INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA CAMPUS ARIQUEMES
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ISAIAS FÉLIX MACHADO DE ALCANTARA

APLICATIVO PARA GERENCIAMENTO INTELIGENTE DE LISTAS DE COMPRAS COM FLUTTER E FIREBASE

Ariquemes
2025

ISAIAS FÉLIX MACHADO DE ALCANTARA

APLICATIVO PARA GERENCIAMENTO INTELIGENTE DE LISTAS DE COMPRAS COM FLUTTER E FIREBASE

Ariquemes
2025

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO.

A347a

Alcantara, Isaias Félix Machado de.

Aplicação Web/Mobile para gestão e análise de listas de compras / Isaias Félix Machado de Alcantara. - Ariquemes, 2025.

37 f. : il.

Orientador(a): Andrey Alencar Quadros.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Ariquemes, 2025.

1. flutter. 2. firebase. 3. aplicativo multiplataforma. 4. lista de compras. 5. colaboração em tempo real. I. Quadros, Andrey Alencar (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

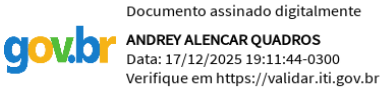
Bibliotecário(a) Responsável: Renilce Silva Morais, CRB-11/906

ISAIAS FÉLIX MACHADO DE ALCANTARA

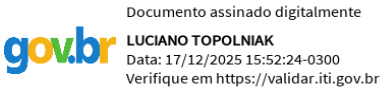
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Analista e Desenvolvedor de Sistemas" e aprovado em sua forma final pelo Curso de Tecnologia em Análise e Desenvolvimento de Sistemas.

Ariquemes, 03 de dezembro de 2025.

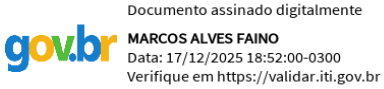
Banca Examinadora:



Prof. Mestre Andrey Alencar Quadros
(orientador)
Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO - Campus Ariquemes)



Prof. Mestre Luciano Topolniak
Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO - Campus Ariquemes)



Prof. Especialista Marcos Alves Faino
Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO - Campus Ariquemes)

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela força e sabedoria concedidas ao longo desta jornada. À minha família, pelo apoio incondicional, paciência e incentivo constantes. Ao meu orientador, Prof. Andrey Alencar Quadros, pela orientação precisa, disponibilidade e pelas valiosas contribuições que tornaram este trabalho possível.

RESUMO

O presente trabalho apresenta o desenvolvimento de um aplicativo multiplataforma para gerenciamento inteligente de listas de compras, implementado com o *framework* Flutter e o conjunto de serviços Firebase. A proposta busca oferecer uma solução prática e colaborativa para o controle de compras domésticas, permitindo que múltiplos usuários criem, editem e compartilhem listas em tempo real, otimizando tanto o processo de aquisição de produtos quanto o acompanhamento de gastos familiares. Para atingir esse objetivo, foi adotada uma abordagem de desenvolvimento ágil, com autenticação via Firebase *Authentication*, armazenamento em *Cloud* Firestore e gráficos dinâmicos para análise de consumo. O resultado é um sistema funcional, escalável e reativo, com sincronização em tempo real e experiência de uso consistente em Android.

Palavras-chave: Flutter. Firebase. Aplicativo multiplataforma. Lista de compras. Colaboração em tempo real.

ABSTRACT

This work presents the development of a multiplatform application for intelligent shopping list management, implemented using the Flutter framework and the Firebase service suite. The proposed solution aims to provide a practical and collaborative tool for household shopping control, allowing multiple users to create, edit, and share lists in real time while optimizing product acquisition and family expense tracking. An agile development approach was adopted, integrating Firebase Authentication for user login, Cloud Firestore for data storage, and dynamic charts for consumption analysis. The resulting system is functional, scalable, and reactive, offering real-time synchronization and a consistent user experience across Android devices.

Keywords: Flutter. Firebase. Cross-platform application. Shopping list. Real-time collaboration.

LISTA DE FIGURAS

Figura 1 – Fluxo de autenticação do aplicativo. 25
Figura 2 – Home page. 26
Figura 3 – Fluxo de listas. 27
Figura 4 – Fluxo de gerenciamento de membros. 28
Figura 5 – Fluxo de produtos. 29
Figura 6 – Fluxo de histórico/análise. 30
Figura 7 – Aplicativo publicado na Google Play Store. 35

LISTA DE ABREVIATURAS E SIGLAS

MVC	Model-View-Controller
MVVM	Model-View-ViewModel
MVP	Minimum Viable Product
NoSQL	Not Only SQL
UID	User Identifier

GLOSSÁRIO

Android Sistema operacional móvel desenvolvido pelo Google

BuildContext Objeto do Flutter que referencia a posição de um widget na árvore de interface

Cloud Modelo de computação em nuvem para fornecimento de recursos via internet

Desktop Computador pessoal utilizado para execução de sistemas operacionais completos

Drawer Menu lateral deslizante utilizado para navegação em aplicações móveis

Feature Funcionalidade específica de um sistema ou aplicação

Firebase Plataforma de serviços em nuvem do Google para desenvolvimento de aplicações

Firestore Banco de dados NoSQL em tempo real do Firebase

Flutter Framework multiplataforma para desenvolvimento de interfaces nativas

Framework Estrutura reutilizável de código que auxilia o desenvolvimento de software

GetX Framework para gerenciamento de estado, navegação e injeção de dependência no Flutter

GitHub Plataforma de hospedagem e controle de versão de código-fonte

Listeners Componentes que monitoram eventos ou alterações em tempo real

Material Design Linguagem de design criada pelo Google para interfaces digitais

offline-first Estratégia de desenvolvimento que prioriza o funcionamento sem conexão com a internet

Push Mensagem enviada de um servidor para o dispositivo do usuário

Scrum Metodologia ágil baseada em ciclos iterativos e incrementais

Sprints Períodos curtos e fixos de desenvolvimento no Scrum

Stack Navigation Modelo de navegação baseado em pilha de telas

Streams Fluxos contínuos de dados processados de forma assíncrona

Time-to-Market Tempo necessário para disponibilizar um produto ao mercado

Web World Wide Web

SUMÁRIO

Lista de Abreviaturas e Siglas **8**
1 INTRODUÇÃO **12**
2 OBJETIVOS **13**
2.1 OBJETIVO GERAL 13
2.2 OBJETIVOS ESPECÍFICOS 13
3 FUNDAMENTAÇÃO TEÓRICA **14**
3.1 ENGENHARIA DE SOFTWARE COMO BASE PARA SISTEMAS EVOLUTIVOS 14
3.2 ARQUITETURA DE SOFTWARE E ARQUITETURA LIMPA . . . 14
3.3 PADRÕES ARQUITETURAIS MVC E MVVM 14
3.4 METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO INCREMENTAL 15
3.5 USABILIDADE E HEURÍSTICAS DE NIELSEN 15
3.6 DESENVOLVIMENTO MULTIPLATAFORMA E PARADIGMAS REATIVOS 15
3.7 COMPUTAÇÃO EM NUVEM E PERSISTÊNCIA DE DADOS . . . 16
3.8 QUALIDADE DE SOFTWARE E CRITÉRIOS NÃO FUNCIONAIS 16
3.9 SÍNTESE DA FUNDAMENTAÇÃO TEÓRICA 16
4 METODOLOGIA **17**
4.1 TIPO DE PESQUISA E ABORDAGEM METODOLÓGICA 17
4.2 MÉTODO E JUSTIFICATIVA 17
4.3 PROCESSO METODOLÓGICO 18
4.4 ARQUITETURA E JUSTIFICATIVA TECNOLÓGICA 18
4.5 PROTOCOLO DE TESTES 19
4.6 CRITÉRIOS DE AVALIAÇÃO 19
4.7 VALIDAÇÃO COM USUÁRIOS 19
4.8 CONSIDERAÇÕES FINAIS DA METODOLOGIA 20
5 DESENVOLVIMENTO **21**
5.1 APOIO AO DESENVOLVIMENTO 21
5.1.1 Camadas e responsabilidades **21**
5.1.1.1 Camada de apresentação (Presentation) *21*
5.1.1.2 Camada de domínio (Domain) *22*
5.1.1.3 Camada de dados (Data) *22*
5.1.2 Síntese do fluxo de dependência **22**
5.2 MODELAGEM DE DADOS 22
5.2.1 Relação entre entidades **23**
5.2.2 Uso de subcoleções **23**
5.2.3 Síntese da abordagem adotada **23**

5.3 FUNCIONALIDADES-CHAVE 24
5.3.1 Autenticação e Gestão de Perfil 24
5.3.2 Tela Principal (*Home Page*) 26
5.3.3 Gestão de Listas de Compras 26
5.3.4 Colaboração em Tempo Real 28
5.3.5 Catálogo de Produtos Personalizado 28
5.3.6 Histórico e Análise de Gastos 29
5.4 USABILIDADE E NAVEGAÇÃO 30
5.4.1 Tecnologia de Navegação: *GetX* 30
5.4.2 Padrões de Navegação do Usuário 31
5.4.3 Síntese de Usabilidade 31
5.5 GERENCIAMENTO DE ERROS E SEGURANÇA 31
5.5.1 Tratamento de Erros 31
5.5.2 Estratégia de Segurança 32
6 RESULTADOS E DISCUSSÕES 33
6.1 SÍNTESE DOS RESULTADOS FUNCIONAIS 33
6.2 DESEMPENHO E ESTABILIDADE 33
6.3 USABILIDADE E ADERÊNCIA 33
6.4 ANÁLISE CRÍTICA 34
6.5 IMPLICAÇÕES PRÁTICAS 34
6.6 DISPONIBILIZAÇÃO DO APLICATIVO NA PLAY STORE 34
7 CONCLUSÃO E CONTRIBUIÇÕES DO TRABALHO 36
7.1 CONTRIBUIÇÕES TÉCNICAS 36
7.2 TRABALHOS FUTUROS 36
7.2.1 Considerações sobre Escalabilidade Futura 36
REFERÊNCIAS 37

1 INTRODUÇÃO

A rotina contemporânea tem dificultado o planejamento e o gerenciamento de tarefas domésticas, como a elaboração de listas de compras. A ausência de ferramentas eficazes torna esse processo mais complexo, podendo resultar em esquecimentos, compras desnecessárias e falta de controle de gastos.

Mesmo com a ampla oferta de aplicativos para listas de compras, muitos ainda carecem de recursos essenciais, como colaboração, sincronização em tempo real e controles de gastos, o que reduz sua eficiência no uso cotidiano.

Diante dessas limitações, este trabalho propõe o desenvolvimento de uma aplicação voltada ao gerenciamento familiar de listas de compras, oferecendo colaboração entre usuários, sincronização em tempo real e controle de gastos. A aplicação utiliza o Firebase para autenticação, armazenamento e atualização instantânea dos dados, enquanto o Flutter é empregado na construção da interface. O sistema foi desenvolvido seguindo princípios de engenharia de software.

Este projeto é relevante por atender à necessidade crescente de ferramentas digitais colaborativas e eficientes. A solução prioriza confiabilidade, manutenção, escalabilidade e experiência do usuário.

A implementação desta aplicação seguiu uma abordagem ágil, com testes de usabilidade e melhorias contínuas. O uso do GitHub garantiu controle eficiente do versionamento, permitindo gerenciar cada etapa de forma incremental e assegurando organização, estabilidade e foco total na experiência do usuário.

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta os objetivos; o Capítulo 3 discute a base teórica; o Capítulo 4 detalha a metodologia; o Capítulo 5 descreve a implementação; o Capítulo 6 apresenta os resultados; e o Capítulo 7 traz as conclusões.

2 OBJETIVOS

Este capítulo apresenta os objetivos deste trabalho, divididos em objetivo geral e objetivos específicos. Eles foram definidos para orientar o desenvolvimento de uma solução digital voltada ao gerenciamento familiar de listas de compras, considerando tanto sua relevância prática quanto sua contribuição acadêmica.

2.1 OBJETIVO GERAL

Desenvolver uma aplicação para gerenciamento familiar de listas de compras com sincronização em tempo real, organização por categorias e controle de gastos, visando facilitar o planejamento financeiro e a experiência do usuário.

2.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral proposto, o trabalho busca cumprir as seguintes metas técnicas e funcionais:

- **Desenvolvimento da Arquitetura e Estrutura de Dados** Projetar uma arquitetura modular visando à escalabilidade da aplicação e facilitando a modelagem e estruturação de dados, garantindo a manutenção e evolução futura do sistema.
- **Implementação das Funcionalidades de Usuário e Sincronização em Tempo Real** Implementar a autenticação e o gerenciamento de usuários por meio do Firebase *Authentication*, bem como a sincronização em tempo real dos dados, utilizando *Listeners* do Firestore para refletir as alterações imediatamente na interface.
- **Documentação e Prospecção de Evoluções Futuras** Documentar o processo de desenvolvimento e prospectar evoluções futuras do sistema, incluindo recursos como notificações *Push* e sincronização *offline-first*.

O conjunto desses objetivos visa não apenas a entrega de um produto funcional, mas também a criação de uma solução que possa servir como referência para futuras aplicações colaborativas.

3 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo discute sobre os principais referenciais teóricos da Engenharia de *Software* que fundamentam as decisões no desenvolvimento da aplicação. O foco é sobre a análise crítica de arquiteturas, padrões e metodologias, relacionando-os diretamente às necessidade de sistemas colaborativos, multiplataforma e orientados à experiencia do usuário (Sommerville, 2011).

3.1 ENGENHARIA DE SOFTWARE COMO BASE PARA SISTEMAS EVOLUTIVOS

A Engenharia de *Software* pode ser compreendia como campo orientado à tomada de decisões técnicas com restrições de prazo, custo e qualidade. Conforme argumenta Sommerville (2011), seu papel central não está apenas na produção de *software*, mas também na garantia de que sistemas possam evoluir de forma controlada.

Nesse contexto, aspectos como manutenção, testes e confiabilidade tornam-se critérios determinantes para aplicações colaborativas, nas quais múltiplos usuários interagem e alterações são frequente. Assim, processos estruturados e métricas de qualidade é um meio para reduzir riscos técnicos e aumentar a previsibilidade do sistema

3.2 ARQUITETURA DE SOFTWARE E ARQUITETURA LIMPA

A arquitetura de *software* exerce influência direta sobre a capacidade de evolução de um sistema. A proposta de Arquitetura Limpa, comentada por Martin (2017), enfatiza o desacoplamento entre interface, regras de negócio e mecanismos de persistência, buscando minimizar dependências externas.

Diferentemente de abordagens fortemente acopladas a Framework, a Arquitetura Limpa tem como prioridade a independência tecnológica, permitindo mudanças em camadas como interface não causem impacto em uma outra camada da aplicação. Essa particularidade é muito importante em projetos grandes, nos quais a longevidade do sistema dependente sempre da adaptabilidade de novos serviços. Além da facilidade de mexer em camadas separadas, a separação clara de responsabilidades favorece nos testes e manutenção, reduzindo custos associados a evolução do sistema.

3.3 PADRÕES ARQUITETURAIS MVC E MVVM

Os padrões de arquiteturas MVC (Model-View-Controller) e MVVM são muito utilizados para facilitar a separação de responsabilidades em aplicações interativas (Gamma et al., 1994; Microsoft, 2012). Ambos buscam reduzir o acoplamento entre interface e lógica, porém adotam estratégias distintas.

O padrão MVC estabelece uma divisão clássica entre visão, controlados e modelo, sendo associado diretamente a aplicações *web* e Desktop. Já o MVVM introduz o conceiro

de *ViewModel*, que atua como uma ponte direta entre interface do usuário e o domínio, favorecendo ligações de dados e atualizações.

Em aplicações móveis modernas, especialmente as que usam base de frameworks reativos, o MVVM (Model-View-ViewModel) tende a apresentar maior facilidade de aplicar, uma fez que facilidade o gerenciamento de estado e a sincronização da interface, reduzindo a complexidade e código redundante.

3.4 METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO INCREMENTAL

Metodologias ágeis surgem como resposta às limitações de modelos prescritivos em contextos de requisitos voláteis (Beck et al., 2001). Ao priorizar entregas e colaboração contínua, essa abordagens se mostram adequadas para projetos de software orientados a validação frequente.

O *scrum*, conforme descrito por Schwaber e Sutherland (2017), organiza o desenvolvimento em ciclos curtos, favorecendo a inspeção constante. Práticas oriundas do *Extreme Programming*, como integração e desenvolvimento orientados a testes, contribuem a redução de defeitos e melhoria de qualidade (Beck, 2004). Nessa cenário, o uso de metodologias ágeis é uma estratégia para alinhar o desenvolvimento a necessidades do usuários.

3.5 USABILIDADE E HEURÍSTICAS DE NIELSEN

A usabilidade contribui para um fator crítico para o sucesso de aplicações, nas quais limitações de telas e interações exigem interfaces claras. Segundo Nielsen (1994), atributos como eficiência e satisfação do usuário devem orientar decisões de projeto.

As heurísticas propostas por Nielsen, como consistência, *feedback* imediato e prevenção de erros, favorecem critérios práticos para avaliação de interfaces. Em sistemas colaborativos, tais diretrizes contribuem para reduzir a carga cognitiva do usuário e aumentar percepção de fluxo durante a interação.

3.6 DESENVOLVIMENTO MULTIPLATAFORMA E PARADIGMAS REATIVOS

O desenvolvimento multiplataforma busca maximizar reutilização de código sem comprometer o desempenho e experiência do usuário (Sommerville, 2011). *Frameworks* como Flutter e React Native adotam abordagens distintas para esse assunto.

O Flutter utiliza um motor de renderização próprio, permitindo maior estabilidade visual e menor dependência de componentes. Em contrapartida, soluções baseadas em *bridge*, como React Native, favorecem integração com ecossistemas Web, porém introduz camadas adicionais de comunicação que podem causar impacto no desempenho. O uso de paradigmas reativos e gerência de estado contribui para interface mais alinhadas a expectativas dos usuários móveis(Bass; Clements; Kazman, 2012).

3.7 COMPUTAÇÃO EM NUVEM E PERSISTÊNCIA DE DADOS

A computação em nuvem da suporte à escalabilidade, disponibilidade e sincronização em tempo real, características essenciais para sistemas colaborativos (Armbrust et al., 2010). Plataformas como Firebase integram serviçoes de autenticação, comunicação e persistência de dados reduzindo complexidade arquitetural. O uso de bancos de dados NoSQL (Not Only SQL), como o Firestore, favorece a evolução do domínio da aplicação. Embora soluções como Mongo DB apresentam características semelhantes, a integração nativa do Firebase com aplicações móveis representa uma vantagem relevante em termos de simplicidade e produtividade (Sadalage; Fowler, 2012).

3.8 QUALIDADE DE SOFTWARE E CRITÉRIOS NÃO FUNCIONAIS

Requisitos não funcionais exercem papel central na avaliação da qualidade de um sistema. O modelo ISO/IEC 25010 organiza esses critérios em dimensões como desempenho, confiabilidade, usabilidade, segurança e manutenibilidade.

As escolhas arquiteturais e tecnológicas do projeto dialogam diretamente com essas dimensões, buscando equilibrar desempenho, facilidade de manutenção e experiência do usuário. Dessa forma, a qualidade do software emerge não apenas da implementação, mas da coerência entre teoria, arquitetura e prática.

3.9 SÍNTESE DA FUNDAMENTAÇÃO TEÓRICA

A análise dos referenciais teóricos evidencia que o projeto se apoia em princípios consolidados da Engenharia de Software, arquitetura desacoplada, metodologias ágeis e usabilidade. Essa articulação sustenta decisões técnicas coerentes com os objetivos do sistema, conferindo robustez, flexibilidade e aderência às demandas de aplicações colaborativas contemporâneas.

4 METODOLOGIA

A metodologia adotada para o desenvolvimento dessa aplicação foi estruturada para atender as necessidades do projeto visando garantir a organização, rastreabilidade e verificação contínua de todas as funcionalidades implementadas. O processo se baseou em princípios contemporâneos de engenharia de *software* e em práticas que permitiram construir, testar e evoluir o sistema de forma progressiva. As etapas do projeto foram definidas de modo a assegurar que cada decisão técnica estivesse alinhada a ideia central da aplicação final.

4.1 TIPO DE PESQUISA E ABORDAGEM METODOLÓGICA

Segundo as classificações de Gil (2019), este trabalho caracteriza-se como:

- **Pesquisa aplicada**: orientada à resolução de um problema real por meio do desenvolvimento de uma aplicação colaborativa para organização de compras em um ambiente familiar.
- **Pesquisa exploratória e descritiva**: busca compreender o comportamento dos usuários, permitindo descrever com precisão os fluxos, interações e necessidades que orientam o desenvolvimento da aplicação.
- **Abordagem qualitativa**: utilizada para interpretar como os usuários interagem com a ferramenta, possibilitando identificar ajustes e melhorias relevantes para a evolução do sistema.
- **Método empírico–experimental**: todas as etapas da aplicação são desenvolvidas, testadas e aprimoradas de forma iterativa, conforme as necessidades identificadas ao longo do processo.

A combinação entre pesquisa aplicada, abordagem qualitativa e método empírico-experimental justifica um ciclo de desenvolvimento orientado pela prática e pela observação condições necessárias para validar hipóteses de usabilidade e adequação ao contexto real de uso.

4.2 MÉTODO E JUSTIFICATIVA

O método adotado baseou-se em práticas ágeis, com ênfase no Scrum, por sua adequação a contextos em que os requisitos são instáveis e emergem do uso real (Sommerville, 2011). Conforme Schwaber e Sutherland (2017), o Scrum promove ciclos curtos de entrega, o que possibilitou o acompanhamento individual em cada etapa desenvolvida da aplicação.

A escolha do método Scrum pode ser justificada por três fatores essenciais:

1. **Desenvolvimento por etapas**: cada funcionalidade é desenvolvida individualmente, permitindo foco no que é mais importante a cada ciclo.

2. **Menor risco de perda de foco**: ao trabalhar em ciclos curtos, a atenção é mantida nas prioridades e evita dispersão.
3. **Possibilidade de testes isolados**: a implementação por etapas facilita testar cada funcionalidade separadamente, garantindo que problemas sejam identificados e corrigidos rapidamente.

A metodologia foi organizada em etapas não-lineares, refletindo a natureza evolutiva do software, conforme discutido por Beck (2004).

4.3 PROCESSO METODOLÓGICO

O desenvolvimento seguiu etapas iterativas alinhadas às práticas ágeis do Scrum, cada uma com papéis e entregáveis definidos:

1. **Planejamento**: definição do escopo, objetivos, prioridades e restrições do projeto.
2. **Elicitação de Requisitos**: observação exploratória, análise de sistemas similares e levantamento dos fluxos de atividade para documentar requisitos funcionais e não funcionais.
3. **Modelagem**: documentação dos modelos e fluxos que serviram como base para decisões das arquiteturas.
4. **Construção**: desenvolvimento incremental, organizado em Sprints de uma a uma semana e meia, permitindo a implementação separada das funcionalidades e o acompanhamento do progresso de cada etapa.
5. **Validação Iterativa**: realização de testes funcionais e de usabilidade ao final de cada *sprint*, possibilitando ajustes contínuos com base nos resultados e *feedback* obtido.
6. **Avaliação Final**: comparação entre o produto implementado e os requisitos e propostas iniciais, verificando se todas as funcionalidades planejadas foram corretamente implementadas.

4.4 ARQUITETURA E JUSTIFICATIVA TECNOLÓGICA

A fim de garantir a modularidade e testabilidade, optou-se por uma arquitetura lógica baseada no padrão MVC, complementada por um gerenciador de estado. Com essa escolha facilitou o isolamento de responsabilidades da ferramenta, tais como interface, regras de negócio e domínio. Com isso, foi capaz suportar a evolução incremental do sistema reduzindo as manutenções.

4.5 PROTOCOLO DE TESTES

Para assegurar coerência e confiabilidade, estabeleceu-se o seguinte protocolo de teste:

1. **Cenários de Teste**: baseado em tarefas reais do dia a dia relacionadas ao propósito do aplicativo, tais como criar conta, logar, recuperar senha, criar listas, adicionar itens na lista, editar lista, compartilhar lista com mais de um usuários passando permissões diferentes.
2. **Preparação do Ambiente**: os testes foram realizados de forma simples, testando exatamente a usabilidade do aplicativo, com isso o teste foi feito em um *smartphone* pessoal dos testadores.
3. **Perfil dos participantes**: para realizar os testes foram selecionados 3 participantes adultos com noções básica a intermediária em *smartphone* e envolvimento em atividades domésticas de organização de compras. Esses critérios visam assegurar afinidade com o público-alvo e refletir práticas reais.
4. **Execução**: nos testes os usuários selecionados receberam instruções sobre como executar as tarefas e com isso após a orientação ele executava a tarefa sozinho.
5. **Registro de evidencias**: após os testes foram recolhidos *feadbacks* dos usuários referente a usabilidade da ferramenta a fim de entender a opinião deles a respeito do aplicativo.
6. **Iteração**: resultados de cada ciclo foram analisados e traduzidos em correções ou ajustes priorizados para o *sprint* subsequente.

4.6 CRITÉRIOS DE AVALIAÇÃO

Os critérios adotados estão alinhados às recomendações de Nielsen (Nielsen, 1994) e às métricas de qualidade de software. Foram avaliadas:

- **Desempenho**: latência e tempo de resposta perceptível.
- **Estabilidade**: consistência entre dispositivos e ausência de falhas críticas.
- **Usabilidade**: clareza dos fluxos, carga cognitiva e facilidade de aprendizado.
- **Satisfação do Usuário**: medida por comentários espontâneos e percepção subjetiva.
- **Segurança**: análise das permissões e confiabilidade da autenticação.

4.7 VALIDAÇÃO COM USUÁRIOS

A validação foi realizada por meio de testes de usabilidade com três participantes, num desenho adequado para identificar problemas de usabilidade iniciais em um MVP,

conforme orientação teórica que sustenta que pequenas amostras detectam grande parte dos problemas mais críticos (Nielsen, 2000). Após toda explicação os usuários executaram um conjunto de tarefas sem assistência, foi registrado as dificuldades e erros que os usuários tiveram no processo. Os resultados subsidiaram as iterações subsequentes e orientaram decisões de priorização.

4.8 CONSIDERAÇÕES FINAIS DA METODOLOGIA

A combinação entre a abordagem aplicada, método empírico-experimental e práticas ágeis garantiu o processo metodológico com os objetivos do estudo. Ciclos curtos de entrega, validação interativa e análise integrada de dados permitiram construir e aprimorar a ferramenta com base nas evidências de uso, provendo subsídios suficientes para avaliar a eficácia das funcionalidades do protótipo no contexto proposto.

5 DESENVOLVIMENTO

Este capítulo descreve as decisões de projeto e o processo de implementação do aplicativo desenvolvido em Flutter com integração ao ecossistema Firebase. Em vez de apresentar um manual técnico, o objetivo é explicitar como as escolhas técnicas suportam os requisitos funcionais e não funcionais definidos na proposta, bem como relacioná-las aos fundamentos teóricos da engenharia de software (Pressman; Maxim, 2016; Sommerville, 2011; Bass; Clements; Kazman, 2012).

O desenvolvimento adotou uma abordagem incremental e iterativa, fundamentada em princípios de engenharia de software contemporânea, tais como modularidade, baixo acoplamento e responsabilidade única. Essas escolhas visam garantir manutenibilidade, testabilidade e escalabilidade do sistema, conforme discutido na literatura sobre arquiteturas orientadas a domínio e práticas ágeis (Pressman; Maxim, 2016; Martin, 2017).

5.1 APOIO AO DESENVOLVIMENTO

A estrutura do projeto foi definida para equilibrar a clareza conceitual e produtividade no desenvolvimento. A orientação principal baseou-se na adaptação da *Clean Architecture* (Martin, 2017), privilegiando a separação de responsabilidade entre as camadas e as tornando independentes. Essa configuração ajuda a reduzir a amarração entre interface do usuário e as implementações concretas de persistência, permitindo que decisões de infraestrutura (por exemplo, o uso do Firebase) possam ser alteradas com impacto mínimo sobre a lógica de negócio.

A organização adotada segue dois eixos complementares:

- **Divisão por camadas conceituais**: apresentação, domínio e dados.
- **Modularização por recurso funcional (*feature*)**: A modularização por Feature facilita coesão local (membros da equipe trabalham em um módulo com responsabilidade bem definida) e melhora a navegabilidade do código, o que é desejável em projetos com escopo evolutivo (Bass; Clements; Kazman, 2012).

5.1.1 Camadas e responsabilidades

5.1.1.1 Camada de apresentação (Presentation)

A camada de apresentação é responsável pela interação com o usuário e pela orquestração dos eventos de interface. Em termos de arquitetura, ela deverá expor apenas contratos de interação, por exemplo, entrada de dados, navegação e passar toda instrução do usuário a camada de domínio.

5.1.1.2 Camada de domínio (Domain)

A camada de domínio é onde fica as regras de negócio e onde define o comportamento da aplicação de acordo com os dados passados pelo usuários. Os controladores dessa camada solicitam as informações para camada de apresentação por meio de contratos sem nem mesmo terem um contato direto entre essas 2 camadas. Essa estratégia implementa o princípio de dependências dirigidas para dentro, reduzindo o impacto de mudanças em provedores de dados externos sobre a lógica de negócio (Martin, 2017).

5.1.1.3 Camada de dados (Data)

A camada de dados é onde centraliza as responsabilidades de persistência dos dados e também a comunicação com os serviços externos. É nessa camada que fica os modelos que deixam as informações passadas pela camada de domínio uniformizadas, as implementações dos repositórios e os provedores que realizam a integração com o Firebase. Usando esse formado favorece o desacoplamento, as implementações concretas implementam contratos definidos na camada de domínio, o que permite testes de integração controlados e eventual substituição da infraestrutura.

5.1.2 Síntese do fluxo de dependência

De modo sintético, as dependências conceituais seguem o padrão: Controller (interface) → Camada de Domínio (contrato) → Implementação de Repositório → Provedor de Persistência. Essa topologia reflete explicitamente o princípio de inversão de dependência e é coerente com as recomendações de arquitetura para sistemas de média complexidade (Bass; Clements; Kazman, 2012; Martin, 2017).

5.2 MODELAGEM DE DADOS

A modelagem de dados foi desenvolvida com base na estrutra NoSQL, utilizando o Cloud Firestore para persistência de dados. Essa escolha está totalmente alinhada com as características da aplicação, algo dinâmico, que exige atualizações em tempo real, baixa latência e flexibilidade estrutural. Em bancos de dados não relacionais, a ênfase recai na eficiência de leitura e escalabilidade horizontal, ainda que isso implique menor grau de normalização (Sadalage; Fowler, 2012).

A organização lógica do banco foi estruturada a partir de coleções que representam os domínios centrais do sistema, são elas: usuários, categorias, produtos e listas de compras, e não menos importante as subcoleções responsáveis por armazenar entidades dependentes. Esse formato segue um padrão de decomposição recomendado para bancos NoSQL orientados a documentos, em que entidades de granularidade distinta são separadas para evitar documentos extremamente grandes e para permitir escalabilidade granular.

5.2.1 Relação entre entidades

A coleção `users` concentra os dados essenciais de cada usuário autenticado. Os documentos utilizam como identificador o próprio UID (User Identifier) fornecido pelo serviço de *Firebase Authentication*, garantindo sincronização entre os módulos de autenticação e armazenamento. A separação dessa coleção em relação às demais permite isolar informações pessoais e evitar replicação desnecessária.

A coleção `categories` representa um agrupamento conceitual criado pelos próprios usuários para organizar suas listas. A modelagem reflete um relacionamento simples de autoria: cada categoria pertence a um único usuário, sendo assim somente quem criou pode alterar ou excluir a categoria relacionada, mas pode estar associada a múltiplas listas.

A coleção `products` funciona como um repositório privado mantido por cada usuário, sendo assim somente quem criou pode alterar ou excluir o produto relacionado, contendo itens recorrentes utilizados na composição das listas. Optou-se por permitir que as informações descritivas do produto fossem duplicadas nas listas de compras, estratégia conhecida como desnormalização. Essa prática é comum em sistemas NoSQL e reduz a necessidade de consultas adicionais, minimizando latência em operações de exibição.

A entidade central da aplicação é representada pela coleção `shopping_lists`. Cada lista armazena informações gerais — como proprietário, categoria e status — e ainda mantém estruturas que possibilitam colaboração entre usuários, como listas de membros e permissões. Esse modelo segue princípios de controle de acesso distribuído e facilita o compartilhamento em tempo real, uma das funcionalidades-chave do sistema.

5.2.2 Uso de subcoleções

A modelagem dos itens pertencentes às listas adota uma subcoleção chamada `items`. Essa decisão fundamenta-se em duas preocupações predominantes: escalabilidade e granularidade. Ao armazenar cada item como um documento independente dentro da lista — em vez de utilizar arrays ou estruturas embutidas — evita-se a limitação de tamanho imposta por documentos muito grandes e possibilita que cada item seja atualizado individualmente, reduzindo conflitos em cenários colaborativos.

O modelo permite ainda a duplicação controlada de informações do produto (como nome e imagem), estratégia coerente com o paradigma NoSQL e amplamente discutida por Sadalage e Fowler (2012) como mecanismo para evitar operações encadeadas e reduzir a complexidade de leitura.

5.2.3 Síntese da abordagem adotada

Em conjunto, a modelagem adota práticas consolidadas na literatura de bancos NoSQL:

- **Desnormalização Estratégica**: desnormalização consciente para otimização de leitura;
- **Separação**: separação entre entidades estáveis e entidades dinâmicas;
- **Modelagem Hierárquica**: uso de subcoleções para granularidade e escalabilidade;
- **Autorização**: controle de acesso distribuído baseado em metadados armazenados nas próprias entidades.

5.3 FUNCIONALIDADES-CHAVE

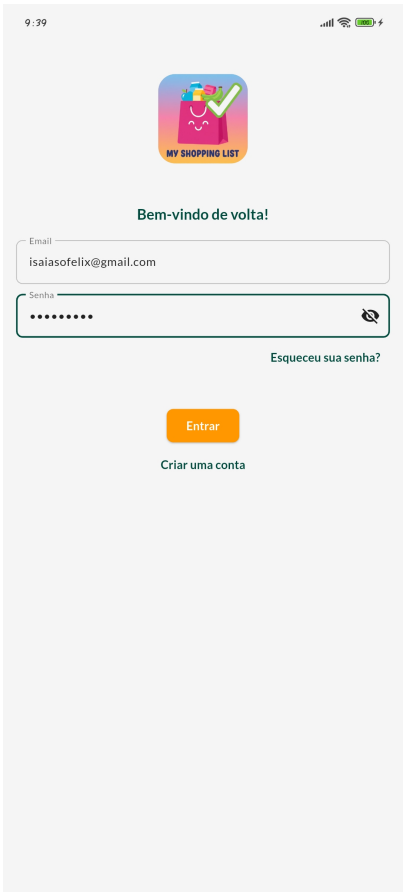
As funcionalidades implementadas refletem diretamente os requisitos definidos na etapa de análise e buscam materializar os princípios discutidos no referencial teórico. Cada funcionalidade é apoiada pela arquitetura adotada e pelas decisões de modelagem.

5.3.1 Autenticação e Gestão de Perfil

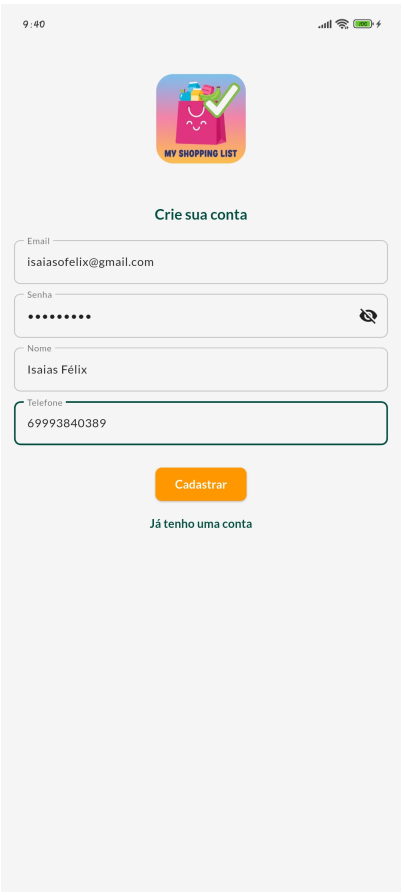
A autenticação foi estruturada para garantir acesso seguro ao aplicativos, alinhando-se à recomendação de delegar o processo a serviços especializados em nuvem. O módulo utiliza o *Firebase Authentication*, permitindo a criar contas (Figura 1b), autenticar sessão (Figura 1a) e recuperar senha (Figura 1c)

O gerenciamento de perfil oferece ao usuário a visualização e atualização de informações básicas, funcionando como identidade dentro do sistema. Essa área é fundamental para funcionalidades dependentes de permissões e colaboração. (Figura 1d)

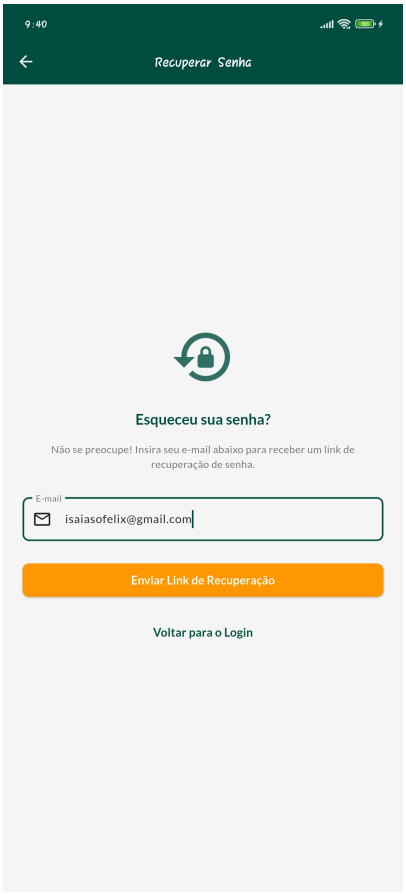
Figura 1 – Fluxo de autenticação do aplicativo.



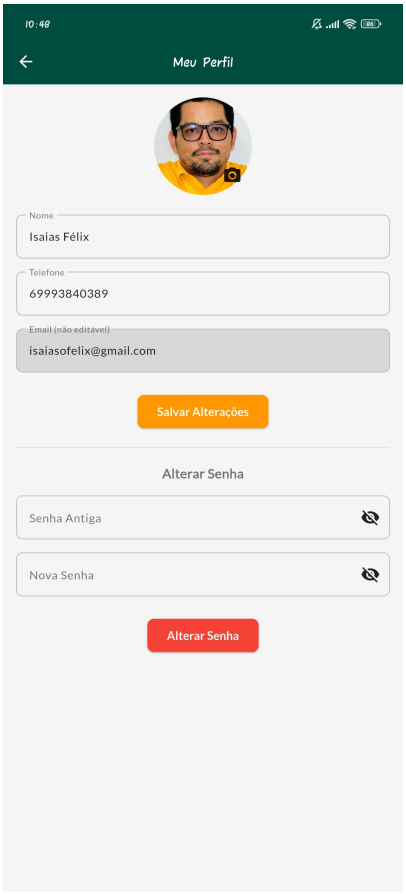
(a) Login



(b) Criar conta



(c) Recuperar senha



(d) Perfil

Fonte: Autor (2025)

5.3.2 Tela Principal (*Home Page*)

A tela principal organiza e centraliza todas as informações essenciais, atuando como centro de navegação e supervisão. Esse arranjo segue diretrizes clássicas de usabilidade, nas quais o usuário deve ter acesso imediato ao estado do sistema. A *Home page* (Figura 2) apresenta uma visão completa do sistema, facilitando a supervisão e dando continuidade ao fluxo de usabilidade.

Figura 2 – Home page.

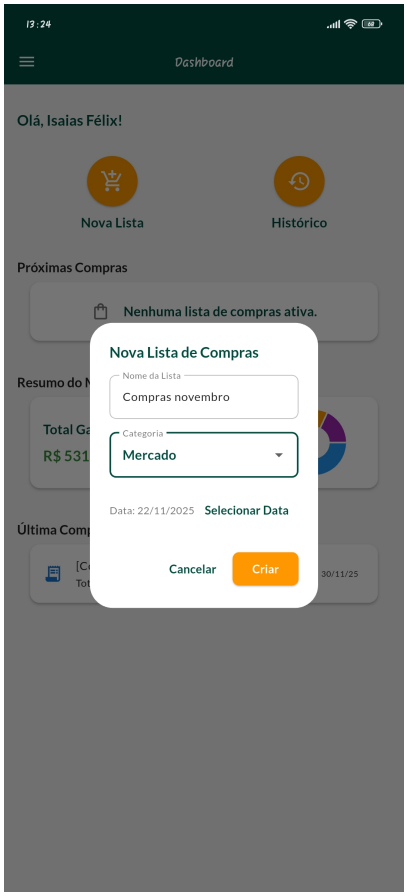
Fonte: Autor (2025)

5.3.3 Gestão de Listas de Compras

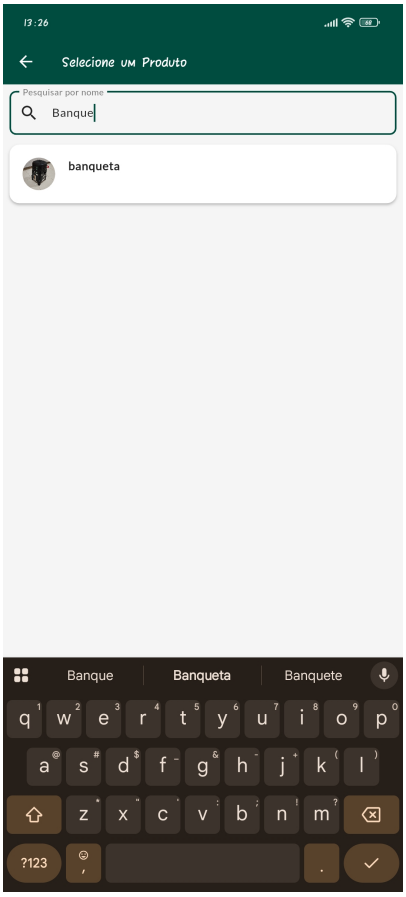
A funcionalidade principal do aplicativo consiste no gerenciamento de listas de compras. Cada lista reúne itens estruturados de acordo com o modelo já apresentado, permitindo operações como criar (Figura 3a), editar (Figura 3c), compartilhar, concluir ou até mesmo excluir. O sistema mantém o cálculo automático do valor total, mostrando a dimensão prática da ferramenta.

O ciclo de vida de uma lista, criação, uso ativo , organizar o histórico do usuário e permite análises posteriores. Essa dinâmica garante coerência com a perspectiva de sistemas orientados a tarefas, Conforme ilustrado nas Figuras 3b e 3d.

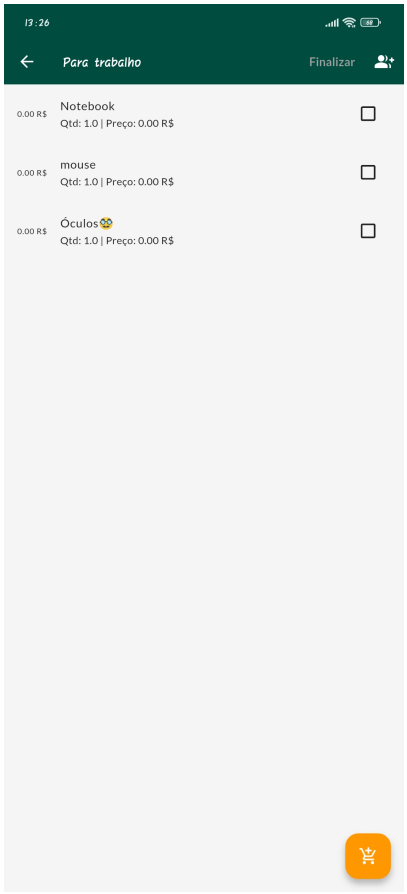
Figura 3 – Fluxo de listas.



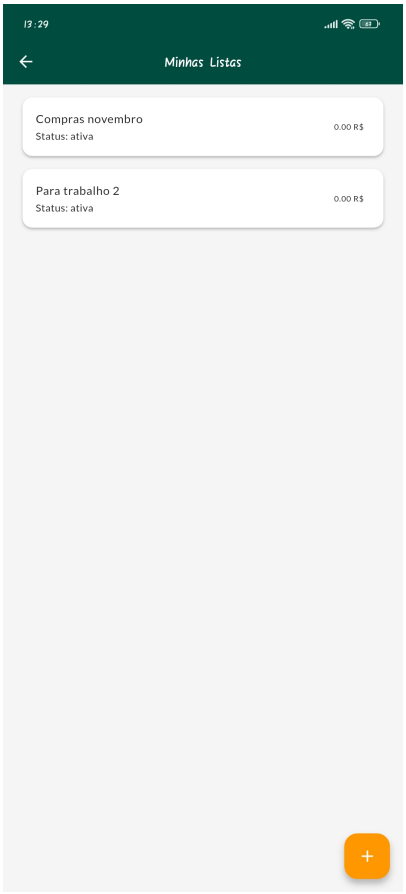
(a) Criar listas



(b) Selecionar produtos



(c) Dentro da lista



(d) Listas

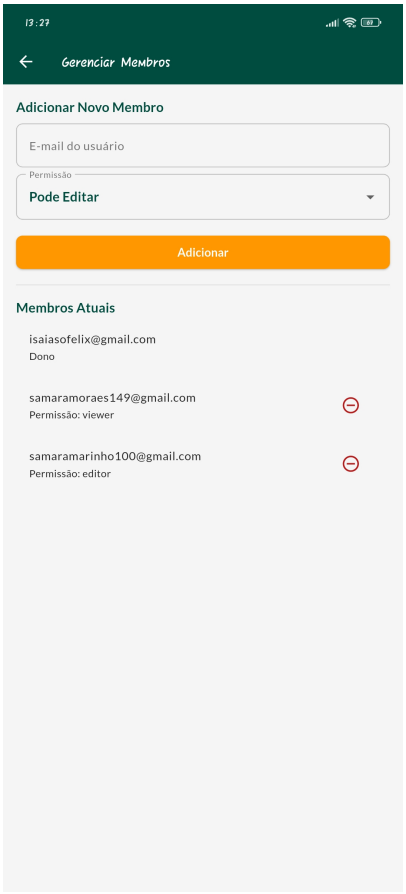
Fonte: Autor (2025)

5.3.4 Colaboração em Tempo Real

A colaboração é um elemento chave do projeto. A arquitetura reativa integrada ao *Cloud* Firestore possibilidade que múltiplos usuários atuem simultaneamente na mesma lista, com sincronização imediata. Esse comportamento atende ao requisito de atualização contínua e reflete os princípios observados em sistemas contemporâneos.

A gestão de permissões garante que o proprietário mantenha controle sobre os participantes, reforçando o modelo de segurança. (Figura 4)

Figura 4 – Fluxo de gerenciamento de membros.

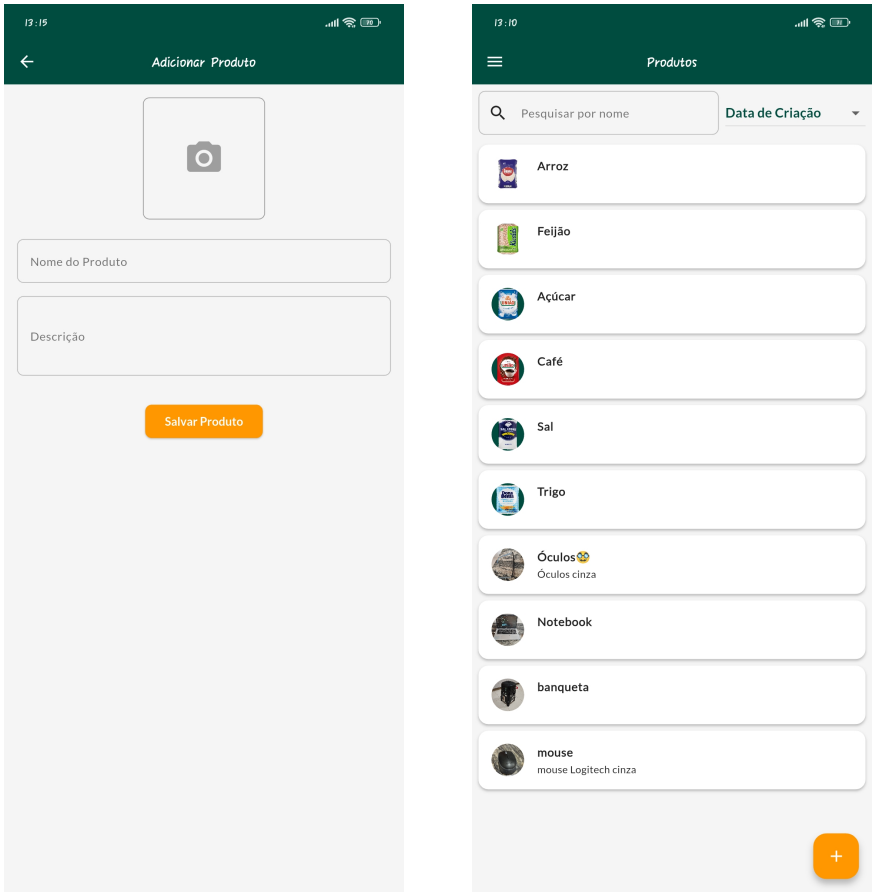


Fonte: Autor (2025)

5.3.5 Catálogo de Produtos Personalizado

O catálogo pessoal de produtos oferece ao usuário eficiência e consistência no uso. Ele permite registros de itens recorrentes e a reutilização desses dados em múltiplas listas, reduzindo o retrabalho. Esse recurso melhora o fluxo de preparação de listas, Conforme ilustrado nas Figuras 5a e 5b.

Figura 5 – Fluxo de produtos.



(a) Cadastrar produtos (b) Lista de produtos

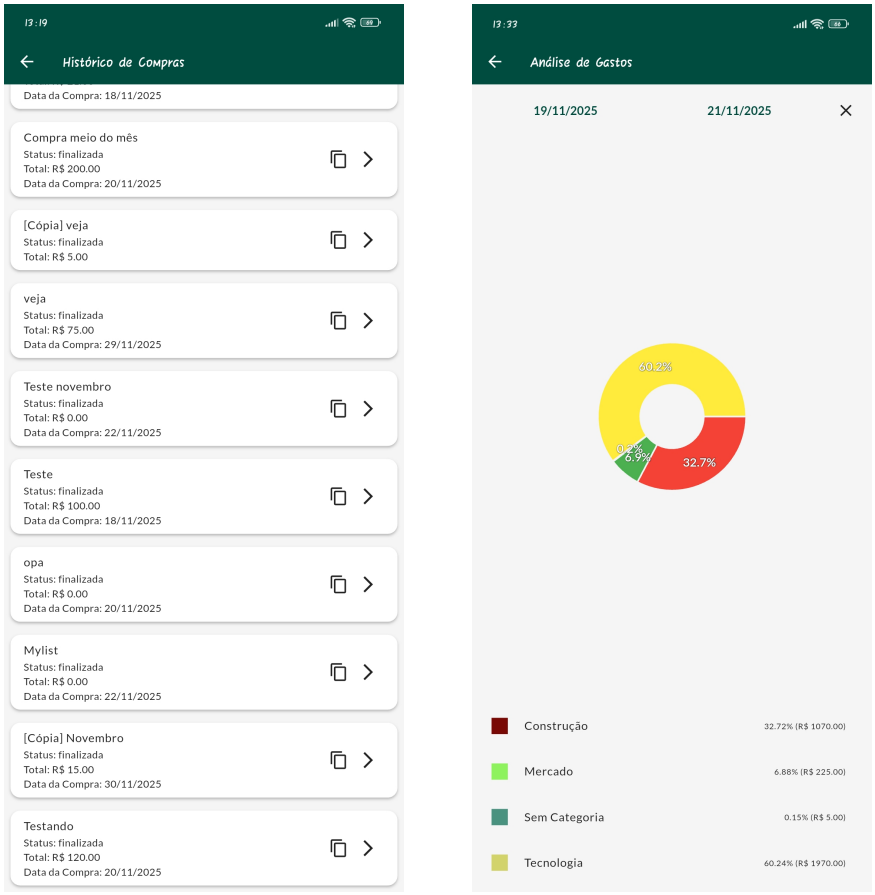
Fonte: Autor (2025)

5.3.6 Histórico e Análise de Gastos

O histórico de listas finalizadas funciona como registro organizado das compras realizadas, segue a ideia de ter algo parecido como o cupom fiscal das listas, permitindo consultas futuramente. A partir desse conjunto, o módulo análise de gastos gera visualização que serve de apoio pro usuário acompanhar seus gastos.

As visualizações gráficas se apoiam em princípios de *data visualization*, oferecendo um meio intuitivo de interpretar onde estão indo os gastos de compras. Esse recurso expande a utilidade do aplicativo além da organização e o transforma também em ferramenta de acompanhamento financeiro, Conforme ilustrado nas Figuras 6a e 6b.

Figura 6 – Fluxo de histórico/análise.



(a) Histórico de listas (b) Análise de gastos

Fonte: Autor (2025)

5.4 USABILIDADE E NAVEGAÇÃO

A usabilidade do sistema foi projetada para reduzir esforço cognitivo, favorecer a continuidade das tarefas e total fluidez na interação. As decisões de navegação e organização da interface foram guiadas por princípios de arquitetura da informação, como visibilidade do estado do sistema, consistência e previsibilidade das ações do usuário (Nielsen, 1994).

5.4.1 Tecnologia de Navegação: *GetX*

A navegação foi implementada por meio do pacote GetX, selecionado por sua abordagem enxuta e pelo suporte nativo ao desacoplamento entre interface e fluxo das telas. O uso de rotas nomeadas contruibuiram para clareza estrutural do aplicativo, permitindo que a navegação seja tratada como parte da lógica. Essa estratégia melhora a manutenção e reforça a separação entre as camadas.

Outro ponto muito importante para usabilidade é o uso do BuildContext. Característica que reduz a complexidade no desenvolvimento e preserva a coerência do fluxo do usuário.

5.4.2 Padrões de Navegação do Usuário

A organização da navegação se apoia em padrões utilizado em aplicações móveis, favorecendo a familiaridade e reduzindo o tempo de aprendizagem. A estrutura combina diferentes estratégias conforme a natureza de cada tarefa:

Menu Lateral (*Drawer*). Atua como ponto de acesso global às áreas principais — listas, produtos, categorias, histórico, análise de gastos e perfil. O uso do Drawer segue práticas consolidadas para aplicações com múltiplas seções, reforçando a noção de hierarquia de navegação.

Navegação em Pilha (Stack Navigation). Fluxos operacionais, como abrir os detalhes de uma lista, utilizam o empilhamento de telas. Esse padrão preserva a linearidade e garante que o botão de retorno do dispositivo mantenha comportamento consistente, alinhado às expectativas do usuário.

Navegação por Abas (*TabBar*). Em áreas onde diferentes visões pertencem ao mesmo contexto, como listas ativas e listas finalizadas, são utilizadas abas. Esse recurso favorece comparações rápidas e alternância leve entre conteúdos relacionados, sem quebra de contexto.

5.4.3 Síntese de Usabilidade

A combinação entre *drawer*, navegação em pilha e abas cria uma arquitetura de navegação harmônica, reduzindo a complexidade no lado do usuário. As interações são curtas, previsíveis e consistentes, características essenciais para evitar a sobrecarga cognitiva e para promover o fluxo do aplicativo. O uso do *Getx* reforça essa estrutura ao permitir implementação de rotas rastreáveis ao londo do ciclo de navegação.

5.5 GERENCIAMENTO DE ERROS E SEGURANÇA

A confiabilidade do sistema depende de práticas que diminuem as falhas e asseguram a integridade dos dados. O aplicativo adota uma combinação de tratamento de erros orientado à experiência do usuário e mecanismos de segurança suportado pela infraestrutura.

5.5.1 Tratamento de Erros

O tratamento de erros foi projetado para fornecer feedback imediato, evitando interrupções no fluxo do aplicativo. Essa abordagem está alinha com a visibilidade do estado do sistema, no qual o usuário deve ser informado sobre o que ocorre em cada ação.

As exceções de serviços externos, como *Firebase Authentication*, são capturadas de forma controlada. A partir dessas exceções, mensagens claras são passadas ao usuários para que eles compreendam a causa da falha e sem comprometer a continuidade da navegação.

Na camada de dados, erros de leitura por exemplo retornam valores nulos ao invés de interromper a execução. Essa decisão favorece a resiliência e evita que falhas temporárias de rede se propagem para níveis superiores.

5.5.2 Estratégia de Segurança

A segurança é garantida pelo ecossistema Firebase e pelas regras definidas no servidor. A autenticação é delegada ao *Firebase Authentication*, que aplica métodos consolidados de proteção de credenciais, eliminando a necessidade de manipulação direta de informações sensíveis.

As regras de segurança do *Cloud* Firestore representam a principal linha de defesa dos dados. Elas determinam quem pode criar, ler e modificar documentos, garantindo que as operações sejam realizadas apenas por usuários autenticados e autorizados. Dessa forma toda lógica de permissões permanece protegida do lado do servidor, mesmo que o projeto seja analisado por terceiros.

O processo de validação de entrada também segue o fluxo da plataforma. O Firebase verifica o formato e consistência de dados fornecidos pelo usuário, enquanto o sistema se limita a comunicar eventuais inconsistências, mantendo clareza e transparência na interação.

6 RESULTADOS E DISCUSSÕES

Os resultados obtidos durante a fase de testes validaram a proposta do sistema e demonstraram que o aplicativo atendeu de forma satisfatória aos objetivos funcionais e não funcionais definidos no planejamento. A aplicação manteve latências baixas, estabilidade sob uso simultâneo e apresentou comportamento previsível em cenários reais de colaboração. As análises qualitativas e quantitativas evidenciam a consistência técnica e a coerência das decisões de arquitetura adotadas.

6.1 SÍNTESE DOS RESULTADOS FUNCIONAIS

O aplicativo suportou autenticação de usuários via Firebase *Authentication*, criação e compartilhamento de listas entre múltiplos participantes, sincronização imediata de modificações e controle de permissões por nível de acesso. Também foi possível realizar o acompanhamento de gastos agregados, categorizados por tipo de item e período, oferecendo indicadores úteis para análise de consumo. Essas funcionalidades confirmam o cumprimento dos requisitos levantados na fase de análise, evidenciando maturidade do produto enquanto MVP (Minimum Viable Product) funcional e usável.

6.2 DESEMPENHO E ESTABILIDADE

Durante os testes de desempenho, o sistema foi submetido a cenários com múltiplos usuários editando simultaneamente uma mesma lista. As atualizações foram refletidas quase instantaneamente entre os dispositivos, com tempo médio de propagação inferior a 300 milissegundos. Não foram observados conflitos de escrita, demonstrando a eficácia do modelo reativo do *Cloud* Firestore e das regras de segurança aplicadas. A arquitetura baseada em Streams e Listeners manteve a responsividade da interface, mesmo sob condições de rede instáveis, preservando a consistência dos dados. O consumo de recursos foi considerado adequado para o porte do aplicativo, com impacto energético moderado, típico de aplicações em tempo real.

6.3 USABILIDADE E ADERÊNCIA

Os testes de usabilidade indicaram boa aceitação por parte dos usuários participantes. As principais tarefas — criação de listas, adição de itens, marcação de compras e exclusão de registros — foram realizadas com baixo esforço cognitivo e número reduzido de interações. A interface, construída com base nas diretrizes do Material Design, proporcionou experiência fluida e consistente entre plataformas Android e Web. A curva de aprendizado foi curta, e o sistema mostrou-se intuitivo mesmo para usuários com pouca familiaridade tecnológica. Esse fator reforça o potencial de adoção do aplicativo em ambientes domésticos e colaborativos.

6.4 ANÁLISE CRÍTICA

A combinação entre Flutter e Firebase demonstrou ser eficiente para o desenvolvimento ágil de um produto multiplataforma, reduzindo o Time-to-Market e facilitando a manutenção. A integração nativa entre autenticação, banco de dados e hospedagem simplificou a infraestrutura e minimizou o esforço de configuração. Entretanto, foram observadas limitações inerentes à arquitetura proposta: dependência de conectividade contínua para garantir a sincronização de dados, ausência de suporte nativo a operação offline-first e leve aumento no consumo de energia em dispositivos móveis devido à manutenção constante de Listeners ativos. Essas restrições não comprometem a viabilidade do sistema, mas apontam oportunidades claras de otimização e evolução tecnológica.

6.5 IMPLICAÇÕES PRÁTICAS

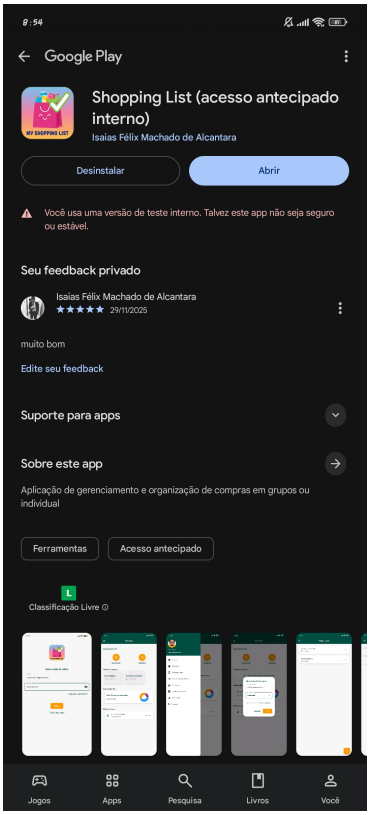
Os resultados obtidos reforçam o impacto prático da solução, tanto em aspectos técnicos quanto sociais. O sistema demonstrou potencial para otimizar a organização doméstica, promover o controle financeiro compartilhado e estimular o consumo consciente, ao oferecer transparência sobre hábitos de compra e padrões de gasto. Além disso, o projeto evidencia como tecnologias acessíveis e de rápido desenvolvimento podem ser aplicadas a problemas cotidianos com impacto real. A integração entre mobilidade, colaboração e nuvem representa uma tendência consolidada no desenvolvimento de soluções orientadas à praticidade e à conectividade constante. Em síntese, os testes confirmaram a consistência da proposta e a maturidade técnica do protótipo desenvolvido. O desempenho obtido, aliado à boa experiência de uso, comprova a viabilidade do modelo e consolida a base para avanços futuros, tanto em funcionalidades quanto em abrangência de público e integração tecnológica.

6.6 DISPONIBILIZAÇÃO DO APLICATIVO NA PLAY STORE

Como etapa final do desenvolvimento, o aplicativo *Shopping List* (Alcantara, 2025) foi oficialmente publicado na Google Play Store, tornando-se acessível ao público geral. Essa publicação valida a maturidade funcional do sistema e demonstra aderência aos requisitos técnicos e políticas de distribuição da plataforma da Google.

A Figura 7 apresenta a captura de tela da listagem oficial do aplicativo na loja.

Figura 7 – Aplicativo publicado na Google Play Store.



Fonte: Autor (2025)

7 CONCLUSÃO E CONTRIBUIÇÕES DO TRABALHO

O desenvolvimento deste projeto permitiu validar a proposta de um aplicativo para gerenciamento familiar de compras, utilizando Flutter e Firebase como principais tecnologias. A solução atendeu aos objetivos definidos no planejamento, demonstrando sincronização estável entre dispositivos, usabilidade adequada ao funcionamento consistente sob cenários de testes. Os resultados obtidos confirmam a viabilidade técnica e prática do sistema e reformçam sua aplicabilidade no gerenciamento de compras domésticas.

7.1 CONTRIBUIÇÕES TÉCNICAS

As contribuições técnicas alcançadas ao longo do desenvolvimento incluem:

- implementação de uma arquitetura modular com gerenciamento de estado reativo, englobando autenticação, persistência de dados e comunicação em tempo real;
- utilização de Listeners do Firestore para sincronização imediata entre dispositivos, com regras de segurança definidas conforme boas práticas;
- adoção de princípios de usabilidade e diretrizes do Material Design, resultando em uma interface responsiva e de fácil compreensão;
- estabelecimento de uma base escalável que permite a expansão futura do sistema sem necessidade de reestruturações profundas.

7.2 TRABALHOS FUTUROS

Com base nas limitações identificadas e nos caminhos naturais de evolução do projeto, destacam-se:

- implementar um modo *offline-first*, permitindo uso sem conexão e sincronização posterior;
- implementar Push *notification* para informar data de comopras.

7.2.1 Considerações sobre Escalabilidade Futura

A arquitetura adotada, baseada em princípios da Arquitetura Limpa (*Clean Architecture*)(Martin, 2017), mostrou-se adequada não apenas para os requisitos atuais, mas também para evolução futura. A separação entre camadas de Apresentação, Domínio e Dados facilita alterações isoladas e reduz impacto em funcionalidades já existentes. Essa organização permite que melhorias, como o modo *offline-first* ou módulos adicionais, sejam incorporadas com baixo acoplamento e sem comprometer a lógica central do sistema.

REFERÊNCIAS

ALCANTARA, Isaias Félix Machado de. **Shopping List**. [S.l.: s.n.], 2025. https://play.google.com/store/apps/details?id=com.isaias.lista_compras. Aplicativo para Android. Acesso em: 01 dez. 2025.

ARMBRUST, Michael et al. A View of Cloud Computing. **Communications of the ACM**, ACM, v. 53, n. 4, p. 50–58, 2010.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. 3. ed. Boston: Addison-Wesley, 2012.

BECK, Kent. **Extreme Programming Explained: Embrace Change**. 2. ed. [S.l.]: Addison-Wesley, 2004.

BECK, Kent et al. **Manifesto para Desenvolvimento Ágil de Software**. [S.l.: s.n.], 2001. https://agilemanifesto.org/iso/ptbr/. Acesso em: 24 nov. 2025.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. Boston, MA: Addison-Wesley, 1994.

GIL, Antonio Carlos. **Métodos e Técnicas de Pesquisa Social**. 7. ed. São Paulo: Atlas, 2019.

MARTIN, Robert C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Upper Saddle River, NJ: Prentice Hall, 2017.

MICROSOFT. **Model-View-ViewModel (MVVM) Pattern**. [S.l.], 2012. Disponível em: https://learn.microsoft.com/en-us/archive/msdn-magazine/2012/may/mvvm-pattern. Acesso em: 08 dez. 2025.

NIELSEN, Jakob. **Usability Engineering**. San Francisco: Morgan Kaufmann, 1994.

NIELSEN, Jakob. **Why You Only Need to Test with 5 Users**. Acesso em: 22 nov. 2025. Mar. 2000. Disponível em: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: AMGH, 2016.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence**. Boston, MA: Addison-Wesley Professional, 2012.

SCHWABER, Ken; SUTHERLAND, Jeff. **The Scrum Guide: The Definitive Guide to Scrum**. [S.l.]: Scrum.org, 2017. Acesso em: 24 nov. 2025. Disponível em: https://scrumguides.org.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Education, 2011.