INSTITUTO FEDERAL
Rondônia

MINISTÉRIO DA EDUCAÇÃO
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Educação, Ciência e Tecnologia de Rondônia

Campus Vilhena

Coordenação do Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

THALYSSON EMANOEL CARNEVALE VARGAS

Sistema de gerenciamento de finanças pessoais

VILHENA - RO

2025

THALYSSON EMANOEL CARNEVALE VARGAS

SISTEMA DE GERENCIAMENTO DE FINANÇAS PESSOAIS

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Gilberto Pereira da Silva.

VILHENA - RO

2025

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

Vargas, Thalysson Emanoel Carnevale.
Sistema de gerenciamento de finanças pessoais / Thalysson Emanoel Carnevale Vargas. - Vilhena, 2025.
28 f.

Orientador(a): Prof. Me. Gilberto Pereira da Silva.

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

1. Gerenciamento de finanças pessoais. 2. Sistema web. 3. Kanban. 4. Arquitetura REST. 5. CI/CD. I. Silva, Gilberto Pereira da (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

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

THALYSSON EMANOEL CARNEVALE VARGAS

SISTEMA DE GERENCIAMENTO DE FINANÇAS PESSOAIS

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Gilberto Pereira da Silva.

Aprovado em 10/12/2025 pela banca examinadora.

gov.br
Documento assinado digitalmente
JOSE LUCAS BRANDAO MONTES
Data: 18/12/2025 15:46:46-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre José Lucas Brandão Lopes

Examinador interno

gov.br
Documento assinado digitalmente
MARCO ANTONIO AUGUSTO DE ANDRADE
Data: 18/12/2025 16:21:00-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Marco Antonio Augusto de Andrade

Examinador interno

gov.br
Documento assinado digitalmente
GILBERTO PEREIRA DA SILVA
Data: 18/12/2025 14:45:10-0300
Verifique em https://validar.iti.gov.br

Prof. Mestre Gilberto Pereira da Silva

Orientador

Resumo

Este artigo apresenta o desenvolvimento do *Financial Record*, um sistema *web* voltado ao gerenciamento de finanças pessoais para auxiliar no controle orçamentário. A metodologia adotada para o desenvolvimento foi o *Kanban*, possibilitando uma administração visual das atividades e ênfase na entrega contínua. O sistema utiliza arquitetura *REST* com *back-end* feito com *node.js* com a biblioteca em *Express*, *front-end* em *Next.js* e banco de dados *MySQL*. A solução implementa a biblioteca *Decimal.js* para precisão em cálculos financeiros e automação de transações recorrentes. A validação ocorreu via testes de unidade, integração e ponta a ponta, garantindo a confiabilidade do software. O *deploy* foi realizado utilizando *pipelines* de *CI/CD*, *Docker* e *Kubernetes*, resultando em uma ferramenta funcional para o planejamento financeiro individual.

Palavras-chave: Gerenciamento de Finanças Pessoais; Sistema *Web*; *Kanban*; Arquitetura *REST*; *CI/CD*.

Abstract

This article presents the development of Financial Record, a web-based system for personal finance management aimed at assisting with budget control. The development methodology adopted was Kanban, enabling visual management of activities and emphasizing continuous delivery. The system uses a REST architecture with a back-end built in Node.js with an Express library, a front-end in Next.js, and a MySQL database. The solution implements the Decimal.js library for financial calculations and automation of recurring transactions. Validation was performed via unit, integration, and end-to-end testing, ensuring software reliability. Deployment was carried out using CI/CD pipelines, Docker, and Kubernetes, resulting in a functional tool for individual financial planning.

Keywords: Personal Finance Management; Web System; Kanban; REST Architecture; CI/CD.

1. Introdução

Nos últimos anos, muitas pessoas enfrentam dificuldades para gerenciar suas receitas e despesas, o que pode comprometer o gerenciamento financeiro. A princípio, tal fato se torna realidade, uma vez que em janeiro de 2025, 76,1% das famílias brasileiras demonstraram ter dívidas a vencer (Confederação Nacional do Comércio de Bens, Serviços e Turismo, 2025). Para diminuir essa problemática, foi criado um sistema *web*, no qual o usuário pode registrar receitas e despesas, as quais podem ser visualizadas por meio de gráficos interativos. Com isso, busca-se simplificar o monitoramento da condição financeira.

Por conseguinte, o comprometimento da renda com dívidas é uma preocupação crescente. De acordo com a Confederação Nacional do Comércio de Bens, Serviços e Turismo (2025), o percentual de consumidores que têm mais da metade de seus rendimentos comprometidos com dívidas atingiu 20,8% em janeiro de 2025, sendo este o maior percentual desde maio de 2024. Além disso, o comprometimento médio da renda com dívidas alcançou 30% no mesmo período. Com isso, torna-se fato que o endividamento é uma situação que, frequentemente, pode ser prevenida com uma educação financeira adequada e inteligente. Isso pode ser feito por meio de metodologias modernas, como *softwares*, sistemas *web* ou aplicativos que ajudam no gerenciamento das finanças pessoais.

Ademais, o principal fundamento para a criação deste trabalho surgiu de situações cotidianas de pessoas físicas que enfrentam desafios para elaborar um planejamento financeiro adequado. Isso ocorre, em parte, porque uma parte significativa da população brasileira está endividada devido à falta de ferramentas para gerenciar suas finanças ou, mais provavelmente, à negligência de métodos de controle financeiro.

A necessidade de ferramentas de gestão é reforçada pelo cenário de endividamento e inadimplência. Segundo a Confederação Nacional do Comércio de Bens, Serviços e Turismo (2023), a proporção média de famílias com dívidas em atraso atingiu o recorde de 29,5% em 2023, e o percentual daquelas sem condições de pagar suas dívidas alcançou a máxima histórica de 12,1%. Um dos principais vetores desse cenário é o cartão de crédito, o meio mais utilizado pelas famílias (87,2% das dívidas), que possui juros rotativos de 434,4% ao ano. Diante disso, a própria CNC (2023) recomenda que os consumidores "devem priorizar o planejamento orçamentário e buscar adquirir conhecimento sobre gestão financeira", o que reforça a importância de um sistema *web* que auxilie nesse controle de forma prática.

Dada a importância do controle das finanças pessoais e o alto índice de informatização da sociedade, o uso de tecnologias digitais surge como um método de facilitar esse processo, promovendo organização, planejamento e redução de desperdícios. Dito isso, este projeto propõe o desenvolvimento de um sistema *web* para registro e visualização de transações financeiras. A proposta se justifica pela necessidade de soluções simples e fáceis que incentivem a educação financeira e o uso consciente dos recursos.

Diante desse contexto, existe o seguinte problema de pesquisa: Como desenvolver um sistema web para o gerenciamento das finanças pessoais, permitindo o

registro e a visualização dos ganhos e gastos, promovendo maior controle e organização?

Nesse contexto, o trabalho atual tem como foco desenvolver uma aplicação *web* que auxilia os usuários a controlarem e entenderem melhor suas finanças pessoais. A plataforma permite o cadastro de receitas, despesas e metas mensais. Além da possibilidade de registrar as transferências entre contas, tendo um melhor entendimento do saldo distribuído existente. O intuito é promover a educação financeira e incentivar a adoção de hábitos mais sustentáveis de consumo, por meio do controle sistemático e da organização orçamentária. Com isso, o objetivo geral é desenvolver uma plataforma *web* para o gerenciamento das finanças pessoais, permitindo o registro e a visualização dos ganhos e gastos, promovendo maior controle e organização.

Por tanto, foram definidos os seguintes objetivos específicos: Investigar os principais desafios enfrentados nas finanças pessoais, elicitar os requisitos necessários para a construção do *software*, fazer modelagens das telas, fazer modelagem do banco de dados, desenvolver, testar e documentar a *API*.

2. Fundamentação Teórica

Para construir um sistema *web* para o gerenciamento de finanças pessoais, é de suma importância compreender os conceitos que fazem parte do objetivo do projeto. Este capítulo refere-se a fundamentação teórica, que reunirá conceitos e estudos utilizados no tema do trabalho. A estrutura deste tópico está organizada em dois subtópicos: arquitetura de *software* e trabalhos correlatos.

2.1. Arquitetura de Software

Em consonância com o referido tema, a principal arquitetura de *software* utilizada no sistema criado foi a *Representational State Transfer (REST)*, a qual é descrita pelas autoras Gowda e Gowda (2024) como uma forma arquitetural simples e portátil, frequentemente utilizada em sistemas *webs* por conta de sua boa compatibilidade com o protocolo *HTTP*.

Dado o contexto anterior e os benefícios citados do recurso descrito, optou-se pela utilização de tal modalidade, implementando a *API RESTful*, que utiliza endereços de endpoint e verbos *HTTP* para acessar e manipular recursos entre o cliente e o servidor. Por conseguinte, os verbos, também chamados de métodos, são responsáveis por indicar a ação que deve ser executada para um determinado recurso (MOZILLA, 2024). Os principais métodos utilizados no projeto foram *GET, POST, PATCH* e *DELETE*. Com isso, foi possível a criação de um padrão de fácil entendimento e manutenção.

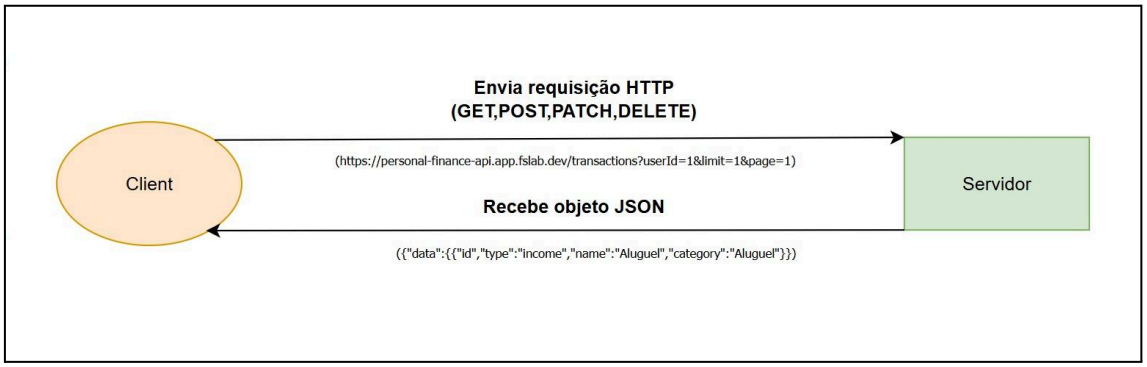


Figura 1. Ilustração da comunicação via HTTP entre cliente e servidor

Sob o mesmo ponto de vista apresentado pela Figura 1, o fluxo da plataforma inicia-se no cliente, que para consultar uma transação específica, por exemplo, envia uma requisição utilizando o verbo *GET*, para o endpoint correspondente no servidor, por exemplo, */transactions*?*userID*=1&*limit*=1&*page*=1. O servidor processa a requisição, busca os dados no banco de dados e retorna uma resposta ao cliente, contendo os detalhes do recurso solicitado. Esse modelo de comunicação promove que cada requisição seja independente, facilita a escalabilidade e a manutenção do sistema.

2.2. Trabalhos Correlatos

Esse tópico se refere aos trabalhos utilizados como referência para criar o projeto atual, com a finalidade de se criar um produto com as práticas já utilizadas no mercado moderno. Ao analisar os sistemas que serão citados, apesar de cumprirem as funções propostas pelos mesmo, eles não possuíam algumas funcionalidades disponibilizadas pelo sistema do tema deste artigo. Enfim, a busca e o estudo pelas referências semelhantes do tema é uma prática de suma importância para a criação de um produto atual e funcional.

2.2.1. Sistema Web para Controle Financeiro Pessoal - *Control*

O trabalho de Oliveira (2023) propõe o desenvolvimento de um sistema *web* para controle financeiro pessoal, utilizando *Java* com *framework Spring* para o *back-end* e *Angular* para o *front-end*. De forma semelhante ao presente artigo, o sistema de Oliveira (2023) permite o cadastro de transações, o gerenciamento de contas e a definição de metas financeiras. Em analogia, um diferencial notável do sistema de Oliveira (2023) é a implementação de notificações via *WhatsApp* (utilizando a *API* Twilio) para alertar sobre o vencimento de transações recorrentes.

Contudo, o sistema proposto neste artigo se diferencia em dois aspectos técnicos cruciais para aplicações financeiras. Primeiramente, o *software* implementa uma lógica específica para o cálculo e processamento de transações parceladas, juntamente com cadastros automáticos de transações marcadas como recorrentes. Em segundo, é abordado diretamente o problema de precisão de ponto flutuante em *JavaScript*, adotando a biblioteca *Decimal.js* para garantir a exatidão em todas as operações financeiras, um requisito fundamental que não é citado pelo artigo discutido neste tópico.

2.2.2. Sistema Web para Gestão de Finanças Pessoais - Finanças

Souza e Armellina (2023) desenvolveram um website para gestão de finanças pessoais com foco em ser uma opção alternativa aos cadernos e planilhas. A aplicação, desenvolvida em *PHP* com *framework Laravel*, permite ao usuário cadastrar, editar e excluir receitas e despesas, além de visualizar gráficos de fluxo de caixa e cadastrar objetivos (metas).

Em comparação, a plataforma apresentada neste trabalho possui algumas funcionalidades há mais do que o projeto desse subtópico, como a automação de transações recorrentes e a exportação de extratos, tais funcionalidades foram apresentadas como trabalhos futuros pelo trabalho correlato, porém elas já existem no *Financial Record*. Dessa forma, o sistema deste artigo apresenta uma solução mais completa, incluindo também o tratamento de transações parceladas, que não é abordado por Souza e Armellina (2023).

3. Metodologia

A metodologia adotada para o desenvolvimento foi baseada no *Kanban*, descrito por Anderson (2011) como um sistema virtual utilizado no desenvolvimento de software para limitar o trabalho-em-progresso (*WIP*) e sinalizar a capacidade de gerar novas tarefas. De acordo com Pressman (2016), a engenharia de software ágil une ideias voltadas para a satisfação do cliente e entrega incremental com princípios que priorizam a adaptação às mudanças em vez de seguir rigidamente os planos. O *Kanban* realiza essa perspectiva ao possibilitar uma administração visual e dinâmica das atividades, com ênfase na entrega constante e na diminuição do tempo de ciclo (*lead time*). De acordo com a definição de Pressman (2016), que afirma que a agilidade exige estruturas de processo adaptáveis às necessidades da equipe e do projeto. O uso do *Kanban* possibilitou a limitação do trabalho em andamento, favorecendo um ritmo sustentável e a identificação rápida de gargalos. Isso garantiu que o *software* evoluísse de maneira fluida e responsiva às demandas de finanças pessoais.

3.1. Tecnologias utilizadas

Inicialmente, no *back-end*, utilizou-se o *Express* sobre o ambiente de execução *Node.js*, que facilita o gerenciamento dinâmico de requisições *HTTP* em diferentes *URLs* (MDN Web Docs, 2025). No *front-end*, foi empregado o *Next.js*, framework baseado em *React*, responsável pela construção das interfaces. O banco de dados escolhido foi o *MySQL*, por ser relacional devido a forma como os dados são relacionados. Por fim, serão apresentadas ferramentas que foram utilizadas para o desenvolvimento de determinadas funcionalidades presentes no projeto.

3.1.1. Decimal.js

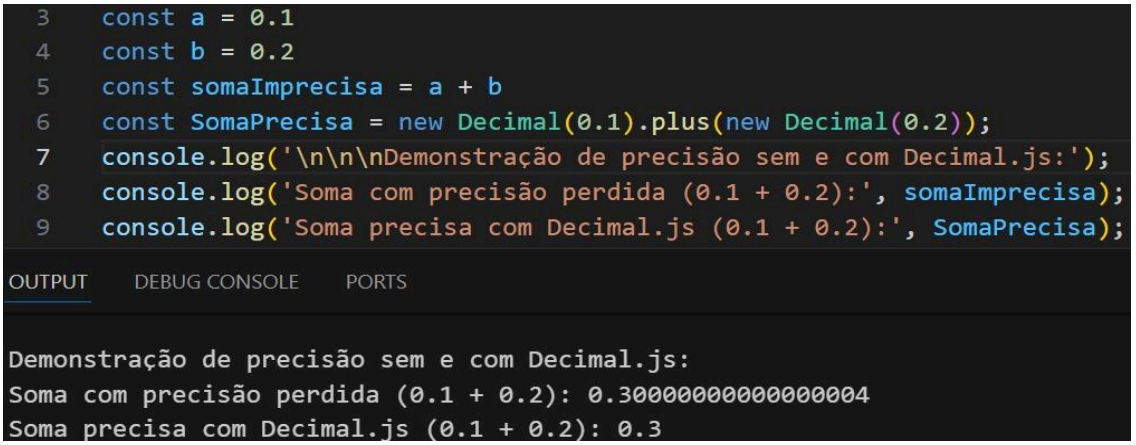
No desenvolvimento do sistema de gerenciamento de finanças, a precisão dos cálculos é um requisito obrigatório. Contudo, o tipo primitivo *Number* do *JavaScript* apresenta limitações conhecidas, que podem provocar erros de arredondamento.

> A representação em ponto flutuante IEEE-754 utilizada em JavaScript(e por praticamente todas as outras linguagens de programação modernas) é uma representação binária que pode descrever frações como 1/2, 1/8 e 1/1024 com exatidão. Infelizmente, as frações que usamos mais comumente (especialmente ao executarmos cálculos financeiros) são decimais: 1/10,

1/100, etc. As representações em ponto flutuante binárias não conseguem representar números simples como 0.1 com exatidão (Flanagan, 2013, p. 34). Em consequência dessa limitação, optou-se pela biblioteca *decimal.js* para lidar com as operações financeiras, garantindo a exatidão decimal.

O Quadro 1 a seguir ilustra a do ponto flutuante: uma operação simples de adição (0.1 + 0.2) resulta em uma dízima imprecisa (0.30000000000000004) devido a erros de arredondamento binário. A mesma figura demonstra que a utilização da biblioteca *decimal.js* mitiga esse problema, garantindo o resultado exato (0.3) através de operações de precisão arbitrária.

Quadro 1. Ilustração do cálculo sem precisão e com precisão



Tendo em vista o exemplo anterior, o Quadro 2 ilustra a aplicação prática do *decimal.js* no sistema, especificamente na função *calculateInstallmentValues*, responsável por calcular os valores de transações parceladas. Para garantir a precisão do cálculo, o valor total da transação *validTransaction.value* é primeiramente instanciado como um objeto Decimal. Em seguida, métodos da própria biblioteca, como o *dividedBy*, são utilizados para realizar a divisão pelo número de parcelas. Ademais, o método *toDecimalPlaces* é aplicado para limitar o resultado em duas casas decimais, assegurando um controle de arredondamento explícito. Essa abordagem impede a ocorrência dos erros de ponto flutuante da aritmética binária padrão do *JavaScript*, o que justifica a escolha da ferramenta para o projeto.

Quadro 2. Ilustração do cálculo interno das transações parceladas

```
/**
 * @private
 * @_calculateInstallmentValues Calcula os valores das parcelas para transações parceladas
 */
static _calculateInstallmentValues(validTransaction) {
  if (validTransaction.number_installments && validTransaction.number_installments > 1) {
    const totalValue = new Decimal(validTransaction.value);
    const numberOfInstallments = validTransaction.number_installments;
    const baseInstallmentValue = totalValue.dividedBy(numberOfInstallments);
    const standardInstallmentValue = baseInstallmentValue.toDecimalPlaces(2, Decimal.ROUND_DOWN);

    validTransaction.value_installment = standardInstallmentValue.toNumber();

    if (validTransaction.current_installment === undefined || validTransaction.current_installment === null) {
      validTransaction.current_installment = 1;
    }
  }
}
```

3.1.2. Cron

O sistema implementa a funcionalidade de automação para transações recorrentes, como salários ou despesas fixas. Com isso, para se automatizar essa situação, optou-se pela biblioteca cron, que permite o agendamento de tarefas (*jobs*) com base em intervalos de tempo predefinidos. Assim, essa biblioteca foi implementada porque ela oferece uma maneira de automatizar o cadastro das transações, eliminando a necessidade de registro manual repetitivo.

Dito isso, para demonstrar a implementação do agendamento, o Quadro 2 apresenta a utilização da classe *CronJob*, importada da biblioteca *cron*. A configuração do agendamento é estabelecida no primeiro parâmetro do construtor por meio de uma expressão *cron* de seis campos (*/2 * * * * *), a qual define o tempo de execução para intervalos de dois segundos. Logo em seguida, há uma função, responsável por conter a lógica do processo a ser automatizado, que no sistema final, seria a execução automática das funções de transações recorrentes e parceladas. Por último, é possível notar como funciona a execução do *script* no terminal do ambiente de execução.

Quadro 3. Ilustração do script do cron

```
JS cron.js > ...
1   import { CronJob } from 'cron';
2
3   console.log('\n\n\n--- Iniciando Exemplo de Cron ---');
4   const job1 = new CronJob(
5       '*/2 * * * * *',//segundo, minuto, hora, dia do mês, mês, dia da semana
6       function () {
7           console.log('Exemplo 1: Executando a cada 2 segundos - ' + new Date().toLocaleTimeString());
8       },
9       null, // onComplete, indica apenas que não há função de conclusão aqui
10      true, // Inicia o job instantaneamente  ao rodar o arquivo.
11      'America/Sao_Paulo'
12  );

OUTPUT   DEBUG CONSOLE   PORTS

--- Iniciando Exemplos de Cron ---
Exemplo 1: Executando a cada 2 segundos - 11:22:40
Exemplo 1: Executando a cada 2 segundos - 11:22:42
Exemplo 1: Executando a cada 2 segundos - 11:22:44
Exemplo 1: Executando a cada 2 segundos - 11:22:46
Exemplo 1: Executando a cada 2 segundos - 11:22:48
Exemplo 1: Executando a cada 2 segundos - 11:22:50
Exemplo 1: Executando a cada 2 segundos - 11:22:52
Exemplo 1: Executando a cada 2 segundos - 11:22:54
--- Exemplo finalizado ---
```

Conforme será ilustrado no Quadro 2, a biblioteca é configurada para executar a rotina de verificação diariamente à 1:00 da manhã ("0 1 * * *"). A justificativa para essa rotina diária é que ela processa não apenas as transações recorrentes *processRecurringTransactions*, mas também as transações parceladas *processInstallmentsTransactions*, que exigem um processamento contínuo em datas futuras. Dessa forma, ao iniciar o serviço com *job.start*(), ambas as lógicas de automação são executadas de forma automática.

Quadro 4. Ilustração do script do cron no sistema do projeto

```
const job = new CronJob("0 1 * * *", async () => {
  await TransactionService.processRecurringTransactions();
  await TransactionService.processInstallmentsTransactions();
});
job.start();        You, 3 months ago • FEAT: Foram feitos os ca
```

3.2. Kanban

Segundo Bessa e Arthaud (2018), as metodologias ágeis emergiram como uma alternativa mais apropriada em comparação com os modelos tradicionais, considerados inflexíveis. Essas metodologias visam proporcionar agilidade na resposta e flexibilidade na adaptação ao processo de desenvolvimento de *software*, possibilitando entregas mais rápidas e de melhor qualidade. Os principais benefícios que incentivam sua adoção incluem a busca por aprimoramentos nos processos, agilidade na entrega do produto, aumento da produtividade e maior colaboração entre os departamentos de negócio e tecnologia (Costa Júnior e Nunes, 2023).

Para a organização das tarefas deste projeto, a metodologia *Kanban* foi implementada utilizando a ferramenta *GitHub Projects*. Nele as tarefas são vinculadas como *Issues* no repositório do projeto e, logo em seguida, vinculadas a colunas do

Github Projects. Essas colunas organizam as tarefas em etapas: *To Do* (A Fazer), *In Progress* (Em Andamento), *Review* (Revisão) e *Done* (Concluído). Com isso, essa estrutura permitiu que fosse visualizado o andamento de cada tarefa, permitindo entender como estava o andamento da construção de cada objetivo proposto.

Por último, foram coletadas métricas para avaliar a eficiência do desenvolvimento, considerando o período em que o trabalho foi iniciado em 5 de maio e finalizado em 23 de novembro. Nesse contexto, os gráficos Figura 1, Figura 2 e Figura 3 mostram, respectivamente,as seguintes métricas: O *Lead Time* é o intervalo total desde a solicitação até a entrega da tarefa, o *Cycle Time* como o tempo de trabalho efetivo que durou a tarefa e o *Throughput* como a taxa de finalização das entregas por período entre "à fazer" e "Concluído". Dito isso, as métricas desses indicadores revelam um *Lead Time* e *Cycle Time* com mediana de 5,0 dias, demonstrando um ritmo de atendimento consistente para a maioria das demandas, e no *Throughput* há uma mediana de 2,5 issues concluídas por semana. Embora os dados apresentem um período de tempo de certa forma ideal, houve tarefas que exigiam um tempo de resolução superior à média devido à complexidade técnica.

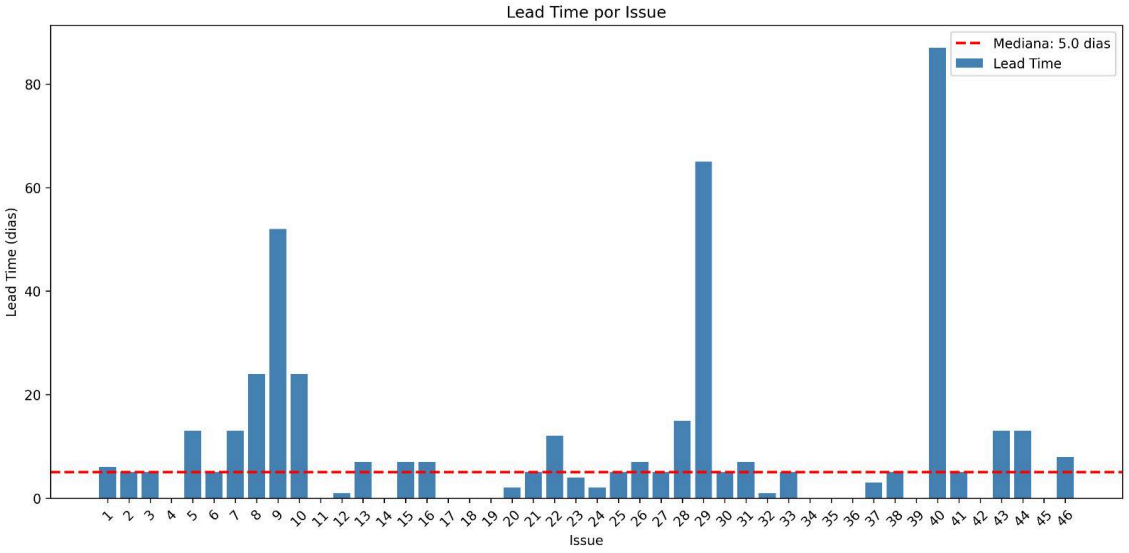


Figura 2. Gráfico Lead Time

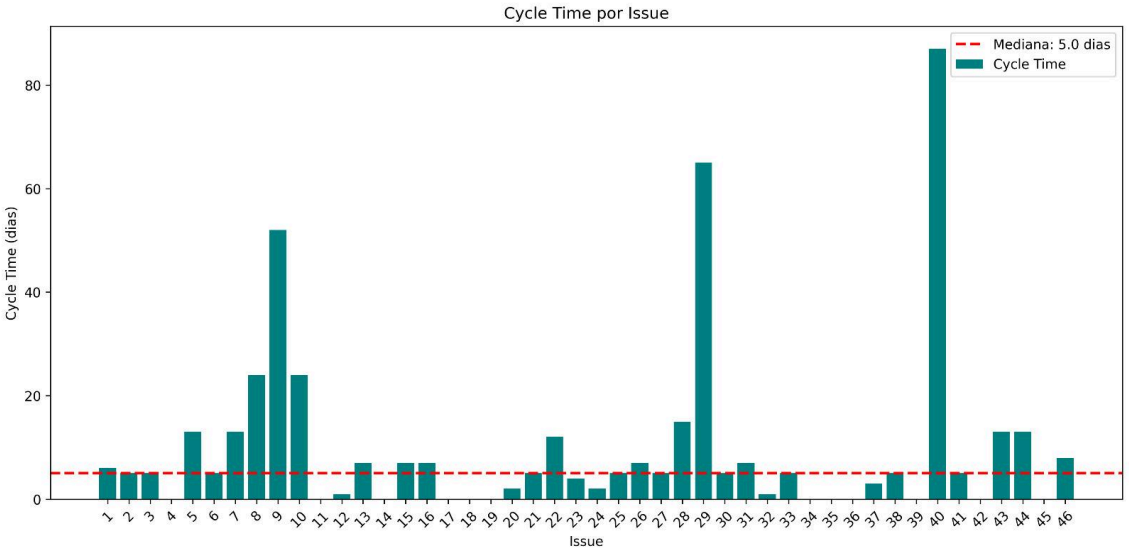


Figura 3. Gráfico Cycle Time

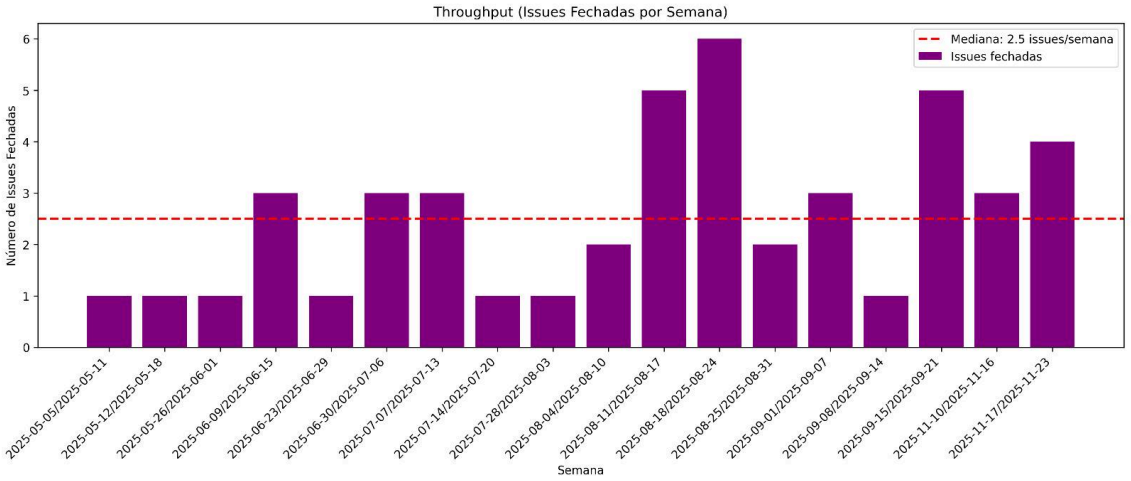


Figura 4. Gráfico Throughput

4. Desenvolvimento

Primordialmente, deve-se descrever as etapas do processo de construção do sistema apresentado neste artigo. Dito isso, o desenvolvimento do produto se iniciou pela modelagem da estrutura do banco de dados do sistema, a qual foi elaborada utilizando a ferramenta *MySQL*. Após a definição do modelo relacional, foi implementado o prisma.io, que consiste em um *Object-Relational Mapper(ORM)* responsável por traduzir consultas *SQL* para estruturas compreensíveis pelo *JavaScript*. Nessa etapa, estabeleceu-se a base de comunicação entre a aplicação e o banco de dados.

Seguindo a lógica de implementação, iniciou-se a construção das rotas, começando pelas rotas básicas de usuários e suas respectivas regras de negócio. Logo depois, foi adicionada a autenticação dos usuários, garantindo o acesso às rotas de forma autenticada. Igualmente, foram inseridos dados de teste para permitir que o sistema pudesse ser utilizado desde o início, simulando usuários reais. Em sequência,

foram construídas as rotas de contas, transferências, transações e metas, cada uma acompanhada de suas regras de negócio específicas e devidamente documentadas utilizando *Swagger*, sendo a documentação criada imediatamente após a implementação de cada rota. Por outro lado, foi desenvolvido um *middleware* de limitação, garantindo que cada usuário tivesse acesso somente às suas próprias informações.

Ademais, houve a configuração das variáveis de ambiente, necessárias para armazenar chaves sensíveis, portas de comunicação, credenciais do banco de dados e tokens de autenticação. Os históricos de atualizações do repositório do projeto demonstram também a implementação do mecanismo de *hash* de senhas utilizando *Bcrypt*. Desse modo, foi possível fazer com que cada usuário tivesse suas senhas criptografadas e o acesso aos seus dados disponível somente após a autenticação com seu *token* de acesso.

Em seguida, o sistema, a parte do *back-end* propriamente dita, passou por melhorias de organização e padronização de código, com a criação de pastas específicas para arquitetura, que separa as responsabilidades da *API* em *controllers*, *services*, *repositories* e *middlewares*, separando responsabilidades e adotando boas práticas de arquitetura. Essa divisão facilitou a manutenção e a expansão futura do *software*, além de permitir que as regras de negócio fossem centralizadas na camada dedicada a ela, o *service*.

Outrossim, foram implementados testes para validação de fluxo e de regras de negócio, garantindo o funcionamento esperado das funcionalidades principais. Também houve, nessa etapa do desenvolvimento, correções de *bugs*, ajustes de validação usando a biblioteca *Zod*, além de melhorias na documentação, tornando o sistema mais precavido com relação às requisições feitas pelo usuário à *API*.

No *front-end*, o desenvolvimento se iniciou com a configuração do ambiente do *framework Next.js*, definindo rotas e páginas principais. Em seguida, foram criados os componentes visuais e, posteriormente, integradas as requisições à *API* do *back-end*, permitindo *login*, cadastro de contas, listagem de transações e criação de metas. Ao longo do processo, há o avanço gradual das telas, a criação de páginas de autenticação, formulários validados, exibição de dados consumidos do servidor e implementação de erros e respostas da *API*, por fim, ambos os serviços (*front-end* e *back-end*) foram ajustados para execução integrada, incluindo a possibilidade de utilização via *Docker*, facilitando a replicação do ambiente e a instalação por outros usuários ou desenvolvedores. Esses procedimentos consolidaram o sistema em um produto funcional, modularizado, autenticado, documentado e capaz de manipular dados financeiros com segurança.

4.1. Diagrama de caso de uso

Com a finalidade de se ter uma orientação de como deverá ocorrer o desenvolvimento do sistema, foram utilizados diagramas *UML*, "*UML* (*Unified Modeling Language*) é uma família de notações gráficas, apoiada por um metamodelo único, que ajuda na descrição e no projeto de sistemas de *software*[...]" (Fowler, 2005, p.25). Dessa forma, a utilização dessas notações forneceu uma base visual clara para a subsequente fase de codificação e implementação das regras de negócio.

Em resumo, o diagrama a ser apresentado agora neste tópico é o de caso de uso (Figura 4), que foi empregado para detalhar o fluxo principal, o qual o usuário irá

percorrer. Segundo Fowler (2005, p.104), os casos de uso são uma técnica narrativa que nos permite especificar as funcionalidades de um *software*, descrevendo a interação entre o sistema e seus usuários. Dessa forma, a figura a seguir demonstra visualmente a principal rota de interação que o sistema de gerenciamento de finanças pessoais oferece.

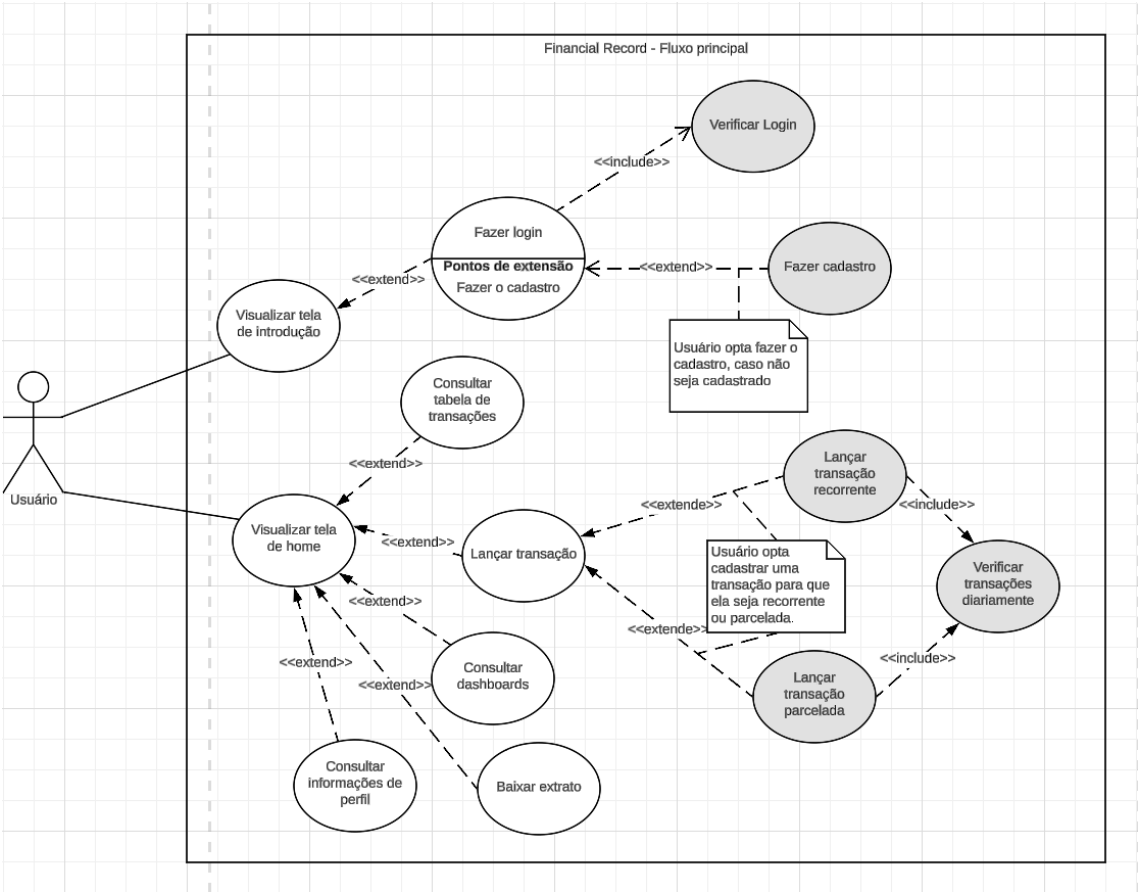


Figura 5. Diagrama de caso de uso

O diagrama de caso de uso (Figura 4) ilustra o fluxo principal do trabalho descrito neste artigo, começando pelo ator “Usuário”. O processo se inicia em “Visualizar tela de Introdução”, que serve como ponto de entrada, que descreve do que se trata o site. A partir desta tela, o usuário tem a opção de iniciar o processo de Fazer *login* ou Fazer o cadastro, conforme indicado pela nota "Usuário” opta fazer o cadastro, caso não seja cadastrado". O caso de uso “Fazer *login*”, por sua vez, aciona obrigatoriamente a funcionalidade “Verificar *Login*” para autenticar o usuário, tal funcionalidade é feita executada pelo próprio sistema.

Uma vez autenticado, o “Usuário” interage com o caso de uso principal “Visualizar tela de *home*”. Esta tela serve como um painel central de onde partem diversas funcionalidades opcionais, incluindo “Consultar tabela de transações”, “Consultar *dashboards*” e “Consultar informações de perfil”. Entretanto, o curso mais importante, a partir da tela inicial, é o de “Lançar transação”, este caso de uso é expansível, permitindo ao usuário optar por “Lançar transação recorrente” ou “Lançar transação parcelada”. Sendo assim, ambos os casos de uso de transações especializadas acionam obrigatoriamente a funcionalidade “Verificar transações diariamente”, que

indica um processo de sistema para processar esses lançamentos de forma agendada, levando em consideração os parâmetros de que foram configurados em cada transação.

4.2. Diagrama de atividade

Por conseguinte, o diagrama de atividades apresentado na Figura 6 foi elaborado para demonstrar o funcionamento do fluxo principal feito pelo usuário no sistema. Segundo a definição de Fowler (2005, p.118), essa notação gráfica é a técnica apropriada para especificar fluxos de trabalho e lógicas de procedimentos complexos, demonstrando o caminho que poderá ser seguido no sistema dependendo da decisão tomada pelo usuário. Dito isso, a figura abaixo demonstra não apenas a interação, mas a ordem cronológica das operações que compõem o sistema de controle financeiro do sistema, garantindo que o processamento das informações ocorra de forma estruturada.

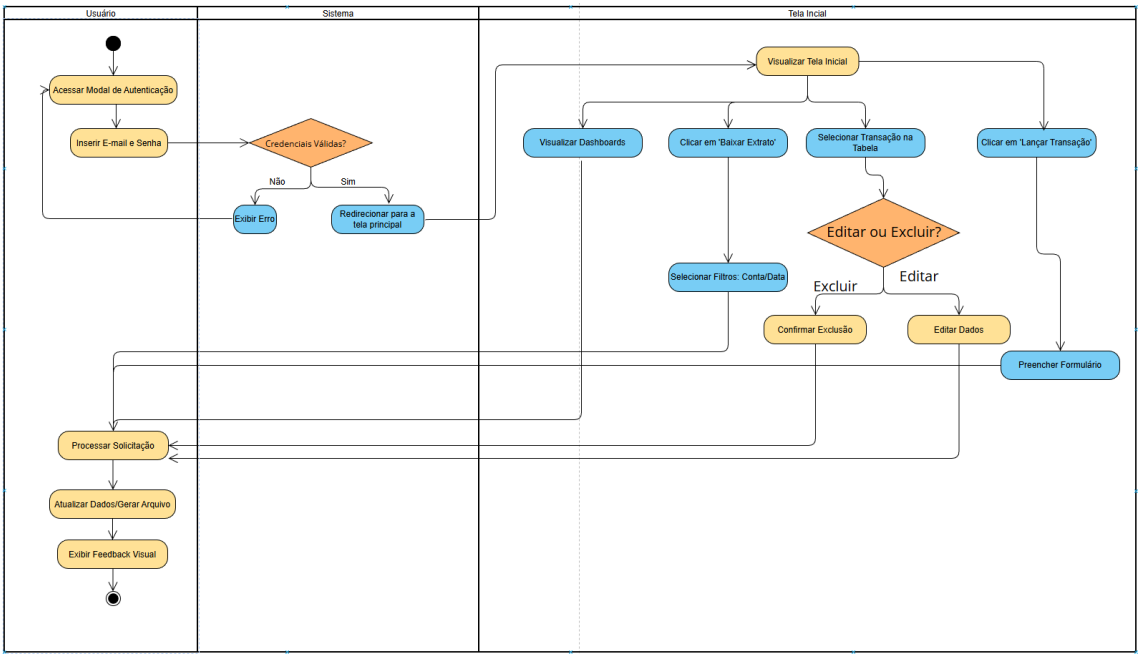


Figura 6. Diagrama de atividades

4.3. Resultados da implementação

Este capítulo apresenta os resultados visuais da implementação do sistema de gerenciamento de finanças pessoais, servindo como validação prática da arquitetura e das tecnologias discutidas nos capítulos anteriores. Contudo, as telas a seguir não são meramente ilustrativas, mas sim a comprovação da funcionalidade do produto final. O fluxo demonstrado se inicia pela tela de autenticação (Figura 5), onde o usuário irá inserir seu *email* e senha e logo após irá adentrar na tela principal (Figura 6).

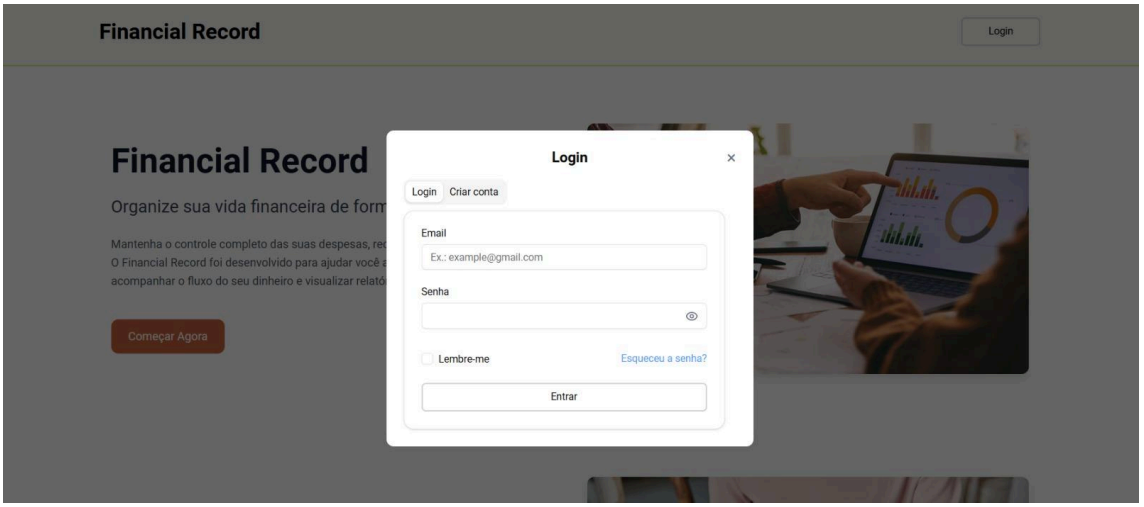


Figura 7. Modal de autenticação

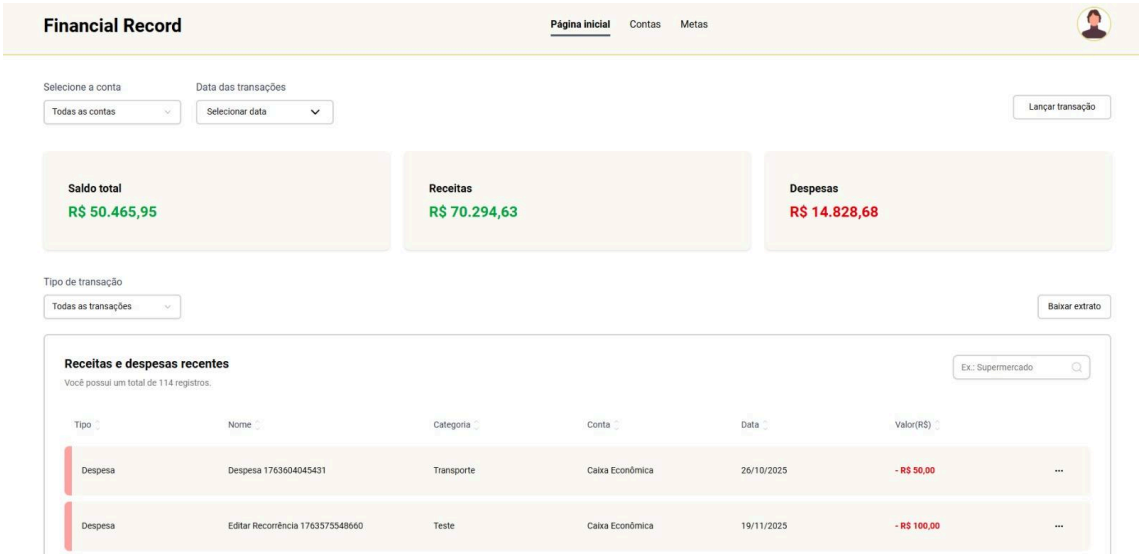


Figura 8. Tela principal

Em seguida, terá o botão de “Lançar transações”, que ao ser pressionado irá abrir um formulário na lateral direita da tela (Figura 6), permitindo ao usuário preencher todas as informações de sua transação, podendo ser recorrente ou parcelada, após lançar a transação, a tabela irá atualizar mostrando o novo lançamento (Figura 7). Após isso, pode ser possível visualizar as informações da transação recém lançada (Figura 8), bem como a sua edição e exclusão (Figura 9) e (Figura 10). Outrora, há o botão de “Baixar extrato”, ao pressioná-lo irá abrir um modal (Figura 8), o qual usuário poderá selecionar certos parâmetros, como, a conta, o período de início e fim, e o tipo de transação que deseja gerar o extrato, logo em seguida baixará todo o histórico de transações com os filtros que foram aplicados.

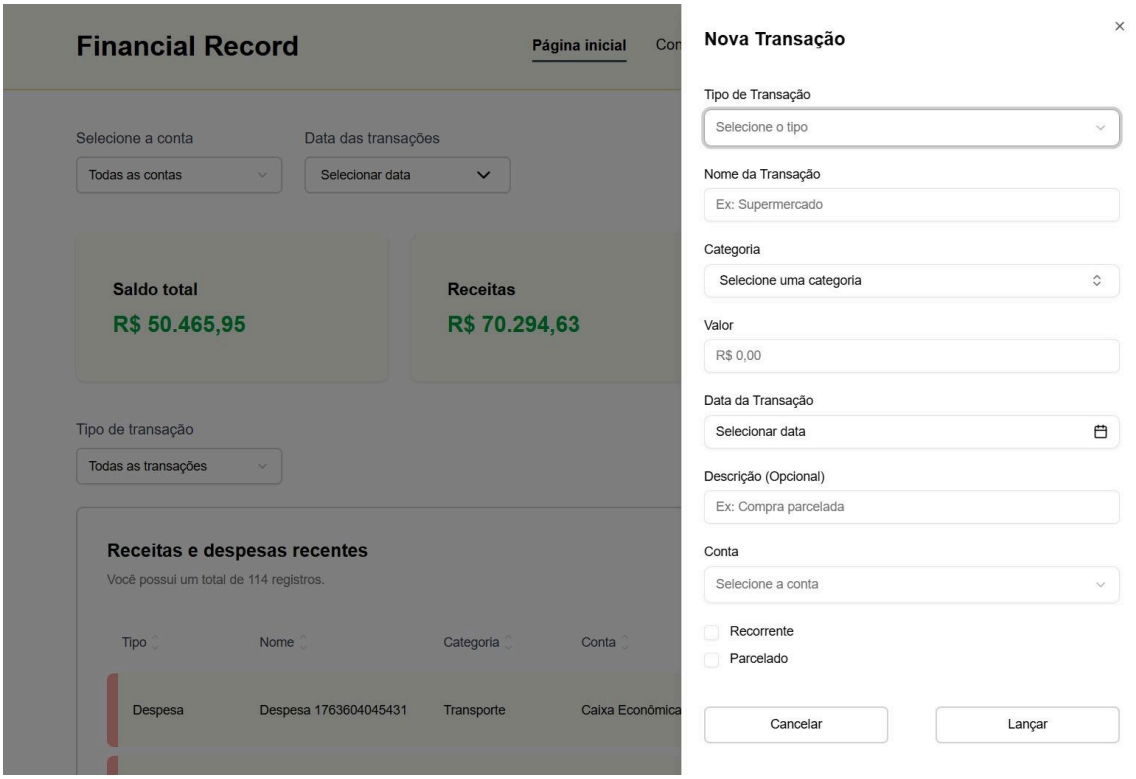


Figura 9. Formulário de uma nova transação

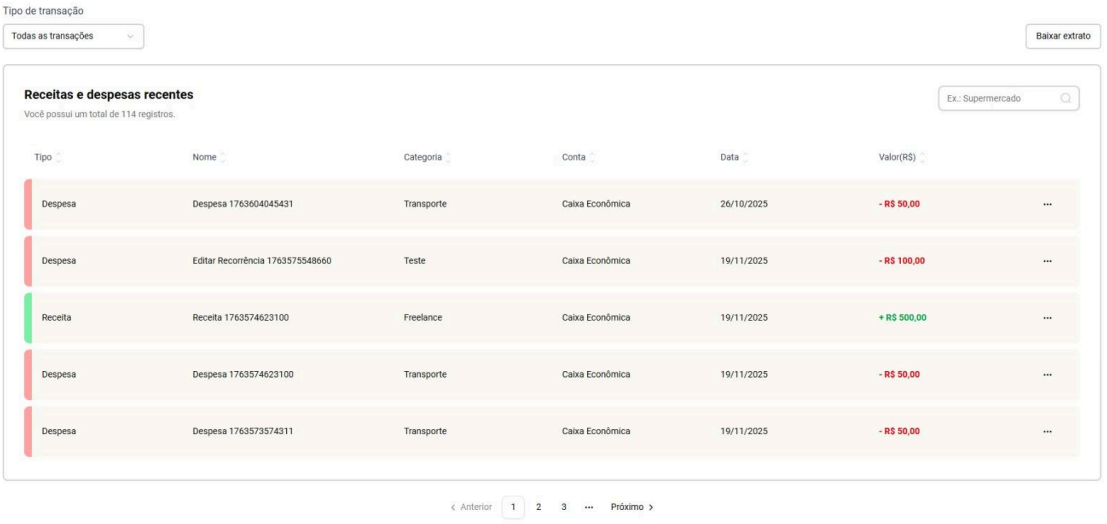


Figura 10. Formulário de uma nova transação

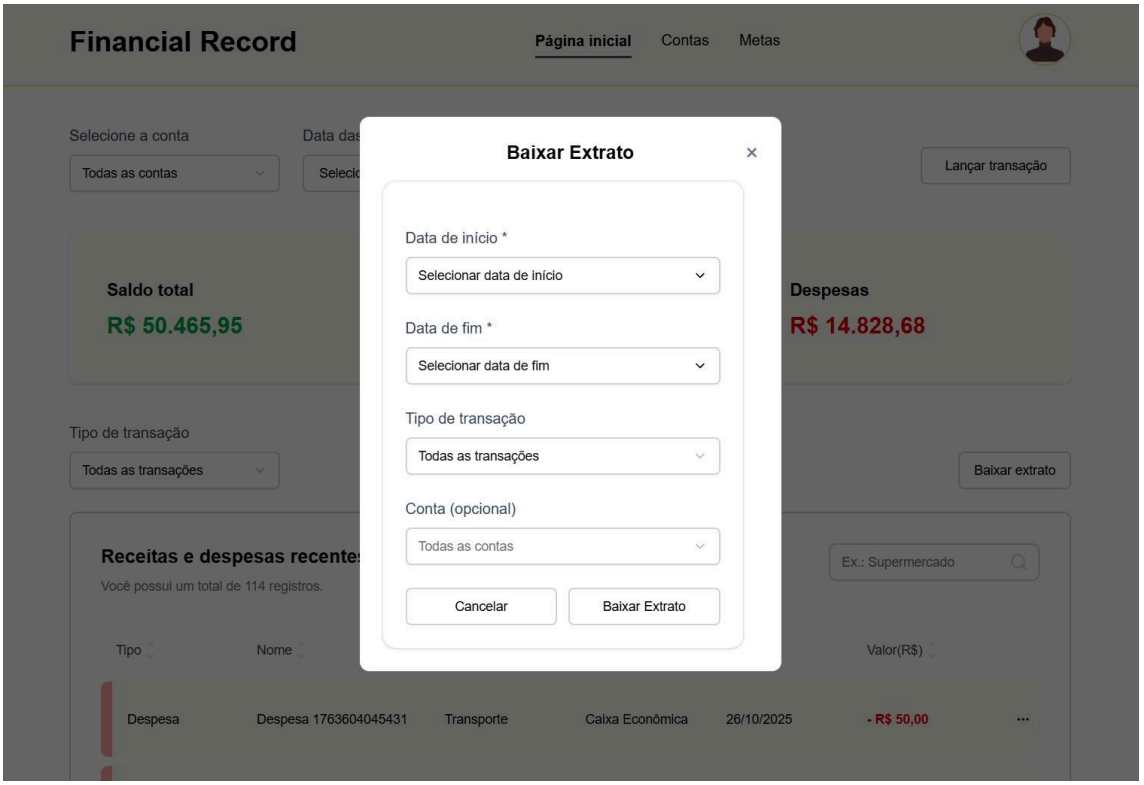


Figura 11. Formulário de uma nova transação

Enfim, o usuário poderá visualizar o resumo de suas transações pelos *dashboards* (Figura 9 e Figura 10), que mostraram um resumo simplificado de suas receitas, despesas e suas respectivas metas com gráficos que utilizam as categorias de cada transação como forma de classificar e filtrar a pesquisa.

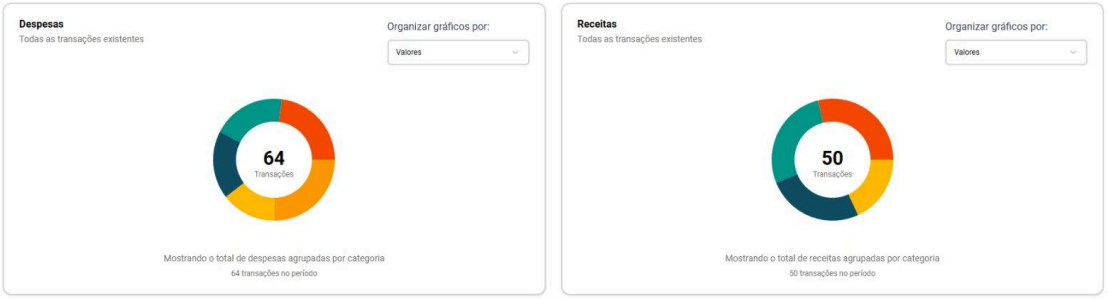


Figura 12. Formulário de uma nova transação

Figura 13. Formulário de uma nova transação

5. Resultados da implementação

Este tópico trata dos resultados que foram atingidos durante a produção deste projeto, detalhando quais requisitos funcionais e não funcionais foram de fato implementados e entregues. A estratégia de testes adotada foi focada no fluxo principal do sistema. Com isso, foram realizados testes de integração para percorrer os módulos essenciais, testes unitários para validar as regras de negócio das transações e testes ponta a ponta para simular o caminho do usuário nas telas da plataforma. Por fim, o capítulo aborda o processo de *deploy* (implementação) do sistema em um ambiente acessível e promove uma discussão comparativa entre a solução finalizada e as expectativas delineadas no início do desenvolvimento.

5.1. Funcionalidades entregues

Esta seção é dedicada à apresentação detalhada dos Requisitos Funcionais e Requisitos Não Funcionais. Primordialmente, os requisitos funcionais são os que guiam o comportamento do sistema em desenvolvimento. Logo, alinhados à definição de Pressman (2016), estes requisitos são compreendidos como o artefato que especifica as funções e serviços que o software deve prover, descrevendo de forma inequívoca o que o sistema fundamentalmente faz. Por conseguinte, eles constituem os critérios centrais para os processos de verificação e validação, servindo como o contrato principal que delimita o escopo de entrega do projeto.

Com isso, o Quadro 1, a seguir, detalha os Requisitos Funcionais (RFs) que definem o escopo do que foi entregue neste projeto. Estes requisitos formam a base para a validação e os testes de aceitação do sistema, onde cada *ID* único permite o rastreamento e a verificação de que o software opera conforme o especificado. Em resumo, o quadro estabelece o que foi especificado e quais eram suas prioridades.

Quadro 5 - Requisitos funcionais

ID	Descrição	Prioridade
RF01	O sistema deve mostrar ao usuário a tela de introdução	Mediana

RF02	O sistema deve permitir ao usuário criar uma conta.	Alta
RF03	O sistema deve permitir que o usuário se autentique.	Alta
RF04	O sistema deve permitir que o usuário adicione receitas ou despesas, sejam elas normais, parceladas ou recorrentes.	Alta
RF05	O sistema deve permitir ao usuário a possibilidade de exportar o histórico de suas transações.	Mediana
RF06	O sistema deve mostrar *dashboards* aos usuários, mostrando dados sobre suas receitas e despesas.	Alta
RF07	O sistema deve permitir ao usuário selecionar filtros como o mês, o tipo de transação e a conta em que foi registrada tal despesa, com a finalidade de organizar a tabela de transações.	Mediana
RF08	O sistema deve permitir que o usuário possa gerenciar suas contas.	Alta
RF09	O sistema deve permitir ao usuário a criação de transferências entre contas.	Alta
RF10	O sistema deve permitir que o usuário possa gerenciar suas metas.	Alta

Em seguida, os requisitos não funcionais (Quadro 2) foram definidos durante a construção desse projeto com a finalidade de implementar os recursos necessários logo no período inicial. Ademais, segundo Pressman (2016), são aqueles que definem os atributos de qualidade, desempenho e segurança de um sistema, ou suas restrições gerais. Em outras palavras, enquanto os Requisitos Funcionais determinam o que o sistema faz, os Não Funcionais determinam como o sistema executa essas funções, estabelecendo os padrões no projeto.

Quadro 6 - Requisitos não funcionais

ID	Descrição	Prioridade
RNF01	O login deve ser processado e devolver uma resposta em menos de 3 segundos.	Mediana
RNF02	O sistema deve criptografar a senha do usuário.	Alta
RNF03	O sistema deve proteger os dados do usuário.	Alta

RNF04	O sistema deve calcular os valores das parcelas e adicionar o restante da divisão à última parcela.	Alta
RNF05	O sistema deve calcular os valores das transações recorrentes e adicionar automaticamente uma nova transação igual à anterior.	Mediana
RNF06	O sistema deve ser capaz de realizar operações matemáticas básicas para gerar gráficos.	Alta

5.2. Testes realizados

Neste subtópico é descrito o processo de criação e a aplicação dos testes no sistema de gerenciamento de finanças pessoais. O objetivo principal da realização desses testes é cobrir o principal fluxo de interação do usuário, validando as regras de negócios, com a finalidade de manter a integridade do sistema, desde funções isoladas até módulos inteiros. Dito isso, as validações dos testes foram centralizadas nos principais fluxos que os usuários, foram empregados, portanto, os testes de unidade, testes de integração e os testes ponta a ponta (*E2E*).

Dado o contexto anterior, conforme Pressman (2016, p. 473), o teste de unidade concentra-se na menor unidade lógica de um projeto de *software*, normalmente um módulo ou componente isolado. Esse tipo de teste busca identificar erros internos, validar comportamentos específicos e garantir que cada parte do sistema funcione corretamente de maneira independente. Em continuidade, os testes de integração têm como propósito verificar a comunicação entre os módulos e assegurar que as funcionalidades interajam de forma consistente. Por fim, os testes E2E simulam o comportamento real do usuário dentro do sistema, analisando o funcionamento completo de um fluxo, desde a entrada até a saída final.

Para a implementação dos testes foram utilizadas as ferramentas disponibilizadas para serem feitos os testes em ambiente com linguagem *javascript* para lidarem com testes de unidade, integração e ponta a ponta respectivamente: *Jest, Supertest e Cypress*. Também, o ambiente de testes utilizou uma instância isolada do banco de dados, garantindo que a execução dos testes não afetasse dados reais e permitindo a recriação do estado inicial a cada execução.

A princípio, os testes de unidade focaram na validação de entrada de dados com *Zod*, cálculo de saldo, cálculo de transações parceladas e cálculo de transações recorrentes, passando pelos métodos específicos de cada funcionalidade. O uso de *mocks* evitou dependência direta do banco e permitiu verificar a lógica interna isoladamente. Em contrapartida, os testes de integração verificaram, principalmente, as rotas de contas, transações, transferências entre contas e metas. Esses testes foram implementados com *Jest* e *Supertest*, validando resposta, estrutura de dados e códigos *HTTP*. Por último, os testes *E2E* foram realizados com *Cypress* e reproduziram o fluxo real de um usuário, incluindo login com credenciais válidas, cadastro de uma nova

transação, verificação do saldo atualizado, edição e remoção de transações e exportação de extratos.

Por fim, os resultados obtidos demonstram eficácia dos testes no sistema. A cobertura geral de código atingiu 88,62% das instruções e 80,5% das *branchs*, com destaque para os serviços de usuários e transações, que superaram 92% de cobertura. Além disso, os testes ponta a ponta validaram 54 cenários, cobrindo fluxos como autenticação e transações em cerca de sete minutos, com 100% de aprovação. Com isso, garante-se a estabilidade e o funcionamento correto das regras de negócio implementadas.

5.3 Deploy

O processo de *deploy* do sistema foi estruturado com base em integração e entrega contínua (*CI/CD).* Para isso, foi utilizado o *GitHub* como plataforma de versionamento e automação. Através do *GitHub Actions*, cada atualização enviada ao repositório executava *pipelines* responsáveis por executar testes, criar uma imagem espelhada do produto com *docker* e publicar o projeto no *kubernetes*. Esse fluxo garantiu que eventuais falhas fossem identificadas antes da disponibilização do sistema no ambiente final.

Ademais, para padronizar o ambiente de execução e garantir portabilidade, toda a aplicação foi espelhada utilizando *Docker*. Foram criadas imagens específicas para o *back-end*, permitindo que o sistema pudesse ser executado de maneira consistente em qualquer infraestrutura compatível. Durante o desenvolvimento local, foi utilizado *Docker Compose*, responsável por orquestrar múltiplos contêineres de forma simples. Essa abordagem permitiu subir simultaneamente a *API* e o banco de dados, reproduzindo um ambiente muito próximo ao de produção.

O *deploy* final foi realizado na infraestrutura *Kubernetes* fornecida pela instituição de ensino. Através de configurações específicas do ambiente, os contêineres gerados pelos *pipelines* foram distribuídos e executados em pods, garantindo escalabilidade, tolerância a falhas e gerenciamento automatizado dos serviços. Além disso, o *Kubernetes* facilitou o controle de recursos, o balanceamento de carga e a reinicialização automática em caso de falhas, consolidando um ambiente robusto para operação do sistema.

6. Conclusão

Este trabalho surgiu a partir da problemática que muitos brasileiros enfrentam no gerenciamento de suas finanças pessoais, um cenário com alto índice de endividamento e pela dependência de métodos manuais ou inexistentes de controle. O objetivo principal foi projetar e desenvolver um sistema *web* de gerenciamento financeiro, o *Financial Record*, que simplifica o monitoramento da condição financeira do usuário. Para alcançar este objetivo, foi adotada a metodologia *Kanban*, com o desenvolvimento de um *back-end* em *Express* e um front-end em *Next.js*.

O sistema desenvolvido oferece contribuições nas áreas de finanças pessoais e desenvolvimento de *softwares* com cunho financeiro. Dito isso, no ramo de finanças pessoais, o produto dispõe de funcionalidades como o cadastro de transações de receitas e despesas do usuário, juntamente com a definição de metas financeiras e o controle dos saldos de suas contas. Por outro lado, no ramo de desenvolvimento de *softwares* com foco financeiro, este trabalho demonstra a utilização de ferramentas que ajudam no cadastro, gerenciamento e visualização de gráficos dinâmicos, servindo como referência para a criação de novos produtos e soluções semelhantes.

Inicialmente, com relação às limitações encontradas, o escopo do projeto restringiu-se às funcionalidades essenciais de um gerenciamento financeiro individual, delimitadas pelo tempo hábil de desenvolvimento e pela priorização dos requisitos funcionais fundamentais. Adicionalmente, por se tratar de uma arquitetura baseada em uma aplicação *web*, a dependência de uma conexão estável com a *internet* para a comunicação com a *API* e o banco de dados constitui uma restrição operacional, impossibilitando a utilização estando no modo *offline*.

Por fim, propõe-se como trabalhos futuros o desenvolvimento de uma versão em aplicativo móvel android e *IOS*, o que facilitaria o acesso rápido. Também, outra melhoria possível seria a implementação de módulos para a leitura e importação de arquivos em formatos *CSV* e *PDF*, permitindo o cadastro de múltiplas transações a partir de extratos bancários, reduzindo drasticamente o esforço manual de inserção de dados.

7. Referências

ANDERSON, David J. **Kanban**: mudança evolucionária de sucesso para seu negócio de tecnologia. Tradução: Andrea Pinto. Sequim, Washington: Blue Hole Press, 2011. ISBN 978-0-9845214-6-3.

(2019). IEEE Std 754-2019: IEEE Standard for Floating-Point Arithmetic. IEEE, New York. Institute of Electrical and Electronics Engineers.

Bessa, T. and Arthaud, D. D. B. (2018). Metodologias ágeis para o desenvolvimento de softwares. Ciência e Sustentabilidade - CeS, 4(2):173–213. Disponível em: https://periodicos.ufca.edu.br/ojs/index.php/cienciasustentabilidade/article/view/314. Acesso em: 20 set. 2025.

Confederação Nacional do Comércio de Bens, Serviços e Turismo (2023). Balanço do endividamento e da inadimplência do consumidor brasileiro em 2023. Disponível em: https://portaldocomercio.org.br/publicacoes_posts/pesquisa-de-endividamento-e-inadimplencia-do-consumidor-peic-perfil-do-endividamento-anual-2023/. Acesso em: 10 nov. 2025.

Confederação Nacional do Comércio de Bens, Serviços e Turismo (2025). Pesquisa de endividamento e inadimplência do consumidor (peic): Edição janeiro 2025. Disponível em:
https://portaldocomercio.org.br/publicacoes_posts/pesquisa-de-endividamento-e-inadimplencia-do-\\consumidor-\\peic-janeiro-de-2025/. Acesso em: 10 nov. 2025.

Costa Junior, R. A. and Nunes, T. S. (2023). O impacto da transformação ágil na cultura organizacional: das práticas e valores organizacionais à gestão da mudança. Perspectivas em Ciência da Informação, 28. DOI: http://dx.doi.org/10.1590/1981-5344/29487. Dispon'ıvel em: https://www.scielo.br/j/pci/a\\/P7Bp6tdMHkw98mHFBZf9wXN/abstract/?lang=pt. Acesso em: 20 set. 2025.

de Oliveira, C. P. (2023). Sistema web para controle financeiro pessoal. Trabalho de conclusão de curso (tecnólogo em análise e desenvolvimento de sistemas), Universidade Tecnológica Federal do Paraná, Pato Branco.

de Souza, J. F. and Dall'Armelina, G. L. (2023). Sistema web para gestão de finanças pessoais. Trabalho de conclusão de curso (graduação em ciência da computação), Faculdade de Tecnologia, Ciências e Educação.

Flanagan, D. (2013). JavaScript: o guia definitivo. Bookman, Porto Alegre, 6 ed.

Fowler, M. (2005). UML Essencial: Um breve guia para a linguagem-padrão de modelagem de objetos. Bookman, Porto Alegre, 3 ed.

Gowda, P. and Gowda, A. N. (2024). Best practices in rest api design for enhanced scalability and security. Journal of Artificial Intelligence, Machine Learning and Data Science, 2(1):827–830. DOI: doi.org/10.51219/JAIMLD/priyanka-gowda/202. Acesso em: 1 out. 2025.

MDN Web Docs (2024). Métodos de requisição http. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods. Acesso em: 7 out. 2025.

Pressman, R. S. e Maxim, B. R. (2016). Engenharia de software: uma abordagem profissional. Bookman, Porto Alegre, 8 ed. Tradução: Ariovaldo Griesi; Mario Moro Fecchio.