CENTRO UNIVERSITÁRIO DO DISTRITO FEDERAL – UDF

COORDENAÇÃO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

DAVI PEREIRA DOS SANTOS - 32075821
EDUARDO ROBERTO LUCENA SILVA - 31363181
THALISSON COSTA MESQUITA COELHO - 31810756

AFT: ATLAS FINTECH

BRASÍLIA

2026

DAVI PEREIRA DOS SANTOS - 32075821
EDUARDO ROBERTO LUCENA SILVA - 31363181
THALISSON COSTA MESQUITA COELHO - 31810756

AFT: ATLAS FINTECH

Trabalho de conclusão de curso apresentado à Coordenação de Ciência da Computação, do Centro Universitário do Distrito Federal - UDF, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Eliel Silva da Cruz

BRASÍLIA

2026

Santos, Davi. Silva, Eduardo, Coelho, Thalisson.

AFT: ATLAS FINTECH / Davi Santos; Eduardo Silva; Thalisson Coelho. Brasília, 2026.

Orientador: Me. Eliel Silva da Cruz.

Trabalho de conclusão de curso (Graduação – Ciência da Computação) -- Centro Universitário do Distrito Federal – UDF. Coordenação de Exatas, Brasília, DF, 2026.

1.Gestão Financeira. 2.Sistema Web. 3.Fluxo de Caixa. I.Gestão Empresarial.

DU:

DAVI PEREIRA DOS SANTOS - 32075821
EDUARDO ROBERTO LUCENA SILVA - 31363181
THALISSON COSTA MESQUITA COELHO - 31810756

AFT: ATLAS FINTECH

Trabalho de conclusão de curso apresentado à Coordenação de Ciência da Computação, do Centro Universitário do Distrito Federal - UDF, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Eliel Silva da Cruz

Brasília, 15 de Junho de 2026.

Banca Examinadora

_____
ELIEL SILVA DA CRUZ
Mestre
Centro Universitário do Distrito Federal – UDF

_____
GABRIEL DE OLIVEIRA ALVES
Mestre
Centro Universitário do Distrito Federal – UDF

_____
GLEIDSON PORTO BATISTA
Mestre
Centro Universitário do Distrito Federal – UDF

NOTA: _____

Dedicamos este trabalho a nossa família e aos nossos professores pelo apoio na realização de todo esforço e realização.

AGRADECIMENTOS

Agradecemos, primeiramente, a Deus, por nos conceder força, sabedoria e perseverança ao longo desta trajetória, tornando possível a concretização deste grande sonho. Ao nosso orientador, Prof. Me. Eliel Silva da Cruz, expressamos nossa sincera gratidão pela dedicação, paciência e valiosas orientações, que contribuíram significativamente para o desenvolvimento deste trabalho. Estendemos nossos agradecimentos às nossas famílias, pelo amor, apoio incondicional e incentivo constante durante toda essa jornada, cujos esforços e renúncias foram essenciais para esta conquista. Por fim, agradecemos a todos que, direta ou indiretamente, contribuíram para a realização deste projeto e para a conclusão desta importante etapa acadêmica.

“*A educação é a arma mais poderosa que você pode usar para mudar o mundo.”*

Nelson Mandela

RESUMO

Em síntese, o trabalho apresenta o desenvolvimento do Atlas FinTech, um sistema web de gestão financeira empresarial criado para superar as dificuldades enfrentadas por pequenas e médias organizações no controle de receitas, despesas e fluxo de caixa, frequentemente conduzido por meio de processos manuais ou ferramentas limitadas. O sistema reúne recursos como registro de transações, acompanhamento do fluxo de caixa, categorização financeira e geração de relatórios gerenciais. O objetivo consiste em oferecer uma solução que torne o monitoramento financeiro mais claro e eficiente, fortalecendo a tomada de decisão estratégica. A metodologia envolve o levantamento e análise de requisitos, modelagem do sistema e implementação. A codificação utiliza *TypeScript*, *Angular* e *Sass* na camada de apresentação, enquanto o *backend* é desenvolvido em *Python* com *FastAPI*, estruturado segundo a arquitetura em três camadas (three-tier architecture). Os dados são armazenados em *PostgreSQL*. O conjunto dessas etapas evidencia que o sistema possui potencial para aprimorar a organização financeira e apoiar decisões mais estruturadas no ambiente empresarial.

Palavras-chave: Gestão financeira. Sistema web. Fluxo de caixa. *Angular. FastAPI. PostgreSQL.*

ABSTRACT

In summary, this work presents the development of Atlas FinTech, a web-based business financial management system designed to overcome the difficulties faced by small and medium-sized organizations in controlling revenue, expenses, and cash flow, often done through manual processes or limited tools. The system brings together features such as transaction logging, cash flow monitoring, financial categorization, and the generation of management reports. The objective is to offer a solution that makes financial monitoring clearer and more efficient, strengthening strategic decision-making. The methodology involves requirements gathering and analysis, system modeling, and implementation. The coding uses TypeScript, Angular, and Sass in the presentation layer, while the backend is developed in Python with FastAPI, structured according to a three-tier architecture. Data is stored in PostgreSQL. These steps demonstrate that the system has the potential to improve financial organization and support more structured decisions in the business environment.

Keywords: Financial management. Web system. Cash flow. *Angular*. *FastAPI*. *PostgreSQL*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABREVIATURAS

AFT	Atlas FinTech
MPME	Micro, Pequenas e Médias Empresas
SEBRAE	Serviço Brasileiro de Apoio às Micro e Pequenas Empresas
UDF	Centro Universitário do Distrito Federal

SIGLAS

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
BaaS	Backend as a Service
CDN	Content Delivery Network
CNPJ	Cadastro Nacional da Pessoa Jurídica
CPF	Cadastro de Pessoas Físicas
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DER	Diagrama de Entidade-Relacionamento
ECDSA	Elliptic Curve Digital Signature Algorithm
FK	Foreign Key
HMAC	Hash-based Message Authentication Code
HTML	HyperText Markup Language
HTML5	HyperText Markup Language 5
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation

JWT	JSON Web Token
KPI	Key Performance Indicator
OAuth	Open Authorization
ORM	Object-Relational Mapping
PaaS	Platform as a Service
PDF	Portable Document Format
PK	Primary Key
REST	Representational State Transfer
RFC	Request for Comments
RSA	Rivest-Shamir-Adleman
SCSS	Sassy Cascading Style Sheets
SDLC	Software Development Life Cycle
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SPA	Single Page Application
SQL	Structured Query Language
TLS	Transport Layer Security
TXT	Text File
UI	User Interface
UML	Unified Modeling Language
UX	User Experience
ZIP	Zigzag Inline Packing

SUMÁRIO

1 INTRODUÇÃO

Segundo o Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE, 2022), em 2022, 29% dos microempreendedores individuais encerraram suas atividades em até cinco anos de funcionamento, enquanto 21,6% das microempresas e 17% das pequenas empresas também não sobreviveram nesse período. Esse cenário evidencia a fragilidade da gestão empresarial no Brasil e mostra que a má administração financeira está entre os principais fatores que comprometem a continuidade dos negócios (PIRES, 2024). A falta de planejamento e organização financeira reduz a competitividade das empresas e aumenta os riscos de fechamento precoce.

Uma gestão financeira eficiente proporciona benefícios essenciais, como maior controle de recursos, prevenção de desperdícios e apoio à tomada de decisões estratégicas (PIRES, 2024). Além disso, a utilização de ferramentas tecnológicas tem se mostrado um fator determinante para o fortalecimento das pequenas e médias empresas, permitindo organizar informações e gerar relatórios confiáveis (SEBRAE, 2023). Isso reforça a importância de sistemas de gestão capazes de oferecer suporte para que empreendedores enfrentem os desafios do mercado de forma mais estruturada e sustentável (LAUDON; LAUDON, 2022).

Apesar dessas vantagens, pesquisas recentes revelam que 39% das micro, pequenas e médias empresas (MPMEs) brasileiras ainda controlam suas despesas manualmente ou por meio de planilhas e cadernos (UOL ECONOMIA, 2025). Esse dado mostra que grande parte dos gestores não dispõe de softwares acessíveis e intuitivos, o que compromete a precisão das informações financeiras e dificulta a tomada de decisões (SEBRAE, 2023). Portanto, a carência de soluções simples e adequadas às necessidades das MPMEs permanece como um problema recorrente no ambiente empresarial.

Diante desse contexto, este trabalho apresenta o desenvolvimento do sistema Atlas FinTech, um software de gestão financeira voltado para pequenas e médias empresas, com foco em simplicidade, eficiência e acessibilidade. A ferramenta proposta possibilita o controle de receitas, despesas e fluxo de caixa, além de fornecer relatórios claros e objetivos para apoiar decisões estratégicas. Dessa forma, busca-se fortalecer a organização financeira das empresas, reduzir riscos e contribuir para sua permanência em um mercado cada vez mais competitivo.

1.1 Objetivos

Esta seção apresenta o objetivo geral e os objetivos específicos do trabalho, que orientam o desenvolvimento do sistema proposto.

1.1.1 Objetivo geral

Desenvolver um sistema de gestão financeira para pequenas e médias empresas que possibilite o controle de receitas, despesas e fluxo de caixa, promovendo maior organização, confiabilidade das informações e apoio à tomada de decisão.

1.1.2 Objetivos específicos

Para atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

a) analisar as principais dificuldades enfrentadas por pequenas e médias empresas na gestão financeira;

b) definir os requisitos funcionais e não funcionais necessários ao sistema proposto;

c) modelar a arquitetura da aplicação, contemplando interface intuitiva e acessível;

d) implementar as funcionalidades de controle financeiro e relatórios estratégicos;

e) validar o sistema por meio de testes unitários, verificando o correto funcionamento das principais rotinas implementadas.

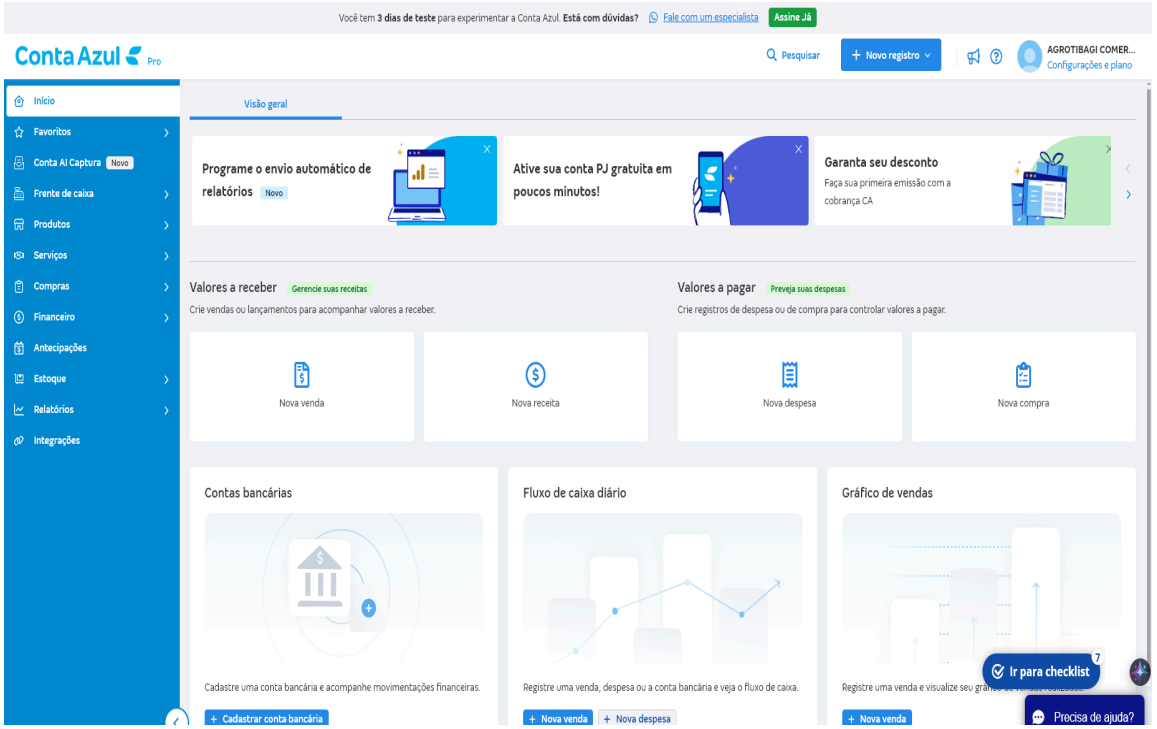
1.2 Trabalhos correlatos

Diversas ferramentas de gestão financeira estão disponíveis no mercado, atendendo empresas de diferentes portes e necessidades específicas. Essas soluções têm como objetivo principal apoiar os gestores na organização de receitas, despesas, fluxo de caixa e na tomada de decisões estratégicas. A análise de sistemas existentes evidencia que, apesar das funcionalidades oferecidas, ainda há demanda por soluções que equilibrem simplicidade, custo acessível e funcionalidades essenciais para pequenas e médias empresas (PIRES, 2024). Essa

demanda é reforçada pelo fato de que 39% das MPMEs brasileiras ainda controlam suas finanças de forma manual ou por meio de planilhas básicas, o que indica que as soluções disponíveis no mercado não atendem plenamente às necessidades desse público (UOL ECONOMIA, 2025; SEBRAE, 2023).

O Conta Azul é um sistema brasileiro de gestão financeira voltado para pequenas e médias empresas, oferecendo integração entre controle de receitas, despesas, emissão de notas fiscais eletrônicas e conciliação bancária automática (CONTA AZUL, 2025). Seu uso permite centralizar a administração financeira em uma única plataforma, proporcionando maior confiabilidade nas informações e suporte à tomada de decisões. Apesar das funcionalidades robustas e da interface intuitiva, o custo de assinatura pode representar um desafio para empresas em fase inicial ou com orçamento limitado, especialmente para microempreendedores que necessitam de soluções mais acessíveis (PIRES, 2024). Além disso, o sistema não oferece funcionalidades como agendamento automático de envio de relatórios por e-mail, interface com tema claro e escuro ou backup completo em arquivo ZIP — recursos presentes no sistema proposto e identificados como diferenciais relevantes para o público-alvo desta pesquisa. A Figura 1 apresenta a interface do sistema Conta Azul.

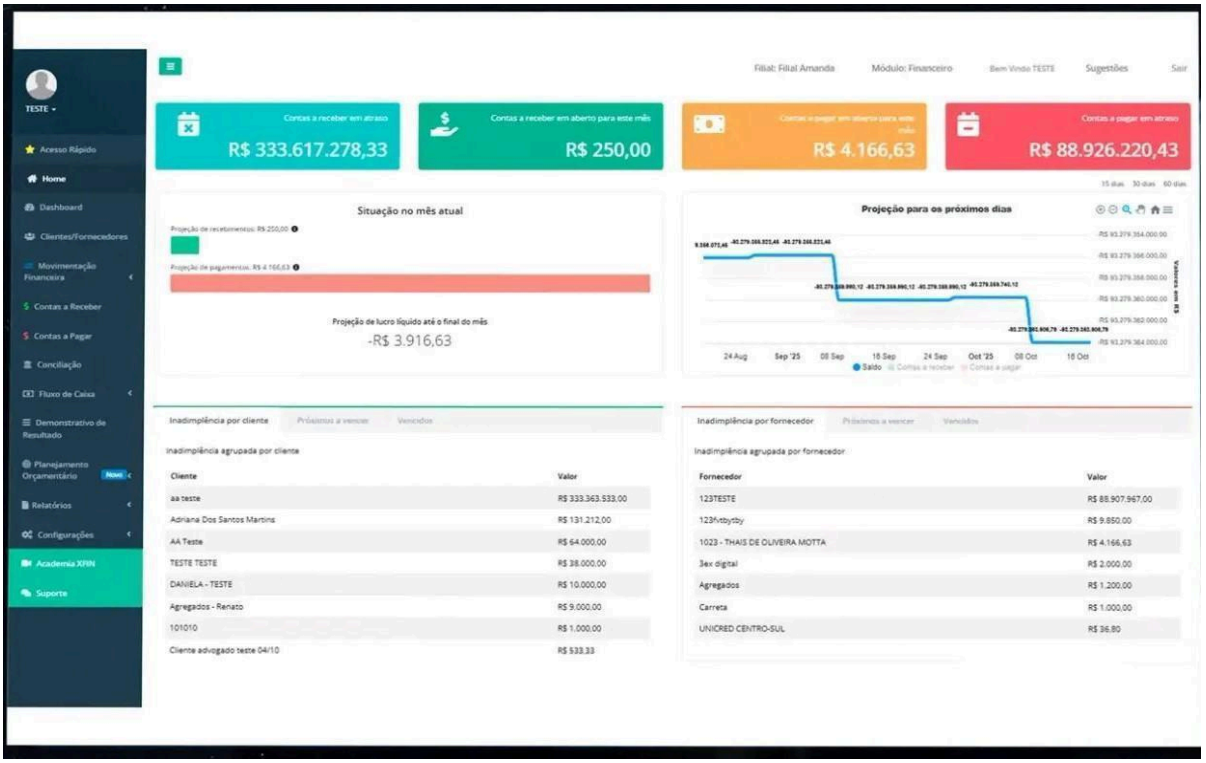
Figura 1 – Ilustração do sistema Conta Azul



Fonte: Conta Azul (2025).

O Xfin é uma plataforma direcionada ao planejamento estratégico e à análise de desempenho financeiro de pequenas empresas, disponibilizando recursos avançados para a criação de relatórios e dashboards que permitem o acompanhamento detalhado de indicadores financeiros (XFIN, 2025). Embora seja eficiente para empresas com algum nível de conhecimento contábil, a complexidade de uso pode limitar sua adoção por microempreendedores ou gestores sem experiência financeira (PIRES, 2024). Essa característica evidencia a necessidade de soluções que combinem funcionalidades relevantes com simplicidade e baixo custo, especialmente voltadas para empresas em crescimento. A Figura 2 apresenta a interface do sistema Xfin.

Figura 2 – Ilustração do sistema Xfin

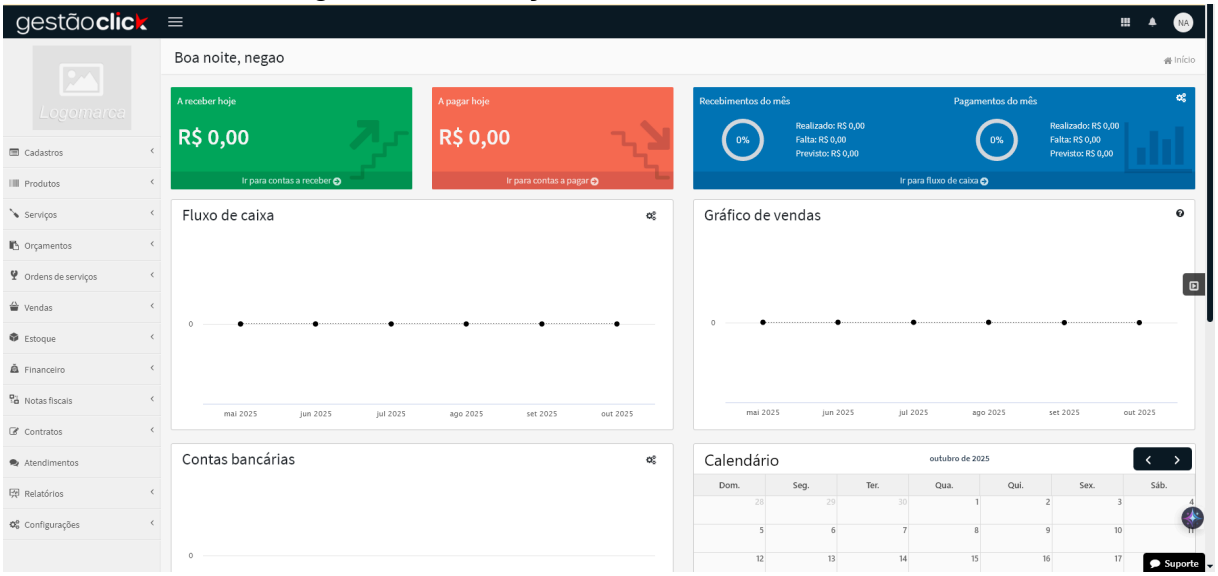


Fonte: Xfin (2025).

O GestãoClick é um software de gestão empresarial desenvolvido para micro e pequenas empresas que buscam centralizar o controle administrativo e financeiro em uma única plataforma (GESTÃOCLICK, 2024). Ele oferece funcionalidades como emissão de notas fiscais, controle de estoque, fluxo de caixa, contas a pagar e a receber, além de relatórios financeiros automatizados. Por ser um sistema online, permite o acesso remoto e o acompanhamento em tempo real das operações da

empresa. No entanto, o sistema não disponibiliza funcionalidades como autenticação social, agendamento automático de relatórios, interface com tema claro e escuro ou backup completo dos dados — ausências identificadas na análise comparativa e que representam lacunas relevantes frente às necessidades do público-alvo (PIRES, 2024). A Figura 3 apresenta a interface do sistema GestãoClick.

Figura 3 – Ilustração do sistema GestãoClick



Fonte: GestãoClick (2024).

A análise desses trabalhos correlatos evidencia que as soluções disponíveis, apesar de consolidadas, apresentam limitações em aspectos como acessibilidade de custo, complexidade de uso e ausência de funcionalidades voltadas à automação e à experiência do usuário. Nesse sentido, o Atlas FinTech é proposto como uma alternativa que supre essas lacunas, reunindo controle de receitas, despesas e fluxo de caixa com diferenciais como autenticação social (OAuth 2.0), envio automático de relatórios por e-mail, backup completo e interface com tema claro e escuro — funcionalidades ausentes nos sistemas analisados e diretamente alinhadas às necessidades das pequenas e médias empresas identificadas ao longo desta análise.

A Tabela 1 apresenta uma análise entre funcionalidades pertinentes entre o Sistema proposto e seus concorrentes:

Tabela 1 – Diferencial competitivo entre as ferramentas similares.

Funcionalidades	Atlas FinTech	Conta Azul	Xfin	Gestão Click
Dashboard	X	X	X	X
Registro de contas bancárias	X	X	X	X
Registro de transações	X	X	X	X
Contas a pagar	X	X	X	X
Contas a receber	X	X	X	X
Fluxo de caixa	X	X	X	X
Análise financeira	X	X		X
Relatórios em csv	X	X	X	X
Relatórios por e-mail	X			
Chatbot básico	X			
Login social com Google (OAuth 2.0)	X	X		X
Recuperação de senha por e-mail	X	X	X	X
Categorias personalizáveis pelo usuário	X	X		
Backup completo em arquivo ZIP	X			
Conciliação bancária por importação de CSV	X	X		
Alertas financeiros automáticos	X	X		X
Calendário financeiro consolidado	X	X		X
Interface com tema claro e escuro	X			
Responsividade (mobile-first)	X	X	X	X
Agendamento automático de envio de relatórios	X			
Previsão de fluxo de caixa (projeção)	X	X		X
Relatórios em PDF	X	X	X	X

Fonte: Elaborado pelo autor (2026).

A partir da tabela 1, observa-se que o Atlas FinTech apresenta diferenciais competitivos relevantes em relação às demais ferramentas analisadas, destacando-se pela oferta de funcionalidades adicionais voltadas à gestão financeira e à experiência do usuário. Embora as plataformas comparadas disponibilizem recursos essenciais, como *dashboard*, registro de contas bancárias, registro de transações, contas a pagar, contas a receber, fluxo de caixa e relatórios em PDF e CSV (*Comma-Separated Values*), o Atlas FinTech incorpora funcionalidades complementares que ampliam sua proposta de valor.

Entre os diferenciais identificados, destacam-se o envio automático de relatórios por e-mail, a integração com um *chatbot* básico para suporte inicial ao usuário, a personalização de categorias financeiras, o agendamento automático do envio de relatórios, a disponibilização de interface com tema claro e escuro, o *backup* completo em arquivo ZIP, além da importação de conciliação bancária por meio de arquivos CSV. O sistema também contempla funcionalidades voltadas à mobilidade e acessibilidade, como responsividade (*mobile-first*), autenticação social por conta Google (*OAuth* 2.0) e recuperação de senha por e-mail.

Além disso, o Atlas FinTech incorpora funcionalidades estratégicas, como alertas financeiros automáticos, calendário financeiro consolidado e previsão do fluxo de caixa (projeção), contribuindo para maior controle financeiro e apoio à tomada de decisão. Esses elementos ampliam a usabilidade, flexibilidade e eficiência da solução, tornando-a especialmente adequada para pequenas e médias empresas que necessitam de uma ferramenta acessível, intuitiva e com recursos avançados para gestão financeira.

Dessa forma, o Atlas FinTech demonstra potencial competitivo ao combinar funcionalidades essenciais do mercado com recursos adicionais voltados à automação, organização e suporte ao usuário, fortalecendo sua proposta como uma solução robusta e adaptável às diferentes necessidades empresariais.

1.3 Solução proposta

A solução proposta consiste no desenvolvimento do Atlas FinTech, um sistema de gestão financeira voltado para pequenas e médias empresas, com o objetivo de disponibilizar uma plataforma intuitiva, acessível e eficiente para o gerenciamento das finanças empresariais. O projeto foi concebido a partir da

necessidade de simplificar processos financeiros e reduzir erros operacionais frequentemente observados em empreendimentos que ainda utilizam métodos manuais de controle, como planilhas eletrônicas e registros físicos.

Diferentemente de softwares genéricos e, em muitos casos, de alto custo disponíveis no mercado, a proposta busca alinhar tecnologia, usabilidade e viabilidade econômica, oferecendo uma solução adaptada às necessidades do público-alvo. O sistema permite que os usuários realizem o controle de receitas, despesas e fluxo de caixa de forma prática e organizada, proporcionando visualização em tempo real da situação financeira da empresa e suporte à tomada de decisões baseadas em dados confiáveis.

Além do foco na eficiência operacional, o desenvolvimento do sistema prioriza aspectos relacionados à acessibilidade, escalabilidade e segurança da informação, permitindo que a plataforma acompanhe o crescimento da empresa sem comprometer o desempenho. A interface foi desenvolvida com base em princípios de *User Experience* (UX) e *User Interface* (UI), buscando oferecer uma navegação simples, responsiva e visualmente organizada, reduzindo a curva de aprendizado e ampliando a experiência do usuário.

Entre as principais funcionalidades implementadas no sistema, destacam-se:

a) cadastro de contas bancárias, possibilitando a separação dos saldos e movimentações por instituição financeira;

b) cadastro de receitas e despesas, permitindo registrar e categorizar movimentações financeiras para melhor acompanhamento das entradas e saídas;

c) gestão de categorias personalizáveis, oferecendo flexibilidade na organização e classificação das movimentações conforme a realidade de cada empresa;

d) controle de fluxo de caixa, disponibilizando informações atualizadas sobre saldo operacional e financeiro;

e) gestão de contas a pagar e a receber, possibilitando a organização de compromissos financeiros e emissão de alertas automáticos de vencimento;

f) *dashboard* financeiro, consolidando dados, gráficos e indicadores para análise gerencial;

g) módulo de análise financeira, com visualizações específicas para apoiar decisões estratégicas a partir do histórico de movimentações;

h) geração de relatórios personalizados, permitindo consultas em diferentes períodos, com filtros e comparativos;

i) envio automático de relatórios por e-mail mediante agendamento, possibilitando maior praticidade no acompanhamento financeiro;

j) chatbot básico, destinado a consultar saldo, contas a pagar e a receber, alertas e projeções, com base nos dados cadastrados pela empresa;

k) autenticação tradicional por e-mail e senha com verificação de e-mail e fluxo de recuperação de senha, assegurando o acesso seguro e a possibilidade de retomada de credenciais;

l) autenticação social por conta Google (*OAuth* 2.0), promovendo maior praticidade no acesso ao sistema;

m) gerenciamento de sessões com *tokens* JWT e renovação automática por *refresh token*, garantindo segurança nas requisições e continuidade da sessão sem autenticações constantes;

n) calendário financeiro consolidado e alertas financeiros automáticos, auxiliando no planejamento e controle de obrigações;

o) importação de conciliação bancária por arquivos CSV e realização de *backup* completo em arquivo ZIP, fortalecendo a integridade e a segurança das informações.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica que sustenta o desenvolvimento do sistema proposto, abordando transformação digital na gestão financeira, engenharia de software, levantamento e engenharia de requisitos, metodologias ágeis, padrões de arquitetura e projeto, UML (*Unified Modeling Language*) e testes de software. Os conceitos discutidos a seguir oferecem suporte teórico para as decisões adotadas no projeto, alinhando práticas consolidadas da literatura acadêmica às necessidades específicas das pequenas e médias empresas.

2.1 Transformação digital na gestão financeira

A transformação digital representa a incorporação de tecnologias digitais em todos os aspectos de uma organização, alterando significativamente a forma como ela opera e entrega valor aos clientes (LEITE et al., 2024). No contexto da gestão financeira, esse movimento tem permitido a automatização de processos, a redução de erros manuais e o aumento da eficiência operacional, além de promover maior transparência e agilidade na tomada de decisões (OLIVEIRA et al., 2024).

O uso de softwares e sistemas de gestão financeira possibilita o monitoramento em tempo real do fluxo de caixa, o controle de contas a pagar e a receber e a geração de relatórios estratégicos que apoiam decisões baseadas em dados (LAUDON; LAUDON, 2022). Essa digitalização também exige dos gestores o desenvolvimento de competências tecnológicas e analíticas, fundamentais para interpretar informações financeiras e extrair *insights* relevantes para o desempenho organizacional (OLIVEIRA et al., 2024).

A importância dessas ferramentas se intensificou nos últimos anos, uma vez que a continuidade das operações empresariais passou a depender de soluções digitais capazes de garantir integração entre setores, segurança da informação e gestão eficiente de processos (LEITE et al., 2024). Ferramentas simples e intuitivas são especialmente relevantes, considerando que cerca de 39% das MPMEs ainda utilizam métodos manuais ou planilhas básicas, o que aumenta a margem de erro e compromete a análise estratégica (UOL ECONOMIA, 2025; SEBRAE, 2023).

A adoção de tecnologias digitais, portanto, não é apenas uma tendência, mas uma necessidade competitiva. Empresas que investem em automação financeira, análise de dados e integração de sistemas obtêm ganhos expressivos em eficiência e precisão, além de melhorarem o processo de tomada de decisão e a sustentabilidade de seus negócios no longo prazo (LEITE et al., 2024).

2.2 Engenharia de software

A engenharia de software consiste em uma disciplina responsável pela aplicação de métodos, práticas e processos sistemáticos no desenvolvimento, operação e manutenção de sistemas computacionais, visando garantir qualidade, confiabilidade, segurança e eficiência do software (SOMMERVILLE, 2021). Sua aplicação permite estruturar o processo de desenvolvimento de maneira organizada, reduzindo falhas e promovendo maior alinhamento entre requisitos técnicos e necessidades dos usuários (PRESSMAN; MAXIM, 2021).

O desenvolvimento de software envolve etapas fundamentais, como levantamento e análise de requisitos, modelagem, implementação, testes, validação e manutenção (PRESSMAN; MAXIM, 2021). Essas atividades possibilitam a construção de sistemas mais robustos e sustentáveis, contribuindo para a redução de retrabalho, custos operacionais e falhas ao longo do ciclo de vida do produto. Em sistemas financeiros, tais práticas tornam-se ainda mais relevantes devido à necessidade de precisão, integridade dos dados e confiabilidade das operações realizadas (SOMMERVILLE, 2021).

Além disso, a engenharia de software favorece a padronização dos processos de desenvolvimento, a reutilização de componentes e a adoção de boas práticas, como modularidade, integração contínua e desenvolvimento incremental (PRESSMAN; MAXIM, 2021). A documentação contínua e a modelagem dos processos também auxiliam no rastreamento de decisões, facilitam futuras manutenções e promovem maior alinhamento entre equipes técnicas e *stakeholders* (SOMMERVILLE, 2021).

Nesse contexto, uma das abordagens mais utilizadas para organizar o processo de desenvolvimento é o ciclo de vida do software, conhecido como *Software Development Life Cycle* (SDLC). Esse modelo organiza as etapas do desenvolvimento de maneira estruturada, promovendo maior controle, previsibilidade

e qualidade no produto final (PRESSMAN; MAXIM, 2021). A Figura 4 apresenta uma representação das fases do SDLC.

Figura 4 – Ciclo de vida do desenvolvimento de software (SDLC)

Fonte: Elaborado pelo autor (2026).

O ciclo de vida do desenvolvimento de software é geralmente estruturado em fases como levantamento de requisitos, planejamento, desenvolvimento, testes e manutenção (SOMMERVILLE, 2021). Essas etapas permitem organizar o processo de construção do software de forma lógica e sequencial, favorecendo a melhoria contínua e a redução de erros ao longo do projeto (PRESSMAN; MAXIM, 2021).

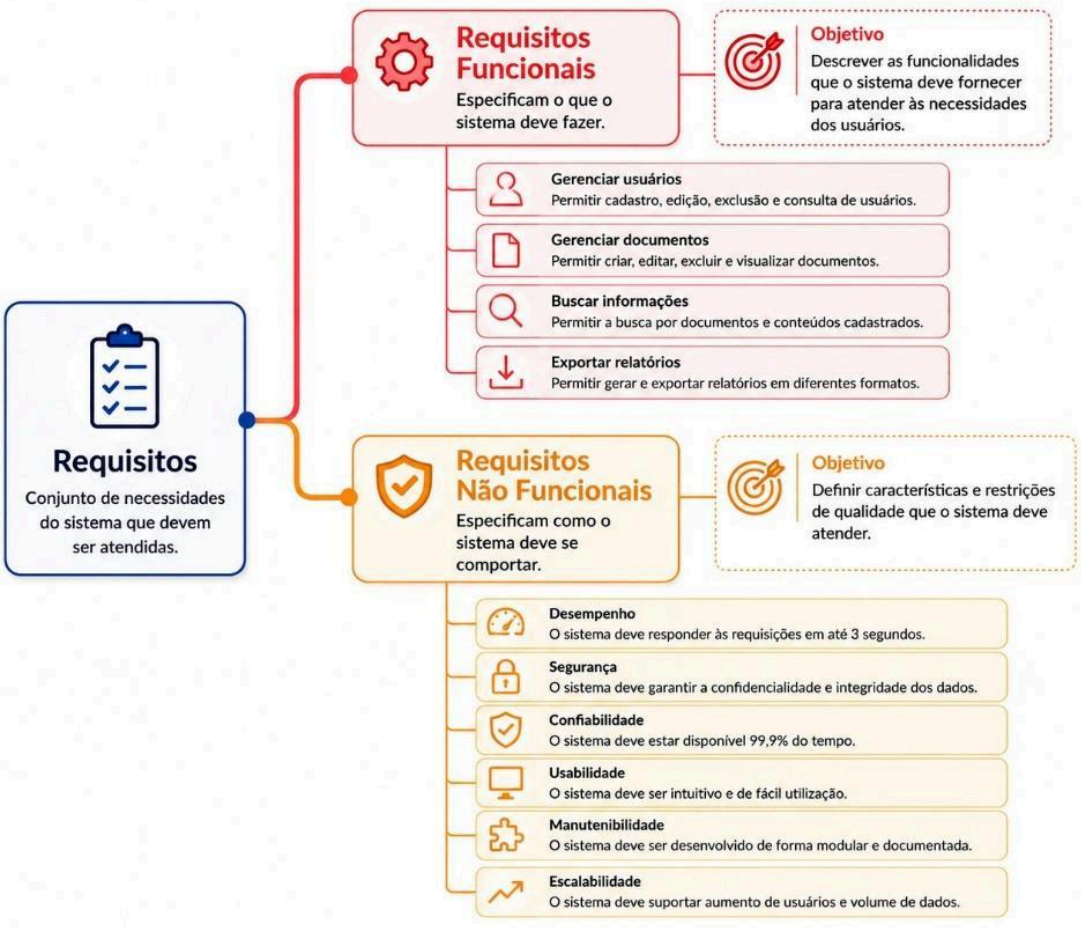
2.3 Levantamento e engenharia de requisitos

O levantamento e a engenharia de requisitos consistem na identificação, análise e documentação das necessidades dos usuários, garantindo que o sistema atenda plenamente às expectativas (SOMMERVILLE, 2021). Esse processo envolve diversas técnicas de coleta, como análise de documentos, observação de sistemas similares e estudos sobre desafios enfrentados pelo público-alvo, contribuindo para a construção de uma base sólida de requisitos (PRESSMAN; MAXIM, 2021). Para o

sistema de gestão financeira, os requisitos foram coletados por meio de artigos sobre MPMEs, análise de softwares similares e estudos sobre desafios financeiros enfrentados por empresas de pequeno porte.

Os requisitos são classificados em funcionais, que definem ações específicas do sistema, como cadastro de despesas e geração de relatórios, e não funcionais, que estabelecem restrições e padrões de qualidade, incluindo desempenho, segurança e usabilidade (SOMMERVILLE, 2021). A engenharia de requisitos também envolve validação contínua junto aos usuários, garantindo que cada funcionalidade seja relevante, completa e viável (PRESSMAN; MAXIM, 2021). Essa abordagem reduz riscos de falhas, evita retrabalho e aumenta a satisfação dos usuários finais. A Figura 5 ilustra essa classificação.

Figura 5 – Requisitos funcionais e não funcionais



Fonte: Elaborado pelo autor (2026).

A Figura 5 apresenta a divisão dos requisitos do sistema em requisitos funcionais e não funcionais. Os requisitos funcionais descrevem as funcionalidades que o sistema deve executar, enquanto os requisitos não funcionais definem critérios relacionados ao desempenho, segurança, usabilidade e confiabilidade, garantindo qualidade e eficiência na aplicação.

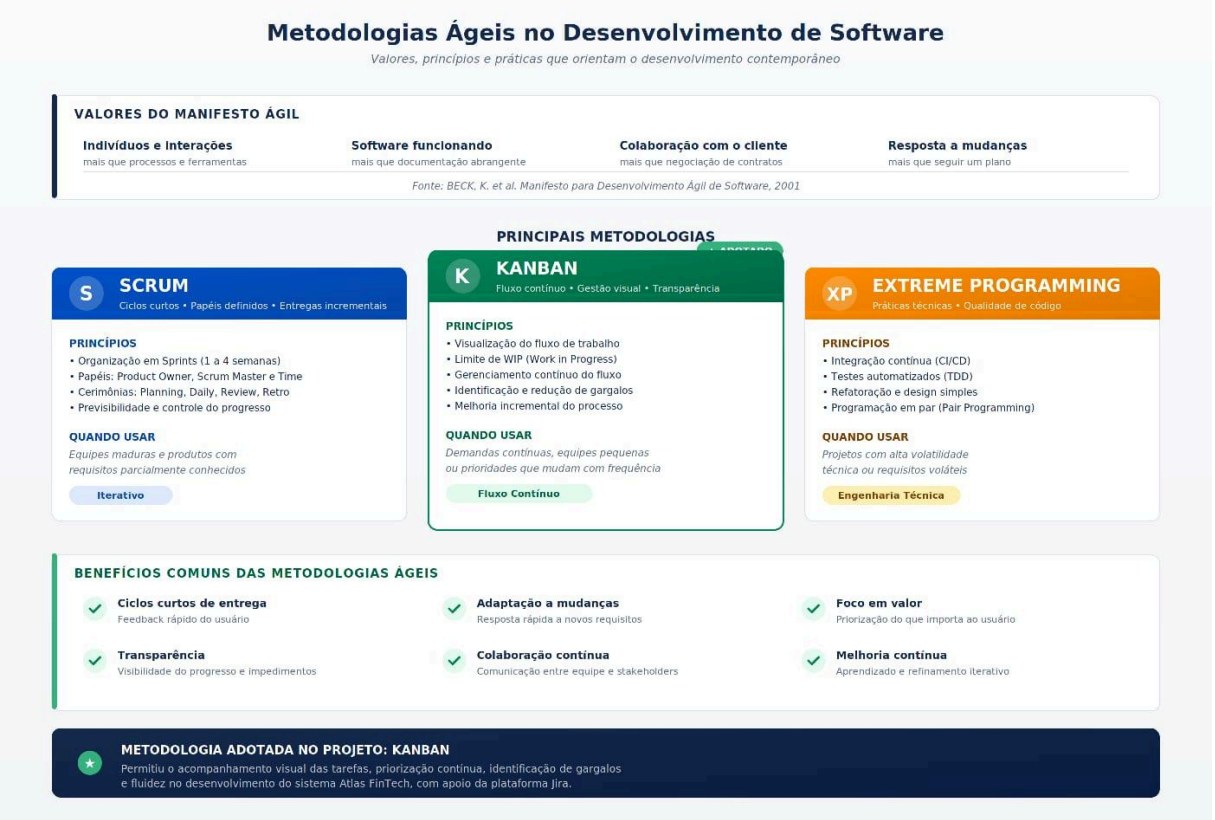
2.4 Metodologias ágeis

As metodologias ágeis são fundamentais no desenvolvimento de software contemporâneo, pois proporcionam ciclos curtos de entrega, foco em valor e adaptação contínua às necessidades do usuário (LOPES et al., 2024). A agilidade tem se consolidado como abordagem dominante para equipes que buscam eficiência, colaboração e resposta rápida a mudanças (PRESSMAN; MAXIM, 2021).

Entre as principais metodologias ágeis utilizadas atualmente, destacam-se *Scrum*, *Kanban* e *Extreme Programming*. O *Scrum* mantém ampla adoção por organizar o desenvolvimento em ciclos curtos com papéis definidos, eventos frequentes e entregas incrementais, permitindo maior controle e previsibilidade (SOMMERVILLE, 2021). O *Kanban* permanece essencial para o gerenciamento visual do fluxo de trabalho, aumentando a transparência, reduzindo gargalos e proporcionando ritmo contínuo de entrega (LOPES et al., 2024). O *Extreme Programming* segue relevante por oferecer práticas técnicas como integração contínua, testes automatizados e refatoração, elevando a qualidade do código (SOMMERVILLE, 2021).

No contexto deste projeto, foi utilizado o *Kanban* como metodologia para a organização do desenvolvimento. O *Kanban* permitiu o acompanhamento visual das tarefas, a priorização contínua das atividades e a identificação de gargalos, garantindo fluidez no processo. A representação das tarefas em colunas que evidenciam o progresso de cada item facilitou a tomada de decisão e a distribuição do trabalho entre os integrantes da equipe, contribuindo para maior previsibilidade, adaptação e eficiência no desenvolvimento do sistema proposto.

Figura 6 – Metodologia kanban no jira



Fonte: Elaborado pelo autor (2026).

A Figura 6 apresenta uma visão geral das principais metodologias ágeis utilizadas no desenvolvimento de software, destacando Scrum, Kanban e Extreme Programming (XP). A figura também evidencia os valores do Manifesto Ágil, os princípios de cada metodologia e os benefícios proporcionados pela adoção de práticas ágeis, como transparência, adaptação a mudanças, colaboração contínua e melhoria contínua. Além disso, destaca o Kanban como metodologia adotada no desenvolvimento do sistema Atlas FinTech, em razão de sua capacidade de promover o acompanhamento visual das atividades, a identificação de gargalos e a organização eficiente do fluxo de trabalho.

2.5 Padrões de arquitetura de software

O conceito de padrão de arquitetura de software refere-se a uma solução estruturada e reutilizável para problemas recorrentes no desenvolvimento de sistemas, definindo a organização dos componentes, suas responsabilidades e as interações entre eles (SOMMERVILLE, 2021). A escolha de um padrão adequado é

essencial para garantir escalabilidade, segurança, modularidade e manutenção eficaz do sistema (PRESSMAN; MAXIM, 2021).

Os padrões de arquitetura de software oferecem diferentes formas de estruturar aplicações conforme as necessidades de escalabilidade, comunicação e distribuição de responsabilidades. A arquitetura de microsserviços organiza o sistema como um conjunto de pequenos serviços independentes que se comunicam geralmente por meio de APIs, permitindo maior flexibilidade, implantação descentralizada e escalabilidade individual (SOMMERVILLE, 2021). Já a arquitetura orientada a eventos baseia-se em componentes que interagem por meio do envio e consumo de eventos, favorecendo sistemas altamente desacoplados, reativos e capazes de processar informações de forma assíncrona. A arquitetura cliente-servidor divide a aplicação entre clientes, que solicitam serviços, e servidores, que processam e respondem às requisições, sendo um dos modelos mais tradicionais e amplamente utilizados em redes e aplicações *web* (PRESSMAN; MAXIM, 2021).

Para o sistema de gestão financeira desenvolvido, foi adotada a arquitetura em três camadas (*three-tier architecture*), que segmenta a aplicação em camadas de apresentação, lógica de negócios e dados (IBM, 2025). Essa abordagem favorece a modularização, permite evolução independente das camadas, facilita testes e manutenção, sendo adequada para sistemas de médio porte em ambientes de MPMEs (SOMMERVILLE, 2021).

No *frontend*, desenvolvido em *Angular*, a interface foi separada em componentes para apresentação, *services* para a comunicação HTTP e *guards* e *interceptors* para a proteção de rotas. No *backend*, construído em *Python* com *FastAPI*, foi adotado o padrão Controller → Service → Repository: o *Controller* recebe e valida a requisição, o *Service* aplica as regras de negócio e o *Repository* acessa o banco de dados por meio do *SQLAlchemy*. Essa separação garante que cada camada tenha uma única responsabilidade, facilitando manutenção e a aplicação de testes unitários em cada camada (SOMMERVILLE, 2021).

A persistência dos dados foi implementada em *PostgreSQL*, banco de dados relacional que garante integridade, consistência, segurança das informações e suporte a transações financeiras críticas. A Figura 7 ilustra a divisão da aplicação em três camadas.

Figura 7 – Arquitetura em camadas

Fonte: Elaborada pelo autor (2026).

A Figura 7 apresenta a arquitetura em camadas adotada no sistema, organizada em apresentação, aplicação, domínio e infraestrutura. Essa abordagem permite a separação de responsabilidades entre os componentes, favorecendo maior organização, reutilização de código, escalabilidade e facilidade de manutenção da aplicação.

2.6 Modelagem do sistema

A modelagem de sistemas é uma etapa fundamental no desenvolvimento de software, permitindo representar de forma estruturada os requisitos, a arquitetura e o comportamento de um sistema antes de sua implementação (SOMMERVILLE, 2021). A UML (*Unified Modeling Language*) é a linguagem padrão utilizada para essa finalidade (OMG, 2017), oferecendo notações gráficas padronizadas que facilitam o planejamento, a documentação e a comunicação entre desenvolvedores, analistas e demais *stakeholders* (SOMMERVILLE, 2021).

Entre os principais diagramas da UML, destacam-se:

a) diagrama de casos de uso: representa as funcionalidades do sistema e a interação dos usuários (atores) com essas funções, permitindo identificar requisitos e comportamentos esperados;

b) diagrama de classes: mostra a estrutura estática do sistema, incluindo classes, atributos, métodos e relacionamentos, sendo essencial para o projeto orientado a objetos;

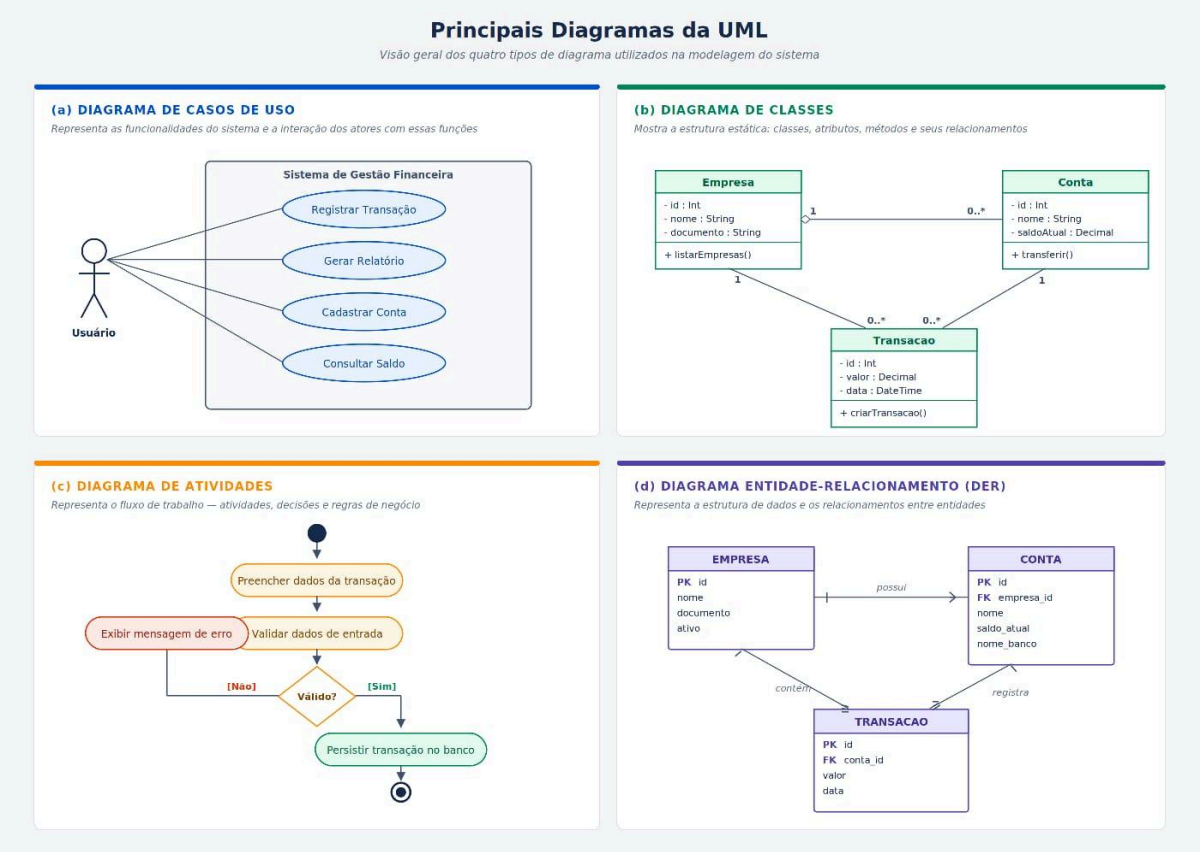
c) diagrama de atividades: representa o fluxo de trabalho do sistema por meio das ações e decisões que ocorrem em um processo, sendo útil para modelar a sequência de atividades, regras de negócio e pontos de ramificação;

d) diagrama de entidade-relacionamento (DER): representa a estrutura de dados e os relacionamentos entre entidades, sendo fundamental para o desenvolvimento do banco de dados do sistema.

O uso da UML proporciona comunicação clara entre a equipe de desenvolvimento e os *stakeholders*, reduzindo erros de interpretação e retrabalho (SOMMERVILLE, 2021). Além disso, facilita a validação e os testes do sistema, garantindo rastreabilidade e consistência entre requisitos, implementação e documentação (PRESSMAN; MAXIM, 2021).

No contexto do sistema de gestão financeira proposto, a UML foi utilizada para elaborar os diagramas de casos de uso e o diagrama de classes, permitindo que toda a equipe compreendesse o fluxo de dados e as responsabilidades de cada módulo antes da implementação. A Figura 8 ilustra um exemplo de diagrama de classes.

Figura 8 – Modelagem do sistema



Fonte: Elaborado pelo autor (2026).

A Figura 8 apresenta os principais diagramas utilizados na Linguagem de Modelagem Unificada (UML), amplamente empregada na análise e no projeto de sistemas de software. A figura ilustra o Diagrama de Casos de Uso, responsável por representar as funcionalidades do sistema e a interação dos atores; o Diagrama de Classes, que descreve a estrutura estática do sistema por meio de classes, atributos, métodos e relacionamentos; o Diagrama de Atividades, utilizado para modelar fluxos de processos, decisões e regras de negócio; e o Diagrama Entidade-Relacionamento (DER), que representa a estrutura de dados e os relacionamentos entre as entidades do banco de dados. Em conjunto, esses diagramas fornecem diferentes perspectivas do sistema, contribuindo para uma modelagem mais completa e organizada.

2.7 Testes de software

Os testes de software constituem um conjunto de práticas sistemáticas voltadas à verificação e validação de produtos de software, com o objetivo de assegurar que o sistema atenda aos requisitos especificados e identificar defeitos antes que cheguem ao usuário final (SOMMERVILLE, 2021; MYERS; SANDLER; BADGETT, 2011). De modo geral, os testes são classificados em duas categorias principais: os testes funcionais, que avaliam o comportamento do software em relação ao que foi especificado nos requisitos funcionais, e os testes não funcionais, que analisam atributos de qualidade como desempenho, segurança, confiabilidade e usabilidade (PRESSMAN; MAXIM, 2021).

No âmbito dos testes funcionais, a literatura organiza as estratégias em diferentes níveis de granularidade, frequentemente representados pela chamada Pirâmide de Testes (COHN, 2009): na base encontram-se os testes unitários, que verificam a menor unidade testável do software de forma isolada; no nível intermediário, os testes de integração, que validam a interação entre componentes; e, no topo, os testes de sistema e de aceitação, que avaliam a aplicação como um todo, sob a perspectiva do usuário final. Essa organização orienta a distribuição do esforço de testes, recomendando maior concentração nos níveis inferiores, por serem mais rápidos, mais econômicos e mais estáveis frente a mudanças no código.

Os testes unitários, em particular, exercem papel central no ciclo de desenvolvimento, pois isolam cada componente de suas dependências externas — bancos de dados, *Application Programming Interfaces* (*APIs*) e serviços de terceiros — por meio de técnicas conhecidas como test doubles, que incluem *mocks*, *stubs*, *fakes* e *spies* (FOWLER, 2007). Essa abordagem possibilita a detecção precoce de defeitos, favorece a refatoração segura do código e produz uma documentação executável do comportamento esperado de cada módulo. O padrão Arrange-Act-Assert (AAA) é amplamente adotado para estruturar testes unitários de forma legível, separando claramente a fase de preparação do cenário, a execução da operação sob teste e a verificação do resultado obtido (BECK, 2003).

Em sistemas financeiros, a aplicação de testes automatizados torna-se ainda mais relevante, pois a precisão dos cálculos monetários, a integridade dos registros de transações e a confiabilidade dos relatórios gerenciais são requisitos críticos do domínio. Qualquer inconsistência pode gerar impactos significativos em decisões

estratégicas, comprometer a integridade dos dados e prejudicar a confiança do usuário no sistema. Para o presente projeto, adotou-se uma estratégia de testes unitários em ambas as camadas da aplicação, prática viabilizada pela arquitetura em camadas descrita na seção 2.5, na qual a separação entre *controllers*, *services* e *repositories* permite que cada componente seja testado de forma isolada, com suas dependências substituídas por *test doubles*, garantindo assim a verificação independente das regras de negócio.

Na camada de *back-end*, utilizou-se o *framework* Pytest, em conjunto com a biblioteca pytest-mock para a simulação de repositórios e dependências externas. Foram cobertos os principais serviços de regras de negócio, incluindo autenticação (*AuthService*), categorias (*CategoriaService*), contas bancárias (*ContaBancariaService*), contas a pagar (*ContaPagarService*), empresas (*EmpresaService*) e transações (*TransacaoService*). Na camada de *front-end*, adotaram-se as ferramentas Karma e Jasmine, oficialmente suportadas pelo Angular, cobrindo serviços de comunicação HTTP, *guards* de autenticação, interceptadores, componentes compartilhados e *handlers* de erro. Essa estratégia de testes em múltiplos níveis amplia a cobertura de validação e contribui para a detecção precoce de falhas em diferentes pontos da aplicação. A Figura 9 apresenta um exemplo de teste unitário implementado no sistema.

Figura 9 – Teste unitário

```python
def test_receita_confirmada_credita_saldo_da_conta(self, service, repo):
    # ARRANGE – Preparação do cenário de teste
    conta_banco = make_conta(saldo_atual=Decimal("1000.00"))
    repo.criarTransacao.return_value = make_transacao(id=1)
    repo.session.get.return_value = conta_banco
    repo.buscarComRelacionamentos.return_value = make_transacao(
        id=1, transacao_id=int(TipoTransacaoEnum.RECEITA)
    )

    # ACT – Execução da operação sob teste
    service.criarTransacao(1, 1, _dados_transacao(
        tipo=TipoTransacaoEnum.RECEITA,
        situacao=SituacaoTransacaoEnum.CONFIRMADO,
        valor=Decimal("200.00"),
        conta_id=1,
    ))

    # ASSERT – Verificação do resultado esperado
    assert conta_banco.saldo_atual == Decimal("1200.00")
```

Fonte: Elaborado pelo autor (2026).

A Figura 9 apresenta o teste unitário *test_receita_confirmada_credita_saldo_da_conta*, responsável por validar uma das regras críticas do domínio: a atualização do saldo de uma conta bancária após o registro de uma transação de receita confirmada. O teste segue o padrão Arrange-Act-Assert e está estruturado em três fases bem delimitadas. Na fase *Arrange* (linhas 2 a 7), o cenário é preparado: uma conta bancária com saldo inicial de R$1.000,00 é criada, e o repositório é substituído por um *mock* que retorna objetos simulados, isolando o serviço de qualquer dependência real com o banco de dados. Na fase *Act* (linhas 10 a 15), o método *criarTransacao* é invocado com uma receita confirmada de R$ 200,00. Na fase *Assert* (linha 19), verifica-se que o saldo da conta foi corretamente atualizado para R$ 1.200,00, confirmando que a regra de crédito está funcionando conforme o esperado.

Cabe destacar o uso do tipo *Decimal* para representação dos valores monetários, em conformidade com as recomendações de Sommerville (2021) quanto à precisão em sistemas financeiros, evitando os erros de arredondamento característicos da aritmética de ponto flutuante. Por meio de testes como este, é possível garantir a integridade dos cálculos financeiros, oferecer maior confiabilidade às operações do sistema e prevenir regressões durante a manutenção evolutiva do *software*.

3 REFERENCIAL TECNOLÓGICO

Neste capítulo, são apresentadas as tecnologias que compõem o desenvolvimento da aplicação, detalhando as linguagens, *frameworks* e ferramentas utilizadas tanto no *front-end* quanto no *back-end* do sistema de gestão financeira. As tecnologias adotadas foram selecionadas com base em critérios de produtividade, segurança e escalabilidade, buscando criar uma solução eficiente, robusta e intuitiva para o usuário final.

3.1 Tecnologias usadas no front-end

O *front-end* é a camada do sistema responsável pela interface com o usuário, ou seja, pelo ambiente em que ele interage diretamente com o software (MDN WEB DOCS, 2025). Em projetos de sistemas, sites e aplicativos, o *front-end* e o *back-end* são partes complementares de uma aplicação completa. No *front-end*, além da preocupação com a estética e a usabilidade, é fundamental garantir que a interface seja responsiva e eficiente, fornecendo uma experiência de uso satisfatória ao usuário final (PRESSMAN; MAXIM, 2021).

As tecnologias utilizadas no desenvolvimento do *front-end* deste sistema trabalham de forma integrada e complementar. O *HTML5* (*HyperText Markup Language*) é responsável pela estruturação dos conteúdos das páginas, enquanto o *Sass*, um pré-processador de estilos, define a apresentação visual e o *layout*. A interatividade e a tipagem segura são fornecidas pelo *TypeScript*, linguagem que estende as capacidades do *JavaScript*. Por fim, o *Angular* atua como *framework* principal, organizando toda a aplicação em componentes modulares e garantindo escalabilidade, manutenibilidade e padronização do código. Tecnologias auxiliares como *RxJS*, *Angular Router*, *HttpClient*, *Karma*, *Jasmine* e *Vercel* complementam essa estrutura, oferecendo suporte à programação reativa, gerenciamento de rotas, comunicação HTTP, testes automatizados e implantação contínua da aplicação.

3.1.1 HTML5 (*HyperText Markup Language*)

O *HTML5* é a linguagem de marcação utilizada para estruturar e organizar conteúdos em páginas *web*, como textos, imagens, formulários e tabelas (W3C, 2025). No desenvolvimento do sistema, o *HTML5* foi utilizado para criar a estrutura

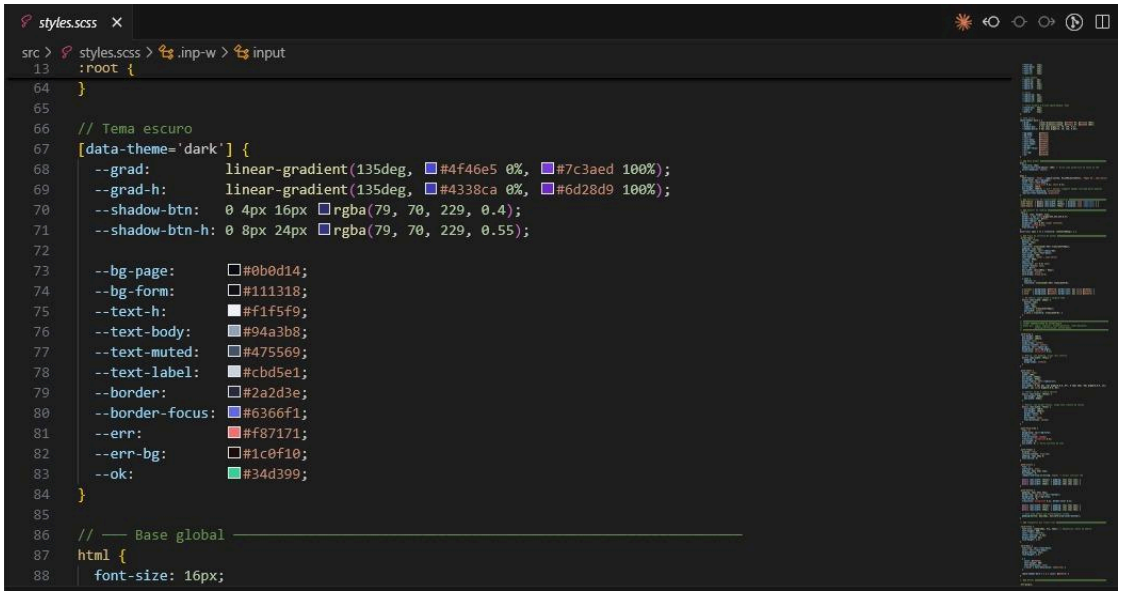
das telas, possibilitando a correta disposição de menus, *dashboards* e campos de entrada de dados financeiros. Seus elementos semânticos, como <*header*>, <*section*> e <*article*>, garantem maior acessibilidade e compatibilidade com diferentes navegadores, além de fornecer suporte a multimídia e armazenamento local (MDN WEB DOCS, 2025).

O *HTML5* foi adotado por ser o padrão atual da web, oficialmente recomendado pela W3C e suportado por todos os navegadores modernos. Sua versão atual oferece elementos semânticos como <*header*>, <*section*> e <*nav*>, que melhoram a acessibilidade e o SEO, além de APIs nativas para armazenamento local, multimídia e validação de formulários.

3.1.2 *Sass* (*Syntactically Awesome Style Sheets*)

O *Sass* é um pré-processador *CSS* que estende as funcionalidades do *CSS* tradicional, permitindo o uso de variáveis, *mixins*, aninhamento de seletores e organização modular do código de estilo (SASS, 2025). No sistema, o *Sass* foi aplicado para criar uma interface moderna e responsiva, adaptável a diferentes dispositivos e tamanhos de tela. Recursos como variáveis para padronização de cores e fontes, aninhamento de seletores e divisão dos estilos em arquivos parciais permitiram tornar o desenvolvimento mais organizado, reutilizável e de fácil manutenção, proporcionando uma experiência visual consistente ao usuário (MDN WEB DOCS, 2025). A Figura 10 apresenta um exemplo de código com *Sass*.

Figura 10 – Código em Sass



Fonte: Elaborado pelo autor (2026).

A Figura 10 apresenta um trecho de código *Sass* implementado no sistema, demonstrando a definição de variáveis para o tema escuro da aplicação. Por meio desse recurso, é possível padronizar cores, gradientes, sombras e tipografia, contribuindo para a organização do código e a manutenção visual consistente da interface.

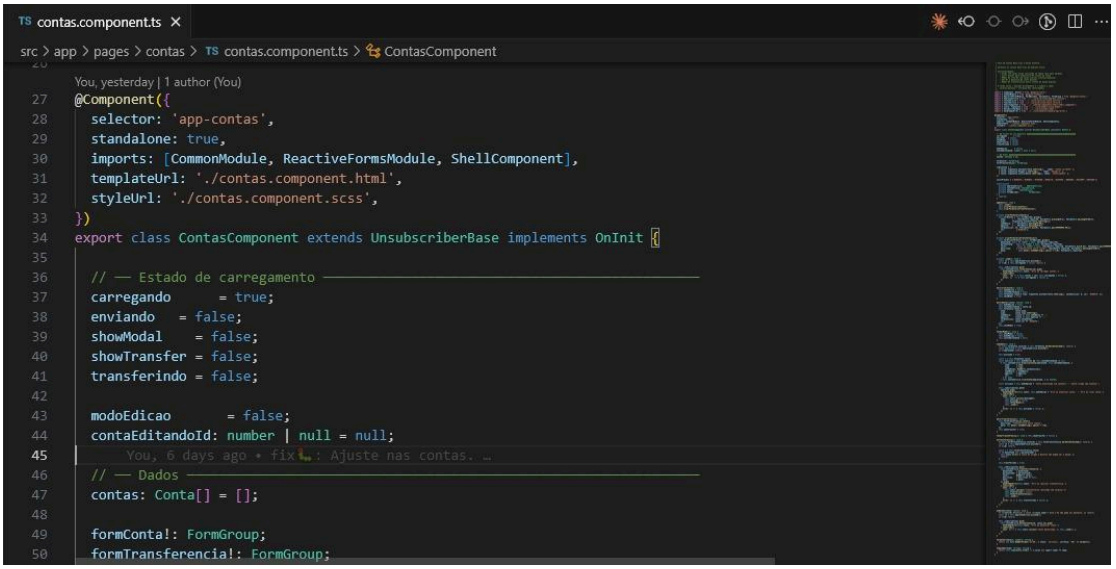
A escolha do Sass em vez do *CSS* puro se justifica pela necessidade de manter um sistema de design coerente em uma aplicação com mais de 20 telas. Recursos como variáveis, mixins e aninhamento de seletores permitiram centralizar a paleta de cores e a tipografia do tema escuro em um único arquivo, garantindo consistência visual em toda a aplicação. Alternativas como *CSS-in-JS (styled-components)* ou *Tailwind CSS* foram avaliadas, mas o *Sass* venceu por integrar-se nativamente ao Angular sem dependências adicionais.

3.1.3 *TypeScript*

O *TypeScript* é uma linguagem que estende o *JavaScript*, adicionando tipagem estática e recursos de orientação a objetos, aumentando a organização e a segurança do código (MICROSOFT, 2024). No desenvolvimento do sistema, o *TypeScript* foi utilizado junto ao *Angular*, possibilitando a criação de componentes modulares, reutilizáveis e de fácil manutenção. A tipagem estática facilita a

identificação de erros durante o desenvolvimento, tornando o sistema mais confiável, o que é fundamental em aplicações que lidam com dados financeiros sensíveis (MDN WEB DOCS, 2025). A Figura 11 ilustra um trecho de código desenvolvido em *TypeScript*.

Figura 11 – Trecho de código em TypeScript



Fonte: Elaborado pelo autor (2026).

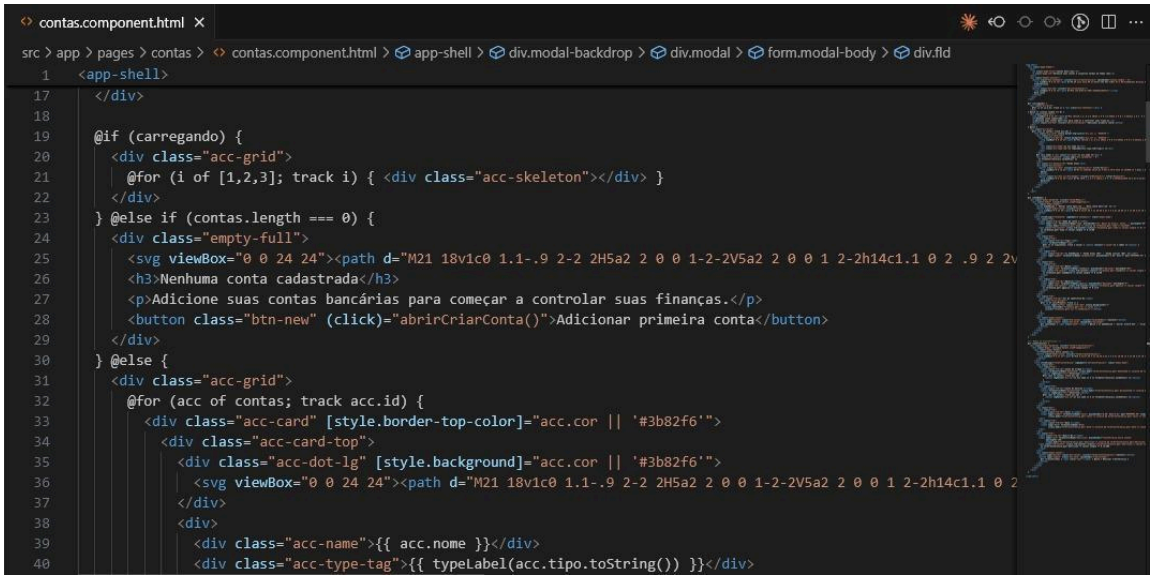
A Figura 11 apresenta um trecho de código *TypeScript* implementado no sistema, demonstrando a estrutura de um componente *Angular*. Por meio da tipagem estática e do uso de decoradores, é possível organizar o código em componentes reutilizáveis e identificar erros ainda na fase de desenvolvimento, contribuindo para a manutenção e a confiabilidade do sistema.

O *TypeScript* foi escolhido em detrimento do *JavaScript* puro pela necessidade de garantir robustez em uma aplicação que manipula dados financeiros sensíveis. A tipagem estática permite identificar erros de tipo ainda em tempo de desenvolvimento antes que cheguem ao usuário, fornece autocompletar inteligente nos editores e documenta implicitamente o código. Em sistemas financeiros, onde um erro de tipo pode significar um cálculo incorreto, essa segurança adicional é especialmente valiosa.

3.1.4 *Angular*

O *Angular* é um *framework front-end* mantido pelo Google, destinado à criação de *Single Page Applications* (SPA) que garantem carregamento rápido e navegação fluida (GOOGLE, 2024). Utilizando *TypeScript* como base, o *Angular* permite desenvolver aplicações modulares e escaláveis, com componentes reutilizáveis e integração eficiente com serviços de *back-end* (MDN WEB DOCS, 2025). No sistema de gestão financeira, o *Angular* foi utilizado para organizar a interface, implementar rotas e atualizar dados dinamicamente, garantindo uma experiência consistente e responsiva ao usuário. A Figura 12 ilustra um trecho de código desenvolvido em *Angular*.

Figura 12 – Trecho de código em Angular



Fonte: Elaborado pelo autor (2026).

A Figura 12 apresenta um trecho de template *Angular* implementado no sistema, demonstrando a estrutura responsável por exibir a listagem de contas bancárias. Por meio das diretivas de controle de fluxo (*@if*, *@else*, *@for*) e da vinculação de dados a propriedades dos componentes, é possível organizar a disposição visual das informações de forma dinâmica e responsiva, garantindo uma experiência consistente ao usuário.

O *Angular* foi escolhido entre as três principais alternativas do mercado *Angular, React e Vue* por três razões. Primeiro, por ser um *framework* completo, com

soluções nativas para roteamento, formulários, injeção de dependência e comunicação *HTTP*, eliminando a necessidade de avaliar e integrar bibliotecas terceiras. Segundo, pelo suporte nativo a *TypeScript* desde sua versão 2, garantindo a tipagem forte que o domínio financeiro exige. Terceiro, por sua arquitetura baseada em componentes e módulos, que facilita a manutenção de longo prazo do sistema.

3.1.5 RxJS

O *RxJS* (*Reactive Extensions for JavaScript*) é uma biblioteca de programação reativa baseada em *Observables*, amplamente integrada ao *Angular* (RXJS, 2024). A programação reativa permite representar fluxos assíncronos de dados — como respostas HTTP, eventos de interface e mudanças de estado — de forma declarativa, utilizando operadores como *map*, *filter*, *switchMap* e *catchError* para compor as transformações sobre esses fluxos (RXJS, 2024).

No projeto, o *RxJS* foi utilizado em praticamente todos os serviços responsáveis pela comunicação com o servidor, como o de autenticação, de transações e de relatórios, representando as chamadas HTTP de forma reativa. O *BehaviorSubject* foi empregado para manter o estado do usuário autenticado e do *access token*, disponibilizando essas informações de maneira reativa aos componentes interessados. O operador *switchMap* foi aplicado no interceptor de autenticação para encadear a renovação do *token* de acesso sempre que uma requisição retornasse status HTTP 401, garantindo a continuidade da sessão sem interrupção para o usuário.

O *RxJS* foi adotado para gerenciar o fluxo assíncrono de dados de forma declarativa, em vez de utilizar apenas *Promises* ou *callbacks*. Sua abordagem baseada em *Observables* permite combinar, filtrar e transformar fluxos de dados de maneira composicional algo particularmente útil no interceptor de autenticação, que precisa interceptar requisições com erro 401, renovar o token e refazer a requisição de forma transparente. Como o *RxJS* é integrado nativamente ao *Angular*, sua adoção não acrescenta dependências adicionais ao projeto.

3.1.6 Angular Router e lazy loading

O *Angular Router* é o módulo responsável pelo gerenciamento da navegação entre as diferentes telas da aplicação, permitindo organizar rotas, aplicar regras de

proteção e carregar conteúdos sob demanda (GOOGLE, 2024). No sistema, as rotas foram declaradas em um arquivo central e organizadas em duas categorias: rotas públicas, como *login*, registro, recuperação de senha e verificação de e-mail, e rotas protegidas, como *dashboard*, contas, transações, contas a pagar, contas a receber, fluxo de caixa, análise, relatórios, categorias, assistente de inteligência artificial e configurações.

O acesso às rotas protegidas é controlado por meio de guardas de rota (*guards*): o *authGuard* verifica se o usuário possui um *token* de acesso válido, enquanto o *empresaGuard* valida a existência de uma empresa ativa selecionada. Todos os componentes de página são carregados sob demanda por meio da função *loadComponent()*, técnica conhecida como *lazy loading*, que fragmenta o pacote final em arquivos *JavaScript* menores e reduz o tempo de carregamento inicial percebido pelo usuário (GOOGLE, 2024).

O *Angular Router* foi adotado por ser a solução nativa para roteamento no Angular, integrada aos mecanismos de *guards* e *resolvers* do *framework*. A técnica de *lazy loading* se justifica pela quantidade de telas do sistema. Em vez de carregar todo o código *JavaScript* na primeira visita, o *lazy loading* divide a aplicação em *bundles* menores, carregados sob demanda. Isso reduz o tempo de carregamento inicial e melhora a experiência percebida pelo usuário, especialmente em conexões mais lentas.

3.1.7 *HttpClient* e interceptores HTTP

A comunicação entre o *front-end* e o *back-end* é realizada pelo *HttpClient* do *Angular*, módulo responsável pela execução das requisições HTTP a partir da aplicação (GOOGLE, 2024). Sua configuração foi feita por meio da função *provideHttpClient*, que permite registrar uma lista de interceptores HTTP a serem aplicados em todas as requisições. Os interceptores são funções executadas de forma centralizada em cada requisição enviada, possibilitando a aplicação de transformações sem necessidade de repetição de código nos serviços.

No projeto foi implementado o *authInterceptor*, responsável por duas tarefas críticas. A primeira consiste em injetar automaticamente o cabeçalho de autorização contendo o *token* de acesso em todas as requisições destinadas à API, dispensando os serviços individuais de tratarem essa lógica. A segunda consiste em capturar

respostas com status HTTP 401, indicativas de *token* expirado, e tentar renová-lo por meio do *endpoint* de *refresh*. Caso a renovação falhe, o interceptor realiza o *logout* do usuário e o redireciona para a tela de *login*, evitando que requisições não autorizadas continuem sendo realizadas indefinidamente.

O *HttpClient* foi escolhido por ser a solução nativa do *Angular* para requisições *HTTP*, com integração natural ao *RxJS*. Os interceptores em particular se justificam pela necessidade de centralizar a lógica de autenticação: em vez de adicionar manualmente o token *JWT* em cada chamada de serviço, o interceptor injeta automaticamente o cabeçalho de autorização em todas as requisições. Essa abordagem segue o princípio DRY *(Don't Repeat Yourself)* e reduz drasticamente o risco de esquecer essa lógica em algum ponto da aplicação.

3.1.8 *Karma* e *Jasmine*

Para a escrita e execução de testes automatizados no *front-end* foi adotada a *stack* padrão do *Angular*, composta pelo *Jasmine* e pelo *Karma* (GOOGLE, 2024). O *Jasmine* é um *framework* de escrita de testes baseado no paradigma de *behavior-driven development*, que organiza os testes em blocos descritivos e legíveis, facilitando o entendimento do comportamento esperado de cada funcionalidade (JASMINE, 2024). O *Karma*, por sua vez, atua como executor de testes, orquestrando a execução em navegadores reais ou em modo *headless*, fornecendo retorno imediato sobre o resultado de cada execução (KARMA, 2024).

No sistema, os testes de componente e de serviço foram organizados em arquivos com sufixo *.spec.ts*, mantidos próximos aos arquivos que testam. Essa organização favorece a manutenção do código e permite verificar rapidamente o comportamento das principais rotinas implementadas, contribuindo para a confiabilidade do *software*.

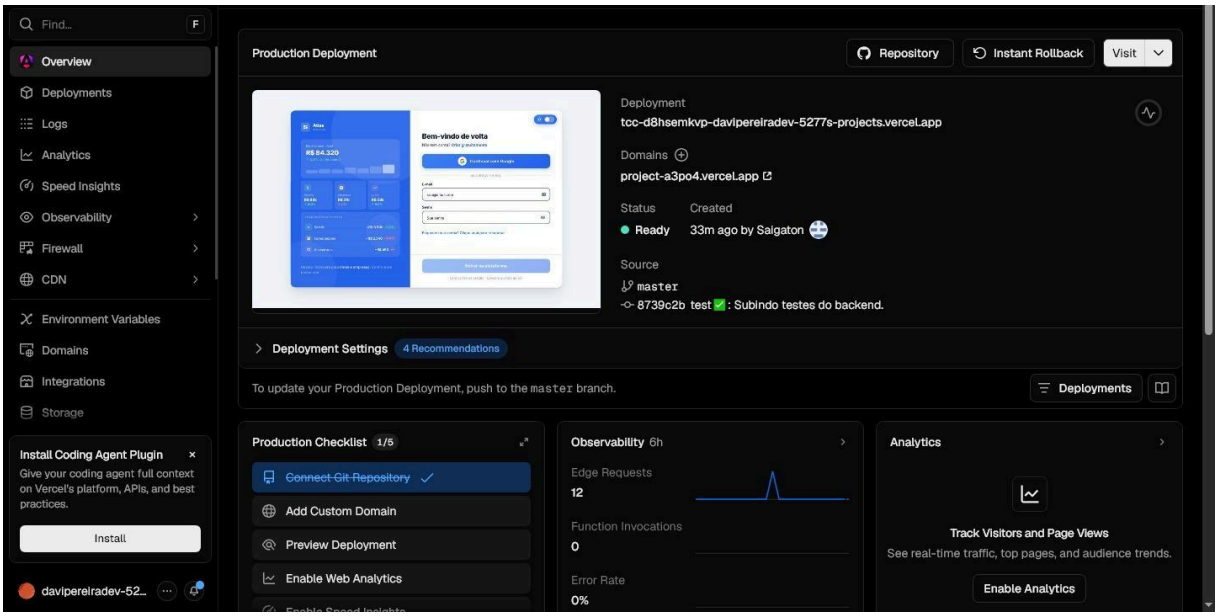
Karma e *Jasmine* foram escolhidos por serem a *stack* oficial de testes do *Angular*, recomendada na documentação do próprio *framework*. Essa decisão dispensou configurações adicionais, pois ambos vêm pré-configurados no projeto criado pelo *Angular CLI*. Alternativas como *J*est seriam tecnicamente válidas, mas exigiriam reconfigurações que não trariam benefícios proporcionais ao esforço.

3.1.9 *Vercel*

A *Vercel* é uma plataforma de implantação de aplicações *front-end* com suporte nativo a *frameworks* modernos, integração com repositórios *Git* e *CDN* global para distribuição de conteúdo estático (VERCEL, 2025). Sua utilização permite a publicação contínua de novas versões da aplicação a partir de cada atualização realizada no repositório, tornando o processo de implantação mais ágil e padronizado.

No projeto, foi incluído um arquivo de configuração que define o roteamento da aplicação como *Single Page Application*, redirecionando todas as rotas não correspondentes a arquivos físicos para o *index.html*. Esse comportamento é essencial para o funcionamento correto do *Angular Router* em produção, garantindo que os usuários possam acessar diretamente qualquer rota da aplicação sem encontrar erros de páginas inexistentes. A Figura 13 apresenta o painel de gerenciamento da *Vercel* utilizado para a implantação do sistema.

Figura 13 – Painel de gerenciamento na vercel



Fonte: Elaborado pelo autor (2026).

A Figura 13 apresenta o painel de gerenciamento da *Vercel* utilizado para a implantação do sistema, demonstrando o ambiente de *production deployment*, o histórico de versões e o monitoramento da aplicação. Por meio dessa interface, é possível acompanhar o status das publicações, visualizar a aplicação em execução,

gerenciar domínios e analisar métricas de uso, contribuindo para a manutenção e o acompanhamento contínuo do sistema em produção.

A *Vercel* foi escolhida para a hospedagem do *front-end* por três motivos. Primeiro, por oferecer plano gratuito robusto. Segundo, pela integração nativa com o *GitHub*, permitindo *deploys* automáticos a cada *push* na *branch* principal. Terceiro, por sua *CDN* global, que distribui o conteúdo estático em servidores próximos ao usuário, garantindo carregamento rápido. Alternativas como *Netlify* e *GitHub Pages* foram consideradas, mas a *Vercel* se destacou pelo suporte específico a aplicações *Angular* configuradas como SPA.

3.2 Tecnologias usadas no *back-end*

O *back-end* é a camada do sistema responsável pelo processamento de dados, pelas regras de negócio e pelo armazenamento seguro das informações, atuando como o servidor que fornece os recursos consumidos pelo *front-end* (MDN WEB DOCS, 2025). É nessa camada que estão implementados os mecanismos de autenticação, a validação dos dados recebidos, as regras de negócio da aplicação e a comunicação com o banco de dados, sendo fundamental garantir desempenho, segurança e confiabilidade para que as operações realizadas pelos usuários sejam executadas de forma consistente e íntegra (PRESSMAN; MAXIM, 2021).

As tecnologias utilizadas no desenvolvimento do *back-end* deste sistema trabalham de forma integrada e complementar. O *Python* é a linguagem de programação adotada, escolhida por sua simplicidade, legibilidade e amplo ecossistema de bibliotecas, enquanto o *FastAPI* atua como *framework* principal, responsável pela construção das *APIs* que conectam o servidor à interface do usuário. A organização do código foi estruturada por meio da arquitetura em três camadas (*three-tier architecture*), separando claramente apresentação, regras de negócio e persistência de dados. Tecnologias complementares como o *Pydantic*, utilizado para a validação automática dos dados; o *SQLAlchemy*, que atua como mapeador objeto-relacional; e o *Alembic*, responsável pelo versionamento do esquema do banco de dados, integram a infraestrutura do *back-end*, garantindo desenvolvimento ágil, manutenibilidade e confiabilidade da aplicação.

3.2.1 *Python*

O *Python* é uma linguagem de programação de alto nível, dinâmica, interpretada e multiparadigma, conhecida por sua simplicidade, legibilidade e versatilidade (PYTHON SOFTWARE FOUNDATION, 2024). Sua sintaxe clara e próxima da linguagem natural permite o desenvolvimento ágil de aplicações, reduzindo o tempo de codificação e facilitando a manutenção do software (HUB ASIMOV, 2025). No sistema de gestão financeira, o *Python* foi utilizado para implementar a lógica de negócios responsável pelo processamento dos lançamentos financeiros, pelo cálculo de saldos, pela consolidação de informações para relatórios gerenciais e pela integração com o banco de dados.

A linguagem se destaca também por sua ampla comunidade e por um vasto ecossistema de bibliotecas, que oferece soluções prontas para tarefas como manipulação de dados, validação, autenticação e geração de relatórios. Essa característica contribuiu para acelerar o desenvolvimento do projeto e para garantir maior robustez nas operações realizadas pelo sistema (PYTHON SOFTWARE FOUNDATION, 2024).

O *Python* foi escolhido por equilibrar produtividade, legibilidade e robustez. Sua sintaxe clara reduz o tempo de codificação em comparação com linguagens como *Java* ou C*#,* enquanto seu vasto ecossistema de bibliotecas oferece soluções maduras para todos os domínios necessários ao projeto: validação de dados (*Pydantic*), ORM (*SQLAlchemy*), autenticação (*PyJWT*, *Passlib*), testes (*Pytest*) e agendamento (*APScheduler*). Linguagens como *Node.js* também foram consideradas, mas o *Python* venceu pela maior aderência da comunidade *Python* ao desenvolvimento de APIs financeiras e ao processamento numérico com precisão decimal.

3.2.2 *FastAPI*

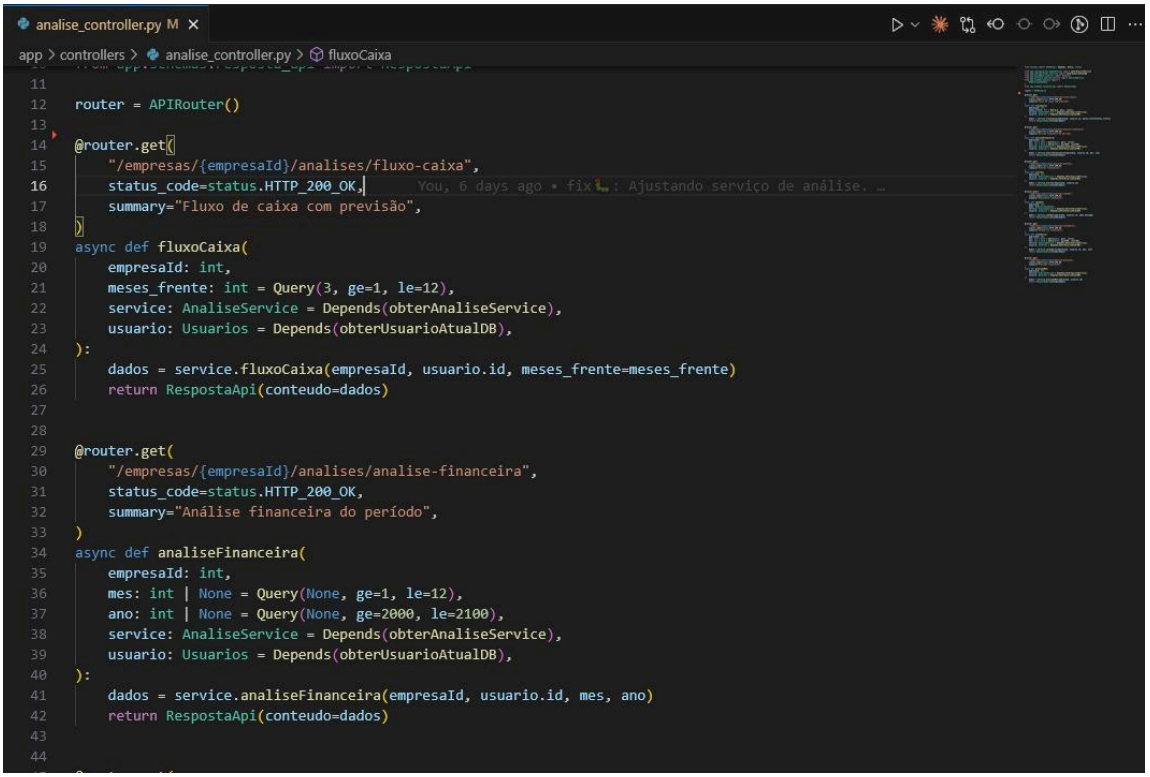
O *FastAPI* é um *framework Python* de alto desempenho voltado à construção de *APIs RESTful*, baseado em padrões modernos como *type hints* do *Python*, *Pydantic* para validação de dados e *Starlette* para o tratamento assíncrono de requisições (FASTAPI, 2024). Esse conjunto de tecnologias oferece desempenho comparável ao de linguagens compiladas, sendo uma das opções mais utilizadas

para o desenvolvimento de *APIs* em ambientes que exigem rapidez e escalabilidade (FASTAPI, 2024).

No sistema, o *FastAPI* foi empregado para criar as interfaces de comunicação entre o *front-end* e a camada de dados, expondo *endpoints* responsáveis pelas operações de cadastro, edição, consulta e exclusão de lançamentos financeiros. O *framework* também foi utilizado na implementação da autenticação por *JWT* e na validação automática dos dados recebidos nas requisições, contribuindo para a segurança e a integridade das informações.

Entre as principais vantagens do *FastAPI* estão a geração automática de documentação interativa por meio do *Swagger UI* e do *Redoc*, a validação estruturada de dados de entrada e saída e o suporte nativo a operações assíncronas. Essas características tornaram o *framework* adequado para o desenvolvimento de um sistema financeiro confiável, ágil e bem documentado (FASTAPI, 2024). A Figura 14 ilustra um trecho de código desenvolvido em *FastAPI.*

Figura 14 – Trecho de código em *FastAPI*



Fonte: Elaborado pelo autor (2026).

A Figura 14 apresenta um trecho de código *FastAPI* implementado no sistema, demonstrando a definição de *endpoints* para análise financeira. Por meio

dos decoradores e da injeção de dependências, é possível organizar as rotas de forma clara e modular, com validação automática dos parâmetros recebidos nas requisições.

O *FastAPI* foi escolhido em detrimento de *Django* e *Flask* por três vantagens decisivas. Primeiro, performance superior por ser baseado em ASGI (*Asynchronous Server Gateway Interface*), permitindo o tratamento assíncrono de requisições. Segundo, geração automática de documentação *OpenAPI/Swagger*, eliminando a necessidade de manter a documentação manualmente. Terceiro, tipagem forte integrada ao *Pydantic*, garantindo validação automática de dados de entrada e saída. *Django* seria excessivo para a API necessária, e *Flask* exigiria muitas bibliotecas adicionais para alcançar o mesmo nível de produtividade oferecido pelo *FastAPI* de fábrica.

3.2.3 Arquitetura em camadas (*three-tier architecture*)

A arquitetura em três camadas (*three-tier architecture*) é um padrão arquitetural que separa a aplicação em três camadas independentes: a camada de apresentação, responsável pela interface com o usuário; a camada de lógica de negócios, que processa as regras e fluxos da aplicação; e a camada de dados, encarregada da persistência das informações (VALENTE, 2020). Essa separação facilita a manutenção, o reuso de código e o trabalho em equipe, permitindo que cada camada evolua de forma independente sem comprometer as demais (IBM, 2025).

No *back-end* do sistema, essa arquitetura foi implementada por meio do padrão *Controller* → *Service* → *Repository*. O *Controller* é responsável por receber as requisições HTTP, validar os parâmetros recebidos e direcionar a chamada para a camada correspondente. O *Service* concentra a lógica de negócio, aplicando as regras necessárias ao processamento das informações financeiras. Já o *Repository* realiza o acesso ao banco de dados por meio do *SQLAlchemy*, abstraindo as operações de persistência (VALENTE, 2020).

Essa organização garante que cada camada tenha uma única responsabilidade, em conformidade com o princípio da separação de responsabilidades, que é considerado um pilar fundamental da engenharia de software moderna (VALENTE, 2020). Como resultado, o sistema apresenta maior

coesão, menor acoplamento e maior facilidade na realização de testes unitários, uma vez que cada camada pode ser testada de forma isolada (SOMMERVILLE, 2021). A Figura 15 ilustra a organização da arquitetura em três camadas adotada no projeto.

Figura 15 – Arquitetura em três camadas

Fonte: Elaborado pelo autor (2026).

A Figura 15 representa a arquitetura em camadas adotada no projeto, organizada em quatro níveis com responsabilidades bem definidas. A camada de apresentação cuida da interface com o usuário; a de Negócio concentra as regras e a lógica da aplicação; a de Dados gerencia o acesso e a persistência no banco; e a de Infraestrutura fornece os serviços externos, como integrações com *APIs*, e-mail e gerenciamento de configurações.

3.2.4 *Pydantic*

O *Pydantic* é uma biblioteca de validação de dados para *Python*, baseada em anotações de tipo (*type hints*) (PYDANTIC, 2024). Sua proposta é permitir que

estruturas de dados sejam declaradas como classes tipadas e que a validação ocorra automaticamente em tempo de execução, sem necessidade de implementação manual de regras de verificação. Quando os dados recebidos não correspondem ao tipo esperado, a biblioteca gera mensagens de erro detalhadas e padronizadas, contribuindo para o aumento da confiabilidade das aplicações que dependem de entradas externas.

A versão 2 do *Pydantic*, lançada em 2023, teve seu núcleo de validação reescrito na linguagem *Rust*, o que resultou em ganhos significativos de desempenho em relação à versão anterior, mantendo a compatibilidade da API pública (PYDANTIC, 2024). A biblioteca também oferece recursos como geração automática de *JSON Schema*, serialização de modelos para diferentes formatos, modo estrito de validação e suporte a tipos personalizados. Por essas características, o *Pydantic* tornou-se uma das bibliotecas mais utilizadas em aplicações *Python* modernas, sendo adotada como base para diversos *frameworks*, entre eles o *FastAPI*.

O *Pydantic* foi adotado por ser a biblioteca padrão de validação de dados no ecossistema *FastAPI*, com integração nativa que dispensa configuração adicional. Sua principal vantagem em relação a validações manuais ou bibliotecas alternativas é a geração automática de erros estruturados, mensagens detalhadas e documentação *OpenAPI* a partir das próprias classes. A versão 2 da biblioteca, com núcleo reescrito em *Rust*, oferece desempenho até 50 vezes superior à versão anterior.

3.2.5 SQLAlchemy

O *SQLAlchemy* é uma biblioteca de código aberto que atua como *toolkit* SQL e como mapeador objeto-relacional (*Object-Relational Mapping — ORM*) para a linguagem *Python* (SQLALCHEMY, 2024). Sua arquitetura é dividida em duas camadas complementares: o *Core*, que fornece uma linguagem expressiva para construção de consultas SQL diretamente em *Python*, e o *ORM*, que permite mapear classes *Python* para tabelas de bancos de dados relacionais, abstraindo a manipulação direta de SQL nas operações de leitura e escrita.

O uso de um *ORM* contribui para a redução do acoplamento entre a camada de aplicação e o banco de dados, favorecendo a manutenibilidade do código e a

portabilidade entre diferentes sistemas gerenciadores de banco de dados (SOMMERVILLE, 2021). A versão 2.0 do *SQLAlchemy* introduziu uma API unificada baseada em tipagem estática, com a sintaxe *Mapped[...]* e *mapped_column()*, alinhando-se às boas práticas modernas da linguagem *Python* e permitindo melhor integração com ferramentas de análise estática de código (SQLALCHEMY, 2024).

O *SQLAlchemy* foi escolhido por ser o ORM mais maduro do ecossistema *Python*, com mais de 15 anos de desenvolvimento ativo. Sua adoção evitou a necessidade de escrever SQL manualmente em todas as operações de persistência, reduzindo o risco de erros sintáticos e ataques de *SQL Injection*. Alternativas como o Tortoise ORM ou o *Peewee* foram consideradas, mas o *SQLAlchemy* venceu pela maturidade, pela compatibilidade com o *Alembic* (sua ferramenta oficial de migração) e pela API tipada introduzida na versão 2.0, que integra perfeitamente com o Pydantic e o *FastAPI*.

3.2.6 *Alembic*

O *Alembic* é uma ferramenta de migração de esquema de banco de dados associada ao *SQLAlchemy*, desenvolvida pelo mesmo autor da biblioteca (ALEMBIC, 2024). Seu objetivo é permitir que a estrutura do banco de dados seja versionada por meio de *scripts* incrementais, denominados migrações, que descrevem de forma programática as alterações aplicadas ao esquema relacional ao longo do tempo.

O versionamento da estrutura do banco de dados é considerado uma boa prática em engenharia de software, pois garante que diferentes ambientes — como desenvolvimento, homologação e produção — mantenham o mesmo esquema relacional, além de tornar as alterações rastreáveis e reversíveis (SOMMERVILLE, 2021). O *Alembic* gera automaticamente as migrações a partir das diferenças entre o modelo definido no código e o estado atual do banco, oferecendo também a possibilidade de edição manual dos *scripts* para casos em que a detecção automática não é suficiente (ALEMBIC, 2024).

O *Alembic* foi escolhido por ser a ferramenta oficial de migração do *SQLAlchemy*, desenvolvida pelo mesmo autor da biblioteca, garantindo total compatibilidade. Sua adoção permite versionar a estrutura do banco de dados como

código, com scripts incrementais armazenados junto ao repositório do projeto. Em comparação com a aplicação manual de SQL via interface administrativa.

3.2.7 APScheduler

O *APScheduler* (*Advanced Python Scheduler*) é uma biblioteca de código aberto destinada ao agendamento de tarefas em aplicações *Python*, permitindo a execução de funções de forma periódica, em datas específicas ou em intervalos definidos por expressões *cron* (APSCHEDULER, 2024). Sua arquitetura é composta por quatro componentes principais: *triggers*, responsáveis por definir quando as tarefas devem ser executadas; *job stores*, encarregados de armazenar as tarefas agendadas; *executors*, que executam as tarefas em diferentes contextos; e *schedulers*, que coordenam todos os demais componentes (APSCHEDULER, 2024).

O agendamento automatizado de tarefas é considerado uma prática essencial em sistemas de software que necessitam realizar operações em segundo plano de forma autônoma, como geração de relatórios, envio de notificações e execução de rotinas de manutenção (SOMMERVILLE, 2021). O uso de bibliotecas especializadas para esse fim contribui para a automação de processos, a redução de intervenções manuais e a garantia de execução confiável de tarefas críticas, fortalecendo a robustez e a previsibilidade das aplicações (PRESSMAN; MAXIM, 2021).

O *APScheduler* foi escolhido para o agendamento de tarefas em segundo plano por sua simplicidade e adequação ao porte do projeto. A funcionalidade de envio mensal de relatórios não exige a complexidade de soluções distribuídas como *Celery* com *Redis*, que demandariam infraestrutura adicional. O *APScheduler* executa em memória do próprio processo da aplicação, com armazenamento das tarefas no próprio banco *PostgreSQL*, oferecendo confiabilidade suficiente para o caso de uso sem agregar complexidade operacional. Em um cenário de crescimento futuro, a migração para *Celery* seria uma evolução natural, não uma reescrita.

3.2.8 Pytest

O *Pytest* é um *framework* de testes para *Python*, amplamente adotado pela comunidade de desenvolvimento de software por sua sintaxe simples, baseada em funções, e por sua extensibilidade por meio de *plugins* (PYTEST, 2024). A ferramenta oferece recursos como *fixtures*, que permitem a reutilização de contextos

de teste; parametrização, que possibilita a execução do mesmo teste com diferentes conjuntos de dados; e reescrita de *assertions*, que fornece mensagens de erro detalhadas quando ocorrem falhas, facilitando a identificação rápida de problemas no código (PYTEST, 2024).

A escrita de testes automatizados é considerada uma das principais práticas da engenharia de software moderna, contribuindo para a redução de defeitos, o aumento da confiabilidade e a manutenibilidade do código (SOMMERVILLE, 2021). Testes unitários, em particular, permitem verificar o comportamento de unidades isoladas do software, como funções e classes, sendo fundamentais para garantir que as alterações realizadas durante o desenvolvimento não introduzam regressões no sistema (PRESSMAN; MAXIM, 2021).

O *Pytest* foi escolhido em detrimento do *framework unittest* da biblioteca padrão por sua sintaxe mais expressiva e pelo ecossistema de plugins. Sua abordagem baseada em funções (em vez de classes herdadas) torna os testes mais legíveis, e recursos como *fixtures* permitem reaproveitar contextos de teste de forma elegante. O plugin *pytest-mock* foi adotado em conjunto para a criação de *test doubles*, possibilitando isolar os serviços de suas dependências externas. Essa combinação é considerada padrão de mercado no desenvolvimento com *Python* moderno.

3.2.9 *Resend*

O *Resend* é uma plataforma de envio de e-mails transacionais voltada a desenvolvedores, que oferece uma *API REST* e *SDKs* oficiais para diversas linguagens de programação (RESEND, 2024). Os e-mails transacionais, diferentemente dos e-mails de marketing, são mensagens enviadas de forma individual em resposta a ações específicas realizadas pelos usuários, como confirmações de cadastro, recuperação de senha, notificações de movimentações e envio de relatórios automáticos (RESEND, 2024).

A utilização de plataformas especializadas para envio de e-mails, em vez de servidores próprios, contribui para garantir alta taxa de entrega, segurança das mensagens, autenticação adequada e monitoramento detalhado dos eventos relacionados a cada envio (RESEND, 2024). Em sistemas que demandam confiabilidade no envio de notificações, a adoção de mecanismos como tentativas

sucessivas de envio (*retry*) e tratamento de falhas transitórias é recomendada para aumentar a resiliência da aplicação diante de instabilidades em serviços externos (PRESSMAN; MAXIM, 2021).

O *Resend* foi escolhido entre as plataformas de envio transacional de e-mails por sua simplicidade de integração e plano gratuito generoso. Sua *API REST* permite enviar e-mails com poucas linhas de código, e o monitoramento integrado fornece visibilidade sobre a entrega de cada mensagem. A decisão de usar uma plataforma especializada, em vez de configurar um servidor *SMTP* próprio, foi essencial para garantir alta taxa de entrega.

3.2.10 Render

A hospedagem em nuvem (*cloud hosting*) consiste no provisionamento de recursos computacionais por meio de plataformas que abstraem a complexidade da infraestrutura física, permitindo que desenvolvedores publiquem e escalem aplicações sem a necessidade de gerenciar diretamente servidores, sistemas operacionais ou componentes de rede (SOMMERVILLE, 2021). No contexto contemporâneo, soluções de Plataforma como Serviço (*Platform as a Service* — PaaS) têm se consolidado como uma alternativa eficiente para o desenvolvimento ágil, uma vez que combinam recursos de implantação automatizada, escalabilidade sob demanda e integração contínua a partir de repositórios de código (PRESSMAN; MAXIM, 2021).

O *Render* é uma plataforma de nuvem que oferece um modelo unificado para a implantação de aplicações *web*, serviços de *back-end*, tarefas em segundo plano e bancos de dados gerenciados, por meio de uma infraestrutura que abstrai o provisionamento manual, fornecendo de forma automática certificados TLS, balanceadores de carga e instâncias de computação (RENDER, 2026). A plataforma realiza a integração direta com repositórios de código hospedados em serviços como *GitHub*, *GitLab* e *Bitbucket*, possibilitando a configuração de implantações automáticas a cada novo *commit* enviado à *branch* monitorada (RENDER, 2026).

Entre os recursos disponibilizados pela plataforma destacam-se a verificação automática de integridade dos serviços por meio de *health checks*, o reescalonamento horizontal baseado em utilização de CPU e memória, a possibilidade de reversão imediata para versões anteriores em caso de falhas e o

suporte a variáveis de ambiente para o armazenamento seguro de credenciais e parâmetros sensíveis de configuração (RENDER, 2026). Esses mecanismos contribuem para a confiabilidade, a disponibilidade e a manutenibilidade das aplicações hospedadas, atributos considerados essenciais em sistemas de produção segundo a literatura de engenharia de software (SOMMERVILLE, 2021).

O *Render* foi escolhido para hospedagem do *back-end* por oferecer plano gratuito, suporte nativo a aplicações *Python* com *FastAPI* e deploy automático a partir do *GitHub*. Alternativas como *Heroku*, anteriormente popular, perderam o plano gratuito em 2022. Plataformas como *AWS*, *Google Cloud* e *Azure* ofereceriam maior escalabilidade mas exigiriam configuração manual de infraestrutura. O *Render* abstrai essa complexidade, fornecendo *HTTPS*, balanceador de carga e variáveis de ambiente seguras automaticamente.

3.2.11 Supabase

O armazenamento de dados em nuvem constitui uma das principais demandas no desenvolvimento de aplicações *web* modernas, exigindo soluções que ofereçam, simultaneamente, escalabilidade, alta disponibilidade e segurança no acesso às informações persistidas (SOMMERVILLE, 2021). Nesse cenário, plataformas baseadas no modelo *Backend as a Service* (BaaS) têm se consolidado como uma alternativa eficiente para o provisionamento de bancos de dados, autenticação e serviços auxiliares, reduzindo significativamente o tempo de desenvolvimento e a complexidade da infraestrutura associada (PRESSMAN; MAXIM, 2021).

O *Supabase* é uma plataforma de desenvolvimento de código aberto que oferece, como núcleo de sua arquitetura, um banco de dados *PostgreSQL* totalmente gerenciado, complementado por serviços de autenticação baseada em *tokens* JWT, armazenamento de arquivos, funções em tempo real e geração automática de APIs REST a partir do esquema do banco (SUPABASE, 2026). A plataforma foi concebida como uma alternativa de código aberto a soluções proprietárias do mesmo segmento, mantendo a aderência integral ao padrão *PostgreSQL* e permitindo que aplicações utilizem todas as funcionalidades nativas do sistema gerenciador de banco de dados (SUPABASE, 2026).

Entre os recursos disponibilizados pelo *Supabase* destacam-se a realização automática de cópias de segurança (*backups*), o suporte a extensões nativas do *PostgreSQL*, a disponibilização de uma interface gráfica de administração e o controle de acesso por meio do mecanismo de *Row Level Security*, que permite definir, em nível de registro, quais usuários têm permissão para visualizar ou modificar cada linha das tabelas (SUPABASE, 2026). Esses mecanismos contribuem para a integridade, a confidencialidade e a disponibilidade das informações armazenadas, atributos considerados essenciais em sistemas que manipulam dados sensíveis, como aplicações de gestão financeira (SOMMERVILLE, 2021).

O *Supabase* foi escolhido para hospedar o *PostgreSQL* por três razões. Primeiro, por oferecer um plano gratuito robusto, com *backups* automáticos diários. Segundo, pelo painel administrativo gráfico, que facilitou inspeções pontuais sobre os dados sem necessidade de instalar ferramentas adicionais como *pgAdmin*. Terceiro, por manter aderência total ao padrão *PostgreSQL*, permitindo que o sistema utilize todas as funcionalidades nativas sem dependências proprietárias. Alternativas como *Neon* ou *ElephantSQL* foram avaliadas, mas o *Supabase* ofereceu o melhor conjunto de recursos no plano gratuito.

3.3 Segurança

A segurança da informação é um aspecto crítico no desenvolvimento de sistemas de software, especialmente em aplicações que manipulam dados financeiros, considerados entre os ativos mais sensíveis das organizações. Esse domínio se apoia em três princípios fundamentais — confidencialidade, integridade e disponibilidade — que orientam a definição dos mecanismos de proteção a serem adotados em cada camada do sistema (SOMMERVILLE, 2021). A confidencialidade assegura que as informações sejam acessadas apenas por usuários autorizados; a integridade garante que os dados não sejam alterados de forma indevida; e a disponibilidade refere-se à capacidade de o sistema continuar operacional e acessível quando necessário (PRESSMAN; MAXIM, 2021).

No sistema desenvolvido, foram adotadas técnicas consolidadas para atender a esses princípios. A autenticação dos usuários é realizada por meio do padrão *JSON Web Token* (JWT), responsável por gerenciar sessões de forma segura e *stateless*. O armazenamento seguro das senhas é garantido pela combinação da

biblioteca *Passlib* com o algoritmo *bcrypt*, que impede a recuperação de credenciais em caso de vazamento de dados. Adicionalmente, o sistema oferece autenticação social baseada no protocolo *OAuth* 2.0, permitindo que o usuário acesse a aplicação utilizando uma conta do Google, sem a necessidade de criar uma senha exclusiva para o sistema. As próximas subseções detalham cada uma dessas tecnologias.

3.3.1 Autenticação por *JSON Web Token* (JWT)

O *JSON Web Token* (JWT) é um padrão aberto, definido pela RFC 7519, que descreve uma forma compacta e autocontida para a transmissão segura de informações entre partes, na forma de objetos *JSON* assinados digitalmente (JONES; BRADLEY; SAKIMURA, 2015). Os *tokens* podem ser assinados por meio de algoritmos baseados em chave secreta, como o HMAC, ou por meio de pares de chaves pública e privada, como RSA ou ECDSA, oferecendo flexibilidade quanto ao nível de segurança e ao modelo de confiança adotado (JONES; BRADLEY; SAKIMURA, 2015).

A autenticação baseada em *tokens* é caracterizada por ser *stateless*, ou seja, o servidor não precisa manter informações de sessão para cada usuário autenticado, o que favorece a escalabilidade horizontal das aplicações *web* modernas (SOMMERVILLE, 2021). Uma prática amplamente adotada nesse modelo é a utilização de dois *tokens* complementares: um *access token*, de curta duração, responsável por autorizar as requisições; e um *refresh token*, de duração estendida, utilizado para obter novos *tokens* de acesso sem que o usuário precise realizar autenticação novamente, equilibrando segurança e experiência de uso (HARDT, 2012).

O JWT foi escolhido em vez do modelo tradicional de sessões com *cookies* por ser *stateless* o servidor não precisa armazenar informações da sessão, o que favorece a escalabilidade horizontal e simplifica o *deploy*. A estratégia de combinar *access token* de curta duração com *refresh token* de longa duração equilibra segurança e experiência: caso o *access token* seja comprometido, sua validade limitada reduz o impacto; ao mesmo tempo, o usuário não precisa fazer login frequentemente, pois o *refresh token* permite renovações silenciosas. Esse padrão é amplamente adotado em APIs modernas.

3.3.2 Proteção de senhas com *bcrypt*

O *bcrypt* é uma função de derivação de chave projetada por Provos e Mazières (1999), baseada na cifra *Blowfish*, voltada ao armazenamento seguro de senhas em sistemas computacionais. A função aplica um *salt* aleatório em cada operação, evitando ataques baseados em tabelas pré-computadas (*rainbow tables*), e introduz um fator de custo configurável que torna o cálculo do *hash* progressivamente mais demorado, mantendo o algoritmo resistente mesmo diante da evolução do poder computacional disponível para ataques (PROVOS; MAZIÈRES, 1999).

Para integrar essa função à aplicação *Python*, foi utilizada a biblioteca *Passlib*, que oferece uma interface unificada para diferentes algoritmos de *hash* de senhas, incluindo o *bcrypt* (PASSLIB, 2024). A biblioteca abstrai detalhes de implementação, como a geração do *salt* e a aplicação do fator de custo, e fornece métodos de alto nível para a criação e a verificação dos *hashes*, simplificando o uso correto da função criptográfica e reduzindo o risco de erros de implementação relacionados à segurança (PASSLIB, 2024).

Em sistemas que armazenam credenciais de usuários, é considerada má prática a persistência de senhas em texto puro, devendo-se armazenar apenas o *hash* gerado por uma função criptográfica adequada (SOMMERVILLE, 2021). A verificação da autenticidade no momento do *login* ocorre por meio da comparação entre o *hash* armazenado e o resultado do mesmo processamento aplicado à senha informada, sem que seja necessário, em momento algum, recuperar o valor original da credencial, fortalecendo a confidencialidade das informações mesmo em cenários de vazamento de dados (PRESSMAN; MAXIM, 2021).

O *bcrypt* foi escolhido em detrimento de funções criptográficas mais rápidas como SHA-256 ou MD5 justamente por sua lentidão proposital. Funções rápidas são vulneráveis a ataques de força bruta acelerados por GPU, capazes de testar bilhões de *hashes* por segundo. O *bcrypt*, por sua característica adaptativa com fator de custo configurável, torna esse tipo de ataque inviável computacionalmente. Adicionalmente, o *salt* embutido em cada hash impede ataques baseados em tabelas pré-computadas *(rainbow tables)*. Essas características fazem do *bcrypt* a recomendação oficial da *OWASP* para o armazenamento seguro de senhas.

3.3.3 Autenticação social com *OAuth* 2.0

O *OAuth* 2.0 é um protocolo padronizado de autorização que permite a um aplicativo obter acesso limitado a recursos de um usuário em um serviço de terceiros, sem a necessidade de compartilhar suas credenciais (HARDT, 2012). No contexto de autenticação social, esse protocolo possibilita que os usuários acessem aplicações utilizando contas previamente cadastradas em provedores externos, como Google, Facebook ou Microsoft, sem a necessidade de criar e gerenciar uma senha específica para cada serviço (HARDT, 2012).

A adoção de autenticação social contribui para a melhoria da experiência do usuário, reduzindo o atrito no processo de cadastro e diminuindo a quantidade de credenciais que cada indivíduo precisa memorizar (SOMMERVILLE, 2021). Do ponto de vista da segurança, esse modelo delega o gerenciamento de credenciais a provedores especializados, que implementam mecanismos robustos de proteção e detecção de fraudes, transferindo parte da complexidade de segurança para infraestruturas dedicadas a esse fim (PRESSMAN; MAXIM, 2021).

A autenticação social via *OAuth* 2.0 com Google foi adotada por dois motivos. Do ponto de vista da experiência do usuário, elimina o atrito de criar e memorizar uma nova senha, aumentando a taxa de conversão no cadastro. Do ponto de vista da segurança, delega o gerenciamento de credenciais a um provedor especializado, que aplica autenticação multifator e mecanismos avançados de detecção de fraude. Essa estratégia segue o princípio de não reinventar mecanismos críticos de segurança quando provedores estabelecidos já oferecem soluções robustas.

3.4 Banco de dados

Um banco de dados é uma coleção organizada de informações armazenadas eletronicamente, que permite o gerenciamento, o acesso e a atualização de dados de forma eficiente, garantindo a integridade, a disponibilidade e a segurança das informações armazenadas (SOMMERVILLE, 2021). O Sistema de Gerenciamento de Banco de Dados (SGBD) é o conjunto de programas responsável pelo controle e pela administração desse repositório, oferecendo funcionalidades como controle de transações, definição de esquemas, gerenciamento de usuários, controle de concorrência e mecanismos de recuperação em caso de falhas (PRESSMAN; MAXIM, 2021).

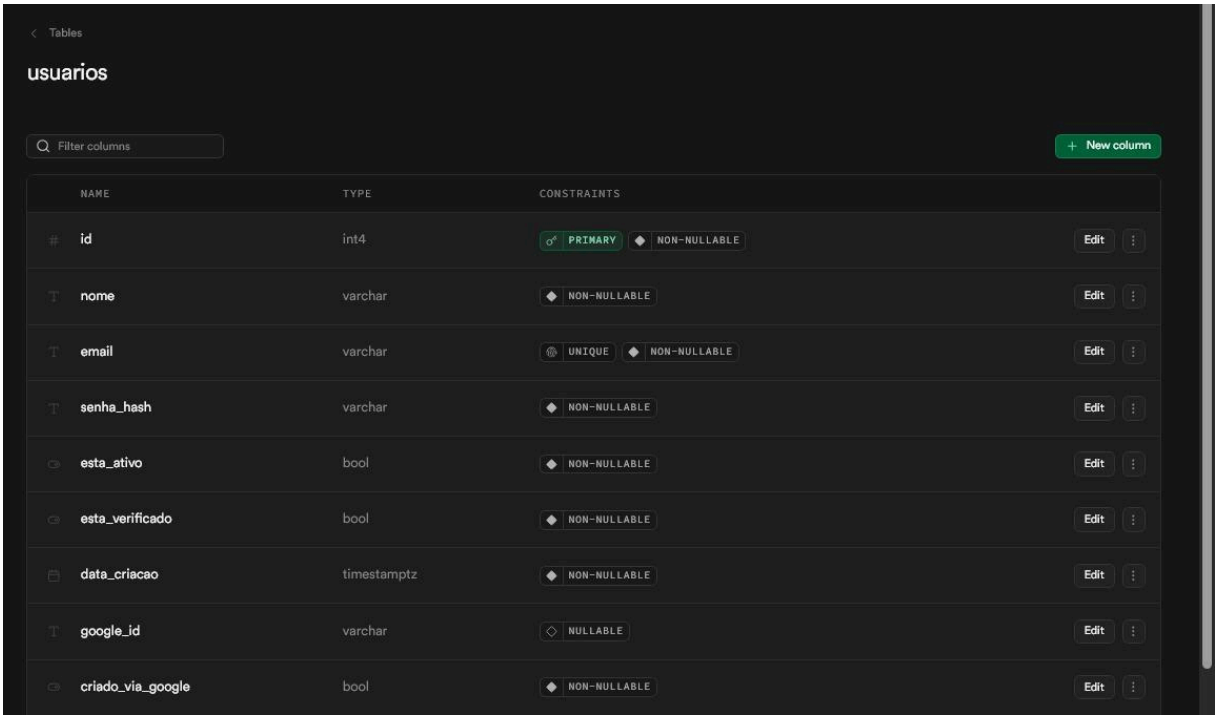
Os bancos de dados relacionais, em particular, organizam as informações em tabelas estruturadas, compostas por linhas e colunas, e estabelecem relacionamentos entre essas tabelas por meio de chaves primárias e estrangeiras. Esse modelo, baseado nos fundamentos teóricos propostos por Codd (1970), continua sendo amplamente adotado no mercado por sua maturidade, confiabilidade e suporte a transações que seguem as propriedades ACID — *atomicity*, *consistency*, *isolation* e *durability* — que garantem a integridade dos dados em ambientes multiusuário e tolerantes a falhas (HARDER; REUTER, 1983).

3.4.1 *PostgreSQL*

O *PostgreSQL* é um sistema de gerenciamento de banco de dados objeto-relacional de código aberto, conhecido por sua robustez, extensibilidade, conformidade com padrões SQL e suporte a transações complexas (POSTGRESQL, 2025). Originário do projeto POSTGRES da Universidade da Califórnia em Berkeley, iniciado em 1986, o *PostgreSQL* é mantido atualmente pelo *PostgreSQL Global Development Group* e conta com décadas de desenvolvimento ativo da comunidade de software livre (POSTGRESQL, 2025).

Entre suas principais características destacam-se a conformidade com as propriedades ACID, fundamentais para garantir a confiabilidade das operações em ambientes que manipulam dados sensíveis; o suporte a recursos avançados como índices, *views*, *triggers*, *stored procedures* e linguagens procedurais; e a possibilidade de criação de tipos de dados personalizados, o que confere ao sistema grande flexibilidade na modelagem de domínios complexos (POSTGRESQL, 2025). Em sistemas que manipulam informações financeiras, características como integridade transacional, capacidade de auditoria e desempenho em consultas analíticas tornam o *PostgreSQL* uma escolha amplamente adotada no mercado (SOMMERVILLE, 2021).

Figura 16 – Estrutura da tabela de usuários no PostgreSQL



Fonte: Elaborado pelo autor (2026).

A Figura 16 apresenta a estrutura da tabela de usuários no banco de dados *PostgreSQL*, exibindo os principais campos utilizados para o armazenamento das informações dos usuários do sistema. Entre os atributos cadastrados, destacam-se identificador (*id*), nome, e-mail, senha criptografada (*senha_hash*), status de atividade, verificação da conta, data de criação e informações relacionadas à autenticação por conta Google. Essa estrutura permite o gerenciamento seguro e organizado dos dados dos usuários, contribuindo para a integridade e controle de acesso ao sistema.

O *PostgreSQL* foi escolhido entre os principais bancos de dados do mercado por sua adequação ao domínio financeiro. Bancos *NoSQL* como *MongoDB* e *Firebase* oferecem consistência eventual, inadequada para operações financeiras que exigem atomicidade absoluta. Entre os relacionais, o *PostgreSQL* supera o *MySQL* em conformidade ao padrão SQL, em suporte a tipos de dados avançados e em integridade transacional em cenários de alta concorrência. Suas propriedades ACID garantem que operações compostas ocorram de forma atômica, evitando inconsistências.

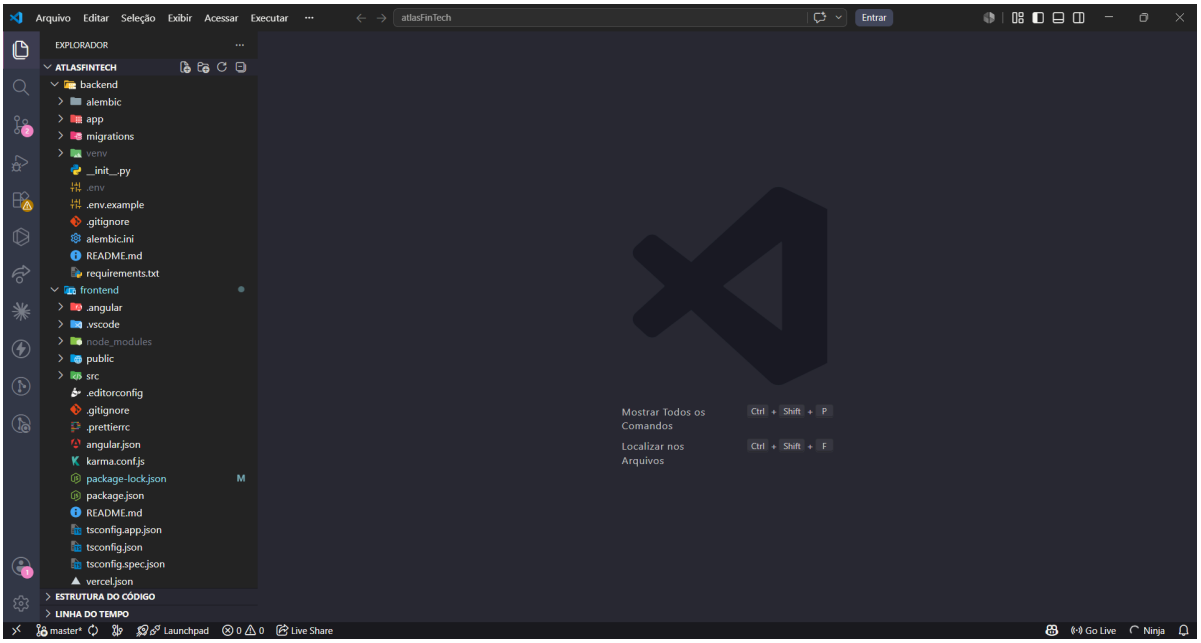
3.5 Ferramentas de gestão e colaboração

Para o desenvolvimento deste trabalho, foram utilizadas ferramentas essenciais que auxiliaram na construção, organização e gestão do projeto. O *Visual Studio Code* foi adotado como ambiente de desenvolvimento integrado para edição e manutenção do código. O *GitHub* permitiu o controle de versões e a colaboração entre os integrantes da equipe durante o desenvolvimento. O *Jira* foi utilizado para o gerenciamento de tarefas e o acompanhamento do progresso do projeto. Essas ferramentas, em conjunto, contribuíram para a organização, eficiência e qualidade do desenvolvimento da aplicação.

3.5.1 Visual Studio Code

O *Visual Studio Code* (*VS Code*) é um editor de código-fonte gratuito, leve e extensível, desenvolvido pela Microsoft é compatível com diversas linguagens de programação (MICROSOFT, 2024). Sua principal característica é a possibilidade de extensão por meio de um vasto ecossistema de complementos, que oferecem suporte a recursos como depuração, análise estática de código (*linting*), testes automatizados, integração com sistemas de controle de versão e formatação automática (MICROSOFT, 2024). O uso de editores e ambientes de desenvolvimento integrados é considerado uma boa prática na engenharia de software, pois aumenta a produtividade da equipe, reduz a ocorrência de erros e padroniza o ambiente de trabalho dos desenvolvedores (SOMMERVILLE, 2021). A Figura 17 ilustra o ambiente de desenvolvimento utilizado no projeto.

Figura 17 – Ambiente de desenvolvimento no Visual Studio Code



Fonte: Elaborado pelo autor (2026).

A Figura 17 apresenta a interface do *Visual Studio Code* utilizada no desenvolvimento do sistema, exibindo a organização das pastas e arquivos do projeto, incluindo os módulos de *backend* e *frontend*. O editor foi utilizado para a implementação e o gerenciamento do código-fonte, contribuindo para maior organização e eficiência no desenvolvimento da aplicação.

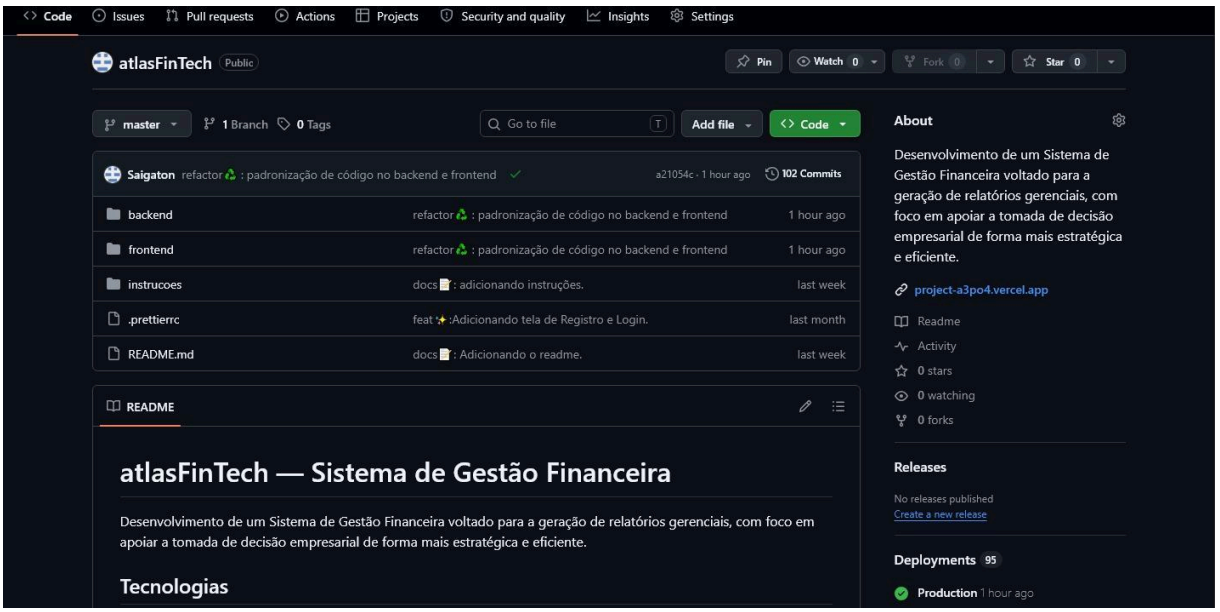
O *Visual Studio Code* foi adotado em vez de IDEs específicas como *PyCharm* ou *WebStorm* por ser gratuito, leve e multilinguagem. Em um projeto *fullstack* que envolve *Python* no *back-end* e *TypeScript* no *front-end*, manter um único editor para ambas as *stacks* reduz o tempo de troca de contexto e padroniza a experiência de desenvolvimento. Sua extensibilidade permitiu instalar plugins específicos para *Python*, *Angular*, *ESLint* e *Git*, criando um ambiente personalizado equivalente ao de IDEs dedicadas, mas sem o consumo elevado de memória que essas costumam apresentar.

3.5.2 GitHub

O *GitHub* é uma plataforma de hospedagem de código baseada no sistema de controle de versão *Git*, amplamente utilizada para o armazenamento, o versionamento e a colaboração no desenvolvimento de software (GITHUB, 2024). A

plataforma oferece recursos como repositórios públicos e privados, controle de alterações por meio de *commits*, fluxos de revisão de código por *pull requests*, gerenciamento de *issues* e integração com ferramentas de integração contínua (GITHUB, 2024). O controle de versão é considerado um pilar fundamental da engenharia de software moderna, pois permite o rastreamento de alterações, a recuperação de versões anteriores e o trabalho colaborativo de equipes distribuídas, contribuindo para a integridade e a manutenibilidade do código (SOMMERVILLE, 2021). A Figura 18 apresenta o repositório do projeto na plataforma.

Figura 18 – Repositório do projeto no GitHub



Fonte: Elaborado pelo autor (2026).

A Figura 18 apresenta o repositório do projeto no *GitHub*, exibindo a estrutura dos arquivos, os diretórios e o histórico de *commits* utilizados no desenvolvimento do sistema. A plataforma foi utilizada para o controle de versão e o gerenciamento colaborativo do código-fonte, contribuindo para maior organização e rastreabilidade das alterações realizadas no projeto.

O *GitHub* foi escolhido em detrimento do *GitLab* ou *Bitbucket* por sua adoção massiva pela comunidade de desenvolvimento e pela integração direta com a *Vercel* e o *Render*. Essa integração permitiu configurar deploys automáticos a cada *push*,

sem necessidade de pipelines manuais. Adicionalmente, o *GitHub* oferece recursos como *pull requests, code review, issues* e *Actions* no plano gratuito.

3.5.3 Jira

O *Jira* é uma ferramenta de gerenciamento de projetos desenvolvida pela Atlassian, voltada ao planejamento, ao acompanhamento e à organização de tarefas em equipes de desenvolvimento de software (ATLASSIAN, 2024). A ferramenta oferece recursos como criação de quadros visuais no estilo *Kanban*, controle de prazos, classificação de tarefas por prioridade e geração de relatórios sobre o andamento das atividades (ATLASSIAN, 2024). O uso de ferramentas de gerenciamento de tarefas é uma prática essencial em projetos de software, pois permite a organização das atividades, o acompanhamento do progresso e a comunicação eficiente entre os membros da equipe (PRESSMAN; MAXIM, 2021). A Figura 19 ilustra o quadro *Kanban* utilizado no gerenciamento do projeto.

Figura 19 – Quadro Kanban do projeto no Jira

Fonte: Elaborado pelo autor (2026).

A Figura 19 apresenta o quadro Kanban utilizado no gerenciamento das atividades do projeto, permitindo a organização e o acompanhamento das tarefas por meio das etapas do fluxo de trabalho, desde o backlog até a conclusão.

O *Jira* foi escolhido como ferramenta de gestão em vez de alternativas mais simples como *Trello*, *Notion* ou *GitHub Projects* por ser a referência de mercado em ambientes corporativos. Sua adoção neste projeto também teve um objetivo de aprendizado: ganhar familiaridade com uma ferramenta amplamente utilizada no mercado profissional. Seu suporte nativo a quadros *Kanban*, com possibilidade de customizar colunas, etiquetas e fluxos, permitiu organizar o desenvolvimento exatamente conforme a metodologia ágil adotada no projeto.

4 MODELAGEM DO SISTEMA

Este capítulo apresenta a modelagem e a análise do Atlas FinTech, descrevendo como o sistema foi estruturado para atender aos requisitos identificados junto aos usuários. A modelagem de sistemas é uma etapa fundamental do processo de desenvolvimento de software, pois permite representar o comportamento, a estrutura e a arquitetura da solução de forma organizada e padronizada, servindo como base para a implementação e para a manutenção futura.

Para isso, o capítulo parte do levantamento de requisitos funcionais e não funcionais, que definem o que o sistema deve fazer e como deve se comportar. Em seguida, são apresentados o diagrama de caso de uso, que descreve os fluxos de interação entre os atores e o sistema, e o diagrama de classes, que representa a estrutura estática das entidades do domínio. O dicionário de dados documenta de forma padronizada os elementos manipulados pelo sistema, enquanto a seção de arquitetura detalha como as camadas da solução se organizam e se comunicam. Por fim, a implementação do sistema apresenta as telas desenvolvidas, demonstrando a materialização de todos os elementos modelados nas etapas anteriores.

4.1 Levantamento de requisitos

O levantamento de requisitos é uma das etapas mais críticas do processo de desenvolvimento de software, pois tem como objetivo identificar, compreender e registrar as necessidades dos usuários e do sistema. Essa etapa permite definir de forma clara as funcionalidades e restrições que o sistema deve atender, servindo como base fundamental para todas as fases subsequentes do projeto (SOMMERVILLE, 2021).

Pressman e Maxim (2021) destacam que o levantamento de requisitos envolve atividades colaborativas entre engenheiros de software e partes interessadas, como entrevistas, questionários e análise de documentos, sendo que requisitos mal definidos ou incompletos constituem a principal causa de falhas e retrabalho nos projetos de software. Sommerville (2021) complementa ao classificar os requisitos em funcionais, que descrevem o que o sistema deve fazer, e não funcionais, que definem restrições de desempenho, segurança e usabilidade,

distinção essencial para organizar o escopo do projeto e garantir que todas as necessidades sejam devidamente documentadas.

Larman (2007) reforça que o levantamento de requisitos é um processo iterativo e contínuo, no qual os requisitos tendem a evoluir à medida que os usuários aprofundam sua compreensão sobre o sistema, tornando indispensável a adoção de práticas que permitam revisar e refinar as definições de forma controlada ao longo do desenvolvimento.

4.1.1 Requisitos funcionais

Os requisitos funcionais descrevem as funcionalidades, serviços e comportamentos que o sistema deve executar a fim de atender às necessidades dos usuários e das operações do negócio. Eles definem ações específicas, entradas, saídas e interações esperadas, servindo como base para todas as etapas de desenvolvimento, desde o projeto até a validação (SOMMERVILLE, 2021). Pressman e Maxim (2021) reforçam que os requisitos funcionais devem ser descritos de forma precisa e verificável, de modo que cada funcionalidade possa ser testada e validada individualmente, garantindo que o sistema execute exatamente aquilo que é necessário para solucionar o problema proposto e apoiar os processos organizacionais.

A Tabela 2 apresenta os requisitos funcionais do Atlas FinTech, especificando as funcionalidades que o sistema deve oferecer para atender às necessidades operacionais e de negócio identificadas durante o levantamento de requisitos.

Tabela 2 – Requisitos funcionais

Código	Requisito	Descrição
RF1	Cadastrar usuário	O sistema deverá permitir o cadastro de um novo usuário com nome, e-mail e senha
RF2	Cadastrar empresa	O sistema deverá permitir o cadastro da empresa do usuário autenticado após a verificação do e-mail
RF3	Verificar e-mail do usuário	O sistema deverá permitir a verificação do e-mail do

		usuário por meio de um link enviado após o cadastro
RF4	Autenticar usuário	O sistema deverá autenticar emails e senhas válidas
RF5	Autenticar usuário via Google	O sistema deverá permitir a autenticação do usuário através de uma conta Google
RF6	Trocar senha	O sistema deverá permitir que o usuário autenticado altere sua senha atual por uma nova senha
RF7	Recuperar senha	O sistema deverá permitir a recuperação de senha do usuário por meio de um link de redefinição enviado ao e-mail cadastrado
RF8	Gerar dashboard financeiro	O sistema deverá gerar um dashboard com KPIs, transações recentes, gráficos mensais, gráficos por conta
RF9	Cadastrar contas bancárias	O sistema deverá permitir o cadastro de contas bancárias da empresa do usuário
RF10	Transferir entre contas bancárias	O sistema deverá permitir a transferência de valores entre contas bancárias cadastradas da empresa do usuário
RF11	Visualizar contas bancárias	O sistema deverá permitir a visualização das contas bancárias cadastradas da empresa do usuário
RF12	Cadastrar transações bancárias	O sistema deverá permitir o cadastro de transações bancárias da empresa do usuário
RF13	Visualizar transações bancárias	O sistema deverá permitir a visualização de transações bancárias da empresa do usuário
RF14	Gerar transações recorrentes	O sistema deverá permitir a geração automática de transações recorrentes a partir de uma transação base, conforme a periodicidade definida pelo usuário
RF15	Cadastrar contas a pagar	O sistema deverá permitir o

		cadastro de contas a pagar da empresa do usuário
RF16	Visualizar contas a pagar	O sistema deverá permitir a visualização de contas a pagar da empresa do usuário
RF17	Cadastrar contas a receber	O sistema deverá permitir o cadastro de contas a receber da empresa do usuário
RF18	Visualizar contas a receber	O sistema deverá permitir a visualização de contas a receber da empresa do usuário
RF19	Realizar pagamento de conta	O sistema deverá permitir o registro do pagamento de uma conta a pagar da empresa do usuário, debitando o valor da conta bancária correspondente
RF20	Realizar recebimento de conta	O sistema deverá permitir o registro do recebimento de uma conta a receber da empresa do usuário, creditando o valor na conta bancária correspondente
RF21	Gerar fluxo de caixa	O sistema deverá gerar o fluxo de caixa com base nas transações financeiras da empresa em um período de tempo definido pelo usuário
RF22	Gerar análises financeiras	O sistema deverá gerar análises financeiras com base nas transações financeiras da empresa em um período de tempo definido pelo usuário
RF23	Realizar conciliação bancária	O sistema deverá permitir a conciliação de transações bancárias da empresa do usuário a partir da importação de extratos em arquivo
RF24	Gerenciar perfil do usuário	O sistema deverá permitir a visualização e a edição dos dados cadastrados do usuário autenticado
RF25	Gerenciar categorias	O sistema deverá permitir o cadastro, visualização, edição e exclusão de categorias da empresa do usuário
RF26	Interagir com assistente	O sistema deverá permitir a interação do usuário com um

		assistente para consulta de informações financeiras da empresa
RF27	Agendar envio de relatórios	O sistema deverá permitir o agendamento do envio recorrente de relatórios financeiros da empresa do usuário por e-mail
RF28	Emitir relatórios financeiros	O sistema deverá permitir a emissão de relatórios financeiros da empresa do usuário em formatos CSV, TXT e ZIP
RF29	Encerrar sessão	O sistema deverá permitir que o usuário encerre sua sessão no sistema

Fonte: Elaborado pelo autor (2026).

4.1.2 Requisitos não funcionais

Os requisitos não funcionais estabelecem qualidades, restrições e padrões de desempenho que o sistema deve atender, incluindo segurança, usabilidade, eficiência, confiabilidade, escalabilidade e manutenção. Segundo Chung, Nixon e Yu (2022), esses requisitos definem como o sistema deve operar e influenciam diretamente a experiência do usuário, a robustez da solução e as decisões arquitetônicas do software. Sommerville (2021) complementa ao destacar que os requisitos não funcionais frequentemente impõem restrições mais críticas do que os funcionais, pois o não atendimento de um requisito de desempenho ou segurança pode inviabilizar o sistema como um todo, mesmo que todas as funcionalidades estejam corretamente implementadas.

A Tabela 3 apresenta os requisitos não funcionais do Atlas FinTech, especificando os critérios de segurança, compatibilidade, usabilidade e disponibilidade que o sistema deve atender para operar de forma confiável em ambiente de produção.

Tabela 3 – Requisitos não funcionais

Código	Requisito	Descrição
RNF1	Armazenar senhas dos usuários	O sistema deverá armazenar as senhas dos usuários utilizando o algoritmo bcrypt, com salt único por senha e fator de custo configurável
RNF2	Autenticar usuário no sistema	O sistema deverá gerar e autenticar os usuários com tokens JWT
RNF3	Compatibilidade com navegadores	O sistema deverá ser compatível com as últimas duas versões dos navegadores Google Chrome, Mozilla Firefox, Microsoft Edge e Safari
RNF4	Usabilidade do sistema	O sistema deverá apresentar interface intuitiva, permitindo que um usuário sem conhecimento técnico prévio realize as operações básicas de cadastro e consulta após um único contato com o sistema
RNF5	Responsividade	O sistema deverá ser responsivo, adaptando-se a diferentes tamanhos de tela (desktop, tablet e dispositivos móveis)
RNF6	Manutenibilidade	O sistema deverá ser estruturado em arquitetura em camadas, favorecendo a manutenção, evolução e testabilidade do código
RNF7	Documentação da API	O sistema deverá disponibilizar documentação técnica das APIs no padrão OpenAPI/Swagger

Fonte: Elaborado pelo autor (2026).

4.2 Diagrama de caso de uso

O diagrama de caso de uso é um dos principais artefatos da UML (*Unified Modeling Language*), utilizado para representar, de maneira simples e em alto nível, como os usuários externos interagem com o sistema. Ele descreve os requisitos funcionais do software, evidenciando as funcionalidades disponíveis e os atores que

participam de cada uma delas, contribuindo para a compreensão do escopo do sistema e para a comunicação entre analistas e *stakeholders* (SOMMERVILLE, 2021). Além disso, esse tipo de diagrama fornece uma visão clara do comportamento esperado do sistema, servindo como base para o detalhamento posterior em outros modelos e documentos de requisitos (PRESSMAN; MAXIM, 2021).

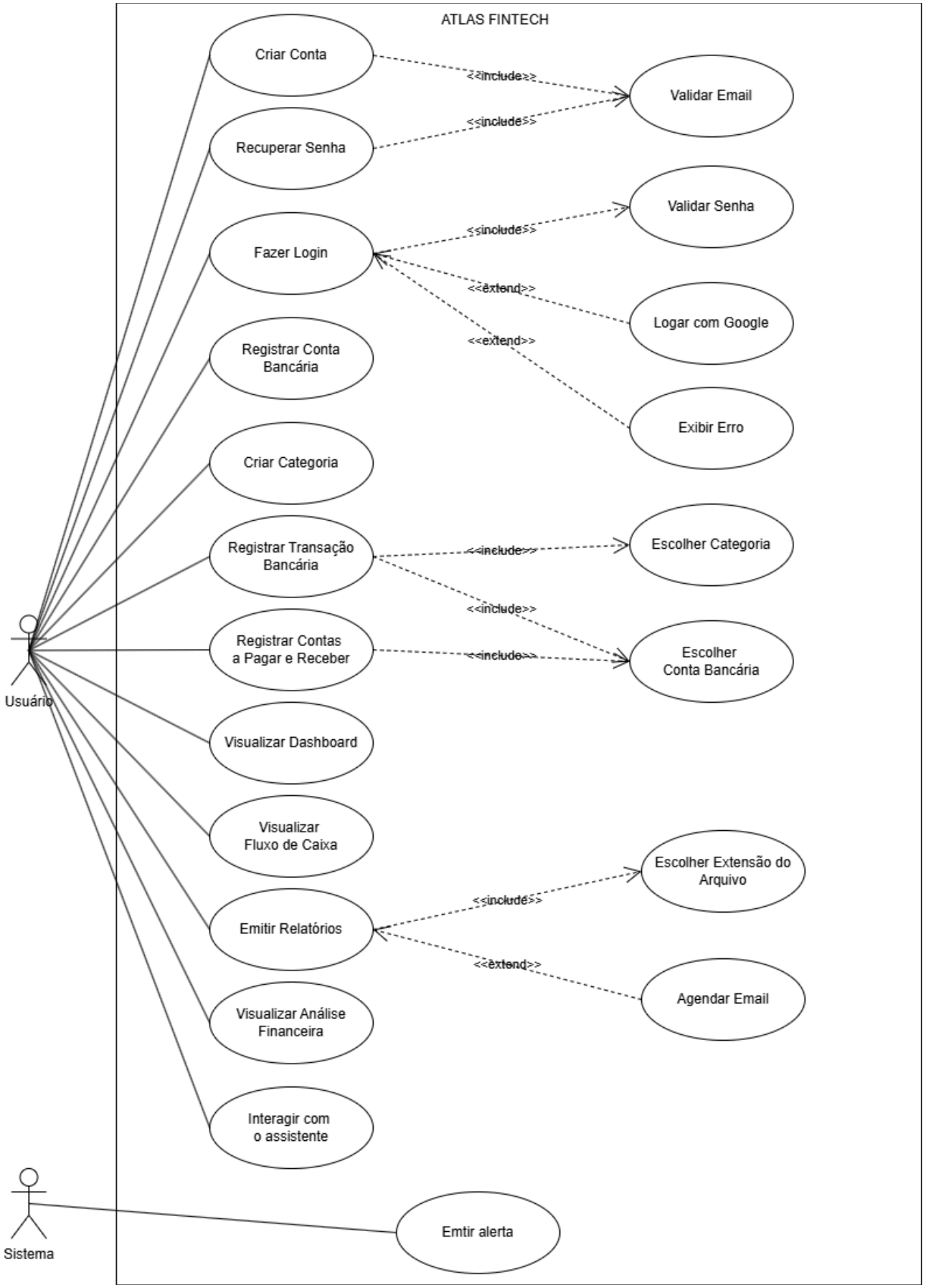
De acordo com Booch, Rumbaugh e Jacobson (2006), criadores da UML, os diagramas de caso de uso surgiram da necessidade de representar o comportamento de um sistema a partir da perspectiva do usuário, capturando o que o sistema deve fazer sem detalhar como o fará. Essa separação entre requisitos funcionais e decisões de implementação é considerada fundamental para a engenharia de software orientada a objetos, pois permite que os requisitos sejam validados pelos próprios usuários antes que qualquer código seja produzido.

Larman (2007) complementa essa perspectiva ao afirmar que os casos de uso constituem uma técnica de descoberta de requisitos amplamente adotada, por serem compreensíveis tanto para desenvolvedores quanto para clientes não técnicos. Segundo o autor, a narrativa associada a cada caso de uso descreve um contrato de comportamento do sistema, estabelecendo o que acontece quando um ator interage com ele em um determinado contexto. Essa característica torna os diagramas de caso de uso especialmente úteis nas etapas iniciais de levantamento e validação de requisitos.

De acordo com a especificação oficial da UML, os diagramas de caso de uso auxiliam na identificação das fronteiras do sistema e na organização dos casos de uso principais, sendo fundamentais nas etapas iniciais de análise (OMG, 2017). Fowler (2004) reforça essa visão ao destacar que, apesar de sua simplicidade aparente, os diagramas de caso de uso são poderosos instrumentos de comunicação entre equipes de desenvolvimento e áreas de negócio, pois sintetizam em um único modelo visual quem usa o sistema e para quê.

O diagrama apresentado na Figura 20 foi elaborado pelos autores e representa as interações entre os atores e o sistema, evidenciando as funcionalidades disponíveis e as ações associadas a cada caso de uso do Atlas FinTech.

Figura 20 – Diagrama de caso de uso



Fonte: Elaborado pelo autor (2026).

O diagrama de casos de uso do Atlas FinTech descreve as principais funcionalidades da aplicação e as interações dos atores com o sistema. Foram identificados dois atores: o Usuário e o Sistema, cada um desempenhando papéis específicos.

O Usuário interage com os casos de uso de criar conta, recuperar senha, fazer login, registrar conta bancária, criar categoria, registrar transação bancária, registrar contas a pagar e receber, visualizar dashboard, visualizar fluxo de caixa, emitir relatórios, visualizar análise financeira e interagir com o assistente. O Sistema, por sua vez, é responsável pela emissão automática de alertas.

O diagrama também evidencia as relações entre os casos de uso. O estereótipo *«include»* representa funcionalidades obrigatórias incorporadas a outras ações, como a validação de e-mail ao criar conta, a validação de senha ao fazer login, a escolha de categoria ao registrar transação bancária, a escolha de conta bancária ao registrar contas a pagar e receber, e a escolha da extensão do arquivo ao emitir relatórios. Já o estereótipo *«extend»* indica comportamentos opcionais ou condicionais, como o login via Google e a exibição de erro durante a autenticação, além do agendamento de e-mail na emissão de relatórios.

A escolha entre os estereótipos *«include»* e *«extend»* seguiu critérios funcionais e de segurança do sistema. O *«include»* foi aplicado nos casos em que a funcionalidade incluída é obrigatória para que o caso de uso principal seja concluído com sucesso, como a validação de e-mail ao criar conta, que não pode ser ignorada, pois garante a autenticidade do usuário antes de liberar o acesso ao sistema. Já o *«extend»* foi utilizado para comportamentos opcionais ou condicionais, que ocorrem apenas em circunstâncias específicas, como o login via Google, que representa uma alternativa ao fluxo padrão de autenticação, e a exibição de erro, que só ocorre quando as credenciais informadas são inválidas. Essa distinção reflete uma decisão deliberada de design, separando o que é essencial ao funcionamento do sistema daquilo que representa uma variação ou extensão do comportamento esperado.

4.2.1 Criar conta

Este caso de uso representa como o usuário pode criar sua conta no sistema e como a aplicação pode se comportar, conforme a Tabela 4:

Tabela 4 – Especificação caso de uso - Criar conta

Ator Principal	Usuário
Atores Secundários	Serviço de e-mail
Resumo	Viabiliza o cadastro de usuário no sistema
Pré Condições	Não ter cadastro no sistema
Pós Condições	Usuário cadastrado e com acesso ao sistema
Fluxo Principal	
1. O usuário seleciona a opção “Criar Conta” 2. O sistema exibe o formulário de cadastro com os campos "nome", "e-mail", "senha" e "confirmar senha" 3. O usuário preenche os campos e confirma o cadastro 4. O sistema valida os dados, registra o usuário e envia um link de verificação para o e-mail informado 5. O usuário acessa o link de verificação recebido 6. O sistema confirma a verificação e redireciona o usuário para a tela de login 7. O usuário realiza o login no sistema 8. O sistema identifica que o usuário ainda não possui empresa cadastrada e exibe o formulário de cadastro de empresa, com os campos "nome" e "documento" (CPF ou CNPJ) 9. O usuário preenche os dados da empresa e confirma o cadastro 10. O sistema valida os dados, registra a empresa e libera o acesso às demais funcionalidades	
Fluxo de Exceção I - Campos do formulário inválidos	
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento então exibe um alerta com a mensagem “Campos inválidos”	
Fluxo de Exceção II - E-mail já cadastrado	
1. O usuário preenche o campo "e-mail" com um endereço já cadastrado no sistema 2. O sistema exibe a mensagem "E-mail já cadastrado" e impede a finalização do cadastro	
Fluxo de Exceção III - Senhas não coincidem	
1. O usuário preenche os campos "senha" e "confirmar senha" com valores diferentes 2. O sistema exibe a mensagem "As senhas não coincidem" e impede a finalização do cadastro	
Fluxo de Exceção IV - Documento inválido	
1. O usuário preenche o campo "documento" da empresa com um CPF ou CNPJ em formato inválido 2. O sistema exibe a mensagem "Documento inválido" e impede a finalização do cadastro	

Fonte: Elaborado pelo autor (2026).

4.2.2 Recuperar senha

Este caso de uso representa como o usuário pode recuperar a senha de acesso ao sistema e como a aplicação pode se comportar, conforme a Tabela 5:

Tabela 5 – Especificação caso de uso - Recuperar senha

Ator Principal	Usuário
Atores Secundários	Serviço de e-mail
Resumo	Permitir recuperação de senha
Pré Condições	Ter cadastro no sistema
Pós Condições	Usuário com nova senha definida e acesso ao sistema restaurado
Fluxo Principal	
1. Usuário acessa "Esqueci minha senha" 2. Sistema exibe formulário com campo "email" 3. Usuário informa o email 4. Sistema envia link de redefinição por email 5. Usuário acessa o link recebido 6. O sistema exibe o formulário para definir nova senha com os campos "nova senha" e "confirmar senha" 7. Usuário define a nova senha e confirma 8. Sistema valida e atualiza a senha 9. Sistema redireciona para tela de login	
Fluxo de Exceção I - Campos do formulário inválidos	
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento, então exibe um alerta com a mensagem "Campos inválidos"	
Fluxo de Exceção II - E-mail não cadastrado	
1. O usuário informa um e-mail que não está cadastrado no sistema 2. O sistema exibe a mensagem genérica "Se o e-mail existir, um link de recuperação será enviado", por motivos de segurança	
Fluxo de Exceção III - Link de recuperação expirado	
1. O usuário acessa o link de redefinição fora do prazo de validade 2. O sistema exibe a mensagem "Link expirado" e oferece a opção de solicitar um novo link	

Fonte: Elaborado pelo autor (2026).

4.2.3 Fazer login

Este caso de uso representa como o usuário pode fazer login no sistema e como a aplicação pode se comportar, conforme a Tabela 6:

Tabela 6 – Especificação caso de uso - Fazer login

Ator Principal	Usuário
Atores Secundários	Provedor Google
Resumo	Permitir login do usuário no sistema
Pré Condições	Ter cadastro no sistema
Pós Condições	Usuário autenticado com acesso ao sistema
Fluxo Principal	
1. O usuário acessar a página de login do sistema 2. O sistema exibe o formulário com os campo “email” e “senha”e a opção “Entrar com o Google” 3. O usuário preenche suas informações cadastrais 4. O sistema valida o dados preenchidos 5. O sistema autentica o usuário e permite o acesso ao sistema	
Fluxo Alternativo I - Login com Google	
1. O usuário acessar a página de login do sistema 2. O sistema exibe o formulário com os campo “email” e “senha”e a opção “Entrar com o Google” 3. O usuário seleciona a opção “Entrar com o Google” 4. O sistema redireciona o usuário para a interface de autenticação oficial do Google 5. O Provedor Google valida as credenciais e devolve um token de autenticação seguro para o sistema 6. O sistema recebe o token, valida a autenticidade e o usuário tem acesso ao sistema	
Fluxo de Exceção I - Campos do formulário inválidos	
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento então exibe um alerta com a mensagem “Campos inválidos”	
Fluxo de Exceção II - Dados de autenticação inválidos	
1. O usuário preenche os campos de login com dados inválidos 2. O sistema exibe a mensagem "E-mail ou senha inválidos" 3. O usuário pode fazer novas tentativas	

Fonte: Elaborado pelo autor (2026).

4.2.4 Registrar contas bancárias

Este caso de uso representa como o usuário pode registrar suas contas bancárias no sistema e como a aplicação pode se comportar, conforme a Tabela 7:

Tabela 7 – Especificação caso de uso - Registrar contas bancárias

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Permitir o registro de contas bancárias no sistema
Pré Condições	Estar autenticado no sistema
Pós Condições	Conta bancária registrada e disponível para uso em transações e contas a pagar/receber
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, “Contas Bancárias” 2. O sistema exibe as contas cadastradas anteriormente pelo usuário e a opção “Nova Conta” para o cadastro de novas contas 3. O usuário acessa a opção “Nova Conta” 4. O sistema exibe um formulário com os campos “Nome da Conta”, “Tipo”, “Saldo Inicial”, “Agência”, “Nome do banco” e “cor” 5. O usuário preenche os campos e confirma o cadastro 6. O sistema valida os campos 7. O sistema cria uma nova conta bancária e exibe na listagem	
Fluxo de Exceção I - Campos do formulário inválidos	
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento então exibe um alerta com a mensagem “Campos inválidos”	

Fonte: Elaborado pelo autor (2026).

4.2.5 Criar categoria

Este caso de uso representa como o usuário pode criar uma categoria no sistema e como a aplicação pode se comportar, conforme a Tabela 8:

Tabela 8 – Especificação caso de uso - Criar categoria

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Permitir criar categoria no sistema
Pré Condições	Estar autenticado no sistema
Pós Condições	Categoria registrada e disponível para uso em transações e contas a pagar/receber
Fluxo Principal	
1. O usuário acessa a página de ‘Categorias’	

2. O usuário pressiona o botão ‘Nova Categoria’ 3. O usuário preenche os dados ‘Nome’, ‘Tipo’ e ‘cor’ 4. O usuário preenche os campos do formulário 5. O sistema valida os dados preenchidos 6. O sistema cria a categoria e exibe na listagem
Fluxo de Exceção I - Campos do formulário inválidos
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento então exibe um alerta com a mensagem “Campos inválidos”

Fonte: Elaborado pelo autor (2026).

4.2.6 Registrar transações bancárias

Este caso de uso representa como o usuário pode registrar suas transações bancárias no sistema e como a aplicação pode se comportar, conforme a Tabela 9:

Tabela 9 – Especificação caso de uso - Registrar transações bancárias

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Permitir o registro de transações bancárias no sistema
Pré Condições	Estar autenticado no sistema, ter contas bancárias e categorias cadastradas
Pós Condições	Transação registrada e refletida no saldo da conta correspondente
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, “Transações” 2. O sistema exibe as transações cadastradas anteriormente pelo usuário e a opção “Nova Transação” para o cadastro de novas transações 3. O usuário acessa a opção “Nova Transação” 4. O sistema exibe um formulário com os campos “Conta”, “Descrição”, “Tipo”, “Valor”, “Data da transação”, “Categoria”, “Situação”, “Recorrência“ e “Notas” 5. O usuário preenche os campos e confirma o cadastro 6. O sistema valida os campos 7. O sistema cria a transação e atualiza o saldo da conta bancária correspondente	
Fluxo de Exceção I - Campos do formulário inválidos	
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento então exibe um alerta com a mensagem “Campos inválidos”	

Fonte: Elaborado pelo autor (2026).

4.2.7 Registrar contas a pagar

Este caso de uso representa como o usuário pode registrar suas contas a pagar no sistema e como a aplicação pode se comportar, conforme a Tabela 10:

Tabela 10 – Especificação caso de uso - Registrar contas a pagar

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Permitir o registro de contas a pagar no sistema
Pré Condições	Estar autenticado no sistema e ter contas bancárias e categorias cadastradas
Pós Condições	Conta a pagar registrada e visível na listagem
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, “Contas a Pagar” 2. O sistema exibe as contas a pagar cadastradas anteriormente pelo usuário e a opção "Nova Conta a Pagar" 3. O usuário acessa a opção “Nova Conta a Pagar” 4. O sistema exibe um formulário com os campos "Descrição", "Valor", "Vencimento", "Parcelamento", "Conta Bancária", "Categoria" e "Observação" 5. O usuário preenche os campos e confirmar o cadastro 6. O sistema valida os campos 7. O sistema cria a conta a pagar com situação "Pendente" e exibe na listagem	
Fluxo de Exceção I - Campos do formulário inválidos	
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento então exibe um alerta com a mensagem “Campos inválidos”	

Fonte: Elaborado pelo autor (2026).

4.2.8 Registrar contas a receber

Este caso de uso representa como o usuário pode registrar suas contas a receber no sistema e como a aplicação pode se comportar, conforme a Tabela 11:

Tabela 11 – Especificação caso de uso - Registrar contas a receber

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Permitir o registro de contas a receber no sistema
Pré Condições	Estar autenticado no sistema e ter contas

	bancárias e categorias cadastradas
Pós Condições	Conta a receber registrada e visível na listagem
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, “Contas a Receber” 2. O sistema exibe as contas a receber cadastradas anteriormente pelo usuário e a opção "Nova Conta a Receber" 3. O usuário acessa a opção “Nova Conta a Receber” 4. O sistema exibe um formulário com os campos "Descrição", "Cliente/Pagador", "Valor", "Vencimento", "Conta Bancária", "Categoria" e "Observação" 5. O usuário preenche os campos e confirmar o cadastro 6. O sistema valida os campos 7. O sistema cria a conta a receber com situação "Pendente" e exibe na listagem	
Fluxo de Exceção I - Campos do formulário inválidos	
1. O usuário preenche o campo do formulário com dados fora do padrão exigido 2. O sistema adiciona uma borda na cor vermelha sobre o campo e exibe logo abaixo a forma correta de preenchimento então exibe um alerta com a mensagem “Campos inválidos”	

Fonte: Elaborado pelo autor (2026).

4.2.9 Visualizar dashboard

Este caso de uso representa como o sistema pode gerar dashboard na aplicação e como o usuário pode se comportar, conforme a Tabela 12:

Tabela 12 – Especificação caso de uso - Visualizar dashboard

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Estar autenticado no sistema e ter movimentações bancárias cadastradas
Pré Condições	Estar autenticado no sistema e ter movimentações bancárias cadastradas pelo usuário
Pós Condições	Dashboard exibido com indicadores financeiros atualizados
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, “Dashboard” 2. O sistema gera e exibe um dashboard com os dados de "Saldo Total", "Receitas (mês)", "Despesas (mês)", "Fluxo Líquido", "Contas a Pagar" e "Contas a Receber" 3. O sistema exibe gráficos mensais e gráficos por conta bancária 4. O sistema exibe uma lista de "Transações Recentes" realizadas pelo usuário	

Fonte: Elaborado pelo autor (2026).

4.2.10 Visualizar fluxo de caixa

Este caso de uso representa como o usuário pode gerar fluxo de caixa periódicos no sistema e como a aplicação pode se comportar, conforme a Tabela 13:

Tabela 13 – Especificação caso de uso - Visualizar fluxo de caixa

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Gerar fluxo de caixa de acordo com as movimentações do usuário em um período
Pré Condições	Estar autenticado no sistema e ter movimentações bancárias cadastradas
Pós Condições	Fluxo de caixa exibido para o período informado
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, “Fluxo de Caixa” 2. O sistema exibe uma tela com o campo "Ano" 3. O usuário preenche o ano desejado 4. O sistema exibe o resultado com as informações de "Total Entradas", "Total saídas" e "Saldo Líquido", além das transações realizadas no ano 5. O sistema exibe um gráfico de “Fluxo de Caixa ao longo do tempo” com valores de “Entradas” e “Saídas” 6. O sistema exibe um gráfico de “Entradas e Saídas”	

Fonte: Elaborado pelo autor (2026).

4.2.11 Emitir relatórios

Este caso de uso representa como o sistema e o usuário podem emitir relatórios periódicos e como a aplicação pode se comportar, conforme a Tabela 14:

Tabela 14 – Especificação caso de uso - Emitir relatórios

Ator Principal	Usuário
Atores Secundários	Serviço de e-mail
Resumo	Permitir a emissão de relatórios financeiros nos formatos TXT, CSV e ZIP
Pré Condições	Estar autenticado no sistema e ter movimentações bancárias cadastradas
Pós Condições	Relatório gerado e disponibilizado para o usuário

Fluxo Principal
1. O usuário acessa a opção do sidebar esquerdo, “Relatórios” 2. O sistema exibe uma tela com os relatórios solicitados anteriormente e a opção "Novo Relatório" 3. O usuário acessa a opção “Novo Relatório” 4. O sistema exibe uma lista de tipos de relatório (PDF financeiro, CSV de transações, CSV de contas a pagar, CSV de contas a receber, ZIP de backup) 5. O usuário seleciona o tipo de relatório desejado 6. O sistema gera o relatório e disponibiliza para download
Fluxo Alternativo I - Envio por e-mail
1. O usuário, opcionalmente, marca a opção "Enviar relatório por e-mail" 2. O sistema envia o relatório gerado para o e-mail do usuário através do serviço de e-mail

Fonte: Elaborado pelo autor (2026).

4.2.12 Visualizar análise financeira

Este caso de uso representa como o usuário pode gerar análise financeira periódicos no sistema e como a aplicação pode se comportar, conforme a Tabela 15:

Tabela 15 – Especificação caso de uso - Visualizar análise financeira

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Gerar análise financeira de acordo com as movimentações do usuário em um período
Pré Condições	Estar autenticado no sistema e ter movimentações bancárias cadastradas
Pós Condições	Análise financeira exibida para o período informado
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, "Análise Financeira" 2. O sistema exibe uma tela com os filtros de período (mês e ano) 3. O usuário seleciona o período desejado 4. O sistema exibe o resultado com a comparação de "Receitas vs Despesas" e indicadores de desempenho financeiro do período	

Fonte: Elaborado pelo autor (2026).

4.2.13 Interagir com o assistente

Este caso de uso representa como o usuário pode interagir com o assistente e como a aplicação pode se comportar, conforme a Tabela 16:

Tabela 16 – Especificação caso de uso - Interagir com o assistente

Ator Principal	Usuário
Atores Secundários	Nenhum
Resumo	Permitir ao usuário interagir com o assistente integrado ao sistema para consultar informações financeiras
Pré Condições	Estar autenticado no sistema
Pós Condições	Usuário obtém informações sobre suas movimentações financeiras
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, “Assistente” 2. O sistema exibe uma tela com uma mensagem de boas-vindas e perguntas pré-definidas 3. O usuário digita uma pergunta no campo de texto e clica em enviar 4. O sistema interpreta a pergunta e exibe a resposta com as informações solicitadas	
Fluxo Alternativo I - Perguntas pré-definidas	
1. O usuário seleciona uma das perguntas pré-definidas, como "Qual o meu saldo?", "Como está o mês?", "O que tenho a pagar?", "O que tenho a receber?" ou "Há algum alerta?" 2. O sistema processa a pergunta selecionada e exibe a resposta correspondente	

Fonte: Elaborado pelo autor (2026).

4.2.14 Emitir alerta

Este caso de uso representa como o sistema pode gerar alertas e informar o usuário sobre informações relevantes sobre sua conta, conforme a Tabela 17:

Tabela 17 – Especificação caso de uso - Emitir alerta

Ator Principal	Sistema
Atores Secundários	Nenhum
Resumo	Exibir ao usuário alertas financeiros gerados automaticamente a partir da análise das suas movimentações
Pré Condições	Estar autenticado no sistema e ter movimentações bancárias cadastradas
Pós Condições	Alertas exibidos ao usuário para conhecimento dos eventos detectados
Fluxo Principal	
1. O usuário acessa a opção do sidebar esquerdo, "Análise Financeira" 2. O sistema processa as movimentações cadastradas e identifica eventos relevantes, como	

saldo bancário negativo e contas a pagar vencidas 3. O sistema exibe a lista de alertas com a descrição do evento detectado 4. O usuário visualiza os alertas exibidos
Fluxo Alternativo I - Sem alertas detectados
1. O sistema analisa as movimentações e não identifica eventos relevantes 2. O sistema exibe a mensagem "Nenhum alerta no momento"

Fonte: Elaborado pelo autor (2026).

4.3 Diagrama de classes

O diagrama de classes é um dos principais diagramas estruturais da UML (*Unified Modeling Language*), utilizado para representar a organização estática de um sistema por meio de classes, atributos, operações e relacionamentos. Esse diagrama descreve os elementos fundamentais da estrutura do software e como eles se conectam, servindo como base para o desenvolvimento orientado a objetos e para a modelagem de arquiteturas mais detalhadas (SOMMERVILLE, 2021).

Segundo Pressman e Maxim (2021), o diagrama de classes é essencial para compreender a arquitetura interna do sistema, permitindo identificar responsabilidades, dependências e hierarquias entre classes, o que facilita o processo de implementação e manutenção. A especificação oficial da UML destaca que esse diagrama captura os principais blocos estruturais do sistema, representando associações, agregações, composições e generalizações de forma padronizada, sendo amplamente utilizado nas fases de análise e projeto (OMG, 2017).

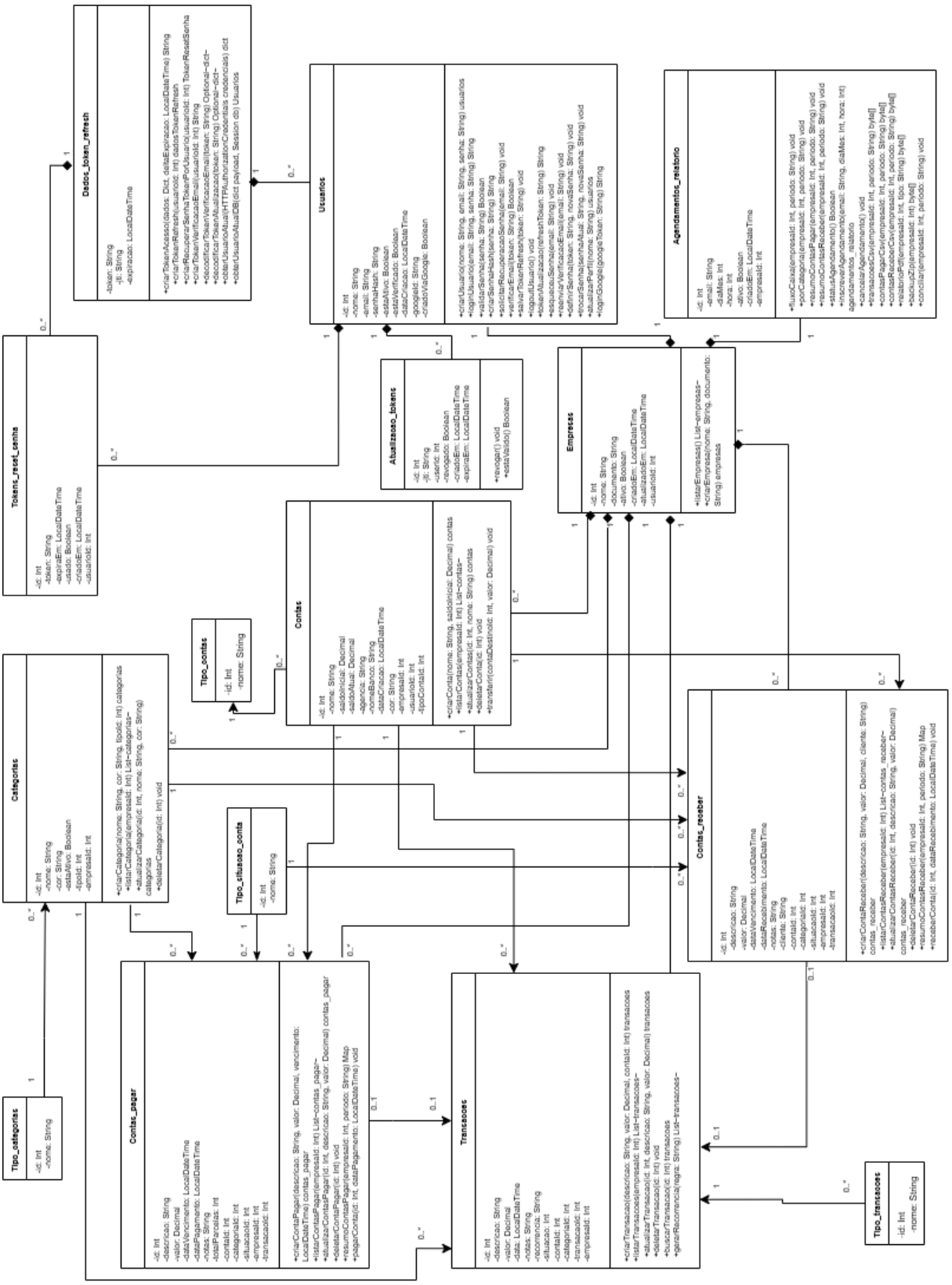
Booch, Rumbaugh e Jacobson (2006) reforçam que o diagrama de classes constitui o alicerce da modelagem orientada a objetos, pois permite visualizar não apenas a estrutura das entidades do sistema, mas também a natureza dos vínculos entre elas, sejam esses vínculos de dependência, associação ou herança. Para os autores, a correta definição dessas relações é determinante para a coesão e o baixo acoplamento do software, princípios fundamentais para sistemas de qualidade.

Larman (2007) acrescenta que o diagrama de classes pode ser elaborado em diferentes níveis de abstração ao longo do ciclo de desenvolvimento: no nível conceitual, durante a análise de domínio; no nível de especificação, ao detalhar interfaces e contratos; e no nível de implementação, quando representa diretamente

as classes do código-fonte. Essa flexibilidade torna o diagrama de classes um dos artefatos mais versáteis e amplamente utilizados na engenharia de software orientada a objetos.

A fim de detalhar a estrutura estática da solução proposta, apresenta-se a seguir a Figura 21, que ilustra o Diagrama de Classes do sistema.

Figura 21 – Diagrama de classes



Fonte: Elaborado pelo autor (2026).

O diagrama apresentado na Figura 21 foi elaborado pelos autores e representa a estrutura estática do Atlas FinTech, composta por doze entidades principais que modelam o domínio financeiro do sistema. A entidade central do modelo é Usuarios, que mantém composição direta com Empresas, indicando que cada empresa pertence exclusivamente a um usuário, e se relaciona com as entidades Tokens_reset_senha, Atualizacao_tokens e Dados_token_refresh, responsáveis pelo ciclo de vida da autenticação e pela segurança do acesso via JWT. Essa estrutura reflete a decisão arquitetural de centralizar o controle de identidade e sessão no domínio do usuário, separando as responsabilidades de autenticação das demais operações financeiras.

As entidades financeiras do sistema orbitam ao redor de Empresas, que agrega Contas, Transacoes, Contas_pagar, Contas_receber, Categorias e Agendamentos_relatorio. A entidade Transacoes possui auto-relacionamento, permitindo o registro de transferências entre contas por meio de transações espelhadas, e se associa tanto a Contas_pagar quanto a Contas_receber, viabilizando a conciliação entre lançamentos financeiros e movimentações reais de caixa. As entidades Categorias, Tipo_contas, Tipo_categorias, Tipo_situacao_conta e Tipo_transacoes atuam como tabelas de domínio, padronizando a classificação das operações financeiras e garantindo a integridade referencial do modelo. Essa organização traduz diretamente os requisitos funcionais levantados na seção 4.1, evidenciando o alinhamento entre a modelagem de domínio e as necessidades operacionais do Atlas FinTech.

4.4 Dicionário de dados

O dicionário de dados é um artefato utilizado para documentar, de forma padronizada e organizada, todos os elementos de dados manipulados por um sistema. Ele descreve campos, tipos, tamanhos, restrições, relacionamentos e regras associadas às informações, garantindo consistência e clareza no desenvolvimento e na manutenção do software. Segundo Sommerville (2021), o dicionário de dados é fundamental durante a análise de requisitos, pois permite que analistas e desenvolvedores compartilhem um entendimento comum sobre os dados utilizados no sistema.

De acordo com Pressman e Maxim (2021), o dicionário de dados funciona como um repositório central que apoia atividades de modelagem, implementação e validação, reduzindo ambiguidades e erros relacionados ao tratamento das informações. A documentação estruturada dos elementos de dados, conforme recomendado pela UML e por boas práticas de engenharia de software, auxilia na integração entre modelos, garantindo que os atributos definidos em diagramas e estruturas lógicas sejam descritos de forma precisa e padronizada (OMG, 2017).

Yourdon (1990) destaca que o dicionário de dados é um dos componentes centrais da análise estruturada, responsável por registrar formalmente cada elemento de informação do sistema, incluindo sua definição, composição e restrições. Para o autor, a ausência de um dicionário de dados bem elaborado compromete a rastreabilidade entre os modelos de análise e o produto final, gerando inconsistências difíceis de corrigir nas fases posteriores do desenvolvimento.

Larman (2007) complementa ao afirmar que o dicionário de dados está diretamente relacionado ao modelo de domínio do sistema, pois formaliza os conceitos identificados durante a análise, associando a cada entidade e atribuindo uma descrição precisa de seu propósito, formato e regras de negócio.

A Tabela 18 apresenta a estrutura da entidade Empresas, responsável por armazenar os dados cadastrais fundamentais e as configurações globais da organização.

Tabela 18 – Dicionário de Dados da Tabela Empresas

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único incremental da empresa.
nome	VARCHAR(120)	-	Não	Razão social ou nome fantasia da empresa.
documento	VARCHAR(20)	-	Sim	Documento de identificação jurídica ou fiscal (CNPJ/CPF).

ativo	BOOLEAN	-	Não	Indica se a empresa está ativa para movimentações.
criado_em	DATETIME	-	Não	Data e hora do cadastro da empresa.
atualizado_em	DATETIME	-	Não	Data e hora da última modificação dos dados cadastrais.
usuario_id	INTEGER	FK	Não	Chave estrangeira referenciando o usuário proprietário (usuarios.id).

Fonte: Elaborado pelo autor (2026).

A Tabela 19 define a entidade Atualizacao_tokens, controla o ciclo de vida e a revogação de Refresh Tokens utilizados na autenticação JWT.

Tabela 19 – Dicionário de Dados da tabela Atualizacao_tokens

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único incremental do token de atualização.
jti	VARCHAR(255)	-	Não	Identificador único global do JWT (JWT ID) para prevenção de reutilização.
user_id	INTEGER	FK	Não	Chave estrangeira mapeando o usuário associado ao token.
revogado	BOOLEAN	-	Não	Sinaliza se o token foi explicitamente invalidado/colocado em blacklist.
criado_em	DATETIME	-	Não	Data e hora exata da execução/emissão do token.

expira_em	DATETIME	-	Não	Data e hora limite de validade do token de atualização.

Fonte: Elaborado pelo autor (2026).

A Tabela 20 descreve a entidade Usuarios, que armazena as informações cadastrais e credenciais de acesso dos usuários ao sistema.

Tabela 20 – Dicionário de Dados da Tabela Usuarios

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único incremental do usuário.
nome	VARCHAR(100)	-	Não	Nome completo do usuário.
email	VARCHAR(255)	-	Não	Endereço de e-mail utilizado como login institucional.
senha_hash	VARCHAR(255)	-	Não	Hash criptográfico da senha do usuário para autenticação.
esta_ativo	BOOLEAN	-	Não	Sinaliza se o usuário está ativo no sistema.
esta_verificado	BOOLEAN	-	Não	Indica se o e-mail passou pela validação de segurança.
data_criacao	DATETIME	-	Não	Data e hora de registro do usuário.
google_id	VARCHAR(255)	-	Sim	Identificador único para autenticação via Google OAuth2.
criado_via_google	BOOLEAN	-	Não	Sinaliza se a conta foi gerada via provedor externo Google.

Fonte: Elaborado pelo autor (2026).

A Tabela 21 detalha a entidade Agendamentos_relatorio, utilizada para configurar os agendamentos periódicos de envio automatizado de relatórios financeiros por e-mail.

Tabela 21 – Dicionário de Dados da Tabela Agendamentos_relatorio

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do agendamento.
email	VARCHAR(255)	-	Não	Endereço eletrônico de destino do relatório gerado.
dia_mes	INTEGER	-	Não	Dia específico do mês configurado para o disparo automático.
hora	INTEGER	-	Não	Hora cheia do dia configurada para a execução do agendamento.
ativo	BOOLEAN	-	Não	Define se o agendamento está ativo ou temporariamente pausado.
criado_em	DATETIME	-	Não	Data e hora de criação da configuração de agendamento.
empresa_id	INTEGER	FK	Não	Chave estrangeira vinculando o agendamento à empresa correspondente.

Fonte: Elaborado pelo autor (2026).

A Tabela 22 exibe a estrutura da entidade Contas, que mantém o registro das instituições financeiras utilizadas e o controle dos saldos atuais.

Tabela 22 – Dicionário de Dados da Tabela Contas

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único incremental da conta.

nome	VARCHAR(100)	-	Não	Nome de identificação da conta (ex: Banco do Brasil Corrente).
saldo_inicial	DECIMAL(10,2)	-	Não	Valor de saldo inicial registrado na abertura da conta.
saldo_atual	DECIMAL(10,2)	-	Não	Saldo financeiro consolidado em tempo real na conta.
agencia	VARCHAR(8)	-	Sim	Número identificador da agência bancária.
nomeBanco	VARCHAR(100)	-	Sim	Nome da instituição financeira vinculada.
data_criacao	DATETIME	-	Não	Data e hora do cadastro da conta bancária.
cor	VARCHAR(8)	-	Não	Cor de representação visual da conta no painel principal.
empresa_id	INTEGER	FK	Não	Chave estrangeira associando a conta à empresa respectiva.
usuario_id	INTEGER	FK	Não	Chave estrangeira que indica o usuário que realizou a abertura.
tipo_conta_id	INTEGER	FK	Não	Chave estrangeira definindo a modalidade da conta (tipo_contas.id).

Fonte: Elaborado pelo autor (2026).

A Tabela 23 apresenta a entidade Tokens_reset_senha, que registra os tokens temporários gerados para os fluxos de recuperação e redefinição de senhas.

Tabela 23 – Dicionário de Dados da Tabela Tokens_reset_senha

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do token de redefinição.
token	VARCHAR(255)	-	Não	Código gerado enviado ao usuário para validar a troca de senha.

expira_em	DATETIME	-	Não	Data e hora limite para utilização do token.
usado	BOOLEAN	-	Não	Sinaliza se o token já foi consumido na troca de senha.
criado_em	DATETIME	-	Não	Data e hora em que a solicitação foi registrada.
usuario_id	INTEGER	FK	Não	Chave estrangeira apontando para o usuário solicitante.

Fonte: Elaborado pelo autor (2026).

A Tabela 24 define a entidade Contas_pagar, que armazena os lançamentos financeiros referentes a obrigações e despesas futuras ou liquidadas.

Tabela 24 – Dicionário de Dados da Tabela Contas_pagar

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do título a pagar.
descricao	VARCHAR(100)	-	Não	Histórico ou descrição detalhada da obrigação financeira.
valor	DECIMAL(10,2)	-	Não	Valor financeiro bruto do título.
data_vencimento	DATETIME	-	Não	Data limite estipulada para o pagamento sem encargos.
data_pagamento	DATETIME	-	Sim	Data em que o pagamento foi efetivamente realizado.
notas	TEXT	-	Sim	Observações gerais adicionais relativas ao título.
total_parcelas	INTEGER	-	Sim	Número identificador da parcela ou total de recorrências.
conta_id	INTEGER	FK	Sim	Chave estrangeira apontando para a conta de débito (contas.id).

categoria_id	INTEGER	FK	Sim	Chave estrangeira vinculando a categoria de custo (categorias.id).
situacao_id	INTEGER	FK	Não	Chave estrangeira definindo o status (tipo_situacao_conta.id).
empresa_id	INTEGER	FK	Não	Chave estrangeira associada à empresa devedora.
transacao_id	INTEGER	FK	Sim	Chave estrangeira opcional vinculando ao Fluxo de Caixa consolidado.

Fonte: Elaborado pelo autor (2026).

A Tabela 25 descreve a entidade Contas_receber, que armazena os lançamentos patrimoniais referentes a direitos, faturamentos e receitas futuras ou liquidadas.

Tabela 25 – Dicionário de Dados da Tabela Contas_receber

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do título a receber.
descricao	VARCHAR(100)	-	Não	Histórico descritivo da receita ou faturamento esperado.
valor	DECIMAL(10,2)	-	Não	Valor nominal bruto esperado para a entrada.
data_vencimento	DATETIME	-	Não	Data prevista para a liquidação por parte do cliente.
data_recebimento	DATETIME	-	Sim	Data real em que o recurso financeiro deu entrada no caixa.
notas	TEXT	-	Sim	Anotações e dados complementares da fatura.
cliente	VARCHAR(100)	-	Sim	Nome ou Razão Social do cliente/pagador.

conta_id	INTEGER	FK	Sim	Chave estrangeira da conta de destino padrão (contas.id).
categoria_id	INTEGER	FK	Sim	Chave estrangeira associando à categoria de receita.
situacao_id	INTEGER	FK	Não	Chave estrangeira definindo o status de recebimento.
empresa_id	INTEGER	FK	Não	Chave estrangeira vinculando o direito à empresa credora.
transacao_id	INTEGER	FK	Sim	Chave estrangeira opcional ligando o recebimento ao Fluxo de Caixa.

Fonte: Elaborado pelo autor (2026).

A Tabela 26 detalha a entidade Transacoes, que registra o extrato consolidado de todas as movimentações reais de fluxo de caixa efetuadas no sistema.

Tabela 26 – Dicionário de Dados da Tabela Transacoes

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do registro de transação.
descricao	VARCHAR(100)	-	Não	Descrição curta para identificação do lançamento efetuado.
valor	DECIMAL(10,2)	-	Não	Valor monetário líquido (positivo para receitas, negativo para despesas).
data	DATETIME	-	Não	Data e hora do fato gerador ou efetivação bancária.
notas	TEXT	-	Sim	Detalhamento e observações extras da movimentação.
recorrencia	VARCHAR(20)	-	Não	Definição de regra de repetição automática (ex: Mensal).

situacao	INTEGER	-	Não	Status interno da transação (ex: Conciliado, Pendente).
conta_id	INTEGER	FK	Sim	Chave estrangeira indicando a conta bancária impactada.
categoria_id	INTEGER	FK	Sim	Chave estrangeira que classifica a transação efetuada.
tipo_transacao_id	INTEGER	FK	Não	Chave estrangeira indicando o tipo de transação.
empresa_id	INTEGER	FK	Não	Chave estrangeira vinculando a movimentação à empresa dona dos dados.

Fonte: Elaborado pelo autor (2026).

A Tabela 27 apresenta a entidade Categorias, permitindo a classificação e agrupamento de receitas, despesas e transações financeiras.

Tabela 27 – Dicionário de Dados da Tabela Categorias

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único incremental da categoria.
nome	VARCHAR(100)	-	Não	Nome descritivo da categoria (ex: Alimentação, Salário).
cor	VARCHAR(7)	-	Sim	Código hexadecimal da cor utilizada na interface visual para relatórios.
esta_ativo	BOOLEAN	-	Não	Indica se a categoria está disponível para novos lançamentos.
tipo_categoria_id	INTEGER	FK	Não	Chave estrangeira que classifica a categoria (tipo_categorias.id).
empresa_id	INTEGER	FK	Não	Chave estrangeira delimitando a categoria à empresa proprietária.

Fonte: Elaborado pelo autor (2026).

A Tabela 28 define a entidade Tipo_categorias, que classifica as categorias estritamente entre receitas ou despesas.

Tabela 28 – Dicionário de Dados da Tabela Tipo_categorias

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do tipo de categoria.
nome	VARCHAR(100)	-	Não	Nome do domínio de classificação (ex: Receita, Despesa).

Fonte: Elaborado pelo autor (2026).

A Tabela 29 descreve a entidade Tipo_contas, contendo os tipos permitidos de contas financeiras suportadas pelo software.

Tabela 29 – Dicionário de Dados da Tabela Tipo_contas

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do tipo de conta.
nome	VARCHAR(100)	-	Não	Descrição da modalidade da conta.

Fonte: Elaborado pelo autor (2026).

A Tabela 30 apresenta a entidade Tipo_situacao_conta, que define o ciclo de status dos títulos de contas a pagar e receber.

Tabela 30 – Dicionário de Dados da Tabela Tipo_situacao_conta

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador do status da situação.
nome	VARCHAR(100)	-	Não	Nomenclatura do estado do título (ex: Aberto, Liquidado, Atrasado, Cancelado).

Fonte: Elaborado pelo autor (2026).

A Tabela 31 apresenta a entidade Tipo_transacoes, contendo os tipos operacionais aplicáveis a lançamentos de fluxo de caixa direto.

Tabela 31 – Dicionário de Dados da Tabela Tipo_transacoes

Nome do Atributo	Tipo de Dado	Chave (PK/FK)	Nulo	Descrição
id	INTEGER	PK	Não	Identificador único do tipo de transação.
nome	VARCHAR(100)	-	Não	Nome descritivo da transação (ex: Transferência, Ajuste de Saldo, Pagamento Direto).

Fonte: Elaborado pelo autor (2026).

4.5 Arquitetura do sistema

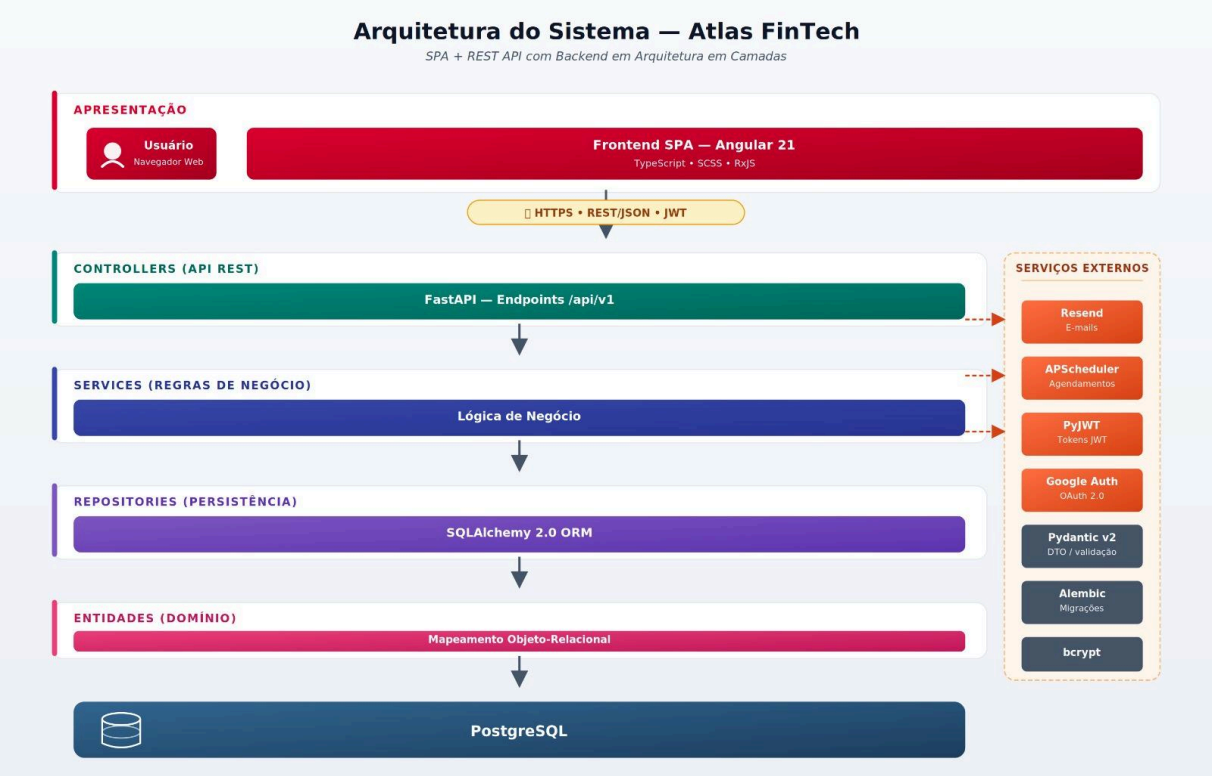
A arquitetura de software define a organização estrutural de um sistema, estabelecendo como seus componentes se relacionam, comunicam e colaboram para atender aos requisitos funcionais e não funcionais definidos (SOMMERVILLE, 2021). Uma arquitetura bem definida é fundamental para garantir a manutenibilidade, a escalabilidade e a segurança do sistema ao longo de seu ciclo de vida, servindo como referência técnica para todas as decisões de implementação (PRESSMAN; MAXIM, 2021).

A arquitetura em três camadas é um dos padrões arquiteturais mais consolidados na engenharia de software, pois promove a separação de responsabilidades entre os diferentes níveis do sistema, reduzindo o acoplamento e facilitando a evolução independente de cada componente (LARMAN, 2007). Fowler (2004) reforça que esse padrão organiza o sistema em níveis hierárquicos, nos quais cada camada possui responsabilidades bem definidas e se comunica apenas com as camadas adjacentes, tornando o sistema mais coeso, testável e fácil de manter.

O Atlas FinTech adota a arquitetura em três camadas, composta pela camada de apresentação, pela camada de backend e pela camada de dados. O frontend é implementado como uma SPA (*Single Page Application*) em Angular, que se comunica com o backend por meio de uma API REST desenvolvida em FastAPI,

utilizando o protocolo HTTPS e autenticação via tokens JWT para garantir a segurança das requisições. O backend é organizado internamente nas subcamadas de Controllers, Services, Repositories e Entidades, tendo o PostgreSQL como sistema gerenciador de banco de dados. A Figura 22 ilustra a arquitetura completa do sistema.

Figura 22 – Arquitetura do sistema Atlas FinTech



Fonte: Elaborado pelo autor (2026).

4.5.1 Camada de apresentação

A camada de apresentação é responsável por toda a interface com o usuário, sendo implementada como uma SPA (*Single Page Application*) desenvolvida em Angular 21. Essa abordagem elimina recarregamentos completos de página, proporcionando uma experiência de uso mais fluida e responsiva. A aplicação é construída com TypeScript, que adiciona tipagem estática ao JavaScript, reduzindo erros em tempo de desenvolvimento, e utiliza SCSS para estilização e RxJS para o gerenciamento de fluxos de dados assíncronos, como requisições à API e atualizações em tempo real da interface.

4.5.2 Camada de controllers

A camada de Controllers é responsável por receber, validar e encaminhar as requisições HTTP provenientes do frontend para as camadas internas do sistema. No Atlas FinTech, essa camada é implementada com o framework FastAPI, que expõe os endpoints da API REST sob o prefixo */api/v1*. Toda a comunicação entre o frontend e o backend é realizada via HTTPS, garantindo a criptografia dos dados em trânsito, e autenticada por meio de tokens JWT, que asseguram que apenas usuários devidamente autenticados possam acessar os recursos protegidos do sistema.

4.5.3 Camada de services

A camada de Services concentra as regras de negócio do sistema, sendo responsável por processar as requisições recebidas dos Controllers e aplicar as lógicas necessárias antes de acionar a camada de persistência. Essa separação garante que as regras de negócio permaneçam independentes da interface e da infraestrutura de dados, facilitando a manutenção e a realização de testes automatizados. É nessa camada que são executadas operações como o cálculo de fluxo de caixa, a geração de relatórios financeiros e o processamento de conciliações bancárias.

4.5.4 Camada de repositories e entidades

A camada de repositories é responsável pelo acesso e pela manipulação dos dados persistentes, abstraindo as operações de banco de dados por meio do ORM SQLAlchemy 2.0. Essa abordagem permite que as operações de leitura e escrita sejam realizadas diretamente sobre objetos Python, sem a necessidade de escrever consultas SQL manualmente, reduzindo a complexidade e os riscos de inconsistência. As entidades representam o modelo de domínio do sistema, definindo as estruturas de dados mapeadas diretamente para as tabelas do banco de dados. As migrações de esquema são gerenciadas pelo Alembic, garantindo que as alterações no modelo de dados sejam aplicadas de forma controlada e rastreável.

4.5.5 Banco de dados

O Atlas FinTech utiliza o PostgreSQL como sistema gerenciador de banco de dados relacional. O PostgreSQL é reconhecido por sua robustez, conformidade com o padrão SQL, suporte a transações ACID e capacidade de lidar com grandes volumes de dados, características essenciais para um sistema de gestão financeira. A escolha por um banco de dados relacional justifica-se pela natureza estruturada e interdependente dos dados financeiros manipulados pelo sistema, como transações, contas bancárias, categorias e relatórios, que demandam integridade referencial e consistência nas operações.

4.5.6 Serviços externos

O Atlas FinTech integra um conjunto de serviços e bibliotecas externas para ampliar suas capacidades sem aumentar a complexidade interna do sistema. O envio de e-mails transacionais, como confirmações de cadastro e relatórios agendados, é realizado por meio do serviço Resend. O agendamento de tarefas recorrentes é gerenciado pelo APScheduler, que permite a execução automática de rotinas em intervalos configuráveis. A autenticação via Google é implementada com o protocolo OAuth 2.0, oferecendo ao usuário uma alternativa segura ao login tradicional. A validação e a serialização dos dados trafegados entre as camadas são realizadas pelo Pydantic v2, enquanto o bcrypt é utilizado para o armazenamento seguro de senhas por meio de funções de hash criptográfico.

4.6 Implementação do sistema

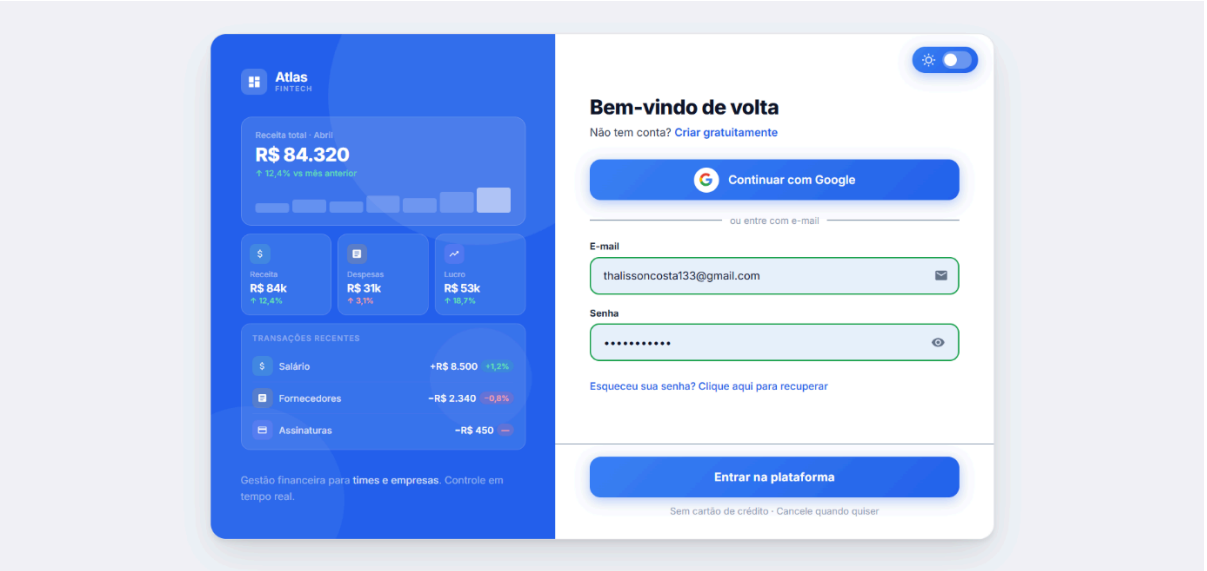
Esta seção apresenta a implementação do Atlas FinTech, demonstrando as telas desenvolvidas e as funcionalidades entregues ao longo do projeto. A implementação representa a materialização de todos os elementos definidos nas etapas anteriores de modelagem, traduzindo os requisitos levantados, os diagramas elaborados e a arquitetura projetada em um sistema funcional e utilizável.

Para cada tela são descritos os campos, ações e comportamentos disponíveis ao usuário, permitindo compreender como cada requisito funcional foi implementado na interface. As telas estão organizadas seguindo os principais fluxos de uso do sistema, desde a autenticação até as funcionalidades financeiras avançadas, como análise financeira, conciliação bancária e emissão de relatórios.

4.6.1 Tela de login

A Figura 23 apresenta a tela de Login do sistema, utilizada para autenticação e acesso à plataforma. A interface permite que o usuário realize o login por meio do endereço de e-mail e senha cadastrados, além de disponibilizar a opção de autenticação com conta Google e recuperação de senha em caso de esquecimento.

Figura 23 – Tela de login



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário realize o acesso ao sistema informando suas credenciais de autenticação, como e-mail e senha. Além disso, a tela oferece funcionalidades complementares, como login com conta Google, recuperação de senha e criação de uma nova conta, proporcionando maior praticidade e segurança no acesso à plataforma.

Os campos apresentados na tela, conforme a Figura 23, podem ser descritos de acordo com a Tabela 32:

Tabela 32 – Campos da interface de login

Item	Campo	Ação	Restrições/Observações
01	E-mail	Receber o e-mail de login	N/A
02	Senha	Receber a senha de login	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 23 será descrito de acordo com a Tabela 33:

Tabela 33 – Botões da interface da tela de login

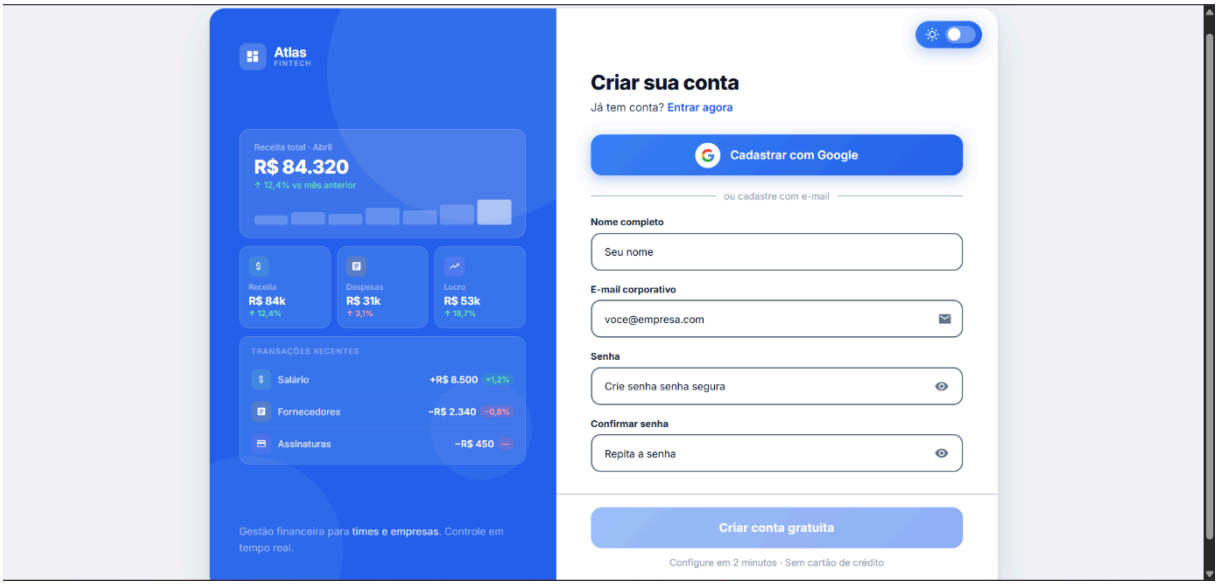
Item	Comando	Ação	Restrições/Observações
01	Botão de modo claro e escuro	Trocar a cor da tela para claro ou escuro.	N/A
02	Link Criar gratuitamente	Criar conta de forma gratuita	N/A
03	Continuar com google	Usar conta com google para login.	N/A
04	Esqueceu a senha? Clique aqui para recuperar	Recuperar senha de acesso ao sistema.	N/A
05	Entrar na plataforma	Entrar na plataforma.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.1.1 Tela de cadastro de conta

A Figura 24 apresenta a tela de Cadastro do sistema, utilizada para o registro de novos usuários na plataforma. A interface permite o preenchimento de informações como nome completo, endereço de e-mail e senha de acesso, além de disponibilizar a opção de cadastro por meio da conta Google, proporcionando maior praticidade no processo de criação da conta.

Figura 24 – Tela de cadastro de conta



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário realize o cadastro de uma nova conta, informando dados como nome completo, e-mail, senha e confirmação de senha. Além disso, a tela oferece a opção de autenticação com conta Google, facilitando o acesso e tornando o processo de registro mais rápido e seguro.

Os campos apresentados na tela, conforme a Figura 24, podem ser descritos de acordo com a Tabela 34:

Tabela 34 – Campos da interface de cadastro de conta

Item	Campo	Ação	Restrições/Observações
01	Nome completo	Adicionar o nome completo do usuário	N/A
02	E-mail corporativo	Receber o e-mail de login	N/A
03	Senha	Criar a senha de login	N/A
04	Confirmar senha	Confirmar senha de login	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 24 será descrito de acordo com a Tabela 35:

Tabela 35 – Botões da tela de cadastro de conta

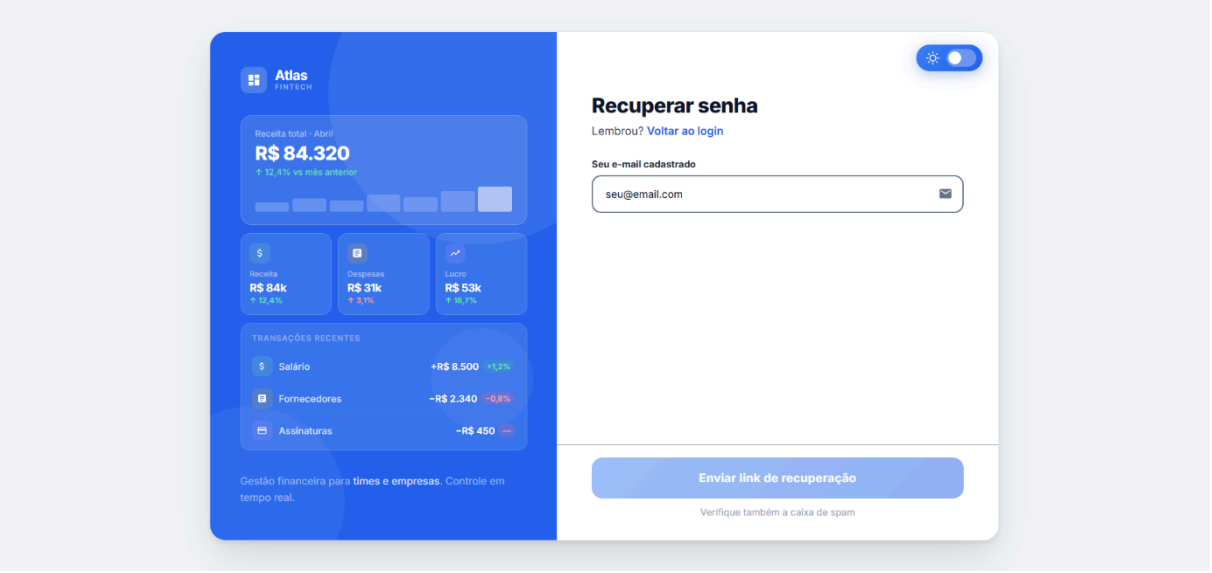
Item	Comando	Ação	Restrições/Observações
01	Link entrar agora	Link de direcionamento para tela de login	N/A
02	Cadastrar com google	Criar conta com o google.	N/A
03	Criar conta gratuita	Criar uma conta na plataforma.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.1.2 Tela de recuperação de senha

A Figura 25 apresenta a tela de Recuperação de Senha do sistema, utilizada para auxiliar o usuário na redefinição do acesso à plataforma. A interface permite o envio de um link de recuperação por meio do endereço de e-mail cadastrado, possibilitando a restauração da senha de forma segura e prática.

Figura 25 – Tela de recuperação de senha



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário solicite a recuperação da senha de acesso ao sistema, informando o e-mail cadastrado na plataforma. Após o preenchimento da informação, o processo pode ser iniciado por meio do botão “Enviar link de recuperação”, contribuindo para um acesso mais seguro e eficiente à conta do usuário.

Os campos apresentados na tela, conforme a Figura 25, podem ser descritos de acordo com a Tabela 36:

Tabela 36 – Campos da interface de recuperação de senha

Item	Campo	Ação	Restrições/Observações
01	Seu e-mail de cadastro	Adicionar e-mail de cadastro para recuperação de senha.	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 25 será descrito de acordo com a Tabela 37:

Tabela 37 – Botões da tela de recuperação de senha

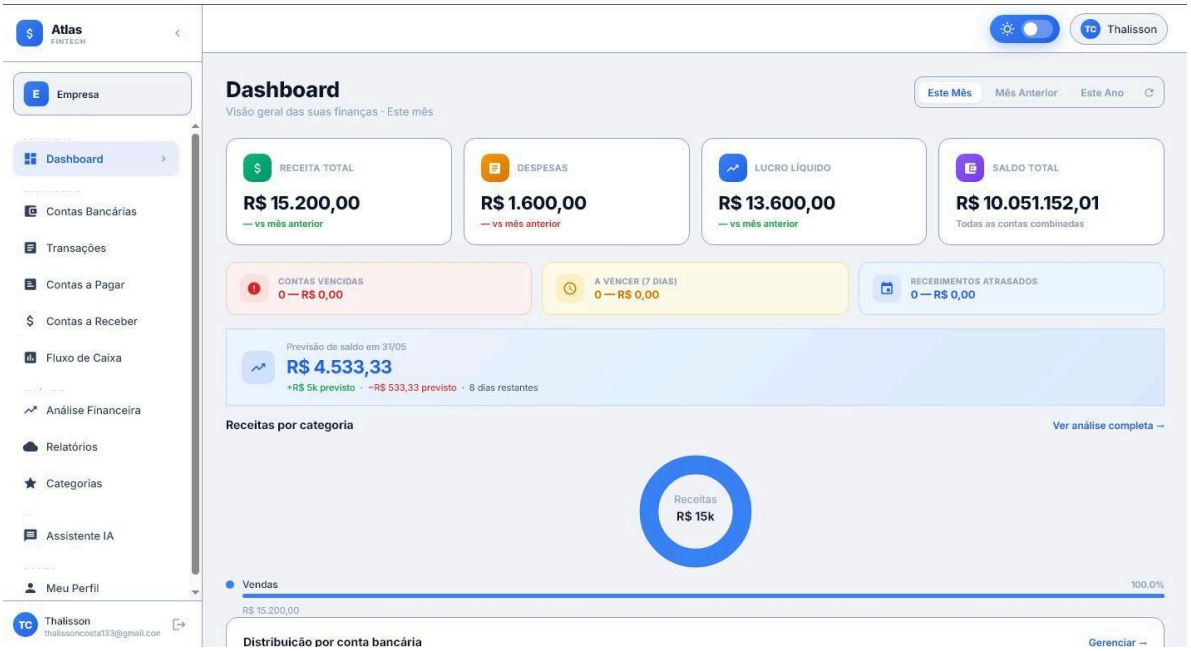
Item	Comando	Ação	Restrições/Observações
01	Link voltar ao login	Link de direcionamento para tela de login	N/A
02	Enviar link de recuperação	Enviar link de recuperação no e-mail cadastrado.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.2 Tela de dashboard

A Figura 26 apresenta a tela de *Dashboard* do sistema, na qual são exibidos indicadores financeiros, métricas e informações consolidadas sobre receitas, despesas, saldo e fluxo de caixa. Essa visualização permite ao usuário acompanhar de forma simples e objetiva os principais dados da plataforma, contribuindo para maior controle e análise financeira.

Figura 26 – Tela de dashboard web



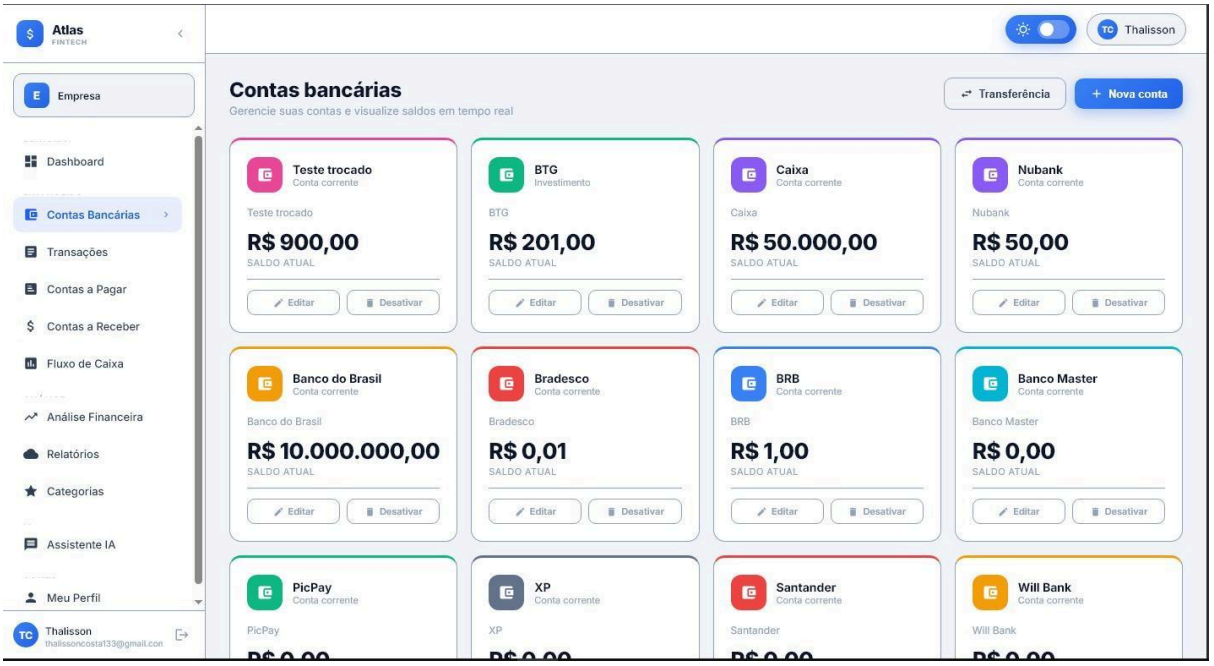
Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize de forma consolidada os principais indicadores financeiros da empresa. O dashboard apresenta os KPIs de Saldo Total, Receitas do mês, Despesas do mês, Fluxo Líquido, Contas a Pagar e Contas a Receber, além de gráficos mensais de movimentação financeira, gráficos por conta bancária e uma listagem das transações recentes, proporcionando uma visão abrangente e atualizada da situação financeira para apoiar a tomada de decisões.

4.6.3 Tela de contas bancárias e registros

A Figura 27 apresenta a tela de contas bancárias do sistema, responsável por exibir os saldos e informações das contas cadastradas, incluindo contas correntes, investimentos e outros recursos financeiros. A interface permite ao usuário acompanhar o capital disponível, registrar novas contas, realizar transferências, editar informações e gerenciar os registros cadastrados de forma simples e intuitiva.

Figura 27 – Tela de contas bancárias e registros



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize com mais clareza os valores que a empresa possui à disposição, seja em dinheiro, cartão de crédito, investimentos ou valores em conta bancária. Essa organização facilita a tomada de decisões.

O comando da tela observado na Figura 27 será descrito de acordo com a Tabela 38:

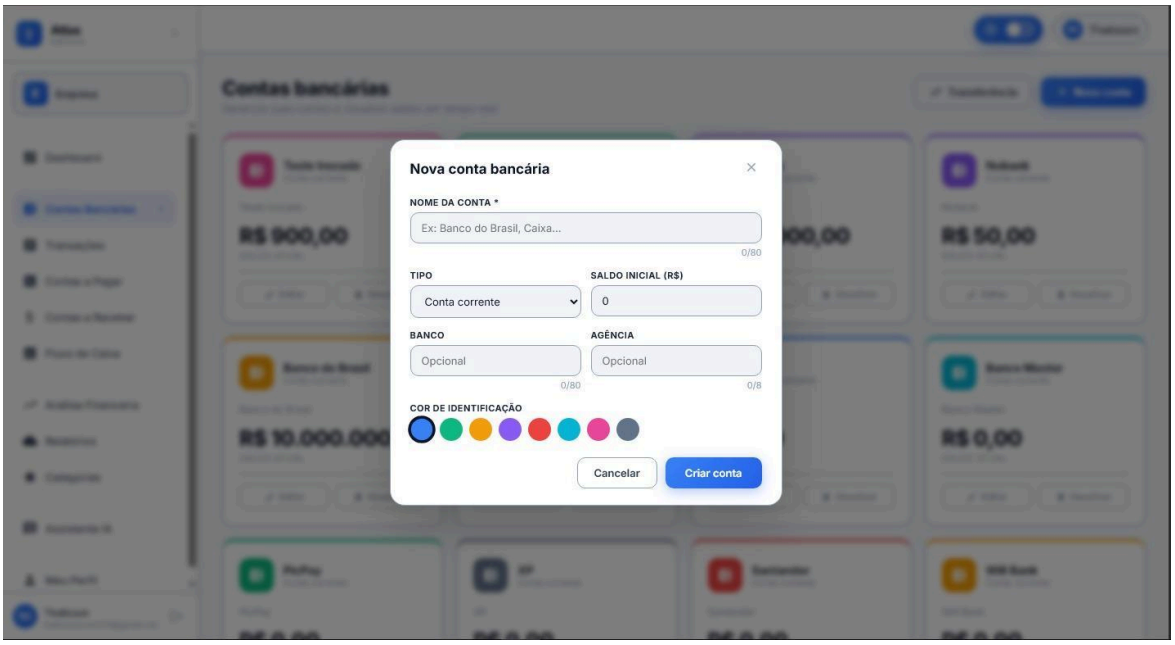
Tabela 38 – Botões da interface da tela de contas bancárias

Item	Comando	Ação	Restrições/Observações
01	Editar	Editar o card e mudar informações	N/A
02	Apagar	Apagar o card	N/A
03	Nova Conta	Registrar um Card novo	Essa função vai criar uma conta bancária.
04	Transferência	Transferência entre contas	Transfere valor de uma conta para outra

Fonte: Elaborado pelo autor (2026).

A Figura 28 apresenta a tela de cadastro de contas bancárias do sistema, utilizada para registrar contas do tipo corrente, poupança e investimento. Nessa interface, o usuário pode inserir informações da conta, como saldo inicial, banco e agência, além de personalizar sua identificação, facilitando o gerenciamento financeiro da plataforma.

Figura 28 – Tela que registra a conta bancária



Fonte: Elaborado pelo autor (2026).

A interface acima permite ao usuário cadastrar uma nova conta bancária, informando dados como nome, tipo da conta, saldo inicial, banco e agência. Após o preenchimento das informações, o registro é realizado por meio do botão “Criar conta”, possibilitando o gerenciamento centralizado das contas cadastradas no sistema. Adicionalmente, o botão “Cancelar” permite interromper a operação e retornar à tela anterior sem efetivar o cadastro, evitando registros indevidos ou incompletos.

Os campos apresentados na tela, conforme a Figura 28, podem ser descritos de acordo com a Tabela 39:

Tabela 39 – Campos da interface para criar uma nova conta

Item	Campo	Ação	Restrições/Observações
01	Nome da conta	Receber o nome	N/A
02	Tipo	Receber o tipo da conta	Conta corrente, poupança ou investimento.
03	Saldo inicial	Receber o valor na hora de registrar	N/A
04	Banco	Receber nome do banco	Nome do banco
05	Agência	Receber número da agência	Número da agência
06	Cor de Identificação	Identificar pela cor a sua conta	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 28 será descrito de acordo com a Tabela 40:

Tabela 40 – Botões da interface da tela de criação de conta bancária

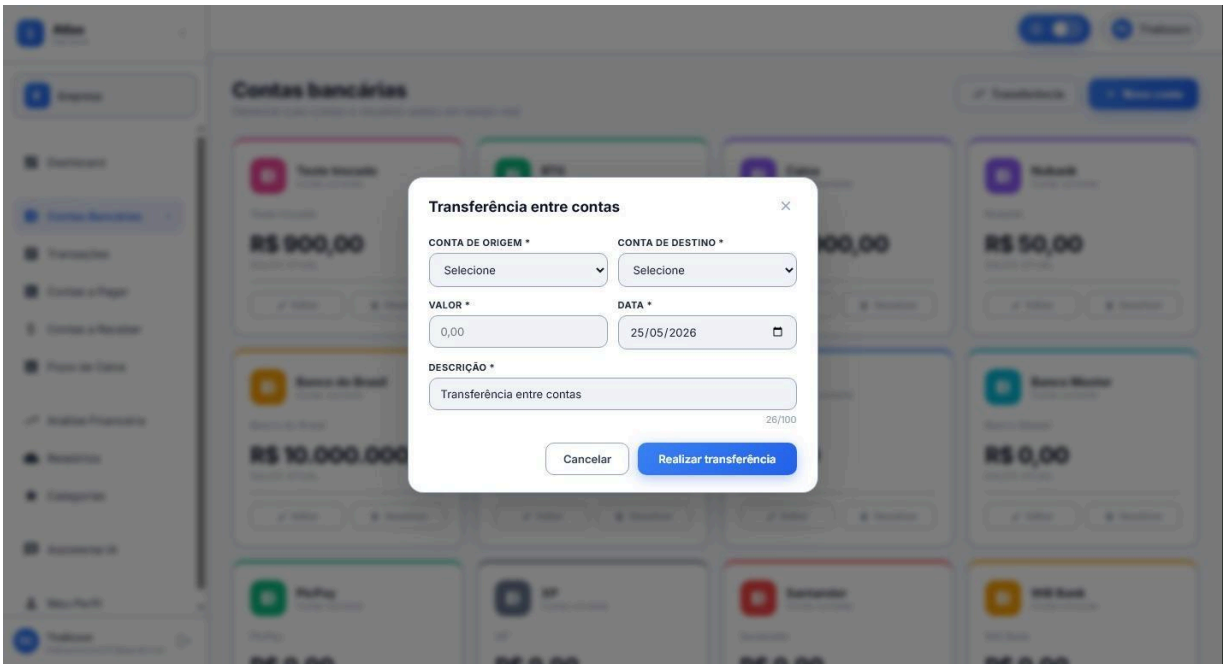
Item	Comando	Ação	Restrições/Observações
01	Botão de fechar tela	Fechar tela	N/A
02	Criar Conta	Criar uma nova conta	N/A
03	Cancelar	Cancelar registro de conta	N/A

Fonte: Elaborado pelo autor (2026).

A Figura 29 apresenta a tela de transferência entre contas do sistema, permitindo ao usuário selecionar a conta de origem e destino, informar o valor, a data e a descrição da operação. Essa funcionalidade possibilita o gerenciamento e

movimentação de recursos financeiros entre contas cadastradas de forma simples e organizada.

Figura 29 – Tela de transferência entre contas



Fonte: Elaborado pelo autor (2026).

A interface acima permite ao usuário realizar transferências entre contas cadastradas no sistema, selecionando a conta de origem e a conta de destino, além de informar o valor, a data e uma descrição da operação. Após o preenchimento das informações, a transferência é realizada por meio do botão “Realizar transferência”, registrando a movimentação financeira no sistema. Adicionalmente, o botão “Cancelar” permite interromper a operação e retornar à tela anterior sem efetivar a transferência, evitando registros incorretos ou indesejados.

Os campos apresentados na tela, conforme a Figura 29, podem ser descritos de acordo com a Tabela 41:

Tabela 41 – Campos da tela de transferência entre contas

Item	Campo	Ação	Restrições/Observações
01	Conta de Origem	Selecionar conta de origem	Selecionar conta que vai sair os valores.

02	Conta de Destino	Selecionar conta de destino	Selecionar conta que vai receber os valores.
03	Valor	Receber o valor da transferência.	Receber o valor da transferência.
04	Data	Data de transferência	N/A
05	Descrição	Descrição da transferência.	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 29 será descrito de acordo com a Tabela 42:

Tabela 42 – Botões da tela de transferência entre contas

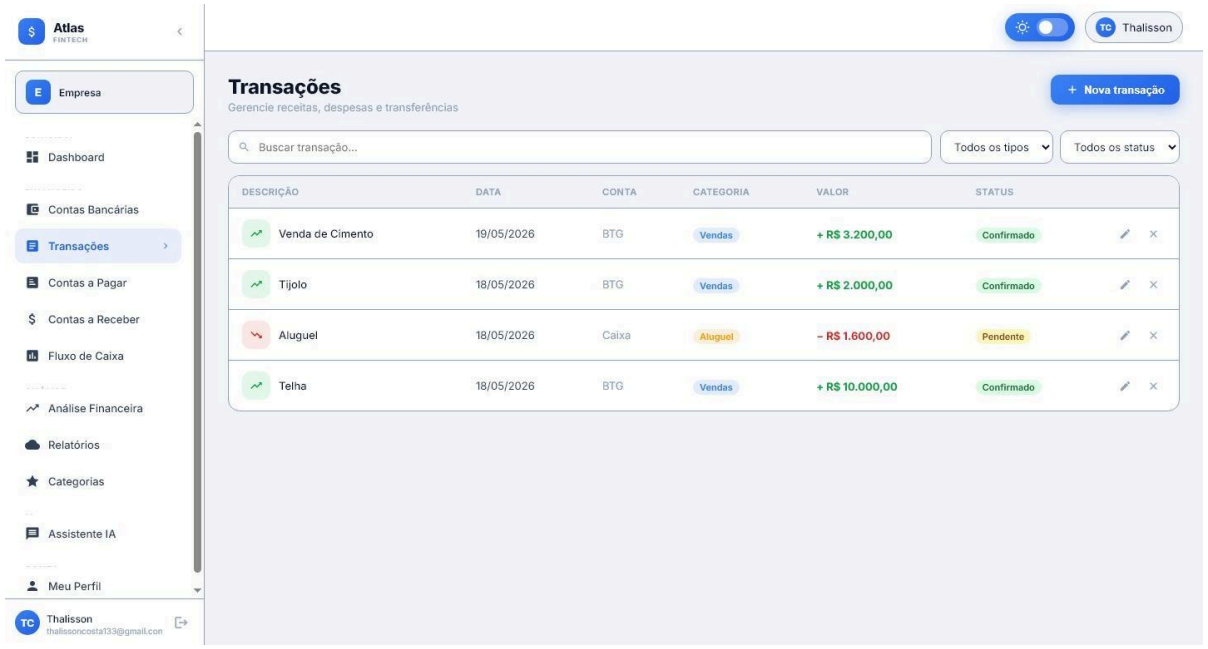
Item	Comando	Ação	Restrições/Observações
01	Botão de fechar tela	Fechar tela	N/A
02	Realizar transferência	Transferir um valor de uma conta para outra	Transfere o valor de uma conta para outra conta.
03	Cancelar	Cancelar transferência	N/A

Fonte: Elaborado pelo autor (2026).

4.6.4 Tela de transações registradas

A Figura 30 apresenta a tela de transações do sistema, na qual são exibidas as movimentações financeiras de entradas, saídas e transferências realizadas pela empresa. A interface disponibiliza filtros por tipo de transação e status, além de um campo de busca por nome para localização de registros específicos. Também é possível criar, editar e excluir transações, facilitando o acompanhamento das operações financeiras de forma rápida e organizada.

Figura 30 – Tela de transações



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite ao usuário visualizar de forma organizada as transações financeiras registradas no sistema, possibilitando o acompanhamento de entradas, saídas e transferências. A tela também disponibiliza ferramentas de filtragem por tipo de transação e status, além de um campo de busca por nome, contribuindo para a localização rápida de registros específicos e para um controle mais eficiente das movimentações financeiras da empresa.

Os campos da tela apresentados conforme a Figura 30 podem ser descritos conforme a Tabela 43:

Tabela 43 – Campos da tela transações

Item	Campo	Ação	Restrições/Observações
01	Todos os tipos	Filtrar por receita, despesa e transferência.	N/A
02	Todos os status.	Filtrar por status de confirmado, pendente e cancelado.	N/A
03	Buscar	Buscar por descrição	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 30 será descrito de acordo com a Tabela 44:

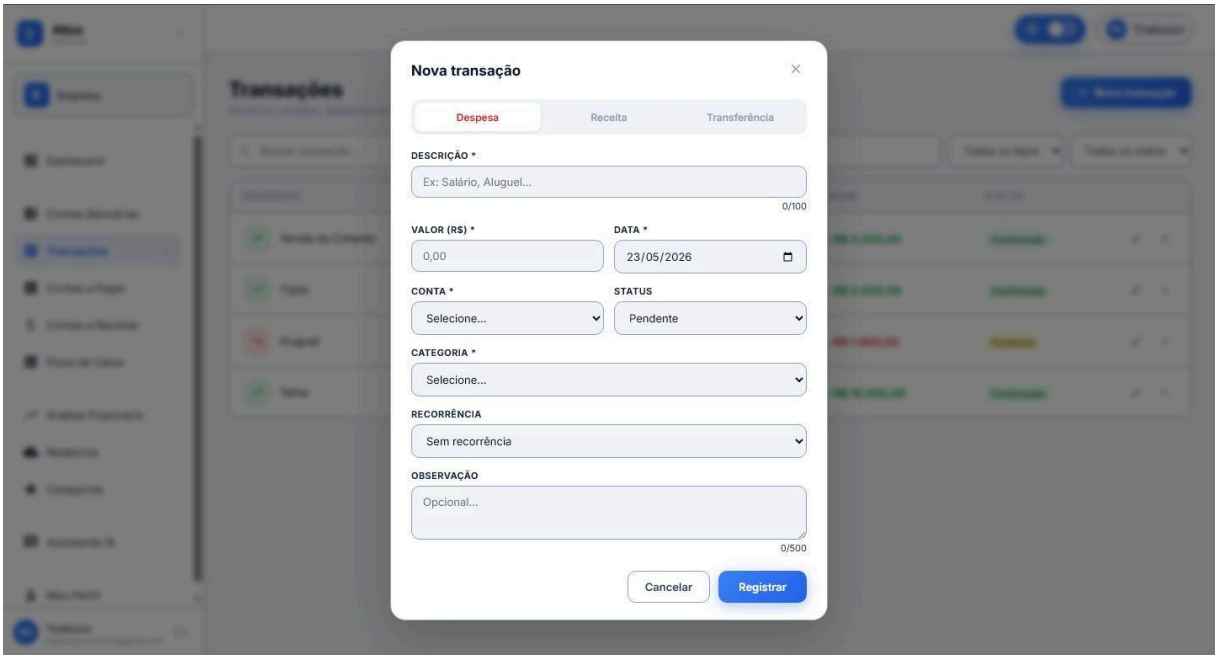
Tabela 44 – Botões da tela transações

Item	Comando	Ação	Restrições/Observações
01	Nova Transação	Criar uma transação nova	N/A
02	Editar	Editar uma transação existente	N/A
03	Apagar	Apagar uma transação	N/A

Fonte: Elaborado pelo autor (2026).

A Figura 31 apresenta a tela de registro de despesas do sistema, utilizada para o cadastro de saídas financeiras. Nessa interface, o usuário pode inserir informações detalhadas da movimentação, permitindo maior organização e controle das despesas registradas na plataforma.

Figura 31 – Tela de registro de despesas



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite ao usuário registrar uma nova despesa, informando dados como descrição, valor, data, conta, categoria e status da movimentação. Após o preenchimento das informações, o registro é realizado por meio do botão “Registrar”, enquanto o botão “Cancelar” permite interromper a operação sem salvar os dados inseridos.

Os campos apresentados na tela, conforme a Figura 31, podem ser descritos de acordo com a Tabela 45:

Tabela 45 – Campos da tela nova transação de despesas

Item	Campo	Ação	Restrições/Observações
01	Descrição	Nome da despesa	N/A
02	Valor	Valor da despesa.	N/A
03	Data da transação	Data da despesa.	N/A
04	Conta bancária	Selecionar uma conta	Conta que vai debitar o valor da despesa.
05	Status da despesa	Selecionar status	Confirmado e Pendente
06	Categoria	Selecionar uma categoria	N/A
07	Recorrência	Selecionar a recorrência	Recorrência mensal, semanal e anual.
08	Observação	Informar algo que seja útil na transação	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 31 será descrito de acordo com a Tabela 46:

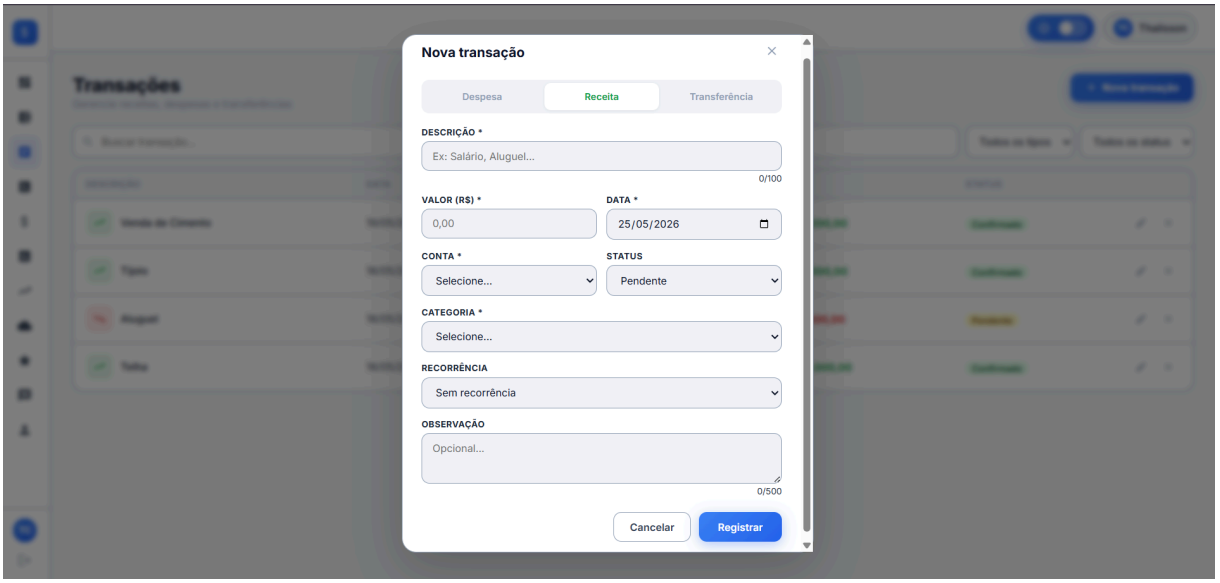
Tabela 46 – Botões da tela nova transação de despesas

Item	Comando	Ação	Restrições/Observações
01	Botão de Fechar	Fechar a tela	N/A
02	Botão de Criar transação	Criar uma nova transação	N/A
03	Cancelar	Cancelar transação	N/A

Fonte: Elaborado pelo autor (2026).

A Figura 32 apresenta a tela de registro de receitas do sistema, utilizada para o cadastro de entradas financeiras. Nessa interface, o usuário pode inserir informações detalhadas da movimentação, permitindo maior organização e controle das receitas registradas na plataforma.

Figura 32 – Tela de registro de receitas



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite ao usuário registrar uma nova receita, informando dados como descrição, valor, data, conta, categoria e status da movimentação. Após o preenchimento das informações, o registro é realizado por

meio do botão “Registrar”, enquanto o botão “Cancelar” permite interromper a operação sem salvar os dados inseridos.

Os campos apresentados na tela, conforme a Figura 32, podem ser descritos de acordo com a Tabela 47:

Tabela 47 – Campos da tela nova transação de receita

Item	Campo	Ação	Restrições/Observações
01	Descrição	Nome da receita	N/A
02	Valor	Valor da receita.	N/A
03	Data da transação	Data da receita.	N/A
04	Conta bancaria	Selecionar uma conta	Conta que vai receber o valor da receita.
05	Status da receita	Selecionar status	Confirmado e Pendente
06	Categoria	Selecionar uma categoria	N/A
07	Recorrência	Selecionar a recorrência	Recorrência mensal, semanal e anual.
08	Observação	Informar algo que seja útil na transação	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 32 será descrito de acordo com a Tabela 48:

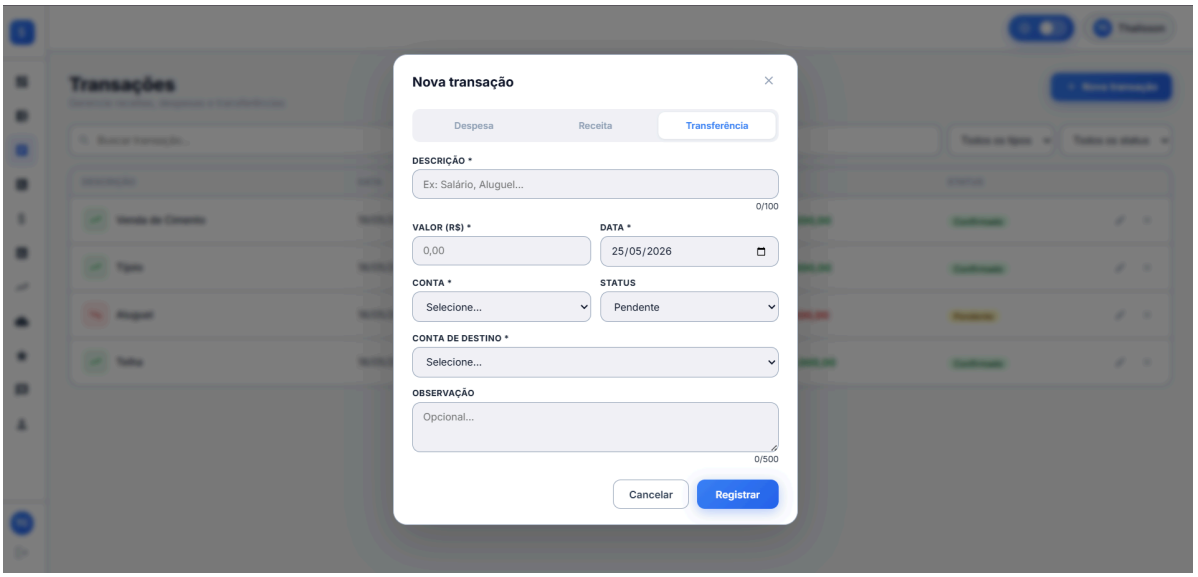
Tabela 48 – Botões da tela nova transação de receita

Item	Comando	Ação	Restrições/Observações
01	Botão de Fechar	Fechar a tela	N/A
02	Botão de registrar	Criar uma nova transação	N/A
03	Cancelar	Cancelar transação	N/A

Fonte: Elaborado pelo autor (2026).

A Figura 33 apresenta a tela de registro de transações do sistema, com destaque para a aba de receitas, utilizada para o cadastro de entradas financeiras. Nessa interface, o usuário pode inserir informações detalhadas da movimentação, como descrição, valor, data, conta, status, conta de destino e observações, permitindo maior organização e controle das receitas registradas na plataforma.

Figura 33 – Tela de registro de transferência



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite ao usuário registrar uma nova transferência entre contas no sistema, possibilitando o preenchimento de informações como descrição, valor, data, conta de origem, conta de destino, status da movimentação e

observações. Após a inserção dos dados, o processo pode ser concluído por meio do botão “Registrar”, enquanto a opção “Cancelar” permite interromper a operação sem efetivar o registro da transferência.

Os campos apresentados na tela, conforme a Figura 33, podem ser descritos de acordo com a Tabela 49:

Tabela 49 – Campos da tela transferência

Item	Campo	Ação	Restrições/Observações
01	Descrição	Nome da transferência	N/A
02	Valor	Valor da transferência.	N/A
03	Data da transação	Data da transferência.	N/A
04	Conta bancaria	Selecionar uma conta	Conta que vai sair o valor da transferência.
05	Status da transferência	Selecionar status	Confirmado e Pendente
06	Conta destino	Selecionar a conta destino	Conta que vai receber o valor da transferência
07	Observação	Informar algo que seja útil na transferência	N/A

Fonte: Elaborado pelo autor (2026)

O comando da tela observado na Figura 33 será descrito de acordo com a Tabela 50:

Tabela 50 – Botões da tela nova transação de transferência

Item	Comando	Ação	Restrições/Observações
01	Botão de Fechar	Fechar a tela	N/A

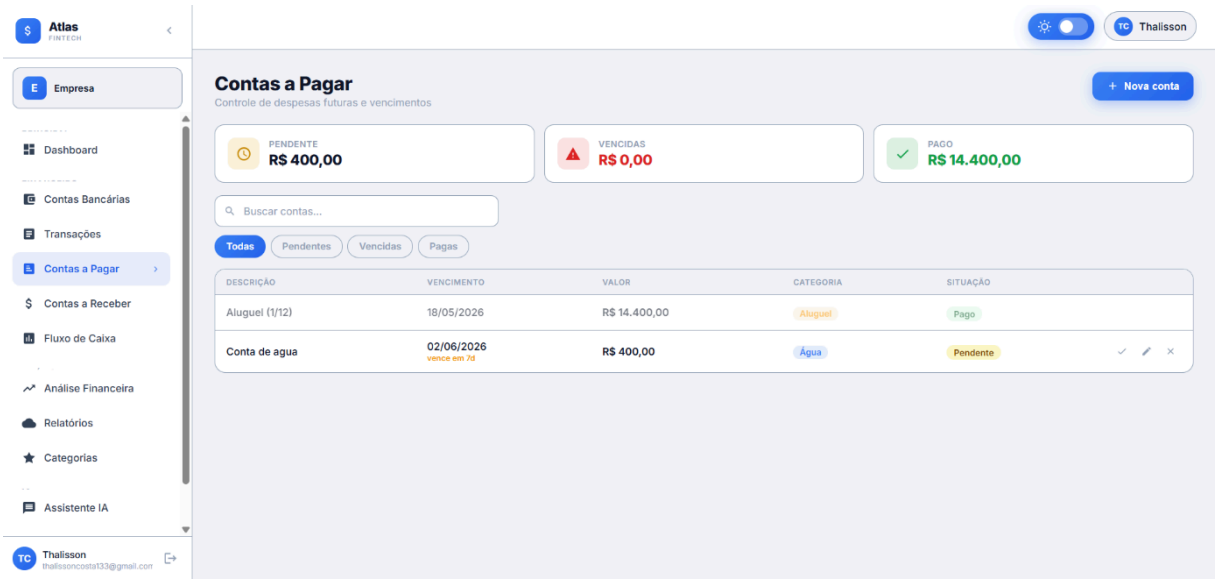
02	Botão de registrar	Criar uma nova transação	N/A
03	Cancelar	Cancelar transação	N/A

Fonte: Elaborado pelo autor (2026).

4.6.5 Tela de contas a pagar

A Figura 34 apresenta a tela de Contas a Pagar, permitindo que o usuário visualize todas as obrigações financeiras que precisam ser quitadas. A interface também oferece um botão para cadastrar novas contas a pagar, além de possibilitar a visualização das informações relacionadas aos pagamentos, como descrição, vencimento, valor, categoria e situação. Essa visualização permite identificar de forma rápida e organizada as despesas financeiras da empresa.

Figura 34 – Tela de contas a pagar



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite ao usuário visualizar e gerenciar as contas a pagar cadastradas no sistema, apresentando informações como vencimento, valor, categoria e situação do pagamento. Além disso, a tela possibilita o cadastro de novas contas, contribuindo para um melhor controle financeiro da empresa.

Os campos apresentados na tela, conforme a Figura 34, podem ser descritos de acordo com a Tabela 51:

Tabela 51 – Campos da tela contas a pagar

Item	Campo	Ação	Restrições/Observações
01	Buscar contas	Buscar contas pelo nome.	N/A
02	Filtrar por categoria	Todas as contas, contas pendentes, vencidas e pagas.	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 34 será descrito de acordo com a Tabela 52:

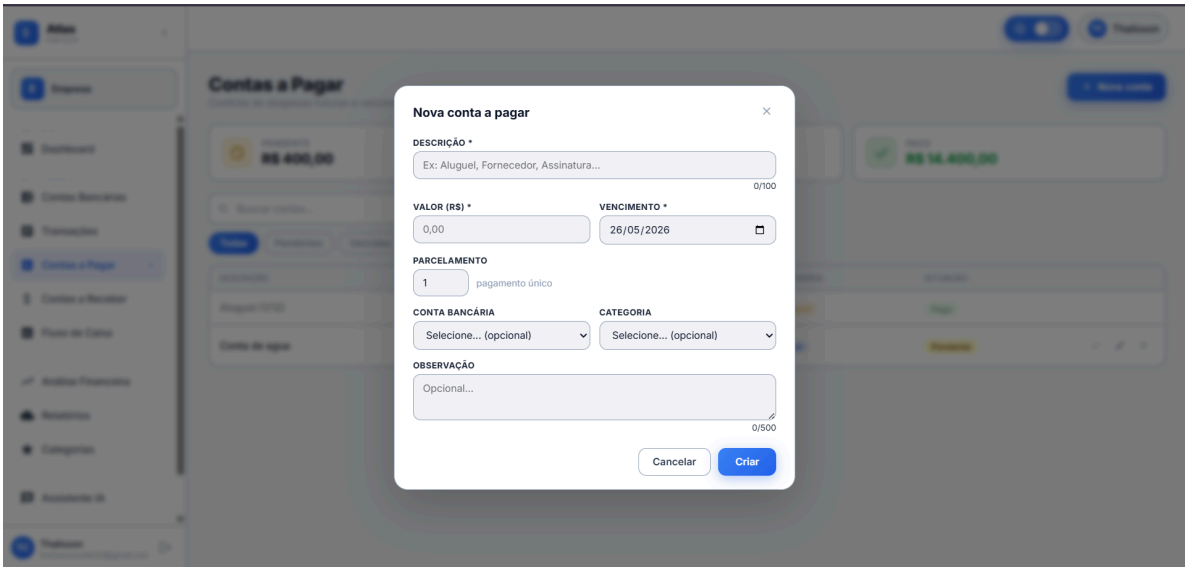
Tabela 52 – Botões da tela contas a pagar

Item	Comando	Ação	Restrições/Observações
01	Conta a pagar	Criar uma nova conta	N/A
02	Registrar pagamento	Registrar conta como paga.	N/A
03	Editar	Editar conta a pagar	N/A
04	Cancelar	Cancelar conta a pagar	N/A

Fonte: Elaborado pelo autor (2026).

A Figura 35 apresenta a tela de cadastro de contas a pagar do sistema, utilizada para o registro de novas obrigações financeiras. Nessa interface, o usuário pode informar dados como descrição, valor, vencimento, parcelamento, conta bancária, categoria e observações, permitindo um cadastro mais detalhado das contas. Após o preenchimento das informações, o registro pode ser realizado por meio do botão “Criar”, enquanto o botão “Cancelar” permite interromper a operação sem salvar os dados inseridos.

Figura 35 – Tela de registro das contas a pagar



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário registre uma nova conta a pagar, informando dados como descrição, valor, data de vencimento, parcelamento, conta bancária, categoria e observações adicionais. Após o preenchimento das informações, o cadastro é concluído por meio do botão “Criar”, enquanto o botão “Cancelar” permite interromper a operação sem salvar os dados inseridos.

Os campos apresentados na tela, conforme a Figura 35, podem ser descritos de acordo com a Tabela 53:

Tabela 53 – Campos tela contas a pagar

Item	Campo	Ação	Restrições/Observações
01	Descrição	Nome da conta a pagar	N/A
02	Valor	Valor da conta a pagar	N/A
03	Data de Vencimento	Adicionar data de vencimento	N/A
04	Parcelamento	Parcelar a conta	N/A

05	Conta bancária	Selecionar conta que vai sair o pagamento.	N/A
06	Categoria	Categoria de conta a pagar	N/A
07	Observação	Digitar alguma informação complementar	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 35 será descrito de acordo com a Tabela 54:

Tabela 54 – Botões da tela contas a pagar

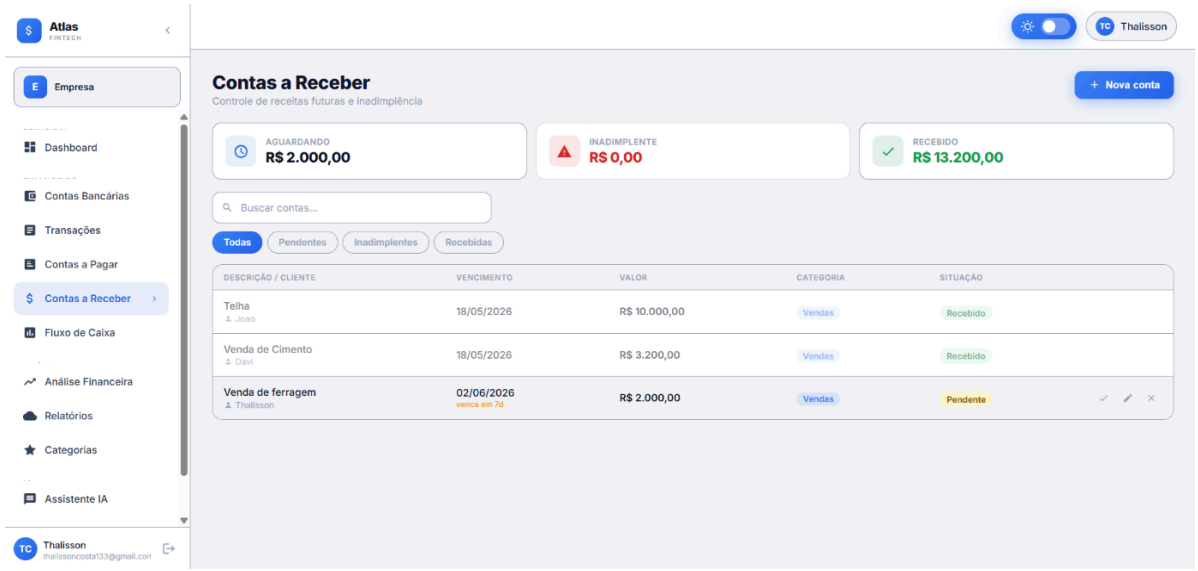
Item	Comando	Ação	Restrições/Observações
01	Fechar tela	Fechar a tela	N/A
02	Criar conta a pagar	Criar uma nova conta a pagar	N/A
03	Cancelar	Cancelar registro de conta a pagar	N/A

Fonte: Elaborado pelo autor (2026).

4.6.6 Tela de contas a receber

A Figura 36 apresenta a tela de Contas a Receber, permitindo que o usuário visualize e acompanhe os valores previstos para recebimento. A interface também oferece um botão para cadastrar novas contas a receber, além de possibilitar o gerenciamento dos registros existentes. Essa visualização permite identificar de forma rápida as entradas financeiras, apresentando informações como cliente, vencimento, valor, categoria e situação do recebimento.

Figura 36 – Tela de contas a receber



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize e gerencie as contas a receber cadastradas no sistema, apresentando informações como descrição, vencimento, valor, categoria e situação do recebimento. Além disso, a tela possibilita o cadastro de novas contas, contribuindo para um melhor controle das entradas financeiras da empresa.

O comando da tela observado na Figura 36 será descrito de acordo com a Tabela 55:

Tabela 55 – Campos da tela contas a receber

Item	Campo	Ação	Restrições/Observações
01	Buscar contas	Buscar contas pelo nome.	N/A
02	Filtrar por categoria	Todas as contas, contas pendentes, inadimplentes e recebidas.	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 36 será descrito de acordo com a Tabela 56:

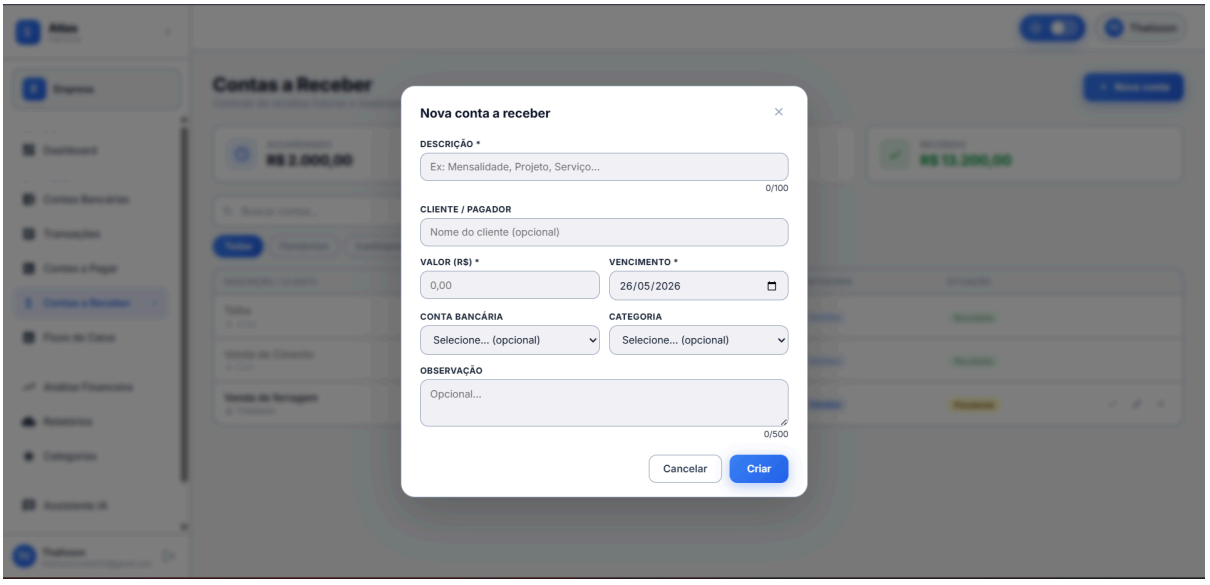
Tabela 56 – Botões da tela contas a receber

Item	Comando	Ação	Restrições/Observações
01	Nova conta a receber	Criar uma nova conta	N/A
02	Confirmar recebimento	Registrar conta como recebida.	N/A
03	Editar	Editar conta a receber	N/A
04	Cancelar	Cancelar conta a receber	N/A

Fonte: Elaborado pelo autor (2026).

A Figura 37 apresenta a tela de cadastro de contas a receber do sistema, utilizada para o registro de novas entradas financeiras. Nessa interface, o usuário pode cadastrar informações como descrição, cliente ou pagador, valor, data de vencimento, conta bancária, categoria e observações, permitindo maior organização e controle dos recebimentos. O processo pode ser concluído por meio do botão “Criar”, enquanto a opção “Cancelar” permite interromper a operação sem salvar os dados inseridos.

Figura 37 – Tela de registro de contas a receber



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário registre uma nova conta a receber, informando dados como descrição, cliente ou pagador, valor, vencimento, conta bancária, categoria e observações adicionais. Após o preenchimento das informações, o cadastro é concluído por meio do botão “Criar”, enquanto o botão “Cancelar” permite interromper a operação sem salvar os dados inseridos.

Os campos apresentados na tela, conforme a Figura 37, podem ser descritos de acordo com a Tabela 57:

Tabela 57 – Campos tela contas a receber

Item	Campo	Ação	Restrições/Observações
01	Descrição	Nome da conta	N/A
02	Cliente / Pagador	Nome do cliente que vai pagar a conta.	N/A
03	Valor	Valor a receber	N/A
04	Data de Vencimento	Adicionar data de vencimento	N/A
05	Conta bancaria	Selecionar conta que vai cair o dinheiro	N/A
06	Categorias	Categoria que vai receber o valor	N/A
07	Observação	Adicionar alguma informação útil	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 37 será descrito de acordo com a Tabela 58:

Tabela 58 – Botões da tela contas a receber

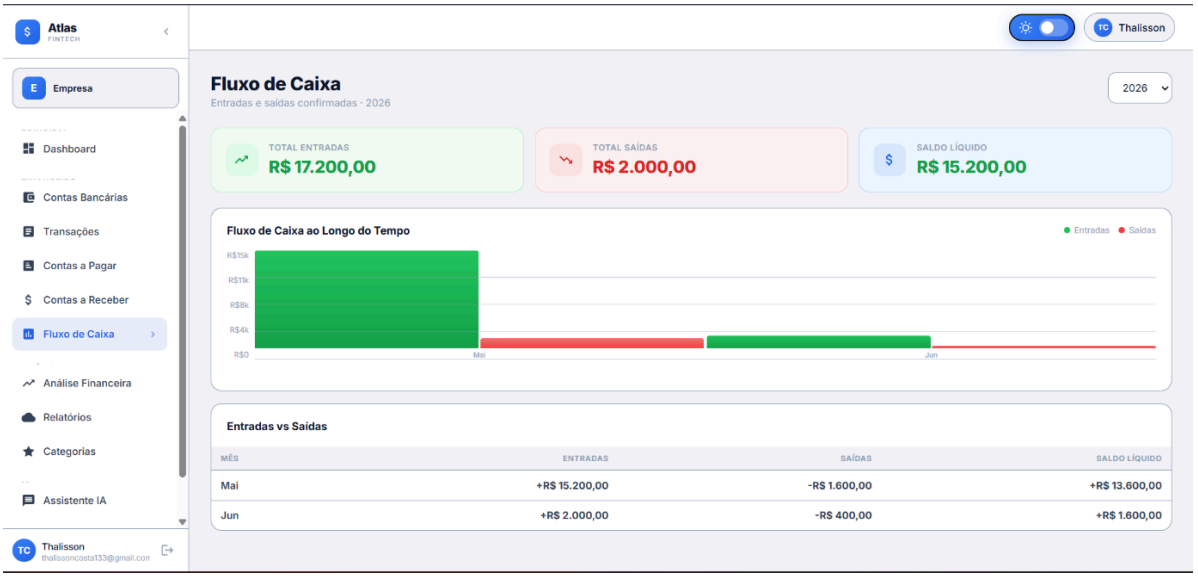
Item	Comando	Ação	Restrições/Observações
01	Fechar tela	Fechar a tela	N/A
02	Criar conta a receber	Criar uma nova conta a receber	N/A
03	Cancelar	Cancelar o registro de conta a receber	N/A

Fonte: Elaborado pelo autor (2026).

4.6.7 Tela de fluxo de caixa

A Figura 38 apresenta a tela de Fluxo de Caixa, permitindo que o usuário visualize informações e métricas financeiras relacionadas às entradas e saídas da empresa. A interface apresenta indicadores de total de entradas, total de saídas e saldo líquido, além de um gráfico de movimentação financeira ao longo do tempo e uma tabela detalhada dos registros. Essa visualização possibilita identificar de forma rápida a situação financeira da empresa.

Figura 38 – Tela de fluxo de caixa



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize de forma organizada as entradas, saídas e o saldo líquido da empresa, por meio de indicadores financeiros, gráficos e registros detalhados. Além disso, a tela oferece um filtro por período anual, contribuindo para um acompanhamento mais eficiente do fluxo de caixa e das finanças da empresa.

Os campos apresentados na tela, conforme a Figura 38, podem ser descritos de acordo com a Tabela 59:

Tabela 59 – Campos da tela de fluxo de caixa

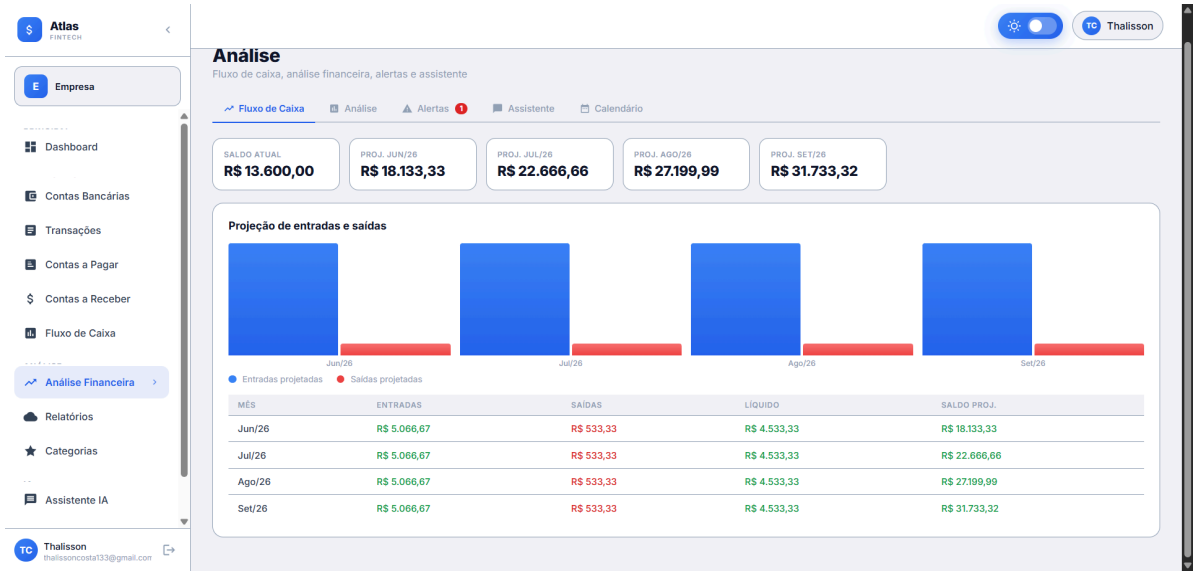
Item	Campo	Ação	Restrições/Observações
01	Filtros	Filtrar por ano.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.8 Tela de análise financeira

A Figura 39 apresenta a tela de Análise Financeira do sistema, permitindo ao usuário visualizar indicadores, métricas e gráficos relacionados ao desempenho financeiro da empresa. A interface apresenta informações como saldo atual, projeção de lucro, projeção de entradas, projeção de gastos e saldo estimado, além de um gráfico de projeção de entradas e saídas, possibilitando uma análise mais detalhada das movimentações financeiras.

Figura 39 – Tela de análise financeira



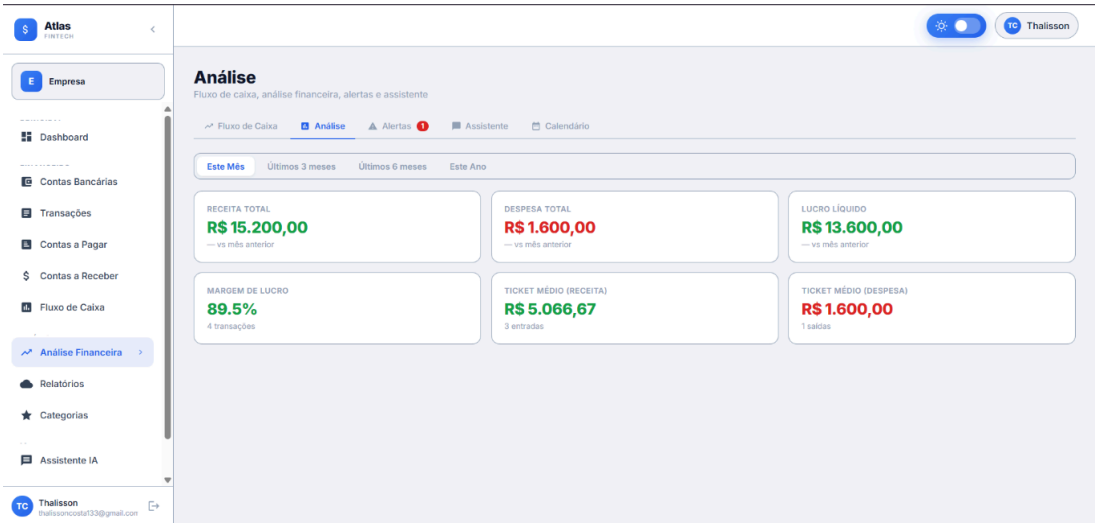
Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize de forma organizada os indicadores financeiros da empresa por meio de gráficos e projeções financeiras. Além disso, a tela apresenta informações sobre entradas, saídas, lucro e saldo projetado, contribuindo para um acompanhamento mais eficiente das finanças da organização.

4.6.8.1 Tela de análise

A Figura 40 apresenta a tela de Análise Financeira do sistema, permitindo ao usuário visualizar indicadores e métricas relacionadas ao desempenho financeiro da empresa. A interface disponibiliza informações como receita total, despesa total, lucro líquido, margem de lucro e ticket médio, além de filtros por período, possibilitando uma análise mais detalhada das movimentações financeiras.

Figura 40 – Tela de análise



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize de forma organizada indicadores financeiros da empresa, incluindo receitas, despesas, lucro líquido, margem de lucro e ticket médio. Além disso, a tela oferece opções de filtragem por período, contribuindo para um acompanhamento mais eficiente do desempenho financeiro da organização.

Os campos apresentados na tela, conforme a Figura 40, podem ser descritos de acordo com a Tabela 60:

Tabela 60 – Campos da tela de análise

Item	Campo	Ação	Restrições/Observações
01	Filtros	Filtrar por mês, 3 meses, 6 meses e este ano.	N/A

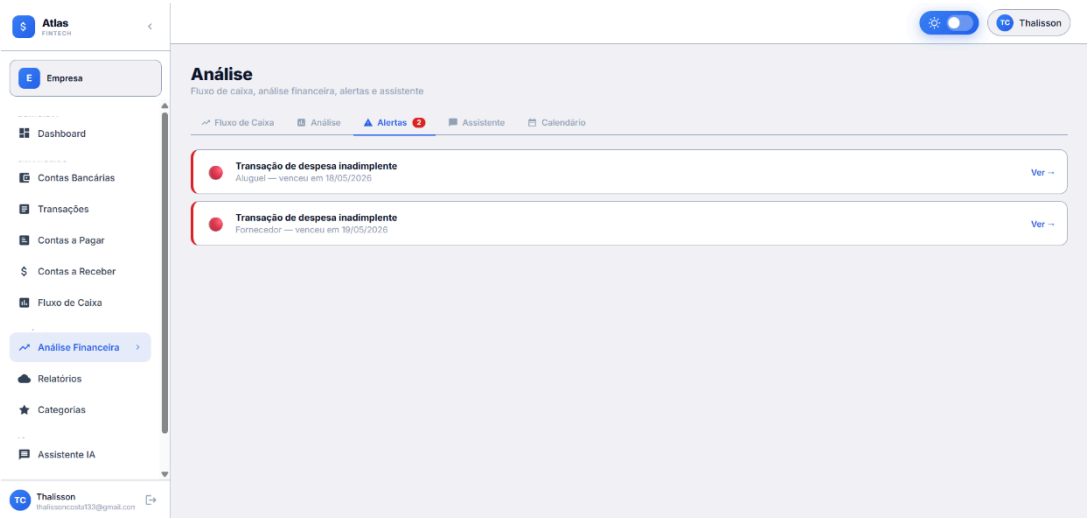
Fonte: Elaborado pelo autor (2026).

4.6.8.2 Tela de alertas

A Figura 41 apresenta a tela de Alertas da Análise Financeira do sistema, permitindo ao usuário visualizar notificações relacionadas às movimentações financeiras da empresa. A interface exibe alertas de despesas inadimplentes, apresentando informações como descrição da transação e data de vencimento,

possibilitando um acompanhamento mais eficiente das obrigações financeiras pendentes.

Figura 41 – Tela de alertas



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize de forma organizada alertas financeiros do sistema, especialmente relacionados a despesas inadimplentes. Além disso, a tela apresenta informações resumidas sobre as transações pendentes e oferece acesso rápido aos detalhes por meio da opção de visualização, contribuindo para um maior controle financeiro da empresa.

O comando da tela observado na Figura 41 será descrito de acordo com a Tabela 61:

Tabela 61 – Botões da tela de alertas

Item	Comando	Ação	Restrições/Observações
01	Link	Link que encaminha direto para as transações	N/A

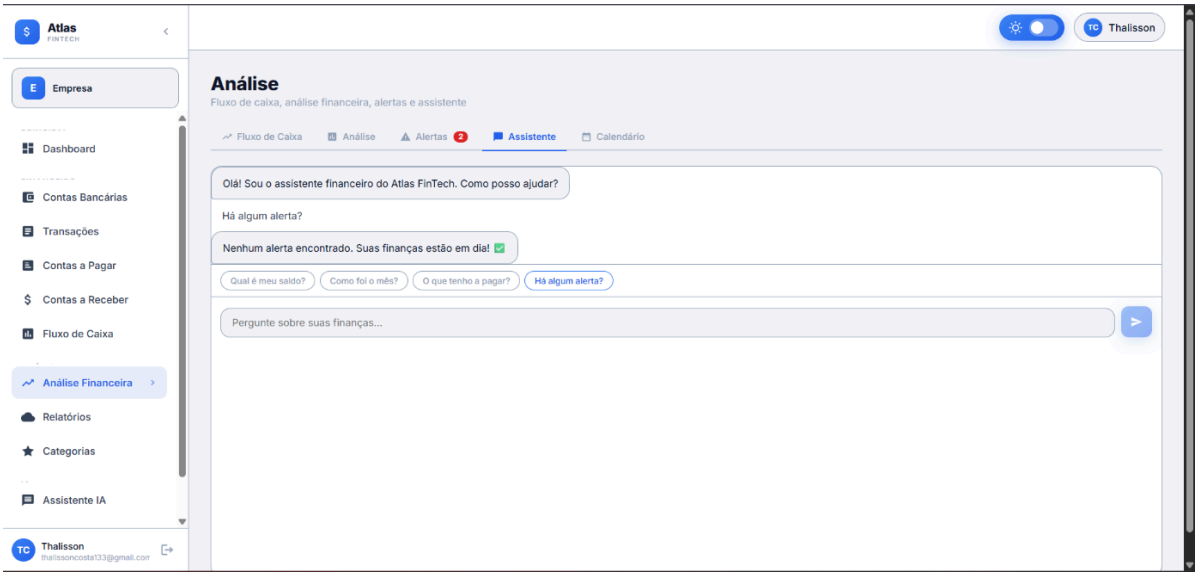
Fonte: Elaborado pelo autor (2026).

4.6.8.3 Tela do assistente

A Figura 42 apresenta a tela do Assistente Financeiro do sistema, integrada ao módulo de Análise Financeira, permitindo ao usuário interagir com uma inteligência artificial para obter informações sobre a situação financeira da empresa.

A interface disponibiliza sugestões de perguntas rápidas, consulta de alertas financeiros e um campo de interação para envio de mensagens, proporcionando maior praticidade no acompanhamento das finanças.

Figura 42 – Tela do assistente



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário consulte informações financeiras de forma interativa por meio do assistente virtual, possibilitando o acompanhamento de alertas, saldo financeiro e demais informações relacionadas às movimentações da empresa. Além disso, a tela oferece sugestões de perguntas rápidas e um campo de entrada para facilitar a comunicação com o sistema.

Os campos apresentados na tela, conforme a Figura 42, podem ser descritos de acordo com a Tabela 62:

Tabela 62 – Campos da tela de assistente

Item	Campo	Ação	Restrições/Observações
01	Busca	Digite a sua pergunta	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 42 será descrito de acordo com a Tabela 63:

Tabela 63 – Botões da tela assistente

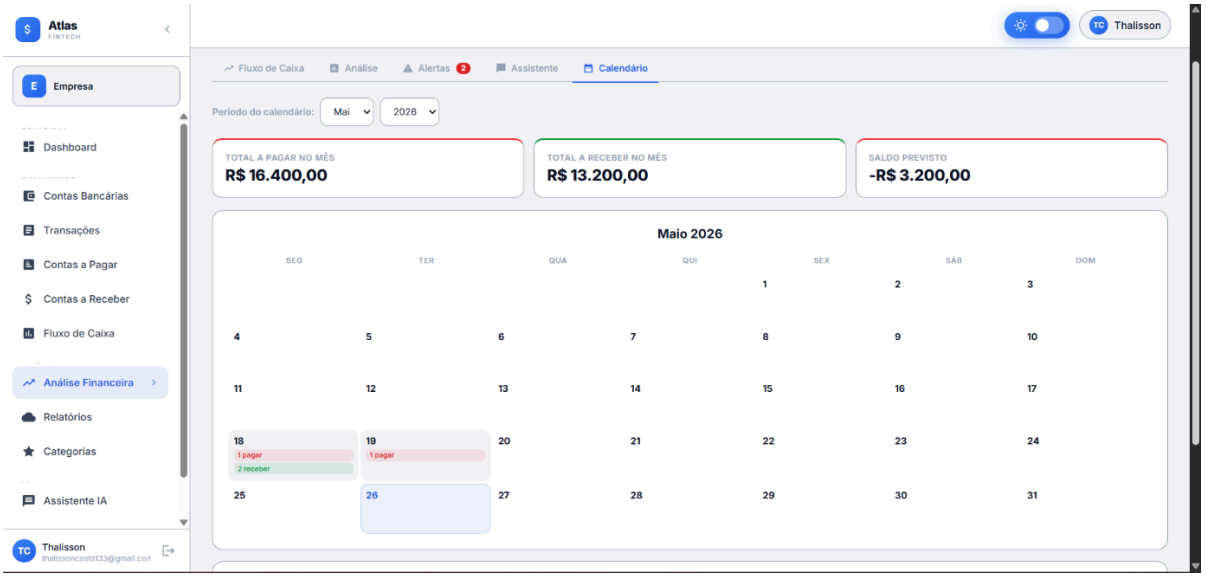
Item	Comando	Ação	Restrições/Observações
01	Qual é o meu saldo?	Saber o saldo atual	N/A
02	Como foi o mês?	Resultado do mês em tempo real	N/A
03	O que tenho a pagar?	Contas a pagar atual	N/A
04	Há algum alerta?	Mostra o que está em alerta	N/A
05	Botão de enviar pergunta	Envia a pergunta digitada.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.8.4 Tela de calendário

A Figura 43 apresenta a tela de Calendário do sistema, integrada ao módulo de Análise Financeira, permitindo ao usuário visualizar de forma organizada os compromissos financeiros previstos ao longo do mês. A interface exibe informações sobre valores a pagar, valores a receber e saldo previsto, além de um calendário mensal com a distribuição das movimentações financeiras, possibilitando um melhor acompanhamento das obrigações e recebimentos da empresa.

Figura 43 – Tela do calendário



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize de forma organizada as contas a pagar e a receber por meio de um calendário financeiro, facilitando o acompanhamento das movimentações previstas ao longo do mês. Além disso, a tela apresenta indicadores de total a pagar, total a receber e saldo previsto, contribuindo para um planejamento financeiro mais eficiente.

Os campos apresentados na tela, conforme a Figura 43, podem ser descritos de acordo com a Tabela 64:

Tabela 64 – Campos da tela calendário

Item	Campo	Ação	Restrições/Observações
01	Filtrar por mês	Selecione o mês que deseja ver.	N/A
02	Filtrar por ano	Selecionar o ano que deseja ver.	N/A

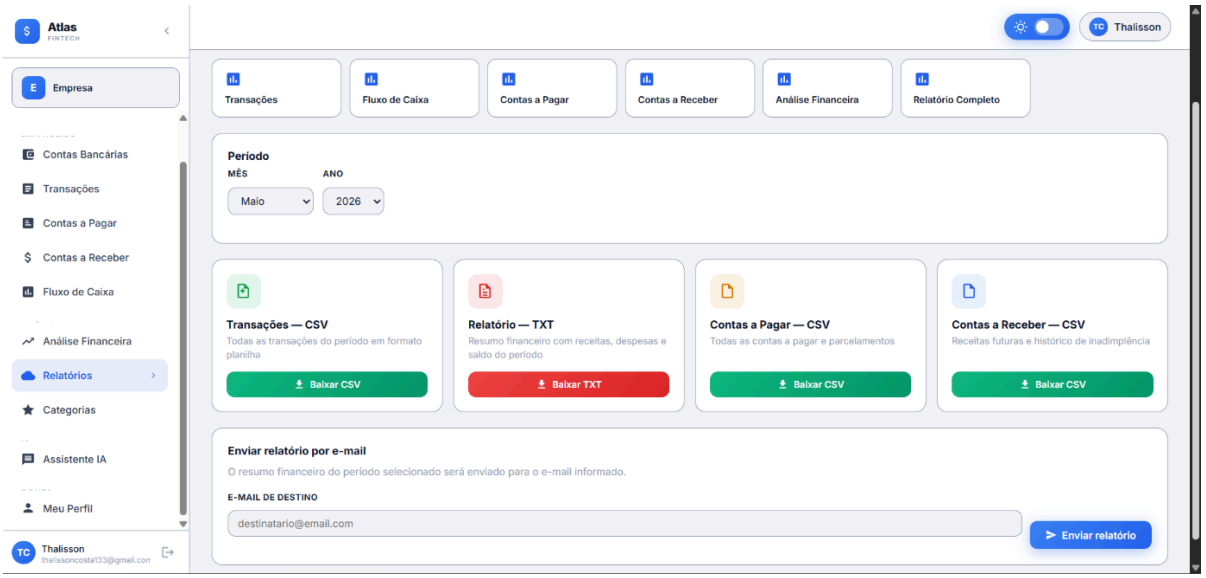
Fonte: Elaborado pelo autor (2026).

4.6.9 Tela de relatórios

A Figura 44 apresenta a tela de Relatórios do sistema, permitindo ao usuário gerar e exportar informações financeiras da empresa. A interface disponibiliza

diferentes tipos de relatórios, como transações, fluxo de caixa, contas a pagar, contas a receber e análise financeira, além de oferecer filtros por mês e ano para seleção do período desejado. Também é possível realizar o download dos relatórios em formatos específicos e enviá-los por e-mail.

Figura 44 – Tela de relatórios



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize e gere relatórios financeiros de forma organizada, possibilitando a exportação de informações relacionadas às transações, fluxo de caixa, contas a pagar e contas a receber. Além disso, a tela oferece filtros por período e a opção de envio dos relatórios por e-mail, contribuindo para um acompanhamento mais eficiente das finanças da empresa.

Os campos apresentados na tela, conforme a Figura 44, podem ser descritos de acordo com a Tabela 65:

Tabela 65 – Campos da tela relatórios

Item	Campo	Ação	Restrições/Observações
01	Filtrar por mês	Selecione o mês que deseja ver.	N/A
02	Filtrar por ano	Selecionar o ano que deseja ver.	N/A

03	E-mail de destino	Digitar e-mail para envio de relatório.	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 44 será descrito de acordo com a Tabela 66:

Tabela 66 – Comandos da tela relatórios

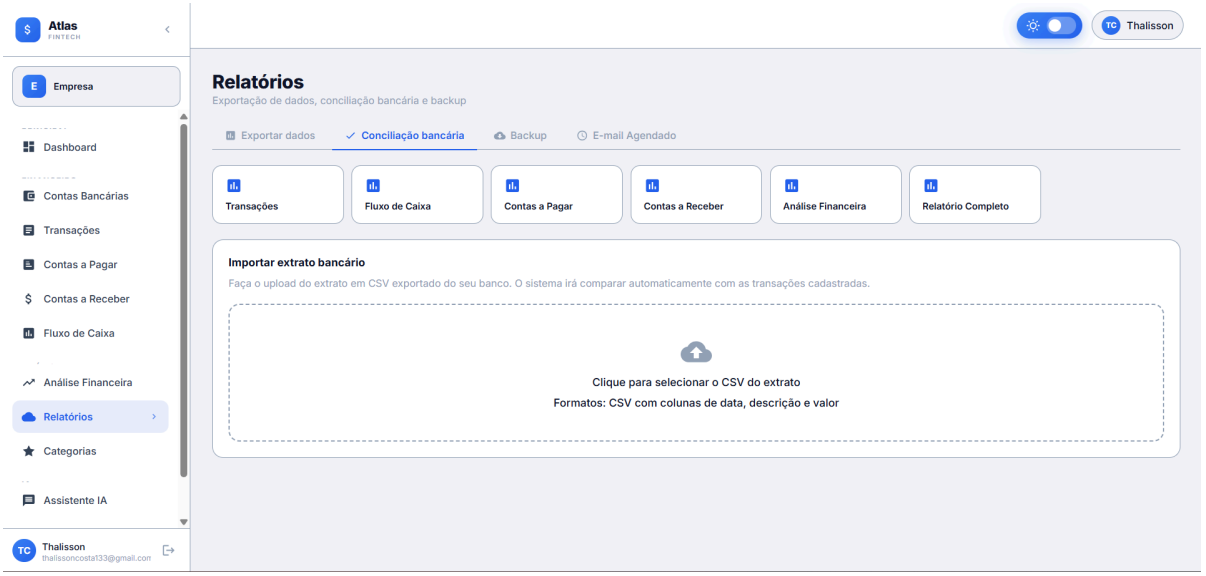
Item	Comando	Ação	Restrições/Observações
01	Transações	Baixar o relatório em CSV/Excel	O relatório será baixado e pode ser utilizado no Excel.
02	Fluxo de caixa	Baixar o relatório em CSV/Excel	O relatório será baixado e pode ser utilizado no Excel.
03	Contas a Pagar	Baixar o relatório em CSV/Excel	O relatório será baixado e pode ser utilizado no Excel.
04	Contas a Receber	Baixar o relatório em CSV/Excel	O relatório será baixado e pode ser utilizado no Excel.
05	Análise Financeira	Relatório será baixado em TXT.	N/A
06	Relatórios Completo	Baixar o relatório em ZIP	O relatório será baixado em forma de backup em zip.
07	Transações – CSV	Baixar o relatório em CSV/Excel	N/A
08	Relatórios – TXT	Relatório será baixado em TXT.	N/A
09	Contas a Pagar - CSV	Baixar o relatório em CSV/Excel	N/A
10	Contas a Receber – CSV	Baixar o relatório em CSV/Excel	N/A
11	Botão enviar relatório	Botão que envia e-mail.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.9.1 Tela de conciliação bancaria

A Figura 45 apresenta a tela de Conciliação Bancária do módulo de Relatórios, permitindo ao usuário importar extratos bancários para comparação automática com as transações cadastradas no sistema. A interface possibilita o envio de arquivos no formato CSV, facilitando a identificação e conferência das movimentações financeiras registradas.

Figura 45 – Tela de conciliação bancária



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário realize a importação de extratos bancários por meio de arquivos CSV, possibilitando a comparação automática entre os dados do banco e as transações cadastradas no sistema. Essa funcionalidade contribui para um controle financeiro mais organizado, preciso e eficiente.

Os campos apresentados na tela, conforme a Figura 45, podem ser descritos de acordo com a Tabela 67:

Tabela 67 – Comandos da tela conciliação bancária

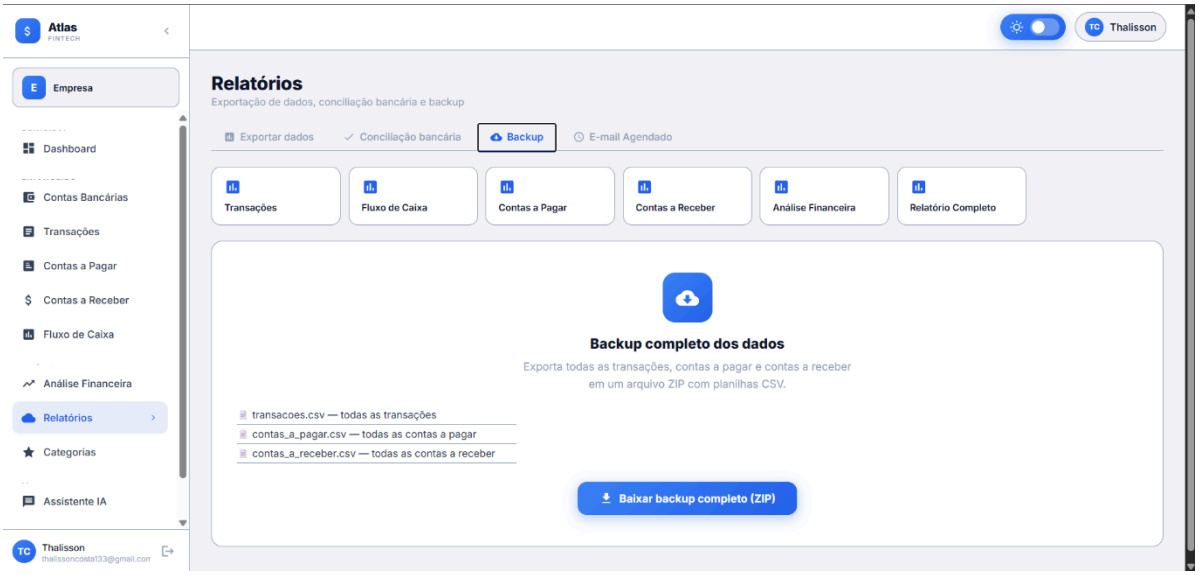
Item	Campo	Ação	Restrições/Observações
01	Importar extrato bancário	Faça upload do extrato em csv do seu banco.	O sistema irá comparar automaticamente com as transações cadastradas.

Fonte: Elaborado pelo autor (2026).

4.6.9.2 Tela de backup

A Figura 46 apresenta a tela de Backup do módulo de Relatórios, permitindo ao usuário realizar a exportação completa dos dados cadastrados no sistema. A interface possibilita a geração de um arquivo compactado no formato ZIP, contendo informações relacionadas às transações, contas a pagar e contas a receber, facilitando a segurança, armazenamento e recuperação dos dados.

Figura 46 – Tela de backup



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário realize o backup completo das informações do sistema por meio de um arquivo compactado no formato ZIP. Essa funcionalidade contribui para a preservação dos dados financeiros, proporcionando maior segurança e facilidade no armazenamento e recuperação das informações da empresa.

O comando apresentados na tela, conforme a Figura 46, podem ser descritos de acordo com a Tabela 68:

Tabela 68 – Comandos da tela backup

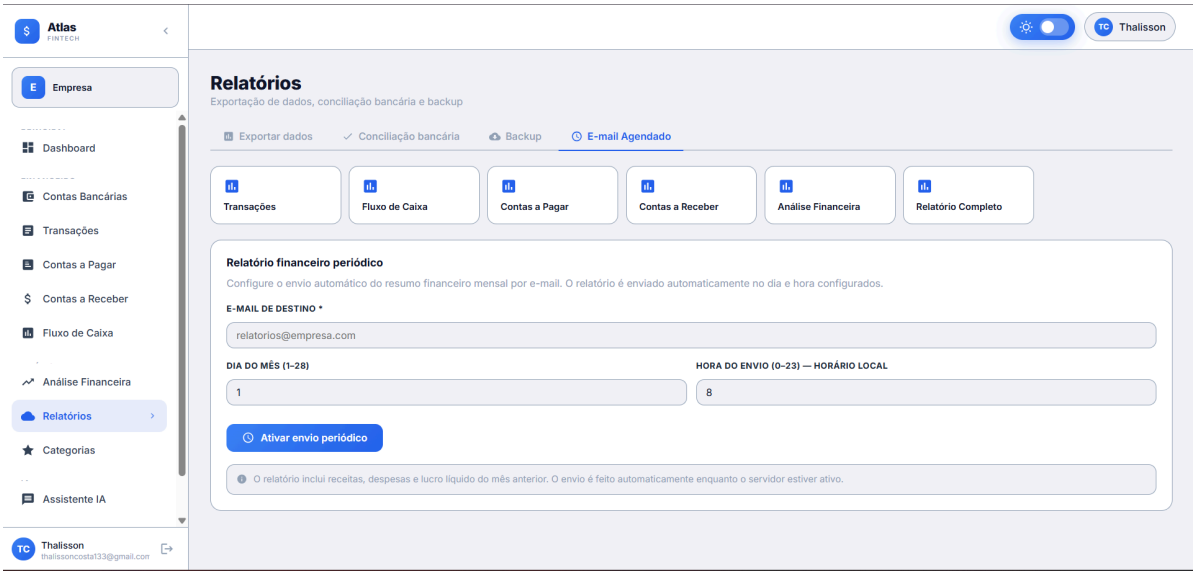
Item	Comando	Ação	Restrições/Observações
01	Botão backup completo (zip)	Exportar em um arquivo em zip, com as planilhas em csv.	Transações, contas a pagar e contas a receber estão dentro do zip.

Fonte: Elaborado pelo autor (2026).

4.6.9.3 Tela de e-mail agendado

A Figura 47 apresenta a tela de E-mail Agendado do módulo de Relatórios, permitindo ao usuário configurar o envio automático de relatórios financeiros periódicos. A interface possibilita definir o e-mail de destino, o dia do mês para envio e o horário programado, automatizando o compartilhamento das informações financeiras da empresa.

Figura 47 – Tela de e-mail agendado



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário configure o envio periódico de relatórios financeiros por e-mail, definindo informações como destinatário, dia do envio e horário programado. Essa funcionalidade contribui para um

acompanhamento financeiro mais prático e automatizado, facilitando o acesso às informações da empresa.

Os campos apresentados na tela, conforme a Figura 47, podem ser descritos de acordo com a Tabela 69:

Tabela 69 – Campos da tela e-mail agendado

Item	Campo	Ação	Restrições/Observações
01	E-mail de destino	Digitar e-mail que vai receber relatório.	N/A
02	Filtrar por dia	Selecionar o dia que deseja receber o relatório.	N/A
03	Filtrar por hora	Selecionar a hora que deseja receber o relatório.	N/A

Fonte: Elaborado pelo autor (2026).

Os comandos apresentados na tela, conforme a Figura 47, podem ser descritos de acordo com a Tabela 70:

Tabela 70 – Comandos da tela e-mail agendado

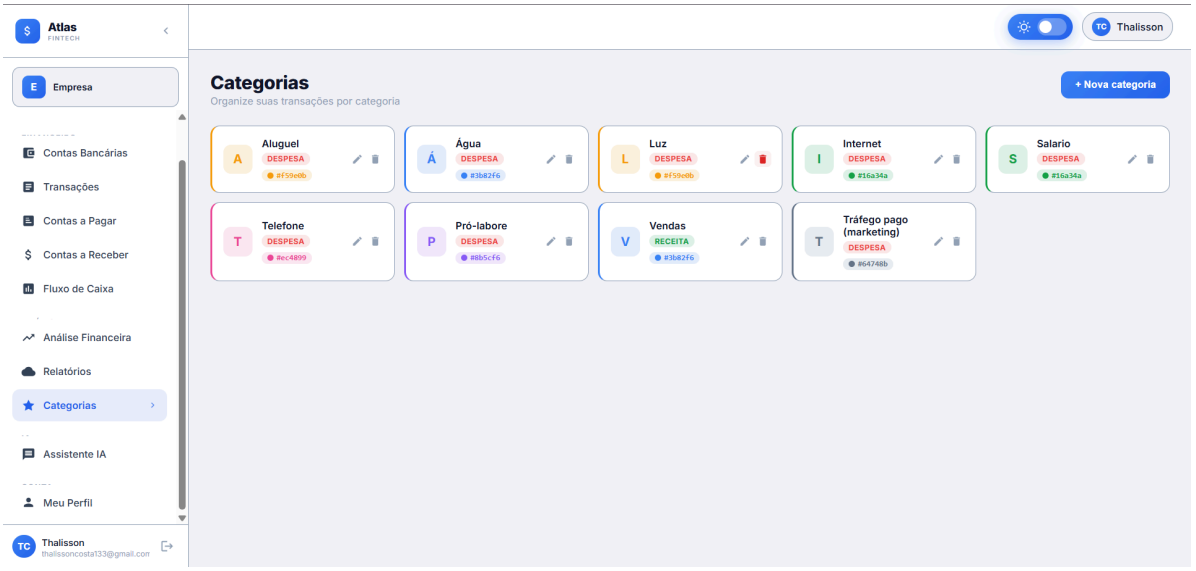
Item	Comando	Ação	Restrições/Observações
01	Botão ativar envio periódico	Enviar e-mail de forma agendada.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.10 Tela de categorias

A Figura 48 apresenta a tela de Categorias do sistema, permitindo ao usuário visualizar e gerenciar as categorias utilizadas nas movimentações financeiras. A interface organiza as categorias de receitas e despesas, exibindo informações como nome, tipo e quantidade de registros vinculados, além de disponibilizar a opção de cadastro de novas categorias e edição dos registros existentes.

Figura 48 – Tela de categorias



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize e organize as categorias financeiras cadastradas no sistema, diferenciando receitas e despesas de forma estruturada. Além disso, a tela possibilita o cadastro, edição e gerenciamento das categorias, contribuindo para uma organização mais eficiente das movimentações financeiras da empresa.

O comando da tela observado na Figura 48 será descrito de acordo com a Tabela 71:

Tabela 71 – Botões da tela de categorias

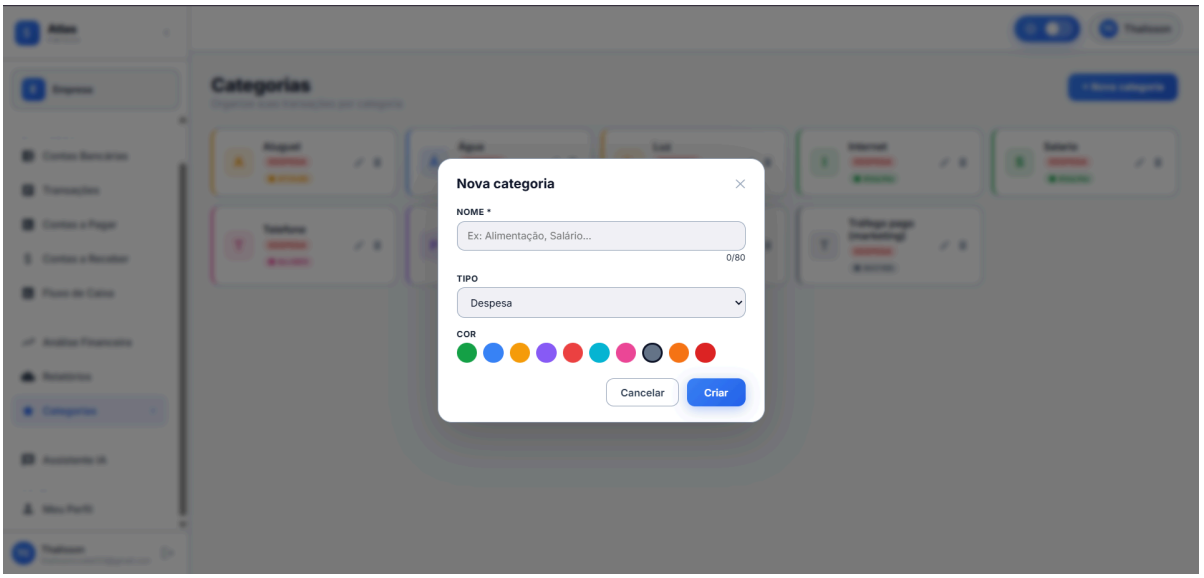
Item	Comando	Ação	Restrições/Observações
01	Nova categoria	Registrar uma nova categoria	N/A
02	Editar	Editar categoria	N/A
03	Excluir	Excluir categoria	N/A

Fonte: Elaborado pelo autor (2026).

4.6.10.1 Tela de registro de categoria

A Figura 49 apresenta a tela de cadastro de categorias do sistema, utilizada para o registro de novas categorias financeiras. Nessa interface, o usuário pode definir o nome da categoria, selecionar o tipo da movimentação, como receita ou despesa, e escolher uma cor de identificação, permitindo uma melhor organização e personalização das transações financeiras.

Figura 49 – Tela de registro de categorias



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário registre uma nova categoria financeira, informando dados como nome, tipo e cor de identificação. Após o preenchimento das informações, o cadastro pode ser concluído por meio do botão “Criar”, enquanto a opção “Cancelar” permite interromper a operação sem salvar os dados inseridos.

Os campos apresentados na tela, conforme a Figura 49, podem ser descritos de acordo com a Tabela 72:

Tabela 72 – Campos de registro de categorias

Item	Campo	Ação	Restrições/Observações
01	Nome categoria	Digite o nome da categoria	N/A

02	Tipo da categoria	Selecionar o tipo por despesa, receita ou ambos.	N/A
03	Cor da categoria	Selecionar cor da categoria.	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 49 será descrito de acordo com a Tabela 73:

Tabela 73 – Botões da tela registro de categorias

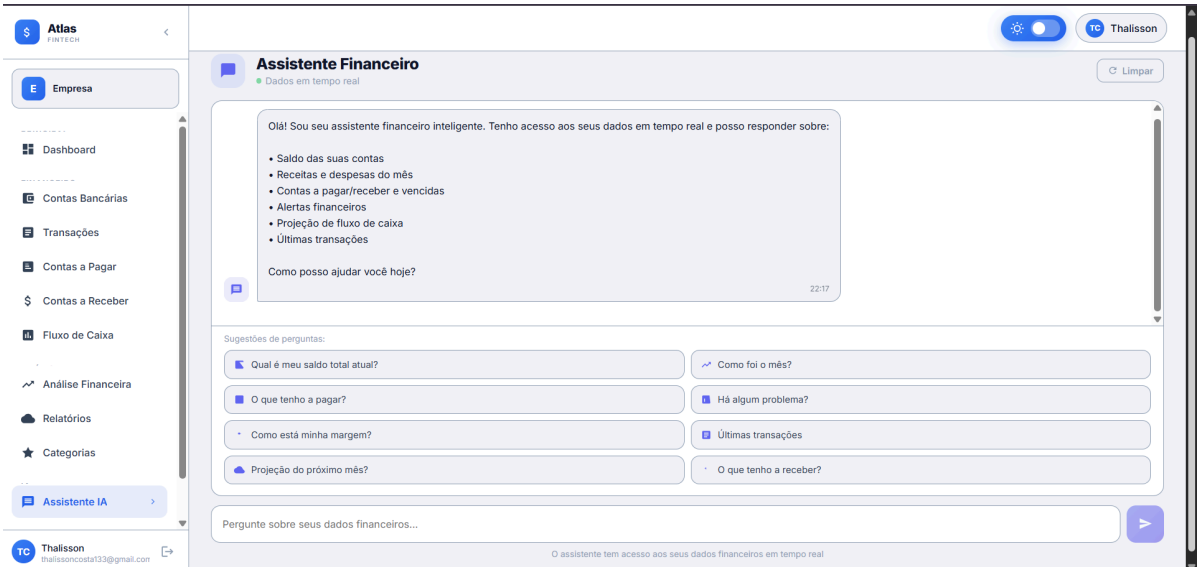
Item	Comando	Ação	Restrições/Observações
01	Fechar tela	Fechar tela de categoria	N/A
02	Criar categoria	Criar uma categoria	N/A
03	Cancelar categoria	Cancelar a criação de categoria.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.11 Tela de assistente financeiro

A Figura 50 apresenta a tela do Assistente Financeiro do sistema, permitindo ao usuário interagir com uma inteligência artificial para obter informações relacionadas à situação financeira da empresa. A interface possibilita consultas sobre saldo, receitas, despesas, contas a pagar e a receber, projeções financeiras e movimentações recentes, oferecendo sugestões de perguntas rápidas para facilitar a navegação e o acesso às informações.

Figura 50 – Tela de assistente financeiro



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário consulte informações financeiras de forma interativa por meio do assistente virtual, utilizando perguntas personalizadas ou sugestões rápidas disponibilizadas pelo sistema. Essa funcionalidade contribui para um acompanhamento mais prático, rápido e organizado das finanças da empresa.

Os campos apresentados na tela, conforme a Figura 50, podem ser descritos de acordo com a Tabela 74:

Tabela 74 – Campos de assistente financeiro

Item	Campo	Ação	Restrições/Observações
01	Digitar pergunta	Digitar uma pergunta sobre as suas finanças.	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 50 será descrito de acordo com a Tabela 75:

Tabela 75 – Botões da tela assistente financeiro

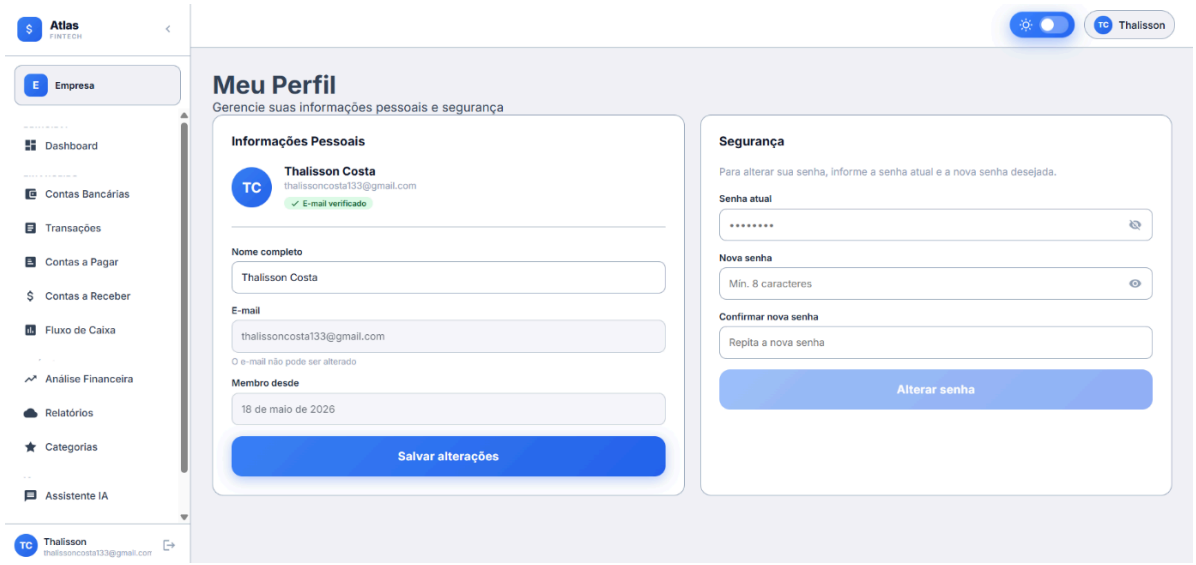
Item	Comando	Ação	Restrições/Observações
01	Limpar	Limpar conversa do chat.	N/A
02	Qual é meu saldo total atual?	Saldo da conta	N/A
03	O que tenho a pagar?	Contas a pagar	N/A
04	Como está a minha margem?	Margem de lucro	N/A
05	Projeção do próximo mês?	Projeção baseada nos últimos 3 meses	N/A
06	Como foi o mês?	Desenho do mês.	N/A
07	Há algum problema?	Alerta de contas não pagas ou recebidas.	N/A
08	Últimas transações	Últimas transações	N/A
09	O que tenho a receber?	Contas a receber	N/A
10	Botão de enviar pergunta	Enviar pergunta do usuário.	N/A

Fonte: Elaborado pelo autor (2026).

4.6.12 Tela de perfil do usuário

A Figura 51 apresenta a tela de Meu Perfil do sistema, permitindo ao usuário visualizar e gerenciar suas informações pessoais e configurações de segurança. A interface disponibiliza dados como nome, e-mail e data de criação da conta, além de oferecer funcionalidades para atualização das informações cadastrais e alteração da senha de acesso.

Figura 51 – Tela de perfil do usuário



Fonte: Elaborado pelo autor (2026).

A interface exibida acima permite que o usuário visualize e atualize suas informações pessoais, como nome e endereço de e-mail, além de realizar alterações de segurança por meio da redefinição da senha de acesso. Essa funcionalidade contribui para uma gestão mais segura e personalizada da conta do usuário no sistema.

Os campos apresentados na tela, conforme a Figura 51, podem ser descritos de acordo com a Tabela 76:

Tabela 76 – Campos de perfil do usuário

Item	Campo	Ação	Restrições/Observações
01	Nome completo	Nome completo do usuário	N/A
02	Senha atual	Digite a senha atual	N/A
03	Nova senha	Digite a sua nova senha	N/A
04	Confirmar nova senha	Digite a confirmação da senha	N/A

Fonte: Elaborado pelo autor (2026).

O comando da tela observado na Figura 51 será descrito de acordo com a Tabela 77:

Tabela 77 – Botões da tela perfil do usuário

Item	Comando	Ação	Restrições/Observações
01	Salvar alterações	Salvar alterações feita no nome do usuário	N/A
02	Alterar senha	Salvar troca de senha.	N/A

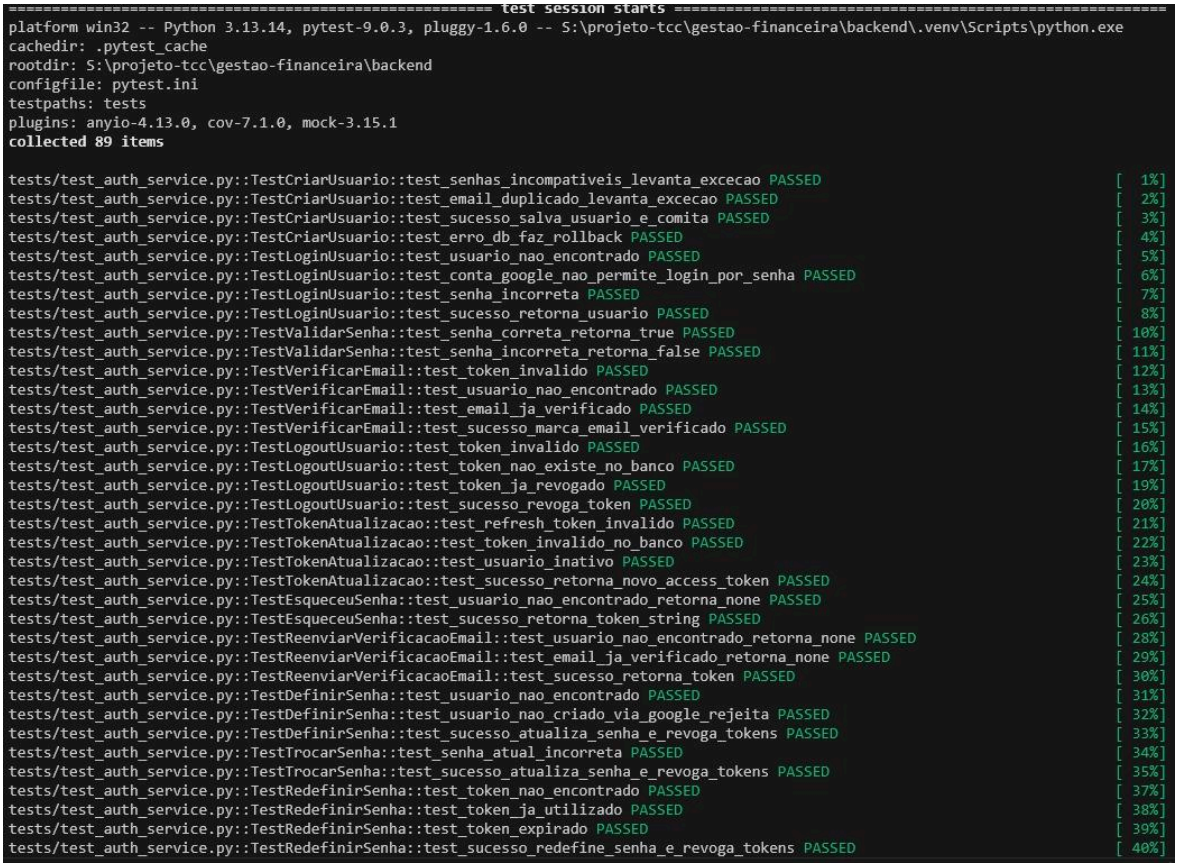
Fonte: Elaborado pelo autor (2026).

4.7 Evidências de testes frontend e backend

Com o objetivo de verificar a qualidade e o correto funcionamento do sistema desenvolvido, foram realizados testes no front-end e no back-end da aplicação. No front-end, foram avaliadas as interfaces, validações dos formulários e a apresentação das informações ao usuário. No back-end, foram testadas as regras de negócio, o processamento das requisições e a integração com o banco de dados PostgreSQL.

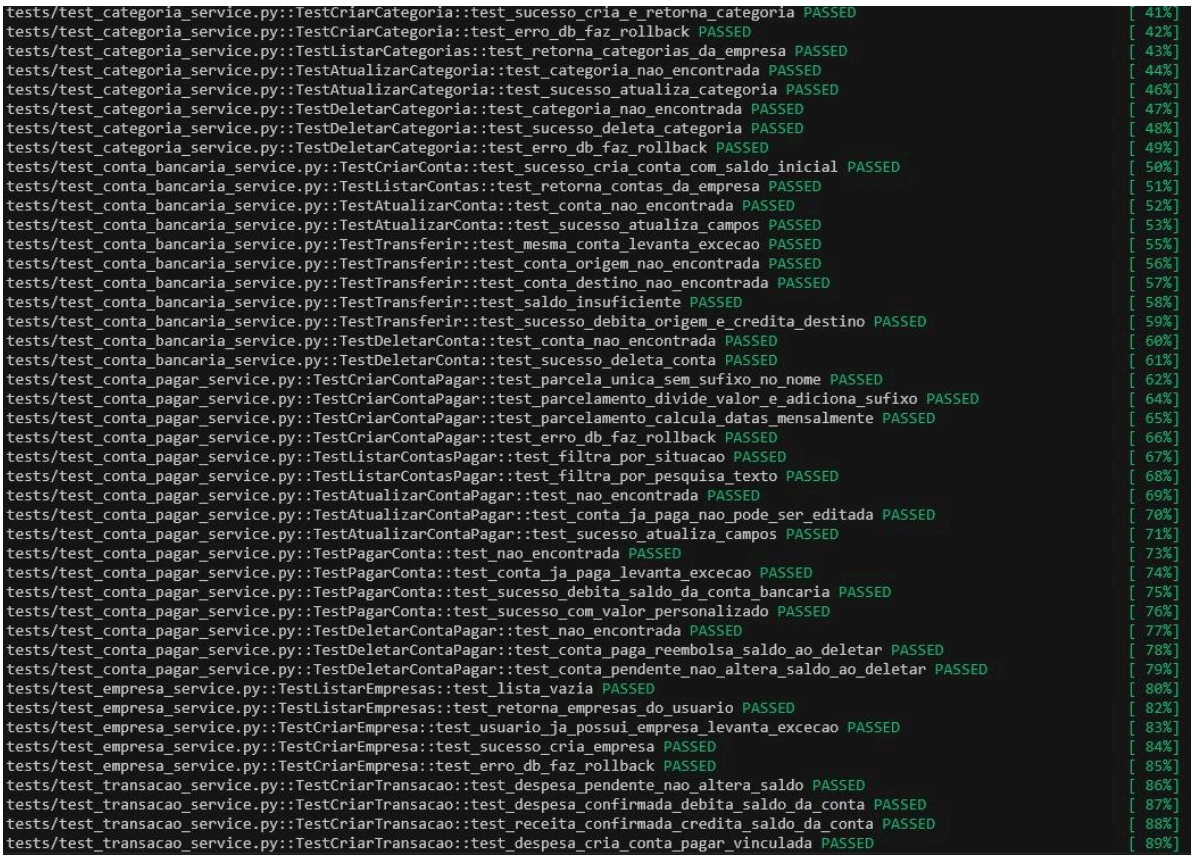
Os resultados obtidos demonstraram que as funcionalidades implementadas operam de forma adequada, atendendo aos requisitos estabelecidos para o sistema. As evidências apresentadas nesta seção, por meio de registros e capturas de tela, comprovam a execução bem-sucedida dos testes e a estabilidade da aplicação.

Figura 52 – Resultado testes unitários back-end

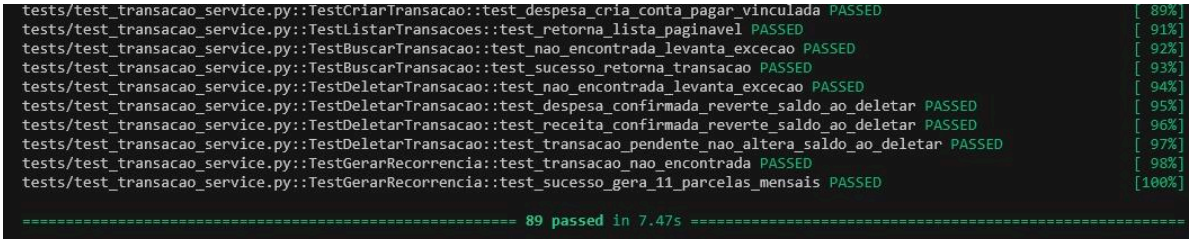


```
============================= test session starts =============================
platform win32 -- Python 3.13.14, pytest-9.0.3, pluggy-1.6.0 -- S:\projeto-tcc\gestao-financeira\backend\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: S:\projeto-tcc\gestao-financeira\backend
configfile: pytest.ini
testpaths: tests
plugins: anyio-4.13.0, cov-7.1.0, mock-3.15.1
collected 89 items

tests/test_auth_service.py::TestCriarUsuario::test_senhas_incompativeis_levanta_excecao PASSED [  1%]
tests/test_auth_service.py::TestCriarUsuario::test_email_duplicado_levanta_excecao PASSED [  2%]
tests/test_auth_service.py::TestCriarUsuario::test_sucesso_salva_usuario_e_comita PASSED [  3%]
tests/test_auth_service.py::TestCriarUsuario::test_erro_db_faz_rollback PASSED [  4%]
tests/test_auth_service.py::TestLoginUsuario::test_usuario_nao_encontrado PASSED [  5%]
tests/test_auth_service.py::TestLoginUsuario::test_conta_google_nao_permite_login_por_senha PASSED [  6%]
tests/test_auth_service.py::TestLoginUsuario::test_senha_incorreta PASSED [  7%]
tests/test_auth_service.py::TestLoginUsuario::test_sucesso_retorna_usuario PASSED [  8%]
tests/test_auth_service.py::TestValidarSenha::test_senha_correta_retorna_true PASSED [ 10%]
tests/test_auth_service.py::TestValidarSenha::test_senha_incorreta_retorna_false PASSED [ 11%]
tests/test_auth_service.py::TestVerificarEmail::test_token_invalido PASSED [ 12%]
tests/test_auth_service.py::TestVerificarEmail::test_usuario_nao_encontrado PASSED [ 13%]
tests/test_auth_service.py::TestVerificarEmail::test_email_ja_verificado PASSED [ 14%]
tests/test_auth_service.py::TestVerificarEmail::test_sucesso_marca_email_verificado PASSED [ 15%]
tests/test_auth_service.py::TestLogoutUsuario::test_token_invalido PASSED [ 16%]
tests/test_auth_service.py::TestLogoutUsuario::test_token_nao_existe_no_banco PASSED [ 17%]
tests/test_auth_service.py::TestLogoutUsuario::test_token_ja_revogado PASSED [ 19%]
tests/test_auth_service.py::TestLogoutUsuario::test_sucesso_revoga_token PASSED [ 20%]
tests/test_auth_service.py::TestTokenAtualizacao::test_refresh_token_invalido PASSED [ 21%]
tests/test_auth_service.py::TestTokenAtualizacao::test_token_invalido_no_banco PASSED [ 22%]
tests/test_auth_service.py::TestTokenAtualizacao::test_usuario_inativo PASSED [ 23%]
tests/test_auth_service.py::TestTokenAtualizacao::test_sucesso_retorna_novo_access_token PASSED [ 24%]
tests/test_auth_service.py::TestEsqueceuSenha::test_usuario_nao_encontrado_retorna_none PASSED [ 25%]
tests/test_auth_service.py::TestEsqueceuSenha::test_sucesso_retorna_token_string PASSED [ 26%]
tests/test_auth_service.py::TestReenviarVerificacaoEmail::test_usuario_nao_encontrado_retorna_none PASSED [ 28%]
tests/test_auth_service.py::TestReenviarVerificacaoEmail::test_email_ja_verificado_retorna_none PASSED [ 29%]
tests/test_auth_service.py::TestReenviarVerificacaoEmail::test_sucesso_retorna_token PASSED [ 30%]
tests/test_auth_service.py::TestDefinirSenha::test_usuario_nao_encontrado PASSED [ 31%]
tests/test_auth_service.py::TestDefinirSenha::test_usuario_nao_criado_via_google_rejeita PASSED [ 32%]
tests/test_auth_service.py::TestDefinirSenha::test_sucesso_atualiza_senha_e_revoga_tokens PASSED [ 33%]
tests/test_auth_service.py::TestTrocarSenha::test_senha_atual_incorreta PASSED [ 34%]
tests/test_auth_service.py::TestTrocarSenha::test_sucesso_atualiza_senha_e_revoga_tokens PASSED [ 35%]
tests/test_auth_service.py::TestRedefinirSenha::test_token_nao_encontrado PASSED [ 37%]
tests/test_auth_service.py::TestRedefinirSenha::test_token_ja_utilizado PASSED [ 38%]
tests/test_auth_service.py::TestRedefinirSenha::test_token_expirado PASSED [ 39%]
tests/test_auth_service.py::TestRedefinirSenha::test_sucesso_redefine_senha_e_revoga_tokens PASSED [ 40%]
```



```
tests/test_categoria_service.py::TestCriarCategoria::test_sucesso_cria_e_retorna_categoria PASSED [ 41%]
tests/test_categoria_service.py::TestCriarCategoria::test_erro_db_faz_rollback PASSED [ 42%]
tests/test_categoria_service.py::TestListarCategorias::test_retorna_categorias_da_empresa PASSED [ 43%]
tests/test_categoria_service.py::TestAtualizarCategoria::test_categoria_nao_encontrada PASSED [ 44%]
tests/test_categoria_service.py::TestAtualizarCategoria::test_sucesso_atualiza_categoria PASSED [ 46%]
tests/test_categoria_service.py::TestDeletarCategoria::test_categoria_nao_encontrada PASSED [ 47%]
tests/test_categoria_service.py::TestDeletarCategoria::test_sucesso_deleta_categoria PASSED [ 48%]
tests/test_categoria_service.py::TestDeletarCategoria::test_erro_db_faz_rollback PASSED [ 49%]
tests/test_conta_bancaria_service.py::TestCriarConta::test_sucesso_cria_conta_com_saldo_inicial PASSED [ 50%]
tests/test_conta_bancaria_service.py::TestListarContas::test_retorna_contas_da_empresa PASSED [ 51%]
tests/test_conta_bancaria_service.py::TestAtualizarConta::test_conta_nao_encontrada PASSED [ 52%]
tests/test_conta_bancaria_service.py::TestAtualizarConta::test_sucesso_atualiza_campos PASSED [ 53%]
tests/test_conta_bancaria_service.py::TestTransferir::test_mesma_conta_levanta_excecao PASSED [ 55%]
tests/test_conta_bancaria_service.py::TestTransferir::test_conta_origem_nao_encontrada PASSED [ 56%]
tests/test_conta_bancaria_service.py::TestTransferir::test_conta_destino_nao_encontrada PASSED [ 57%]
tests/test_conta_bancaria_service.py::TestTransferir::test_saldo_insuficiente PASSED [ 58%]
tests/test_conta_bancaria_service.py::TestTransferir::test_sucesso_debita_origem_e_credita_destino PASSED [ 59%]
tests/test_conta_bancaria_service.py::TestDeletarConta::test_conta_nao_encontrada PASSED [ 60%]
tests/test_conta_bancaria_service.py::TestDeletarConta::test_sucesso_deleta_conta PASSED [ 61%]
tests/test_conta_pagar_service.py::TestCriarContaPagar::test_parcela_unica_sem_sufixo_no_nome PASSED [ 62%]
tests/test_conta_pagar_service.py::TestCriarContaPagar::test_parcelamento_divide_valor_e_adiciona_sufixo PASSED [ 64%]
tests/test_conta_pagar_service.py::TestCriarContaPagar::test_parcelamento_calcula_datas_mensalmente PASSED [ 65%]
tests/test_conta_pagar_service.py::TestCriarContaPagar::test_erro_db_faz_rollback PASSED [ 66%]
tests/test_conta_pagar_service.py::TestListarContasPagar::test_filtra_por_situacao PASSED [ 67%]
tests/test_conta_pagar_service.py::TestListarContasPagar::test_filtra_por_pesquisa_texto PASSED [ 68%]
tests/test_conta_pagar_service.py::TestAtualizarContaPagar::test_nao_encontrada PASSED [ 69%]
tests/test_conta_pagar_service.py::TestAtualizarContaPagar::test_conta_ja_paga_nao_pode_ser_editada PASSED [ 70%]
tests/test_conta_pagar_service.py::TestAtualizarContaPagar::test_sucesso_atualiza_campos PASSED [ 71%]
tests/test_conta_pagar_service.py::TestPagarConta::test_nao_encontrada PASSED [ 73%]
tests/test_conta_pagar_service.py::TestPagarConta::test_conta_ja_paga_levanta_excecao PASSED [ 74%]
tests/test_conta_pagar_service.py::TestPagarConta::test_sucesso_debita_saldo_da_conta_bancaria PASSED [ 75%]
tests/test_conta_pagar_service.py::TestPagarConta::test_sucesso_com_valor_personalizado PASSED [ 76%]
tests/test_conta_pagar_service.py::TestDeletarContaPagar::test_nao_encontrada PASSED [ 77%]
tests/test_conta_pagar_service.py::TestDeletarContaPagar::test_conta_paga_reembolsa_saldo_ao_deletar PASSED [ 78%]
tests/test_conta_pagar_service.py::TestDeletarContaPagar::test_conta_pendente_nao_altera_saldo_ao_deletar PASSED [ 79%]
tests/test_empresa_service.py::TestListarEmpresas::test_lista_vazia PASSED [ 80%]
tests/test_empresa_service.py::TestListarEmpresas::test_retorna_empresas_do_usuario PASSED [ 82%]
tests/test_empresa_service.py::TestCriarEmpresa::test_usuario_ja_possui_empresa_levanta_excecao PASSED [ 83%]
tests/test_empresa_service.py::TestCriarEmpresa::test_sucesso_cria_empresa PASSED [ 84%]
tests/test_empresa_service.py::TestCriarEmpresa::test_erro_db_faz_rollback PASSED [ 85%]
tests/test_transacao_service.py::TestCriarTransacao::test_despesa_pendente_nao_altera_saldo PASSED [ 86%]
tests/test_transacao_service.py::TestCriarTransacao::test_despesa_confirmada_debita_saldo_da_conta PASSED [ 87%]
tests/test_transacao_service.py::TestCriarTransacao::test_receita_confirmada_credita_saldo_da_conta PASSED [ 88%]
tests/test_transacao_service.py::TestCriarTransacao::test_despesa_cria_conta_pagar_vinculada PASSED [ 89%]
```

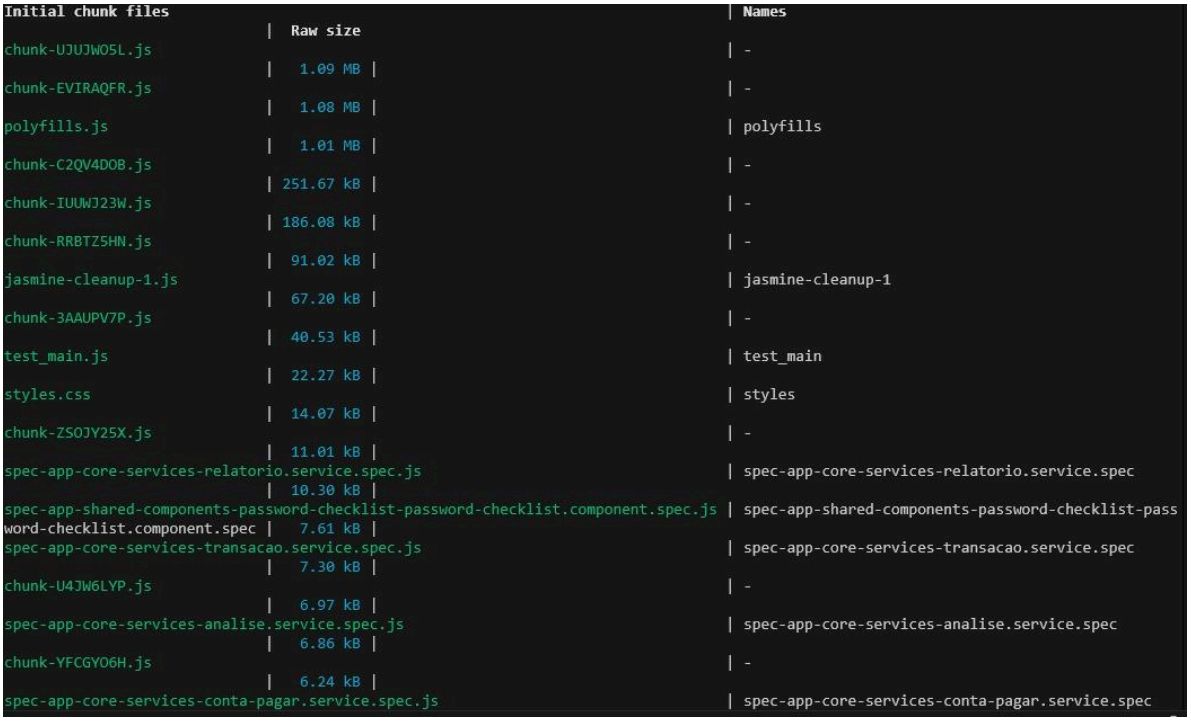


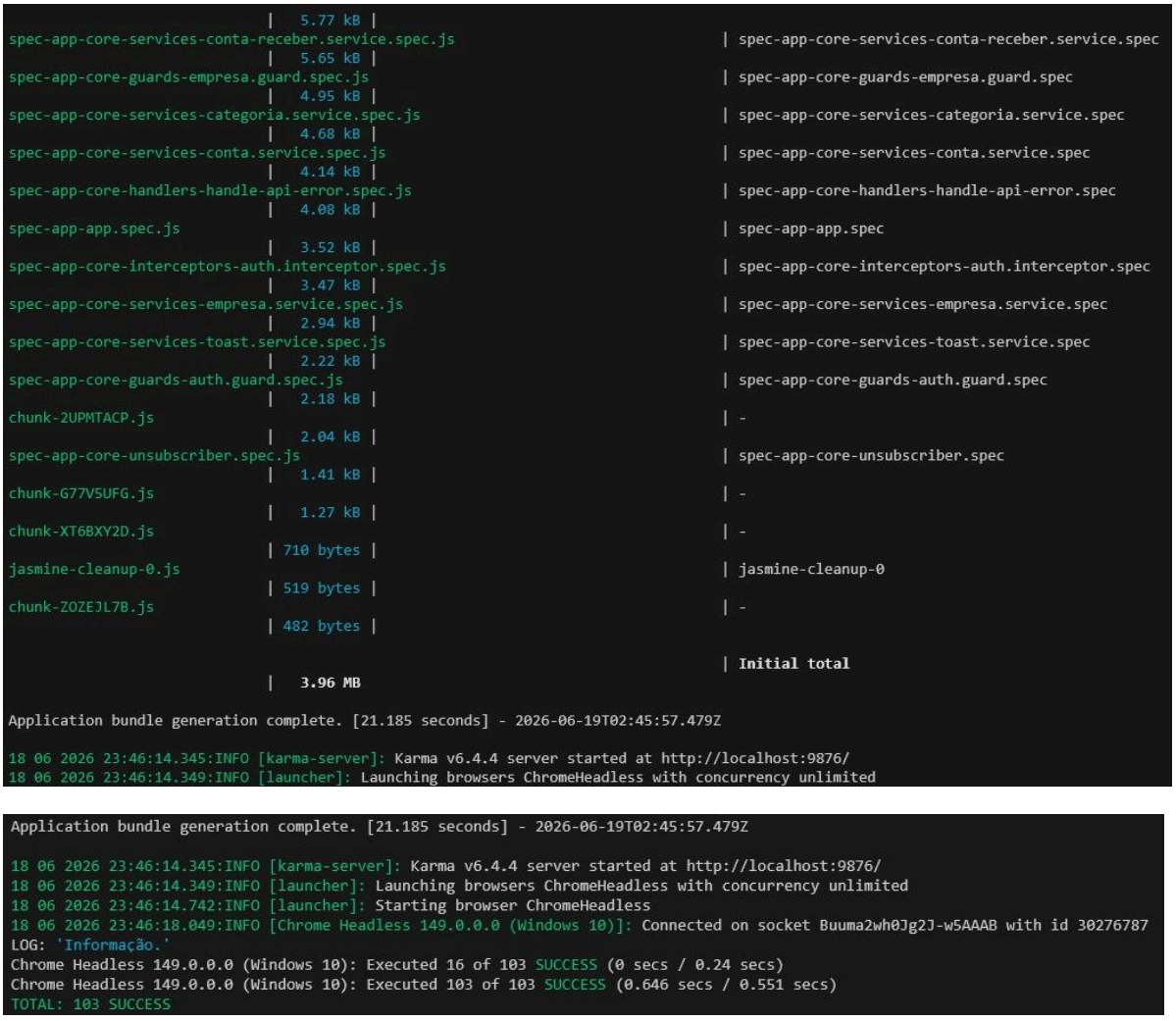
Fonte: Elaborado pelo autor (2026).

A Figura 52 apresenta a execução completa da suíte de testes automatizados do *back-end* do sistema, executada por meio do *framework Pytest*. A suíte é composta por 89 casos de teste, todos aprovados em uma única execução de 7,47 segundos, demonstrando a estabilidade e a eficiência do conjunto de testes implementado.

A organização dos testes seguiu a estrutura da arquitetura em camadas adotada no projeto, com um arquivo de teste correspondente para cada *service* implementado.

Figura 53 – Resultado testes unitários front-end





Fonte: Elaborado pelo autor (2026).

A Figura 53 apresenta a execução completa da suíte de testes automatizados do *front-end* do sistema, executada por meio do *Karma* 6.4.4 com o *framework Jasmine*, utilizando o navegador *ChromeHeadless* em modo de execução não interativa. A suíte é composta por 103 casos de teste, todos aprovados em uma única execução, demonstrando a cobertura dos principais pontos críticos da camada de apresentação.

A organização dos testes seguiu a estrutura modular do *Angular*, mantendo arquivos com sufixo *.spec.ts* próximos aos arquivos testados. Essa convenção, recomendada pela documentação oficial do *framework*, favorece a manutenibilidade e facilita a localização dos testes pelo desenvolvedor.

5 CONCLUSÃO

O desenvolvimento do Atlas FinTech demonstrou que soluções tecnológicas simples, acessíveis e bem estruturadas podem contribuir de maneira significativa para a organização financeira de pequenas e médias empresas. A análise inicial do cenário brasileiro evidenciou que grande parte das MPMEs ainda utiliza métodos manuais ou ferramentas limitadas para controlar receitas, despesas e fluxo de caixa, o que compromete a precisão dos dados e dificulta a tomada de decisões estratégicas. Nesse contexto, o sistema proposto mostrou-se uma alternativa viável para reduzir esses problemas e apoiar a profissionalização da gestão financeira.

A construção do projeto baseou-se em fundamentos sólidos da engenharia de software, englobando levantamento de requisitos, modelagem e implementação. O uso de tecnologias como *Angular*, *TypeScript*, *Python* com *FastAPI* e *PostgreSQL* permitiu desenvolver uma solução moderna, escalável e alinhada às boas práticas de desenvolvimento. Além disso, a adoção de princípios de usabilidade e experiência do usuário contribuiu para uma interface intuitiva e acessível, reduzindo a complexidade de uso para empreendedores com pouco conhecimento técnico.

Os resultados obtidos demonstraram que o sistema atende aos requisitos definidos, oferecendo um conjunto abrangente de funcionalidades financeiras, entre as quais se destacam o registro e a categorização de transações, o controle de contas a pagar e a receber, o *dashboard* interativo com indicadores consolidados, o fluxo de caixa, a análise financeira com projeções, a conciliação bancária por importação de arquivos, o agendamento automático de relatórios por e-mail, o calendário financeiro e o assistente virtual integrado. Esses recursos reforçaram o diferencial competitivo da ferramenta em relação a sistemas já consolidados no mercado.

Em relação aos objetivos específicos definidos neste trabalho, todos foram atendidos: as principais dificuldades das MPMEs na gestão financeira foram analisadas e serviram de base para a definição dos requisitos; os requisitos funcionais e não funcionais foram especificados e documentados; a arquitetura da aplicação foi modelada com foco em usabilidade e acessibilidade; as funcionalidades de controle financeiro e relatórios estratégicos foram implementadas; e a validação foi realizada por meio de testes unitários nas camadas de *frontend* e *backend*, verificando o correto funcionamento das principais rotinas do

sistema. Dessa forma, o Atlas FinTech consolida-se como uma alternativa eficiente e aderente às necessidades de empreendedores que buscam soluções práticas e confiáveis.

Por fim, recomenda-se, como trabalhos futuros, a expansão das funcionalidades, incluindo integração bancária direta, módulos de análise preditiva por meio de inteligência artificial, aplicações *mobile*, integração do assistente virtual com modelos de linguagem de grande escala para respostas mais precisas e contextualizadas, e o desenvolvimento de um módulo de precificação de produtos, permitindo que o usuário calcule e gerencie os preços de seus produtos com base nos custos operacionais registrados no sistema. Essas melhorias podem ampliar ainda mais a capacidade da ferramenta e fortalecer seu papel como aliada estratégica na gestão financeira empresarial.

REFERÊNCIAS

ALEMBIC. *Welcome to Alembic's documentation!* [S. l.]: SQLAlchemy, 2024. Disponível em: https://alembic.sqlalchemy.org/. Acesso em: 24 maio 2026.

APSCHEDULER. *APScheduler: Advanced Python Scheduler documentation.* 2024. Disponível em: https://apscheduler.readthedocs.io/. Acesso em: 25 nov. 2025.

ATLASSIAN. *Jira Software: documentação oficial.* Sydney: Atlassian, 2024. Disponível em: https://www.atlassian.com/software/jira. Acesso em: 25 maio 2026.

BECK, K. *Test-driven development: by example.* Boston: Addison-Wesley, 2003.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. *UML: guia do usuário*. 2. ed. Rio de Janeiro: Elsevier, 2006.

CHUNG, Lawrence; NIXON, Brian; YU, Eric. *Non-functional requirements in software engineering.* Cham: Springer, 2022.

CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM,* v. 13, n. 6, p. 377-387, jun. 1970. DOI: 10.1145/362384.362685.

COHN, M. *Succeeding with agile: software development using Scrum.* Boston: Addison-Wesley, 2009.

CONTA AZUL. *Controle financeiro empresarial – Conta Azul.* 2025. Disponível em: https://contaazul.com/. Acesso em: 9 nov. 2025.

FASTAPI. *FastAPI: framework web moderno e de alta performance para construção de APIs com Python.* 2024. Disponível em: https://fastapi.tiangolo.com/. Acesso em: 11 out. 2025.

FOWLER, Martin. *UML essencial*: um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2004

FOWLER, M. Mocks aren't stubs. *martinfowler.com,* 2007. Disponível em: https://martinfowler.com/articles/mocksArentStubs.html. Acesso em: 30 maio 2026.

GESTÃOCLICK. *GestãoClick: o seu software de gestão empresarial.* 2024. Disponível em: https://www.gestaoclick.com.br/. Acesso em: 29 out. 2025.

GITHUB. *GitHub Docs: documentação oficial.* San Francisco: GitHub, 2024. Disponível em: https://docs.github.com. Acesso em: 25 maio 2026.

GOOGLE. *Angular: aplicações web modernas.* 2024. Disponível em: https://angular.dev/. Acesso em: 11 out. 2025.

HÄRDER, T.; REUTER, A. Principles of transaction-oriented database recovery. *ACM Computing Surveys,* v. 15, n. 4, p. 287-317, dez. 1983. DOI: 10.1145/289.291.

HARDT, D. (Ed.). *The OAuth 2.0 Authorization Framework.* RFC 6749. Internet Engineering Task Force, out. 2012. Disponível em: https://www.rfc-editor.org/rfc/rfc6749. Acesso em: 25 nov. 2025.

HUB ASIMOV. *O que é Python e para que serve?* 2025. Disponível em: https://hub.asimov.academy/blog/python-o-que-e/. Acesso em: 23 nov. 2025.

IBM. *What is three-tier architecture?* Armonk: IBM, 2025. Disponível em: https://www.ibm.com/think/topics/three-tier-architecture. Acesso em: 24 maio 2026.

JASMINE. *Jasmine: a behavior-driven JavaScript testing framework.* 2024. Disponível em: https://jasmine.github.io/. Acesso em: 23 nov. 2025.

JONES, M.; BRADLEY, J.; SAKIMURA, N. *JSON Web Token (JWT).* RFC 7519. Internet Engineering Task Force, maio 2015. Disponível em: https://www.rfc-editor.org/rfc/rfc7519. Acesso em: 25 nov. 2025.

KARMA. *Karma: spectacular test runner for JavaScript.* 2024. Disponível em: https://karma-runner.github.io/. Acesso em: 23 nov. 2025.

LARMAN, Craig. *Utilizando UML e padrões*: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

LAUDON, K. C.; LAUDON, J. P. *Sistemas de informação gerenciais.* 17. ed. São Paulo: Pearson, 2022.

LEITE, C. M.; FIGUEIREDO, P. S.; LOPES, S. P. M.; PASSOS, F. U. Conceituando e medindo a transformação digital: proposta de um modelo de mensuração. *Cadernos EBAPE.BR,* Rio de Janeiro, v. 22, n. 5, e2023-0081, 2024. DOI: 10.1590/1679-395120230081. Disponível em: https://www.scielo.br/j/cebape/a/R649VyDgsJqpMBnxq9HxXSr/. Acesso em: 23 nov. 2025.

LOPES, L.; SOUZA, G. J. R.; ALVES, A. F.; NUNES, R. B. Metodologias ágeis: explorando o impacto do Scrum e do Kanban na qualidade e produtividade do software. *Revista Multidisciplinar do Nordeste Mineiro,* v. 12, n. 2, 2024. DOI: 10.61164/rmnm.v12i2.3060. Disponível em:

https://remunom.ojsbr.com/multidisciplinar/article/view/3060. Acesso em: 23 nov. 2025.

MDN WEB DOCS. *Documentação para desenvolvedores web.* Mozilla, 2025. Disponível em: https://developer.mozilla.org/pt-BR/. Acesso em: 23 nov. 2025.

MICROSOFT. *TypeScript: JavaScript with syntax for types.* 2024. Disponível em: https://www.typescriptlang.org/. Acesso em: 11 out. 2025.

MICROSOFT. *Visual Studio Code: documentação oficial.* Redmond: Microsoft, 2024. Disponível em: https://code.visualstudio.com/docs. Acesso em: 25 maio 2026.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The art of software testing.* 3. ed. Hoboken: Wiley, 2011.

OLIVEIRA, P. M. F.; DIAS, S. B. A.; MENDES, A. V.; ROCHA, W. P. *Inteligência artificial aplicada à gestão financeira.* 2024. Trabalho de Conclusão de Curso (Graduação em Administração) – Pontifícia Universidade Católica de Goiás, Uberlândia, 2024.

OMG. *OMG Unified Modeling Language (OMG UML), version 2.5.1.* Needham: Object Management Group, 2017. Disponível em: https://www.omg.org/spec/UML/2.5.1/. Acesso em: 23 nov. 2025.

PASSLIB. *Passlib: comprehensive password hashing framework for Python.* 2024. Disponível em: https://passlib.readthedocs.io/. Acesso em: 25 nov. 2025.

PIRES, S. P. Um estudo sobre a gestão financeira em micro e pequenas empresas na Quarta Colônia. *Saber Humano,* v. 1, n. 1, p. 394-421, 2024. Disponível em: https://saberhumano.emnuvens.com.br/sh/article/view/685. Acesso em: 29 out. 2025.

POSTGRESQL. *PostgreSQL: the world's most advanced open source relational database.* PostgreSQL Global Development Group, 2025. Disponível em: https://www.postgresql.org/. Acesso em: 25 nov. 2025.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional.* 9. ed. Porto Alegre: AMGH, 2021.

PROVOS, N.; MAZIÈRES, D. A future-adaptable password scheme. In: USENIX ANNUAL TECHNICAL CONFERENCE, 1999, Monterey. *Proceedings...* Monterey: USENIX Association, 1999. p. 81-91. Disponível em: https://www.usenix.org/legacy/events/usenix99/provos.html. Acesso em: 25 nov. 2025.

PYDANTIC. *Welcome to Pydantic.* [S. l.]: Pydantic, 2024. Disponível em: https://docs.pydantic.dev/latest/. Acesso em: 24 maio 2026.

PYTEST. *Pytest: helps you write better programs.* 2024. Disponível em: https://docs.pytest.org/. Acesso em: 25 nov. 2025.

PYTHON SOFTWARE FOUNDATION. *Python documentation.* 2024. Disponível em: https://docs.python.org/. Acesso em: 23 nov. 2025.

RENDER. *Render Documentation: official documentation.* San Francisco: Render Services, 2026. Disponível em: https://render.com/docs. Acesso em: 26 maio 2026.

RESEND. *Resend: email API for developers.* 2024. Disponível em: https://resend.com/docs. Acesso em: 25 nov. 2025.

RXJS. *RxJS: reactive extensions library for JavaScript.* 2024. Disponível em: https://rxjs.dev/. Acesso em: 23 nov. 2025.

SASS. *Sass: syntactically awesome style sheets.* 2025. Disponível em: https://sass-lang.com/. Acesso em: 23 nov. 2025.

SEBRAE. *Sobrevivência das empresas no Brasil.* Brasília: SEBRAE, 2022. Disponível em: https://sebrae.com.br/sites/PortalSebrae/artigos/a-taxa-de-sobrevivencia-das-empresas-no-brasil. Acesso em: 8 out. 2025.

SEBRAE. *Transformação digital nas micro e pequenas empresas brasileiras.* Brasília: SEBRAE, 2023. Disponível em: https://www.sebrae.com.br. Acesso em: 30 out. 2025.

SOMMERVILLE, I. *Engenharia de software.* 11. ed. São Paulo: Pearson, 2021.

SQLALCHEMY. *SQLAlchemy Documentation.* [S. l.]: SQLAlchemy, 2024. Disponível em: https://docs.sqlalchemy.org/en/20/. Acesso em: 24 maio 2026.

SUPABASE. *Supabase Documentation: official documentation.* San Francisco: Supabase Inc., 2026. Disponível em: https://supabase.com/docs. Acesso em: 26 maio 2026.

UOL ECONOMIA. Pesquisa mostra que 39% das MPMEs ainda controlam despesas de forma manual. *UOL Economia,* São Paulo, 20 jan. 2025. Disponível em: https://economia.uol.com.br/noticias/estadao-conteudo/2025/01/20/pesquisa-mostra-que-39-das-mpmes-ainda-controlam-despesas-de-forma-manual.html. Acesso em: 8 out. 2025.

VALENTE, M. T. *Engenharia de software moderna: princípios e práticas para desenvolvimento de software com produtividade.* Belo Horizonte: UFMG, 2020. Disponível em: https://engsoftmoderna.info/. Acesso em: 24 maio 2026.

VERCEL. *Vercel: the frontend cloud platform.* 2025. Disponível em: https://vercel.com/. Acesso em: 23 nov. 2025.

W3C. *HTML Living Standard.* World Wide Web Consortium, 2025. Disponível em: https://html.spec.whatwg.org/. Acesso em: 23 nov. 2025.

XFIN. *Sistema XFIN de gestão financeira.* 2025. Disponível em: https://www.sistemaxfin.com.br/. Acesso em: 9 nov. 2025.

YOURDON, Edward. *Análise estruturada moderna*. Rio de Janeiro: Campus, 1990.