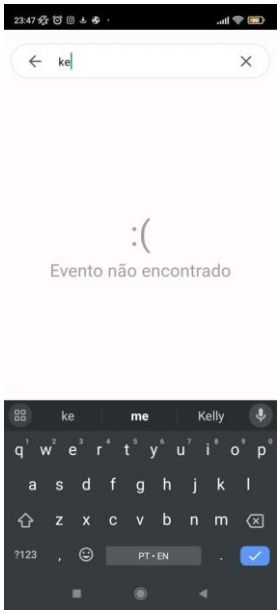


Figura 22. Tela de eventos em modo pesquisa sem resultados encontrados. Fonte: Autoria própria (2023).

Quando o usuário seleciona um evento, o aplicativo abre uma nova tela que exibe as informações detalhadas sobre ele, como apresentado na Figura 23.

O topo desta página contém uma imagem de divulgação do evento, quando esta existir. Se o evento estiver associado a um local, um botão de navegação será exibido na parte inferior da tela. Os usuários podem tocar neste botão para obter uma rota para o local do evento.

O meio da pagina é ocupado pelos detalhes do evento, incluindo título, data, hora, localização e descrição.

Figura 23. Tela de detalhes dos eventos. Fonte: Autoria própria (2023).

3.5.4 Telas de Ensalamento

O fluxo das telas de ensalamento é dividido com base nas unidades acadêmicas da UFAM. A primeira tela, ilustrada na Figura 24, possui a lista com todas as unidades contidas no banco de dados. Cada item na lista representa uma unidade e exibe seu acrônimo e nome. O topo da tela contém um cabeçalho que exibe o nome da página, no caso, ensalamento.

Figura 24. Tela das unidades acadêmicas. Fonte: Autoria própria (2023).

Ao selecionar uma unidade na tela inicial, o usuário é direcionado para uma nova tela, mostrada na Figura 25, que exibe uma lista de salas de aula associadas a essa unidade. O topo da tela possui um cabeçalho que exibe o nome da unidade acadêmica. Cada sala de aula é representada como um item na lista. Ao tocar em uma sala de aula, o usuário é levado para outra tela para obter mais informações sobre essa sala de aula.

Figura 25. Tela das salas de uma unidade acadêmica. Fonte: Autoria própria (2023).

Conforme mostrado na Figura 26, tela da sala de aula exibe informações como nome, um botão de retorno e, se disponível, um botão para visualizar a rota até a sala no mapa. Os detalhes da sala de aula são organizados em uma lista expansível. Cada dia da semana pode ser tocado para expandir e exibir os horários das aulas, conforme a Figura 27. Se não houver aula em um determinado horário, a tela exibe "Tempo livre".

Figura 26. Tela de horários de uma sala de aula. Fonte: Autoria própria (2023).

Figura 27. Tela de horários de uma sala de aula com a lista expandida. Fonte: Autoria própria (2023).

3.5.5 Tela de Ônibus

Conforme a Figura 28, o topo da tela contém um cabeçalho que exibe o nome da página, no caso, ônibus. O restante é ocupado pelo conteúdo principal, uma lista de cartões expansíveis que contem as informações de cada linha de ônibus, como o nome, local de saída e um cronometro com o tempo até de saída do próximo ônibus.

Cada cartão, quando clicado, se expande para mostrar os horários de saída do ônibus para os dias úteis e fins de semana, como mostrado na Figura 29. Isso ajuda os usuários a se programarem para pegar a linha de ônibus próximo do seu horário desejado.

Figura 28. Tela de informações dos ônibus. Fonte: Autoria própria (2023).

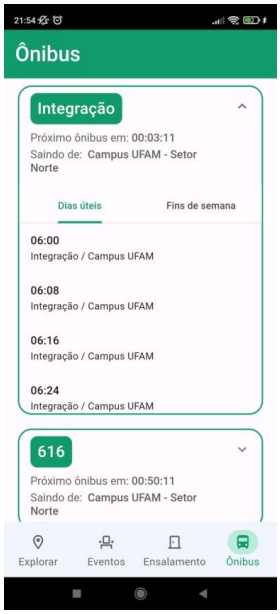


Figura 29. Tela de informações dos ônibus com a lista expandida. Fonte: Autoria própria (2023).

3.5.6 Tela de configurações do idioma

Conforme a Figura 30, na parte superior da tela de explorar é possível encontrar um botão que quando clicado leva a tela de configurações onde é possível alterar o idioma do sistema.

O corpo principal da tela é uma lista com um único elemento quando clicado permite ao usuário escolher o idioma do aplicativo, como ilustrado na figura 45. No topo dessa tela encontramos o título "Configurações", indicando claramente a sua função. À esquerda da barra de navegação, há um botão de retorno, representado por uma seta apontando para a esquerda. Quando pressionado, esse botão ativa uma caixa de diálogo de confirmação, perguntando ao usuário se deseja salvar as configurações feitas antes de sair da tela, como mostrado na Figura 31.

Figura 30. Tela de configurações do idioma. Fonte: Autoria própria (2023).

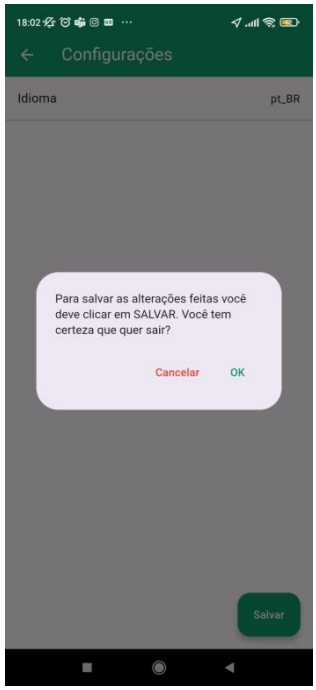


Figura 31. *Dialog* apresentado quando o botão de voltar é clicado. Fonte: Autoria própria (2023).

De acordo com a Figura 32, ao tocar no elemento “idioma” da lista configuração, uma caixa de diálogo é exibida com uma lista de idiomas disponíveis. O idioma atualmente selecionado é exibido como um texto à direita da configuração.

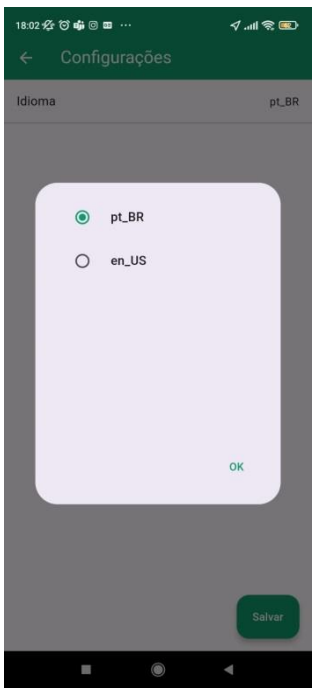


Figura 32. *Dialog* com as opções de idiomas disponíveis. Fonte: Autoria própria (2023).

No final da tela, há um botão "Salvar", representado por um botão flutuante. Ao tocar neste botão, uma nova caixa de diálogo de confirmação é exibida perguntando ao usuário se deseja salvar todas as configurações feitas, como ilustrado na figura 33. Se confirmado, as configurações são salvas, e o aplicativo é reiniciado para aplicar as alterações.

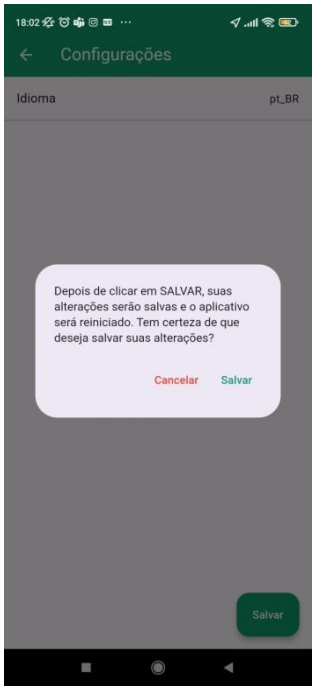


Figura 33. *Dialog* apresentado quando o botão de salvar é clicado. Fonte: Autoria própria (2023).

4 Testes e validação

O aplicativo passou por uma série de testes em diferentes dispositivos e ambientes para garantir sua estabilidade, desempenho e funcionalidade. Dois dispositivos utilizados nos testes foram o *Redmi Note* 9S, com *Android* 11, e o *Galaxy TAB* A8, com *Android* 13.

Nesta aplicação os casos de testes foram divididos de acordo com as quatro principais funcionalidades do aplicativo e suas requisições do banco de dados. A seguir são apresentados estes casos:

4.1 Teste de requisições de dados bem sucedidas do Firebase

Descrição**:** Esse caso de teste verifica se a aplicação é capaz de fazer uma solicitação bem sucedida ao Firebase Firestore para obter os dados relacionados aos *Events*, *Places*, *BasePoints*, *BusLines*, *Teachers*, *Clasrooms*, *Subjects* e *Colleges*.

Passos:

1. Abrir o aplicativo;
2. Acessar todas as telas que realizam a requisição dos dados;
3. Verificar se os dados são exibidos;
4. Verificar no console do Android Studio se o valor da propriedade status é igual a Status.success.

Resultados:

Conforme ilustrado na Figura 34, a validação deste teste se mostrou bem sucedida.

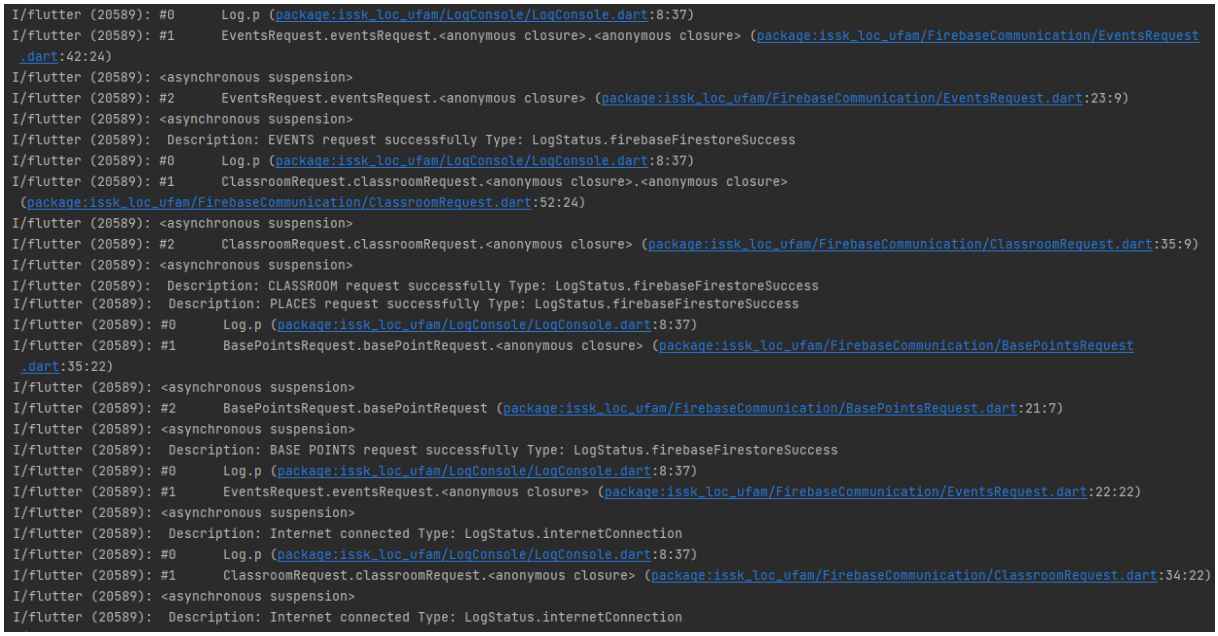


Figura 34. Requisições bem sucedidas do Firebase. Fonte: Autoria própria (2023).

4.2 Teste de requisições de dados mal sucedidas do Firebase devido a falta de conexão com a internet

Descrição**:** Esse caso de teste verifica se a aplicação é capaz de fazer uma solicitação mal sucedida, devido a falta de conexão com a internet, ao Firebase Firestore para obter os dados relacionados aos *Events*, *Places*, *BasePoints*, *BusLines*, *Teachers*, *Clasrooms*, *Subjects* e *Colleges*.

Passos:

1. Desativar o acesso a internet do telefone, seja via Wi-fi ou dados móveis;
2. Abrir o aplicativo;
3. Acessar todas as telas que realizam a requisição dos dados;
4. Verificar se uma mensagem de falta de conexão com a internet é mostrada;
5. Verificar no console do Android Studio se o valor da propriedade status é igual a Status.internetError.

Resultados:

Conforme ilustrado nas Figuras 35 e 36, a validação deste teste se mostrou bem sucedida.

```
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      Requests.subjectsRequest (package:issk_loc_ufam/FirebaseCommunication/Requests.dart:64:20)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439): #2      Future.wait.<anonymous closure> (dart:async/future.dart:522:21)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      Requests.teachersRequest (package:issk_loc_ufam/FirebaseCommunication/Requests.dart:89:20)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439): #2      Future.wait.<anonymous closure> (dart:async/future.dart:522:21)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      EventsRequest.eventsRequest.<anonymous closure> (package:issk_loc_ufam/FirebaseCommunication/EventsRequest.dart:49:22)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      ClassroomRequest.classroomRequest.<anonymous closure> (package:issk_loc_ufam/FirebaseCommunication/ClassroomRequest.dart:59:22)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      BusLinesRequest.busRequest (package:issk_loc_ufam/FirebaseCommunication/BusLinesRequest.dart:44:20)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      Requests.placeRequest (package:issk_loc_ufam/FirebaseCommunication/Requests.dart:123:20)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      Requests.placeRequest (package:issk_loc_ufam/FirebaseCommunication/Requests.dart:123:20)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439): #2      Future.wait.<anonymous closure> (dart:async/future.dart:522:21)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
I/flutter (30439): #0      Log.p (package:issk_loc_ufam/LogConsole/LogConsole.dart:8:37)
I/flutter (30439): #1      Requests.collegesRequest (package:issk_loc_ufam/FirebaseCommunication/Requests.dart:37:20)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439): #2      Future.wait.<anonymous closure> (dart:async/future.dart:522:21)
I/flutter (30439): <asynchronous suspension>
I/flutter (30439):  Description: Internet disconnected Type: LogStatus.internetConnection
```

Figura 35. Requisições mal sucedidas devido a falta de conexão com a internet do Firebase. Fonte: Autoria própria (2023).

Figura 36. Tela sem conexão com a internet. Fonte: Autoria própria (2023).

4.3 Teste de requisições de dados mal sucedidas do Firebase devido a erro desconhecido

Descrição**:** Esse caso de teste verifica se a aplicação é capaz de fazer uma solicitação mal sucedida, devido a um erro desconhecido do software, ao Firebase Firestore para obter os dados relacionados aos *Events*, *Places*, *BasePoints*, *BusLines*, *Teachers*, *Clasrooms*, *Subjects* e *Colleges*.

Passos:

1. Simular um erro na classe que realiza a requisição dos dados;
2. Compilar a aplicação;
3. Abrir o aplicativo;
4. Acessar todas as telas que realizam a requisição dos dados;
5. Verificar se uma mensagem de erro desconhecido é mostrada;
6. Verificar no console do Android Studio se o valor da propriedade status é igual a Status.error.

Resultados:

Conforme ilustrado nas Figuras 37 e 38, a validação deste teste se mostrou bem sucedida.

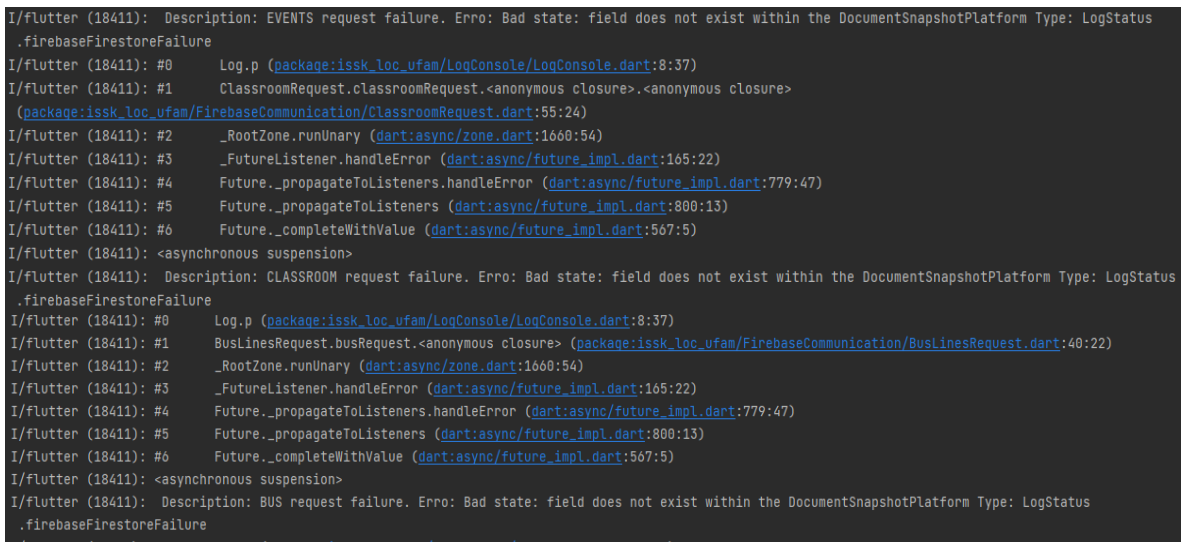


Figura 37. Requisições mal sucedidas devido a erro desconhecido do Firebase. Fonte: Autoria própria (2023).

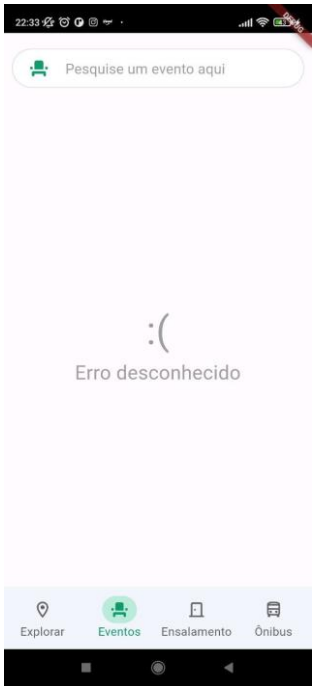


Figura 38. Tela de erro desconhecido. Fonte: Autoria própria (2023).

4.4 Teste de criação de rotas quando o usuário está dentro da UFAM

Descrição**:** Esse caso de teste verifica se a aplicação é capaz de traçar uma rota para um local de interesse quando a localização do GPS do usuário se encontra dentro da UFAM.

Passos:

1. Abrir o aplicativo;
2. Escolher um local para navegar;
3. Clicar no botão “rota”;
4. Verificar se não aparece um *dialog* para escolher um ponto de início;
5. Verificar se a rota é detalhada e precisa;
6. Verificar se a distância é exibida corretamente;
7. Navegar através do mapa até o local escolhido.

Resultados:

Conforme ilustrado na Figura 39, a validação deste teste se mostrou bem sucedida.

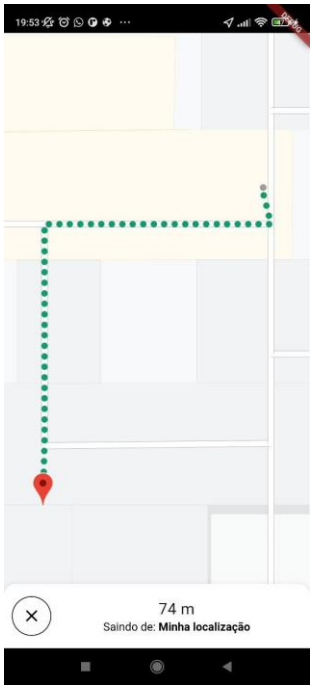


Figura 39. Tela de rota traçada a partir da localização do usuário. Fonte: Autoria própria (2023)

4.5 Teste de criação de rotas quando o usuário está fora da UFAM

Descrição**:** Esse caso de teste verifica se a aplicação é capaz de traçar uma rota para um local de interesse quando a localização do GPS do usuário se encontra fora da UFAM.

Passos:

1. Abrir o aplicativo;
2. Escolher um local para navegar;
3. Clicar no botão “rota”;
4. Verificar se aparece um *dialog* para escolher um ponto de início;
5. Escolher um ponto de início;
6. Clicar em “OK”;
7. Verificar se a rota é detalhada e precisa;
8. Verificar se a distância é exibida corretamente;
9. Navegar através do mapa até o local escolhido.

Resultados:

Conforme ilustrado nas Figuras 40 e 41, a validação deste teste se mostrou bem sucedida.

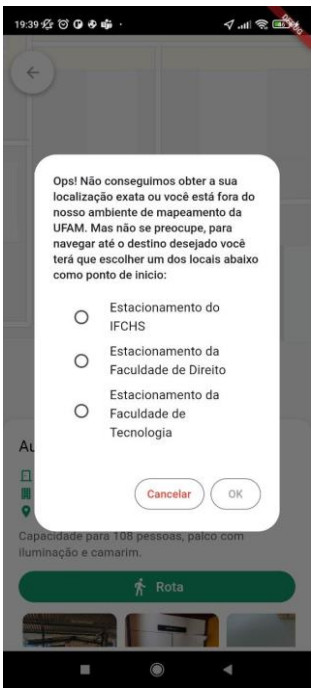


Figura 40. Tela de escolha de ponto de início de rota. Fonte: Autoria própria (2023).

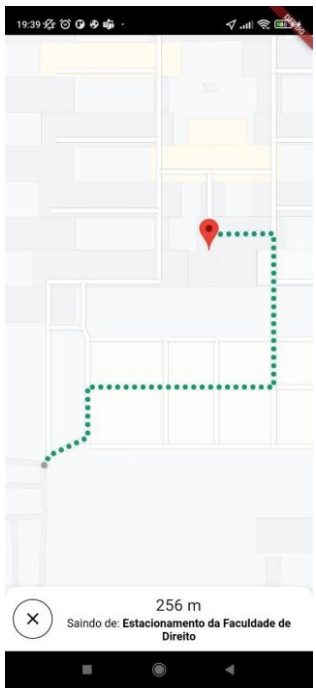


Figura 41. Tela de rota traçada de um ponto escolhido. Fonte: Autoria própria (2023).

4.6 Teste de busca bem sucedida de lugares e eventos

Descrição**:** Esse caso de teste verifica se através da aplicação os usuários podem pesquisar e encontrar locais e eventos específicos que estejam listados no banco de dados.

Passos:

1. Abrir o aplicativo;
2. Escolher a aba de explore ou eventos;
3. Clicar no campo de texto da parte superior da tela;
4. Digitar o nome de um local ou evento que esteja listado;
5. Verificar se esse local ou evento aparece na tela;

Resultados:

Conforme ilustrado nas Figuras 42 e 43, a validação deste teste se mostrou bem sucedida.

Figura 42. Tela de pesquisa bem sucedida de lugares. Fonte: Autoria própria (2023).

Figura 43. Tela de pesquisa bem sucedida de eventos. Fonte: Autoria própria (2023).

4.7 Teste de busca mal sucedida de lugares e eventos

Descrição**:** Esse caso de teste verifica se através da aplicação os usuários podem pesquisar e encontrar locais e eventos específicos que não estejam listados no banco de dados.

Passos:

1. Abrir o aplicativo;
2. Escolher a aba de explore ou eventos;
3. Clicar no campo de texto da parte superior da tela;
4. Digitar o nome de um local ou evento que não esteja listado;
5. Verificar se é mostrada uma tela de local ou evento não encontrado.

Resultados:

Conforme ilustrado nas Figuras 44 e 45, a validação deste teste se mostrou bem sucedida.

Figura 44. Tela de pesquisa mal sucedida de lugares. Fonte: Autoria própria (2023).

Figura 45. Tela de pesquisa mal sucedida de eventos. Fonte: Autoria própria (2023).

5 Considerações finais

Em conclusão, este trabalho atingiu o objetivo de desenvolver uma aplicação que atendesse os interesses da comunidade acadêmica da UFAM, oferecendo um software que permite acessar o mapeamento interno da universidade, a disposição das salas e laboratórios, os horários de aula, os eventos e informações das linhas de ônibus presentes na estação do campus.

Uma das funcionalidades essenciais deste aplicativo é fornecer aos usuários o acesso detalhado ao mapeamento interno da universidade. Isso permite que os usuários explorem o campus com facilidade, identificando a localização precisa de salas, laboratórios, restaurantes e outras instalações. Além de simplificar a navegação, essa funcionalidade também cria uma rota até o local de destino.

Uma característica importante do aplicativo é a capacidade de acessar informações sobre eventos universitários. Isso inclui palestras, conferências, workshops, atividades culturais e outros eventos importantes que ocorrem no campus. Os usuários podem explorar detalhes, como datas, locais e descrições, permitindo que participem ativamente da vida universitária e aproveitem ao máximo as oportunidades oferecidas.

O aplicativo disponibiliza os horários de aulas, permitindo que os usuários acessem informações atualizadas sobre os cursos e disciplinas oferecidos pela universidade. Isso ajuda os estudantes a planejar seus dias com antecedência, garantindo que estejam presentes nas aulas no momento correto. Além disso, a funcionalidade também pode ser útil para professores, facilitando a gestão dos horários das aulas.

A funcionalidade de verificação de horários das linhas de ônibus é crucial para quem depende do transporte público para se locomover no campus. Os usuários podem acessar informações atualizadas sobre os horários de partida das linhas de ônibus na estação do campus, permitindo que planejem suas viagens com precisão. Isso ajuda a reduzir o tempo de espera e a facilitar o deslocamento de estudantes e membros da comunidade acadêmica.

Referente ao processo de desenvolvimento, todo o código foi escrito para que pudesse ter uma fácil manutenção para que a implementação de novas funcionalidades fosse feita da maneira mais eficiente possível. Além disso, o banco

de dados foi arquitetado de modo a oferecer uma manipulação e requisição de informações sempre eficaz. Por fim, o desenvolvimento deste trabalho modernizará a relação da comunidade acadêmica com os espaços físicos da universidade, no entanto é crucial continuar expandindo o conjunto de informações disponíveis no aplicativo, abrangendo ainda mais setores do campus.

5.1 Propostas para trabalhos futuros

Como este é um trabalho que visa atender toda a comunidade acadêmica da UFAM, ainda há espaço para diversos pontos de melhoramento e incrementação do sistema. É sugerido como proposta para futuros trabalhos:

- Mapeamento de toda a UFAM, englobando o Setor Sul e as unidades fora do campus, como Faculdade de Medicina e Faculdade e Odontologia;
- Criação de um site e de um aplicativo *iOS* com as mesmas funcionalidades do aplicativo *Android*;
- Criação do serviço de *login* do sistema integrado ao *ecampus*;
- Criação de serviços de *push notification* informando o começo de eventos e aulas pré-salvos pelo usuário;
- Adição de novas opções de idiomas do aplicativo, além de inglês e português;
- Adição ao sistema de uma funcionalidade que informa o cardápio diário do restaurante universitário;
- Agregar ao sistema novas funcionalidades de acordo com a demanda da universidade.

6 Referências

ALMEIDA, J.V.R. *O que é gerenciamento de estados no Flutter e principais ferramentas*, mar. 2023. Disponível em: https://www.alura.com.br/artigos/gerenciamento-de-estados-flutter-principais-ferramentas. Acesso em: 06 de setembro de 2023.

AMORIM, R.S. *O que é um BaaS – Backend as a Service*? 2017. Disponível em: http://rsamorim.azurewebsites.net/2017/12/05/o-que-e-um-baas-backend-as-a-service. Acesso em: 17 de setembro de 2023.

AZURE. *Banco de Dados NoSQL – O que é NoSQL*? Uma visão geral como introdução, 2023. Disponível em: https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-nosql-database. Acesso em: 24 de setembro de 2023.

BARRO, B.B. *Conheça a Linguagem Dart e Entenda as Polêmicas que a Envolvem*, mar. 2023. Disponível em: https://www.hostinger.com.br/tutoriais/linguagem-dart. Acesso: em 10 de setembro de 2023.

BURBULHAN, F. UBER, WAZE, TINDER: Reflexos da Guerra Fria. 2020. Disponível em: http://apoiogeomatica.com.br/blog/tag/gps/. Acesso em: 15 de setembro de 2023.

CODLABS. *Adicionar o Google Maps ao app*, 2023. Disponível em: https://codelabs.developers.google.com/codelabs/google-maps-in-flutter?hl=pt-br#3. Acesso em: 22 de setembro de 2023.

CONSOLE CLOUD. *Começar a usar o Google Cloud Platform*, s.d. Disponível em: https://console.cloud.google.com/getting-started?pli=1. Acesso em: 07 de setembro de 2023.

CONSOLE FIREBASE. Olá! Este é o Firebase, s.d. Disponível em: Acesso em 07 de setembro de 2023.

DEVELOPER. *Conhecer o Android Studio*. Disponível em: https://developer.android.com/studio/intro?hl=pt-br. Acesso em: 23 de setembro de 2023.

DEVMEDIA. Modelagem de Sistemas Através de UML - Uma Visão Geral. Disponível em: https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913. Acesso em: 07 de setembro de 2023

FIREBASE. *Cloud Firestore*, 2023. Disponível em: https://firebase.google.com/docs/firestore?hl=ptbr#:~:text=O%20Cloud%20Firestore%20%C3%A9%20um,usando%20listeners%20em%20tempo%20real. Acesso em: 25 de setembro de 2023.

_____. *Cloud Storage para Firebase*, 2023. Disponível em: https://firebase.google.com/docs/storage?hl=pt-br)%3A7. Acesso em: 07 de setembro de 2023.

FLUTTER. *Flutter architectural* overview, s.d. Disponível em: https://docs.flutter.dev/resources/architectural-over view. Acesso em: 07 de setembro de 2023.

_____. *Developing packages & plugins*, s.d. Disponível em: https://docs.flutter.dev/packages-and-plugins/developing-packages. Acesso em: 07 de setembro de 2023.

GEEKHUNTER. *Firebase:* o que é e quando usar no desenvolvimento mobile? 2021. Disponível em: https://blog.geekhunter.com.br/firebase-o-que-e-e-quando-usar-no-desenvolvimento-mobile/. Acesso em: 07 de setembro de 2023.

INTELLIJ IDEA. *IntelliJ IDEA – the Leading Java and Kotlin IDE*, 2023. Disponível em: https://www.jetbrains.com/idea/. Acesso em: 16 de setembro de 2023.

JOURNEY NORTH. Understanding Latitude and Longitude. Disponível em: https://journeynorth.org/tm/LongitudeIntro.html. Acesso em: 15 de setembro de 2023.

KHAN, S*. App Lifecycle In Flutter*, out. 2021. Disponível em: https://medium.flutterdevs.com/app-lifecycle-in-flutter-c248d894b830. Acesso em: 07 de setembro de 2023.

MALING, D. F. Coordinate Systems and Map Projections. 2nd ed. Oxford: Pergamon, 1992.

MARCORATTI, J. C. *Flutter - Exibindo SnackBars*, 2019. Disponível em: https://www.macoratti.net/19/06/flut_widgt1.htm. Acesso em: 1 de setembro de 2023.

PARKINSON, BW; SPILKER, JJ (Eds.). *Global Positioning System: Theory and Applications, Vol. I, II*. Vol. 163. American Institute of Aeronautics and Astronautics, 1996.

PINHEIRO, F*. Flutter: O que são widgets e qual sua importância*, 2020. Disponível em: https://www.treinaweb.com.br/blog/flutter-o-que-sao-widgets-e-qual-sua importância. Acesso em: 19 de setembro de 2023.

PUB.DEV. *Flutter_icons* 1.1.0. Jul, 2023. Disponível em: https://pub.dev/packages/flutter_icons. Acesso em: 07 de setembro de 2023a.

_____.*Photo_view* 0.14.0. Jun, 2023. Disponível em: https://pub.dev/packages/photo_view. Acesso em: 07 de setembro de 2023b.

_____.*Google_maps_flutter* 2.5.0 . set, 2023. Disponível em: https://pub.dev/packages/google_maps_flutter. Acesso em: 07 de setembro de 2023c.

_____..*Get_storage 2.1.1*. Jul, 2023. Disponível em: https://pub.dev/packages/get_storage. Acesso em: 07 de setembro de 2023d.

_____.*Restart_app 1.2.1*. Jul, 2023. Disponível em: https://pub.dev/packages/restart_app. Acesso em: 07 de setembro de 2023e.

_____intl 0.18.1 mai, 2023. Disponível em: https://pub.dev/packages/intl. Acesso em: 07 de setembro de 2023f.

_____.firebase_core 2.16.0. set, 2023. Disponível em: https://pub.dev/packages/firebase_core. Acesso em: 07 de setembro de 2023g.

_____cloud_firestore 4.9.2, set, 2023. Disponível em: https://pub.dev/packages/cloud_firestore. Acesso em: 07 de setembro de 2023h.

_____.uuid 4.1.0 set, 2023. Disponível em: https://pub.dev/packages/uuid. Acesso em: 07 de setembro de 2023i.

_____.Geolocator 10.1.0. Jul, 2023. Disponível em:https://pub.dev/packages/geolocator. Acesso em: 07 de setembro de 2023j.

_____..Permission_handler 11.0.0. Jul, 2023. Disponível em: https://pub.dev/packages/permission_handler. Acesso em: 07 de setembro de 2023l.

_____. Get 4.6.6. Jul, 2023. Disponível em: https://pub.dev/packages/get. Acesso em: 07 de setembro de 2023m.

_____,internet_connection_checker 1.0.0+1 2022. Disponível em: https://pub.dev/packages/internet_connection_checker. Acesso em: 07 de setembro de 2023n.

_____,maps_toolkit 3.0.0, set. 2023. Disponível em: https://pub.dev/packages/maps_toolkit. Acesso em: 07 de setembro de 2023o.

SILVA, Gizele. O que é API? Coodesh, 2023. Disponível em: https://coodesh.com/blog/dicionario/o-que-e-api/. Acesso em: 13 de setembro de 2023.

TREINA WEB. *O que é Firebase?*, 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-firebase. Acesso em: 25 de setembro de 2023.

WILHELM, V. E. *Problema do Caminho Mínimo,* s.d. Departamento de Engenharia de Produção – UFPR. Disponível em: https://docs.ufpr.br/~volmir/PO_II_10_caminho_minimo.pdf. Acesso em: 25 de setembro de 2023.

XU, G. GPS: Theory, Algorithms and Applications. 2. ed. Berlin: Springer, 2007.

Apêndice A

Instalação de *packages*

Packages utilizados no desenvolvimento da aplicação:

- flutter_*icons*: conjunto de ícones personalizáveis para *Flutter*. Esse *package* permite utilizar uma ampla biblioteca de ícones que serão utilizados como *widgets* na interface do aplicativo.
- *get*: *package* do *GetX* que combina gerenciamento de estado de alto desempenho, injeção de dependência inteligente e gerenciamento de rotas de forma rápida e prática. Esse *package* será responsável pelo gerenciamento de estado do aplicativo de modo geral.
- *google_maps_flutter*: *package* que permite utilizar a API do *Google Maps* em um projeto *Flutter*. Neste trabalho será utilizado para o mapeamento e gerenciamento de rotas para as localizações desejadas dentro da universidade.
- *get_storage*: é um *package* em que é possível criar um valor-chave rápido, extra leve e síncrono na memória, que faz *backup* dos dados no disco a cada operação. Através dele será possível armazenar localmente qual idioma será escolhido pelo usuário para ser utilizado na aplicação.
- *restart_*app: é um plug-in *Flutter* que reinicia todo o aplicativo *Flutter* com uma única chamada de função. Será utilizado para reiniciar o aplicativo toda vez que o idioma for alterado a fim de renderizar a mudança de texto em todo o sistema.
- *intl*: fornece recursos de internacionalização e localização, incluindo tradução de mensagens, plurais e gêneros, formatação e análise de data/número e texto bidirecional. Utilizada para a formatação de datas e tradução de dias da semana conforme o idioma escolhido par ao aplicativo.
- *firebase_core*: *plug*-in do *Flutter* para usar a *Firebase Core* API, que permite a conexão com vários aplicativos do *Firebase*.
- *cloud_firestore*: *plug*-in para utilizar o *Cloud Firestore*. Este *package* permitirá utilizar o banco de dados *Flutter* com todas as informações pertinentes a ônibus, ensalamento, eventos, lugares e mapeamento.

- *uuid*: *package* para geração simples e rápida *de UUIDs (Universally Unique IDentifier)*. Será utilizada para gerar identificadores únicos para cada objeto adicionado ao *Firebase.*
- *geolocator*: um *plug-in* de geolocalização do *Flutter* que fornece acesso fácil a serviços de localização específicos da plataforma. Será utilizado para obter a localização exata do usuário.
- *permission_handler*: este plug-in fornece uma API para solicitar permissões e verificar seu status. Através deste *package* será possível solicitar a permissão do usuário para ter acesso a sua localização exata.
- *photo_view*: o *PhotoView* permite que as imagens sejam ampliadas com gestos do usuário, como beliscar, girar e arrastar. Através desse *package* será possível mostrar imagens de forma dinâmica na aplicação.

Para instalar um *package* no desenvolvimento de um aplicativo Flutter, é necessário seguir os seguintes passos:

1. Abrir o arquivo pubspec.yaml no diretório raiz do projeto Flutter;
2. Na seção dependencies, adicionar o nome do pacote que se deseja instalar e a versão desejada. Por exemplo:

```
dependencies:
  flutter:
    sdk: flutter
  nome_do_pacote: ^versao_desejada
```

3. Substituir o "nome_do_pacote" pelo nome do pacote que se deseja instalar e "versao_desejada" pela versão específica ou por um intervalo de versões, como ^1.0.0 para obter a versão mais recente compatível;
4. Salve o arquivo pubspec.yaml;
5. No terminal, dentro do diretório do projeto Flutter, executar o comando flutter pub get. Isso baixará e instalará o pacote especificado no arquivo pubspec.yaml.

Após seguir esses passos, o pacote estará instalado no projeto Flutter e será possível importá-lo nos arquivos de código para usá-lo.

Apêndice B

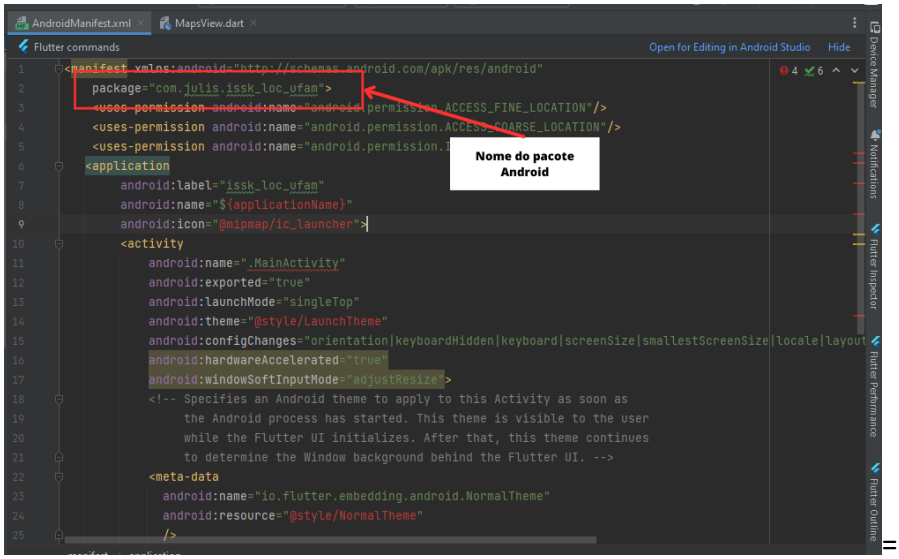
Instalação de *Firebase*

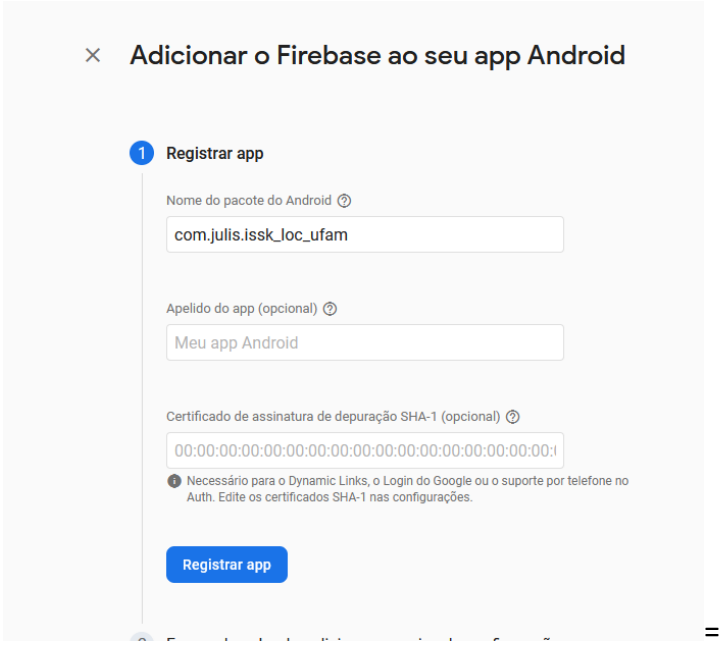
Com os *packages* do *Firebase* já adicionados ao código da aplicação é necessário então criar um projeto no Console Firebase. Para configura-lo basta seguir as seguintes etapas:

1. Criar e dar um nome ao projeto;

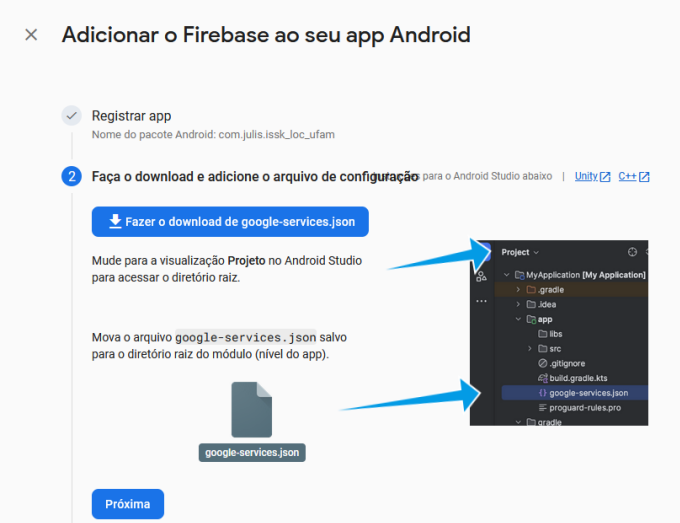
2. Escolher a configuração para a plataforma *Android*;

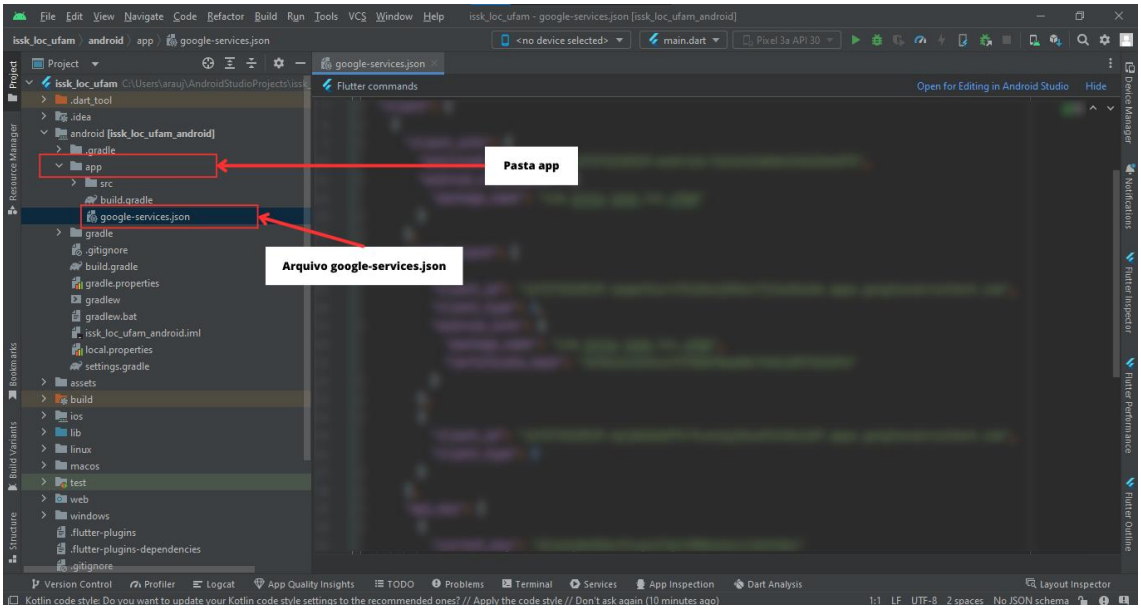
3. Encontrar o nome de pacote *Android* no arquivo AndroidManifest.xm e registrar o aplicativo no Firebase utilizando esse nome;





4. Realizar o download do arquivo *google-service.json* e adiciona-lo na pasta *android/app* do diretório do projeto;





5. Adicionar os *plug-ins* de serviços do *Google* e as dependências do *Firebase* ao arquivo *build.gradle* na pasta *android/app* do diretório do projeto;

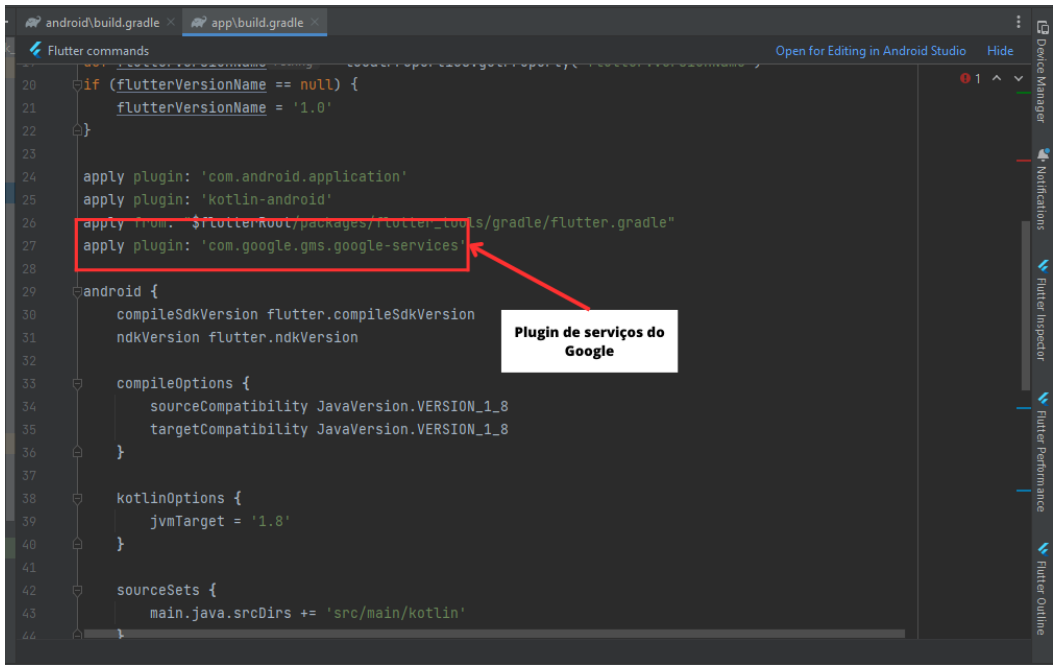
2. Em seguida, no arquivo `build.gradle` **do módulo (nível do app)**, adicione o plug-in `google-services` e todos os SDKs do Firebase que você quer usar no aplicativo:

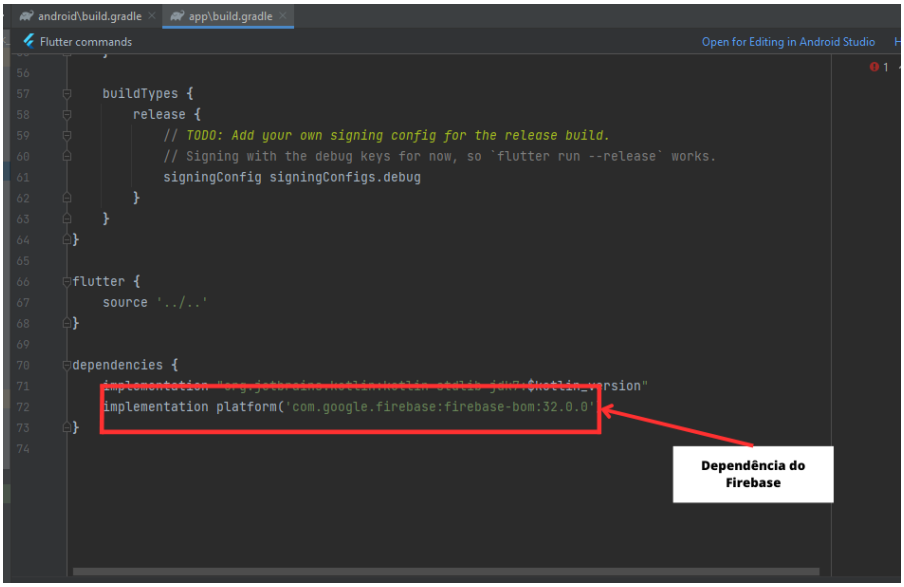
(•) Kotlin ( ) Java

Arquivo do Gradle do módulo (nível do app) (`<project>/<app-module>/build.gradle`):

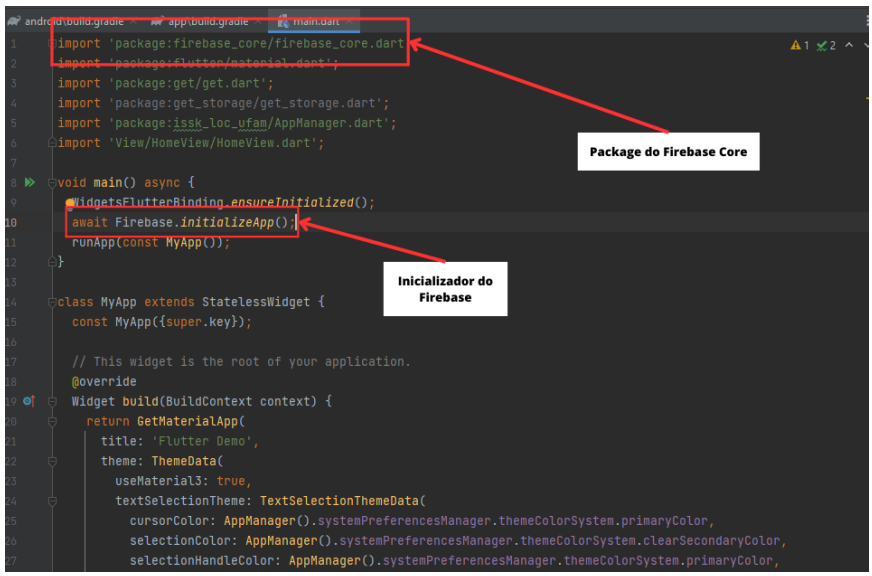
```
plugins {
  id 'com.android.application'
  // Add the Google services Gradle plugin
  id 'com.google.gms.google-services'
  ...
}

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:32.2.2')

  // TODO: Add the dependencies for Firebase products you want to use
  // When using the BoM, don't specify versions in Firebase dependencies
  implementation 'com.google.firebase:firebase-analytics-ktx'

  // Add the dependencies for any other desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

Ao usar a BoM do Firebase para Android, o app sempre vai utilizar versões da biblioteca do Firebase compatíveis.
Saiba mais





6. Importar o *package firebase_core* no arquivo *main.dart* e inicializar Firebase no método *main*() antes de *runApp*();

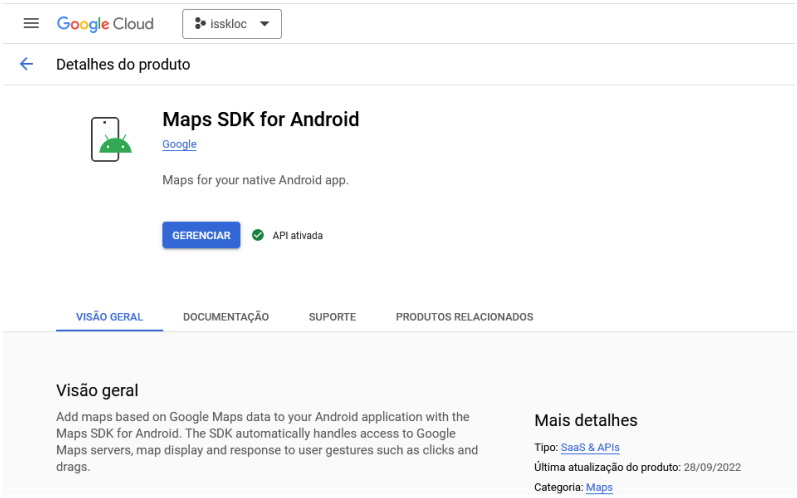


Apêndice C

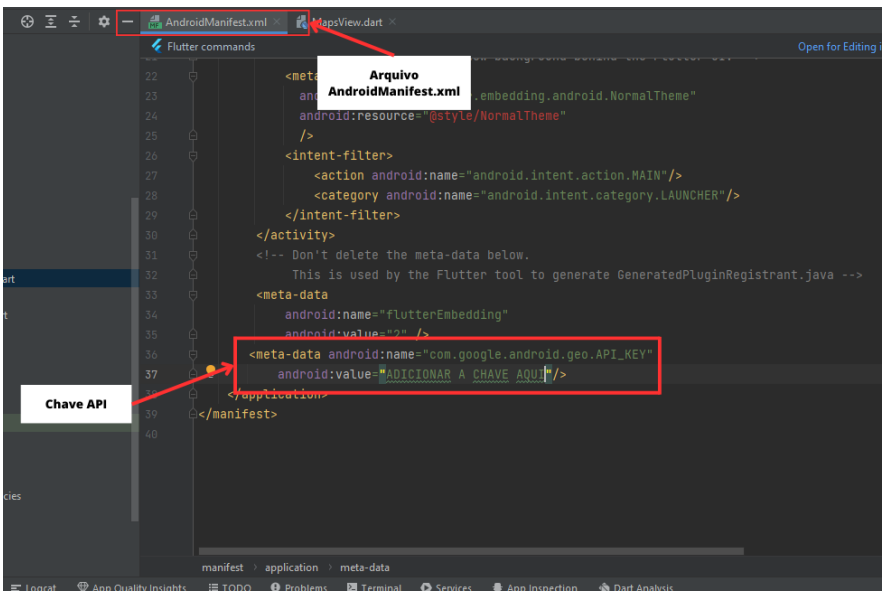
Instalação da API do Google Maps

Com o *package* do Google Maps instalado, para adicionar a chave API ao projeto é necessário seguir as seguintes etapas:

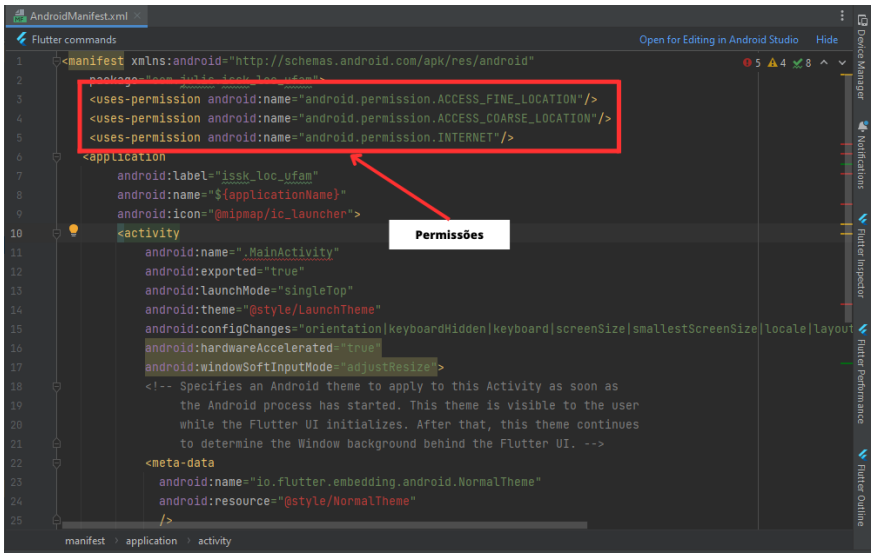
1. Criar uma chave API individualizada de acesso aos servidores da Google. Para obter a chave é necessário criar uma conta e cadastrar um cartão de crédito no site *Google Cloud*



2. Adicionar a chave ao arquivo AndroidManifest.xml;



3. Adicionar no arquivo AndroidManifest.xml as seguintes permissões: *android.permission.ACCESS_FINE_LOCATION* (permite que um aplicativo acesse a localização precisa), *android.permission.ACCESS_COARSE*_LOCATION (permite que um aplicativo acesse a localização aproximada) e *android.permission.INTERNET* (permite que um aplicativo abra conexões de rede);



4. Adicionar o *widget* do mapa no código do aplicativo;