***Campus* Vilhena**
Coordenação do Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

VINICIUS DANIEL MONTEIRO DE SOUZA

Aplicativo para gerenciamento de metas financeiras - Gooaling

VILHENA - RO

2025

VINICIUS DANIEL MONTEIRO DE SOUZA

Aplicativo para gerenciamento de metas financeiras - Gooaling

Monografia entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), *Campus* Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Hedi Carlos Minin.

VILHENA - RO

2025

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

Souza, Vinicius Daniel Monteiro de.
Aplicativo para gerenciamento de metas financeiras - Gooaling / Vinicius Daniel Monteiro de Souza. - Vilhena, 2025.
77 f. : il.

Orientador(a): Prof.Me. Hedi Carlos Minin.

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

1. Gooaling. 2. Goal. 3. Gooling. 4. Metas. 5. Objetivos. I. Minin, Hedi Carlos (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

Bibliotecário(a) Responsável: Rosilene Maria do Couto Marques, CRB-11/321

VINICIUS DANIEL MONTEIRO DE SOUZA

Aplicativo para gerenciamento de metas financeiras - Gooaling

Monografia entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), *Campus* Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Hedi Carlos Minin.

Aprovado em: 10/12/2025 pela banca examinadora.

gov.br
Documento assinado digitalmente
GILBERTO PEREIRA DA SILVA
Data: 26/01/2026 09:57:17-0300
Verifique em https://validar.iti.gov.br

Gilberto Pereira da Silva

Examinador Interno

gov.br
Documento assinado digitalmente
MARCO ANTONIO AUGUSTO DE ANDRADE
Data: 02/02/2026 09:45:27-0300
Verifique em https://validar.iti.gov.br

Marco Antônio Augusto de Andrade

Examinador Interno

gov.br
Documento assinado digitalmente
HEDI CARLOS MININ
Data: 22/01/2026 20:58:00-0300
Verifique em https://validar.iti.gov.br

Hedi Carlos Minin

Orientador

As pessoas costumam dizer que a motivação não dura sempre.
Bem, nem o efeito do banho, por isso recomenda-se diariamente..

Agradecimentos

A realização deste trabalho não teria sido possível sem o apoio de muitas pessoas, às quais expresso minha mais sincera gratidão.

Primeiramente, agradeço ao meu orientador, Hedi Carlos Minin, pela paciência, orientação e conselhos valiosos durante todas as fases deste projeto. Sua dedicação e conhecimento foram essenciais para a conclusão deste trabalho.

Agradeço também aos professores José Lucas Brandão Montes, Rosa Maria da Silva Gonçalves, Wesley Jhonnes Ramos Rolim e Erick Leonardo Weil, Gilberto Pereira da Silva e Marco Antonio Augusto de Andrade que contribuíram significativamente com seus ensinamentos e sugestões ao longo da minha formação acadêmica.

Um agradecimento especial à minha família, que sempre me apoiou e acreditou em mim, oferecendo amor e suporte em todas as fases deste trabalho.

Por fim, agradeço ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - Campus Vilhena, pela oportunidade e pelos recursos disponibilizados, que foram fundamentais para a conclusão deste estudo.

Resumo

Este projeto apresenta o desenvolvimento de um aplicativo móvel destinado à criação, ao gerenciamento e ao acompanhamento de metas financeiras pessoais, nomeado “Gooaling”; foi concebido para estruturar o processo de definição de objetivos, registro de aportes e visualização de progresso ao longo do tempo, organizando de forma centralizada as etapas relacionadas ao acompanhamento financeiro. Ao longo do desenvolvimento, foram estruturadas funcionalidades que incluem a criação de contas bancárias internas concebidas para funcionar como um espelho manual das contas reais do usuário, permitindo que ele utilize essa estrutura para registrar aportes e transações, o cadastro de metas com parâmetros personalizáveis, o registro de transações e o acompanhamento contínuo por meio de gráficos e indicadores. A implementação utilizou arquitetura e práticas adequadas ao ambiente mobile, envolvendo o uso do Jetpack Compose para construção das interfaces, a gestão de estados e a definição de fluxos de navegação que garantem consistência no uso do aplicativo. Foram utilizados serviços de API e banco de dados para armazenamento e sincronização das informações financeiras, incluindo o envio, consulta e atualização de dados por meio de endpoints específicos e validações no backend. O processo de desenvolvimento e testes descrito neste trabalho evidencia a viabilidade técnica da solução proposta e registra as principais decisões, limitações e possibilidades de aprimoramento para versões futuras do aplicativo.

Palavras-chave: gooaling. goal. gooling. metas. objetivos. financeiro.

Abstract

This project presents the development of a mobile application designed for the creation, management, and monitoring of personal financial goals, named "Gooaling"; it was conceived to structure the process of defining objectives, recording contributions, and visualizing progress over time, centrally organizing the steps related to financial monitoring; throughout the development, functionalities were structured that include the creation of internal bank accounts designed to function as a manual mirror of the user's real accounts, allowing them to reproduce within the application the financial organization they have in their daily life and use this structure to record contributions and transactions, the registration of goals with customizable parameters, the recording of transactions, and continuous monitoring through graphs and indicators; the implementation used architecture and practices appropriate to the mobile environment, involving the use of Jetpack Compose for building interfaces, managing states, and defining navigation flows that ensure consistency in the use of the application; API and database services were used for storing and synchronizing financial information, including sending, querying, and updating data through specific endpoints and validations in the backend; The development and testing process described in this work demonstrates the technical feasibility of the proposed solution and records the main decisions, limitations, and possibilities for improvement in future versions of the application.

Keywords: gooaling. goal. gooling. goals. objectives. financial.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CRUD	Acrônimo para Create (criação), Read (consulta), Update (atualização) e Delete (destruição) de dados
MVP	Most Value Product
QA	Quality Assurance

Sumário

1 Introdução

1.1 Contexto e problema

O controle das finanças pessoais é essencial para a saúde financeira de qualquer indivíduo. A falta de educação financeira, combinada com maus hábitos, perpetua a dificuldade em administrar finanças ao longo da vida e contribui significativamente para a inadimplência. Dados do Serviço de Proteção ao Crédito (SPC) indicam que uma parcela considerável da população brasileira, especialmente os mais jovens, enfrentam dívidas devido à carência de conhecimento sobre gestão financeira adequada. Essa realidade é ainda mais complexa com o acesso facilitado ao crédito e a crescente influência de práticas consumistas, que aumentam a vulnerabilidade ao endividamento (SPC Brasil, 2021).

Este cenário contribui para o acúmulo de dívidas e dificulta a conquista de objetivos financeiros de longo prazo, como a compra de um imóvel ou a formação de uma reserva de emergência. Como destacado em uma reportagem do G1 (2022), "fazer um bom planejamento financeiro ajuda a organizar as contas, entender gastos e receitas, e prever quando alguns objetivos mais robustos poderão ser realizados". Essa afirmação ressalta a importância de um planejamento adequado para o sucesso financeiro.

De acordo com Sousa (2008), muitas pessoas com pouco conhecimento financeiro não sabem avaliar uma compra ou analisar o melhor investimento para seus recursos. Essa lacuna na gestão financeira pessoal pode levar a decisões inadequadas, dificultando e, em muitos casos, impossibilitando a realização de sonhos significativos. Assim, a ausência de uma solução prática e eficiente para atender a essas necessidades destaca a importância de desenvolver uma ferramenta que permita aos usuários planejar, acompanhar e ajustar seus propósitos financeiros de maneira eficaz.

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo geral deste projeto é desenvolver um aplicativo móvel que auxilie os usuários no gerenciamento de seus objetivos financeiros, permitindo o estabelecimento, monitoramento e alcance de metas financeiras.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalhos são:

- Identificar os requisitos necessários para que uma meta seja gerenciada de maneira prática e eficaz.
- Validar os requisitos levantados.
- Desenvolver uma solução intuitiva e robusta para a gestão de metas financeiras.
- Avaliar a eficácia da solução proposta por meio de feedback dos usuários, e realizar ajustes conforme necessário para melhorar a experiência e os resultados.

1.3 Justificativa

De acordo com um estudo do SPC Brasil (SPC Brasil, 2021), oito em cada dez brasileiros não sabem controlar adequadamente suas próprias despesas. Além disso, um relatório da Associação Brasileira das Entidades dos Mercados Financeiros e de Capitais (ANBIMA), realizado em 2017 com 3.374 pessoas em 152 municípios, revela que 40% dos entrevistados não conseguem manter saldo positivo ao final do mês, visto que todo o seu rendimento é destinado ao pagamento de contas (ANBIMA, 2017).

Segundo Santos (2018), a participação da população na economia é prejudicada pela falta de acesso ao sistema financeiro, o que dificulta a geração e o acúmulo de riqueza. Para abordar essa questão, o autor identifica estratégias para melhorar a educação financeira, promover a inclusão e ampliar a conscientização sobre a importância dos produtos e serviços do sistema financeiro. Além disso, destaca-se a necessidade de ampliar o acesso a plataformas digitais, aplicativos e fintechs sociais, que desempenham um papel fundamental na melhoria da gestão financeira pessoal. Nesse contexto, um aplicativo móvel voltado ao controle de metas financeiras apresenta-se como uma solução viável e conveniente em resposta à crescente demanda por ferramentas eficazes que auxiliem na gestão das finanças pessoais.

2 Fundamentação teórica

Este capítulo descreve os principais conceitos explorados durante o trabalho, tratando elementos presentes no desenvolvimento do software, sua arquitetura e tecnologias utilizadas.

2.1 Educação financeira pessoal e sua importância

A educação financeira pode ser entendida como a capacidade de entender e gerenciar o dinheiro de forma consciente e responsável. De acordo com Lusardi (2009), essa prática envolve o conhecimento sobre diferentes opções de investimento, a habilidade de compreender cálculos financeiros e a familiaridade com conceitos como inflação, juros compostos, tributação e diversificação de investimentos.

Além disso, Lusardi (2007) destaca que a educação financeira é uma ferramenta essencial para que decisões financeiras sejam tomadas de forma mais consciente. Isso se torna especialmente relevante em um contexto onde a complexidade do mercado financeiro e a abundância de opções de crédito exigem maior preparação dos indivíduos. Assim, o desenvolvimento dessa competência permite não apenas evitar erros financeiros, mas também explorar de maneira estratégica as oportunidades disponíveis.

Já segundo a Organização de Cooperação e Desenvolvimento Econômico (2005), a educação financeira pode ser definida como:

> [...] o processo mediante o qual os indivíduos e as sociedades melhoram a sua compreensão em relação aos conceitos e produtos financeiros, de maneira que, com informação, formação e orientação, possam desenvolver os valores e as competências necessários para se tornarem mais conscientes das oportunidades e riscos neles envolvidos e, então, poderem fazer escolhas bem informadas, saber onde procurar ajuda e adotar outras ações que melhorem o seu bem-estar. Assim, podem contribuir de modo mais consistente para a formação de indivíduos e sociedades responsáveis, comprometidos com o futuro.

Além disso, de acordo com Ministério do Meio Ambiente (2022), a educação financeira vai além do que aprender a acumular dinheiro, a investir, a poupar, economizar e cortar gastos, ela é essencial para quem busca valorizar seu próprio tempo, seu próprio trabalho, sua própria renda. Educação Financeira é qualidade de vida.

Pelo exposto até o momento, pode-se dizer que a educação financeira é um conceito amplo que não se limita apenas ao acúmulo de dinheiro, mas se expande para uma compreensão mais profunda sobre como o conhecimento financeiro impacta o bem-estar pessoal e social.

Para Pereira (2003), a educação financeira deveria ter inicio aproximadamente aos dois ou três anos de idade, quando a criança solicita dinheiro pela primeira vez para comprar doces e brinquedos. O processo de educação financeira inicia com a compreensão dos valores das moedas. É possível afirmar que uma nova perspectiva está surgindo através da conscientização sobre o ensino da educação financeira nas escolas, destinado a crianças e jovens do Brasil. Certamente, existem alterações nos valores, mudando da fase do "ter"para a fase do "ser".

2.2 Metas financeiras

A palavra "metas"em latim é "metae", e tem um significado relacionado a um marco, objetivo ou ponto de chegada. Por sua vez, a palavra financeira em latim não tem uma tradução direta e exata, mas o termo mais próximo seria "financiariae", que é um adjetivo derivado de "financia"(finanças).

Dessa forma, a tradução de "metas financeiras"para o latim seria "metae financialis"ou "metae financiariae", utilizando "metae"(metas) e "financialis"ou "financiariae"(relativo a finanças). Essa expressão pode ser usada para se referir a objetivos financeiros ou planos que uma pessoa ou organização deseja alcançar, relacionados a dinheiro, investimentos ou gestão de recursos.

As metas financeiras pessoais são objetivos definidos por um indivíduo ou família para alcançar uma situação financeira desejada. Elas servem como um guia para as decisões sobre o dinheiro, ajudando a manter o foco e a disciplina, bem como permitir que o indivíduo avalie o progresso em relação aos seus objetivos (INVESTIMENTOS, 2023).

Logo, um bom planejamento financeiro é a principal arma estratégica para que as pessoas sejam capazes de determinar suas metas financeiras de curto e longo prazo, a partir da análise da própria situação financeira. Neste sentido, um planejamento financeiro consiste em uma ferramenta administrativa que o indivíduo utiliza para reconhecer o seu cenário atual financeiro, estuda os caminhos possíveis que se poderia tomar e viabiliza a rota para essas metas serem alcançadas, com a prospecção dos recursos disponíveis.

Segundo Ross, o planejamento financeiro estabelece o método pelo qual as metas financeiras devem ser atingidas (ROSS; WESTERFIELD; JAFFE, 2002). Ou seja, vai além de simplesmente definir metas financeiras, ele estabelece a metodologia necessária para atingir esses objetivos de forma eficiente. O planejamento permite que o indivíduo

ou organização analise suas receitas, despesas e investimentos de maneira estruturada, considerando o tempo e os recursos necessários para alcançar as metas. Além disso, o planejamento financeiro envolve a criação de um plano de ação com etapas claras e mensuráveis, o que aumenta as chances de sucesso na realização dos objetivos financeiros de curto, médio e longo prazo.

2.3 Trabalhos similares

Aplicativos como Mobills e Organizze são exemplos de soluções proprietárias que ajudam os usuários a manterem suas finanças em ordem. Embora esses aplicativos sejam eficazes no controle das finanças pessoais, eles não possuem um foco principal no gerenciamento de metas financeiras. A ênfase desses aplicativos está, predominantemente, na organização e monitoramento de despesas e receitas, com funcionalidades limitadas ou menos desenvolvidas para o acompanhamento de metas financeiras específicas ao longo do tempo.

- Mobills[1]: Site do aplicativo Mobills
- Organizze[2]: Site do aplicativo Organizze

[1] Disponível em https://www.mobills.com.br/
[2] Disponível em https://www.organizze.com.br/

3 Materiais e métodos

3.1 Ciclo de vida e processos

3.1.1 Etapas do desenvolvimento de um aplicativo

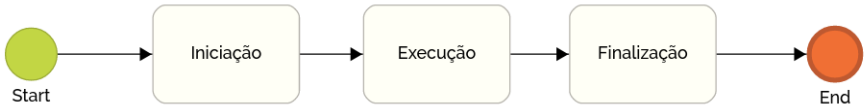
O ciclo de vida do desenvolvimento de um Aplicativo pode ser composto por três etapas: Iniciação, Execução e Finalização.

Inicialização: É composta pelas etapas de Ideação, Planejamento e Design UI/UX, nas quais são definidos os objetivos, requisitos e o escopo do projeto;

Execução: Abrange o Desenvolvimento e Testes e Validação, que correspondem à criação, implementação e verificação do sistema;

Finalização: Contempla a Documentação, Publicação e as atividades de entrega do produto, encerrando o ciclo de vida do aplicativo.

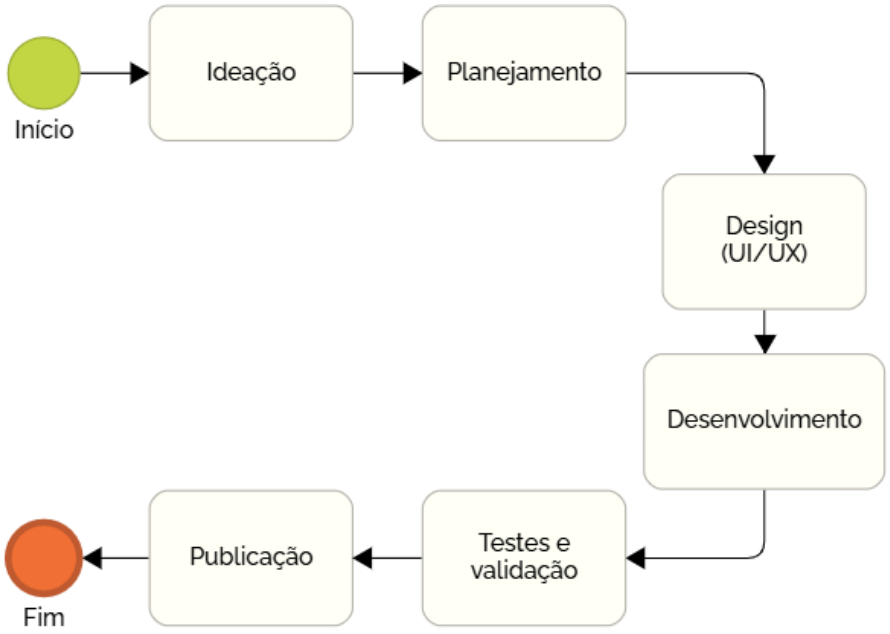
Figura 1 – Ciclo de vida do aplicativo em três etapas



Fonte: elaborado pelo autor (2025)

Cada etapa contém subprocessos fundamentais para a construção da aplicação de forma consistente e escalável. Os subprocessos dessas etapas podem incluir a Ideação, Planejamento, Design UI/UX, Desenvolvimento, Testes, Validação e Publicação. Essa organização de subprocessos segue boas práticas consolidadas, na qual permite uma definição explícita dos processos, atividades e tarefas ao longo do ciclo de vida do desenvolvimento dessa aplicação (BOURQUE; FAIRLEY, 2014) e (ISO/IEC/IEEE, 2017).

Figura 2 – Ciclo de vida do aplicativo em seis subprocessos



Fonte: elaborado pelo autor (2025)

3.1.1.1 Iniciação

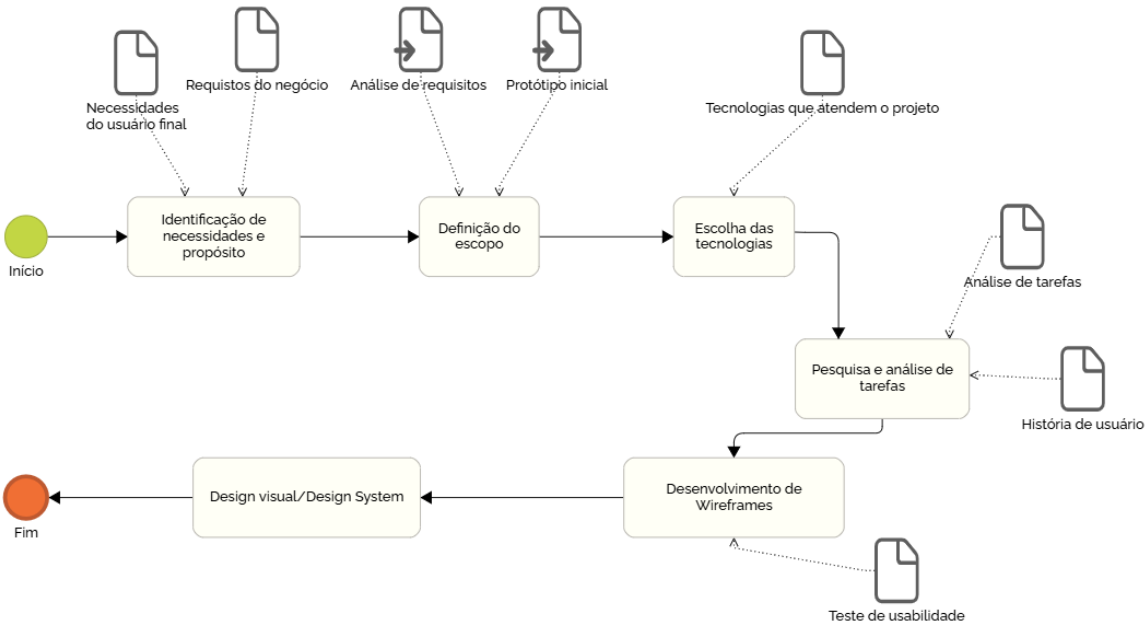
A fase de **Iniciação** marca o início do projeto e tem como objetivo estabelecer a base conceitual, técnica e estratégica do desenvolvimento. Nessa etapa, foi essencial compreender o problema, identificar as necessidades do público-alvo, especificar os objetivos, desenvolver os diagramas e definir um propósito claro, de modo a preparar as condições necessárias para a construção do aplicativo.

A coleta e análise de requisitos permitiram definir o escopo do projeto, orientando o direcionamento das próximas fases. Esse processo envolveu também a seleção das tecnologias mais adequadas. Para o desenvolvimento do aplicativo, optou-se por utilizar Kotlin[1], Jetpack compose[2], pois essas tecnologias atendem o escopo do aplicativo. Além disso, a realização de pesquisas e análises de tarefas, foram fundamentais para compreender as interações e expectativas do usuário.

Com base nesses insumos, foram elaborados os wireframes iniciais, que possibilitaram visualizar e avaliar a estrutura de navegação e a usabilidade do sistema, garantindo uma visão preliminar do funcionamento do aplicativo antes do desenvolvimento efetivo.

[1] Disponível em https://kotlinlang.org/docs/
[2] Disponível em https://developer.android.com/compose

Figura 3 – Fase de iniciação



Fonte: elaborado pelo autor (2025)

3.1.1.2 Execução

A fase de **Execução**, corresponde à implementação prática dos artefatos definidos na etapa anterior. Nessa fase, as ideias e estruturas planejadas são traduzidas em código e em funcionalidades do aplicativo, abrangendo desde a preparação do ambiente de desenvolvimento até a realização dos testes.

O processo iniciou-se com a configuração do ambiente e da arquitetura, definindo padrões de organização do código, dependências, bibliotecas e camadas do sistema. Nessa etapa, também foram criadas a estrutura inicial do projeto, as ferramentas de versionamento e de controle de dependências e o registro da arquitetura adotada.

Na sequência, ocorreu o planejamento da codificação, com definição das funcionalidades, da ordem de desenvolvimento e das estratégias de integração entre o aplicativo móvel e o backend. Desenvolveu-se, adicionalmente, um backend para o aplicativo: uma API *RESTful*[3] pela comunicação do app com o banco de dados e regras de negócio, implementada em Node.js[4] com o framework Express.js[5] e uso de MongoDB[6] para armazenamento e consulta de dados.

[3] Disponível em <https://aws.amazon.com/pt/what-is/restful-api/>
[4] Disponível em <https://nodejs.org/pt>
[5] Disponível em <https://expressjs.com/>
[6] Disponível em <https://www.mongodb.com/what-is-mongodb>

A organização do código do backend seguiu o padrão Model–View–Controller (MVC) (PERICAS-GEERTSEN; RIEM, 2015). Os *models* foram definidos com a biblioteca Mongoose[7] para mapeamento de esquemas e validações; os *controllers* trataram as requisições e a lógica de negócio; e as *rotas* definiram os *endpoints* expostos ao aplicativo.

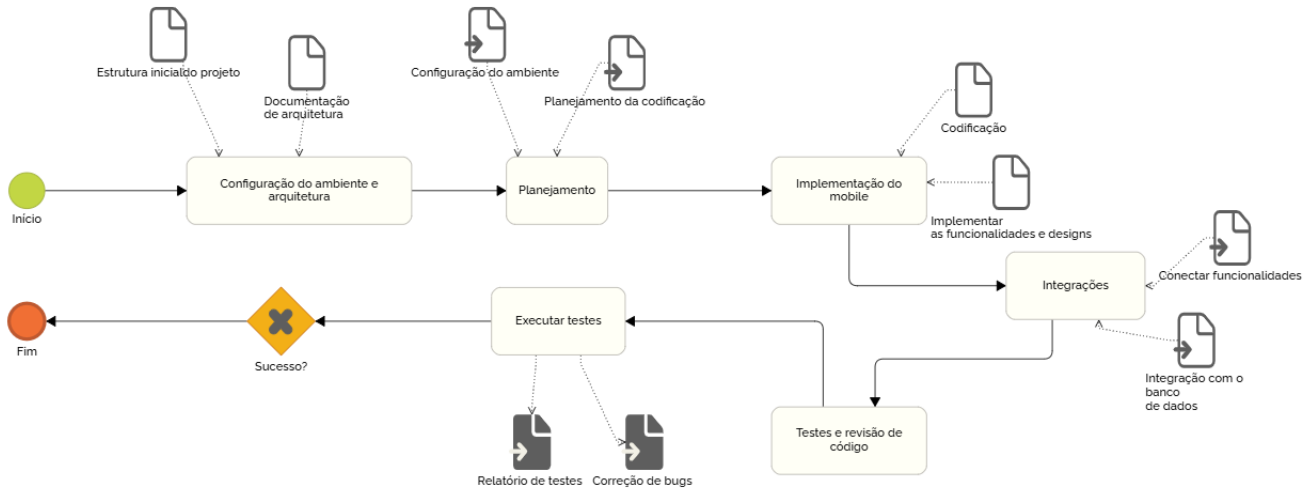
A etapa seguinte compreendeu a implementação do aplicativo, com codificação de telas, componentes visuais e fluxos de navegação conforme o design elaborado.

Em paralelo, foram realizadas as integrações entre aplicativo e API, além de serviços complementares, estabelecendo a comunicação necessária para carregamento, armazenamento e sincronização dos dados.

Posteriormente, ocorreram testes e revisões de código para identificação de falhas de funcionamento, problemas de layout e inconsistências lógicas. Realizaram-se testes de responsividade para verificação em diferentes tamanhos e resoluções de tela, bem como revisões de código para legibilidade, padronização e conformidade com boas práticas.

O monitoramento do progresso permitiu ajustes no código, nas funcionalidades e no cronograma, mantendo o projeto alinhado ao planejamento. Em síntese, a fase de Execução resultou em um protótipo funcional com integração ao backend e base para evolução.

Figura 4 – Fase de execução



Fonte: elaborado pelo autor (2025)

[7] Disponível em <https://mongoosejs.com/>

3.1.1.3 Finalização

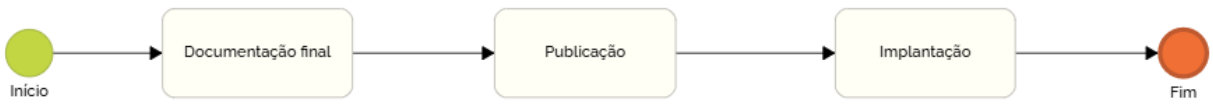
A fase de **Finalização**, corresponde ao encerramento do ciclo de desenvolvimento do aplicativo, momento em que todas as funcionalidades foram revisadas, testadas e preparadas para entrega. Essa etapa teve como propósito consolidar os resultados obtidos, garantir a qualidade do sistema e documentar de forma organizada todo o processo desenvolvido.

Inicialmente, foram realizados testes de unidade e integração na parte do backend com o objetivo de verificar o funcionamento conjunto dos módulos implementados e assegurar a estabilidade do sistema. Em seguida, procedeu-se à validação do aplicativo, etapa que consistiu em simular o uso real do Gooaling em ambiente controlado, confirmando o atendimento aos requisitos definidos nas fases iniciais.

Com o sistema validado, foi realizada a entrega interna do protótipo funcional, garantindo que a aplicação estivesse pronta para uso em ambiente de testes. Nessa etapa, também foram elaboradas instruções de utilização e orientações técnicas voltadas à manutenção e continuidade do projeto.

Por fim, efetuou-se a documentação final, reunindo informações sobre a arquitetura, dependências e funcionalidades principais. Após essa consolidação, realizou-se a revisão geral e o encerramento do projeto, assegurando que todas as entregas planejadas fossem concluídas com sucesso e que o aplicativo atendesse às expectativas estabelecidas.

Figura 5 – Fase de finalização



Fonte: elaborado pelo autor (2025)

3.2 Ferramentas e tecnologias Utilizadas

Interface de Programação de Aplicações (API): APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos. Por exemplo, o sistema de software do instituto meteorológico contém dados meteorológicos diários. A aplicação para a previsão do tempo em seu telefone “fala” com esse sistema por meio de APIs e mostra atualizações meteorológicas diárias no telefone. (Amazon Web Services, 2024c)

A arquitetura da API geralmente é explicada em termos de cliente e servidor. A aplicação que envia a solicitação é chamada de cliente e a aplicação que envia a resposta é chamada de servidor. Então, no exemplo do clima, o banco de dados meteorológico do instituto é o servidor e o aplicativo móvel é o cliente. (Amazon Web Services, 2024c)

As interfaces de programação de aplicações, ou APIs, são essenciais para conectar e comunicar entre vários sistemas e aplicações. Eles estabelecem padrões para que diferentes softwares interajam de forma segura e eficaz. No exemplo anterior, a API permite que a aplicação de previsão do tempo acesse dados do sistema meteorológico sem precisar saber o processo interno de coleta ou armazenamento de dados. Como resultado, a integração é facilitada e a reutilização de componentes de software é promovida.

Portanto, o desenvolvimento de aplicações é mais fácil e ajuda a reduzir tempo e custos de produção. O uso de APIs na criação de novas ferramentas e no gerenciamento de ferramentas existentes resulta em uma maior flexibilidade, oportunidades de inovação e melhor gerenciamento.

Protocolo Hypertext Transfer Protocol: O Hypertext Transfer Protocol (HTTP) é um protocolo usado para transmitir textos, mídias e outros tipos de dados pela Internet. Ele opera com um modelo de requisição e resposta, no qual um cliente se comunica com um servidor, faz uma solicitação e espera pela resposta que o servidor irá fornecer.

Por exemplo, quando deseja visualizar alguns dados de um site, você envia a solicitação HTTP GET. Se quiser enviar algumas informações, como preencher um formulário de contato, você envia a solicitação HTTP PUT.

Da mesma forma, o servidor envia diferentes tipos de respostas HTTP na forma de códigos numéricos e dados.

200 - OK

400 - Solicitação inválida

404 - Recurso não encontrado

Essa comunicação de solicitação/resposta geralmente é invisível para os usuários. É o método de comunicação que o navegador e os servidores Web usam. (Amazon Web

Services, 2024a)

Hypertext Transfer Protocol Secure: O Hypertext Transfer Protocol Secure (HTTPS) é a versão segura do HTTP, que é o principal protocolo usado para enviar dados entre um navegador web e um site. O HTTPS é criptografado para aumentar a segurança da transferência de dados. Isso é particularmente importante quando usuários transmitem dados sensíveis, como quando fazem login em uma conta de banco, serviço de e-mail ou provedor de seguro saúde.(Cloudflare, 2024)

Arquitetura REST: O REST, do inglês Representational State Transfer, é um modelo de arquitetura baseado no protocolo HTTP, que especifica regras para a criação de implementações de software. Desta forma, se uma API é criada com base na arquitetura REST, pode-se dizer que ela é RESTful (FIELDING, 2000)

Dessa maneira, se ela é RESTful significa que a API é simples e escalável, usando as habilidades do protocolo HTTP para manipular recursos usando métodos HTTP comuns como GET, POST, PUT e DELETE. A arquitetura REST facilita a interação entre clientes e servidores, permitindo que as operações sobre recursos sejam realizadas de forma fácil e eficaz. A flexibilidade no formato das representações e a utilização de URLs para identificar recursos são componentes importantes que contribuem para a capacidade das APIs RESTful de funcionar bem em uma variedade de contextos e aplicações.

A interface uniforme é fundamental para o design de qualquer serviço da Web RESTful. Indica que o servidor transfere informações em formato-padrão. O recurso formatado é chamado de representação em REST. Esse formato pode ser diferente da representação interna do recurso na aplicação do servidor. Por exemplo, o servidor pode armazenar dados como texto, mas enviá-los em um formato de representação HTML. (Amazon Web Services, 2024c)

Model–View–Controller (MVC)

A arquitetura Model View Controller mantém uma organização arquitetural amplamente adotada em aplicações Web, na qual o **model** representa os dados e o estado da aplicação, a **view** define a apresentação desses dados e o **controller** gerencia a entrada do usuário e orquestra a atualização do modelo e a produção da resposta. A proposta enquadra-se na família *action-based*, isto é, requisições HTTP são roteadas a controladores que disparam ações específicas (PERICAS-GEERTSEN; RIEM, 2015).

Model (Modelo).

O **model** concentra os dados, o estado e as regras associadas às entidades do domínio que serão usados para compor a resposta. Na especificação, o modelo pode ser exposto às visões por diferentes mecanismos (por exemplo, um mapa nome–objeto acessível

pela view), de modo que a camada de apresentação obtenha os valores necessários para renderização. Em termos práticos, o modelo é a fonte autoritativa das informações compartilhadas com a view (PERICAS-GEERTSEN; RIEM, 2015).

View (Visão).

A **view** define a estrutura da saída entregue ao cliente. Ela é processada por um *mecanismo de visão* (*view engine*), que consome dados do(s) modelo(s) para produzir o documento final. Embora a especificação ilustre o uso de tecnologias como JSP, o princípio é independente da tecnologia adotada: a view é a representação final apresentada ao usuário (ou ao consumidor da API) construída a partir do modelo (PERICAS-GEERTSEN; RIEM, 2015).

Controller (Controlador).

O **controller** recebe e trata a requisição, coordena o acesso ao modelo e seleciona a view (ou a forma de resposta) a ser retornada. Na especificação, controladores mapeiam endpoints HTTP e podem devolver caminhos de view, objetos *Viewable* ou uma resposta completa, inclusive com suporte a redirecionamentos. Em síntese, o controlador é o ponto de orquestração entre entrada, modelo e saída no estilo *action-based* (PERICAS-GEERTSEN; RIEM, 2015).

NoSQL: NoSQL, que significa "Not Only SQL", é uma categoria de sistemas de gerenciamento de banco de dados que se distingue dos bancos de dados relacionais tradicionais, oferecendo uma abordagem mais flexível e escalável para o armazenamento e recuperação de dados.

Os bancos de dados NoSQL são criados especificamente para modelos de dados específicos e armazenam dados em esquemas flexíveis que se escalam facilmente para aplicações modernas. Esses bancos de dados são amplamente reconhecidos por sua facilidade de desenvolvimento, funcionalidade e performance em escala. Além disso, geralmente fornecem esquemas flexíveis que permitem um desenvolvimento mais rápido e iterativo.

O modelo de dados flexível torna os bancos de dados NoSQL ideais para dados semiestruturados e não estruturados. Ademais, são normalmente projetados para aumentar a escala horizontalmente usando clusters distribuídos de hardware, em vez de serem ampliados adicionando servidores caros e robustos. Alguns provedores de nuvem lidam com essas operações nos bastidores como um serviço totalmente gerenciado. (Amazon Web Services, 2024b).

Javascript: Segundo o site (Mozilla Developer Network, 2024b), JavaScript (frequentemente abreviado como JS) é uma linguagem de programação leve, interpretada

e orientada a objetos com funções de primeira classe, conhecida como a linguagem de scripting para páginas Web, mas também utilizada em muitos ambientes fora dos navegadores. Ela é uma linguagem de scripting baseada em protótipos, multi-paradigma e dinâmica, suportando os estilos orientado a objetos, imperativo e funcional.

JavaScript pode funcionar tanto como uma linguagem procedural como uma linguagem orientada a objetos. As capacidades dinâmicas de JavaScript incluem a construção de objetos em tempo de execução, listas variáveis de parâmetros, variáveis de funções, criação dinâmica de scripts (através da função eval), introspecção de objetos (através da estrutura for ... in), e recuperação de código fonte (programas escritos em JavaScript podem descompilar funções de volta a seus textos originais).

Node.js: Node.js é uma ambiente de execução de JavaScript disponível para várias plataformas, de código aberto e gratuita, que permite os programadores criar servidores, aplicações da Web, ferramentas de linha de comando e programas de automação de tarefas (Node.js, 2024). Node é usado fora do contexto de um navegador (ou seja executado diretamente no computador ou no servidor). Como tal, o ambiente omite APIs JavaScript específicas do navegador e adiciona suporte para APIs de sistema operacional mais tradicionais, incluindo bibliotecas de sistemas HTTP e arquivos (Mozilla Developer Network, 2024a).

Express.js: O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móveis (Express, 2024).

Uma das principais características do Express é seu sistema de middleware, que permite a adição de funcionalidades em várias etapas do ciclo de vida de uma requisição. Isso inclui o tratamento de requisições e respostas, autenticação, manipulação de erros, e muito mais. O framework também é compatível com diversos tipos de templates, o que possibilita a geração dinâmica de conteúdo HTML, facilitando o desenvolvimento de interfaces interativas (Express, 2024).

MongoDB: MongoDB é um sistema de gerenciamento de banco de dados (DBMS) não relacional, baseado em software livre, que utiliza documentos flexíveis em vez de tabelas e linhas para processar e armazenar várias formas de dados. Como uma solução debanco de dados NoSQL, o MongoDB não requerer um sistema de gerenciamento de banco de dados relacional (RDBMS), portanto, ele oferece um modelo de armazenamento de dados elástico, que permite aos usuários armazenar e consultar tipos de dados variados com facilidade. Isso não apenas simplifica o gerenciamento do banco de dados para os desenvolvedores, como também cria um ambiente altamente escalável para aplicativos e serviços multiplataforma.

Os documentos ou coleções de documentos do MongoDB são as unidades básicas

de dados. Formatados como Binary JSON (Java Script Object Notation), esses documentos podem armazenar vários tipos de dados e ser distribuídos para diversos sistemas. Como o MongoDB emprega um design de esquema dinâmico, os usuários têm uma flexibilidade incomparável ao criar registros de dados, consultar coleções de documento por meio da agregação do MongoDB e analisar grandes quantidades de informações (IBM, 2024).

Swagger: Swagger é uma ferramenta poderosa para o design, construção e documentação de APIs RESTful.

Trata-se de uma aplicação open source que auxilia desenvolvedores nos processos de definir, criar, documentar e consumir APIs REST. Em suma, o Swagger visa padronizar este tipo de integração, descrevendo os recursos que uma API deve possuir, como endpoints, dados recebidos, dados retornados, códigos HTTP e métodos de autenticação, entre outros.

Ele simplifica o processo de escrever APIs, especificando os padrões e fornecendo as ferramentas necessárias para escrever APIs seguras, com alto desempenho e escaláveis.

No mundo do software de hoje, não há sistemas rodando on-line sem expor uma API. Passamos de sistemas monolíticos para microsserviços. E todo o design de microsserviços é baseado em APIs REST (GR1D, 2022).

Kotlin é uma linguagem de programação moderna que foi oficialmente reconhecida como uma linguagem de primeira classe para o desenvolvimento de aplicativos Android pela Google em 2017. Criada pela JetBrains, Kotlin é uma linguagem estática, que combina características de programação orientada a objetos e programação funcional. Sua sintaxe é concisa e expressiva, o que permite aos desenvolvedores escreverem código de forma mais eficiente e legível, reduzindo o número de linhas de código necessárias em comparação com Java, o que a torna ideal para o desenvolvimento de aplicativos móveis.

Uma das principais características do Kotlin é a interoperabilidade com Java, permitindo que desenvolvedores utilizem bibliotecas Java existentes sem a necessidade de reescrever código. Essa característica tem sido crucial para a adoção do Kotlin em projetos Android, pois permite uma transição suave para equipes que já têm um histórico de uso de Java. Além disso, o Kotlin oferece recursos avançados, como corrotinas, que facilitam a programação assíncrona e melhoram a experiência do desenvolvedor ao lidar com tarefas de longa duração, como chamadas de rede ou operações de entrada e saída (GOOGLE, 2024).

Jetpack Compose: O Jetpack Compose é um toolkit moderno para criação de interface nativa do Android. Ele simplifica e acelera o desenvolvimento de IUs no Android com menos código, ferramentas com a mais alta tecnologia e APIs Kotlin intuitivas (Android Developers, 2024).

A principal vantagem do desenvolvimento em Jetpack é que o fluxo de criar a UI é intuitivo e acelerado, com muito menos código do que com o XML. O framework ainda

permite a interoperabilidade com o framework nativo do Android, ou seja, o Compose pode ser adotado no nativo assim como componentes do nativo podem ser usados no Compose (Medium, 2024).

Aplicativos Móveis: Aplicações são conjuntos de instruções desenvolvidas para cumprir um objetivo específico, funcionando como programas de computador. A mobilidade agrega fatores relevantes, permitindo ao usuário uma interação mais dinâmica e presente em qualquer lugar. Atualmente, vivemos na era das multitarefas, onde não se considera mais ficar impossibilitado de realizar outra atividade enquanto, por exemplo, o carro está sendo lavado. Por exemplo, não é mais aceitável justificar o atraso no envio de um documento por ter ficado preso no trânsito.

Os aplicativos são normalmente conhecidos como “apps” ou “app mobile”. A sigla “app” é uma abreviatura de “aplicação de software”. “Em 2010, o termo se tornou tão popular que foi assinalado como ‘palavra do ano’ pela Sociedade Americana de Dialeto” (FERNANDES, 2016).

Atualmente, os aplicativos fazem cada vez mais parte da rotina das pessoas. Eles podem ser usados para diversas finalidades, como chamar um táxi, pedir uma refeição, baixar músicas e assistir a vídeos. Sem dúvida, são uma ótima opção para tornar a vida mais prática e encontrar opções de lazer com facilidade (Prodest, 2023).

Segundo Mandel (2017), “cada vez mais indivíduos estarão utilizando aplicativos através dos celulares para a interface das suas casas, suas viagens, seus entretenimentos, seus carros, suas escolas, seus provedores de saúde e seus governos estaduais e locais”.

Hoje em dia se encontram os mais variados aplicativos de inúmeras categorias. Pode-se utilizar um aplicativo para pedir comida no conforto de sua casa, como o iFood, ou até mesmo aplicativos para locomoção, como o Uber. Basta entrar no aplicativo, traçar uma rota e um carro virá te buscar. A versatilidade dessas aplicações transforma a forma como interagimos com o mundo, tornando as atividades do dia a dia mais simples e eficientes.

Em suma, os aplicativos móveis desempenham um papel vital na modernidade, oferecendo soluções que não apenas atendem às necessidades diárias, mas também moldam comportamentos e expectativas. À medida que a tecnologia avança, a integração e a inovação nas funcionalidades dos aplicativos continuarão a expandir, reafirmando sua importância na vida contemporânea e na forma como nos conectamos e interagimos com o mundo ao nosso redor.

3.3 Requisitos

Os requisitos funcionais e não funcionais foram definidos nas etapas iniciais do projeto, com base no problema identificado e nas necessidades observadas durante a fase

de ideação. As informações coletadas permitiram direcionar o desenvolvimento e esclarecer o caminho a ser seguido no levantamento de cada requisito.

Cada requisito contribui diretamente para que o aplicativo se consolide como uma solução viável e eficaz ao problema proposto, garantindo não apenas o funcionamento das principais funcionalidades, mas também a qualidade, desempenho e usabilidade esperadas.

Tabela 1 – Requisitos funcionais do Gooaling

ID	**História de usuário**	**Descrição**
RF01	Como usuário, quero me cadastrar informando nome, e-mail e senha.	Permitir o cadastro de usuários com verificação de unicidade do e-mail e armazenamento seguro da senha (hash).
RF02	Como usuário, quero fazer login com meu e-mail e senha.	Autenticar credenciais e iniciar sessão com token, respeitando expiração.
RF03	Como usuário, quero recuperar minha senha se eu esquecê-la.	Gerar `tokenRecuperarSenha` com `tempoExpiracaoToken` e permitir redefinição de senha com validações.
RF04	Como usuário, quero editar meu perfil (nome e senha).	Atualizar dados do usuário com validação e confirmação de senha ao alterá-la.
RF05	Como usuário, quero excluir minha conta com segurança.	Remover conta mediante confirmação dupla e checagens de vínculos (metas/contas), preservando a integridade dos dados.
RF06	Como usuário, quero cadastrar uma conta espelhada (sem conexão bancária).	Criar conta com `nome_conta`, `tipo_conta`, `inst_financeira`, `cor_inst_financeira`, `url_img`, `valor_conta` e `conta_principal`.
RF07	Como usuário, quero editar e excluir minhas contas espelhadas.	Atualizar campos da conta e remover contas respeitando regras (ex.: metas vinculadas).
RF08	Como usuário, quero visualizar todas as minhas contas e seus saldos.	Listar contas do usuário com paginação, ordenação e detalhe por conta.
RF09	Como usuário, quero definir uma conta principal.	Marcar `conta_principal = true` e desmarcar outra previamente principal do mesmo usuário.
RF10	Como usuário, não quero que meu saldo fique negativo.	Bloquear operações que resultem em `valor_conta < 0`; validar na criação/edição e nos lançamentos.

Continuação na próxima página

ID	História de usuário	Descrição
RF11	Como usuário, quero registrar ajustes de saldo manuais (depósitos/retiradas) na conta.	Criar/registrar entrada/saída manual com histórico básico para auditoria.
RF12	Como usuário, quero ver o saldo total somando todas as minhas contas.	Exibir somatório de `valor_conta` das contas do usuário em tempo real.
RF13	Como usuário, quero criar uma meta financeira.	Cadastrar meta com `tipo_meta`, `nome_meta`, `data_inicial_meta`, `data_final_meta`, `valor_meta`, `valor_guardado` inicial e `conta` vinculada, calculando campos derivados (dias/meses totais e restantes, porcentagem).
RF14	Como usuário, quero editar e excluir minhas metas.	Atualizar dados da meta e excluir com validações de integridade (ex.: regras de reversão de valores na conta).
RF15	Como usuário, quero listar e ver detalhes de cada meta.	Exibir lista com filtros básicos e tela de detalhes com progresso, valores e histórico.
RF16	Como usuário, quero registrar um aporte em uma meta.	Inserir item em `valor_arrecadar_mes` (valor, data, `tipo_arrecadamento`, descrição), atualizar `valor_guardado`, `porcentagem_avanco_meta` e ajustar `valor_conta` da conta vinculada.
RF17	Como usuário, quero editar ou excluir um aporte registrado.	Atualizar/remover item de `valor_arrecadar_mes` recalculando `valor_guardado`, porcentagem e saldo de conta.
RF18	Como usuário, quero que o progresso da meta seja recalculado automaticamente.	Reprocessar `porcentagem_avanco_meta`, dias/meses restantes e `status` a cada alteração relevante.
RF19	Como usuário, quero alterar o status da meta (em andamento, pausada, concluída, cancelada).	Mudar `status` conforme regras; concluir automaticamente quando `valor_guardado >= valor_meta`.
RF20	Como usuário, quero transferir valores de arrecadação entre duas metas minhas.	Transferência entre metas do mesmo usuário, gerando registros nos dois históricos e ajustes nos saldos/porcentagens.

Continuação na próxima página

ID	História de usuário	Descrição
RF21	Como usuário, quero simular o impacto de um aporte antes de confirmar.	Exibir preview do efeito no `valor_guardado`, porcentagem e `valor_conta` sem persistir até confirmar.
RF22	Como usuário, quero um dashboard com gráficos para acompanhar minha evolução.	Exibir evolução de saldos por conta, aportes por mês, avanço por meta e distribuição por tipo.
RF23	Como usuário, quero insights automáticos que me ajudem a decidir.	Destacar ritmo atual vs recomendado, risco de atraso, meses com melhor desempenho e conta mais utilizada.
RF24	Como usuário, quero ver um mapa de progresso visual das minhas metas.	Exibir trilha com marcos alcançados e próximos (estilo “Duolingo-like”), com tema Gooaling.
RF25	Como usuário, quero um onboarding que explique rapidamente o app.	Apresentar conceito de contas espelhadas, metas e mapa de progresso com navegação guiada inicial.

Tabela 2 – Requisitos Não Funcionais do Gooaling (Aplicativo Android)

ID	Requisito não funcional	Descrição
RNF01	O aplicativo deve ser nativo para Android usando Kotlin e Jetpack Compose.	Define a stack oficial (AndroidX/Material 3), garantindo compatibilidade, performance e adoção das diretrizes do Jetpack.
RNF02	A arquitetura deve seguir MVVM.	Separação de camadas (domínio, dados e apresentação) e testabilidade dos componentes.
RNF03	Privacidade deve seguir LGPD.	Minimização de dados, consentimento quando necessário, termos de uso, portabilidade e exclusão sob solicitação do titular.
RNF04	O app deve ser resiliente a conexões instáveis.	Cache local para consultas recentes, reconciliação de alterações quando a rede retornar e mensagens de erro claras.

Continuação na próxima página

ID	Requisito não funcional	Descrição
RNF05	Cálculos financeiros devem ser precisos.	Utilizar BigDecimal ou inteiros em centavos e arredondamento bancário; formatação monetária em pt-BR.
RNF06	Internacionalização e tempo devem ser consistentes.	Formatação pt-BR por padrão; timezone `America_Porto_Velho` para exibição; armazenamento de horários em UTC no backend.
RNF07	Compatibilidade com versões do Android.	Suporte mínimo Android 8.0 (API 26) ou superior, múltiplas densidades, tema claro/escuro (dark mode).
RNF08	Qualidade deve ser assegurada por testes.	Testes unitários (regras de negócio), instrumentados/UI (compose-ui-test) e E2E para fluxos críticos, com metas de cobertura no domínio.
RNF09	UX deve ser estável e consistente.	Feedbacks claros (toasts/snackbars), skeletons, estados de loading sem flicker, animações com duração adequada e navegação previsível.

3.4 Arquitetura de software

A arquitetura do aplicativo segue o padrão *Model–View–ViewModel* (MVVM), recomendado pela própria documentação do Android (DEVELOPERS, 2025b), dividido em três partes:

- **View (a tela):** é tudo o que o usuário vê e toca. São as telas, textos, botões e listas.
- **ViewModel (o "intérprete"):** recebe as ações do usuário (toques e escolhas), decide o que fazer e envia para a tela apenas o que precisa ser mostrado. Ele guarda o **estado da tela** para que, mesmo mudando a orientação do celular, o app não "perca" o que estava sendo exibido (DEVELOPERS, 2025d).
- **Model (os dados e as regras):** é onde ficam as regras do negócio e os dados (o que vem da internet e o que está salvo no aparelho). Ele não sabe nada sobre a tela; apenas fornece e guarda informações.

O aplicativo utiliza Jetpack Compose. Conforme a documentação oficial:

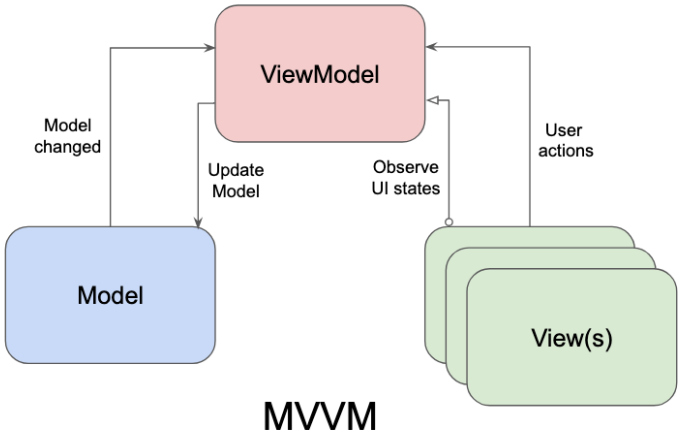


Figura 6 – Padrão Model–View–ViewModel. Fonte: CWI Software (2023)

> O Jetpack Compose é o kit de ferramentas moderno recomendado pelo Android para a criação de interfaces de usuário nativas. Ele simplifica e acelera o desenvolvimento de interfaces de usuário no Android. Dê vida ao seu aplicativo rapidamente com menos código, ferramentas poderosas e APIs Kotlin intuitivas. (DEVELOPERS, 2025c)

Essa arquitetura foi escolhida porque sua ideia central é simples: a tela sempre reflete o estado. Para isso, o "estado da tela" é organizado no *ViewModel*, enquanto a tela observa esse estado e se atualiza conforme necessário. De acordo com Morua (2022), no Jetpack Compose cada elemento da interface é criado por meio de funções chamadas *composables*, responsáveis por descrever como a tela deve ser exibida e reagir automaticamente às mudanças de estado. Sempre que o estado é alterado, o Compose recompõe apenas os elementos visuais necessários, mantendo a interface sincronizada e eficiente.

3.4.0.1 O que cada camada faz no Gooaling

A arquitetura do Gooaling foi estruturada em três camadas principais: **View**, **ViewModel** e **Model**. Cada uma possui um papel bem definido dentro do funcionamento do aplicativo, permitindo que o sistema seja mais organizado, fácil de manter e com menor chance de erros.

A **View** representa as telas do aplicativo, ou seja, é a parte que o usuário vê e interage diretamente. Nela estão presentes os *composables* do Jetpack Compose, como botões, campos de texto, cartões e listas. Essa camada tem a função de exibir as informações que vêm do ViewModel, como o progresso de uma meta ou o saldo de uma conta, e também de capturar as ações do usuário por exemplo, quando ele toca em um botão para adicionar um novo aporte ou cadastrar uma conta. A View não realiza cálculos nem acessa

dados diretamente; ela apenas mostra o que deve ser exibido e informa ao ViewModel o que o usuário deseja fazer.

O **ViewModel** atua como o elo de ligação entre a tela e a camada de dados. Ele recebe as ações enviadas pela View, interpreta o que precisa ser feito e coordena as operações necessárias, como buscar informações, validar regras e salvar dados. Além disso, o ViewModel monta e mantém o estado da tela, garantindo que todas as informações apresentadas estejam atualizadas. Mesmo que o usuário gire o celular ou o aplicativo passe por uma mudança de configuração, o ViewModel preserva o estado atual, evitando que a tela seja recarregada ou que dados sejam perdidos (DEVELOPERS, 2025d). Essa camada é o "cérebro" da interface, garantindo que tudo aconteça de forma fluida e reativa.

Por fim, o **Model** é a camada responsável por intermediar a comunicação com a API e gerenciar os dados locais. É nela que o aplicativo envia e recebe informações do servidor, como os cadastros de contas, metas e aportes. Após realizar a requisição inicial, o Model armazena os dados retornados em cache local, exibindo-os diretamente ao usuário nas próximas consultas. Dessa forma, o aplicativo evita requisições desnecessárias à API, realizando novas chamadas apenas quando há inserção, atualização ou exclusão de informações.

Sempre que uma alteração ocorre, um evento é disparado internamente, notificando as demais telas para que realizem uma nova sincronização com a API e atualizem seus dados. Esse comportamento garante que o usuário visualize sempre as informações mais recentes, sem comprometer o desempenho ou a experiência de uso.

```
object AppEvents {

    private val _metasChanged = MutableStateFlow( value = 0L)  3 Usages
    val metasChanged: StateFlow<Long> = _metasChanged  2 Usages
    fun notifyMetasChanged() { _metasChanged.update { it + 1 } }  12 Usages

    private val _contasChanged = MutableStateFlow( value = 0L)  3 Usages
    val contasChanged: StateFlow<Long> = _contasChanged  2 Usages
    fun notifyContasChanged() { _contasChanged.update { it + 1 } }  12 Usages

    fun clearAll() {  2 Usages
        _metasChanged.value = 0L
        _contasChanged.value = 0L
    }
}
```

Figura 7 – Objeto responsável por disparar eventos globais no aplicativo. Fonte: o autor.

```
// Aqui é a lógica para capturar mudanças no aplicativo e refazer a req na API.
val contasChanged by AppEvents.contasChanged.collectAsStateWithLifecycle()
val metasChanged by AppEvents.metasChanged.collectAsStateWithLifecycle()

var lastHandledContasTick by rememberSaveable { mutableStateOf<Long?>( value = null) }
var lastHandledMetasTick  by rememberSaveable { mutableStateOf<Long?>( value = null) }

// Notificação de mudanças em contas
LaunchedEffect( key1 = contasChanged) {
    if (contasChanged > 0 && lastHandledContasTick != contasChanged) {
        lastHandledContasTick = contasChanged
        contaViewModel.getContas( pagina = 1,  limite = 10, forceRefresh = true)
        metaViewModel.getMetas( pagina = 1,  limite = 10, forceRefresh = true)
        metaViewModel.ultimasTransacoes()
    }
}
// Notificação de mudança em metas
LaunchedEffect( key1 = metasChanged) {
    if (metasChanged > 0 && lastHandledMetasTick != metasChanged) {
        lastHandledMetasTick = metasChanged
        metaViewModel.getMetas( pagina = 1,  limite = 10, forceRefresh = true)
        contaViewModel.getContas( pagina = 1,  limite = 10, forceRefresh = true)
        metaViewModel.ultimasTransacoes()
    }
}
```

Figura 8 – HomeScreen lidando com os eventos. Fonte: o autor.

Para o armazenamento local, o Gooaling utiliza o **Jetpack DataStore**, tecnologia oficial do Android para salvar preferências e dados leves de forma segura e confiável (DEVELOPERS, 2025a).

Com essa separação de responsabilidades, cada camada do Gooaling cumpre um papel específico e independente. A View mostra, o ViewModel coordena e o Model guarda. Essa estrutura facilita a manutenção, os testes e futuras expansões do aplicativo, além de seguir as boas práticas recomendadas pela documentação oficial do Android.

3.4.0.2 Organização de pastas do aplicativo

A Figura 11 mostra como o projeto do aplicativo foi organizado, seguindo a ideia do MVVM:

- `ui/`: tudo que é tela.
  - `screens/`: cada arquivo é uma tela do aplicativo.
  - `components/`: componentes de tela que são reaproveitados.
  - `theme/`: cores, fontes e estilos do aplicativo.
  - `utils/`: funções simples que ajudam a montar a UI.
- `viewModel/`: um ViewModel por módulo , guardando o estado e a lógica daquela de cada módulo.

- `api/`: é responsável pela comunicação com o backend. Nela estão definidos os serviços e os métodos responsáveis por enviar e buscar informações do servidor. Essa pasta reflete a integração do aplicativo com a API central, que contém as regras de negócio do sistema.
- `datastore/`: é destinada ao armazenamento local de informações metas, contas e os aportes, evitando realizar inúmeras chamadas desnecessárias para a API.

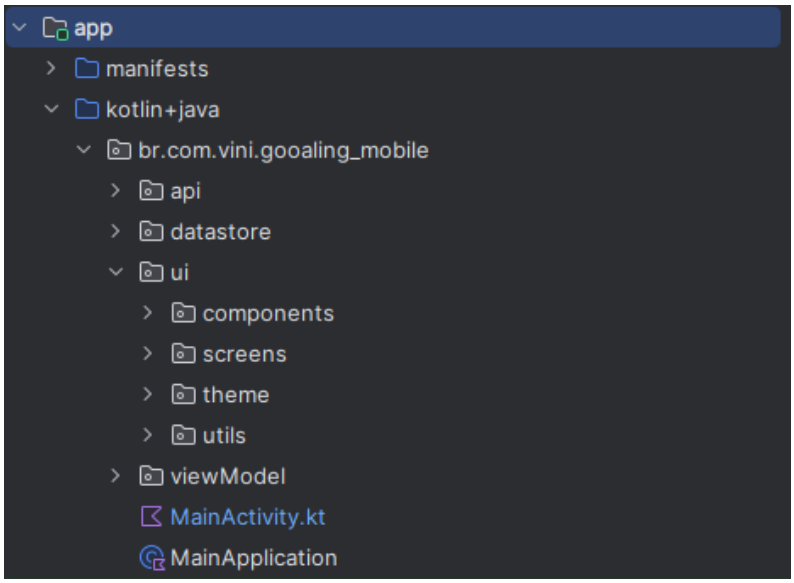


Figura 9 – Estrutura de pastas do projeto Gooaling. Fonte: o autor.

3.5 Modelagem

Na fase de **Iniciação**, foi desenvolvido o diagrama de classe e o diagrama de atividade. O diagrama de classes representa a estrutura estática do domínio, especificando classes, atributos e relacionamentos.

No contexto do aplicativo, foram modeladas as entidades *Usuário*, *Conta*, *Meta* e o subdocumento *ValorArrecadarMes*, com multiplicidades que expressam as regras do negócio (por exemplo, um usuário pode possuir várias contas e metas; uma meta pertence a uma conta e agrega vários registros de arrecadação). Esse artefato serviu de guia para o mapeamento em *MongoDB/Mongoose* e para a definição de validações e consistência referencial no código.

Já o **diagrama de atividades** visou descrever o comportamento do aplicativo ao longo de um fluxo de uso, evidenciando ações, decisões e resultados. A figura abaixo ilustra um fluxo básico de interação do usuário com o aplicativo, desde o login até o registro de um valor para guardar ou a criação de uma meta financeira.

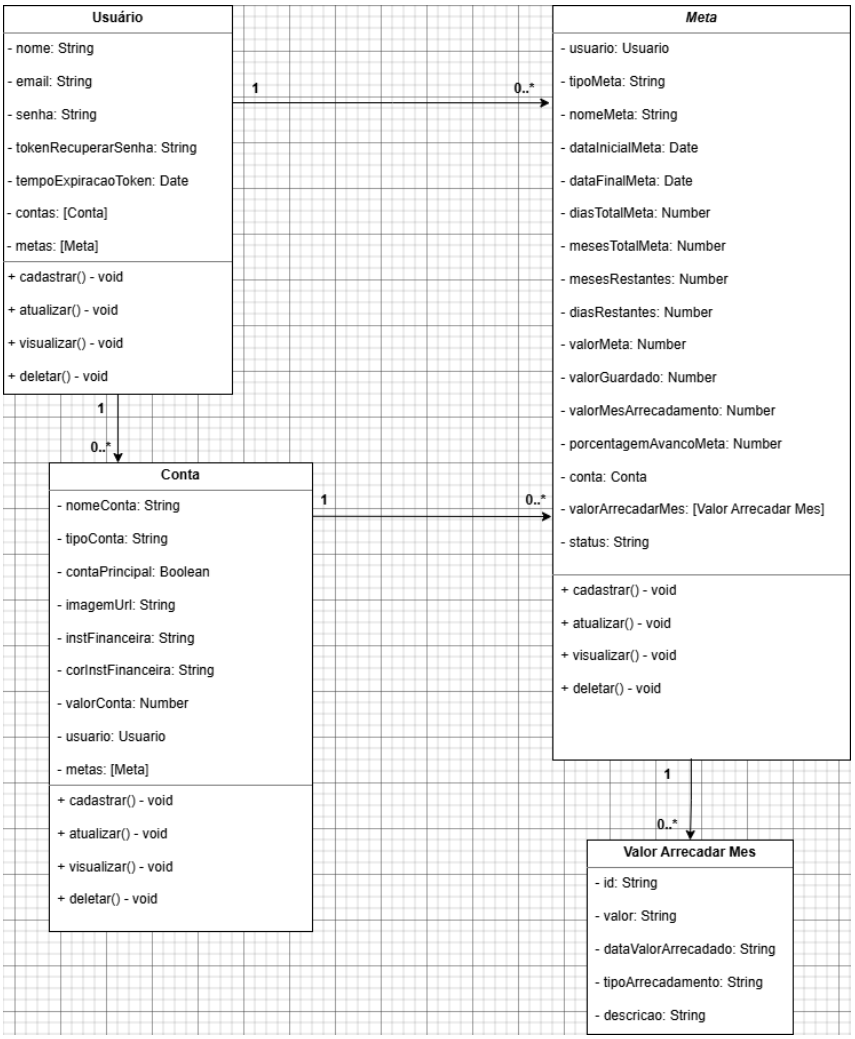


Figura 10 – Fonte: elaborado pelo autor (2025).

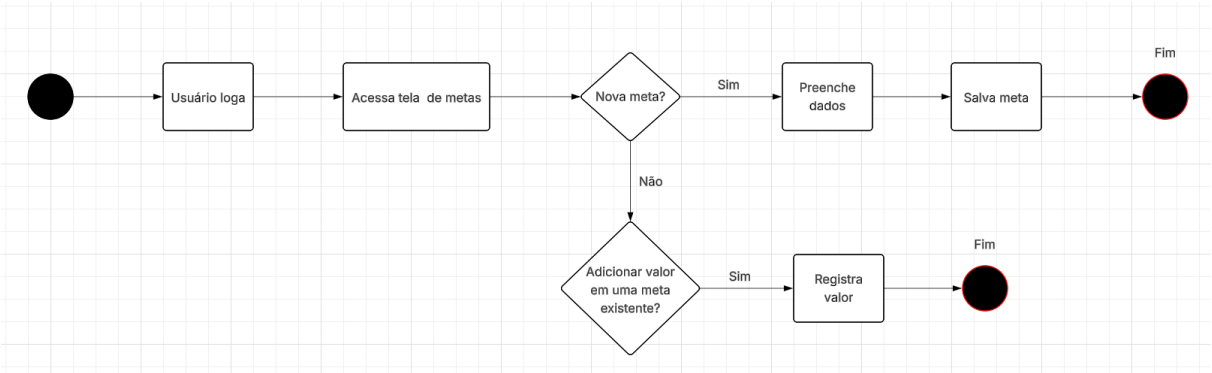


Figura 11 – Fonte: elaborado pelo autor (2025).

3.6 Persistência de dados

O banco de dados NoSQL **MongoDB**[8] foi adotado para o armazenamento das informações do aplicativo, por oferecer uma estrutura flexível que se adapta a diferentes tipos de dados e volumes de informação. Essa característica é especialmente útil em aplicações que lidam com entidades de natureza diversa, como usuários, metas e contas, cujos relacionamentos e atributos podem variar conforme a necessidade do sistema.

A conexão entre a API e o banco de dados foi implementada com o auxílio da biblioteca **Mongoose**[9], que possibilita a definição de esquemas (schemas) e a aplicação de validações diretamente no nível da modelagem. Essa abordagem garante maior consistência nos dados armazenados, pois cada campo é verificado antes de ser inserido ou atualizado no banco. Além disso, o Mongoose simplifica a manipulação de documentos e coleções, oferecendo uma camada de abstração que facilita operações como consultas, inserções, atualizações e exclusões.

No contexto do aplicativo, o Mongoose foi empregado para estruturar os modelos responsáveis por representar as principais entidades do sistema, sendo elas Usuário, Metas e Contas, bem como as rotas de autenticação e recuperação de senha. Essa organização foi feita para garantir um código mais modular e coerente. A modelagem dos dados reflete diretamente as regras e interações do domínio da aplicação. Abaixo, estão os esquemas (schemas) usados para o aplicativo.

3.7 Licença de Uso

A licença escolhida para este projeto foi a **MIT License**[10], originalmente desenvolvida pelo Massachusetts Institute of Technology (MIT). Trata-se de uma licença amplamente adotada em projetos de código aberto devido à sua redação simples e caráter permissivo.

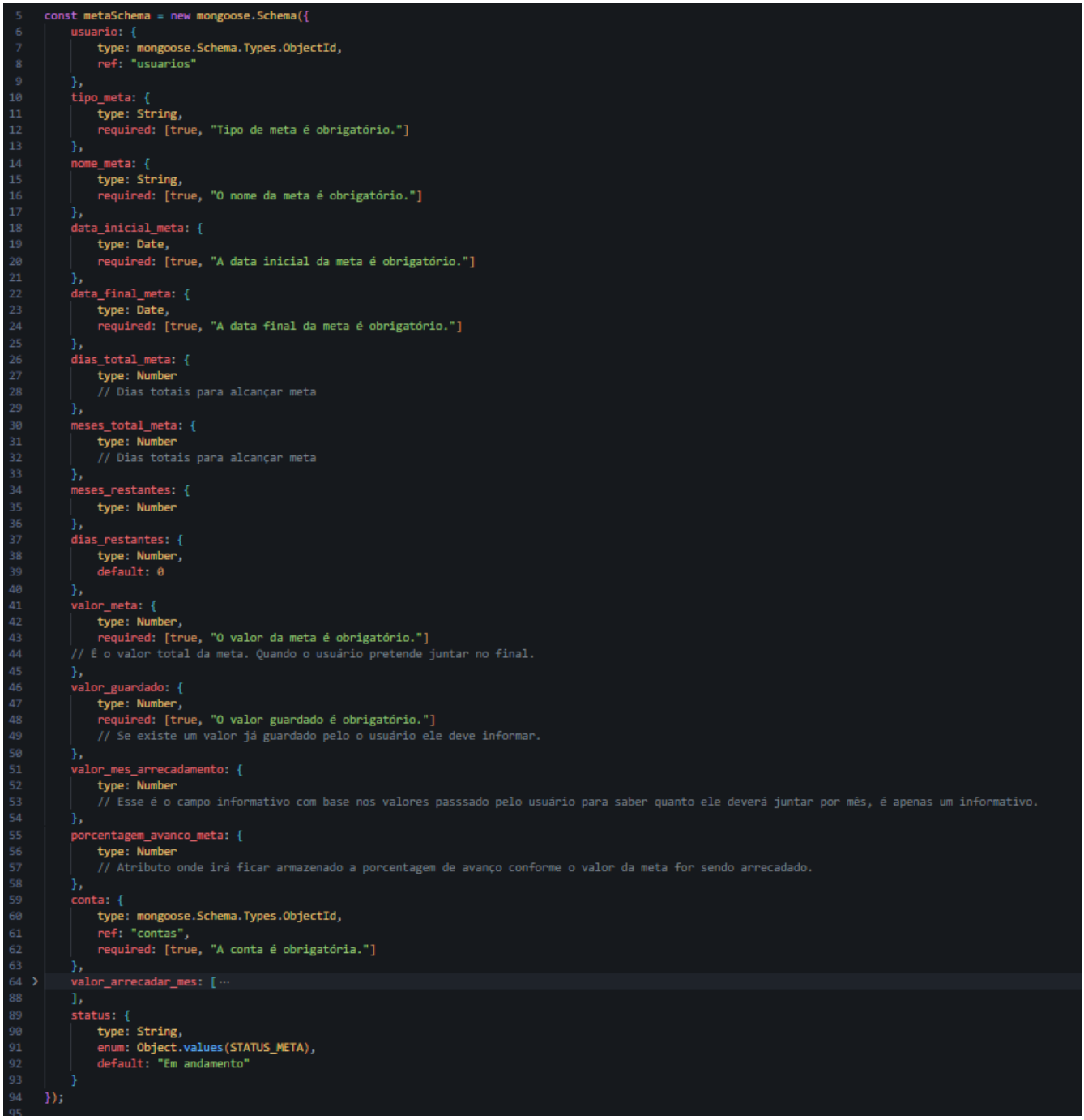
[8] Disponível em: <https://www.mongodb.com>
[9] Disponível em: <https://www.mongodb.com/pt-br/docs/drivers/node/current/integrations/mongoose-get-started/>
[10] Disponível em: <https://mit-license.org/>

Figura 12 – Schema de usuário

```
const usuarioSchema = new mongoose.Schema({
    nome: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    senha: {
        type: String,
        required: true
    },
    tokenRecuperarSenha: {
        type: String,
        select: false
    },
    tempoExpiracaoToken: {
        type: Date,
        select: false
    },
    contas: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "contas"
        }
    ],
    metas: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "metas"
        },
    ]
});
```

Fonte: elaborado pelo autor (2025)

Figura 13 – Schema de metas



```
const metaSchema = new mongoose.Schema({
    usuario: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "usuarios"
    },
    tipo_meta: {
        type: String,
        required: [true, "Tipo de meta é obrigatório."]
    },
    nome_meta: {
        type: String,
        required: [true, "O nome da meta é obrigatório."]
    },
    data_inicial_meta: {
        type: Date,
        required: [true, "A data inicial da meta é obrigatório."]
    },
    data_final_meta: {
        type: Date,
        required: [true, "A data final da meta é obrigatório."]
    },
    dias_total_meta: {
        type: Number
        // Dias totais para alcançar meta
    },
    meses_total_meta: {
        type: Number
        // Dias totais para alcançar meta
    },
    meses_restantes: {
        type: Number
    },
    dias_restantes: {
        type: Number,
        default: 0
    },
    valor_meta: {
        type: Number,
        required: [true, "O valor da meta é obrigatório."]
    // É o valor total da meta. Quando o usuário pretende juntar no final.
    },
    valor_guardado: {
        type: Number,
        required: [true, "O valor guardado é obrigatório."]
        // Se existe um valor já guardado pelo o usuário ele deve informar.
    },
    valor_mes_arrecadamento: {
        type: Number
        // Esse é o campo informativo com base nos valores passsado pelo usuário para saber quanto ele deverá juntar por mês, é apenas um informativo.
    },
    porcentagem_avanco_meta: {
        type: Number
        // Atributo onde irá ficar armazenado a porcentagem de avanço conforme o valor da meta for sendo arrecadado.
    },
    conta: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "contas",
        required: [true, "A conta é obrigatória."]
    },
    valor_arrecadar_mes: [ ···
    ],
    status: {
        type: String,
        enum: Object.values(STATUS_META),
        default: "Em andamento"
    }
});
```

Fonte: elaborado pelo autor (2025)

Figura 14 – Schema de contas

```
const contaSchema = new mongoose.Schema({
    nome_conta: {
        type: String,
        required: true
    },
    tipo_conta: {
        type: String,
        enum: Object.values(TIPO_CONTA),
        required: true
    },
    conta_principal: {
        type: Boolean,
        default: false
    },
    url_img: {
        type: String
    },
    inst_financeira: {
        type: String,
        enum: Object.values(INST_FINANCEIRA),
        required: true
    },
    cor_inst_financeira: {
        type: String,
        enum: Object.values(COR_INST_FINANCEIRA)
    },
    valor_conta: {
        type: Number,
        default: 0,
        min: [0, 'O valor da conta não pode ser negativo']
    },
    usuario: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "usuarios"
    },
    metas: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "metas",
        }
    ]
});
```

Fonte: elaborado pelo autor (2025)

4 Resultados

O resultado do projeto foi a criação de um aplicativo móvel funcional que oferece aos usuários uma plataforma intuitiva para o gerenciamento de suas metas financeiras. O aplicativo permite a definição de metas, o acompanhamento do progresso e a visualização de indicadores por meio de dashboards interativos, que apresentam de forma clara e organizada o desempenho e a evolução de cada meta ao longo do tempo.

4.1 Gerenciamento de configuração e mudanças

O controle do código-fonte é um dos elementos centrais no desenvolvimento de um software, pois assegura a integridade e a rastreabilidade do projeto ao longo de todo o seu ciclo de vida. A adoção de um processo de versionamento bem estruturado é indispensável mesmo em trabalhos individuais, uma vez que reduz o risco de perda de alterações e de falhas decorrentes de um gerenciamento inadequado. Nesse contexto, o uso de um sistema de controle de versões robusto, como o Git, mostrou-se uma escolha estratégica para a condução bem-sucedida deste projeto. Nesse viés a plataforma GitLab foi escolhida, para esse projeto.

4.1.0.1 GitLab

O **GitLab**[1] é uma plataforma online que utiliza o Git para hospedar e organizar projetos de software. Ele permite armazenar o código em repositórios, acompanhar o histórico de versões e colaborar em equipe com segurança. Também oferece recursos para registrar tarefas (*issues*) e revisar alterações antes de integrá-las ao projeto por meio de *Merge Requests*.

Neste trabalho, o GitLab foi utilizado para centralizar o código da API e do aplicativo móvel em dois repositórios independentes. Apesar dessa separação, a estratégia de versionamento adotada foi a mesma em ambos os projetos, conforme apresentado a seguir:

- **Back End (API):** https://gitlab.fslab.dev/Vinicius51/gooaling-api-tcc
- **Mobile (Aplicativo):** https://gitlab.fslab.dev/gooaling-tcc/gooaling-mobile

A estrutura de ramificações (*branches*) foi organizada da seguinte forma:

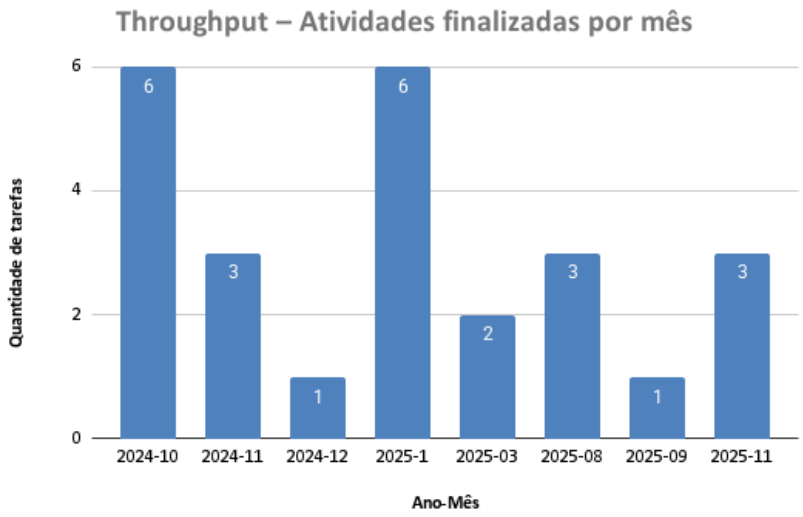
[1] Disponível em: <https://about.gitlab.com>

- **master:** Ramo principal protegido que representa a versão final e estável do sistema, destinada à disponibilização em produção. Nenhuma modificação é realizada diretamente nesse ramo, garantindo a integridade do código liberado.
- **development:** Ramo central de desenvolvimento, responsável por concentrar as novas implementações e ajustes do sistema. Todas as funcionalidades são integradas nesse ramo antes de serem incorporadas à versão principal (*master*), garantindo que apenas o código testado e validado siga para o ambiente estável.
- **feature/nome-da-funcionalidade:** Ramos temporários derivados do *development*, utilizados para o desenvolvimento isolado de novas funcionalidades ou correções específicas. Cada tarefa é implementada em um ramo próprio, o que garante maior organização, facilita o controle de mudanças e reduz o risco de conflitos durante o processo de integração ao código principal.

4.2 Processo de desenvolvimento

Para tornar o desenvolvimento do trabalho mais claro, foram selecionados e exibidos alguns dados em formato de gráficos. As visualizações consideradas são: Lead Time (tempo total até a entrega), Cycle Time (tempo de execução do desenvolvimento) e Throughput (taxa de entregas). Esses gráficos possibilitam observar o período completo necessário para concluir uma tarefa, o intervalo dedicado ao desenvolvimento de uma atividade e o volume de entregas realizadas em cada mês. Na sequência, apresenta-se a análise das métricas obtidas.

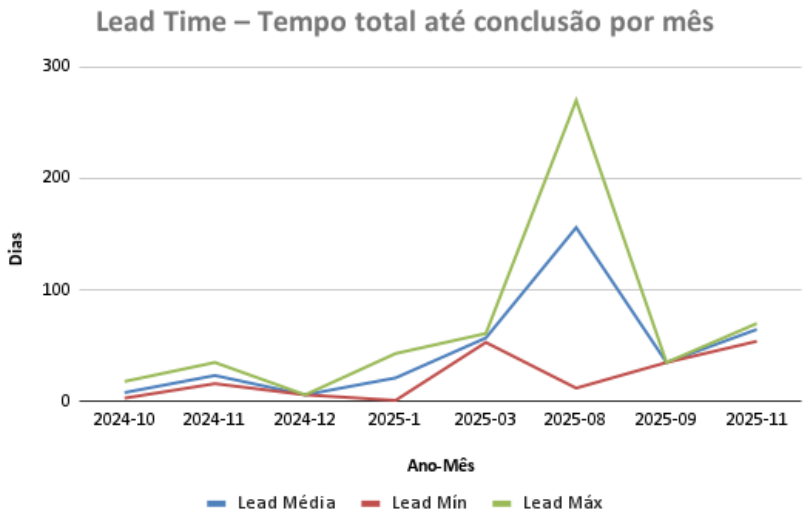
Figura 15 – Throughput



Fonte: elaborado pelo autor (2025)

O gráfico de Throughput, indica o número de atividades concluídas mês a mês, evidenciando períodos de maior volume de entregas e outros com pouca ou nenhuma finalização. Do ponto de vista do processo, essa variação não decorreu de falta de desenvolvimento, mas de dificuldades em conduzir o projeto conforme a metodologia ágil definida. Houve esforços de adaptação; porém, em diversas ocasiões, o rigor metodológico foi flexibilizado para manter o trabalho em andamento. Os gráficos seguintes evidenciam as consequências disso por exemplo, casos em que tarefas concluídas não foram devidamente registradas na ferramenta de acompanhamento e, portanto, aparecem como se tivessem permanecido em aberto por longos períodos.

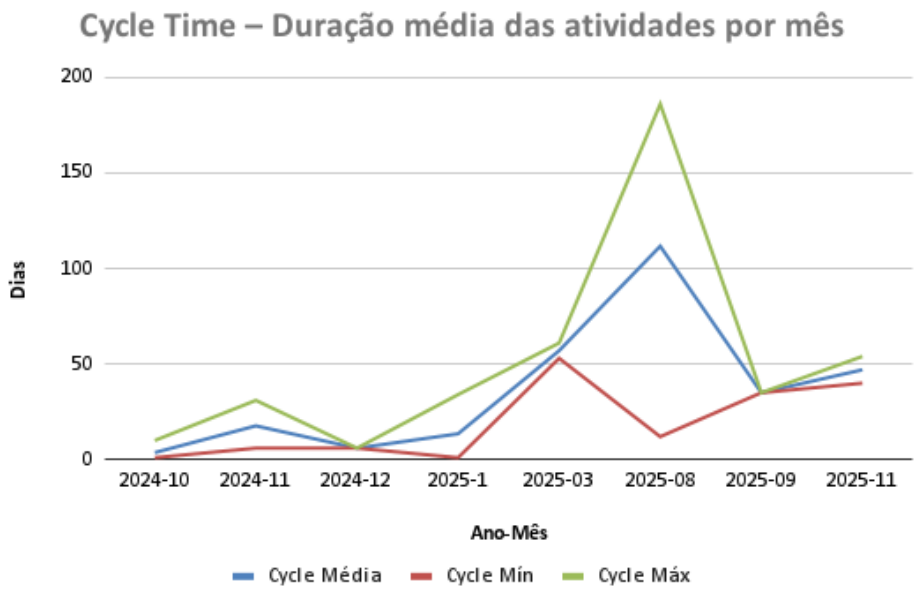
Figura 16 – Lead Time



Fonte: elaborado pelo autor (2025)

O gráfico de Lead Time apresenta, por mês de conclusão, os valores mínimo, médio e máximo do tempo total até a entrega. Observa-se um comportamento baixo a moderado entre 10/2024 e 03/2025, com leves variações. Em 08/2025 ocorre um pico pronunciado: o tempo máximo salta para patamar muito elevado e o tempo médio também sobe de forma expressiva, indicando a presença de divergências de padrões que alongaram significativamente o tempo até a conclusão naquele mês. Em 09/2025 há retorno a níveis reduzidos, e em 11/2025 os valores permanecem moderados, acima dos meses iniciais, mas muito abaixo do pico de agosto. Esse pico de 08/2025 distorce a média geral e explica por que o tempo médio de conclusão aparece inflado no agregado, apesar de a maioria dos meses indicar prazos mais contidos.

Figura 17 – Cycle Time



Fonte: elaborado pelo autor (2025)

O gráfico acima apresenta o Cycle Time, isto é, o tempo efetivo de desenvolvimento entre o início e o término de cada atividade. Nota-se um comportamento baixo a moderado entre 10/2024 e 03/2025, com oscilações suaves e um leve aumento nesse último mês. Em 08/2025 ocorre um pico acentuado: o Tempo máximo atinge o valor mais alto do período e o Tempo médio também se eleva de forma significativa, enquanto o Tempo mínimo sobe bem menos. Em 09/2025 há queda brusca para patamares reduzidos e, em 11/2025, os valores permanecem moderados, ainda acima dos meses iniciais, mas muito abaixo do pico de agosto.

Esse comportamento, alinhado ao visto no Lead Time, sugere inconsistências na aplicação da metodologia ágil e/ou no acompanhamento do fluxo, fazendo com que um mês atípico distorça a média geral — embora, na maior parte do período, as entregas tenham sido concluídas em prazos mais curtos.

4.3 Plano de testes

O plano de testes teve como objetivo assegurar a qualidade e a confiabilidade das funcionalidades desenvolvidas, garantindo que os componentes tanto no backend quanto no aplicativo mobile se comportassem conforme o esperado.

4.3.1 Testes unitários

Os testes unitários foram empregados para validar o comportamento de funções e serviços de forma isolada.

4.3.2 Testes de integração

Os testes de integração tiveram como propósito avaliar o funcionamento conjunto entre diferentes módulos da API, especialmente nas interações entre as camadas.

4.3.3 Testes de ponta a ponta (E2E)

Os testes de ponta a ponta (E2E) foram realizados com o objetivo de validar o fluxo completo de uso do aplicativo mobile, simulando a experiência real do usuário realizando login, cadastrando uma conta, uma meta e adicionando uma transação.

4.3.4 Resultados

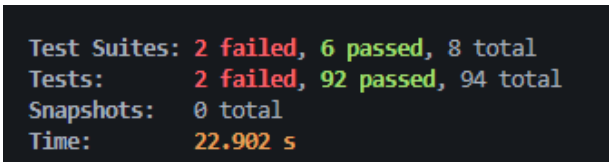
A execução dos testes permitiu validar de forma abrangente o comportamento do sistema em diferentes níveis. Os testes unitários garantiram que as funções e serviços individuais apresentassem resultados consistentes e previsíveis, reduzindo a ocorrência de falhas lógicas no código. Já os testes de integração confirmaram o correto funcionamento das interações entre os módulos da API, assegurando que o fluxo de dados entre as camadas ocorresse de maneira contínua e confiável.

No aplicativo mobile, os testes de ponta a ponta (E2E) demonstraram que os principais fluxos de navegação e uso foram executados com sucesso, sem falhas críticas. As simulações realizadas mostraram que a comunicação entre o aplicativo e a API manteve-se estável, garantindo a integridade das operações e uma experiência adequada ao usuário.

De forma geral, os resultados obtidos indicaram que o sistema atendeu aos requisitos funcionais previstos, apresentando comportamento estável e coerente nas principais operações testadas. As correções pontuais identificadas durante o processo contribuíram para o aprimoramento da aplicação, resultando em uma versão mais robusta e confiável do produto final.

A seguir, são apresentados os registros de execução dos testes unitários, de integração e de ponta a ponta (E2E), fornecendo uma análise detalhada sobre a cobertura do código, o desempenho da API e a verificação do percurso do usuário dentro do aplicativo.

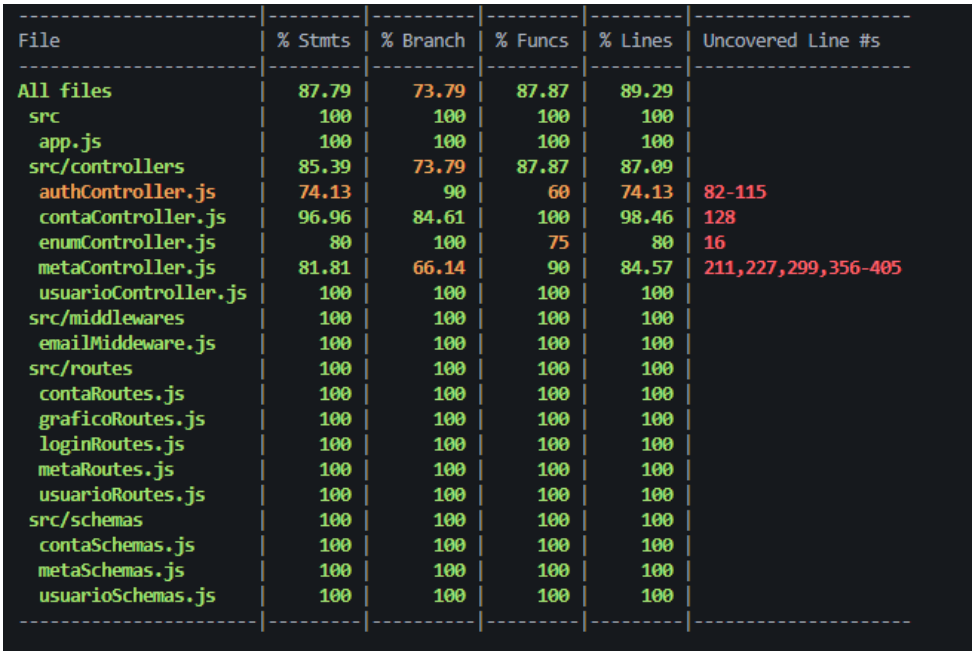
Figura 18 – Resultados da execução dos testes de integração



Fonte: elaborado pelo autor (2025)

A Figura apresenta a quantidade de testes de integração executados, totalizando 92 testes divididos em 6 suítes, sendo 92 finalizados e 2 com falhas. Isso demonstra que as rotas da API estão funcionando como o esperado e evidencia o nível de confiabilidade da API.

Figura 19 – Relatório de cobertura dos testes de integração



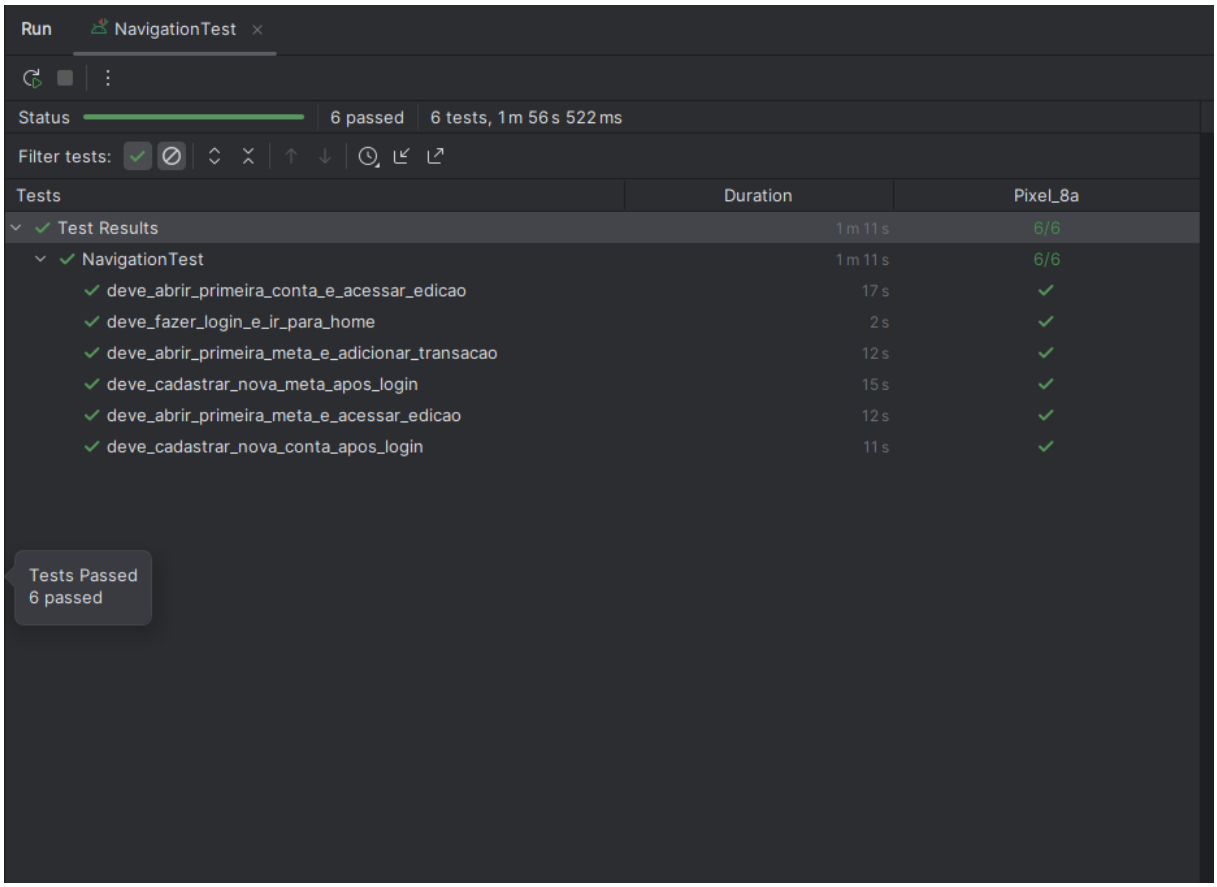
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	87.79	73.79	87.87	89.29	
src	100	100	100	100	
app.js	100	100	100	100	
src/controllers	85.39	73.79	87.87	87.09	
authController.js	74.13	90	60	74.13	82-115
contaController.js	96.96	84.61	100	98.46	128
enumController.js	80	100	75	80	16
metaController.js	81.81	66.14	90	84.57	211,227,299,356-405
usuarioController.js	100	100	100	100	
src/middlewares	100	100	100	100	
emailMiddeware.js	100	100	100	100	
src/routes	100	100	100	100	
contaRoutes.js	100	100	100	100	
graficoRoutes.js	100	100	100	100	
loginRoutes.js	100	100	100	100	
metaRoutes.js	100	100	100	100	
usuarioRoutes.js	100	100	100	100	
src/schemas	100	100	100	100	
contaSchemas.js	100	100	100	100	
metaSchemas.js	100	100	100	100	
usuarioSchemas.js	100	100	100	100	

Fonte: elaborado pelo autor (2025)

A Figura apresenta o relatório de cobertura dos testes de integração. Nele, são exibidas informações relevantes que detalham, de forma geral e por arquivo, as porcentagens de cobertura de linhas executáveis de código (%Stmts), condições de decisão (%Branch), funções testadas (%Funcs) e linhas de código cobertas (%Lines).

Ao analisar o relatório, observa-se que os testes de integração resultaram em um alto índice de cobertura da API, atingindo um total de **87,79%**, o que demonstra a efetividade das validações realizadas sobre os principais componentes do sistema.

Figura 20 – Relatório de cobertura dos testes de ponta a ponta



Fonte: elaborado pelo autor (2025)

A Figura 20 apresenta o relatório dos testes de ponta a ponta realizados. Ao analisá-lo, observa-se que todas as seis suítes de testes executadas foram concluídas com sucesso, totalizando 6 casos de teste aprovados. O tempo total de execução foi de aproximadamente 1 minuto e 11 segundos, o que indica uma boa eficiência na simulação das interações.

Os testes contemplaram fluxos essenciais do aplicativo, tais como:

- Abrir a primeira conta cadastrada e acessar a tela de edição.
- Realizar login e navegar até a página inicial.
- Abrir uma meta existente e adicionar uma transação.
- Cadastrar uma nova meta após o login.
- Cadastrar uma nova conta bancária após o login.

Os resultados demonstram que os principais fluxos do aplicativo estão funcionando conforme o esperado. O sucesso dos testes E2E valida a experiência do usuário na prática e garante que a jornada completa dentro do aplicativo ocorra sem erros ou impedimentos.

4.4 Documentação

A documentação do projeto está centralizada no repositório do backend, organizada de forma a facilitar a navegação e o entendimento das principais camadas do sistema. Nela é possível encontrar descrições detalhadas da arquitetura da aplicação, endpoints da API, fluxos para usabilidade e integração com o aplicativo mobile.

Figura 21 – Readme

Fonte: elaborado pelo autor (2025)

Figura 22 – Documentação Swagger

Fonte: elaborado pelo autor (2025)

A estrutura da documentação swagger como mostra a imagem acima reflete diretamente a organização do código-fonte do backend. As rotas da API foram divididas em categorias, agrupando funcionalidades semelhantes para uma melhor compreensão e manutenção do sistema.

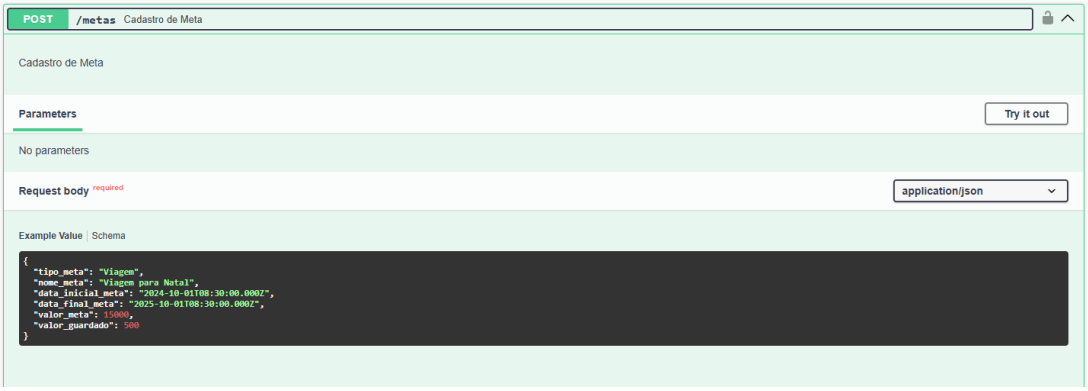
- **Login:** Responsável pela autenticação e geração de tokens de acesso do usuário.
- **Usuários:** Rotas relacionadas ao gerenciamento das contas de usuários, permitindo operações de criação, atualização e consulta de dados.
- **Contas:** Controla as informações financeiras do usuário, incluindo cadastros de contas, atualizações e listagens.
- **Metas:** Define e gerencia as metas financeiras cadastradas pelo usuário, incluindo o acompanhamento do progresso e cálculo de indicadores.

Cada uma dessas seções contém exemplos de requisições e respostas, além de descrições sobre os parâmetros esperados e os retornos possíveis. Essa abordagem não apenas facilita o uso da API por desenvolvedores externos, como também assegura uma padronização no consumo dos serviços oferecidos pelo sistema.

A seguir, são apresentadas imagens da documentação Swagger referente às rotas de metas e contas, destacando as operações disponíveis de GET, POST.

- **POST /metas:** rota responsável por criar uma nova meta no sistema.

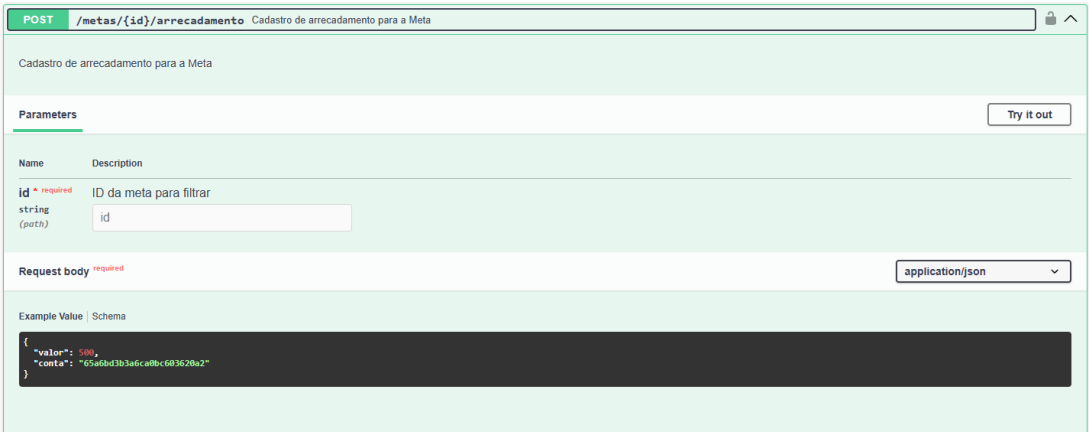
Figura 23 – Requisição POST para criação de uma meta



Fonte: elaborado pelo autor (2025)

- **POST /metas/id/arrecadamento:** rota responsável por criar um arrecadamento de uma meta no sistema.

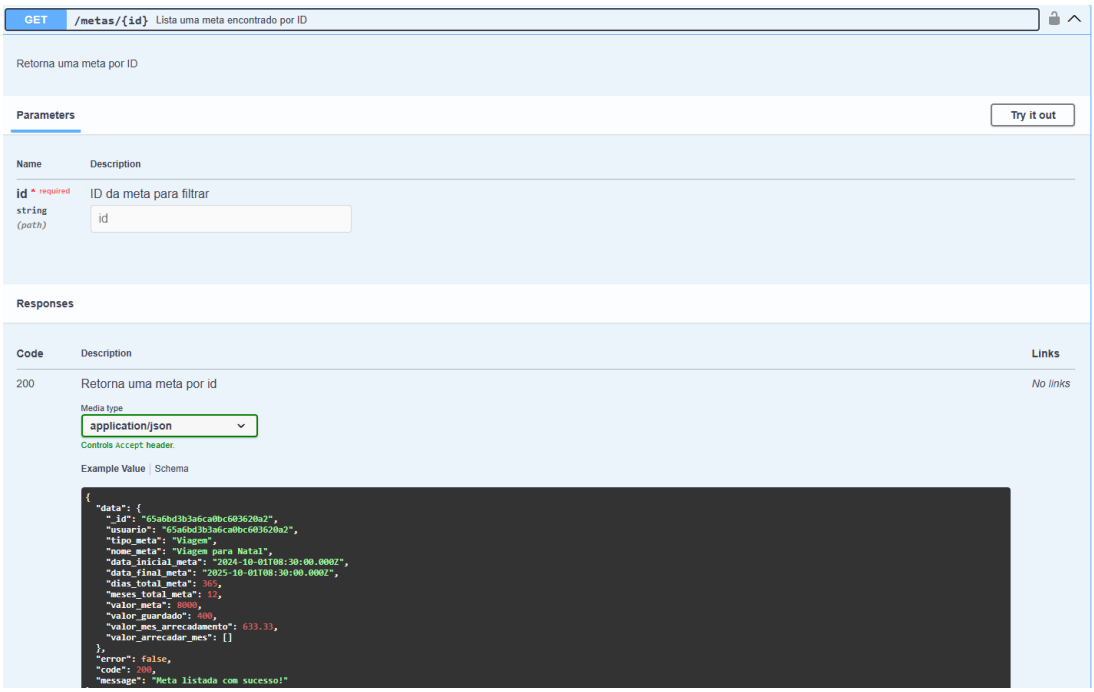
Figura 24 – Requisição POST para criar um novo arrecadamento em uma meta



Fonte: elaborado pelo autor (2025)

- **GET /metas/id:** rota responsável por listar uma meta pelo identificador dela no sistema.

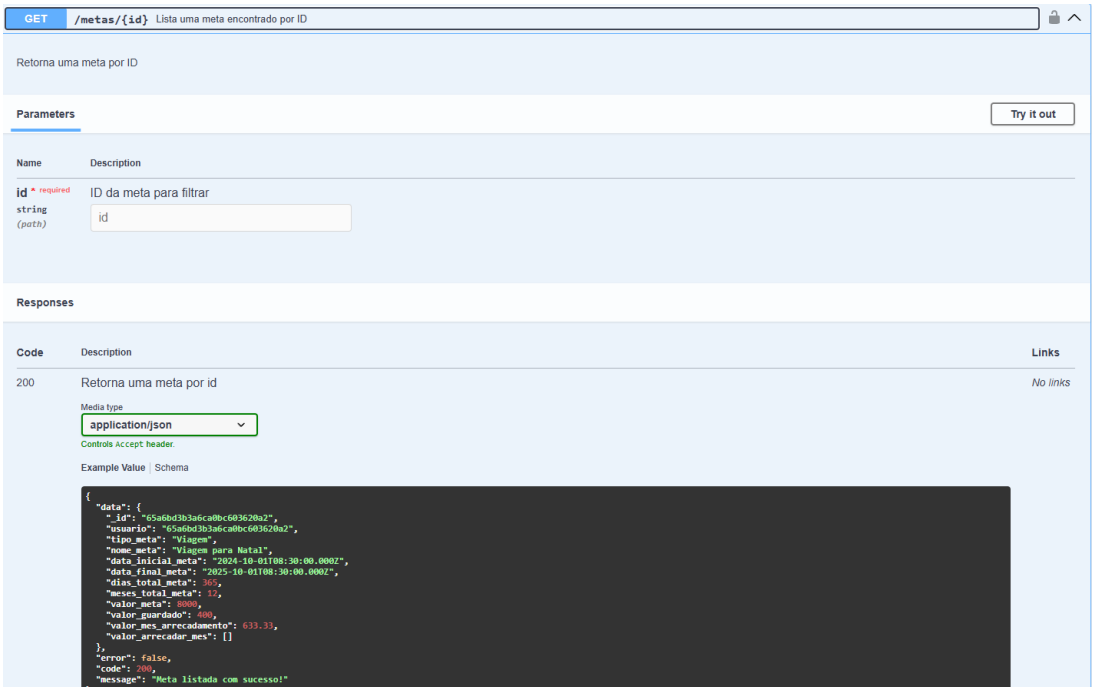
Figura 25 – Requisição GET para listar uma meta



Fonte: elaborado pelo autor (2025)

- **PATCH /contas/id:** rota responsável por atualizar uma conta existente pelo identificador dela no sistema.

Figura 26 – Requisição PATCH para atualização de uma conta



Fonte: elaborado pelo autor (2025)

4.5 Implantação

O processo de implantação da solução foi planejado de forma a assegurar alta disponibilidade, escalabilidade e facilidade de manutenção, adotando práticas atuais de Integração Contínua (CI) e Entrega Contínua (CD), aliadas à orquestração de contêineres. A API encontra-se hospedada em um servidor dedicado, sendo gerenciada por meio do Kubernetes, que coordena os serviços e recursos de forma automatizada.

O fluxo de deploy é inteiramente automatizado por um pipeline configurado no GitLab CI/CD, responsável por integrar, compilar e publicar novas versões do sistema de maneira contínua. A cada commit ou merge na branch principal, o pipeline é acionado, executando automaticamente as etapas de build, testes e implantação da aplicação nos clusters Kubernetes. Essa metodologia assegura que o ambiente permaneça atualizado, estável e pronto para escalar, reduzindo falhas humanas e acelerando o ciclo de entrega de novas funcionalidades.

4.5.1 Detalhes do acesso

- **API:** Documentação da API Gooaling
- **Aplicativo:** Download do Aplicativo Gooaling

4.5.2 Credenciais de acesso

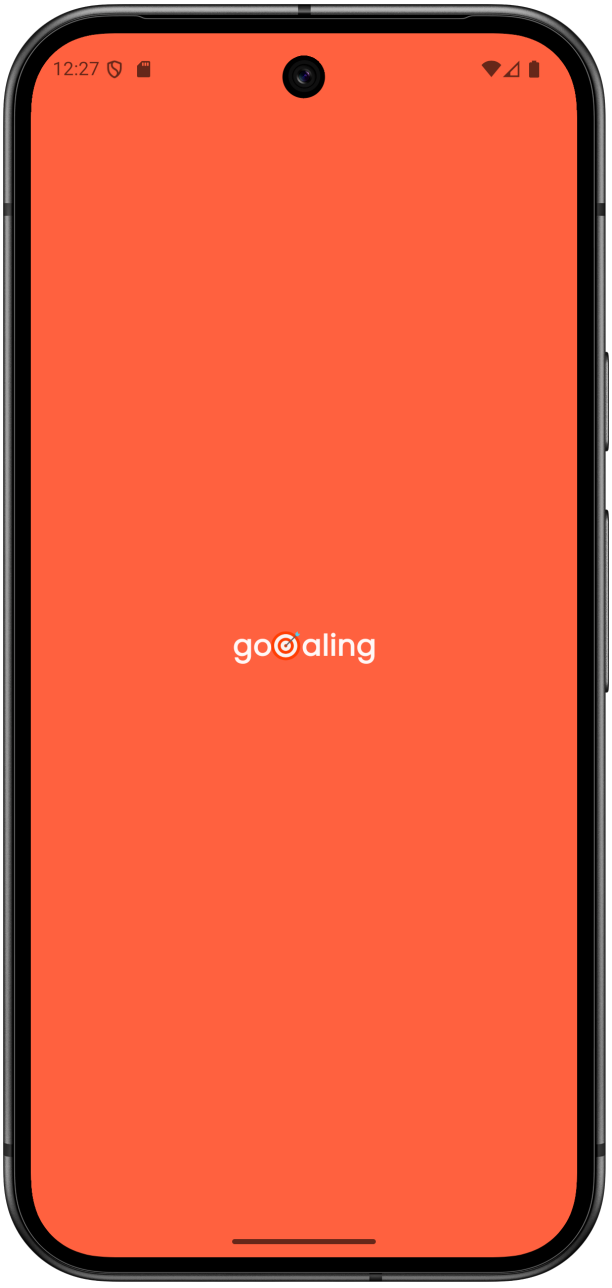
Para utilizar o aplicativo, o usuário deve acessá-lo e realizar o processo de criação de conta, informando os dados solicitados. Após o cadastro, o login pode ser efetuado com as credenciais definidas, permitindo acesso completo às funcionalidades do sistema.

4.6 Demonstração do software

Esta seção apresenta a interface e as principais funcionalidades do aplicativo **Gooaling**, descrevendo o fluxo de uso sob a perspectiva do usuário. São exibidas as telas que compõem o sistema, demonstrando como o aplicativo conduz o usuário desde o acesso inicial até o gerenciamento completo de suas metas financeiras. O objetivo é evidenciar a navegação, usabilidade e integração entre os módulos do sistema, destacando como a solução proporciona uma experiência simples e intuitiva.

4.6.1 Telas Iniciais do aplicativo

Figura 27 – Splash Screen



Fonte: elaborado pelo autor (2025)

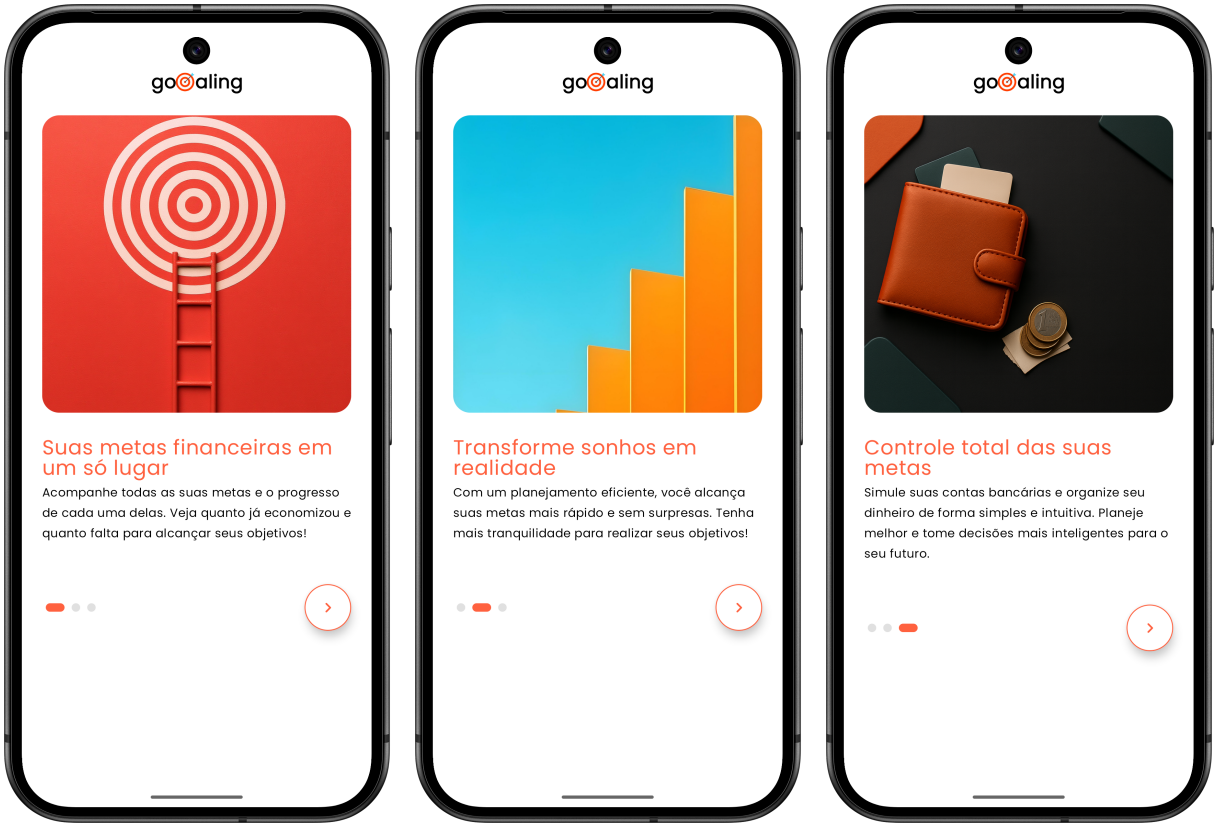
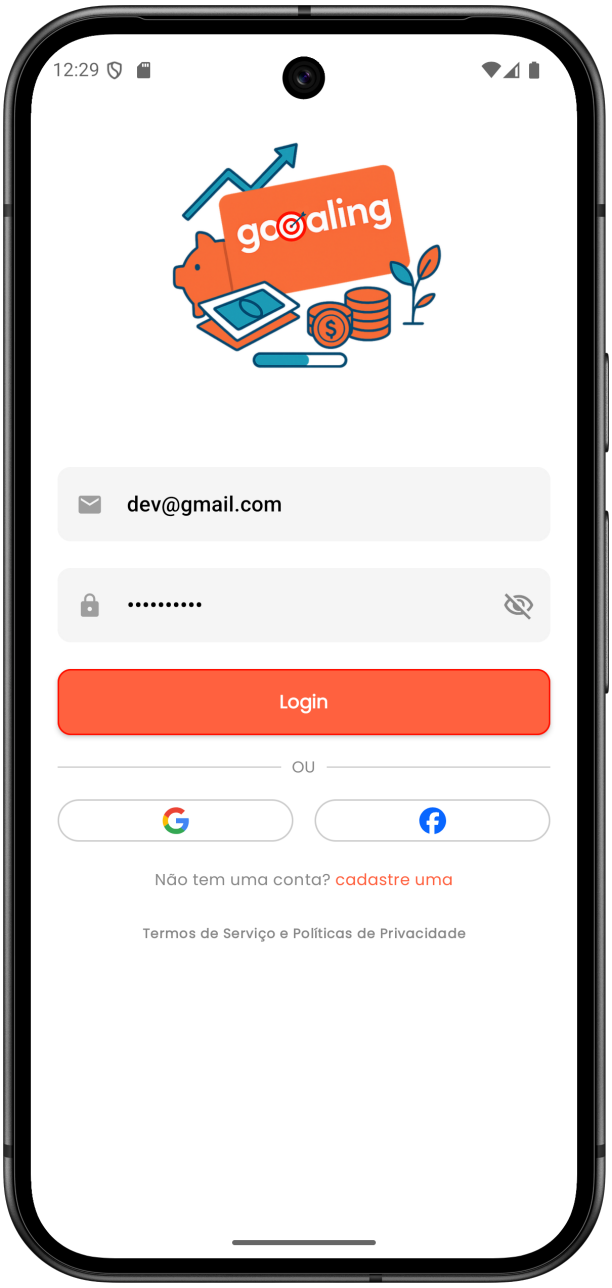


Figura 28 – Sequência de telas introdutórias do aplicativo Gooaling

Fonte: elaborado pelo autor (2025)

Ao abrir o aplicativo, o usuário é recebido pela tela de **splash screen**, seguida de uma sequência introdutória que apresenta os principais recursos e benefícios do Gooaling. Essas telas têm o propósito de contextualizar o funcionamento do sistema e motivar o usuário a alcançar suas metas financeiras. Após a introdução, o usuário é direcionado para a **tela de login**, onde pode acessar sua conta existente ou criar um novo cadastro para iniciar o uso do aplicativo.

Figura 29 – Tela de login do aplicativo



Fonte: elaborado pelo autor (2025)

4.6.2 Navegação após o login

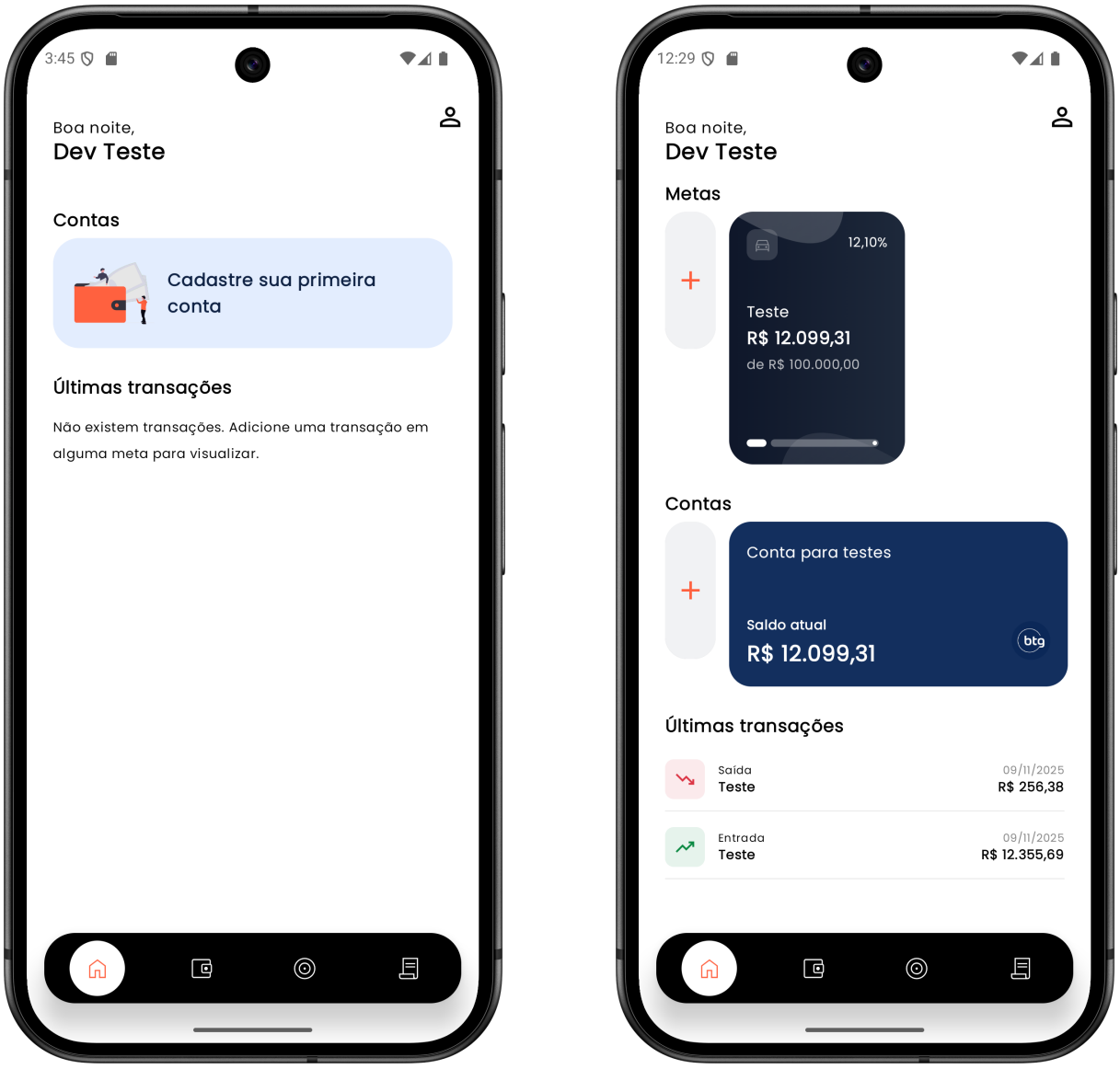
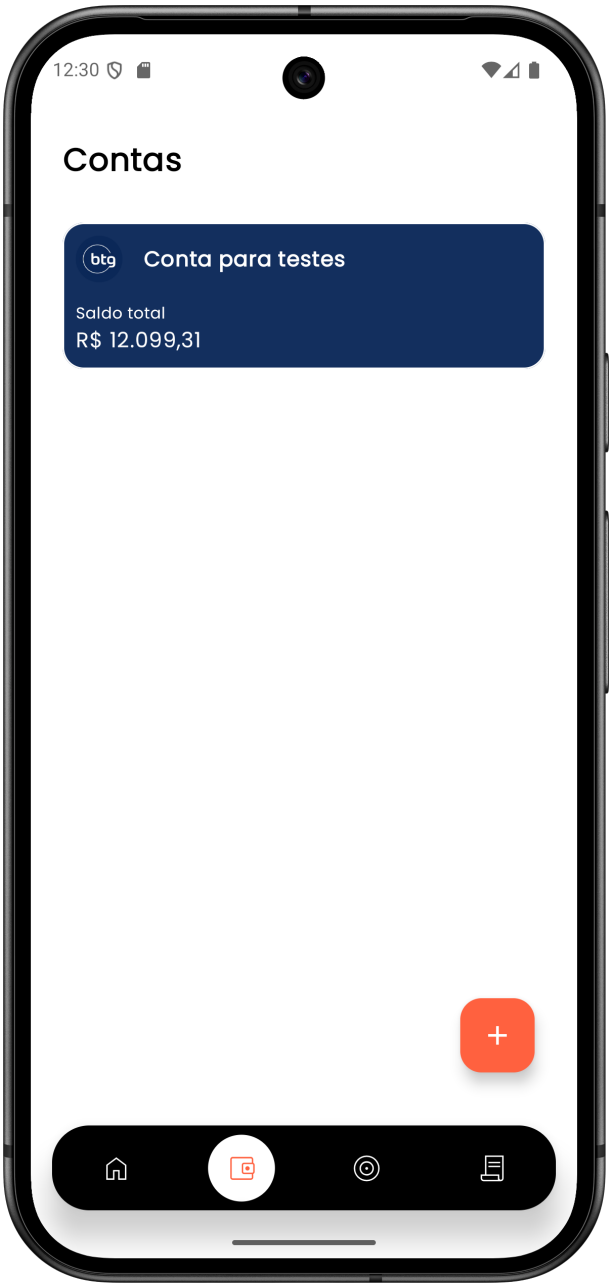


Figura 30 – Telas iniciais do aplicativo com e sem registros

Fonte: elaborado pelo autor (2025)

Após o login, o usuário é direcionado para a **tela inicial (Home)**, que atua como o ponto central de navegação do aplicativo. Nessa tela, é possível visualizar um resumo das **metas financeiras**, das **contas cadastradas** e das **últimas transações**. A partir dela, o usuário pode navegar entre os módulos principais — Metas, Contas e Transações — utilizando a barra inferior de navegação.

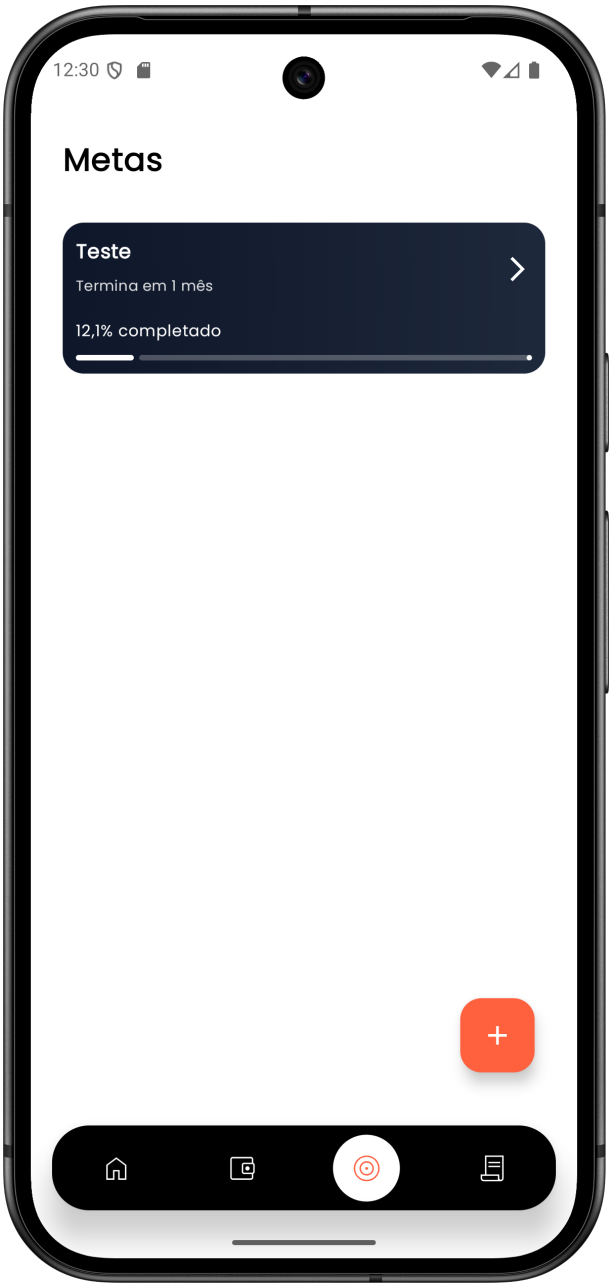
Figura 31 – Tela de listagem de contas



Fonte: elaborado pelo autor (2025)

Ao acessar o módulo **Contas**, o usuário encontra uma lista com todas as suas contas financeiras registradas no sistema. Cada conta exibe o saldo total e a instituição associada. A tela também disponibiliza o botão “+”, no canto inferior direito, que permite cadastrar novas contas de forma prática e rápida.

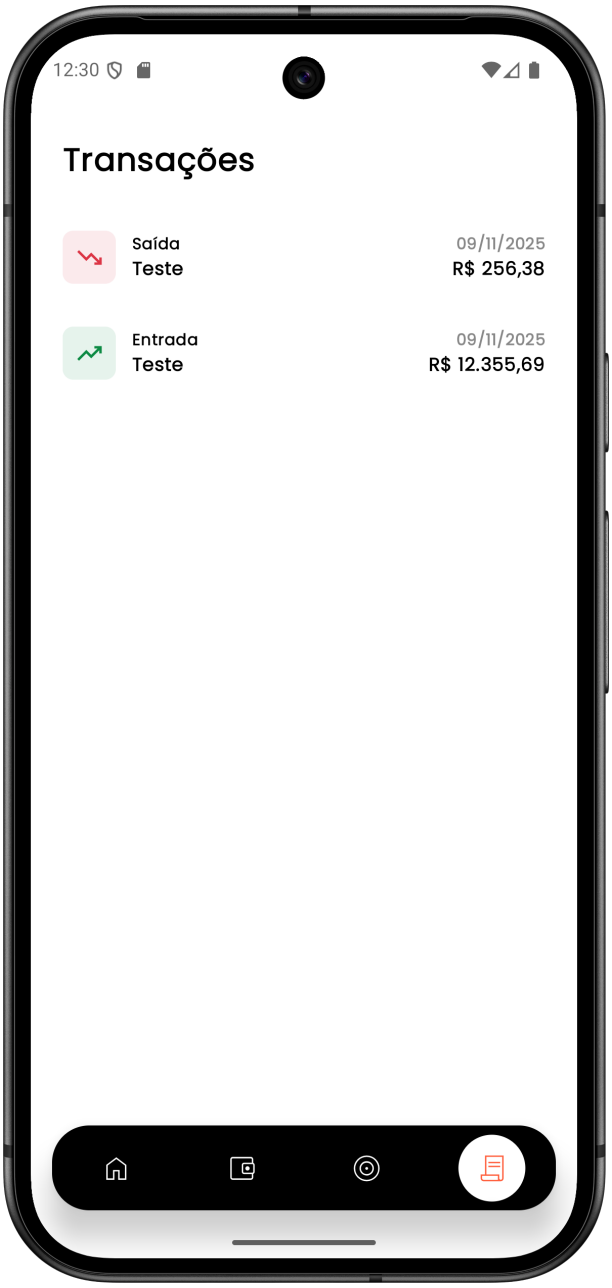
Figura 32 – Tela de listagem de metas



Fonte: elaborado pelo autor (2025)

De forma semelhante, no módulo **Metas**, o usuário pode visualizar todas as metas criadas, acompanhando o progresso financeiro de cada uma. O botão “+” também está disponível nesta tela, permitindo adicionar novas metas de maneira simples e intuitiva.

Figura 33 – Tela de transações

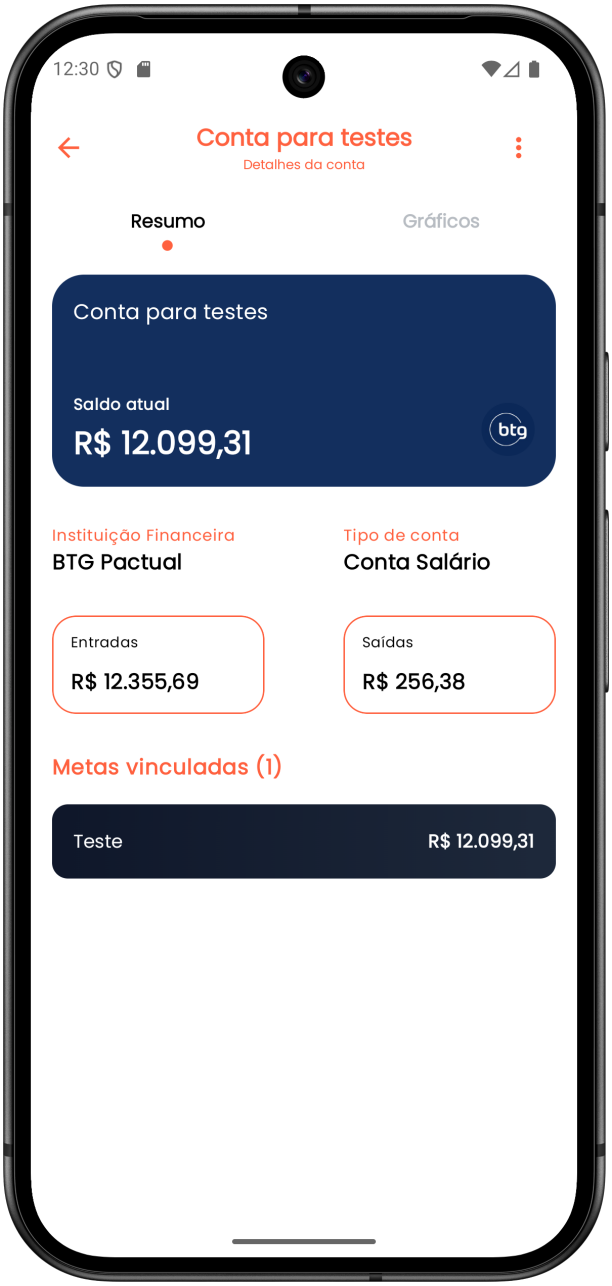


Fonte: elaborado pelo autor (2025)

Por fim, no módulo **Transações**, o usuário pode consultar todas as movimentações realizadas, classificadas entre *entradas* e *saídas*. Essa tela apresenta os valores, as datas e o tipo de operação, oferecendo um controle detalhado do histórico financeiro.

4.6.3 Módulo Contas

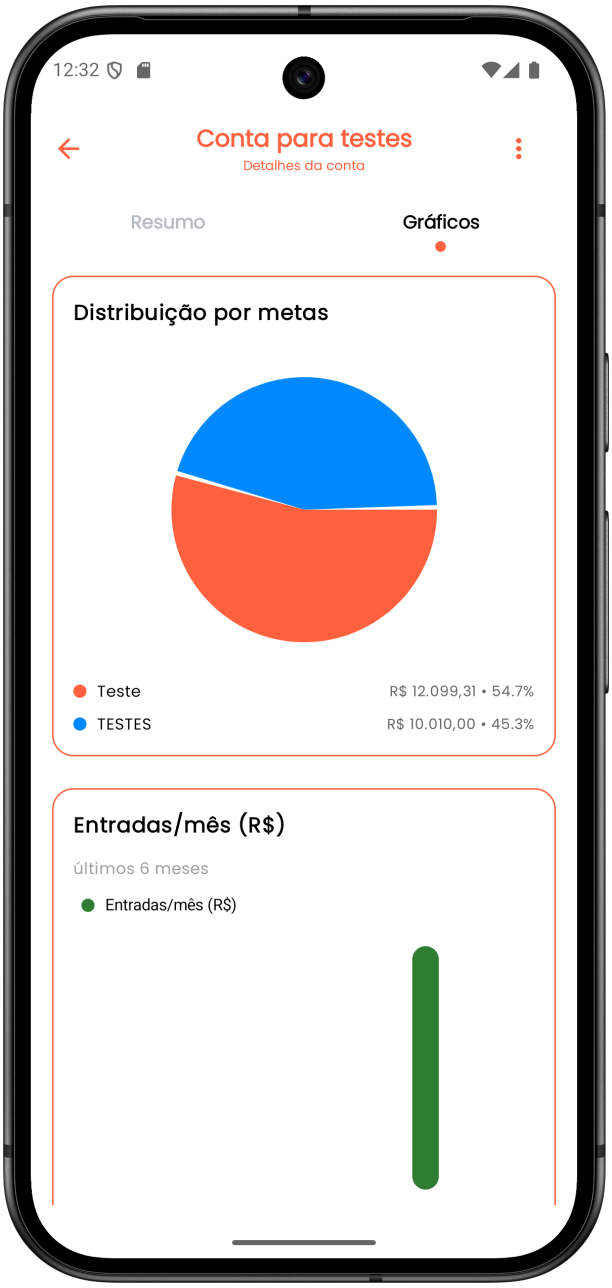
Figura 34 – Tela de visualização de conta: Resumo



Fonte: elaborado pelo autor (2025)

Ao selecionar uma conta específica, o usuário é levado à **tela de detalhes da conta**, que inicialmente apresenta a aba **Resumo**. Nessa seção, é possível visualizar informações como o nome da conta, o saldo atual, a instituição financeira e os valores totais de entradas e saídas. Também são exibidas as metas financeiras vinculadas àquela conta, permitindo uma visão unificada entre os dois módulos.

Figura 35 – Tela de visualização de conta: Gráficos



Fonte: elaborado pelo autor (2025)

Na aba **Gráficos**, o aplicativo apresenta visualizações detalhadas do comportamento financeiro da conta. O usuário pode acompanhar a distribuição dos valores entre metas e o histórico mensal de entradas, facilitando a análise do desempenho financeiro.

4.6.4 Módulo Metas

Figura 36 – Tela de visualização de meta: Resumo

Fonte: elaborado pelo autor (2025)

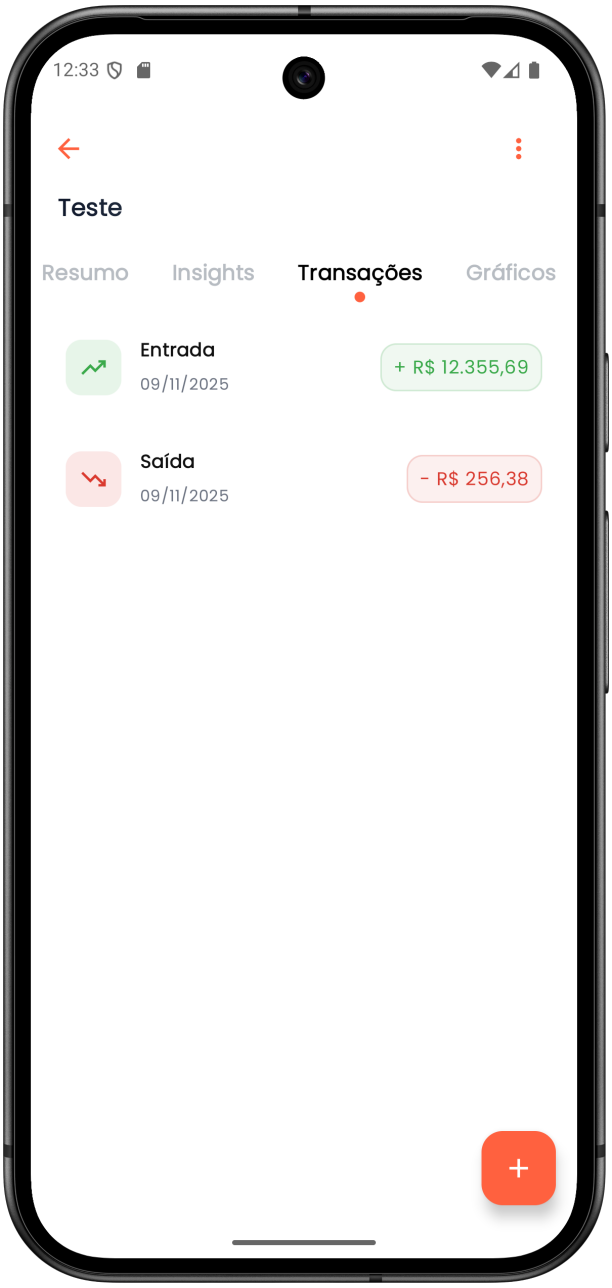
Ao acessar uma meta específica, o usuário é direcionado para a tela de **Resumo da Meta**, onde pode acompanhar o progresso atual, o valor acumulado, o total desejado e a data limite estabelecida.

Figura 37 – Tela de visualização de meta: Insights

Fonte: elaborado pelo autor (2025)

Na aba **Insights**, o aplicativo fornece projeções automáticas, exibindo o acúmulo médio previsto por dia, semana e mês, além do tempo restante para alcançar o objetivo financeiro.

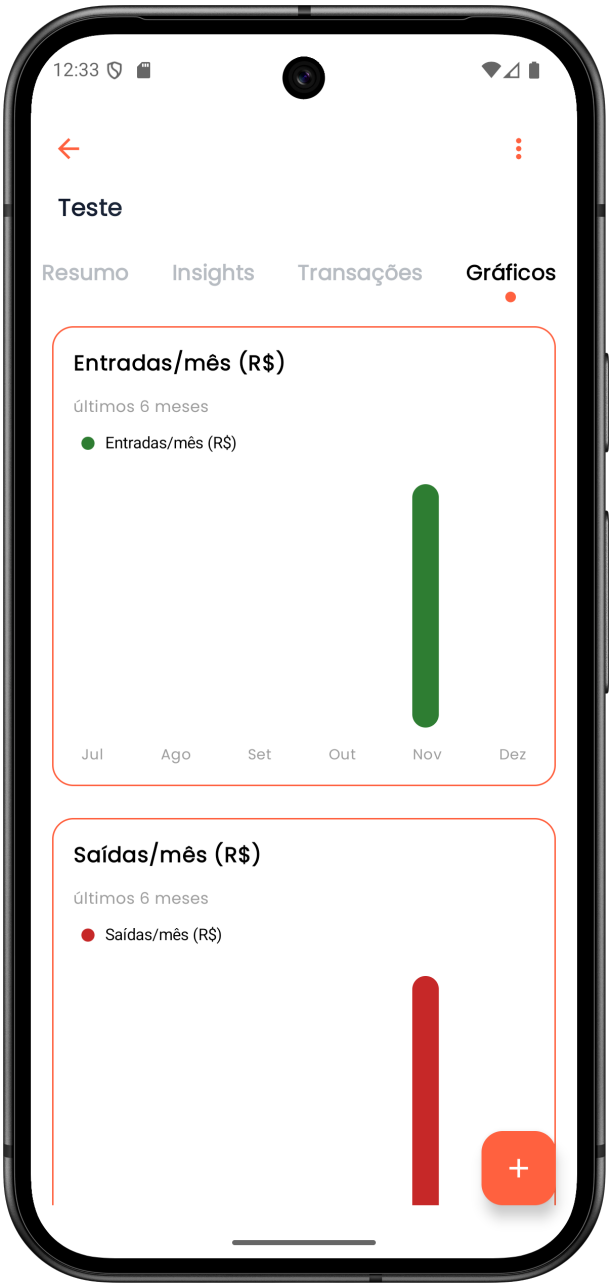
Figura 38 – Tela de visualização de meta: Transações



Fonte: elaborado pelo autor (2025)

Na aba **Transações**, o usuário pode visualizar todas as movimentações vinculadas àquela meta, incluindo aportes (entradas) e retiradas (saídas), mantendo o histórico financeiro organizado e acessível.

Figura 39 – Tela de visualização de meta: Gráficos



Fonte: elaborado pelo autor (2025)

Na seção **Gráficos**, o Gooaling exibe comparativos visuais de entradas e saídas mensais, auxiliando o usuário na análise do comportamento financeiro e no acompanhamento do progresso em direção ao cumprimento da meta.

Figura 40 – Tela de arrecadamento

Fonte: elaborado pelo autor (2025)

Figura 41 – Tela de arrecadamento: Detalhes

Fonte: elaborado pelo autor (2025)

Por fim, o módulo conta com as telas de **Arrecadamento**, que permitem ao usuário realizar e acompanhar seus aportes e retiradas mensais de forma prática e segura. Essas telas complementam a gestão das metas, possibilitando o controle direto dos valores investidos e oferecendo uma visão clara da evolução financeira de cada objetivo.

5 Considerações finais

O desenvolvimento de um aplicativo para o gerenciamento de metas financeiras demonstrou ser uma opção viável para a gestão de metas financeiras. Desta forma, o projeto conseguiu atingir seus principais objetivos, fornecendo uma ferramenta para dispositivos móveis acessível que auxilia os usuários na organização de suas metas.

As principais contribuições deste projeto incluem a implementação de funcionalidades voltadas para o planejamento e acompanhamento de metas financeiras de longo prazo, além de oferecer um recurso de orientação financeira para ajudar os usuários a economizarem de forma mais eficaz. O projeto pode abrir possibilidades para futuras expansões e melhorias, incluindo a incorporação de inteligência artificial para análise de padrões de gastos e recomendações personalizadas.

5.1 Trabalhos futuros

Após o desenvolvimento da solução proposta, diversas possibilidades de aprimoramento e expansão do aplicativo **Gooaling** podem ser bem vindas, visando tornar a experiência do usuário ainda mais completa, interativa e integrada. Entre as principais propostas de trabalhos futuros, destacam-se:

- **Implementação de um mapa de progresso gamificado:** criação de uma funcionalidade interativa no formato de um minigame, em que o usuário avança em um mapa animado conforme realiza aportes em suas metas financeiras. O sistema permitirá a personalização dos mapas e do avatar, tornando a jornada de economia mais envolvente e motivadora.
- **Integração com APIs de instituições financeiras:** desenvolvimento de uma integração com serviços externos que disponibilizem taxas de rendimento de diferentes investimentos. Com isso, o aplicativo poderá gerar comparativos e demonstrativos que indiquem quanto o valor guardado pelo usuário poderia estar rendendo, de acordo com a instituição financeira selecionada.
- **Ampliação dos recursos visuais e analíticos:** adição de novos gráficos e painéis de visualização que ofereçam uma compreensão ainda mais clara do comportamento financeiro do usuário, incluindo análises detalhadas de aportes, metas e saldos de contas.

- **Módulo de despesas mensais:** implementação de um módulo completo para o gerenciamento das despesas do usuário, permitindo o registro e acompanhamento de gastos fixos e variáveis. Essa funcionalidade visa transformar o Gooaling em uma solução financeira totalmente centralizada, reunindo metas, contas, aportes e despesas em um único ambiente.
- **Login multiplataforma:** implementação de autenticação integrada por meio de provedores externos, como Google e Facebook, permitindo que o usuário acesse o aplicativo de forma mais rápida, prática e segura. Essa abordagem amplia a acessibilidade, reduz barreiras de entrada e fortalece a integração do Gooaling com outros serviços utilizados no dia a dia.

Referências

Amazon Web Services. Diferença entre https e http. *AWS*, 2024. Disponível em: <https://aws.amazon.com/pt/compare/the-difference-between-https-and-http/>. Citado na página 25.

Amazon Web Services. *NoSQL*. 2024. Disponível em: <https://aws.amazon.com/pt/nosql/>. Citado na página 26.

Amazon Web Services. O que é uma api? *Amazon Web Services*, 2024. Disponível em: <https://aws.amazon.com/pt/what-is/api/>. Citado 2 vezes nas páginas 24 e 25.

ANBIMA. O raio x do investidor brasileiro. *Relatório*, 2017. Disponível em: <https://cointimes.com.br/wp-content/uploads/2018/08/Relatorio-Raio-X-Investidor-PT.pdf>. Citado na página 15.

Android Developers. *Jetpack Compose: Tutorial*. 2024. Disponível em: <https://developer.android.com/develop/ui/compose/tutorial?hl=pt-br#:~:text=O%20Jetpack%20Compose%20%C3%A9%20um,interface%20simples%20com%20fun%C3%A7%C3%B5es%20declarativas.> Citado na página 28.

BOURQUE, P.; FAIRLEY, R. E. (Ed.). *Guide to the Software Engineering Body of Knowledge (SWEBOK) V3.0*. IEEE Computer Society, 2014. Disponível em: <https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf>. Citado na página 19.

Cloudflare. O que é https? *Cloudflare*, 2024. Disponível em: <https://www.cloudflare.com/pt-br/learning/ssl/what-is-https/>. Citado na página 25.

DEVELOPERS, A. *DataStore: camada de dados*. 2025. Acesso em: 13 out. 2025. Disponível em: <https://developer.android.com/topic/libraries/architecture/datastore?hl=pt-br>. Citado na página 36.

DEVELOPERS, A. *Guia de arquitetura de apps Android*. 2025. Acesso em: 13 out. 2025. Disponível em: <https://developer.android.com/topic/architecture?hl=pt-br>. Citado na página 33.

DEVELOPERS, A. *Jetpack Compose*. 2025. Acesso em: 13 out. 2025. Disponível em: <https://developer.android.com/jetpack/compose?hl=pt-br>. Citado na página 34.

DEVELOPERS, A. *Visão geral do ViewModel*. 2025. Acesso em: 13 out. 2025. Disponível em: <https://developer.android.com/topic/libraries/architecture/viewmodel?hl=pt-br>. Citado 2 vezes nas páginas 33 e 35.

Express. *Express — Fast, unopinionated, minimalist web framework for Node.js*. 2024. Disponível em: <https://expressjs.com/pt-br/>. Citado na página 27.

FERNANDES, M. C. *O que é um aplicativo móvel*. 2016. Disponível em: <https://blog.stone.com.br/aplicativo-movel/>. Citado na página 29.

FIELDING, R. T. *Architectural Styles and the Design of Network-Based Software Architectures*. Tese (PhD Thesis) — University of California, Irvine, 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Citado na página 25.

G1. Planejamento financeiro é aliado para ter uma vida mais tranquila e conquistar sonhos e objetivos. *G1*, 2022. Disponível em: <https://g1.globo.com/pr/parana/economia/educacao-financeira-no-parana/noticia/2022/11/11/planejamento-financeiro-e-aliado-para-ter-uma-vida-mais-tranquila-e-conquistar-sonhos-e-objetivos.ghtml>. Citado na página 14.

GOOGLE. *Aprenda Kotlin*. 2024. Disponível em: <https://developer.android.com/kotlin/learn?hl=pt-br>. Citado na página 28.

GR1D. *Swagger: O que é e como funciona?* 2022. Disponível em: <https://gr1d.io/2022/04/15/swagger/>. Citado na página 28.

IBM. *MongoDB*. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/mongodb>. Citado na página 28.

INVESTIMENTOS, T. *Metas financeiras: como defini-las e alcançá-las*. 2023. Acesso em: 17 nov. 2024. Disponível em: <https://blog.toroinvestimentos.com.br/educacao-financeira/metas-financeiras/#:~:text=Assim%20sendo%2C%20as%20metas%20financeiras,vida%20que%20dependem%20de%20dinheiro.> Citado na página 17.

ISO/IEC/IEEE. *Software life cycle processes*. 2017. Resumo oficial do padrão. Acesso em: 8 out. 2025. Disponível em: <https://www.iso.org/standard/63712.html>. Citado na página 19.

LUSARDI, A. 401(k) pension plans and financial advice: Should companies follow ibm's initiative? *Employee Benefit Plan Review*, p. 16–18, 2007. Citado na página 16.

LUSARDI, A. The importance of financial literacy. *NBER Reporter*, p. 13–16, 2009. Citado na página 16.

MANDEL, M.; LONG, E. A economia de aplicativos no brasil. *Revista Progressive Policy Institute*, 2017. Citado na página 29.

Medium. *Afinal, o que é o Jetpack Compose?* 2024. Disponível em: <https://medium.com/digitalproductsdev/afinal-o-que-%C3%A9-o-jetpack-compose-61e3f6ce602b>. Citado na página 29.

Ministério do Meio Ambiente. *Educação financeira também é qualidade de vida*. 2022. Acesso em: 14 nov. 2024. Disponível em: <https://www.gov.br/mma/pt-br/acesso-a-informacao/programa-de-gestao-e-desempenho-pgd/o-que-temos-para-voce/educacao-financeira/educacao-financeira-tambem-e-qualidade-de-vida>. Citado na página 16.

MORUA, D. *Conhecendo o Jetpack Compose*. 2022. Acesso em: 13 out. 2025. Disponível em: <https://medium.com/wearejaya/conhecendo-o-jetpack-compose-d0ca4398e5c7>. Citado na página 34.

Mozilla Developer Network. *Introdução ao Express e Node.js*. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction>. Citado na página 27.

Mozilla Developer Network. *JavaScript: A linguagem de programação que conflitua*. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/conflicting/Web/JavaScript>. Citado na página 26.

Node.js. *Node.js*. 2024. Disponível em: <https://nodejs.org/pt>. Citado na página 27.

PERICAS-GEERTSEN, S.; RIEM, M. (Ed.). *MVC: Model-View-Controller API — Version 1.0 Early Draft (Second Edition)*. 2015. Specification: JSR-371, Early Draft Review (Second Edition). Disponível em: <https://repo1.maven.org/maven2/javax/mvc/javax.mvc-api/1.0-edr2/javax.mvc-api-1.0-edr2-spec.pdf>. Citado 3 vezes nas páginas 22, 25 e 26.

Organização de Cooperação e Desenvolvimento Econômico (OCDE). *Educação financeira*. 2005. Acesso em: 14 nov. 2024. P. 13. Disponível em: <https://www.oecd.org/>. Citado na página 16.

PEREIRA, G. M. G. *A energia do dinheiro. Como fazer dinheiro e desfrutar dele*. 2. ed.. ed. Rio de Janeiro: Campus, 2003. Citado na página 17.

Prodest. *O uso de aplicativos na sociedade*. 2023. Disponível em: <https://prodest.es.gov.br/o-uso-de-aplicativos-na-sociedade>. Citado na página 29.

ROSS, S. A.; WESTERFIELD, R. W.; JAFFE, J. F. *Administração Financeira: Corporate Finance*. 2ª edição. ed. São Paulo: Atlas, 2002. Citado na página 17.

SANTOS, R. S. D. Análise do crescimento e da concentração do mercado de microsseguros no brasil. *Artigo*, Rio Grande do Sul, RS, Brasil, 2018. Disponível em: <https://www.lume.ufrgs.br/handle/10183/188254>. Citado na página 15.

SOUSA, A.; TORRALVO, C. *Aprenda a administrar o próprio dinheiro*. [S.l.]: Editora Saraiva, 2008. 160 p. Citado na página 14.

SPC Brasil. Endividamento entre jovens: como reverter esse crítico cenário. *Exame*, 2021. Disponível em: <https://exame.com/meu-dinheiro/endividamento-entre-jovens-como-reverter-esse-critico-cenario/>. Citado 2 vezes nas páginas 14 e 15.

Anexos

ANEXO A – Licença MIT