Protótipo de Plataforma Intuitiva para Organização Financeira Pessoal*

Prototype of an Intuitive Platform for Personal Financial Organization

Emanuel Nogueira Moreira da Costa[1]
Pedro Henrique Ribeiro da Silva[2]
Adriane Maria Arantes de Carvalho[3]

Resumo

O crescente uso de tecnologias digitais no cotidiano tem ampliado a necessidade de ferramentas que auxiliem no controle e na organização das finanças pessoais, contribuindo para uma tomada de decisão mais consciente. Este trabalho apresenta o desenvolvimento de uma plataforma intuitiva voltada à gestão de finanças pessoais, com foco no controle e na organização do fluxo de caixa, caracterizando-se como uma pesquisa aplicada, de abordagem qualitativa e caráter descritivo, realizada por meio do desenvolvimento de um protótipo de software funcional. A solução foi desenvolvida utilizando tecnologias como Next.js, GraphQL, Node.js, TypeScript e MongoDB, resultando em uma aplicação responsiva, projetada para permitir o acesso a partir de diferentes dispositivos, especialmente dispositivos móveis, mantendo boa usabilidade, flexibilidade e dinamismo, uma vez que possibilita ao usuário consultar e acompanhar suas informações financeiras a qualquer momento e em qualquer lugar por meio de smartphones, sem prejuízo da experiência de uso. Os resultados demonstram que a aplicação atendeu aos objetivos propostos ao oferecer um ambiente organizado, acessível e funcional, fortalecendo a autonomia do usuário no processo de tomada de decisões financeiras. A plataforma possibilita o registro detalhado de entradas e saídas financeiras, categorizadas de acordo com sua natureza, além de disponibilizar gráficos interativos com filtros, que favorecem a análise dinâmica dos dados, a identificação de padrões de consumo e o acompanhamento da evolução financeira ao longo do tempo, contribuindo tanto para o meio acadêmico quanto para práticas voltadas à educação financeira.

Palavras-chave: Gestão financeira pessoal. Educação financeira. Visualização de dados. Next.js. GraphQL.

*Trabalho de conclusão de curso, Sistemas de Informação, Unidade São Gabriel

[1]PUC Minas – Sistemas de Informação, Brasil– emanuel.costa.1402021@sga.pucminas.br
[2]PUC Minas – Sistemas de Informação, Brasil– pedro.silva.1388147@sga.pucminas.br
[3]Professora do Departamento de Administração da PUC Minas, Brasil– adriane@pucminas.br

Abstract

The increasing use of digital technologies in everyday life has intensified the need for tools that assist in the control and organization of personal finances, contributing to more informed decision-making. This work presents the development of an intuitive platform aimed at personal financial management, focusing on cash flow control and organization, and is characterized as applied research with a qualitative approach and a descriptive nature, conducted through the development of a functional software prototype. The solution was developed using technologies such as Next.js, GraphQL, Node.js, TypeScript, and MongoDB, resulting in a responsive application designed to allow access from different devices, especially mobile devices, while maintaining good usability, flexibility, and dynamism, as it enables users to consult and monitor their financial information at any time and from any location through smartphones, without compromising the user experience. The results demonstrate that the application met the proposed objectives by providing an organized, accessible, and functional environment, strengthening user autonomy in the financial decision-making process. The platform enables the detailed recording of financial inflows and outflows, categorized according to their nature, and also provides interactive charts with filters that support dynamic data analysis, the identification of consumption patterns, and the monitoring of financial evolution over time, contributing both to the academic field and to practices related to financial education.

Keywords: Personal finance management. Financial literacy. Data visualization. Next.js. GraphQL.

1 INTRODUÇÃO

O cenário econômico brasileiro, marcado pelo crescente endividamento e pelo limitado planejamento orçamentário, especialmente entre famílias de baixa renda, reflete uma realidade complexa. De acordo com Fund (2024), a baixa literacia financeira está associada ao uso de empréstimos informais como alternativa de crédito, e o acesso facilitado ao crédito pode aumentar o risco de dificuldades financeiras entre indivíduos menos instruídos.

Nesse contexto, a adoção de ferramentas digitais para o gerenciamento financeiro tem sido amplamente explorada na literatura. Silva, Coelho e Silva (2020) destacam que os aplicativos de gestão financeira atuam como facilitadores no controle e planejamento das finanças pessoais, sobretudo em cenários de exclusão digital e baixo comprometimento com o controle de gastos. Os autores identificam funcionalidades como registro de receitas e despesas, categorização, fluxo de caixa e gráficos interativos, que contribuem para o combate ao endividamento e para a promoção da educação financeira.

A eficácia dessas tecnologias também está relacionada à facilidade de uso percebida por diferentes perfis de usuários, uma vez que fatores como idade e renda influenciam a avaliação de custo, risco e utilidade dos *apps* de pagamento móvel no Brasil (ABEGÃO; FIGUEIREDO, 2023).

Por fim, o registro de entradas e saídas, o acompanhamento do fluxo de caixa e a categorização de gastos são apontados como bases para decisões financeiras mais conscientes. Esse entendimento é corroborado por estudos sobre aplicativos de gestão financeira, que destacam essas funcionalidades como essenciais para o planejamento financeiro e a redução do endividamento (SILVA; COELHO; SILVA, 2020).

O presente trabalho visa desenvolver uma plataforma híbrida (*mobile* e *web*) intuitiva e funcional voltada a usuários com pouca experiência em finanças pessoais. Os objetivos especificos são :

a) Oferecer uma visão clara e organizada da situação financeira do usuário.

b) Possibilitar o acompanhamento da evolução financeira pessoal ao longo do tempo.

c) Permitir o registro e a categorização de receitas e despesas de forma rapida e intuitiva, facilitando a análise do orçamento pessoal.

d) Fornecer visualizações dinâmicas que auxiliem na interpretação dos dados financeiros.

O intuito é disponibilizar uma ferramenta completa, que fortaleça o controle sobre a vida financeira, e incentive a organização, a disciplina e a educação financeira. Ferraz (2023) ressalta a importância da educação financeira na gestão pessoal, destacando que práticas como o planejamento orçamentário e o uso de ferramentas de apoio contribuem diretamente para reduzir o endividamento e promove maior autonomia financeira. Espera-se, assim, que a plataforma proposta auxilie os usuários a tomar decisões mais conscientes, estratégicas e alinhadas com seus objetivos pessoais e de longo prazo.

2 REFERENCIAL TEÓRICO

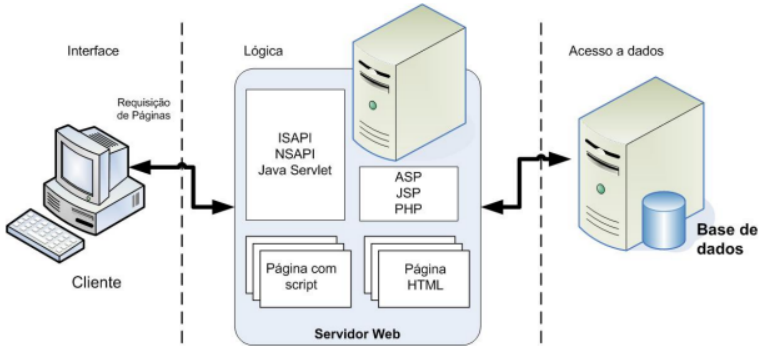
As próximas subseções apresentam conceitos fundamentais para o desenvolvimento de aplicações *Web* modernas, abordando tecnologias e práticas utilizadas neste projeto. Por fim, são exploradas noções essenciais de gestão financeira pessoal, que embasam os requisitos funcionais da aplicação , garantindo que o sistema atenda aos objetivos de organização e controle financeiro do usuário.

2.1 Aplicações *Web*

O desenvolvimento de aplicações *Web* de qualidade exige uma abordagem sistemática, considerando a crescente complexidade e diversidade de usuários. Nesse contexto, surge a Engenharia *Web*, que, embora relacionada à Engenharia de *Software* tradicional, considera características próprias de sistemas *Web*, como a multiplicidade de perfis de usuários e a necessidade de integração com diferentes mídias e plataformas (BIANCHINI, 2008).

Do ponto de vista arquitetural, as aplicações *Web* geralmente seguem o modelo cliente–servidor, no qual se destacam três camadas: Cliente *(browser).*: responsável pela interface e interação com o usuário final; Rede (HTTP/HTTPS): meio de transporte das requisições e respostas entre cliente e servidor; Servidor (*back-end*): camada que concentra regras de negócio, persistência de dados e serviços de integração. Essa divisão de responsabilidades evidencia a natureza dual de uma aplicação *Web*: de um lado, a experiência interativa do usuário (cliente) e, de outro, a lógica e as operações do sistema (servidor). Estudos sobre processos de produção de aplicações *Web* no Brasil destacam a importância de compreender tanto a perspectiva do usuário quanto a do sistema no desenvolvimento de soluções eficazes (SCHEER; SANTOS; WEBER, 2005).

Figura 1 – Aplicações *Web*: camadas Cliente, Rede e Servidor.



Fonte: BIANCHINI (2008), p. 29.

2.1.1 Framework Next

O Next.js é um *framework* de desenvolvimento *Web* full-stack construído sobre o React. Desenvolvido e mantido pela Vercel, ele oferece funcionalidades que vão além de uma aplicação React tradicional, como renderização no servidor *Server-Side Rendering* (SSR), geração estática de páginas *Static Site Generation* (SSG) e regeneração incremental de páginas estáticas *Incremental Static Regeneration* (ISR) (Next.js, 2025).

Segundo Kaur e Tiwari (2023), o ecossistema do React e de seus *frameworks* derivados, como o Next.js, destaca-se por proporcionar maior modularidade, desempenho otimizado e melhor experiência de desenvolvimento em comparação a soluções tradicionais. Esses aspectos tornam o Next.js uma escolha robusta para aplicações modernas e escaláveis.

Seu sistema de roteamento baseado em arquivos simplifica a criação de rotas, páginas e *layouts*, enquanto sua infraestrutura inclui ferramentas modernas como o Turbopack (um *bundler* escrito em Rust), Fast Refresh e otimizações automáticas como divisão de código (*code splitting*) e carregamento antecipado (Next.js, 2025).

Além disso, o Next.js permite criar rotas de API internas, integrar TypeScript, ESLint e estilos avançados (como CSS Modules e Tailwind CSS), acelerando o desenvolvimento com uma experiência moderna e produtiva (Next.js, 2025).

2.2 Interface de Programação de Aplicações (API)

As *Application Programming Interfaces* (API) são componentes fundamentais em aplicações *Web* modernas, permitindo a comunicação entre diferentes sistemas e camadas de *software*. Elas definem regras, formatos e protocolos para que clientes, como navegadores ou aplicativos móveis, possam acessar e manipular dados de forma padronizada. APIs bem projetadas tornam os sistemas mais modularizados, escaláveis e flexíveis, facilitando a integração de novos serviços, automação de processos e manutenção contínua da aplicação (Sabbag Filho, 2025).

2.2.1 Node.js

O Node.js é uma plataforma baseada em JavaScript utilizada para o desenvolvimento de APIs e serviços de back-end de forma rápida e escalável. Baseado no motor V8 do Google, o Node.js adota um modelo de execução assíncrono e orientado a eventos, operando em uma única thread responsável pelo gerenciamento de operações de entrada e saída de forma não bloqueante (TANADECHOPON; KASEMSONTITUM, 2023). De acordo com Tanadechopon e Kasemsontitum (2023), essa arquitetura favorece o processamento eficiente de múltiplas requisições simultâneas em serviços de API executados em ambientes de computação em nuvem, reforçando sua adequação para aplicações *Web* modernas.

Além dos ganhos relacionados à escalabilidade e ao desempenho, o Node.js destaca-se pela integração nativa com o ecossistema JavaScript, o que simplifica o desenvolvimento *full-stack*. Essa característica permite a unificação da lógica entre cliente e servidor, contribuindo para o aumento da produtividade no desenvolvimento de aplicações (LYON, 2022).

2.2.2 Banco de Dados NoSQL e MongoDB

O crescimento das aplicações *Web* e a necessidade de lidar com grandes volumes de dados não estruturados levaram ao surgimento dos bancos de dados NoSQL (Not Only SQL). Diferentemente dos modelos relacionais tradicionais, os bancos NoSQL priorizam flexibilidade na estrutura dos dados, alta escalabilidade e distribuição horizontal características que os tornam mais adequados para aplicações modernas e de grande porte (CHODOROW, 2022).

Entre as principais categorias de bancos NoSQL estão os baseados em documentos, colunas, grafos e chave–valor. Dentre eles, o MongoDB destaca-se como uma das soluções mais utilizadas no desenvolvimento de sistemas *Web*, por adotar o modelo orientado a documentos JSON (JavaScript Object Notation). Nesse formato, os dados são armazenados em coleções compostas por documentos , o que permite representar informações complexas de maneira flexível e próxima da estrutura usada em linguagens de programação atuais, como o JavaScript (ELMASRI; NAVATHE, 2017) .

Segundo a documentação oficial Inc. (2025), o MongoDB foi projetado para garantir alta disponibilidade, tolerância a falhas e replicação de dados entre servidores, facilitando o escalonamento horizontal em ambientes distribuídos. Além disso, sua integração nativa com o ecossistema Node.js o torna particularmente eficiente em aplicações *full-stack* baseadas em JavaScript, reduzindo a complexidade de mapeamento objeto-relacional e aumentando o desempenho em consultas e gravações simultâneas.

No contexto deste trabalho, o MongoDB foi adotado como banco de dados principal da aplicação, armazenando informações como usuários, transações, categorias e grupos financeiros. Sua estrutura orientada a documentos possibilitou a criação de coleções dinâmicas, capazes de representar categorias personalizadas e transações complexas sem comprometer a integridade dos dados. Essa escolha tecnológica proporciona maior agilidade no desenvolvimento, eficiência no acesso às informações e facilidade de manutenção, aspectos essenciais para o funcionamento de uma plataforma de gestão financeira pessoal moderna e escalável(CHODOROW, 2022).

2.2.3 GraphQL

Para otimizar a interação entre cliente e servidor, o projeto utiliza GraphQL, uma linguagem de consulta para APIs que permite requisitar apenas os dados necessários em uma única

requisição (Sabbag Filho, 2025). Diferente de APIs REST tradicionais, o GraphQL previne problemas de over-fetching (recebimento de dados desnecessários) e under-fetching (dados insuficientes), possibilitando consultas mais precisas e eficientes. O GraphQL também facilita a criação de operações complexas, como filtragem, agregação e paginação, em uma única chamada à API. Em conjunto com Node.js e MongoDB, ele garante que as informações da aplicação, como usuários, transações e categorias financeiras, sejam entregues de maneira consistente, rápida e escalável.

2.3 Planejamento Financeiro

De acordo com Souza (2023), o planejamento financeiro é definido como a adoção de uma estratégia de ação que orienta os passos em direção às metas e objetivos pessoais. Esse processo não se restringe apenas ao ato de guardar dinheiro, mas também envolve garantir estabilidade e tranquilidade futura diante das incertezas econômicas. O autor enfatiza que a disciplina e o comprometimento com metas previamente estabelecidas são elementos fundamentais para que o processo resulte em ganhos duradouros.

Nesse sentido, Santos, Franca e Batista (2024) complementa essa perspectiva ao destacar que o planejamento financeiro é um processo multifacetado e adaptativo, influenciado por fatores como renda, riscos e perfil individual. Assim como defendido por Souza (2023) também aponta que o planejamento não se limita ao controle de gastos, mas inclui etapas como definição de objetivos, organização das finanças, gestão de dívidas e até investimentos, compondo um ciclo contínuo de análise e readequação.

O primeiro passo recomendado por Souza (2023) é realizar uma autoanálise da situação financeira atual, identificando objetivos prioritários e definindo estratégias que permitam alcançá-los ao longo do tempo. Essa visão dialoga com a abordagem de Santos, Franca e Batista (2024), que ressalta que o planejamento contribui diretamente para a qualidade de vida, promovendo bem-estar emocional, redução do estresse financeiro e maior resiliência diante de adversidades econômicas.

Por fim, Souza (2023) e Santos, Franca e Batista (2024) convergem ao destacar a importância da organização das finanças pessoais, por meio do uso de ferramentas que permitam registrar entradas e saídas de recursos, inclusive os pequenos gastos. Essa prática amplia a clareza sobre o uso do dinheiro e possibilita ajustes mais conscientes, fortalecendo a capacidade de alcançar metas e construir uma vida financeira mais equilibrada.

3 METODOLOGIA

A pesquisa possuiu caráter aplicado, pois visou desenvolver uma solução prática para auxiliar na gestão de finanças pessoais. Segundo Wazlawick (2017), a pesquisa aplicada distinguiu-

se da pesquisa básica por buscar a utilização do conhecimento para a solução de problemas concretos.

Nesse contexto, o desenvolvimento da aplicação foi estruturado em etapas organizadas, seguindo uma abordagem metodológica que visou garantir a sistematização do processo de construção do *software*, conforme sugerido por Wazlawick (2017). A organização em etapas permitiu maior clareza sobre os procedimentos adotados, desde a concepção do sistema até a validação final. O detalhamento das etapas é apresentado no Quadro 1, abrangendo desde a modelagem inicial até a implementação.

Quadro 1 – Etapas de desenvolvimento

	Etapas
Modelagem	Definição das Entidades do Sistema Requisitos Funcionais e Não Funcionais Desenho das Interfaces
Back-end	Configurações Iniciais e Dependências (TypeScript) Definição de Schemas e Resolvers (GraphQL) Implementação das Queries e Mutations Documentação da API (Playground do GraphQL) Testes de Integração e Unidade
Front-end	Configurações Iniciais e Dependências Necessárias Integração Direta com a API GraphQL Componentização das Interfaces Testes de Usabilidade

Fonte: Elaborado pelos autores.

3.1 Modelagem

A definição das entidades de um sistema foi uma etapa fundamental na modelagem da aplicação, pois permitiu estruturar corretamente os dados e as relações entre eles, garantindo integridade, consistência e facilidade de manutenção (LI; CURRIM; RAM, 2022).

Em um sistema de gestão financeira, as entidades representaram os elementos centrais sobre os quais todas as funcionalidades foram construídas. A correta definição das entidades possibilitou a organização eficiente das informações, o suporte a consultas precisas, a geração de relatórios confiáveis e a criação de uma base sólida para futuras expansões do sistema (LI; CURRIM; RAM, 2022).

O levantamento de requisitos foi fundamental para o sucesso do projeto de *software*, pois garantiu que o sistema atendesse às reais necessidades dos usuários e evitasse retrabalho e desperdício de recursos. Como destacam Balieiro e Pinto (2024), o levantamento de requisitos é uma etapa essencial no processo de desenvolvimento de software, pois define as funcionalidades e restrições do sistema e orienta a aplicação das melhores práticas de planejamento e execução do projeto. Neste trabalho, o levantamento de requisitos foi realizado também por meio da análise de *softwares* de gestão financeira pessoal já consolidados no mercado, incluindo

Mobills (MOBILLS, 2025), Contas Online (ONLINE, 2025), Organizze (ORGANIZZE, 2025) e Minhas Economias (ECONOMIA, 2025).

Segundo Elaine (2024), a Engenharia de Requisitos caracteriza-se como um processo contínuo e iterativo, no qual os requisitos podem evoluir ao longo do desenvolvimento do sistema. No contexto deste estudo, essa característica refletiu-se no refinamento progressivo dos requisitos ao longo das fases de modelagem e implementação, considerando a análise comparativa das soluções existentes e os ajustes técnicos necessários, o que contribuiu para a redução de riscos e para maior coerência entre os objetivos propostos e o sistema desenvolvido.

3.2 Desenvolvimento do *Back-end*

O desenvolvimento do *back-end* foi fundamental para garantir que os dados da aplicação fossem processados, armazenados e disponibilizados de forma eficiente. Para isso, foi utilizado o GraphQL, uma linguagem de consulta para APIs que possibilitou ao cliente especificar exatamente quais dados desejava, minimizando problemas de *over-fetching* (recebimento de dados desnecessários) e *under-fetching* (não recebimento de dados suficientes) (Sabbag Filho, 2025).

O armazenamento das informações foi realizado por meio do MongoDB, um banco de dados estruturado que permitiu armazenar, organizar e recuperar informações de forma eficiente, servindo como base para o funcionamento da aplicação, conforme destacado por Elmasri e Navathe (2017). A combinação do MongoDB com o GraphQL permitiu consultas dinâmicas e personalizadas, garantindo maior eficiência no acesso e na manipulação das informações.

O processo de desenvolvimento ocorreu em etapas, iniciando-se pela configuração do ambiente e pela instalação das dependências essenciais, incluindo Node.js, TypeScript e bibliotecas fundamentais para o funcionamento do servidor. Essa fase estabeleceu uma base estável e organizada, preparando o ambiente para as implementações subsequentes.

Em seguida, foram definidos os *schemas* e os *resolvers* do GraphQL. Os *schemas* estruturaram os tipos de dados e suas relações, garantindo consistência e integridade das informações, enquanto os *resolvers* determinaram como as requisições seriam processadas, assegurando respostas precisas e flexíveis às solicitações, conforme destacado por Sabbag Filho (2025).

Por fim, foram realizados testes de integração e de unidade, com o objetivo de validar a confiabilidade da API. Esses testes garantiram que os módulos funcionassem de forma individual e integrada, assegurando maior robustez ao sistema.

3.3 Desenvolvimento do *Front-end*

A interface do sistema foi desenvolvida utilizando o *framework* Next.js, voltado ao consumo pelo usuário final e estruturado em páginas e componentes reutilizáveis. Para a estilização dos elementos da aplicação, foi utilizado o Tailwind CSS, que possibilitou a criação de inter-

faces modernas e responsivas de maneira ágil e consistente. Além disso, foi incorporada a biblioteca de componentes shadcn/ui, que forneceu tabelas e componentes pré-construídos e altamente customizáveis, acelerando o desenvolvimento e mantendo um padrão visual uniforme em toda a aplicação, conforme destacado por Rajala (2024).

O desenvolvimento do *front-end* orientou-se pelo planejamento apresentado no Quadro 1. Inicialmente, foram realizadas as configurações do ambiente e a instalação das dependências essenciais, incluindo o Tailwind CSS e bibliotecas de componentes reutilizáveis. Essa preparação inicial possibilitou acelerar a implementação das telas, garantindo consistência visual e maior produtividade. Nesse sentido, o uso do Tailwind CSS e de bibliotecas de componentes reutilizáveis contribuiu para maior agilidade e consistência no desenvolvimento das interfaces (TAILWINDCSS, 2025).

Na etapa seguinte, foi realizada a integração direta com a API GraphQL, estabelecendo a comunicação entre o *front-end* e o *back-end*. Com isso, as informações cadastradas ou consultadas pelos usuários passaram a ser exibidas em tempo real na interface, proporcionando uma experiência dinâmica e responsiva, conforme destacado por Sabbag Filho (2025).

O processo de componentização da interface permitiu dividir a aplicação em elementos reutilizáveis, como botões, tabelas de transações e gráficos. Essa abordagem facilitou a manutenção e garantiu a padronização visual em toda a plataforma, tornando o desenvolvimento mais modular e organizado.

Por fim, foram conduzidos testes de usabilidade, com o objetivo de avaliar a clareza da navegação e a eficiência da interface para o usuário final. Essa etapa possibilitou identificar ajustes necessários, assegurando que a aplicação atendesse ao objetivo de ser intuitiva e funcional.

3.4 Disponibilidade do *Software*

O protótipo desenvolvido foi disponibilizado à banca examinadora por meio de repositórios no GitHub, onde estão hospedados o código da API e da interface *web*. Os repositórios incluem as instruções necessárias para instalação, configuração e uso, permitindo que os avaliadores analisem o funcionamento do sistema de forma prática.

Link abaixo para o GitHub Classroom:

<https://github.com/ICEI-PUCMinas-PSG-SI-TI/psg-si-2025-2-p8-tcc-organizacaofinanceirapessoal>

4 DESENVOLVIMENTO

Este capítulo detalha tecnicamente a implementação do protótipo, descrevendo a configuração do ambiente, os módulos desenvolvidos e os principais trechos de código.

4.1 Levantamento de requisitos

O levantamento de requisitos é uma etapa essencial para garantir que o sistema atenda às necessidades dos usuários e evite retrabalho no desenvolvimento (BALIEIRO; PINTO, 2024). Nesse trabalho, essa etapa foi realizada também por meio da análise de *softwares* de gestão financeira pessoal já consolidados no mercado, como:

a)Mobills, que oferece funcionalidades de categorização de despesas, definição de metas e geração de relatórios gráficos (MOBILLS, 2025).

b)Contas *Online*, que disponibiliza recursos como controle de contas a pagar e receber, fluxo de caixa e relatórios de saldo (ONLINE, 2025).

c)Organizze, que permite criação de categorias personalizadas, controle de cartões de crédito, grupos de transações e integração com contas bancárias (ORGANIZZE, 2025).

d)Minhas Economias, que oferece funcionalidades similares, além de permitir acompanhamento de objetivos financeiros (ECONOMIA, 2025).

A análise desses *softwares* serviu como referência para a definição dos requisitos do sistema, identificando funcionalidades essenciais que foram implementadas. Com base nessa análise, foram definidos os requisitos funcionais e não funcionais apresentados no Quadro 2, organizados por prioridade e importância na finalização do sistema.

Quadro 2 – Requisitos Funcionais e Não Funcionais

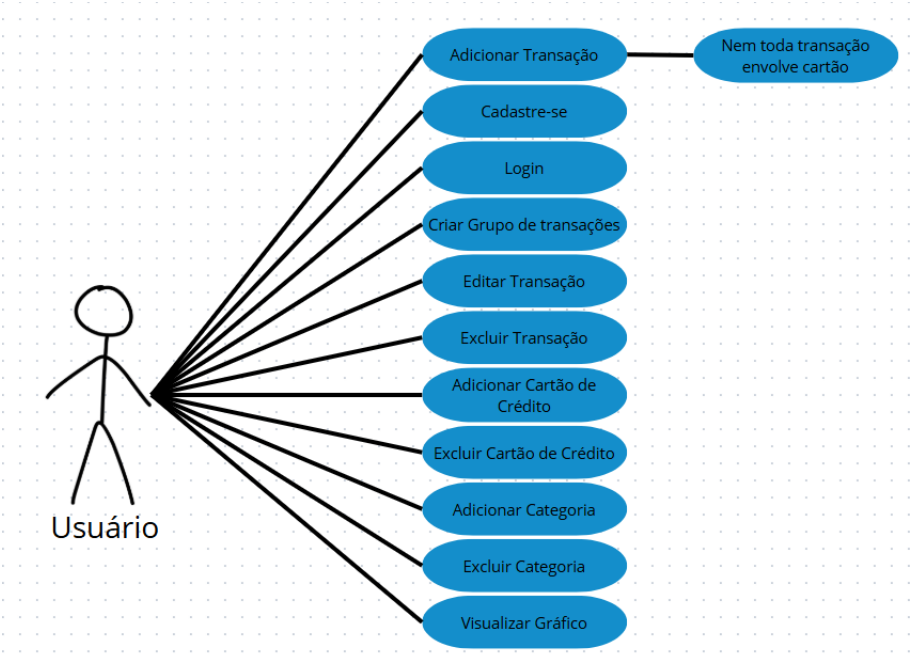
Tipo de Requisito	Código	Descrição	Prioridade
Funcionais	RF01	Manter o cadastro e autenticação de usuários.	Alta
	RF02	Manter o cadastro de contas financeiras (categorias, cartões, etc.).	Alta
	RF03	Registrar transações de entrada (receitas) e saída (despesas).	Alta
	RF04	Manter categorias personalizáveis para as transações.	Alta
	RF05	Apresentar um dashboard com o resumo financeiro geral.	Média
	RF06	Exibir gráficos de despesas por categoria.	Média
	RF07	Permitir a filtragem de transações por data, conta ou categoria.	Baixa
Não Funcionais	RNF01	*Frontend* deve ser uma *Single Page Application (SPA)* reativa.	Alta
	RNF02	*Backend* desenvolvido em TypeScript com API em GraphQL.	Alta
	RNF03	A interface deve ser responsiva para uso em desktops e dispositivos móveis.	Alta
	RNF04	Garantir a segurança e a privacidade dos dados financeiros do usuário.	Alta

Fonte: Elaborado pelos autores.

4.2 Diagrama de Casos de Uso

Segundo Pressman (2011), a UML é uma linguagem padrão para descrever e documentar um projeto de software que reúne diversos grupos de notações de modelagem, cada um com sua devida sintaxe e semântica predeterminadas. Cada um desses elementos pode ser extensível, permitindo, deste modo, que sejam adaptáveis às características específicas do projeto a ser desenvolvido. A UML independe de linguagem de programação e dos processos de desenvolvimento do software, fazendo com que diferentes abordagens possam ser utilizadas durante o processo (BEZERRA, 2007). O diagrama de casos de uso do protótipo proposto pode ser visto na Figura 2.

Figura 2 – Diagrama de Casos de Uso



Fonte: Elaborado pelo autores (2025)

O Quadro 3 a apresenta uma breve descrição de cada caso de uso identificado no diagrama, destacando suas finalidades dentro do contexto da aplicação. Essas funcionalidades têm como objetivo oferecer ao usuário um controle mais eficiente sobre suas finanças pessoais, possibilitando o acompanhamento detalhado de receitas, despesas e o uso de cartões de crédito.

Quadro 3 – Descrição dos Casos de Uso

Caso de Uso	Descrição
Cadastrar-se	Permite que um novo usuário crie uma conta na aplicação, informando seus dados básicos para acesso futuro ao sistema de gestão financeira.
Login	Possibilita que o usuário autenticado acesse sua conta para gerenciar suas finanças pessoais de forma segura.
Adicionar Transação	Permite registrar uma nova transação financeira, seja uma despesa ou receita. Nem toda transação precisa estar vinculada a um cartão de crédito.
Editar Transação	Possibilita ao usuário modificar os dados de uma transação já registrada, como valor, data, categoria ou tipo (receita/despesa).
Excluir Transação	Permite remover uma transação previamente cadastrada, caso tenha sido lançada incorretamente ou não seja mais necessária.
Criar Grupo de Transações	Permite agrupar múltiplas transações sob um mesmo contexto (ex: viagem, projeto, evento), facilitando a análise de gastos específicos.
Adicionar Cartão de Crédito	Permite que o usuário registre um novo cartão de crédito no sistema, associando futuras transações a ele.
Excluir Cartão de Crédito	Possibilita a remoção de um cartão de crédito cadastrado, quando este não for mais utilizado ou estiver desatualizado.
Adicionar Categoria	Permite ao usuário criar novas categorias personalizadas (ex: alimentação, transporte, lazer) para classificar suas transações.
Excluir Categoria	Possibilita a exclusão de uma categoria que não está mais em uso, mantendo a organização das finanças.
Visualizar Gráfico	Exibe representações gráficas do desempenho financeiro do usuário, como despesas por categoria e evolução de receitas e gastos ao longo do tempo.

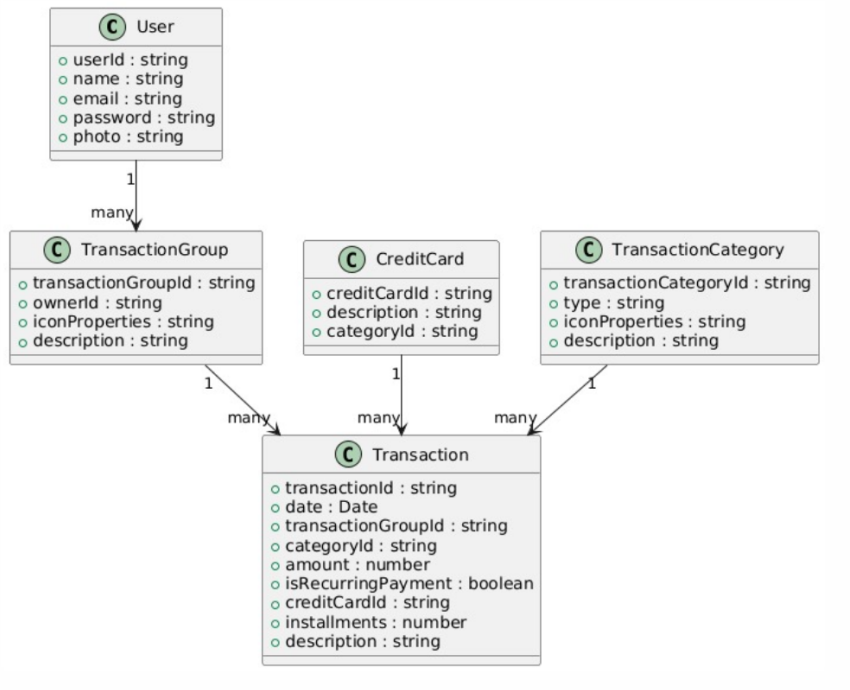
Fonte: Elaborado pelos autores.

4.3 Implementação do *Back-end*

4.3.1 Diagrama de Classes

O diagrama de classes (figura 3) detalha a estrutura de dados do sistema, apresentando as entidades principais, seus atributos e os relacionamentos entre elas. Ele fornece uma visão de como os dados são organizados no *back-end* e como as diferentes entidades interagem, como por exemplo: um usuário pode possuir vários grupos de transações, cada transação está associada a uma categoria e a um cartão de crédito. Esse diagrama auxiliou no entendimento da modelagem do sistema e na manutenção futura do código.

Figura 3 – Diagrama de Classes.



Fonte: Elaborado pelo autores (2025)

O diagrama de classes apresentado representa a estrutura principal do sistema financeiro proposto, destacando as entidades, seus atributos e os relacionamentos entre elas. A classe *User* armazena as informações do usuário, como nome, e-mail, senha e foto, sendo o ponto central do sistema. Cada usuário pode possuir diversos *TransactionGroup*, que funcionam como agrupamentos de transações, permitindo uma melhor organização dos registros financeiros.

Dentro de cada grupo estão as *Transaction*, que representam as transações realizadas, contendo informações como data, valor, descrição, categoria, número de parcelas e indicação se o pagamento é recorrente. Cada transação pode estar vinculada a um *CreditCard*, responsável por armazenar os dados dos cartões de crédito cadastrados pelo usuário, e também a uma *TransactionCategory*, que classifica a natureza da transação (por exemplo, alimentação, transporte ou lazer).

Essa estrutura facilita a organização, consulta e gerenciamento das informações financeiras, além de estabelecer relacionamentos claros entre usuários, grupos, cartões e categorias. Dessa forma, o diagrama de classes contribui para uma modelagem sólida e para a manutenção futura do sistema.

4.3.2 Arquitetura e Implementação do Back-end

Para a implementação do *back-end* foi utilizada a plataforma Node.js com TypeScript, permitindo o desenvolvimento de uma API fortemente tipada e modular. Inicialmente, o ambiente de desenvolvimento foi configurado criando-se um projeto Node.js e definindo-se as opções

de compilação do TypeScript no arquivo *tsconfig.json*, garantindo tipagem forte e compatibilidade com o padrão ECMAScript desejado. Em seguida, foram instaladas as dependências essenciais, cada uma com sua finalidade específica:

- **Apollo Server**: responsável pelo gerenciamento das requisições GraphQL;
- **GraphQL**: utilizado para definição de tipos, consultas e mutações;
- **Mongoose**: empregado na modelagem e na interação estruturada com o banco de dados MongoDB;
- **Dotenv**: utilizado para o gerenciamento seguro de variáveis de ambiente.

Com as dependências instaladas, organizou-se a estrutura de pastas de forma modular, separando:

- os modelos de dados do banco (*models*);
- as definições de tipos e operações GraphQL (*schemas*);
- a lógica responsável pela execução dessas operações (*resolvers*).

A arquitetura do *back-end* seguiu as boas práticas sugeridas por (LYON, 2022), permitindo a separação clara de responsabilidades e facilitando a manutenção futura do sistema.

O *schema* foi definido para refletir as entidades principais do sistema, descrevendo seus campos e relacionamentos. A listagem de categorias, por exemplo, combina registros padrões e customizados, substituindo dinamicamente categorias *default* pelas equivalentes personalizadas do usuário, seguindo a abordagem de integração entre dados padrão e customizados sugerida por (LYON, 2022). Um exemplo simplificado do schema GraphQL implementado pode ser visto no trecho de código na figura 4.

Figura 4 – Trecho do schema GraphQL da aplicação

```
const TransactionModule: GraphQLModule = {
  typeDefs: gql`
    type Installments {
      total: Int!
      current: Int!
    }

    type Transaction {
      _id: ObjectID!
      transactionGroupId: ObjectID!
      category: TransactionCategory!
      date: Date!
      description: String!
      amount: Float!
      installments: Installments
      creditCard: CreditCard
      isRecurringPayment: Boolean!
    }

    extend type Query {
      transactionById(_id: ObjectID!): Transaction
    }
  `
}
```

Fonte: Elaborado pelos autores (2025)

O trecho apresentado na figura 4 mostra parte do *schema GraphQL* da aplicação, onde são definidos os tipos Installments e Transaction. O primeiro representa informações de parcelamento, enquanto o segundo descreve os campos principais de uma transação financeira, como categoria, data, valor e cartão associado. Também é exibida a query transactionById, responsável por recuperar uma transação específica no banco de dados.

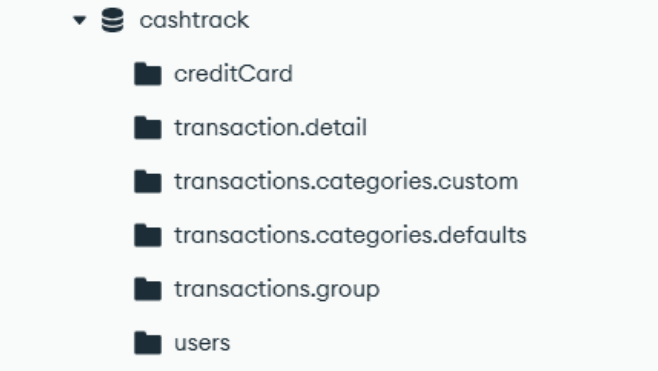
Na sequência, os *resolvers* foram implementados para mapear as operações definidas no *schema* para funções concretas no servidor. Por exemplo, a query *transactionById* executa a busca de uma transação no banco de dados MongoDB.

A conexão com o MongoDB foi realizada via Mongoose, definindo modelos compatíveis com os tipos do GraphQL. Essa abordagem garante consistência entre a camada de API e a de persistência, além de simplificar a criação de novas *queries* e *mutations*.(LYON, 2022) A estrutura do banco de dados foi organizada em coleções distintas, cada uma representando uma entidade essencial do sistema, como usuários, grupos de transações, cartões de crédito e categorias (padrões e personalizadas). Essa organização segue princípios de modularidade e coesão, facilitando a manutenção e futuras expansões da aplicação.

A Figura 5 apresenta a estrutura do banco de dados *casthtrack*, destacando as coleções

principais criadas no MongoDB. Essa disposição evidencia a separação entre dados de usuários, transações e categorias, permitindo consultas mais eficientes e maior clareza na modelagem da aplicação.

Figura 5 – Estrutura das coleções no banco de dados MongoDB.



Fonte: Elaborado pelos autores (2025)

4.4 Implementação do *Front-end*

O *front-end* da aplicação foi desenvolvido com o *framework* Next.js aliado ao React e TypeScript, permitindo a construção de páginas e componentes reutilizáveis de forma tipada e organizada. A escolha dessa combinação tecnológica se justifica pelo fato de o Next.js, assim como outras soluções modernas de *front-end*, apresentar vantagens em termos de performance, escalabilidade e experiência do desenvolvedor, quando comparado a alternativas tradicionais (KAUR; TIWARI, 2023). Inicialmente, configurou-se o ambiente com as dependências principais – Next, React, Typescript, Tailwind, Apollo/client e shadcn/ui – e definiu-se a estrutura de pastas seguindo o padrão *app/* do Next.js, separando páginas, componentes e estilos.

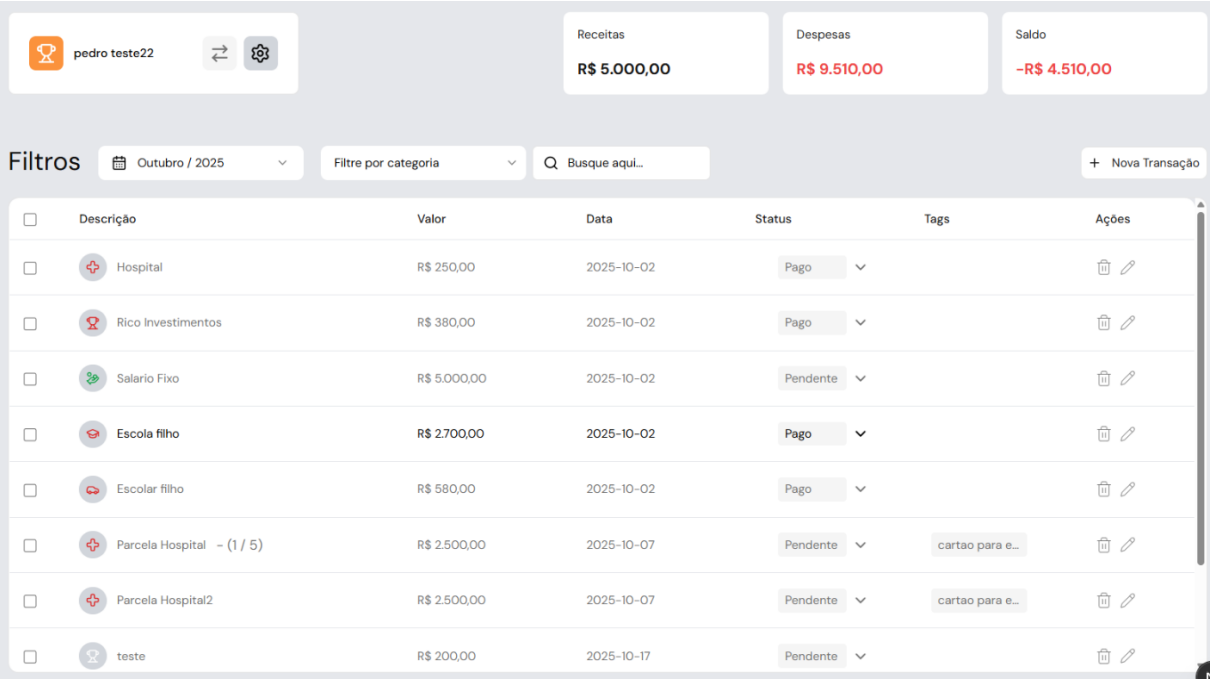
Para a estilização das interfaces foi adotado o Tailwind CSS, que fornece classes utilitárias para criação de *layouts* responsivos e modernos de maneira ágil (TAILWINDCSS, 2025). Em conjunto, utilizou-se a biblioteca shadcn/ui, responsável por disponibilizar componentes pré-construídos e customizáveis, o que acelerou o desenvolvimento e garantiu consistência visual em toda a aplicação (SHADCN/UI, 2025).

A integração com a API GraphQL foi realizada por meio do Apollo Client, que gerencia *queries*, *mutations* e *cache* de dados no *front-end*.Utilizou-se o *InMemoryCache* como mecanismo padrão, com políticas de *fetch* (ex.: *cache-first*, *cache-and-network*) para balancear desempenho e atualização de dados. Em *mutations*, utilizou-se apenas o *refetchQueries* para atualizar o cache local de forma eficiente, evitando recarregar consultas inteiras. Para cenários mais complexos de paginação ou dados relacionados, definiram-se *type policies / field policies* que orientam como o Apollo normaliza e mescla dados no cache. Essa abordagem melhora a responsividade da interface, reduz chamadas redundantes à API e facilita a manutenção coerente do estado local dos dados (APOLLO,)

A interface foi componentizada para facilitar manutenção e evolução futura. Foram criados componentes reutilizáveis como *Transactions*, *ComboboxCategories* e *TransactionsFilter*, responsáveis por listar transações, selecionar categorias e aplicar filtros, respectivamente. Essa abordagem modulariza o código e permite a reaplicação dos mesmos elementos em diferentes partes do sistema.

Após a autenticação, o usuário é direcionado à tela principal (Figura 6), onde é possível visualizar as transações cadastradas, aplicar filtros por data e categoria, além de adicionar novas transações.

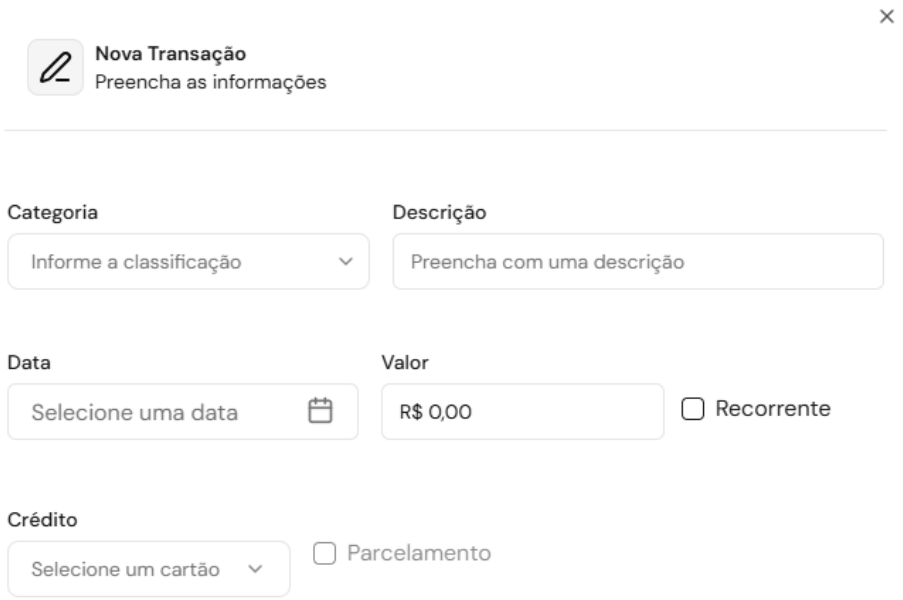
Figura 6 – Tela principal com listagem de transações e filtros dinâmicos



Fonte: Elaborado pelos autores (2025)

A Figura 7 ilustra o formulário de criação de uma nova transação, o qual permite ao usuário registrar informações financeiras de forma estruturada e padronizada. Nesse formulário, são coletados dados como categoria da transação, descrição, valor, data e demais campos necessários para o correto armazenamento e posterior processamento das informações no sistema. A interface foi desenvolvida visando à usabilidade e clareza, garantindo que o usuário consiga realizar o cadastro da transação de maneira simples e intuitiva.

Figura 7 – Formulário de criação de nova transação



Fonte: Elaborado pelos autores (2025)

Um exemplo simplificado de componente React consumindo a API GraphQL pode ser visto na figura 8, que demonstra como as queries são definidas e executadas para buscar informações no *backend* .

Figura 8 – Componente React exibindo query grupo de transações via GraphQL.

```
export default gql`
  query TransactionsGroup($search: String) {
    transactionsGroup(search: $search) {
      _id
      owner {
        name
        _id
      }
      iconProperties {
        background
        color
        icon
      }
      description
    }
  }
`;
```

Fonte: Elaborado pelos autores (2025)

Na figura 8 é possível observar um componente React que realiza uma query via GraphQL para recuperar o grupo de transações. A consulta (*TransactionsGroup*) recebe um parâmetro de

busca (*search*) e retorna informações como o id da transação, dados do proprietário (*owner*), propriedades do ícone (*iconProperties*) e a descrição da transação. Esse exemplo ilustra como o componente consome dados da API de forma tipada e estruturada, permitindo renderizar dinamicamente a lista de transações e facilitar a manutenção e evolução futura do código.

4.4.1 Visualização dos Dados Financeiros

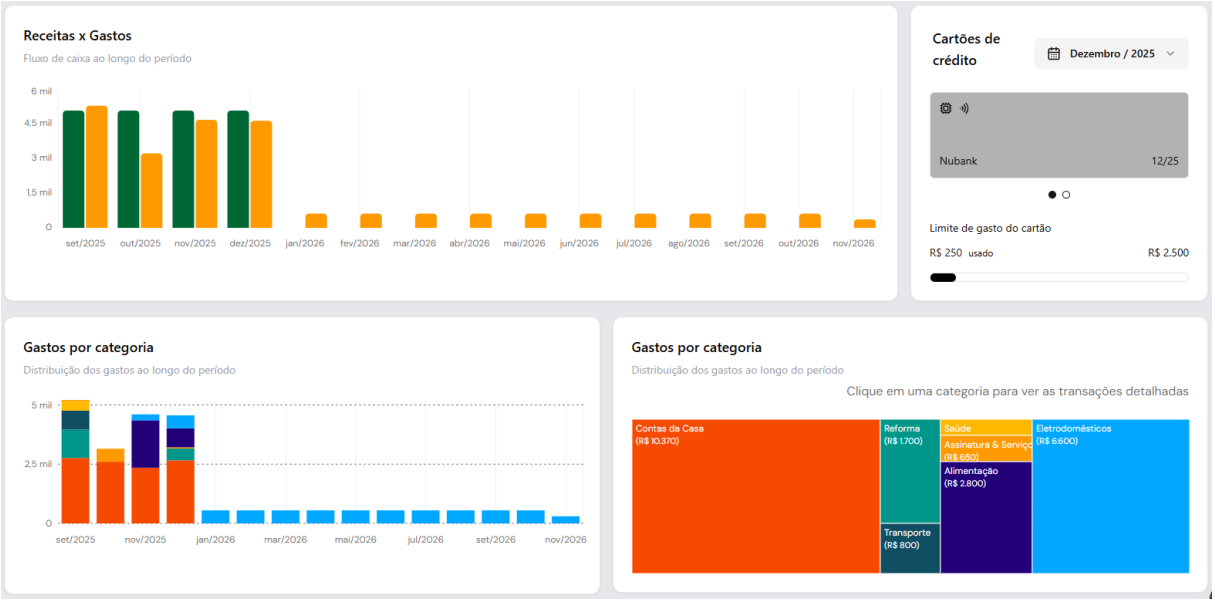
A plataforma inclui um conjunto de gráficos que compõem a principal interface de análise financeira do usuário. Essas visualizações foram desenvolvidas utilizando React, Apollo Client e uma biblioteca de gráficos integrada ao front-end, permitindo exibição dinâmica e atualizada dos dados cadastrados. A adoção dessas representações segue as recomendações de Sharda, Delen e Turban (2020), que destacam a importância da *visual analytics* no suporte à tomada de decisão e na identificação de padrões de consumo.

Foram implementados quatro modelos principais de gráficos:

- **Gráfico de barras para Receitas x Gastos**: utilizado para comparar a variação mensal do fluxo de caixa, facilitando a identificação de períodos de maior ou menor equilíbrio financeiro.
- **Indicadores visuais para cartões de crédito**: exibindo limites utilizados por meio de barras de progresso, permitindo leitura rápida de métricas críticas.
- **Gráfico de barras empilhadas para Gastos por Categoria**: demonstrando a distribuição das despesas entre categorias ao longo do mês.
- **Treemap para análise proporcional de Categorias**: permitindo reconhecer rapidamente quais categorias possuem maior peso no orçamento.

A visualização dos dados pode ser observada no painel da aplicação, conforme ilustrado na Figura 9.

Figura 9 – Painel Principal com Visualização dos Dados Financeiros



Fonte: Elaborado pelos autores (2025)

4.5 Síntese do Desenvolvimento

O desenvolvimento do protótipo permitiu implementar todas as funcionalidades essenciais definidas nos requisitos, incluindo o cadastro de transações, categorização, agrupamento e geração de visualizações financeiras. Além disso, foram realizados testes de integração no back-end, que consistiram na validação da comunicação entre a API GraphQL, as regras de negócio e o banco de dados, assegurando que as operações de criação, consulta, atualização e exclusão de dados funcionassem corretamente de forma integrada. No front-end, foram aplicados testes unitários, com foco na verificação isolada de componentes e funções, garantindo que cada elemento da interface apresentasse o comportamento esperado diante das interações do usuário. Os resultados obtidos demonstram que a plataforma apresenta boa estabilidade e atende aos objetivos específicos propostos, especialmente no que diz respeito à clareza das informações e à facilidade de uso. Esses resultados consolidam a entrega do protótipo e fornecem base para aprimoramentos futuros.

5 CONSIDERAÇÕES FINAIS

O objetivo central, desenvolver uma plataforma intuitiva e responsiva para organização financeira pessoal, projetada desde sua origem para possibilitar o acesso eficiente tanto em computadores quanto em dispositivos móveis, foi plenamente alcançado. A proposta buscava criar uma solução acessível que ajudasse usuários a compreender, acompanhar e gerenciar suas finanças de maneira clara e eficiente. Para isso, utilizou-se um conjunto de tecnologias

— Next.js, GraphQL, Node.js, TypeScript e MongoDB — escolhidas por sua capacidade de oferecer desempenho, escalabilidade e responsividade voltada à experiência em smartphones e computadores, garantindo uma interface adaptável, fluida e adequada ao uso móvel, aspecto essencial para aplicações que lidam com dados dinâmicos e interação frequente (LYON, 2022; CHODOROW, 2022).

A plataforma oferece uma visão organizada da situação financeira do usuário por meio de um dashboard intuitivo, com indicadores essenciais e filtros que facilitam a análise de períodos e variações, mantendo boa usabilidade tanto em telas maiores quanto em dispositivos móveis. O registro e a categorização das transações foram implementados de forma simples, com um formulário direto, categorias personalizáveis e uma tabela que permite gerenciar lançamentos com clareza, preservando a experiência de uso em smartphones, mesmo em contextos de acesso rápido e em movimento. Os elementos visuais, incluindo gráficos, indicadores e listas, contribuem para tornar a interpretação das informações mais acessível. Dessa forma, os resultados obtidos indicam que a solução atende ao que foi proposto, ainda que existam possibilidades de aprimoramento e expansão em trabalhos futuros.

As contribuições do trabalho se destacam ao demonstrar como a integração entre tecnologias atuais pode resultar em soluções úteis, práticas e orientadas para a educação financeira. Além do impacto aplicado, o projeto agrega relevância acadêmica ao discutir o uso combinado de GraphQL, MongoDB e Next.js em um contexto de finanças pessoais. Essa combinação tecnológica evidencia a viabilidade de um desenvolvimento híbrido e multiplataforma, no qual uma única aplicação web responsiva permite o acompanhamento das informações financeiras também por meio de dispositivos móveis, favorecendo a flexibilidade, o dinamismo e o acesso contínuo aos dados. A possibilidade de utilização da plataforma em smartphones amplia seu potencial de uso cotidiano, tornando o controle financeiro mais presente na rotina do usuário, sem comprometer desempenho, consistência visual ou escalabilidade.

Apesar dos resultados positivos, o desenvolvimento enfrentou algumas limitações. O tempo reduzido disponível para implementar todas as funcionalidades desejadas influenciou escolhas de priorização, fazendo com que certos recursos fossem deixados para etapas futuras. Além disso, a não realização de testes com usuários finais e a ausência de testes automatizados ao longo do desenvolvimento limitaram uma avaliação mais aprofundada da experiência do usuário e da robustez do sistema em diferentes cenários de uso, restringindo uma análise mais ampla sobre o grau de intuitividade e confiabilidade da aplicação em contextos variados. Essas limitações, no entanto, não prejudicam a validade da solução entregue, elas apenas indicam caminhos claros para evolução, especialmente porque são justificáveis dentro dos limites naturais de um projeto acadêmico com prazo definido.

Nesse sentido, há diversas possibilidades de continuidade. Entre elas, destacam-se a implementação de relatórios mais completos, que ampliariam a capacidade de análise, a expansão da plataforma para dispositivos móveis por meio de uma versão em React Native, tornando o acesso mais prático e cotidiano, a realização de testes aprofundados com usuários reais, fundamentais para validar e aprimorar a experiência, e a inclusão de recursos avançados de se-

gurança, como autenticação em dois fatores, garantindo maior proteção aos dados financeiros. Esses aprimoramentos indicam um caminho promissor para que a ferramenta se torne cada vez mais robusta, útil e abrangente.

REFERÊNCIAS

ABEGÃO, F. L.; FIGUEIREDO, J. C. B. d. **Effects of age and income moderation on adoption of mobile payments in Brazil**. [S.l.: s.n.], 2023. Disponível em: <https://www.emerald.com/inmr/article/20/4/353/178636/Effects-of-age-and-income-moderation-on-adoption>. Acesso em: 24 set. 2025.

APOLLO. **Apollo Docs**. Disponível em: <https://www.apollographql.com/docs>. Acesso em: 17 nov. 2025.

BALIEIRO, A. F.; PINTO, G. S. A importância do levantamento de requisitos no desenvolvimento de softwares. **Revista Interface Tecnológica**, 2024. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/article/view/1845>. Acesso em: 16 set. 2025.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 2. ed. Rio de Janeiro: Campus, 2007. Disponível em: https://www.tecgraf.puc-rio.br/ftp_pub/lfm/EduardoBezerra-PrincipiosAnaliseProjetoSistemasComUML-2aEd.pdf. Acesso em: 1 nov. 2025.

BIANCHINI, S. L. **Avaliação de métodos de desenvolvimento de aplicações web**. 2008. Tese (Doutorado), 2008. Disponível em: <https://www.teses.usp.br/teses/disponiveis/55/55134/tde-01072008-143726/pt-br.php>. Acesso em: 8 set. 2025.

CHODOROW, K. **MongoDB**. Sebastopol, CA: O'Reilly Media, 2022. Disponível em: <https://pepa.holla.cz/wp-content/uploads/2016/11/MongoDB-The-Definitive-Guide.pdf>. Acesso em: 20 nov. 2025.

ECONOMIA, M. **Minhas Economias – Controle financeiro pessoal**. 2025. Disponível em: <https://minhaseconomias.com.br>. Acesso em: 6 nov. 2025.

ELAINE. **Trabalhando com engenharia de requisitos**. 2024. Disponível em: <https://www.devmedia.com.br/trabalhando-com-engenharia-de-requisitos/30207>. Acesso em: 1 out. 2025.

ELMASRI, R.; NAVATHE, S. **Fundamentals of database systems**. 7. ed. India: Pearson India, 2017. Disponível em: <https://www.devmedia.com.br/trabalhando-com-engenharia-de-requisitos/30207>. Acesso em: 1 out. 2025.

FERRAZ, J. C. A educação financeira e sua importância na gestão financeira pessoal. **Revista Científica Multidisciplinar Núcleo do Conhecimento**, 2023. Disponível em: <https://pt.scribd.com/document/721203851>. Acesso em: 29 out. 2025.

FUND, I. M. Economic resilience amid disinflation. **IMF eLibrary**, 2024. Disponível em: <https://www.elibrary.imf.org/view/journals/002/2024/209/article-A001-en.xml>. Acesso em: 22 set. 2025.

INC., M. **Site oficial - Mongo DB6**. 2025. Disponível em: <https://www.mongodb.com/pt-br/docs/>. Acesso em: 22 nov. 2025.

KAUR, G.; TIWARI, R. G. Comparison and analysis of popular frontend frameworks and libraries: An evaluation of parameters for frontend web development. In: **2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)**. [S.l.: s.n.], 2023. p. 1067–1073. Disponível em: <https://ieeexplore.ieee.org/document/10192987>. Acesso em: 14 nov. 2025.

LI, Y.; CURRIM, F.; RAM, S. Data completeness and complex semantics in conceptual modeling: the need for a disaggregation construct. **ACM Digital Library**, 2022. Disponível em: <https://dl.acm.org/doi/10.1145/3532784>. Acesso em: 3 out. 2025.

LYON, W. **Full stack GraphQL applications**. [S.l.]: Manning Publications, 2022. Disponível em: <https://neo4j.com/blog/graphql/free-book-full-stack-graphql-applications/>. Acesso em: 1 nov. 2025.

MOBILLS. **Mobills – Controle de gastos e finanças pessoais**. 2025. Disponível em: <https://www.mobills.com.br>. Acesso em: 18 out. 2025.

Next.js. **Next.js Docs**. 2025. Disponível em: <https://nextjs.org/docs>. Acesso em: 10 set. 2025.

ONLINE, C. **Contas Online – Sistema de gestão financeira**. 2025. Disponível em: <https://www.contasonline.com.br>. Acesso em: 22 out. 2025.

ORGANIZZE. **Organizze – Controle financeiro pessoal**. 2025. Disponível em: <https://www.organizze.com.br>. Acesso em: 8 nov. 2025.

PRESSMAN, R. S. **Engenharia de software**. 7. ed. New York: AMGH Editoras, 2011. Disponível em: <https://www.kufunda.net/publicdocs/Engenharia%20de%20Software%20-%207.ed.%20(Roger%20S.%20Pressman).pdf>. Acesso em: 19 nov. 2025.

RAJALA, O. Impact of react component libraries on developer experience – an empirical study on component libraries' styling approaches. **ResearchGate**, 2024. Disponível em: <https://www.researchgate.net/publication/383941317>. Acesso em: 1 out. 2025.

Sabbag Filho, N. **Modelagem de APIs com REST e GraphQL**. 2025. Disponível em: <https://leaders.tec.br/artigo/modelagem-de-apis-com-rest-e-graphql-estudo-de-caso-em-projetos-net>. Acesso em: 3 out. 2025.

SANTOS, P. C. d.; FRANCA, P. M.; BATISTA, V. C. O impacto do planejamento financeiro na qualidade de vida: fatores, benefícios e recomendações. **FOCO**, 2024. Disponível em: <https://ojs.focopublicacoes.com.br/foco/article/view/6589>. Acesso em: 7 nov. 2025.

SCHEER, S.; SANTOS, E. R. d.; WEBER, R. F. Uma proposta de processo de produção de aplicações web. **Produção**, v. 15, n. 3, p. 378–389, 2005. Disponível em: <https://www.scielo.br/j/prod/a/7CkmhgmfB3TVXBjKhTjhSMy>. Acesso em: 12 out. 2025.

SHADCN/UI. **Documentação oficial shadcn/ui**. 2025. Disponível em: <https://ui.shadcn.com/docs>. Acesso em: 1 nov. 2025.

SHARDA, R.; DELEN, D.; TURBAN, E. **Business intelligence, analytics, and data science**. 4. ed. [S.l.]: Pearson, 2020. Disponível em: <https://opac.atmaluhur.ac.id/uploaded_files/temporary/DigitalCollection/ZTRjYzc5ODA3OTQ0NTJlMTAxODI5MDlhM2QxMTcwM2E2NTE5Yzc4Mw==.pdf>. Acesso em: 27 nov. 2025.

SILVA, A. C. B. S. d.; COELHO, B. M. L.; SILVA, F. C. L. d. Aplicativos de gestão financeira. **Revista Pesquisa em Administração**, 2020. Disponível em: <https://periodicos.ufpe.br/revistas/index.php/rpa/article/download/244946/35549/172975>. Acesso em: 22 set. 2025.

SOUZA, R. **A importância do planejamento financeiro**. 2023. <https://www.gov.br/investidor/pt-br/penso-logo-invisto/a-importancia-do-planejamento-financeiro> — Acesso em: 10 set. 2025.

TAILWINDCSS. **Por que usar Tailwind CSS**. 2025. Disponível em: <https://tailwindcss.com.br/guia-tailwind/por-que-usar>. Acesso em: 23 out. 2025.

TANADECHOPON, T.; KASEMSONTITUM, B. Performance evaluation of programming languages as api services for cloud environments: a comparative study of php, python, node.js and golang. In: **2023 7th International Conference on Information Technology (InCIT)**. [S.l.: s.n.], 2023. p. 293–297. Disponível em: <https://ieeexplore.ieee.org/document/10413079>. Acesso em: 27 nov. 2025.

WAZLAWICK, R. S. **Metodologia de pesquisa para ciência da computação**. Brasil: Elsevier Brasil, 2017. Disponível em: <https://share.google/7xCr3u3BBqAfD2ivc>. Acesso em: 12 out. 2025.