Politécnico
Castelo Branco
Escola Superior
de Tecnologia

Aplicação móvel para ajuda ao estacionamento de pessoas com mobilidade reduzida
Projeto I & II

Dinis Sokolan Levchenko

Pedro Manuel Matos Mendes

Orientador

Osvaldo Arede dos Santos

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciado em Engenharia Informática, realizado sob a orientação científica do Doutor Osvaldo Arede dos Santos, do Instituto Politécnico de Castelo Branco.

Setembro de 2025

Composição do júri

Presidente do júri

Doutora Ana Paula Neves Ferreira da Silva

Professora Adjunta do Instituto Politécnico de Castelo Branco

Vogais

Doutor José Carlos Meireles Monteiro Metrolho

Professor Coordenador do Instituto Politécnico de Castelo Branco

Doutor Osvaldo Arede dos Santos

Professor Adjunto do Instituto Politécnico de Castelo Branco

Resumo

A acessibilidade a lugares de estacionamento para pessoas com mobilidade reduzida é um fator crítico para a sua autonomia e qualidade de vida. No entanto, a escassez destes lugares, aliada à falta de informação em tempo real sobre a sua localização e disponibilidade, torna a tarefa de estacionar particularmente desafiadora e frustrante. Este trabalho apresenta o processo de desenvolvimento da aplicação móvel ParkFinder, que visa monitorizar em tempo real a ocupação de lugares de estacionamento reservados para pessoas com mobilidade reduzida, combinando tecnologias de visão computacional e uma interface intuitiva.

O relatório inicia com uma contextualização do problema e a sua relevância social, seguida de uma análise comparativa de soluções existentes no mercado, destacando as suas limitações e como o ParkFinder as supera. São detalhadas as tecnologias adotadas incluindo Mapbox para mapas e navegação, Firestore para a base de dados em tempo real, Kotlin e Jetpack Compose para o desenvolvimento Android, e o modelo YOLO para deteção de veículos bem como a justificação para as suas escolhas.

Foram desenvolvidos e evoluídos diagramas de casos de uso que refletem as interações entre o utilizador e o sistema, esboços de UI/UX que orientaram o design da interface, e um modelo entidade-relacionamento que estrutura a base de dados. A aplicação inclui funcionalidades como pesquisa de destinos, cálculo e acompanhamento de rotas com instruções por voz e visuais, reencaminhamento automático em caso de ocupação do lugar ao qual o utilizador se deslocava, e sincronização offline.

Os testes realizados validaram o correto funcionamento da aplicação em diversos cenários, com exceção da integração com Android Auto, que se revelou inviável devido a limitações técnicas. A solução desenvolvida demonstra potencial para melhorar significativamente a eficiência e autonomia dos utilizadores, reduzindo o tempo dedicado à procura de estacionamento e oferecendo uma experiência de utilização fluida e acessível.

Palavras chave

Mobilidade reduzida, Monitorização, Aplicação móvel, Estacionamento.

Abstract

Accessibility to parking spaces for people with reduced mobility is a critical factor for their autonomy and quality of life. However, the scarcity of such spaces, combined with the lack of real-time information regarding their location and availability, makes the task of parking particularly challenging and frustrating. This work presents the development process of the ParkFinder mobile application, which aims to monitor in real time the occupancy of parking spaces reserved for people with reduced mobility by combining computer vision technologies with an intuitive interface.

The report begins with a contextualization of the problem and its social relevance, followed by a comparative analysis of existing market solutions, highlighting their limitations and how ParkFinder overcomes them. The adopted technologies are detailed, including Mapbox for maps and navigation, Firestore for real-time database management, Kotlin and Jetpack Compose for Android development, and the YOLO model for vehicle detection, along with the rationale for their selection.

Use case diagrams reflecting user–system interactions, UI/UX sketches guiding interface design, and an entity-relationship model structuring the database were developed and refined. The application includes features such as destination search, route calculation and guidance with voice and visual instructions, automatic rerouting when a targeted parking space is occupied, and offline synchronization.

Testing validated the correct operation of the application in various scenarios, with the exception of Android Auto integration, which proved unfeasible due to technical limitations. The developed solution demonstrates potential to significantly improve user efficiency and autonomy, reducing the time spent searching for parking and providing a smooth and accessible user experience.

Keywords

Reduced mobility, Monitoring, Mobile application, Parking space

Índice geral

Índice de figuras

Lista de tabelas

Lista de abreviaturas, siglas e acrónimos

API (*Application Programming Interface*)

ER (*Entity-Relationship*)

GFLOP (*Giga Floating Point Operations per Second*)

GPS (*Global Positioning System*)

HTTP (*HyperText Transfer Protocol*)

HTTPS (*HyperText Transfer Protocol Secure*)

IDE (*Integrated Development Environment*)

IoU (*Intersection over Union*)

JS (*JavaScript*)

JSON (*JavaScript Object Notation*)

mAP (*mean Average Precision*)

ms (*Milliseconds*)

MVC (*Model-View-Controller*)

MVP (*Model-View-Presenter*)

SGBD (*Database Management Systems*)

SKU (*Stock Keeping Unit*)

SQL (*Structured Query Language*)

UI (*User Interface*)

UX (*User Experience*)

XML (*Extensible Markup Language*)

YOLO (*You Only Look Once*)

1. Introdução

Neste capítulo vamos abordar alguns temas como o contexto do projeto, apresentando o problema identificado e a relevância da sua resolução.

Inicialmente irá ser descrito o âmbito e a definição do problema onde descrevemos um pouco sobre as dificuldades que pessoas com mobilidade reduzida enfrentam no seu dia-a-dia. De seguida, são apresentados os objetivos, diferenciando as metas do Projeto I e as do Projeto II e por fim, é abordada a organização do documento, proporcionando uma visão clara sobre a estrutura dos capítulos que compõem este relatório.

1.1. Âmbito e definição do problema

As pessoas com mobilidade reduzida enfrentam diariamente grandes desafios para encontrar lugares de estacionamento reservados. O número reduzido destes lugares, aliado à elevada taxa de ocupação e a falta de informação acessível sobre a localização e disponibilidade dos mesmos, agrava ainda mais a situação, tornando esta tarefa difícil e frustrante, comprometendo a autonomia e a mobilidade destas pessoas.

Com o intuito de resolver este problema, desenvolvemos uma aplicação móvel inovadora que disponibiliza, em tempo real, a localização dos lugares de estacionamento, bem como o estado da sua ocupação. Para garantir a precisão da informação, o sistema utiliza câmaras integradas com um sistema de visão computacional, capaz de identificar se os lugares estão ocupados ou disponíveis.

Durante a nossa pesquisa, analisámos as soluções existentes no mercado e verificámos que existem muito poucas aplicações que procuram resolver este problema. As que existem apresentam, na sua maioria, um visual ultrapassado e pouco intuitivo, ou são específicas para determinadas regiões, como é o caso de algumas aplicações desenvolvidas exclusivamente para cidades como Madrid (Madrid PMR 360). Um exemplo de aplicação atualmente disponível é a Park4Dis [1], que ajuda a localizar lugares de estacionamento acessíveis. No entanto, esta plataforma depende principalmente de bases de dados públicas e da colaboração dos utilizadores para manter o sistema atualizado, não informando se o lugar está ou não ocupado.

O nosso projeto diferencia-se, acima de tudo, por oferecer um sistema de monitorização em tempo real, que garante dados mais precisos e atualizados sobre a ocupação dos lugares. Outro ponto de destaque da nossa aplicação é a sua simplicidade e acessibilidade. Diferente do Park4Dis, que tem uma interface ultrapassada esta ainda exige que o utilizador seja obrigado a criar uma conta para aceder ao serviço. A nossa aplicação não requer registo nenhum, permitindo que qualquer pessoa aceda rapidamente às informações sem ter que perder tempo [1].

Com esta solução, pretendemos melhorar significativamente a qualidade de vida das pessoas com mobilidade reduzida, proporcionando-lhes mais autonomia e eficiência na procura por estacionamento acessível, reduzindo o tempo perdido nessa tarefa e tornando a experiência mais fluida e intuitiva.

1.2. Objetivos

No âmbito da unidade curricular de Projeto I, pretendeu-se dar início ao desenvolvimento de uma solução tecnológica capaz de responder às dificuldades enfrentadas por pessoas com mobilidade reduzida na procura de lugares de estacionamento reservados. Para tal, foram definidos os seguintes objetivos principais:

- Pesquisa e análise comparativa de aplicações semelhantes já existentes;
- Definição da arquitetura da solução e seleção das APIs e tecnologias mais adequadas;
- Estudo e escolha do algoritmo de visão computacional a aplicar;
- Desenvolvimento de diagramas e esboços de interface da aplicação.
- Elaboração e implementação da base de dados;
- Criação de um mapa com os lugares identificados.

No âmbito da unidade curricular de Projeto II, o objetivo principal foi transformar a solução conceptual desenvolvida no semestre anterior numa aplicação móvel funcional e completa, capaz de integrar todas as funcionalidades planeadas e disponibilizar informações em tempo real. Para alcançar este propósito, foram delineados os seguintes objetivos:

- Treino do modelo de visão computacional com recurso a fotos e vídeos capturadas por um drone;
- Desenvolvimento do código de integração com as câmaras, para deteção automática da ocupação e envio da informação para a base de dados;
- Atualização em tempo real da disponibilidade dos lugares;
- Criação de rotas até ao lugar selecionado, com auxílio de setas, voz e barra de pesquisa;
- Implementação completa da aplicação móvel com integração das funcionalidades;
- Testes funcionais e validação da aplicação.

1.3. Planeamento do projeto

Para cumprir as metas definidas para o Projeto I, as atividades foram organizadas em seis tarefas principais ao longo do 1.º semestre:

- Tarefa A: **Pesquisa e análise comparativa**
  - Investigação e avaliação de soluções já existentes, visando detetar limitações e oportunidades de melhoria.
- Tarefa B: **Definição da arquitetura e seleção de tecnologias**
  - Escolha das ferramentas, APIs e linguagens de programação mais adequadas ao sistema.
- Tarefa C: **Elaboração do relatório**
  - Redação inicial do documento, com ênfase no enquadramento teórico e na contextualização do problema.
- Tarefa D: **Diagramas e esboços da aplicação**
  - Elaboração de diagramas de casos de uso, modelo entidade-relacional e esboços da interface.
- Tarefa E: **Desenvolvimento da base de dados**
  - Modelação e implementação da estrutura de dados necessária ao funcionamento da aplicação.
- Tarefa F: **Construção do mapa**
  - Desenvolver um mapa com o registo dos lugares de estacionamento reservados em Castelo Branco.

Tabela 1 - Cronograma Projeto I.

	2024				2025								
Tarefas	setembro	outubro	novembro	dezembro	janeiro	fevereiro	março	abril	maio	junho	julho	agosto	setembro
A													
B													
C													
D													
E													
F													

No Projeto II, a realização dos objetivos propostos para o 2.º semestre foi estruturada nas tarefas seguintes:

- Tarefa G: **Continuação do relatório**

    - Atualização e expansão do documento, incorporando os avanços de desenvolvimento e os resultados obtidos.
- Tarefa H: **Treino do modelo de visão computacional**
    - Constituição de um dataset com imagens e vídeos captados por drone e treino do modelo.
- Tarefa I: **Integração com câmaras**
    - Implementação do módulo de deteção da ocupação das vagas e envio dos dados em tempo real para a base de dados.
- Tarefa J: **Desenvolvimento da aplicação móvel**
    - Implementação completa da aplicação Android, incluindo funcionalidades de navegação com rotas, setas, orientação por voz e barra de pesquisa.
- Tarefa K: **Testes funcionais**
    - Execução de testes em cenários diversificados para validar a robustez e a fiabilidade da aplicação.

Tabela 2 - Cronograma Projeto II.

	2025							
Tarefas	fevereiro	março	abril	maio	junho	julho	agosto	setembro
G								
H								
I								
J								
K								

1.4. Estrutura do relatório

Este relatório encontra-se organizado em 9 capítulos, cada um abordando aspetos específicos do desenvolvimento da aplicação ParkFinder. A estrutura adotada segue uma progressão lógica que acompanha todo o ciclo de desenvolvimento do projeto, desde a identificação do problema até à implementação e validação da solução.

Capítulo 1 - Introdução: Identifica-se o problema que nos dispusemos a resolver e apresenta-se a sua possível solução. É também apresentada uma introdução ao trabalho, assim como o enquadramento e as principais tarefas delineadas para ambas as fases relativas ao Projeto I e II. Define-se o âmbito do projeto, os objetivos específicos para cada semestre e o planeamento das atividades necessárias para alcançar as metas propostas.

Capítulo 2 - Arquitetura da Solução: Detalha-se a arquitetura técnica da aplicação, justificando a escolha do padrão arquitetural adotado e apresentando o modelo de dados implementado. São descritas as principais tecnologias utilizadas (Mapbox, Firebase, Android Studio/Kotlin, GitHub e YOLO) e explica-se cada componente.

Capítulo 3 - Modelação da Solução: Apresenta-se a modelação conceptual da aplicação através do diagrama de casos de uso. São descritos os esboços de UI/UX que definem a experiência visual e funcional da aplicação, e expõe-se o modelo de Entidade Relacional.

Capítulo 4 - Primeira Implementação: Documenta-se a versão inicial da aplicação, destacando as funcionalidades básicas implementadas.

Capítulo 5 - Modelo de Computação Visual: Descreve-se detalhadamente o sistema de visão computacional implementado, incluindo a seleção e preparação do dataset, a comparação entre as arquiteturas YOLO testadas (YOLO11x e YOLOv8s), os métodos de treino utilizados e a análise dos resultados obtidos, evidenciando as decisões técnicas para a implementação final.

Capítulo 6 - Desenvolvimento da Aplicação Móvel: Detalha-se a implementação prática do ParkFinder, apresentando os requisitos funcionais e as permissões necessárias. Descrevem-se as tecnologias utilizadas (Kotlin e Jetpack Compose), a estrutura da aplicação e as interfaces principais, evidenciando o fluxo de interação e as decisões de design adotadas.

Capítulo 7 - Integrações: Explica-se como os diferentes componentes do sistema se interligam, focando nas duas integrações principais: a base de dados (Firebase/Firestore) e o modelo de computação visual. Demonstra-se como estas integrações garantem o funcionamento em tempo real e a sincronização entre todos os componentes.

Capítulo 8 - Testes: Apresenta-se a metodologia de teste utilizada, os cenários testados e os resultados obtidos. Aborda-se tanto os testes funcionais como os

testes de usabilidade, validando o correto funcionamento das funcionalidades e a experiência do utilizador.

Capítulo 9 - Conclusão: Apresentam-se os principais resultados alcançados com o desenvolvimento da aplicação, evidenciando o cumprimento dos objetivos definidos. Resume-se a evolução do projeto, desde a conceção e planeamento até à implementação final e validação de testes, destacando a integração entre visão computacional, base de dados em tempo real e aplicação móvel. São também discutidos os desafios enfrentados, as limitações identificadas e perspetivas futuras.

Esta estrutura permite uma compreensão progressiva e completa do projeto, desde a sua conceção teórica até à implementação prática, fornecendo ao leitor uma visão abrangente de todo o processo de desenvolvimento da aplicação. Este relatório resulta da junção do trabalho desenvolvido no âmbito do Projeto I e Projeto II, refletindo assim a evolução natural do projeto desde a fase inicial de planeamento e conceção até à sua implementação final e validação.

A primeira parte do relatório (capítulos 1 a 4) corresponde ao desenvolvimento realizado no Projeto I, enquanto a segunda parte (capítulos 5 a 10) representa as etapas e avanços alcançados durante o Projeto II.

2. Arquitetura da solução

Neste capítulo é detalhada a arquitetura técnica da aplicação: justifica a escolha do padrão *Model View Contoller (*MVC*)*, descreve o modelo de dados, explica o design da API e apresenta as principais tecnologias usadas (Mapbox, Firebase, Android Studio/Kotlin, GitHub e YOLO), mostrando como cada componente se integra para garantir escalabilidade, desempenho e manutenção.

2.1. Âmbito e definição do problema

Na fase do planeamento, duas arquiteturas sobressaíram como opções eficazes na escolha do padrão de arquitetura deste projeto: MVC e MVP.

O padrão MVC representado na **Figura 1** consiste na divisão da aplicação em três camadas:

- ***Model*:** A Camada *model* é responsável pela gestão dos dados de leitura e escrita da aplicação através de regras, lógica e funções;
- ***View*:** A camada *view* tem como função apresentar os dados ao utilizador;
- ***Controller*:** A camada *controller* é responsável por controlar toda a informação que passa pela aplicação e por coordenar a comunicação entre Model e View [4][5].

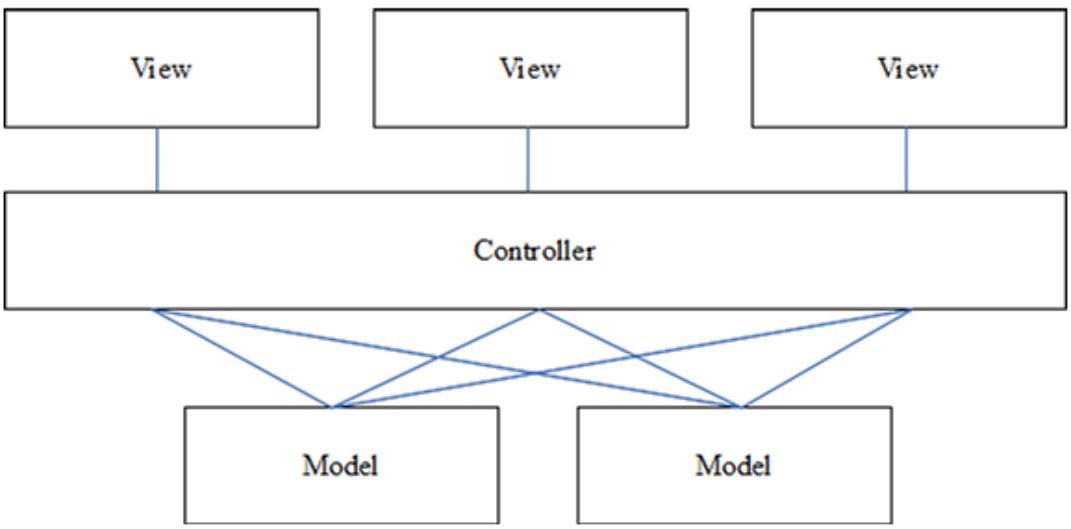

***Figura 1** - Representação da arquitetura do padrão MVC.*

O padrão *Model View Presenter* (MVP) é derivado da arquitetura MVC, considerado por muitos uma melhoria. No padrão MVP representado na **Figura 2**, a aplicação é dividida em quatro camadas:

- ***Model***: A camada *model* é responsável pela gestão dos dados de leitura e escrita da aplicação através de regras, lógica e funções;
- ***View*:** A camada *view* tem como função apresentar os dados ao utilizador;
- **Presenter (Interface):** Camada que define os eventos e métodos que o *presenter* pode executar. Serve para separar a camada *view* da *presenter*;
- **Presenter:** Esta camada atua como uma ponte entre as camadas *view* e *model*. Recebe os eventos da interface (*view*), processa e comunica com a camada *model* que atualiza a *view* com os resultados. Toda a lógica da apresentação encontra-se no *presenter*, o que vai tornar a camada *view* mais simples e separada [6][7][8].

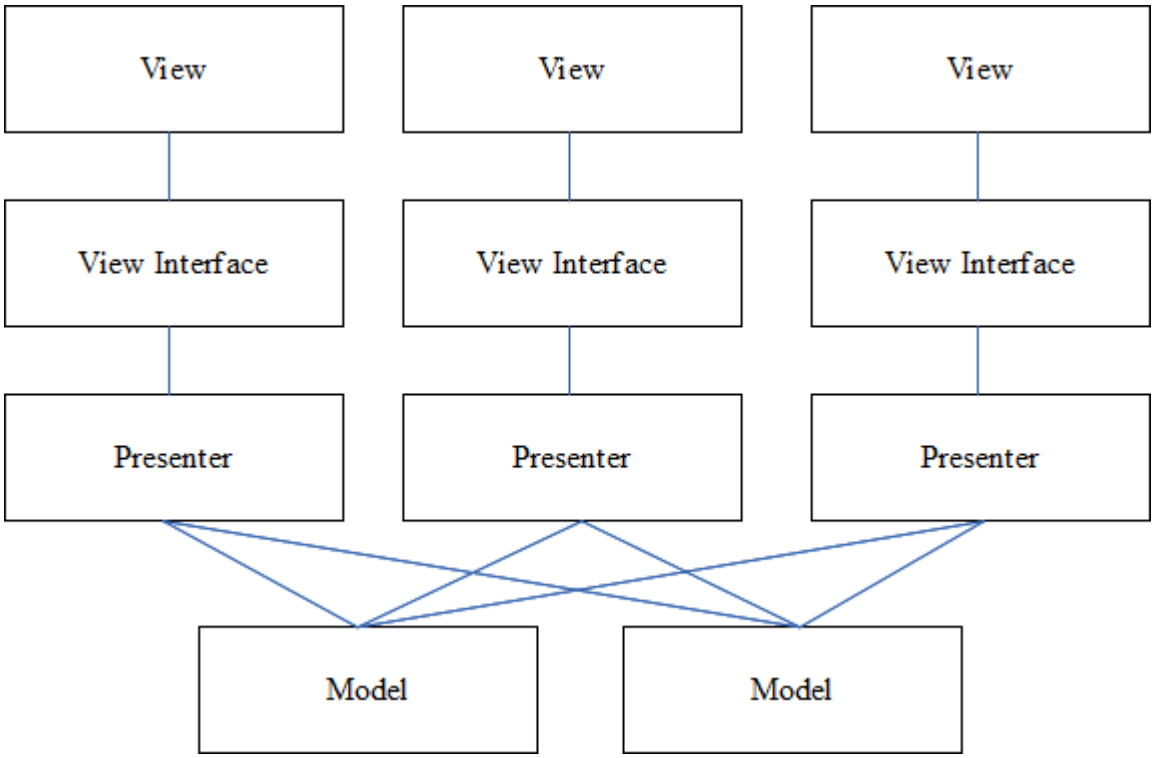

Figura 2 *- Representação da arquitetura do padrão MVP.*

Na prática, ambas não apresentam muitas diferenças. Ambos os padrões ajudam a separar as responsabilidades, mantendo a camada *view* separada da lógica da aplicação, a camada *model*. No MVC, o *controller* pode ser partilhado entre várias *views*. Já no MVP, cada *view* tem o seu próprio *presenter*.

A arquitetura MVP tende a ser mais difícil de implementar, mas é mais adequada para aplicações maiores, pois oferece uma separação mais rigorosa das responsabilidades, o que facilita a realização de testes e alterações. Já o MVC é mais simples, funciona bem para projetos pequenos, facilita a reutilização de código dos *controllers* e mantém uma boa separação das responsabilidades, o que contribui para a manutenção [2][3].

Por isso, para este projeto em específico, o MVC acaba por ser a melhor escolha com as seguintes responsabilidades para cada camada:

- ***Model*:** Representado pelo *Firestore*;
- ***View*:** A aplicação com o *MapBox*;
- ***Controller*:** Scripts que ligam o YOLO ao *Firestore* e a lógica da app Android que administra as atualizações e a *interface.*

Juntando todas as tecnologias, esta é a arquitetura global do sistema. O YOLO é integrado numa câmara posicionada com visão para o lugar de estacionamento onde deteta o seu estado em tempo real, ao observar uma mudança no estado do mesmo atualiza automaticamente a base de dados refletindo-se de imediato na aplicação. Utilizando assim uma arquitetura baseada no padrão MVC (**Figura 3**).

Figura 3 *- Arquitetura do sistema*

2.2. Modelo de dados

O modelo de dados implementado neste projeto baseia-se numa abordagem orientada a documentos recorrendo ao serviço *Cloud Firestore*.

Toda a informação relevante reside numa única coleção, designada por "vagas". Cada documento dentro desta coleção representa um lugar de estacionamento individual, contendo o seu estado e as suas coordenadas geográficas. Esta escolha

permitiu garantir um modelo de dados de fácil manutenção e alto desempenho, ideal para a primeira fase de implementação.

Estrutura dos documentos da coleção "vagas":

- ID do documento: valor automático;
- Estado (*boolean*): indica se o lugar está ocupado ou livre;
- Latitude (*number*): distância em graus de um ponto em relação à Linha do Equador;
- Longitude (*number*): distância em graus de um ponto em relação ao Meridiano de Greenwich.

2.3. API

Uma analogia para compreender o funcionamento de uma API é a interação entre um cliente e o funcionário de uma loja.

Imagine que um cliente quer informações sobre a disponibilidade de um artigo. Ele faz o pedido ao funcionário, que atua como intermediário. O funcionário consulta o responsável pelo *stock* para verificar a disponibilidade. Após receber a confirmação, transmite a informação ao cliente e, se necessário, procede à entrega do produto.

Esta dinâmica entre cliente, funcionário e o responsável pelo *stock* reflete o comportamento de uma API.

Seguidamente, será apresentada uma explicação mais aprofundada sobre o conceito de API.

O termo API é um acrónimo em inglês para *Application Programming Interface*, que, em português, traduz-se como Interface de Programação de Aplicações. Para compreender melhor o seu significado, é fundamental analisar cada um dos seus componentes:

- A - *Application* (Aplicação): Refere-se a um programa ou sistema que permite aos utilizadores realizar determinadas tarefas. Neste contexto, trata-se de uma aplicação desenvolvida para interagir com outro sistema ou serviço. Esta aplicação pode assumir diversas formas, incluindo software, um *website* ou até um serviço;
- P - *Programming* (Programação): Diz respeito ao conjunto de algoritmos, métodos e regras definidas pelos programadores para estruturar o funcionamento da aplicação. A programação estabelece como a aplicação deve processar e responder às solicitações recebidas;
- I - *Interface*: Representa o mecanismo de interconexão, tanto a nível físico como lógico, entre dois sistemas ou dispositivos independentes, com o propósito de regular e facilitar a troca de informação. A interface

define as regras, formatos e protocolos da comunicação entre as partes envolvidas [9][10][11][12][13].

2.4. Tecnologias a usar

No âmbito deste projeto, foram implementadas diferentes tecnologias, nomeadamente *Web Services* e sistema de base de dados.

2.4.1. Web services

As *Web Services* baseiam-se numa interface que permite a comunicação entre diferentes softwares através da web, geralmente usa padrões como HTTP/HTTPS e formatos de dados como JSON ou XML. Funcionam como um "tradutor", permitindo a ligação e a troca de informações de forma segura e eficiente [14][15].

Neste projeto, utilizou-se a Mapbox Services, que permite o acesso às diversas ferramentas e serviços disponibilizados pelo Mapbox [16].

Foi utilizada para a apresentação de um mapa interativo, com funcionalidades de geolocalização, cálculo de rotas, pesquisa de locais e personalização.

2.4.2. Base de dados

A API de base de dados permite a interação com SGBD, disponibilizando operações fundamentais, como consulta, inserção, atualizações e eliminações de dados. Esta API providencia uma interface que facilita a comunicação com diferentes tipos de bases de dados, incluindo MySQL, Oracle, Microsoft SQL, entre outras.

2.4.3. Escolha da API

Na escolha de um serviço de mapas para integrar em projetos, a API do Google Maps, durante muito tempo, foi a escolha dos desenvolvedores devido à sua ampla oferta de serviços. Contudo surgiram outras APIs com funcionalidades que tornam o desenvolvimento mais fácil e a escolha da API mais ajustada às necessidades específicas de cada projeto.

A escolha da API é uma decisão crucial, uma vez que envolve a consideração de múltiplos fatores que devem cumprir os requisitos do projeto. A seguir, estão alguns dos aspetos que foram considerados durante a escolha da API para o nosso projeto:

- **Custo:** A relação custo-benefício das APIs foi analisada, levando em consideração os planos e preços oferecidos.
- **Funcionalidades:** As funcionalidades oferecidas pela API foram avaliadas, com foco nas capacidades de geolocalização, navegação, personalização de mapas e integração com outros serviços.

- **Compatibilidade e Integração:** A facilidade de integrar com a infraestrutura existente e com outras tecnologias utilizadas no projeto foi um fator decisivo.
- **Licenciamento e Condições de Uso:** As condições de licenciamento e os termos de uso das APIs foram cuidadosamente analisados para garantir a conformidade e a viabilidade a longo prazo.
- **Suporte a mapas offline:** A possibilidade de utilizar mapas offline foi considerada, especialmente para cenários em que a conexão e à Internet pode ser limitada ou inexistente.

Após uma pesquisa aprofundada e comparar diversas opções, incluindo Google Maps API, Mapbox, OpenStreetMap, Bing Maps e Here, a decisão final ficou entre duas APIs: Google Maps API e Mapbox. Ambas as opções apresentaram características vantajosas, mas a escolha final levou em consideração os requisitos específicos e a análise detalhada dos pontos mencionados acima.

2.4.3.1. Google maps API

A Google Maps API é um serviço fornecido pela Google que permite aos desenvolvedores integrar mapas interativos nos seus projetos. Esta API disponibiliza uma ampla variedade de recursos de criação de mapas, incluindo funcionalidades como a sua visualização, pesquisa de lugares, cálculo de rotas e direções, definir o comportamento e personalização [17][18][19].

Funciona por meio de comunicação entre os servidores do Google e a infraestrutura do desenvolvedor, o que não permite a utilização offline de certos recursos.

Principais funcionalidades da API:

- **Mapas interativos:** mapas que permitem ao utilizador interagir, dar zoom, arrastar ou mudar de estilo (rua, satélite, terreno).
- **Geocodificação:** Converte moradas em coordenadas geográficas (latitude/longitude) e o contrário, facilitando pesquisas e localizações precisas.
- **Rotas e direções:** Calcula rotas e dar direções entre dois ou mais pontos, por diferentes meios de transporte, tendo em consideração o trânsito em tempo real.
- ***Street view*:** Oferece uma visualização panorâmica 360º das ruas e locais.
- **Pesquisa por lugares (*Places API*):** Pesquisar estabelecimentos e pontos de interesse próximos, disponibilizando detalhes como o nome, morada, avaliações e horários de funcionamento.
- **Visualização aérea (*Aerial View*):** Criar visualizações imersivas em 3D e até vídeos com imagens aéreas de alta resolução.
- **Geolocalização:** Determinar a localização do dispositivo.

- **Informações adicionais:** Dados sobre a qualidade do ar, índices de pólen, fusos horários, clima e previsão do tempo para locais específicos.

O plano de custo da Google Maps API é baseado em um sistema de pagamento por uso, ou seja, os desenvolvedores pagam consoante o número de chamadas realizadas à API e as funcionalidades utilizadas.

A Google substituiu em março de 2025, o crédito mensal fixo de 182,00 € para uma estrutura de chamadas por SKU, que depende do nível (Essentials, Pro, Enterprise) de cada serviço, com limites gratuitos de 10 000 chamadas por SKU por mês para Essentials, 5000 chamadas para Pro e 1000 chamadas para Enterprise.

Tabela 3 mostra os preços oficiais por 1 000 chamadas após os limites gratuitos (segundo o site da Google Maps Platform) [20].

Tabela 3 - Tabela de preços oficiais por 1 000 chamadas do Google Maps.

API / Serviço	Preço por 1 000 requisições	Nível
Maps (Dynamic Maps)	6,37 €	Essentials
Static Maps	1,82 €	Essentials
Street View Static	6,37 €)	Essentials
Street View Dynamic	12,74 €)	Pro
Routes – Directions / Distance / Geocoding / Timezone	4,55 €	Essentials
Places – Detalhes	15,47 €	Pro
Aerial View	14,56 €	Pro
Solar API Data Layers	68,25 €	Enterprise

2.4.3.2. MapBox API

O MapBox é uma plataforma global que oferece serviços de localização para os desenvolvedores. Desenvolvida para ser flexível e escalável, a plataforma é ideal para pequenas aplicações até grandes projetos empresariais.

Esta tecnologia está estruturada em quatro categorias principais: Mapas, Navegação, Pesquisa e Conta [21].

Serviço de mapas

Este serviço inclui diversas APIs destinadas à criação e personalização dos mapas, nomeadamente:

- ***Styles* API:** Disponibiliza diferentes estilos para mapas, fontes e imagens.
- ***Fonts* API:** Fornece múltiplas tipografias para personalização de mapas.
- ***Static images* API:** Gera imagens estáticas de mapas.
- ***Tilequery* API:** Permite consultar informações de um mapa num ponto específico (coordenadas geográficas).
- ***Raster tiles* API:** Oferece *tiles raster* (blocos de imagens pré-renderizados) para o carregamento e visualização rápido de mapas.

Serviço de navegação

Este serviço oferece APIs que possibilitam funcionalidades de navegação nos mapas, incluindo o cálculo de rotas e tempos de deslocação:

- ***Directions* API:** Calcula e apresenta a melhor rota até ao destino.
- ***Isochrone* API:** Identifica áreas acessíveis a partir de um ponto específico dentro de um determinado intervalo de tempo ou distância.
- ***Matrix* API:** Realiza o cálculo de tempos e distâncias entre múltiplos pontos.

Serviço de pesquisa

Este serviço dispõe de duas APIs para a pesquisa de locais, endereços e coordenadas geográficas:

- ***Search box* API:** Retorna resultados com base na localização introduzida, abrangendo lugares, endereços e pontos de interesse.
- ***Geocoding* API:** Converte endereços em coordenadas geográficas (geocodificação) e vice-versa.

Serviço de conta

O serviço de conta inclui a *Tokens* API, uma chave de autenticação que permite o acesso aos serviços do Mapbox. Cada pedido executado a um serviço requer um *access token*, necessária para a identificação do utilizador, garantindo a segurança no acesso às funcionalidades.

A plataforma estabeleceu parcerias para expandir as suas capacidades, sendo a parceria com a Microsoft Azure uma das mais significativas. Esta colaboração permite integrar os serviços do MapBox à infraestrutura em nuvem do Azure.

Oferece suporte para mapas *offline*, o que é uma das suas grandes vantagens em comparação com outras APIs, como o Google Maps.

O plano de preços do Mapbox funciona com base no uso: os programadores pagam consoante o número de pedidos feitos às APIs e aos serviços.

Existe um limite mensal gratuito bastante generoso, que varia conforme o tipo de API utilizada. Depois de ultrapassar esse limite, o custo é calculado por cada pedido, com valores que mudam consoante o serviço e a quantidade.

A **Tabela 4** mostra os preços oficiais por cada 1 000 pedidos após o limite gratuito, de acordo com o site oficial do Mapbox [22].

Tabela 4 - Tabela de preços oficiais por 1 000 pedidos do MapBox.

API / Serviço	Preço por 1 000 pedidos	Limite gratuito
Directions, Map Matching, Optimization, Matrix, Isochrone APIs	1,82 €	100 000 pedidos
Static Images API	0,91 €	50 000 pedidos
Static Tiles API	0,46 €	200 000 pedidos
Vector Tiles API	0,23 €	200 000 pedidos
Raster Tiles API	0,23 €	750 000 pedidos
Maps SDKs para Mobile	3,64 € por 1 000 utilizadores ativos	25000 utilizadores ativos mensais
Search Box API	0,91 €	50 000 pedidos

A **Tabela 5** mostra um resumo das principais diferenças entre cada API:

Tabela 5 - Principais diferenças entre Google Maps e Mapbox.

Característica	Google Maps API	Mapbox API
Fonte de dados	Base própria do Google, ampla e precisa	Base no OpenStreetMap (OSM)
Personalização	Limitada	Muito alta
Funcionalidades offline	Limitada	Completo suporte offline (web/mobile)
Visualização 3D	Simples, restrita	Avançada
Street View	Sim	Não
Trajeto/trânsito	Informações de trânsito em tempo real	Dados em tempo real variam conforme região
Customização de dados	Limitada ao que o Google permite	Ampla integração de dados próprios
Custo/Escala	Alto	Baixo
Documentação/Suporte	Excelente	Boa

O modelo de preços do Mapbox é uma clara vantagem em comparação ao do Google Maps. Além disso, o suporte a mapas *offline* torna-o especialmente essencial para a nossa aplicação, já que nem sempre podemos contar com acesso à internet.

Após comparar as duas APIs, o Mapbox revela-se a opção mais adequada para responder às necessidades deste projeto.

2.4.4. Base de dados API

Apesar de esta aplicação não ter um número elevado de dados, a escolha da API para o armazenamento e gestão dos dados revela-se fundamental, sobretudo em termos de segurança.

Tendo em conta que estamos a lidar com uma aplicação Android, realizou-se uma pesquisa de uma base de dados que atendesse às necessidades específicas do projeto.

Após uma análise das opções disponíveis, a solução mais adequada encontrada foi a *Firestore.*

A *Firestore* é um serviço oferecido pela *firebase* para o armazenamento de dados na cloud da Google, que permite guardar e sincronizar informações entre dispositivos em tempo real.

A sua estrutura é baseada em documentos e coleções. Cada documento pode abrigar diferentes tipos de dados (como texto, números, valores booleanos, listas ou objetos) e ainda subcoleções.

Principais funcionalidades:

- **Leitura de dados:** Permite o acesso a documentos ou coleções em tempo real, aplicando filtros para extrair apenas as informações relevantes.
- **Adição de dados:** Facilita a criação de novos documentos ou a adição de campos às coleções.
- **Atualização de dados:** Permite alterar apenas determinados campos num documento sem alterar o resto dos dados.
- **Remoção de dados:** Oferece a opção de apagar documentos ou coleções, com mecanismos de proteção contra eliminações irreversíveis.
- **Sincronização instantânea:** Qualquer alteração na base de dados é imediatamente atualizada para todos os dispositivos conectados, sem a necessidade de atualizações manuais.

Além disso, o Firestore disponibiliza um sistema de regras de segurança que permite restringir o acesso aos dados com base em critérios como a autenticação do utilizador, o tipo de operação (leitura ou escrita) ou a organização das informações [23][24].

2.4.5. Android studio

O Android Studio é uma IDE disponibilizada pelo Google, que reúne todos os recursos necessários para criar, desenvolver e testar aplicações Android [25][26].

Prevemos utilizar o Android Studio para desenvolver o código da aplicação em linguagem Kotlin e para simular dispositivos Android através do emulador integrado.

Esta funcionalidade permitirá testar a aplicação sem necessidade de um dispositivo físico, agilizando o processo de desenvolvimento.

Planeamos desenvolver a aplicação tendo como base o Android 16.0 ("Baklava"), que corresponde ao API Level 36.0, garantindo compatibilidade com as funcionalidades da plataforma e com a maior parte dos dispositivos.

2.4.6. GitHub

GitHub é uma plataforma que permite o armazenamento de código e ficheiros, incorporando o controlo de versões através do Git [27][28].

Prevemos utilizar o GitHub como plataforma para organizar e guardar o código e os ficheiros do projeto, aproveitando o controlo de versões que a plataforma oferece. Assim ambos podemos manter-nos atualizados sobre as alterações realizadas, facilitando a gestão das versões e assegurando que haverá sempre uma cópia de segurança disponível, caso surja algum problema com o projeto.

2.4.7. IA para reconhecimento de lugares

Para automatizar o estado dos lugares de estacionamento, será implementada uma solução de visão computacional baseada num modelo treinado para detetar objetos em tempo real. Sempre que o modelo identificar um objeto a entrar ou a sair de uma zona pré-definida no vídeo, será disparado um evento que atualiza automaticamente o estado correspondente na Firestore. Desta forma, a entrada de um veículo numa área de estacionamento disponível altera o campo “ocupado” para *true*, enquanto a saída define-o como *false*, garantindo que a base de dados reflete em tempo real a ocupação de cada lugar.

3. Modelação da solução

Neste capítulo é apresentada a modelação da solução para a aplicação: introduz-se o diagrama de casos de uso para justificar as interações entre o utilizador e o sistema, descrevem-se os esboços de UI/UX que definem a experiência visual e funcional, e expõe-se o modelo Entidade Relacional que estruturará a base de dados.

3.1. Diagrama de casos de uso

O diagrama de casos de uso é utilizado para representar visualmente, de forma simples e intuitiva, as principais interações entre os utilizadores aqui designados por atores e o sistema que está a ser desenvolvido. Este tipo de diagrama é utilizado sobretudo na fase de análise de requisitos, pois proporciona uma melhor compreensão do comportamento funcional da aplicação por parte de todas as equipas envolvidas no desenvolvimento do projeto [29][30][31][32][33].

Neste projeto, o diagrama de casos de uso representado na **Figura 4** foi desenvolvido com o apoio do software "Figma" com o objetivo principal de demonstrar as funcionalidades essenciais da aplicação, bem como as ações que o utilizador pode realizar, contribuindo assim para uma melhor organização e planeamento do desenvolvimento.

O diagrama demonstra que o ator interage com o sistema através de nove casos de uso principais, estando no centro de todos e podendo realizar as seguintes ações:

- **Pesquisar localização:** Funcionalidade base que serve como ponto de partida para o utilizador encontrar localidades do seu agrado. Ao inserir um destino, o sistema apresenta uma lista de sugestões.
- **Selecionar localização:** Permite selecionar uma das sugestões que aparecem quando realizamos uma pesquisa.
- **Selecionar lugar:** O utilizador pode escolher um lugar no mapa para onde deseja navegar.
- **Iniciar rota:** Uma vez selecionado o destino ou um lugar específico, o utilizador pode iniciar a navegação até ao local pretendido.
- **Cancelar rota:** Permite interromper a navegação ativa a qualquer momento.
- **Ligar/Desligar assistente de voz:** Permite a possibilidade de ativar ou desativar o suporte de interação por voz.
- **Recentrar:** Permite recentralizar o mapa na localização atual do utilizador;
- **Expandir/Diminuir indicações de navegação:** Oferece controlo sobre o nível de detalhe das indicações apresentadas durante a navegação;

- **Ligar/Desligar reencaminhamento:** Permite ao utilizador ativar ou desativar o reencaminhamento automático de rotas.

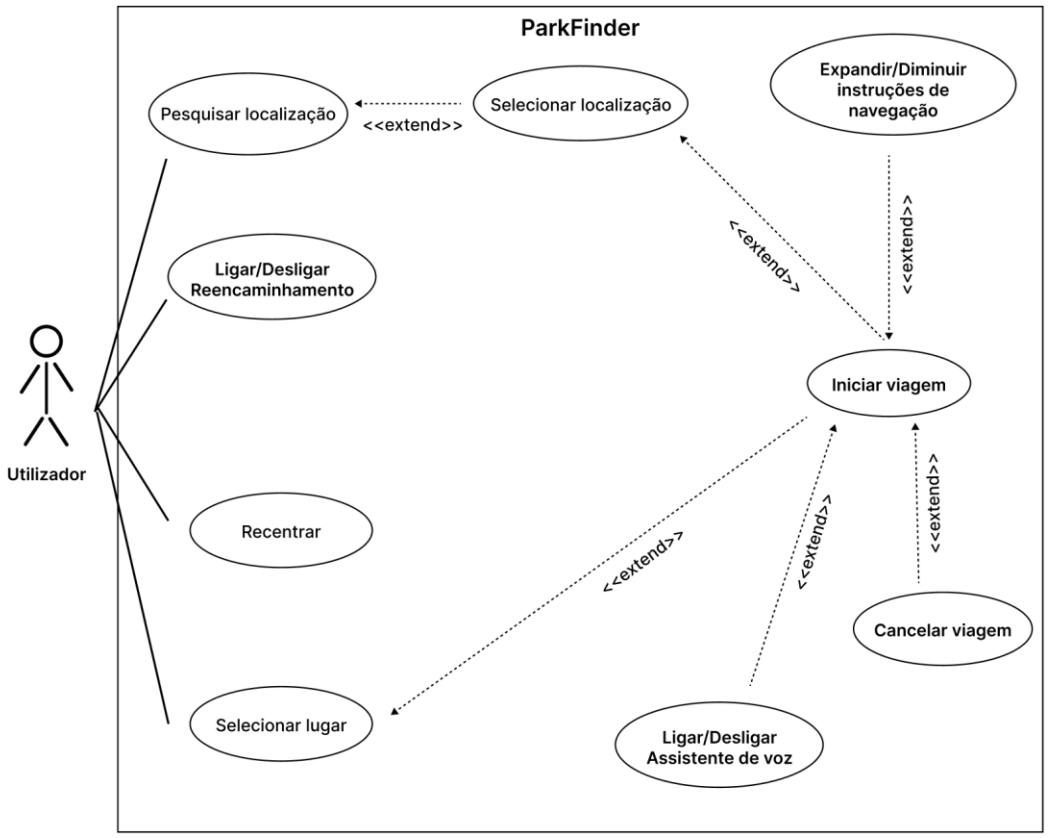

***Figura 4** - Diagrama de Casos de Uso.*

3.2. Esboços de UI/UX

Esta secção apresenta os esboços iniciais desenvolvidos para os principais ecrãs da aplicação ParkFinder, com o objetivo de proporcionar uma experiência intuitiva e eficiente ao utilizador.

3.2.1. Ecrã inicial

O ecrã inicial representado na **Figura 5** foi desenvolvido de forma a oferecer uma interface clara e funcional, permitindo ao utilizador localizar facilmente lugares de estacionamento disponíveis.

As principais funcionalidades incluem:

- **Mapa interativo:** São exibidos os lugares de estacionamento nas proximidades, indicando a sua disponibilidade em tempo real, com estados diferenciados entre "ocupado" e "livre".

- **Barra de pesquisa:** Permite ao utilizador efetuar uma pesquisa dos estacionamentos numa localização específica, facilitando o processo de pesquisa.

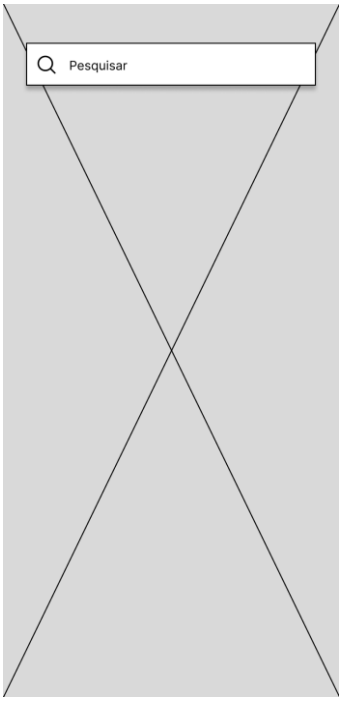

Figura 5 *- Ecrã inicial.*

3.2.2. Seleção de lugares

O ecrã de seleção de lugar permite ao utilizador localizar e selecionar um lugar de estacionamento de forma intuitiva e eficiente:

- **Sugestões automáticas:** Ao clicar na barra de pesquisa, o sistema apresenta automaticamente uma lista de sugestões de lugares de estacionamento. Caso o utilizador pretenda introduzir manualmente a morada da localização pretendida, será exibida uma janela com a opção de lugares perto da mesma.
- **Detalhes do estacionamento:** Ao selecionar um dos lugares sugeridos, é apresentada uma janela com informações detalhadas, incluindo:
    - Fotografia do local, facilitando a sua identificação;
    - Estado do estacionamento (livre ou ocupado);
    - Botão para iniciar a viagem, disponível apenas se o estacionamento estiver livre;
    - Se o lugar estiver ocupado, não será exibida qualquer opção de seleção.

As **Figuras 6 e 7** representam as sugestões automáticas e as **Figuras 8 e 9** demonstram os detalhes do estacionamento.

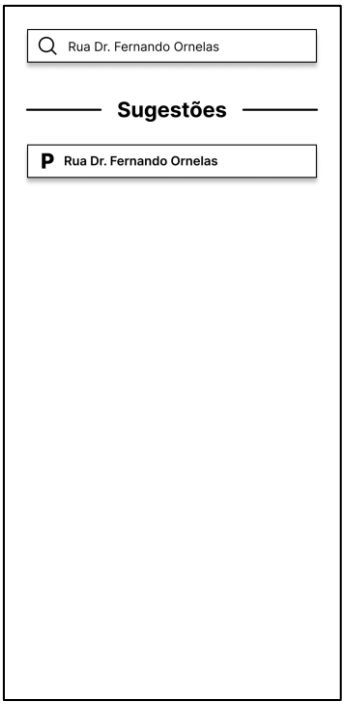

Figura 7 - *Ecrã de sugestão/Pesquisa.*

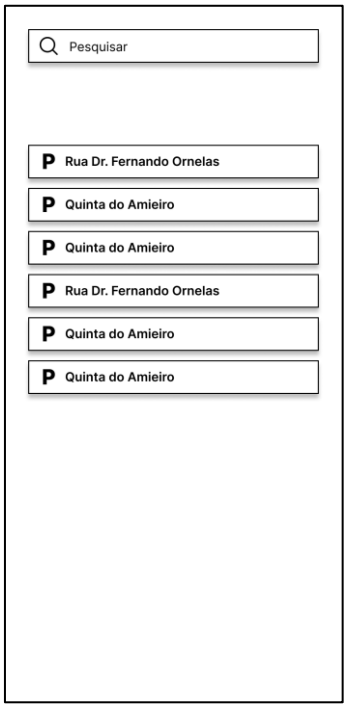

Figura 6 - *Ecrã de sugestões.*

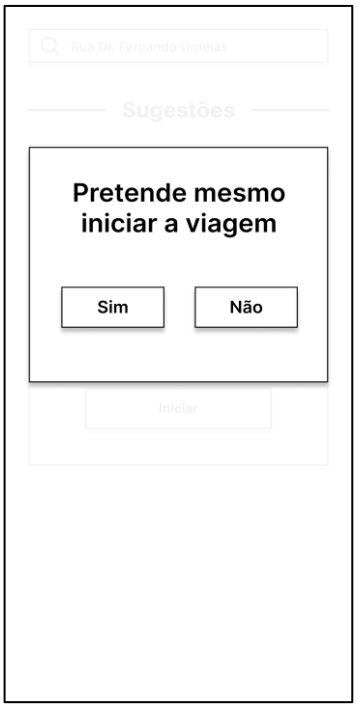

Figura 9 *- Ecrã iniciar viagem.*

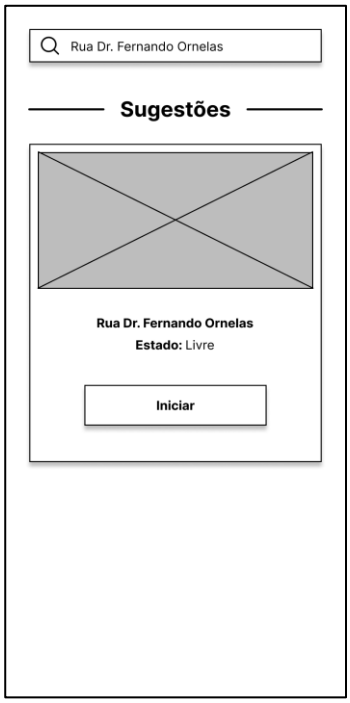

Figura 8 *- Ecrã de aviso.*

3.2.3. Ecrã de navegação

O ecrã de navegação proporciona uma visualização detalhada da rota em tempo real até ao local de estacionamento.

As principais funcionalidades incluem:

- **Visualização da rota em tempo real:** Apresentação clara do trajeto a seguir, facilitando a navegação até ao destino pretendido.
- **Informação essencial sobre a viagem:** Exibição da distância restante, hora de chegada ao local e da duração prevista do percurso.
- **Botão de cancelamento da viagem:** Possibilidade de interromper a navegação a qualquer momento através de uma interface intuitiva e de fácil acesso.
- **Sistema de notificações:** Caso o lugar de estacionamento inicialmente previsto fique indisponível, o sistema apresenta um *"pop-up"* a sugerir um novo destino, permitindo ao utilizador decidir se deseja aceitar a alteração ou negar.
- **Confirmação de ações:** Antes de iniciar ou cancelar a viagem, o sistema solicita a confirmação do utilizador, prevenindo ações involuntárias.

Estas funcionalidades que estão demonstradas nas **Figuras 10, 11 e 12** foram desenvolvidas para melhorar a experiência do utilizador, garantindo um processo de navegação eficiente e adaptável às circunstâncias em tempo real.

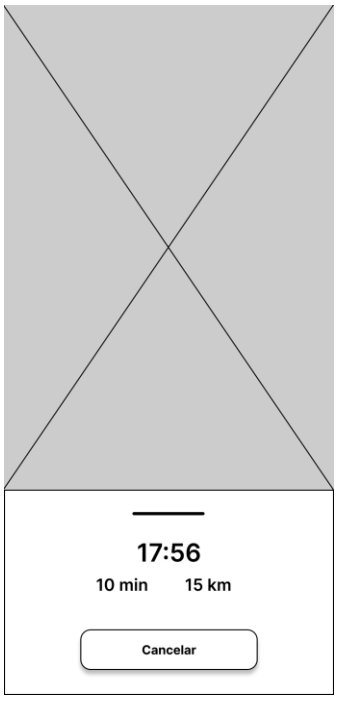

Figura 11 *- Ecrã navegação/cancelar.*

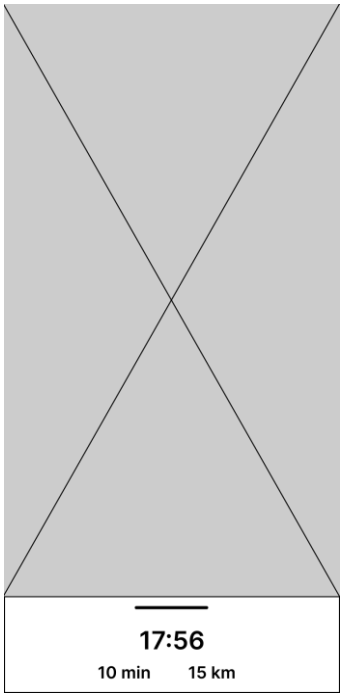

Figura 10 *- Ecrã de navegação.*

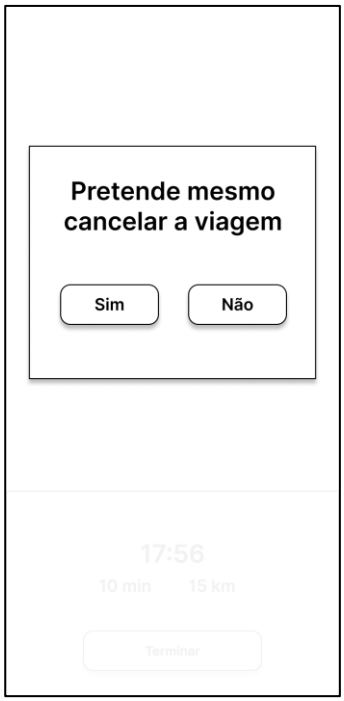

Figura 12 *- Ecrã de cancelamento.*

3.3. Modelo ER

O Modelo ER é utilizado para criar e representar a estrutura lógica de uma base de dados. Através deste modelo, é possível identificar e descrever as entidades relevantes de um sistema, os seus atributos e as relações que existem entre elas.

O modelo é representado graficamente através de um Diagrama de Entidade e Relacionamento, um tipo de fluxograma que organiza de forma clara a forma como diferentes entidades (pessoas, objetos) se interligam numa base de dados. Utiliza símbolos padronizados: retângulos para entidades, losangos para relações e ovais para atributos, todos ligados por linhas que mostram como se relacionam entre si [34].

Na **Figura 13** é apresentado o diagrama onde identificam-se duas entidades principais: “Lugar” e “Estado”. A entidade “Lugar” representa cada lugar de estacionamento registado no sistema, enquanto a entidade “Estado” descreve a condição atual desse lugar, como por exemplo “Livre” ou “Ocupado”.

A ligação entre estas duas entidades é feita através do relacionamento “E”, interpretado como “Encontra-se”. A cardinalidade associada é N:1, o que significa que vários lugares podem partilhar o mesmo estado em simultâneo, mas cada lugar só pode estar associado a um único estado num dado momento. Assim, múltiplos lugares podem estar “Disponíveis” ao mesmo tempo, mas nenhum lugar pode assumir dois estados simultaneamente.

Este modelo assegura uma organização eficiente dos dados relativos à ocupação dos lugares, permitindo consultas rápidas e atualizações imediatas sempre que o estado muda.

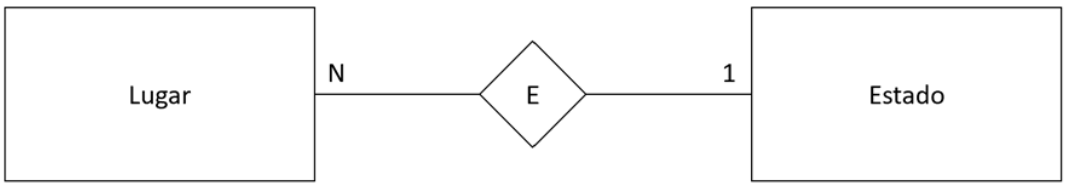

Figura 13 *- Modelo ER.*

4. Primeira implementação

Nesta versão, integramos o Mapbox, permitindo a visualização das vagas disponíveis. As informações das vagas são carregadas diretamente do Firebase, garantindo dados atualizados e centralizados. Além disso, os ícones das vagas mudam de cor de acordo com seu estado, ficando verdes quando disponíveis e vermelhos quando ocupadas, proporcionando uma visualização rápida e intuitiva.

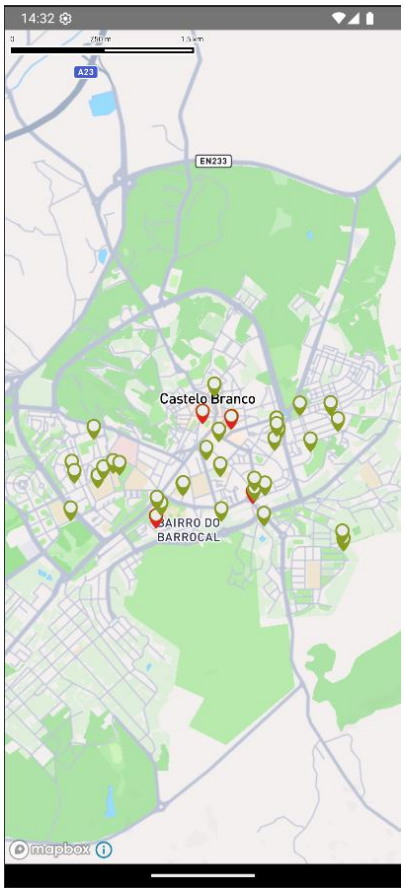

Figura 15 *- Primeira implemtação*

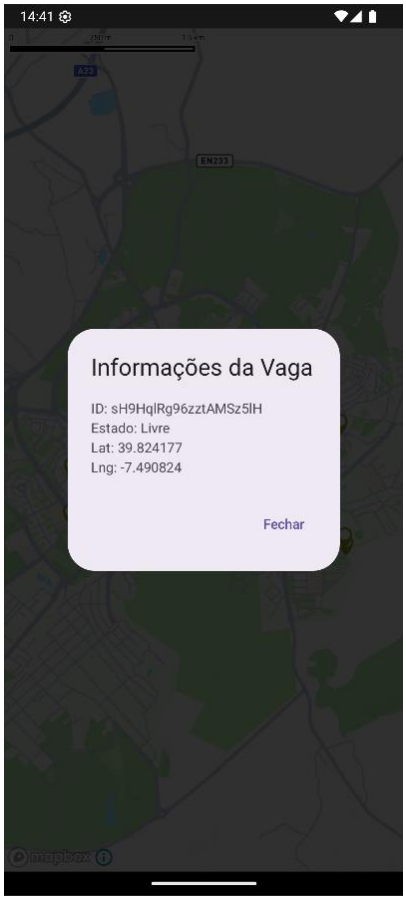

Figura 14 *- Selecionar lugar*

5. Modelo computação visual

Neste capítulo é descrito o modelo de computação visual para a aplicação: apresenta-se o *dataset* selecionado para treino, com justificação da sua escolha; comparam-se as arquiteturas de deteção testadas (YOLO11x e YOLOv8s); detalham-se os métodos de treino utilizados; e procede-se à análise dos resultados, evidenciando o desempenho e as decisões adotadas para a implementação final.

5.1. Dataset

Para o sistema de reconhecimento de veículos, recolheram-se inicialmente imagens captadas por drone, com as devidas autorizações das entidades competentes e sob supervisão do professor coordenador. Contudo, o número reduzido de imagens revelou-se insuficiente para treinar eficazmente o modelo.

Para ultrapassar a limitação do número reduzido de imagens, foi realizada uma pesquisa onde foram analisados *datasets* adequados ao nosso projeto, como o VEDAI, AU-AIR, KITTI, COCO, Munich Vehicle Dataset e DLR-MVDA; contudo, estes revelaram-se menos completos em termos de diversidade de cenários e variedade de classes anotadas, pelo que optou-se pela utilização exclusiva do VisDrone2021, desenvolvido pela equipa AISKYEYE da Universidade de Tianjin. Este conjunto disponibiliza 400 videoclipes (265.228 *frames*) e 10.209 imagens estáticas, com mais de 2,5 milhões de anotações manuais, assegurando uma elevada qualidade das mesmas. Além da sua dimensão, destaca-se pela diversidade de contextos como imagens recolhidas em 14 cidades, cenários urbanos e rurais sob diferentes condições meteorológicas e de iluminação, aproximando o treino das condições reais de aplicação. O *dataset* inclui ainda classes diretamente relevantes para o projeto, como carros, carrinhas, camiões, triciclos, autocarros e motociclos, e apresenta cenários de densidade variável, permitindo avaliar o desempenho do modelo em diferentes níveis de complexidade [35][36].

As **Figuras 16 e 17**, apresentadas abaixo, ilustram exemplos das imagens que compõem o *dataset*.

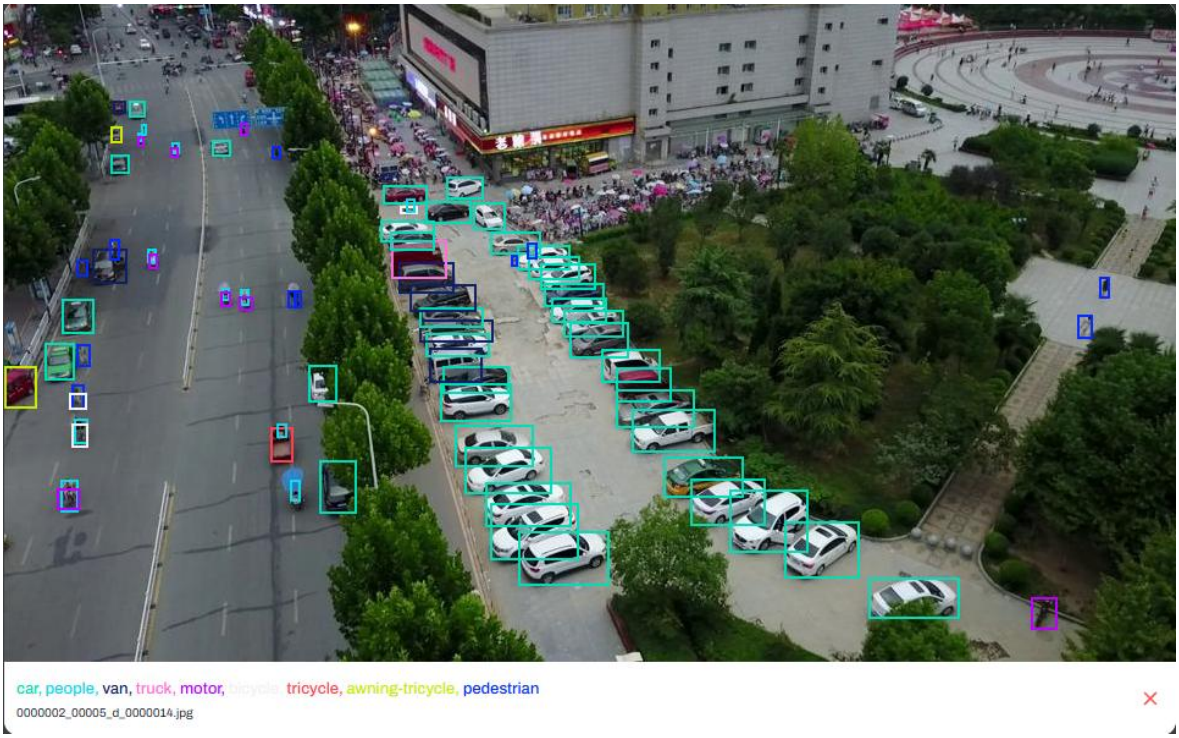

Figura 16 *- Imagem do dataset.*

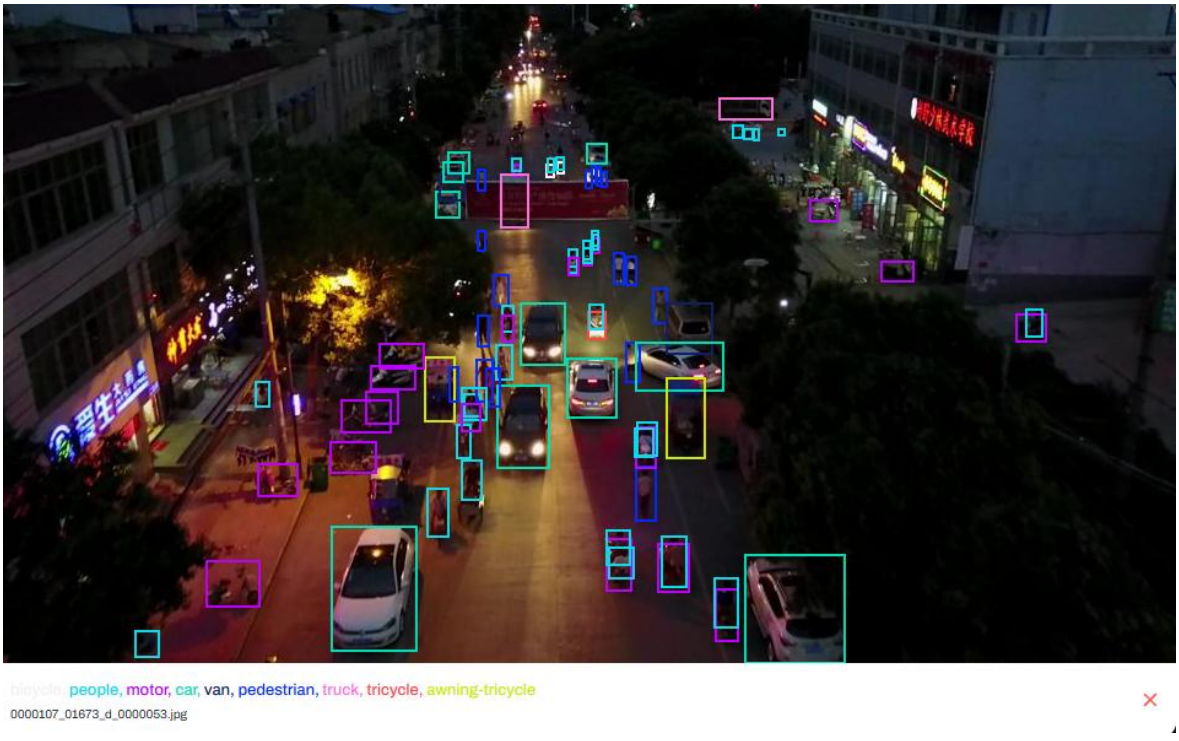

Figura 17 *- Imagem do dataset.*

Assim, a escolha do VisDrone2021 justifica-se pela sua dimensão, diversidade e relevância prática, garantindo uma base de dados robusta e representativa para o treino e validação do sistema de reconhecimento de veículos desenvolvido.

5.2. Arquiteturas testadas

A escolha da arquitetura de deteção de objetos foi orientada por dois requisitos centrais: precisão na identificação de veículos em cenários complexos e eficiência computacional para inferência em tempo real. Deste modo, avaliámos duas arquiteturas YOLO disponibilizadas pelo Ultralytics, YOLO11x e YOLOv8s, que representam extremos de complexidade e desempenho na família YOLO.

A YOLO11x é uma arquitetura profunda e densa, com aproximadamente 56,9 milhões de parâmetros e ≈195 GFLOPs de custo computacional. O seu *backbone* extenso e múltiplas camadas de cabeça de deteção proporcionam maior capacidade de representação de características, traduzindo-se em ganhos de mAP mesmo em limiares de IoU exigentes (mAP@50-95). Contudo, esta robustez acarreta maior latência, com latências de inferência em torno de 24 ms por imagem em PyTorch [37].

Em contraste, a YOLOv8s foi concebida como uma versão "*small*" otimizada para cenários práticos. Com apenas 11,2 milhões de parâmetros e baixo custo de ≈26 GFLOPs, oferece latência significativamente mais rápida (≈4 ms por imagem) e menor uso de memória, mantendo mAP50 competitivo. A sua leveza facilita a implantação em dispositivos com recursos limitados ou numa cadeia de processamento em tempo real [38].

Optámos por testar ambas as arquiteturas durante 50 épocas e, com base nos resultados iniciais, selecionámos a mais adequada para prosseguir com um treino aprofundado de 100 épocas [39].

5.3. Métodos de treino

Para treinar o nosso sistema de reconhecimento de veículos utilizámos o método model.train() do Ultralytics, executado em ambiente Google Colab com GPU Tesla T4. O processo segue os requisitos internos do Ultralytics efetua o *download* e carregamento de pesos pré-treinados, e prepara automaticamente o *dataset* VisDrone através de scripts de conversão de imagens e anotações.

Em cada época de treino, o modelo processa lotes de imagens de dimensão 640×640, aplica transformações de *data augmentation* (mosaic, HSV jitter, flip) e calcula, por amostra:

- ***Loss* de *bounding box* (*box_loss*):** penaliza diferenças nas coordenadas das caixas;
- ***Loss* de classificação (cls_*loss*):** penaliza erros na previsão de classes;

- ***Loss* de distribuição de formatos (dfl_*loss*):** ajusta a regressão de forma mais precisa através de IoU avançados.

O algoritmo de otimização seleciona automaticamente o melhor otimizador e executa *mixed-precision* (AMP) para acelerar os cálculos.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     53/100      5.64G      1.045     0.6276     0.8646        232        640: 100% ──────────── 1618/1618 2.9it/s 9:23
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ──────────── 69/69 4.0it/s 17.4s
                   all        548      38759      0.587      0.459      0.479      0.299
```

Figura 18 *- Output de uma época.*

5.3.1. Tempo de treino

YOLOv8s (50 épocas): duração total de aproximadamente 2 horas, com cada época levando em média 2:30 minutos.

YOLO11x (50 épocas): duração total de cerca de 6 horas, com cada época demorando aproximadamente 9:20 minutos.

YOLO11x (100 épocas): duração total de aproximadamente 12 horas, mantendo cerca de 9:20 minutos por época.

5.4. Análise dos resultados

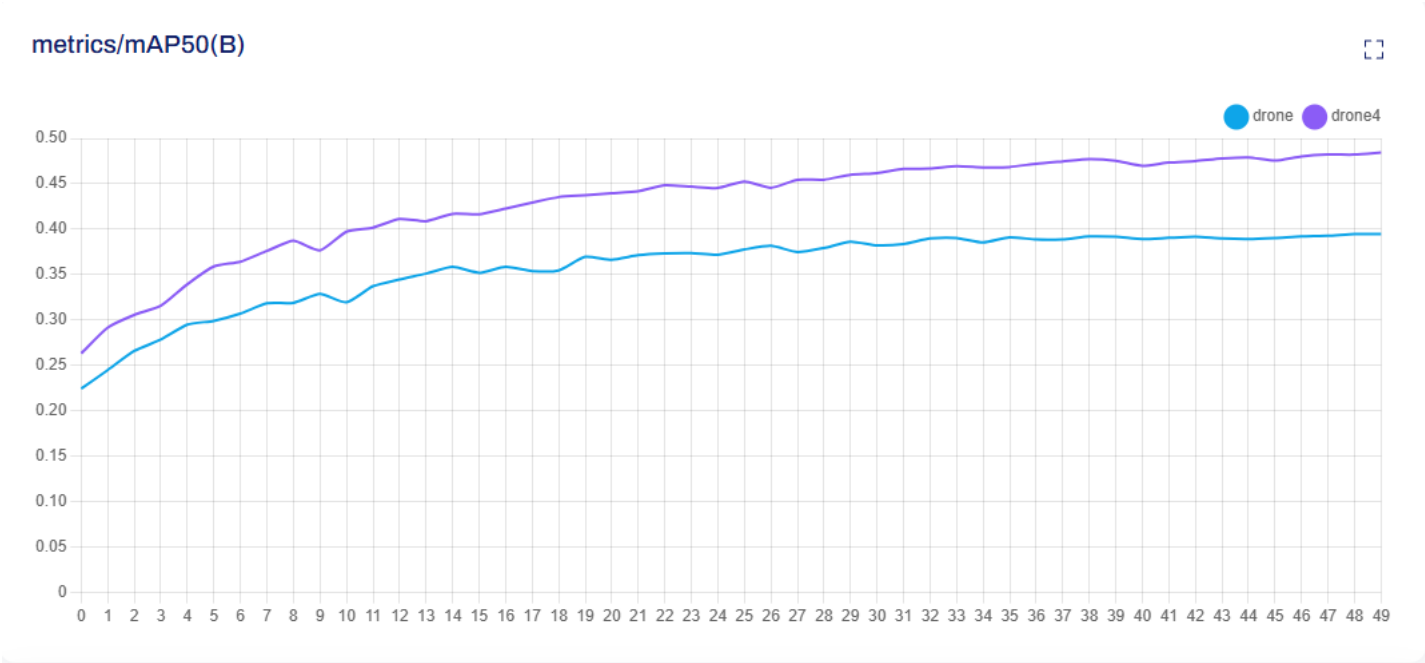

Figura 19 *- Métrica mAP50.*

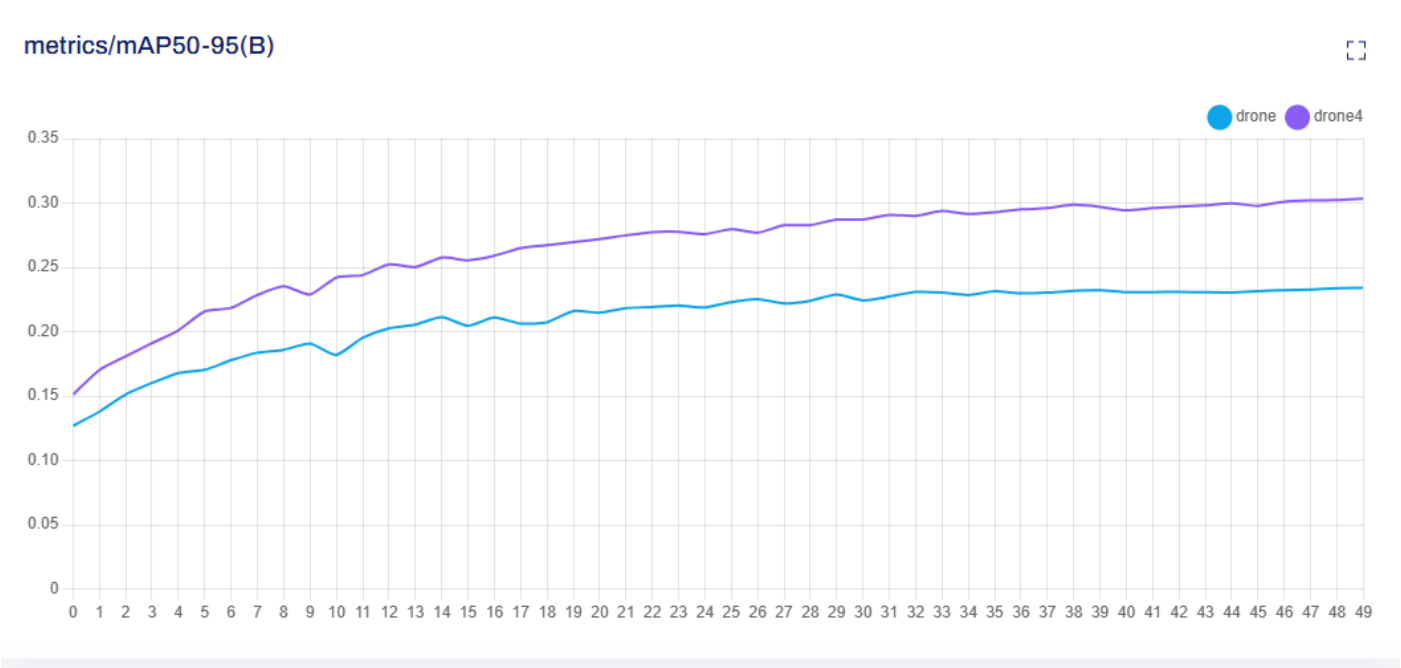

Figura 20 *- Métrica mAP50-95.*

Após 50 épocas de treino, as curvas de mAP50 e mAP50-95 revelam que a variante YOLO11x (Drone4) apresenta precisão consistentemente superior, com mAP50 a estabilizar próximo de 0,50 e mAP50-95 a rondar 0,30. Em contrapartida, a YOLOv8s (Drone) atinge um mAP50 máximo próximo de 0,40 e um mAP50-95 de aproximadamente 0,25.

Este comportamento era previsível, dado que o modelo YOLO11x é consideravelmente maior e mais complexo, o que lhe confere maior capacidade de aprender representações mais sofisticadas e, por conseguinte, alcançar melhor precisão. Contudo, essa superioridade acarreta um custo em termos de desempenho: a latência por inferência da YOLO11x é significativamente maior.

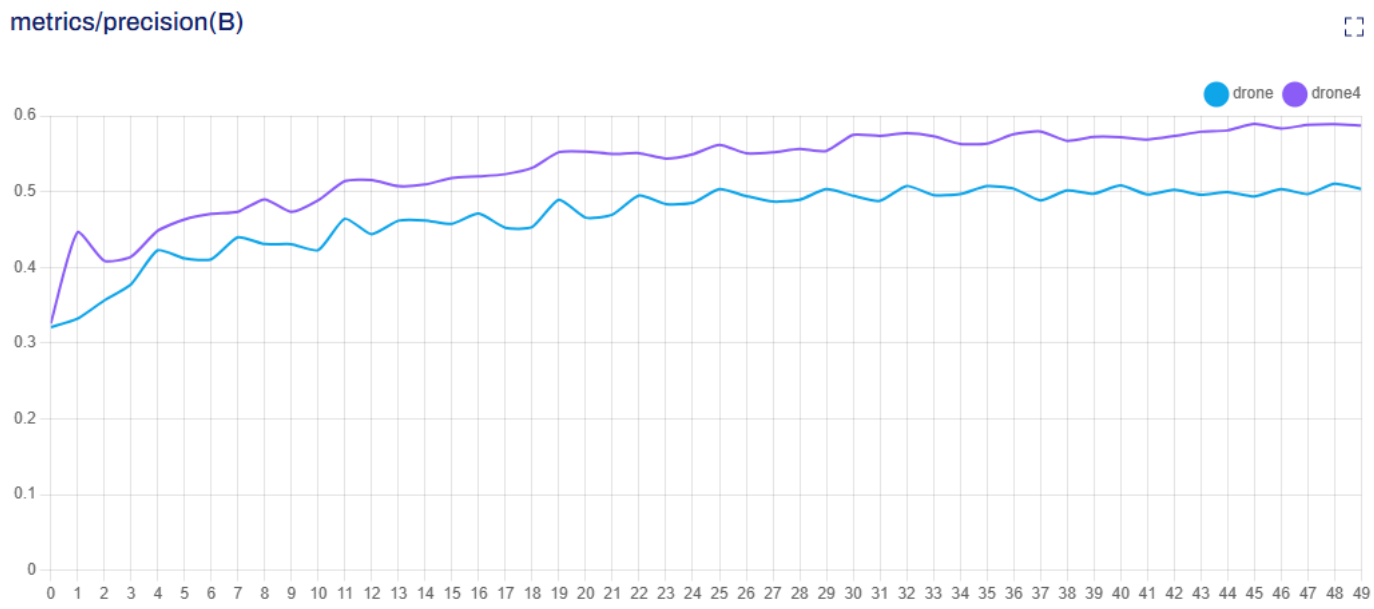

Figura 21 *- Métrica precision.*

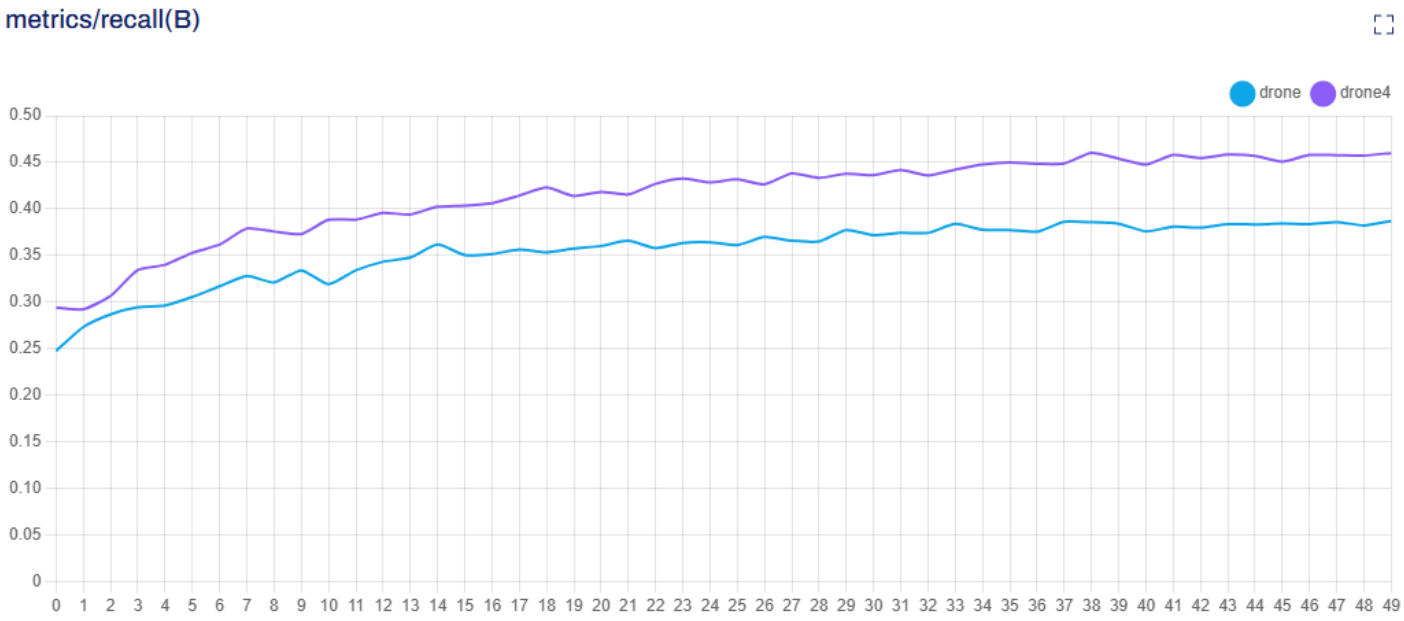

Figura 22 *- Métrica recall.*

A análise das curvas de *precision* e *recall* confirma a superioridade da YOLO11x (Drone4) face à YOLOv8s (Drone) em termos de qualidade de deteção. Ao longo de 50 épocas, a YOLO11x atinge valores de *precision* na ordem de 0,58–0,60, enquanto a YOLOv8s se estabiliza em torno de 0,48–0,50, o que traduz em predições mais fiáveis e menor taxa de falsos positivos para o modelo maior. Quanto ao *recall*, a YOLO11x regista valores entre 0,44 e 0,46, acima dos 0,36–

0,38 obtidos pela YOLOv8s, demonstrando capacidade superior de identificar verdadeiros positivos.

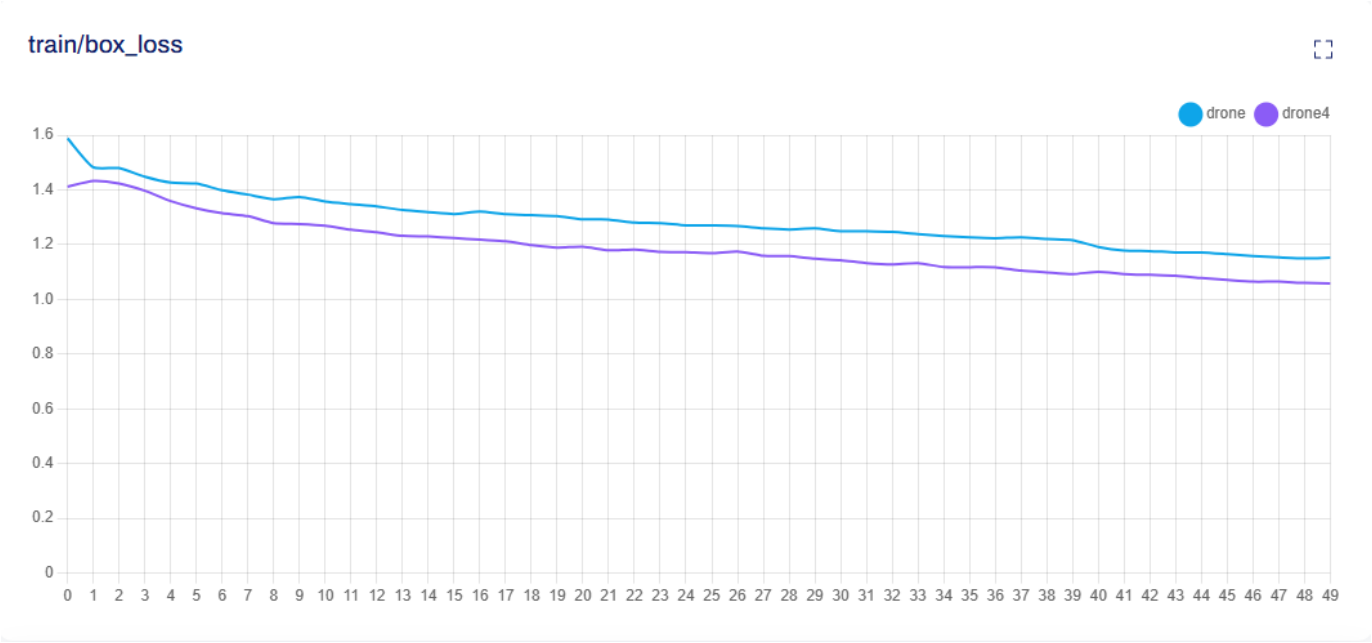

Figura 23 *- Treino box_loss.*

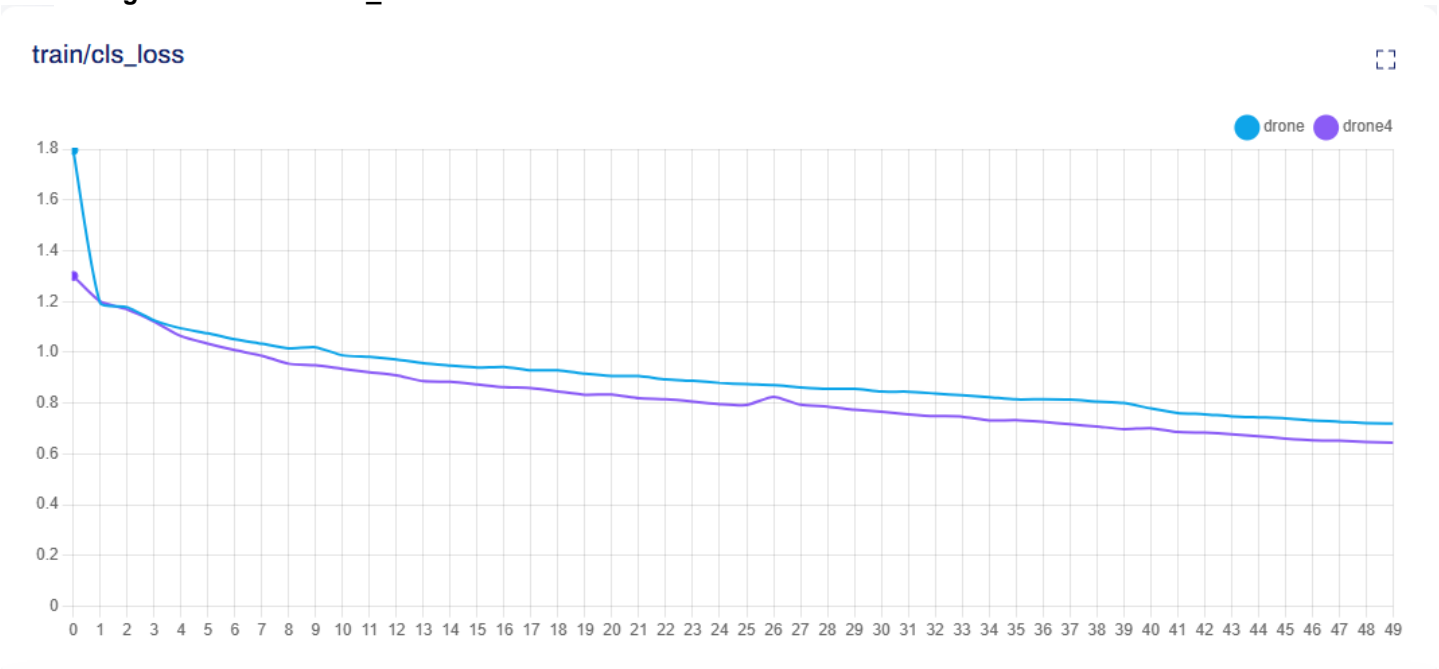

Figura 24 *- Treino cls_loss.*

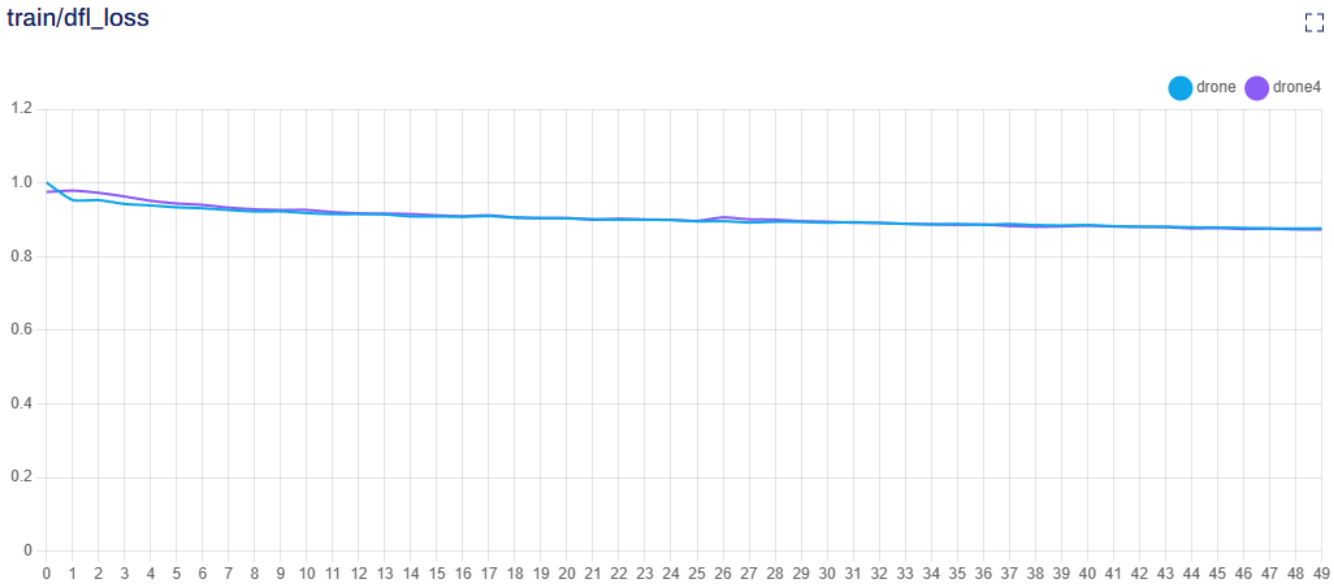

Figura 25 *- Treino dfl_loss.*

As curvas de treino de *box_loss*, cls_loss e dfl_loss revelam uma convergência estável para ambas as variantes, com a YOLO11x (Drone4) a mostrar perdas consistentemente menores do que a YOLOv8s (Drone):

- ***Box loss*:** A YOLOv8s inicia em cerca de 1,58 e desce progressivamente até 1,15 ao fim de 50 épocas, enquanto a YOLO11x parte de 1,42 e converge para aproximadamente 1,05. A diferença indica que o modelo maior regressa as caixas com maior precisão desde as primeiras épocas e mantém essa vantagem ao longo do treino.
- **Cls *loss*:** Nas primeiras épocas, ambas as curvas apresentam queda acentuada – a YOLOv8s cai de 1,80 para cerca de 0,73, e a YOLO11x de 1,32 para 0,63. A diferença entre ambas de ≈0,1 persiste até ao final, demonstrando maior confiança na classificação de objetos pelo modelo mais complexo.
- **DFL *loss*:** As curvas de distribuição focal *loss* começam em cerca de 1,00 para a YOLOv8s e 0,95 para a YOLO11x, convergindo ambas para valores próximos de 0,88–0,90. A sobreposição das curvas sugere que este componente beneficia igualmente das duas arquiteturas, já que o refinamento de forma é aplicado de forma semelhante.

Estes resultados confirmam que o treino decorreu sem instabilidades e que a YOLO11x apresenta desempenho superior em regressão e classificação.

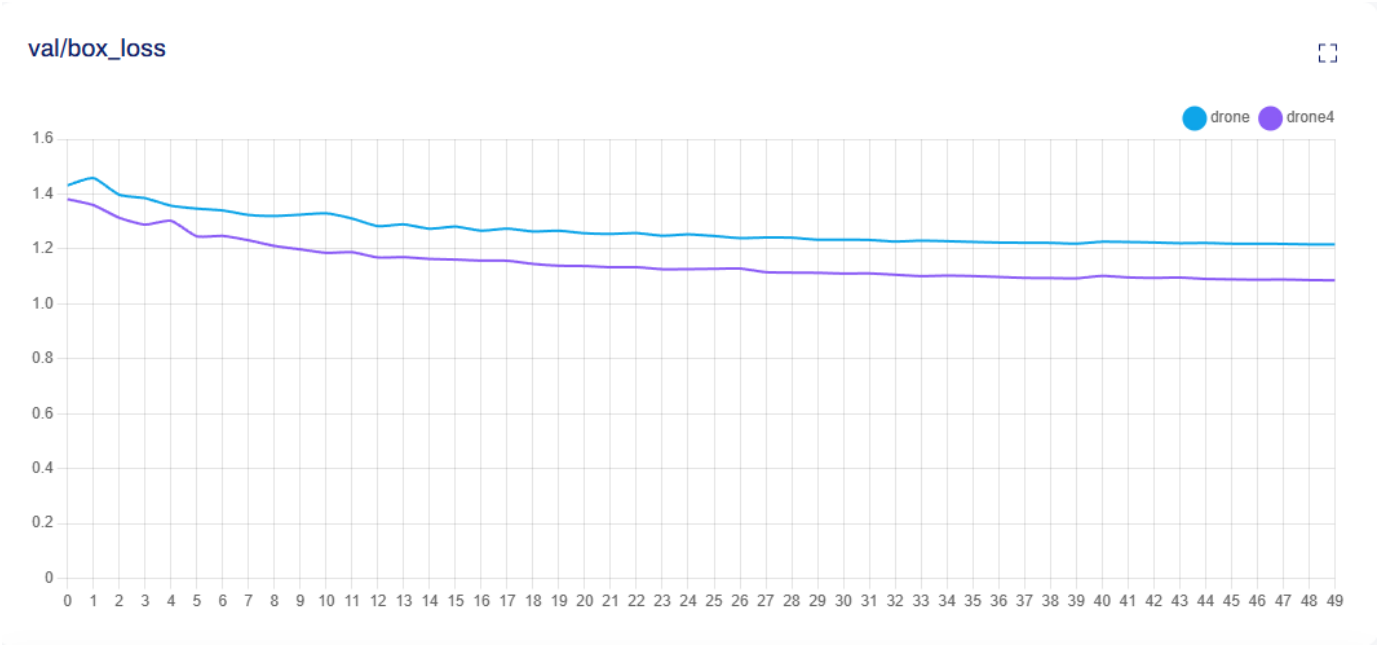

Figura 26 *- Validação box_loss.*

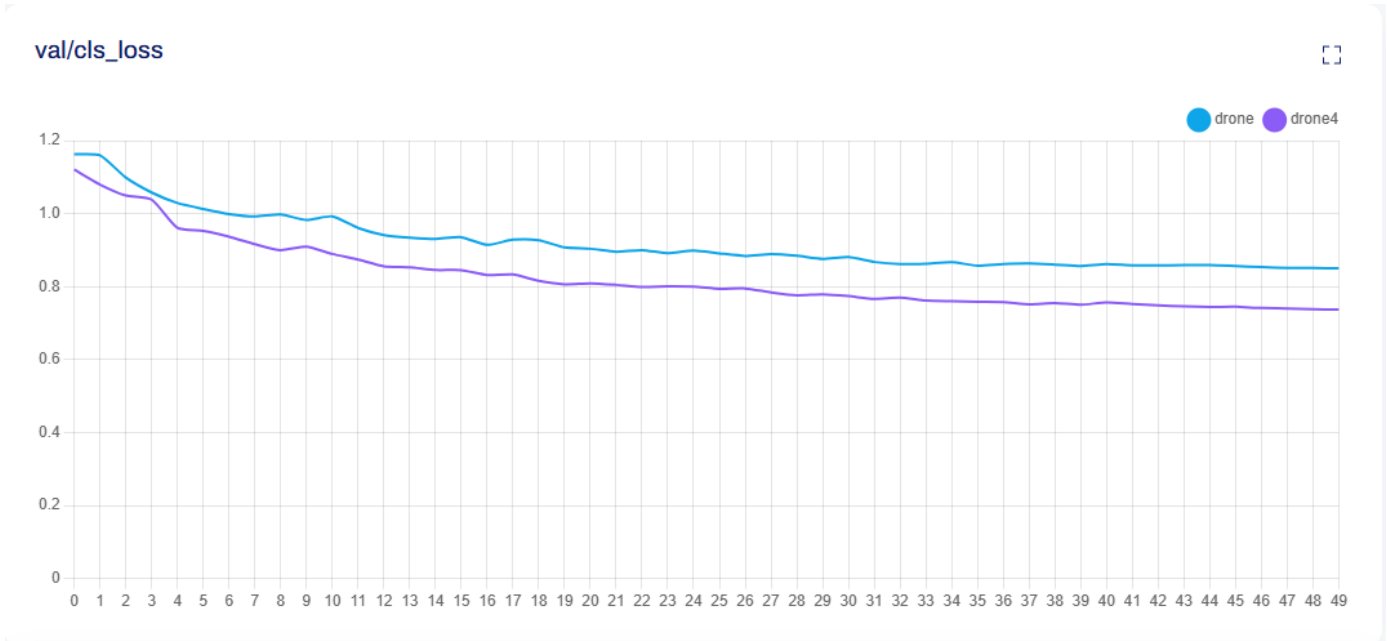

Figura 27 *- Validação box_loss.*

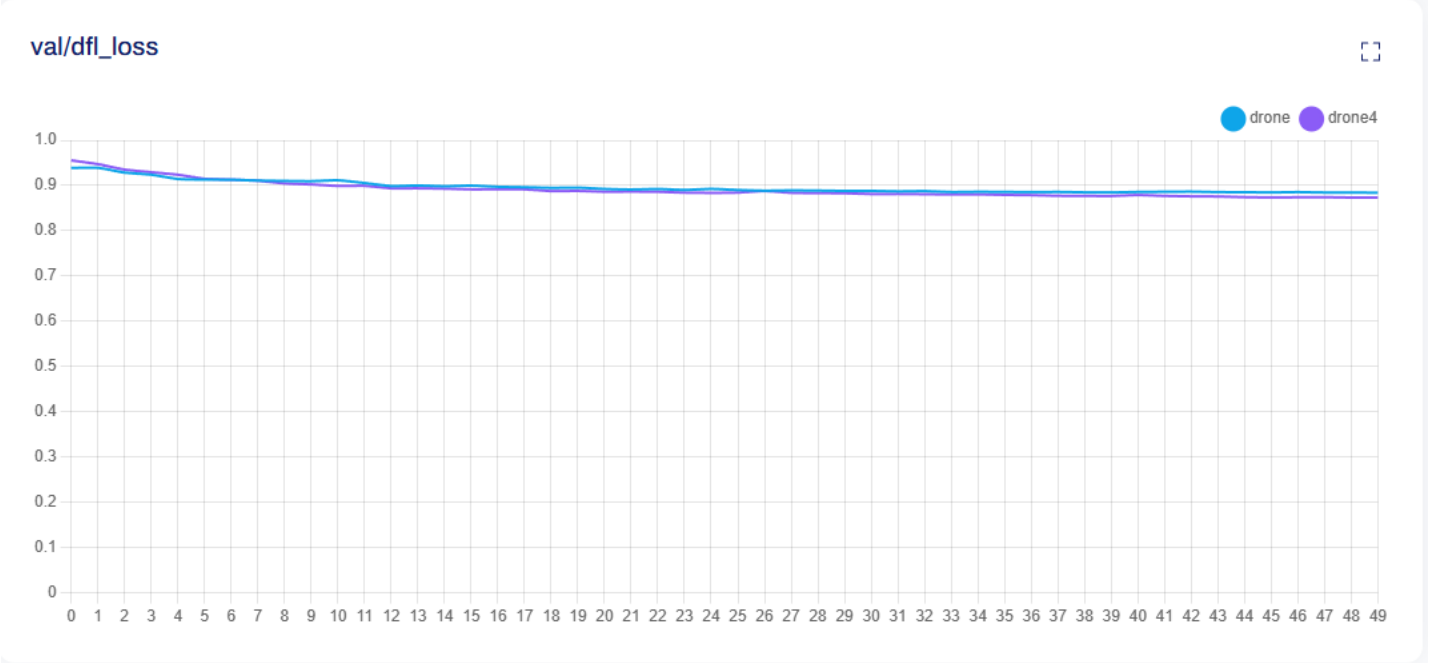

Figura 28 *- Validação dfl_loss.*

Nos gráficos de validação, as perdas mantêm padrões semelhantes aos do treino, com a YOLO11x (Drone4) a registar valores inferiores à YOLOv8s (Drone), confirmando a sua robustez:

- **Val *box loss*:** A YOLOv8s parte de ≈1,45 e desce até cerca de 1,22, enquanto a YOLO11x inicia em ≈1,38 e converge para ≈1,08. A distância estável entre as curvas indica que o modelo maior generaliza melhor a regressão das caixas.
- **Val cls *loss*:** Ambos os modelos apresentam queda rápida nas primeiras épocas, com a YOLOv8s a reduzir de 1,15 para ≈0,85 e a YOLO11x de 1,08 para ≈0,75. A persistente vantagem de ≈0,10 a favor da YOLO11x reflete maior fiabilidade na classificação em dados de validação.
- **Val DFL *loss*:** As curvas partem em torno de 0,95 (Drone) e 0,93 (Drone4), estabilizando ambas em ≈0,88. A quase sobreposição confirma que o *Distribution Focal Loss* beneficia de forma equivalente ambas as arquiteturas durante a validação.

Estes resultados validam que o modelo YOLO11x não só treina com perdas inferiores, mas também mantém melhor desempenho em dados não vistos, justificando o seu uso para o treino alargado de 100 épocas.

6. Desenvolvimento da aplicação móvel

Neste capítulo detalha-se a implementação prática do ParkFinder, apresentando em primeiro lugar os requisitos funcionais e as permissões necessários para o correto funcionamento. Em seguida, descrevem-se em profundidade as interfaces principais e a lógica de navegação, evidenciando o fluxo de interação, as decisões de design de UI/UX e as integrações com os serviços externos.

Figura 29 *– Logo.*

6.1. Requisitos e permissões

Este subcapítulo identifica os pré-requisitos técnicos e as autorizações que a aplicação deve obter para oferecer todas as funcionalidades planeadas de forma segura e eficiente.

6.1.1. Requisitos mínimos do dispositivo

O dispositivo móvel para a utilização da aplicação deve cumprir os seguintes requisitos:

- **Sistema operativo:** Android 10 (API nível 29) ou superior;
- **Serviços de localização:** Ativos;
- **Conectividade:** Acesso à Internet (Wi-Fi ou dados móveis);
- **Hardware:** Sensores de GPS.

6.1.2. Permissões necessárias

6.1.2.1. Acesso à localização precisa

A aplicação exige permissão de localização precisa para:

- Calcular rotas até lugares de estacionamento;
- Exibir os lugares de estacionamento e o seu estado (livre/ocupado);
- Permitir funcionalidades de pesquisa de pontos de interesse.

A **Figura 30** apresenta o aviso exibido ao utilizador para conceder permissões. Caso o utilizador autorize apenas a localização aproximada, a aplicação

disponibiliza apenas um mapa básico, sem rotas, lugares de estacionamento assinalados ou funcionalidades de pesquisa avançada.

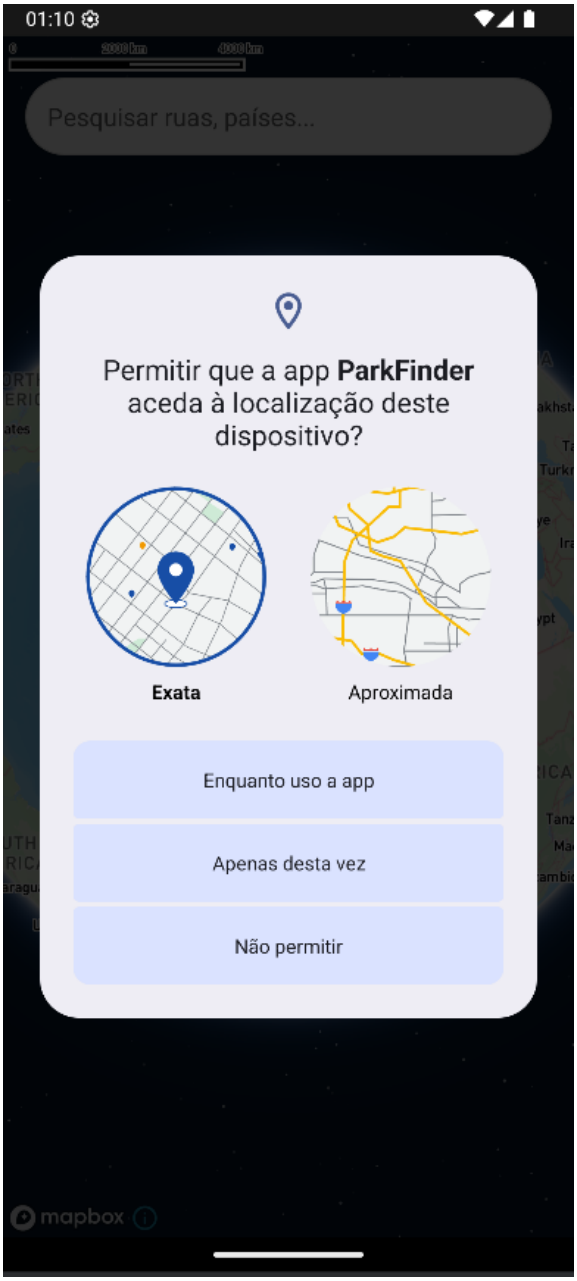

Figura 30 *- Pedir permissão para aceder à localização.*

6.1.2.2. Acesso à internet

Primeira execução sem internet (Figura 31):

- Apresenta um alerta “Sem conexão à Internet”;
- Exibe o mapa a negro até a ligação ser restabelecida, momento em que o mapa carrega automaticamente.

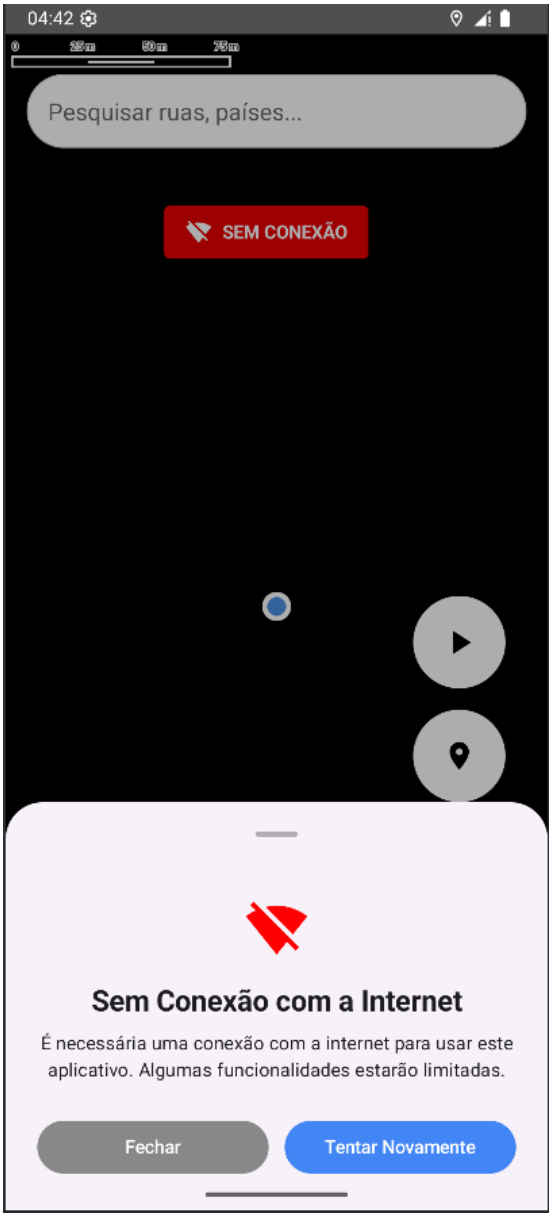

Figura 31 *- Primeira execução da aplicação sem acesso à internet.*

Execuções subsequentes sem internet (Figura 32):

- Exibe o mapa e as vagas carregadas em cache, embora possam não refletir o estado mais recente.
- A funcionalidade de iniciar rota permanece desativada até a Internet voltar.

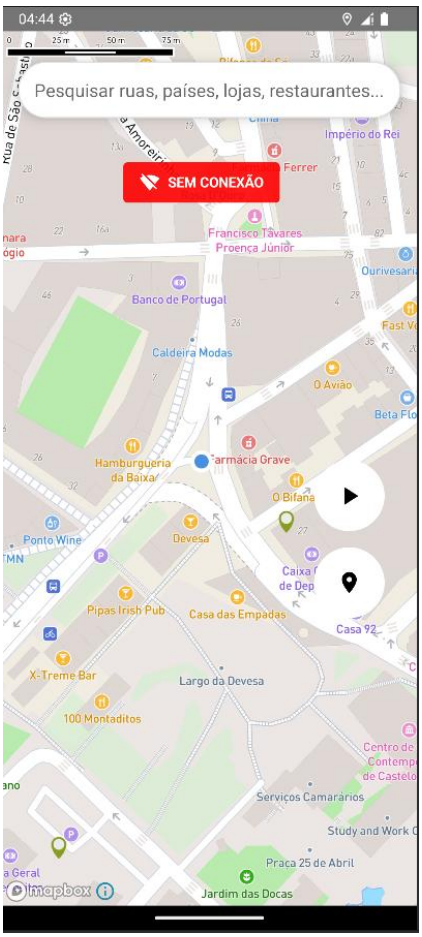

Figura 32 *- Segunda execução da aplicação sem acesso à internet.*

6.1.3. Comportamento em modo offline e sincronização

6.1.3.1. Início de rota sem internet

Se o utilizador tentar iniciar uma rota em modo offline, aparece um aviso que não é possível devido ao modo offline (**Figura 33**).

Quando a ligação é restabelecida, todos os pontos de interesse e o estado da rota são automaticamente atualizados em segundo plano e permite iniciar rota.

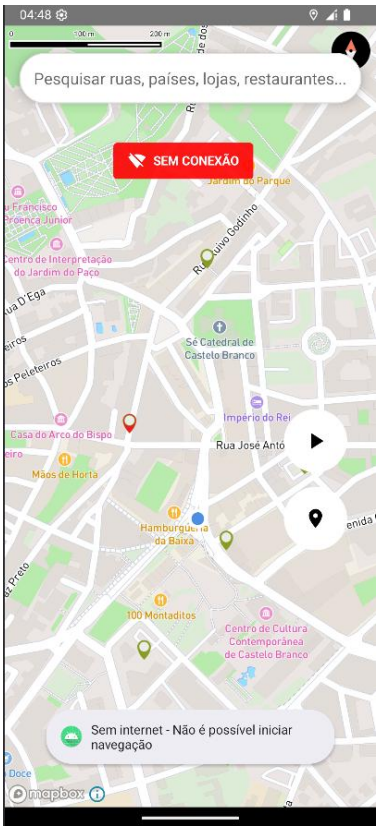

***Figura 33** - Tentativa iniciar rota em modo offline.*

6.1.3.2. Perda de conectividade durante a rota

A rota continua a ser exibida e a travessia prossegue sem interrupções (**Figura 34**).

Quando a Internet retorna, o mapa, as vagas e a rota são sincronizados imediatamente para refletir o estado real.

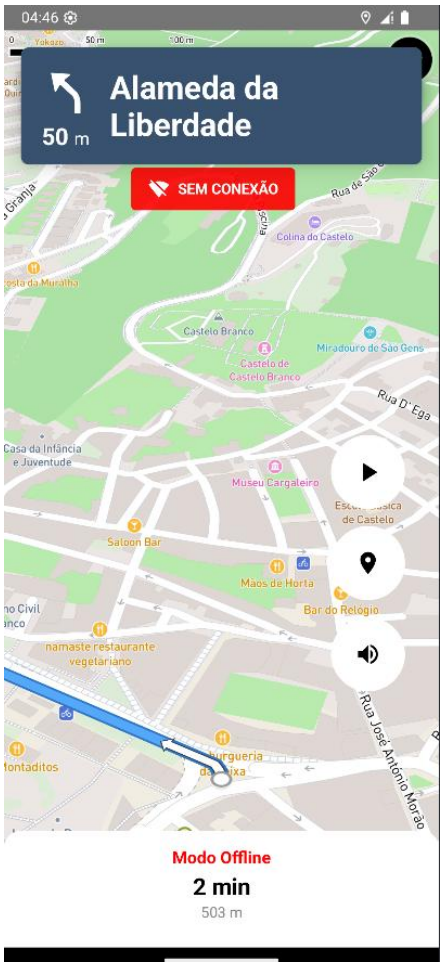

***Figura 34** - Navegação em rota sem acesso à Internet.*

6.1.4. Resumo dos cenários de uso

Tabela 6 - Resumo cenários de uso

Cenário	Permissão de Localização	Internet na 1.ª Execução	Funcionalidade Principal
Localização precisa + Internet disponível	Concedida	Sim	Rotas, vagas em tempo real e pesquisas
Localização precisa + Sem Internet (1.ª)	Concedida	Não	Alerta de erro + mapa a negro
Localização precisa + Sem Internet (sub.)	Concedida	Não	Mapa em cache, vagas não atualizadas, sem rotas
Localização aproximada + Internet	Apenas aproximada	Sim	Mapa simples sem rotas nem vagas
Durante rota + perda de Internet	—	Intermitente	Navegação continua + sincronização ao reconectar

6.2. Tecnologias

Para o desenvolvimento da aplicação móvel, foram escolhidas tecnologias modernas e robustas que permitissem criar uma solução eficiente, escalável e de fácil manutenção. Com base nesses requisitos a escolha recaiu sobre a linguagem Kotlin para a programação principal e Jetpack Compose como framework da interface do utilizador. Ambas as linguagens são amplamente utilizadas pela comunidade Android e recomendadas pela Google. Para tirar o maior proveito destas tecnologias e de forma a conseguirmos realizar testes ao longo de todo o processo as mesmas foram integradas com o apoio do Android Studio.

6.2.1. Kotlin

Kotlin é uma linguagem de programação *open-source*, com tipagem estática e totalmente compatível com Java, criada pela JetBrains em 2011. Destaca-se pelos seguintes pontos:

- **Sintaxe concisa e expressiva:** reduz código repetitivo e torna os programas mais fáceis de ler e manter;
- ***Null safety*:** previne erros de referência nula já durante a compilação, aumentando a fiabilidade da aplicação;
- **Interoperabilidade com Java:** permite utilizar bibliotecas e *frameworks* Java existentes sem esforço adicional;
- **Versatilidade multiplataforma:** além de Android, suporta desenvolvimento para a JVM, aplicações *web* (via Kotlin/JS) e aplicações nativas (via Kotlin/Native).

O Google recomenda Kotlin como linguagem principal para desenvolvimento Android, e mais de 80% das aplicações mais populares na *Play Store* usam-no, demonstrando a sua maturidade e ampla adoção [40][41][42].

6.2.2. Jetpack Compose

Jetpack Compose é uma biblioteca de UI declarativa da Google, escrito em Kotlin e lançado em 2019 como sucessor ao sistema baseado em XML. Seus principais benefícios incluem:

- **Desenvolvimento reativo**: a *interface* é recriada automaticamente a partir do estado da aplicação, eliminando código imperativo de atualização manual;
- **Componentes "*composable*"**: funções Kotlin que representam elementos visuais de forma modular e reutilizável, simplificando a composição de *layouts* complexos;
- **Pré-visualização em tempo real**: o Android Studio exibe alterações na UI instantaneamente, acelerando testes e refinamento de design;
- **Adoção incremental**: compatível com projetos que ainda utilizam XML, permitindo migrações graduais sem impactar o código existente;
- **Suporte nativo a material design**: temas dinâmicos, animações e componentes prontos facilitam a criação de *interfaces* modernas e consistentes [43][44].

Em conjunto, Kotlin e Jetpack Compose formam uma *stack* robusta e alinhada com as melhores práticas Android, viabilizando desenvolvimento rápido, legível e de alta qualidade.

6.3. Interfaces

Neste capítulo, serão descritas detalhadamente todas as funcionalidades centrais de cada ecrã, com o objetivo de ilustrar a experiência de utilizador e o fluxo de interação implementado.

6.3.1. Ecrã inicial

O ecrã inicial da aplicação foi desenvolvido para proporcionar uma experiência intuitiva e eficiente ao utilizador. A interface é composta pelos seguintes elementos:

Mapa interativo:

Ocupa a totalidade do ecrã, apresentando a localização atual do utilizador através de um *icon* circular azul e os lugares de estacionamento devidamente referenciados por *icons* verde ou vermelho consoante o estado de disponibilidade (livre ou ocupado).

Barra de pesquisa:

Localizada na parte superior do ecrã, permite ao utilizador introduzir um destino manualmente.

Botão de recentralização:

Posicionado do lado direito da interface, este botão permite ao utilizador reencontrar a sua localização atual no mapa. Útil para situações em que o utilizador navega manualmente pelo mapa e pretende voltar à sua perspetiva inicial.

Botão de reencaminhamento automático:

Situado acima do botão de recentralização, este controlo permite ativar ou desativar a funcionalidade de redirecionamento inteligente de rotas.

Se ativo: Caso o lugar de estacionamento selecionado pelo utilizador fique ocupado durante o percurso, o sistema redireciona automaticamente para o lugar vago mais próximo da localização atual do utilizador. (**Figura 35**)

Se inativo: A rota mantém-se inalterada, independentemente da alteração do estado do lugar (**Figura 36**).

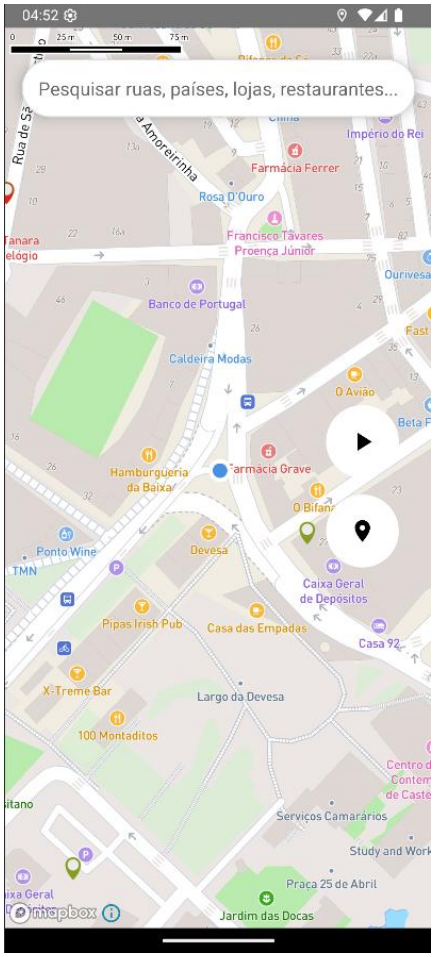

Figura 35 *- Reencaminhamento ativado.*

Figura 36 *- Reencaminhamento desativado.*

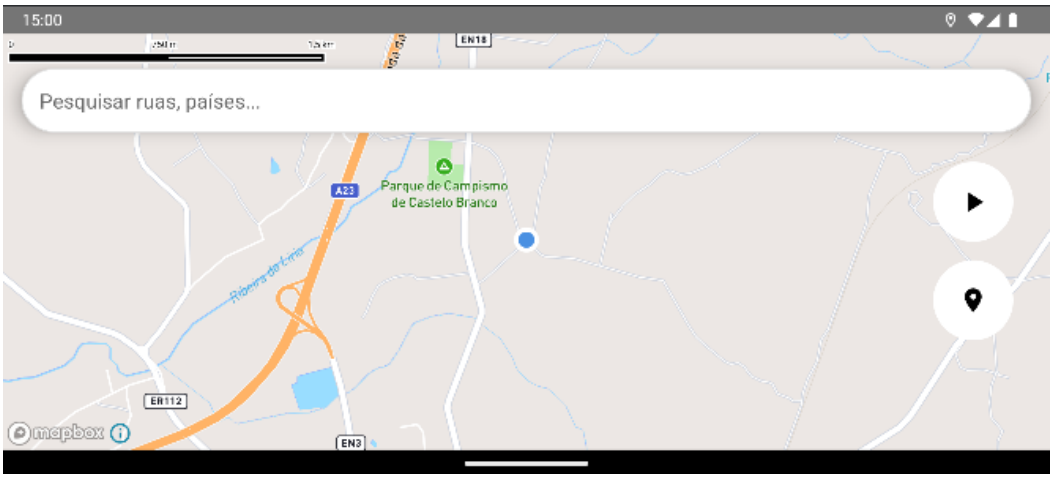

Figura 37 *- Aplicação na horizontal.*

6.3.2. Barra de pesquisa

A barra de pesquisa, localizada na parte superior do ecrã inicial, permite ao utilizador pesquisar destinos específicos. Ao iniciar a pesquisa, são apresentadas sugestões dinâmicas por baixo da barra, baseadas no texto introduzido. (**Figura 39**)

Quando o utilizador seleciona uma das opções sugeridas, é exibida uma caixa informativa na zona inferior do ecrã, contendo os seguintes elementos (**Figura 38**):

- Nome da rua ou localização pesquisada;
- Distância até ao local;
- Botão "Direções".

As métricas funcionam com base nas configurações de localidade do dispositivo:

- EUA (US), Libéria (LR), Myanmar (MM): Sistema imperial (milhas, pés);
- Demais países: Sistema métrico (metros, quilômetros).

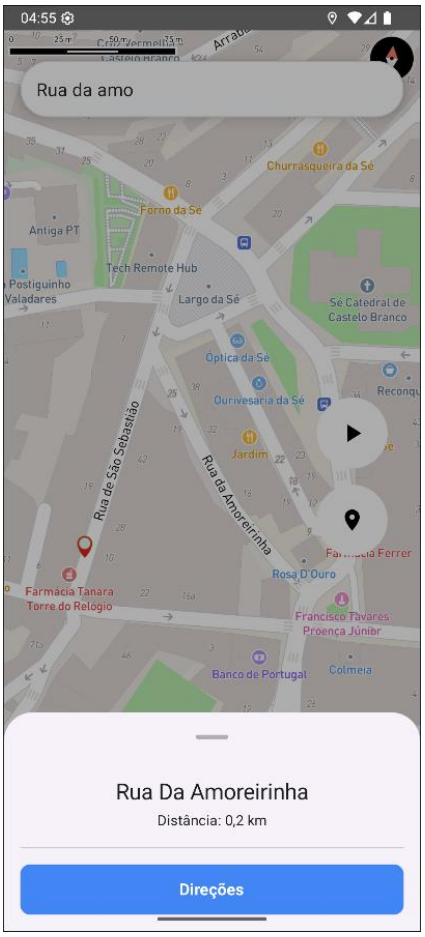

Figura 38 *- Iniciar rota.*

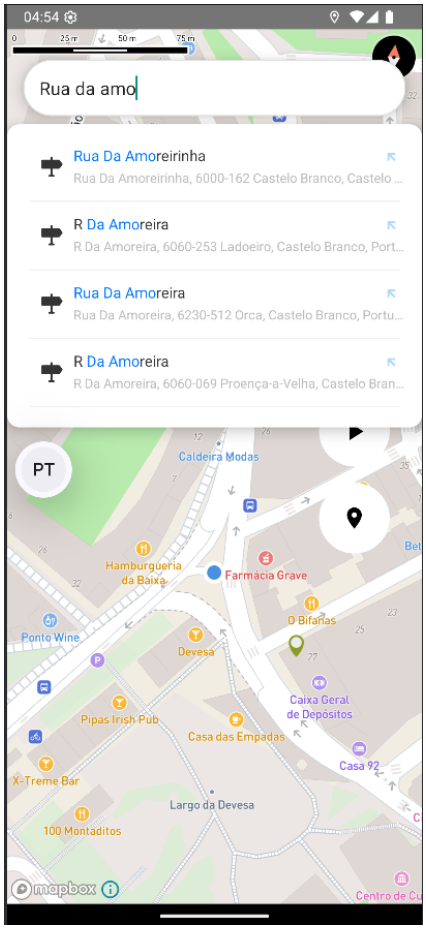

Figura 39 *- Barra de pesquisa.*

6.3.3. Iniciar rota

Após o utilizador clicar no botão "Direções" a aplicação gera automaticamente uma rota desde a localização atual até ao destino selecionado. Simultaneamente, é exibida uma caixa de informações de navegação na parte superior do ecrã, contendo detalhes essenciais para o utilizador como instruções de direções, caso o utilizador necessite de mais informações, pode clicar na caixa de navegação superior, que se expande para revelar instruções adicionais (**Figura 42**). Na parte inferior é também exibida uma caixa de informações que contém a distância restante e o tempo estimado de viagem.

O botão de recentralização, quando acionado durante a navegação, faz com que a câmara siga automaticamente a localização do utilizador. Caso o utilizador mova manualmente a câmara, o seguimento é interrompido.

Durante a navegação ativa, é apresentado um botão de assistência por voz localizado por baixo do botão de recentralização, que permite ser ativado ou desativado (**Figura 40 e 41**).

O sistema de voz da nossa aplicação funciona com base nas configurações de localidade do dispositivo.

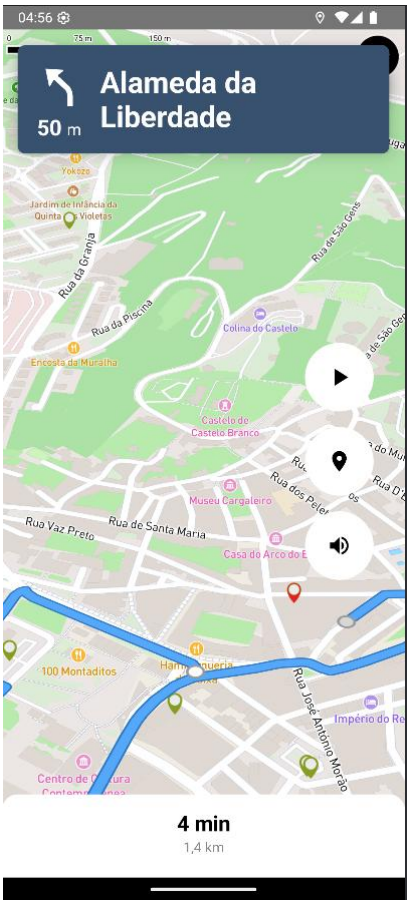

Figura 41 *- Voz ativa.*

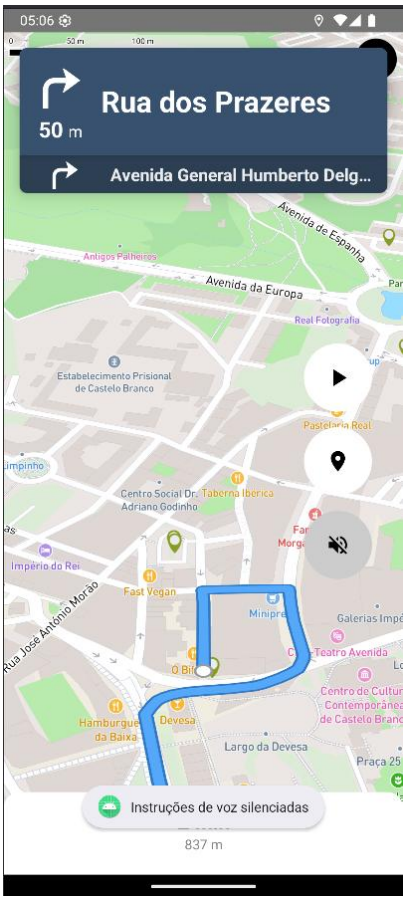

Figura 40 *- Silenciar voz.*

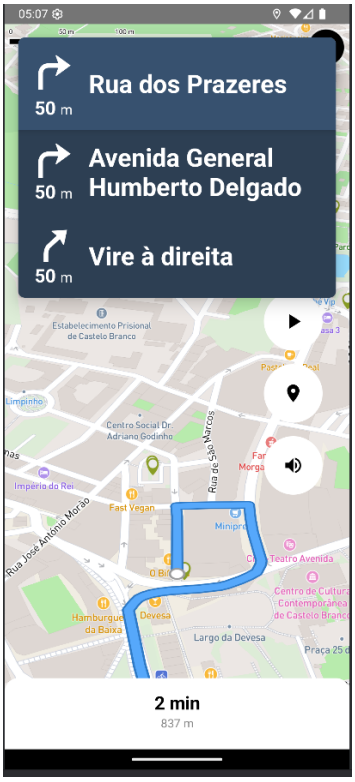

Figura 42 *- Caixa de informações de navegação.*

6.3.4. Cancelar rota

Durante uma navegação ativa, o utilizador pode cancelar a mesma de forma intuitiva através de um gesto simples, para tal, basta arrastar para cima a caixa de informações de navegação localizada na parte inferior do ecrã. Este gesto revela imediatamente o botão "Parar Navegação". Ao pressionar este botão, a rota é cancelada automaticamente (**Figura 43 e 44**).

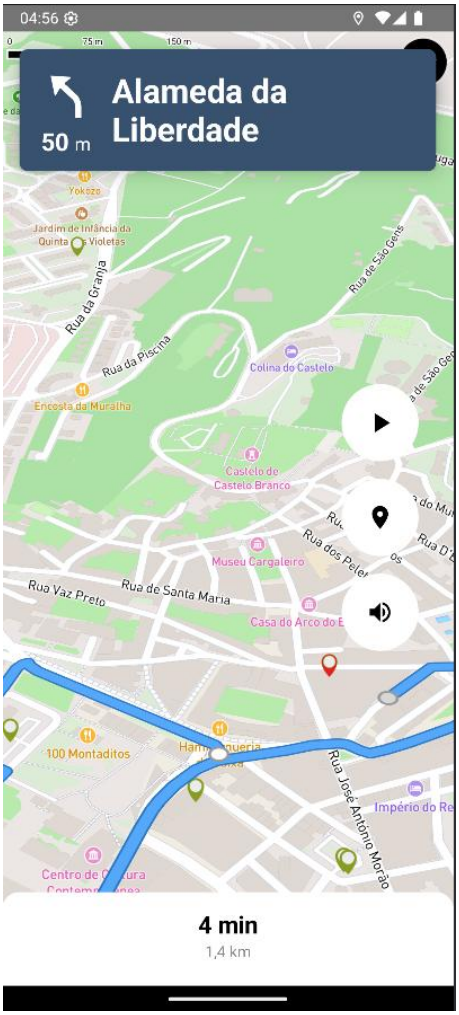

Figura 43 *- Rota ativa.*

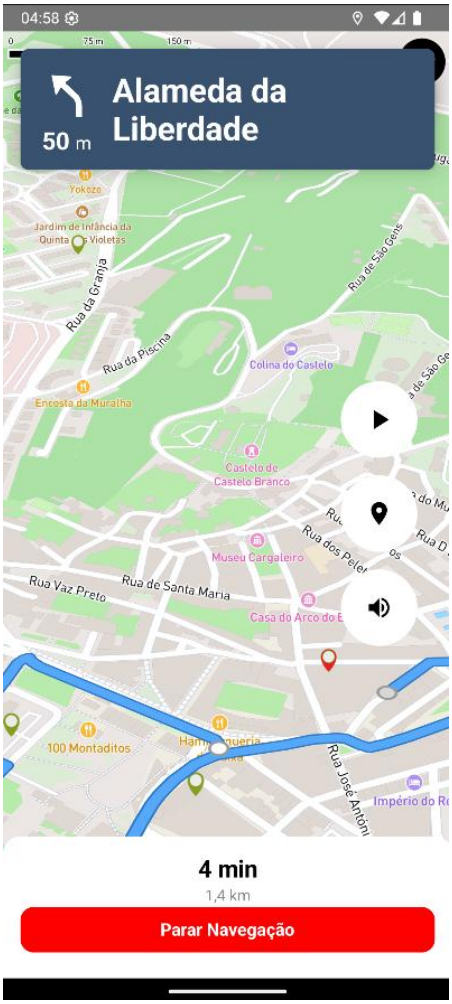

Figura 44 *- Parar Navegação.*

Figura 45 *- Navegação cancelada.*

6.3.5. Chegada ao destino

Quando o utilizador chega ao destino, a navegação é automaticamente concluída e o sistema retorna ao estado inicial do mapa. É então exibida uma mensagem de confirmação - "Chegou ao destino!" (**Figura 46**).

Figura 46 *- Chegada ao destino.*

6.3.6. Ícones de lugares

Os lugares de estacionamento são representados no mapa através de ícones coloridos, que indicam visualmente o seu estado de disponibilidade. Quando o utilizador clica num ícone, é exibida uma caixa informativa na parte inferior da interface com detalhes relevantes e ações possíveis, consoante o estado do lugar.

Lugar livre (Ícone verde) (Figura 47):

- É apresentada uma caixa informativa com o estado do lugar;
- Distância aproximada desde a localização atual do utilizador até ao lugar;
- Botão "Iniciar Navegação"

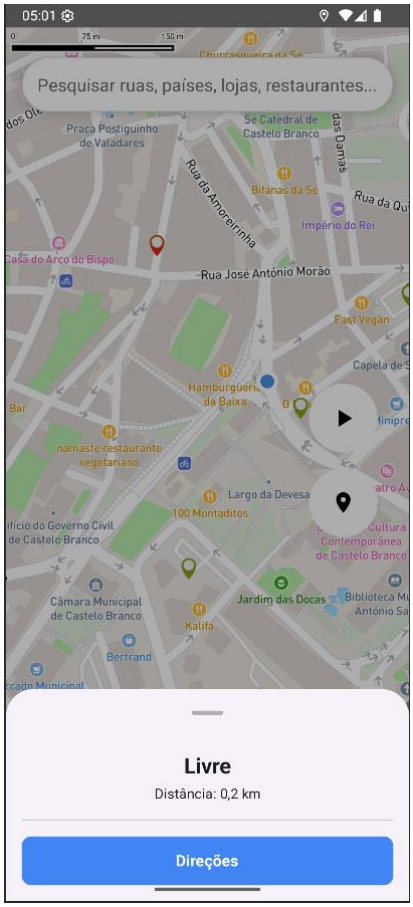

Figura 47 *- Lugar livre selecionada.*

Ao pressionar este botão, o sistema calcula e inicia automaticamente a rota desde a posição atual do utilizador até ao lugar de estacionamento selecionado (**Figura 48**).

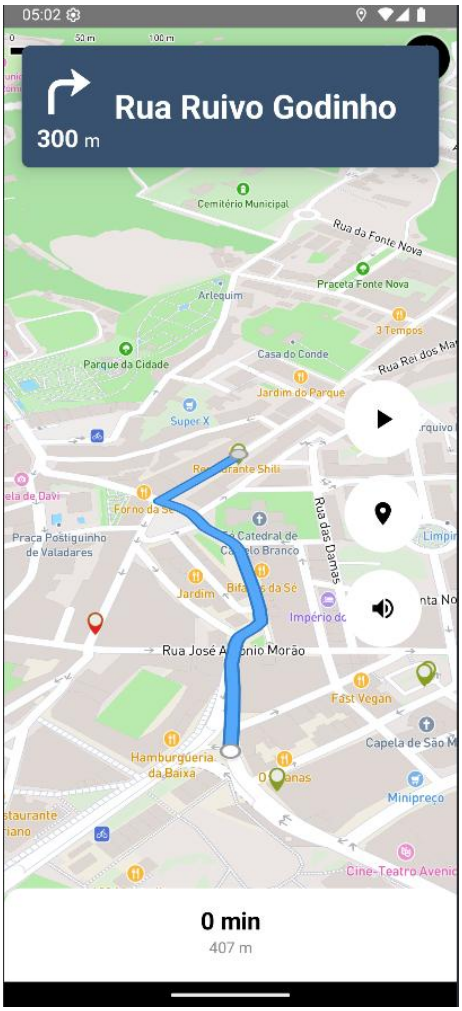

Figura 48 *- Rota iniciada até ao lugar.*

Lugar ocupado (Ícone vermelho) (Figura 49):

- É apresentada uma caixa informativa com indicação clara de "Lugar Indisponível";
- Mensagem a recomendar que o utilizador "Selecione outro lugar disponível";
- Botão "OK".

Ao pressionar este botão, a caixa informativa é fechada, permitindo ao utilizador continuar a explorar o mapa sem interrupções.

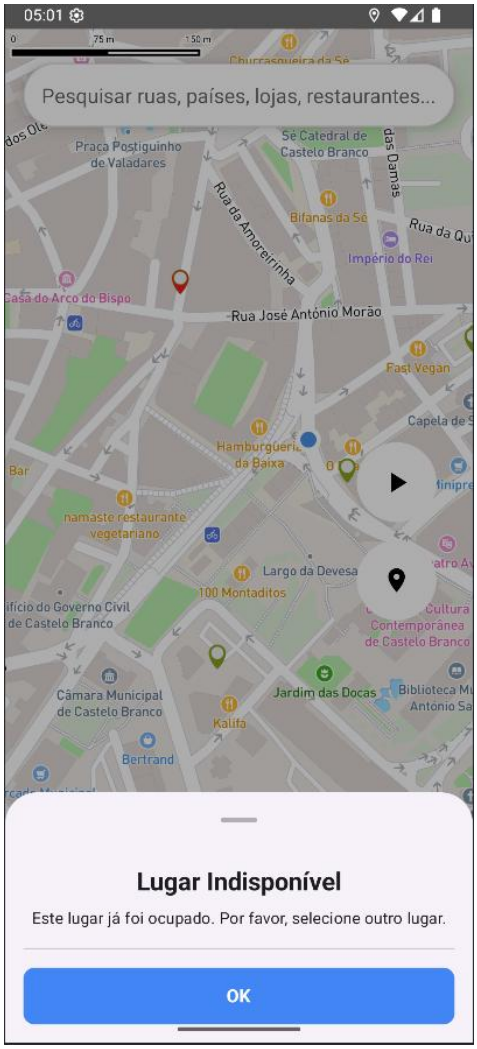

Figura 49 *- Lugar ocupado selecionado.*

7. Integrações

As integrações são a parte central da arquitetura do sistema, pois fazem a ligação entre os diferentes componentes que formam a aplicação. Neste capítulo, explicamos como as duas principais integrações, a base de dados e o modelo de computação visual, funcionam em conjunto para garantir uma experiência fácil e eficiente para os utilizadores.

7.1. Base de dados

A integração da base de dados é o ponto central que assegura a sincronização em tempo real da aplicação. Esta integração foi criada para garantir que as informações sobre os lugares de estacionamento sejam partilhadas instantaneamente entre todos os componentes do sistema.

7.1.1. Padrão repository e listener em tempo real

A integração com o Firestore segue o padrão *Repository*, que concentra toda a lógica de acesso aos dados numa camada dedicada. Esta abordagem garante uma separação nítida entre a lógica de negócio da aplicação e as interações com a base de dados, tornando facilitando a manutenção e possíveis alterações futuras

Na nossa aplicação utilizamos uma classe repositório para todas as operações relacionadas com os dados dos lugares de estacionamento. Esta classe estabelece uma ligação direta com a coleção "vagas" no Firestore, implementando um sistema de *Listener*, escuta em tempo real. Este mecanismo permite que qualquer alteração na base de dados seja imediatamente detetada e transmitida para a aplicação Android.

Graças a este mecanismo, a aplicação mantém sempre a informação atualizada sobre os lugares, sem a necessidade de realizar consultas constantes à base de dados.

Esta abordagem oferece várias vantagens significativas. Em primeiro lugar, reduz drasticamente a latência entre a deteção de uma alteração e a sua visualização na aplicação. Em segundo lugar, minimiza o consumo de dados, uma vez que apenas as alterações são transmitidas, não sendo necessário transferir periodicamente toda a informação.

7.1.2. Estado de conectividade (Online/Offline)

Reconhecendo que os utilizadores podem experienciar intermitências na conectividade de rede, o sistema implementa uma gestão robusta dos diferentes estados de conectividade. A aplicação monitoriza continuamente o estado da ligação à internet através de um *BroadcastReceiver* que escuta as alterações de conectividade do sistema Android.

Quando a aplicação deteta uma perda de conectividade, ativa automaticamente o modo de funcionamento offline. Neste modo, a aplicação utiliza os dados em cache disponibilizados pelo Firestore, permitindo ao utilizador continuar a visualizar as informações dos lugares de estacionamento, ainda que estas possam não refletir o estado mais atual.

Quando a conectividade é restaurada, o sistema executa automaticamente uma reconciliação dos dados, sincronizando todas as alterações que possam ter ocorrido durante o período offline. Esta funcionalidade é particularmente importante numa aplicação de navegação, onde a perda de conectividade é uma ocorrência comum.

7.1.3. Otimização de performance

Para garantir uma performance otimizada, a integração com o Firestore implementa várias estratégias de otimização. O sistema utiliza índices automáticos criados pelo Firestore para acelerar as consultas.

Além disso, a aplicação implementa um sistema de *cache* inteligente que armazena localmente as informações mais relevantes, reduzindo a necessidade de consultas repetidas à base de dados. Este *cache* é atualizado automaticamente sempre que novas informações são recebidas através dos *listeners* em tempo real.

7.2. Modelo de computação visual integrado

A integração do modelo de computação visual combina algoritmos avançados de visão computacional com tecnologias de *cloud computing* para criar um sistema automatizado de monitorização.

7.2.1. Processamento seletivo de frames e redimensionamento

O sistema processa continuamente *streams* de vídeo provenientes de câmaras posicionadas estrategicamente sobre os lugares de estacionamento. Para otimizar a performance computacional, o sistema implementa várias estratégias de otimização.

Uma das otimizações mais importantes é o processamento seletivo de *frames*. Em vez de processar todos os *frames* do vídeo, o sistema analisa apenas um em cada três *frames*, reduzindo significativamente a carga computacional sem comprometer a precisão da deteção. Esta abordagem é possível porque as alterações no estado de ocupação dos lugares de estacionamento ocorrem geralmente de forma relativamente lenta, não havendo necessidade um processamento *frame* a *frame*.

O sistema também implementa redimensionamento automático das imagens para uma resolução padronizada, garantindo consistência no processamento

independentemente da qualidade original do vídeo. Esta normalização é importante para manter a precisão do modelo YOLO, que foi treinado com imagens de dimensões específicas, além de reduzir o consumo de recursos e melhorar a performance em sistemas com menor capacidade de processamento.

7.2.2. Filtragem e classificação inteligente

Para maximizar a precisão do sistema, implementou-se um sofisticado sistema de filtragem que elimina deteções irrelevantes. O algoritmo YOLO é capaz de detetar múltiplas classes de objetos, incluindo pessoas, bicicletas, animais, entre outros. No contexto desta aplicação, apenas a deteção de veículos é relevante para determinar a ocupação dos lugares de estacionamento.

O sistema implementa uma lista de classes excluídas que filtra automaticamente deteções de peões, ciclistas e outros objetos que não sejam veículos. Esta filtragem reduz significativamente o número de falsos positivos, garantindo que apenas a presença de veículos seja considerada para determinar a ocupação de um lugar.

7.2.3. Filtragem de classes e delimitação de áreas

Uma das características do sistema é a capacidade de mapear geometricamente os lugares de estacionamento através de polígonos definidos interativamente. O sistema permite definir as áreas correspondentes a cada lugar de estacionamento desenhando polígonos diretamente sobre a imagem do vídeo.

Cada polígono corresponde a um lugar específico e está associado a um identificador único na base de dados. Quando o sistema deteta um veículo, calcula se o centro geométrico do veículo se encontra dentro dos limites de algum dos polígonos definidos. Se esta condição se verificar, o lugar correspondente é marcado como ocupado na base de dados.

Esta abordagem geométrica oferece grande flexibilidade, permitindo ao sistema adaptar-se a diferentes configurações de estacionamento, desde lugares perpendiculares tradicionais a configurações mais complexas como lugares diagonais ou de formas irregulares.

A **Figura 50** representa a visão da câmara em tempo real, onde cada lugar de estacionamento é delimitado por um polígono. Quando o centro geométrico de um objeto detetado entra nessa área, o seu estado é atualizado na base de dados. A imagem foi extraída de um vídeo previamente gravado com o drone do professor no parque de estacionamento do Intermarché em Castelo Branco.

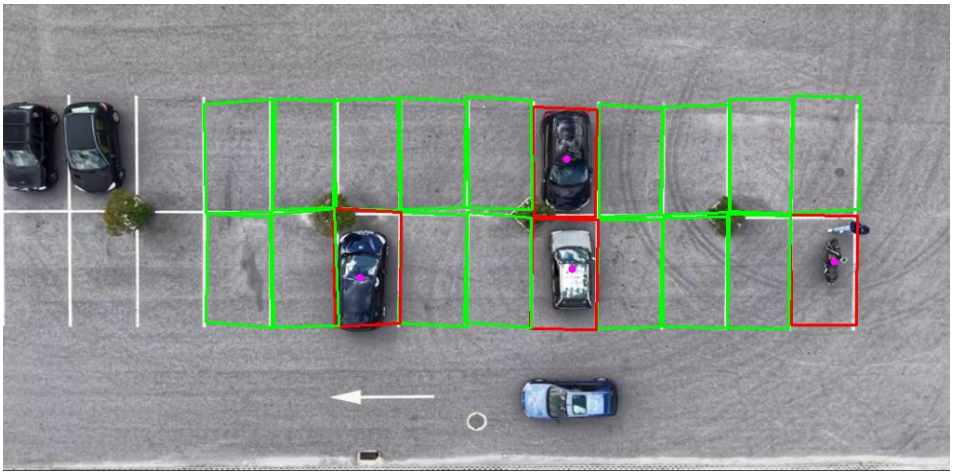

Figura 50 - Visão da câmara.

7.2.4. Polígonos e persistência de configuração

O sistema implementa mecanismos robustos de persistência que garantem que as configurações dos polígonos são mantidas entre diferentes sessões de funcionamento. As coordenadas dos polígonos são armazenadas em ficheiros JSON locais, permitindo que o sistema mantenha a sua configuração mesmo após reinicializações.

Esta persistência é complementada por funcionalidades de edição interativa que permitem adicionar, remover os polígonos conforme necessário. Esta flexibilidade é fundamental para ajustar sempre que houver mudanças na disposição dos lugares de estacionamento ou para corrigir erros na configuração inicial das áreas.

7.2.5. Integração com firebase em tempo real

A integração entre o sistema de visão computacional e o Firebase constitui o elemento que fecha o ciclo de funcionamento do sistema completo. Sempre que o algoritmo de visão deteta uma alteração no estado de ocupação de um lugar, essa informação é imediatamente transmitida para o Firestore através de uma operação de atualização.

Esta integração é implementada através de um mapeamento direto entre os polígonos definidos no sistema de visão e os documentos correspondentes na coleção "vagas" do Firestore. Cada polígono está associado a um identificador único de documento, garantindo que as atualizações são aplicadas no lugar correto.

O sistema implementa também mecanismos de tolerância a falhas de conectividade fazendo com que não comprometem o funcionamento do sistema. Em caso de perda de ligação com o Firebase, o sistema mantém um registo local das alterações detetadas e sincroniza-as automaticamente quando a ligação é restaurada.

8. Testes

Os testes realizados no âmbito do desenvolvimento da aplicação tiveram como objetivo validar o correto funcionamento das funcionalidades, avaliar a experiência do utilizador e testar o desempenho do sistema. Este capítulo descreve a metodologia utilizada, os cenários testados e os resultados obtidos.

8.1. Testes funcionais

Os testes funcionais foram realizados de forma a verificar o bom funcionamento da aplicação. Foram realizados manualmente e através de testes automatizados, abrangendo os principais fluxos de utilização. Incluímos igualmente testes num dispositivo android físico, para validar o comportamento da aplicação em condições reais de utilização, além dos testes executados no emulador.

8.1.1. Cenários testados:

- Pesquisa de localidades através da barra de pesquisa;
- Exibição de sugestões dinâmicas durante a pesquisa;
- Seleção de um lugar livre e início de navegação;
- Tentar iniciar navegação para um lugar Ocupado;
- Cancelamento de rota através do gesto de arrastar a caixa de informações;
- Ativação/desativação do reencaminhamento automático;
- Funcionamento do assistente de voz durante a navegação;
- Comportamento da aplicação em modo *offline*;
- Sincronização de dados após restabelecimento da conectividade;
- Verificar se o botão de recentrar funciona corretamente em modo de navegação e em modo normal;
- Teste de compatibilidade com Android Auto;
- Verificar a atualização do estado dos lugares com o modelo de computação visual;
- Verificar se o utilizador é redirecionado para um novo lugar depois do seu ficar ocupado. (redirecionamento ligado);
- Verificar se o utilizador é redirecionado para um novo lugar depois do seu ficar ocupado. (redirecionamento desligado);

Resultados:

Todos os cenários testados funcionaram conforme esperado há exceção do teste de compatibilidade com Android Auto que não reconhecia a nossa aplicação.

A aplicação respondeu corretamente a interações do utilizador e manteve a consistência dos dados em tempo real. Em modo *offline*, a aplicação manteve funcionalidades básicas, como a visualização do mapa e lugares em cache, e sincronizou os dados automaticamente quando a ligação foi restabelecida

9. Conclusão

O desenvolvimento da aplicação ParkFinder representa uma solução inovadora e robusta para um problema social significativo: a dificuldade enfrentada por pessoas com mobilidade reduzida na localização e acesso a lugares de estacionamento reservados. Ao longo deste projeto, foi criado um sistema integrado que combina tecnologias de ponta em visão computacional, desenvolvimento móvel e bases de dados em tempo real.

Os objetivos estabelecidos foram amplamente cumpridos. No Projeto I, definiu-se uma arquitetura sólida baseada no padrão MVC, implementou-se o modelo de dados no Firebase Firestore, realizou-se uma análise comparativa que levou à escolha do Mapbox como API de mapas e implementou-se a visualização interativa dos lugares diretamente sobre o mapa. Além disso, criaram-se diagramas de casos de uso e esboços de interface que orientaram todo o desenvolvimento subsequente. No Projeto II, a aplicação móvel foi desenvolvida em Kotlin com Jetpack Compose e inclui navegação em tempo real com instruções por voz, cálculo dinâmico de rotas que se ajustam conforme o estado dos lugares, redirecionamento automático para o lugar livre mais próxima quando o selecionado fica ocupado e sincronização offline robusta, mantendo dados em cache no dispositivo e atualizando-os automaticamente ao restabelecer a conexão. O modelo de visão computacional YOLO foi treinado e otimizado, alcançando métricas de precisão superiores a 50% em mAP50, demonstrando eficácia na deteção automática de veículos.

Os testes funcionais validaram a estabilidade e fiabilidade do sistema em diversos cenários operacionais, confirmando comportamento consistente em conectividade intermitente, com cache local e sincronização automática ao restabelecer ligação. O modelo de visão, otimizado por filtragem de classes e processamento seletivo de frames, processa streams de vídeo em tempo real com precisão adequada, e a configuração de polígonos adaptáveis permite cobertura de estacionamento variados.

Com o projeto já concluído e em pleno funcionamento, gostariamos que, no futuro, o ParkFinder seja implementado na prática para apoiar, as pessoas com mobilidade reduzida na localização de lugares de estacionamento reservados.

9.1. Desafios e limitações

Durante o desenvolvimento da aplicação, foram identificados diversos desafios e limitações que influenciaram as escolhas técnicas e metodológicas.

A recolha inicial de imagens por drone revelou-se insuficiente em termos de quantidade e diversidade, exigindo o recurso a datasets públicos como o VisDrone2021 para garantir uma amostragem adequada de cenários urbanos e rurais.

A integração com o Android Auto revelou-se inviável, uma vez que a nossa aplicação não era reconhecida pelo sistema devido aos requisitos específicos exigidos pela Google, e a documentação do Mapbox para suporte ao Android Auto ainda é muito limitada.

Além disso, o aplicativo só é compatível com Android 10 ou versões superiores e não oferece suporte a dispositivos iOS, restringindo o seu alcance aos utilizadores.

Bibliografia

[1] “About – Park4Dis.” Acesso em: setembro de 2024. Disponível em: https://www.park4dis.org/about/?lang=en

[2] “Difference between MVC and MVVM Architecture Pattern in Android – GeeksforGeeks.” Acesso em: outubro de 2024. Disponível em: https://www.geeksforgeeks.org/android/difference-between-mvc-and-mvvm-architecture-pattern-in-android/

[3] L. Barbosa, “Design Patterns: as diferenças entre MVC, MVVM e MVP – Medium.” Acesso em: outubro de 2024. Disponível em: https://medium.com/@luiselbarbosa/design-patterns-as-diferen%C3%A7as-entre-mvc-mvvm-e-mvp-28b56de8698

[4] “MVC – Wikipédia.” Acesso em: outubro de 2024. Disponível em: https://pt.wikipedia.org/wiki/MVC

[5] “Introdução ao padrão MVC – DevMedia.” Acesso em: outubro de 2024. Disponível em: https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308

[6] “Difference Between MVC and MVP – Ask Any Difference.” Acesso em: outubro de 2024. Disponível em: https://askanydifference.com/pt/difference-between-mvc-and-mvp/

[7] “Model–View–Presenter – Wikipedia.” Acesso em: outubro de 2024. Disponível em: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter

[8] “What are MVP and MVC and what is the difference? – Stack Overflow.” Acesso em: outubro de 2024. Disponível em: https://stackoverflow.com/questions/2056/what-are-mvp-and-mvc-and-what-is-the-difference

[9] “What is an API? – AWS.” Acesso em: setembro de 2024. Disponível em: https://aws.amazon.com/what-is/api/

[10] “APIs – IBM Think.” Acesso em: setembro de 2024. Disponível em: https://www.ibm.com/think/topics/api

[11] “What is an API? – Postman.” Acesso em: setembro de 2024. Disponível em: https://www.postman.com/what-is-an-api/

[12] “How do APIs work? – Akamai.” Acesso em: setembro de 2024. Disponível em: https://www.akamai.com/glossary/how-do-apis-work

[13] “What is an API? – GeeksforGeeks.” Acesso em: setembro de 2024. Disponível em: https://www.geeksforgeeks.org/software-testing/what-is-an-api/

[14] “Web Service – OpenSoft.” Acesso em: setembro de 2024. Disponível em: https://opensoft.pt/2016/06/07/web-service/

[15] “Webservice: o que é? – Smartlinks.” Acesso em: setembro de 2024. Disponível em: https://smartlinks.pt/webservice-o-que-e/

[16] “API Guides – Mapbox Docs.” Acesso em: setembro de 2024. Disponível em: https://docs.mapbox.com/api/guides/

[17] “API to Google Maps – Maps Blog.” Acesso em: outubro de 2024. Disponível em: https://maps-blog.com/glossario/api-to-google-maps/

[18] “Como usar a Maps API do Google Maps no site da sua empresa – Geoambiente.” Acesso em: outubro de 2024. Disponível em: https://www.geoambiente.com.br/blog/como-usar-a-maps-api-do-google-maps-no-site-da-sua-empresa/

[19] “Google Maps Products – Google Maps Platform.” Acesso em: outubro de 2024. Disponível em: https://mapsplatform.google.com/maps-products/

[20] “Pricing – Google Maps Platform.” Acesso em: outubro de 2024. Disponível em: https://mapsplatform.google.com/pricing/?utm_referrer=https%3A%2F%2Fwww.bing.com%2F#pricing-calculator

[21] “API Guides – Mapbox Docs.” Acesso em: outubro de 2024. Disponível em: https://docs.mapbox.com/api/guides/

[22] “Pricing – Mapbox.” Acesso em: outubro de 2024. Disponível em: https://www.mapbox.com/pricing

[23] “Cloud Firestore Documentation – Firebase.” Acesso em: outubro de 2024. Disponível em: https://firebase.google.com/docs/firestore?hl=pt-pt

[24] “Data Model – Firestore Documentation – Firebase.” Acesso em: outubro de 2024. Disponível em: https://firebase.google.com/docs/firestore/data-model?hl=pt

[25] “Introduction to Android Studio – Android Developers.” Acesso em: setembro de 2024. Disponível em: https://developer.android.com/studio/intro?hl=pt-br

[26] “Overview of Android Studio – GeeksforGeeks.” Acesso em: setembro de 2024. Disponível em: https://www.geeksforgeeks.org/android/overview-of-android-studio/

[27] “About GitHub and Git – GitHub Docs.” Acesso em: outubro de 2024. Disponível em: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

[28] “What is GitHub and How to Use It? – GeeksforGeeks.” Acesso em: outubro de 2024. Disponível em: https://www.geeksforgeeks.org/git/what-is-github-and-how-to-use-it/

[29] “Use Case Diagram – GeeksforGeeks.” Acesso em: novembro de 2024. Disponível em: https://www.geeksforgeeks.org/system-design/use-case-diagram/

[30] “What is Use Case Diagram? – Visual Paradigm.” Acesso em: novembro de 2024. Disponível em: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

[31] “UML Use Case Diagram Tutorial – Lucidchart.” Acesso em: novembro de 2024. Disponível em: https://www.lucidchart.com/pages/tutorial/uml-use-case-diagram

[32] “Criar um diagrama de casos de utilização UML – Microsoft Support.” Acesso em: novembro de 2024. Disponível em: https://support.microsoft.com/pt-pt/topic/criar-um-diagrama-de-casos-de-utiliza%C3%A7%C3%A3o-uml-92cc948d-fc74-466c-9457-e82d62ee1298

[33] “Make a UML Use Case Diagram – ProcessOn.” Acesso em: novembro de 2024. Disponível em: https://www.processon.io/pt/blog/make-a-uml-use-case-diagram

[34] “Introduction of ER Model – GeeksforGeeks.” Acesso em: outubro de 2024. Disponível em: http://geeksforgeeks.org/dbms/introduction-of-er-model/

[35] “VisDrone 2021 – AIskyeye.” Acesso em: junho de 2025. Disponível em: https://aiskyeye.com/visdrone-2021/

[36] “Datasets – Ultralytics Hub.” Acesso em: junho de 2025. Disponível em: https://hub.ultralytics.com/datasets/1VdHnyICxDpBebII3vfC

[37] “YOLO11 Overview – Ultralytics Docs.” Acesso em: junho de 2025. Disponível em: https://docs.ultralytics.com/pt/models/yolo11/#overview

[38] “YOLOv8 Overview – Ultralytics Docs.” Acesso em: junho de 2025. Disponível em: https://docs.ultralytics.com/pt/models/yolov8/#overview

[39] “Model Training Tips – Ultralytics Docs.” Acesso em: junho de 2025. Disponível em: https://docs.ultralytics.com/pt/guides/model-training-tips/#choosing-between-cloud-and-local-training

[40] “Kotlin Programming Language – Contrast Security.” Acesso em: outubro de 2024. Disponível em: https://www.contrastsecurity.com/glossary/kotlin-programming-language

[41] “Kotlin Introduction – W3Schools.” Acesso em: outubro de 2024. Disponível em: https://www.w3schools.com/kotlin/kotlin_intro.php

[42] “Introduction to Kotlin – GeeksforGeeks.” Acesso em: outubro de 2024. Disponível em: https://www.geeksforgeeks.org/kotlin/introduction-to-kotlin/

[43] “Jetpack Compose Documentation – Android Developers.” Acesso em: outubro de 2024. Disponível em: https://developer.android.com/develop/ui/compose/documentation?hl=pt-br

[44] “Understanding Jetpack Compose (Part 1 of 2) – Medium Android Developers.” Acesso em: outubro de 2024. Disponível em:

https://medium.com/androiddevelopers/understanding-jetpack-compose-part-1-of-2-ca316fe39050