



Antonio Wictor Pereira de Sousa

Diogo Di Torres Alexandre

Gisele Gomes Costa

Nivea Hayane Gomes Miranda

Rafael Alves Rodrigues

Victor Manoel Pinheiro Coutinho

Documentação Técnica - Projeto Integrador I

Projeto e modelagem de um sistema de controle financeiro pessoal: Poupaê

**Crateús, Ceará
2025**

SUMÁRIO

SUMÁRIO.....	2
1. INTRODUÇÃO.....	4
1.1 Apresentação Geral do Sistema.....	4
1.1.1 Tema e Propósito do Sistema.....	4
2. Análise de Requisitos.....	4
2.1. Requisitos Funcionais.....	4
2.2. Requisitos Não Funcionais.....	6
3. Modelos UML.....	7
3.1 Diagrama de Casos de Uso.....	8
3.1.1. Casos das Principais Funcionalidades do Usuário:.....	8
3.1.2. Diagrama de Casos de Uso dos Requisitos Funcionais:.....	8
4. Casos de Uso Detalhados.....	10
4.1. Caso de Uso: Inserir Despesa.....	10
4.2. Caso de Uso: Visualizar Gráficos.....	10
5. Arquitetura e Stack.....	10
5.1 Detalhes da stack e integrações - Back-End:.....	10
5.2 Detalhes da stack e integrações - Front-End:.....	11
5.3 Comunicação entre Camadas.....	11
6. Organização do Código.....	11
6.1 Arquivos e Funções.....	11
7. Estrutura do Banco de Dados.....	12
7.1. Visão Geral.....	12
7.2. Modelo Entidade-Relacionamento (ER).....	13
7.3. Tabela: Usuarios.....	13
7.4. Tabela: Categoria_Receita.....	13
7.5. Tabela: Categoria_Gasto.....	13
7.6. Tabela: Receitas.....	13
7.7. Tabela: Gastos.....	13
7.8. Modelagem do cadastro e login.....	13
8. Regras de Negócio Implementadas.....	15
8.1. Cálculo de Saldo Total.....	15
8.2. Geração do Gráfico de Pizza.....	15
8.3. Cálculo da Porcentagem Gasta.....	16
8.4. Validações de Cadastro.....	16
9. Tratamento de Erros e Feedback ao Usuário.....	16
10. Protótipo de alta fidelidade no Figma.....	17
10.1. Telas Desenvolvidas.....	17
10.2. Fluxo de Navegação.....	17
10.2.1. Representação do fluxo entre telas.....	18
11.3. Responsividade.....	18
11.4. Elementos Reutilizáveis.....	18
11.5. Alinhamento com o Front-End.....	19

11.6. Padrões Visuais.....	19
11.7. Anexos -imagens das telas.....	19
12. Teste de Software.....	22
12.1. Ambiente de Teste.....	23
12.2 Casos de Teste.....	23
12.2.1. Módulo 1: Autenticação (app.py).....	23
12.2.2. Módulo 2: Dashboard e Transações (main.py e view.py).....	24
12.2.3. Módulo 3: Persistência de Dados e Isolamento.....	26
12.3 Anexos.....	26
13. Guia de geração do executável (build).....	33
14. Formulário sobre a avaliação do software.....	35
15. Hospedagem do Código.....	37

1. Introdução

Esta seção oferece uma visão geral do Poupaê, sistema de controle financeiro, abordando a apresentação geral do sistema, que inclui o tema e a problemática, bem como os objetivos do trabalho.

1.1 Apresentação Geral do Sistema

1.1.1 Tema e Propósito do Sistema

Através da percepção coletiva dos alunos da Universidade Federal do Ceará, diversos deles têm que arcar com despesas como aluguel, transporte intramunicipal, alimentação e entre outros gastos pessoais, o que exige um controle e um bom planejamento financeiro. Logo, o sistema a ser desenvolvido tem como foco permitir um controle detalhado das receitas, despesas e rendas a fim de facilitar o planejamento de gastos e metas financeiras do aluno. A aplicação oferece o resumo mensal, por meio de recursos visuais, permitindo uma melhor compreensão da situação financeira do usuário ao longo do tempo.

2. Análise de Requisitos

Nesta seção, são detalhados os requisitos do sistema, divididos em funcionais e não funcionais. Esses requisitos formam a base para o desenvolvimento e a modelagem do sistema.

2.1. Requisitos Funcionais

Código	Requisito Funcional	Descrição Técnica Detalhada
RF01	Autenticação do Usuário	A tela de login deve permitir a entrada do nome de usuário e senha. Ao clicar em "Fazer Login", o sistema deve validar as credenciais. Se estiverem corretas, o acesso será liberado; se não, uma mensagem de erro será exibida.

RF02	Visualizar Senha	O sistema deve permitir ao usuário visualizar ou ocultar o conteúdo digitado no campo de senha.
RF03	Redirecionar para Cadastro	A interface gráfica deve ter um botão onde o usuário poderá clicar e ser redirecionado para a tela de cadastro. O sistema deve permitir a navegação entre a tela de login e a tela de cadastro do usuário.
RF04	Cadastro de novo usuário	O sistema deve aceitar nome de usuário, e-mail, senha e confirmação da senha. Os campos devem ser validados: nenhum pode estar vazio, o e-mail deve ter formato válido e a senha deve estar igual com a confirmação.
RF05	Voltar para Tela de Login	A tela de cadastro possui um botão "Voltar ao Login", que ao clicar ele retorna para a tela inicial de login, onde será necessário inserir as credenciais do RF01.
RF06	Inserir nova Despesa	O sistema deve permitir que o usuário insira uma nova despesa informando categoria (ex.: alimentação, transporte), data (selecionável via calendário) e valor (em formato monetário). O registro será salvo em banco de dados e refletido na tabela e gráficos.
RF07	Inserir nova Receita	Deverá haver um botão para inserir receita, ao ser clicado o usuário deverá inserir o valor, informar a sua categoria (ex: salário, venda, bico, tigrinho), data e clicar em 'adicionar'.
RF08	Exibir tabela detalhada	A partir de uma tabela, deve-se apresentar uma lista com todas as receitas e despesas, com colunas: categoria, data e valor.
RF09	Excluir registros	O sistema deve permitir que o usuário selecione uma linha da tabela e clique em um botão "Remover Selecionado". O dado será removido da lista e os totais e gráficos serão atualizados.
RF10	Calcular total de receitas	O sistema deve somar automaticamente os valores inseridos advindos do RF07 e exibir o resultado no campo de Renda.

RF11	Calcular total de despesas	O sistema deve somar automaticamente os valores inseridos advindos do RF06 e exibir o resultado no campo de Despesas.
RF12	Calcular e exibir o saldo	O saldo deve ser calculado como: total de receitas - total de despesas . Em sincronia com os valores inseridos e tendo como base a última movimentação. Logo, assim como RF10 e RF11, deve exibir o saldo no campo de Saldo.
RF13	Calcular a porcentagem da receita e despesa por categoria	Os valores inseridos para cada receita e cada despesa serão exibidos em um campo com a porcentagem formatada (ex.: 75%), conforme a categoria correspondente.
RF14	Exibir dashboard interativo	O sistema deve exibir um dashboard com gráficos visuais e dinâmicos comparando receitas, despesas e saldos mensais. Os gráficos deverão ser atualizados conforme os dados forem inseridos, editados ou excluídos.

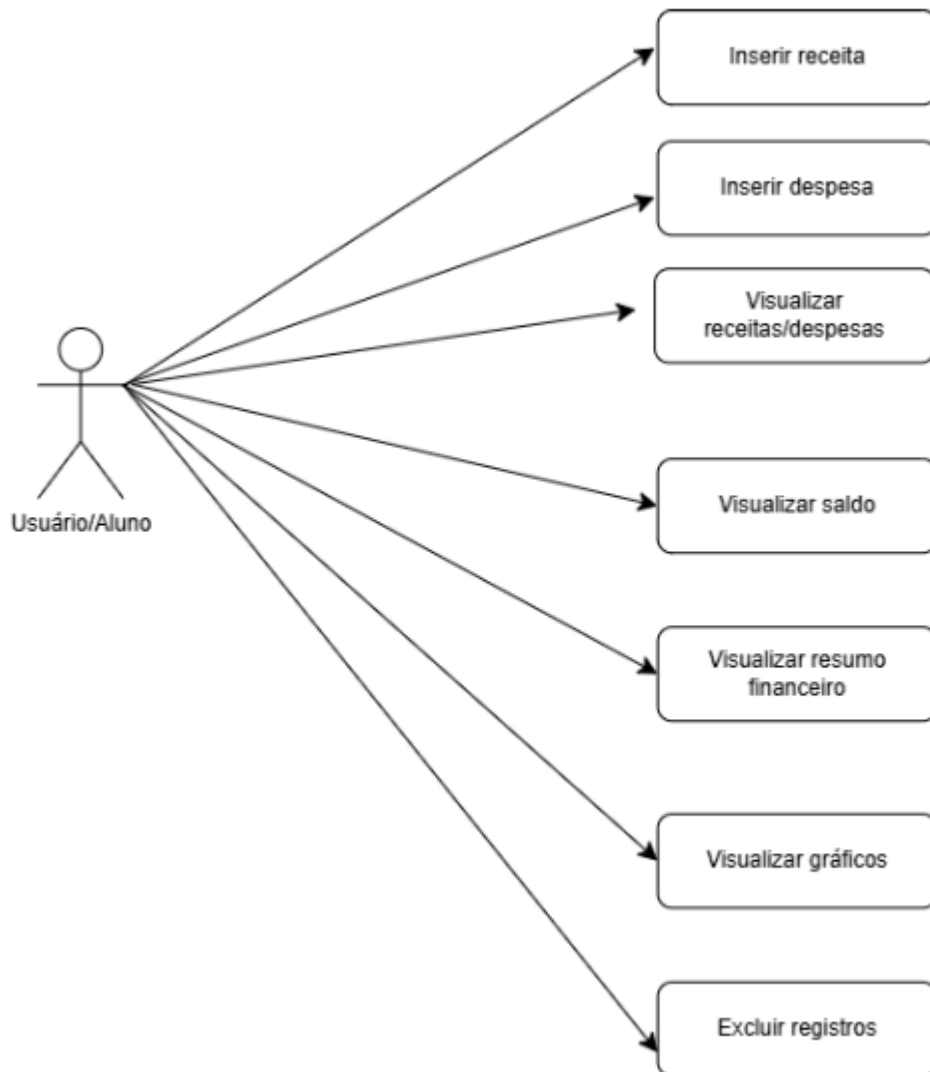
2.2. Requisitos Não Funcionais

Código	Requisito Não Funcional	Detalhamento
RNF01	Validação Básica de Dados	O sistema deve validar os dados inseridos pelo usuário em tempo real, garantindo que campos obrigatórios não fiquem vazios e que os dados estejam no formato correto.
RNF02	Usabilidade	A interface do sistema deve ser intuitiva, com uso de ícones representativos, cores suaves e contrastes adequados para facilitar a leitura e conforto visual para o usuário.
RNF03	Portabilidade	O sistema deve funcionar no Windows, no decorrer do projeto, iremos testar no Linux e no macOS.
RNF04	Desempenho	Em testes realizados em um notebook com processador Intel i5-10300H, 8GB de RAM DDR4 (3200MHz) e placa de vídeo GTX 1650 (4GB VRAM), o sistema foi executado com tempo médio de carregamento de 15 segundos, mesmo com outros aplicativos abertos. O executável principal

		possui cerca de 65,1 MB. O sistema também atualiza os valores e gráficos em tempo real após cada inserção ou exclusão. Observação: o aplicativo ainda está em desenvolvimento e poderá sofrer alterações, o que pode impactar no tamanho do arquivo e no tempo de espera.
RNF05	Manutenibilidade	O sistema deve ter código organizado e modular. Sugere-se que front-end e back-end se comuniquem via sockets, o que facilita a manutenção e escalabilidade.
RNF06	Confiabilidade	As operações devem garantir a confiabilidade dos cálculos, como saldo, porcentagem de gasto e totais mensais. Isso ajuda a manter os dados sincronizados e confiáveis, evitando diferenças visuais e numéricas.
RNF07	Interatividade	O sistema deve permitir, por meio de formulários e botões, uma interação fluida e intuitiva, proporcionando uma boa experiência ao usuário.

3. Modelos UML

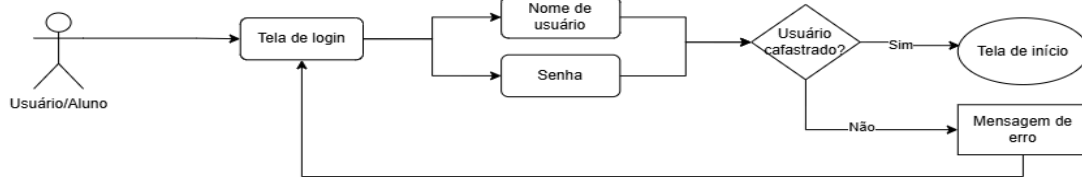
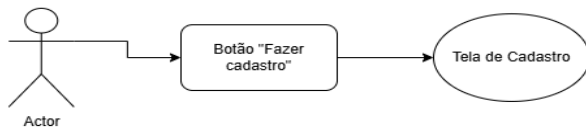
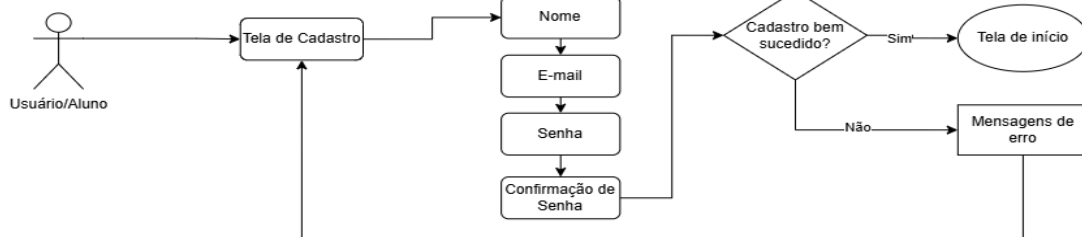
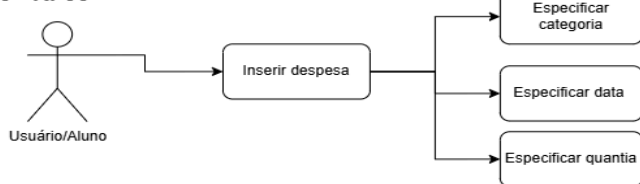
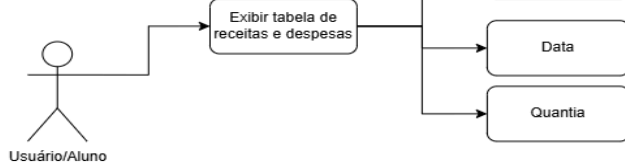
3.1 Diagrama de Casos de Uso



3.1.1. Casos das principais funcionalidades do usuário:

- Inserir receita/despesa
- Visualizar tabela
- Excluir registros

3.1.2. Diagrama de casos de Uso dos Requisitos funcionais:

CU01 - RF01**CU02 - RF03****CU04 - RF04****CU06 - RF06****CU07 - RF07****CU08 - RF08****CU09 - RF09**

4. Casos de Uso Detalhados

4.1. Caso de Uso: Inserir Despesa

- **Ator:** Usuário
- **Descrição:** Adiciona uma nova despesa ao sistema.
- **Pré-condição:** O sistema está em execução
- **Fluxo Principal:**
 - No quadro “Adicionar Nova Despesa”.
 - O usuário preenche os campos: categoria, valor, data.
 - Clica em “Adicionar”.
 - Sistema salva o registro e atualiza a tabela e os gráficos.
- **Fluxo Alternativo:**
 - Se algum campo estiver vazio ou inválido, o sistema exibe uma mensagem de erro.

4.2. Caso de Uso: Visualizar Gráficos

- **Ator:** Usuário
- **Descrição:** Visualiza os gráficos de pizza e de barras.
- **Pré-condição:** O sistema já ter dados cadastrados.
- **Fluxo Principal:**
 1. Usuário consegue visualizar os gráficos de acordo com os dados que estão sendo adicionados no sistema
 2. Sistema gera e exibe os gráficos com base nos dados salvos.

5. Arquitetura e Stack

Este sistema foi desenvolvido como uma aplicação de organização financeira pessoal voltada para alunos. A arquitetura utilizada é monolítica, implementada como uma aplicação **desktop** sem uso de APIs, e com armazenamento local em banco de dados **SQLite**.

O sistema é composto por uma interface gráfica desenvolvida em Python, com estrutura modularizada em arquivos separados para facilitar a organização de responsabilidades.

5.1 Detalhes da stack e integrações - Back-End:

- A implementação é feita em **Python 3**.
- A aplicação utiliza uma arquitetura **monolítica**, sem separação em microserviços ou camadas remotas.
- Persistência de dados é realizada em um banco de dados **SQLite**, utilizando a biblioteca nativa **sqlite3**.
- O banco é armazenado localmente no arquivo **dados.db**, sem necessidade de instalação ou execução de um servidor de banco de dados externo.
- A interface gráfica é construída com:

- **Tkinter**: base para janelas e interações visuais.
- **CustomTkinter**: fornece estilos visuais modernos e customizáveis.
- **tkcalendar**: campos de seleção de datas.
- **ttk.Treeview**: exibição de tabelas com registros financeiros.
- **Pillow (PIL)**: exibição de imagens e ícones.
- **Matplotlib**: geração de gráficos (pizza e barras) no dashboard.

5.2 Detalhes da stack e integrações - Front-End:

As principais bibliotecas e ferramentas utilizadas na construção da interface gráfica incluem:

- **CustomTkinter**: utilizada para criar uma interface gráfica customizada e visualmente moderna.
- **Tkinter**: base da interface gráfica, responsável pela criação das janelas e componentes principais.
- **Pillow (PIL)**: usada para exibir imagens, ícones e logos no sistema.
- **Matplotlib**: responsável pela geração de gráficos financeiros do tipo pizza e barras.
- **tkcalendar**: utilizada para os campos de seleção de data de forma visual e interativa.
- **ttk.Treeview**: componente usado para exibir registros financeiros em forma de tabela, com colunas e rolagem.

5.3 Comunicação entre Camadas

A arquitetura da aplicação é monolítica, ou seja, tanto a interface gráfica (front-end) quanto a lógica de negócios e o acesso ao banco de dados (back-end) operam no mesmo processo e ambiente local. Não há separação por serviços, nem uso de APIs ou comunicação via rede.

A interface gráfica, construída com CustomTkinter, interage diretamente com as funções do módulo `view.py`, responsável pela lógica da aplicação. Esse acoplamento direto facilita o fluxo de dados e comandos entre os componentes da aplicação.

Não são utilizadas requisições HTTP, REST APIs, nem sockets, pois o sistema não é uma aplicação web. Toda a comunicação ocorre por meio de chamadas de função internas no próprio código Python.

6. Organização do Código

6.1 Arquivos e Funções

A estrutura da aplicação é organizada por arquivos Python que dividem responsabilidades de forma lógica:

- **Linguagem**: Python
- **app.py**: responsável pela tela de login e cadastro, incluindo a lógica de autenticação de usuários.

- **main.py**: tela principal do sistema (dashboard), responsável por exibir saldos, gráficos e formulários para entrada de receitas e despesas.
- **view.py**: centraliza a lógica da aplicação e manipulação de dados (camada de regras de negócio).
- **bd.py**: realiza a criação do banco de dados e a conexão com o SQLite, além da criação das tabelas necessárias.

7. Estrutura do Banco de Dados

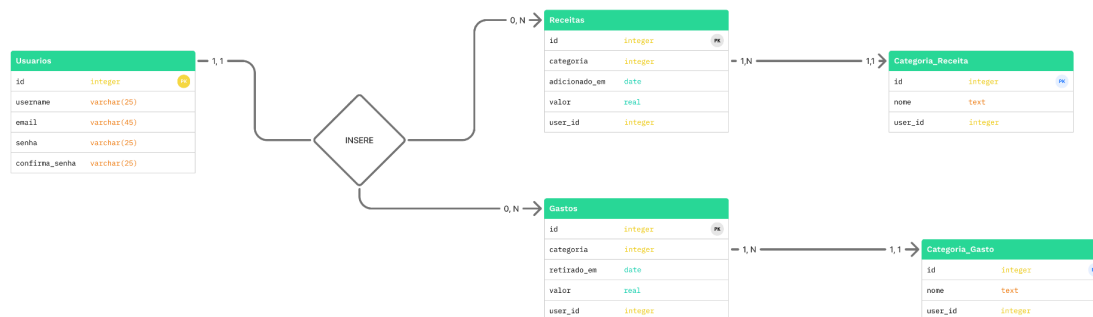
7.1. Visão Geral

O sistema utiliza cinco tabelas principais: `Usuario`, `receita`, `despesa`, `Categoria_Receita` e `Categoria_Gasto`. A tabela `Usuario` armazena os dados de login e identificação dos usuários. As tabelas `receita` e `despesa` registram, respectivamente, as entradas e saídas financeiras de cada usuário, incluindo data, valor e descrição. Cada transação está ligada a uma categoria específica, armazenada nas tabelas `Categoria_Receita` e `Categoria_Gasto`.

O diagrama apresenta a relação de cinco classes principais: `Usuário`, `Receita`, `Gasto`, `Categoria_Receita`, `Categoria_Gasto`.

- **Usuário**: representa quem vai utilizar o sistema. Cada usuário possui um `id`, `nome`, `email` e `senha`.
 - Está relacionado com `Receita` e `Despesa`, indicando que cada lançamento pertence a um único usuário.
- **Categoria_Receita**: classifica os lançamentos referente a entrada de valores (como “Salário”, “Venda”, “Comissão” e etc). Cada categoria possui um `id`, `nome` e `id` do usuário.
- **Categoria_Gasto**: classifica os lançamentos referente a saída de valores (como “Alimentação” e “Transporte”, etc). Cada categoria possui um `id`, `nome` e `id` do usuário.
- **Receitas**: representa os valores recebidos pelo usuário. Possui os campos `id`, `categoria` como chave estrangeira da chave primária `id` da tabela `Categoria_Receita`, `data` em que foi adicionado o valor (`adicionado_em`), `valor` e `id_usuario`.
 - Cada receita é associada a um usuário e uma categoria.
- **Gastos**: Representa os valores gastos pelo usuário. Contém os campos `id`, `categoria` como chave estrangeira da chave primária `id` da tabela `Categoria_Gasto`, `data` em que foi retirado o valor (`retirado_em`), `valor` e `user_id`.
 - Cada gasto também é associado a um usuário e uma categoria.

7.2. Modelo Entidade-Relacionamento (ER)



7.3. Tabela: Usuarios

7.4. Tabela: Categoria_Receita

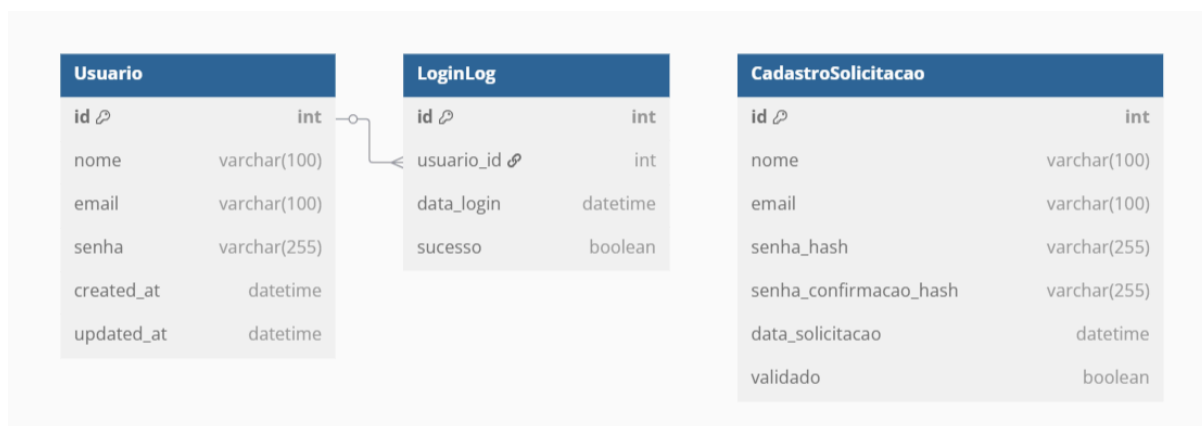
7.5. Tabela: Categoria_Gasto

7.6. Tabela: Receitas

7.7. Tabela: Gastos

<table> <tr><th colspan="3">Usuarios</th></tr> <tr><td>id</td><td>int</td><td>PK</td></tr> <tr><td>username</td><td>varchar(25)</td><td></td></tr> <tr><td>email</td><td>varchar(45)</td><td></td></tr> <tr><td>senha</td><td>varchar(25)</td><td></td></tr> <tr><td>confirma_senha</td><td>varchar(25)</td><td></td></tr> </table> <p>7.3</p>	Usuarios			id	int	PK	username	varchar(25)		email	varchar(45)		senha	varchar(25)		confirma_senha	varchar(25)		<table> <tr><th colspan="3">Categoria_Receita</th></tr> <tr><td>id</td><td>integer</td><td>PK</td></tr> <tr><td>nome</td><td>text</td><td></td></tr> <tr><td>user_id</td><td>integer</td><td></td></tr> </table> <p>7.4</p>	Categoria_Receita			id	integer	PK	nome	text		user_id	integer		<table> <tr><th colspan="3">Categoria_Gasto</th></tr> <tr><td>id</td><td>integer</td><td>PK</td></tr> <tr><td>nome</td><td>text</td><td></td></tr> <tr><td>user_id</td><td></td><td></td></tr> </table> <p>7.5</p>	Categoria_Gasto			id	integer	PK	nome	text		user_id		
Usuarios																																												
id	int	PK																																										
username	varchar(25)																																											
email	varchar(45)																																											
senha	varchar(25)																																											
confirma_senha	varchar(25)																																											
Categoria_Receita																																												
id	integer	PK																																										
nome	text																																											
user_id	integer																																											
Categoria_Gasto																																												
id	integer	PK																																										
nome	text																																											
user_id																																												
<table> <tr><th colspan="3">Receitas</th></tr> <tr><td>id</td><td>integer</td><td>PK</td></tr> <tr><td>categoria</td><td>integer</td><td></td></tr> <tr><td>adicionado_em</td><td>date</td><td></td></tr> <tr><td>valor</td><td>real</td><td></td></tr> <tr><td>user_id</td><td>integer</td><td></td></tr> </table> <p>7.6</p>	Receitas			id	integer	PK	categoria	integer		adicionado_em	date		valor	real		user_id	integer		<table> <tr><th colspan="3">Gastos</th></tr> <tr><td>id</td><td>integer</td><td>PK</td></tr> <tr><td>categoria</td><td>integer</td><td></td></tr> <tr><td>retirado_em</td><td>date</td><td></td></tr> <tr><td>valor</td><td>real</td><td></td></tr> <tr><td>user_id</td><td>integer</td><td></td></tr> </table> <p>7.7</p>	Gastos			id	integer	PK	categoria	integer		retirado_em	date		valor	real		user_id	integer								
Receitas																																												
id	integer	PK																																										
categoria	integer																																											
adicionado_em	date																																											
valor	real																																											
user_id	integer																																											
Gastos																																												
id	integer	PK																																										
categoria	integer																																											
retirado_em	date																																											
valor	real																																											
user_id	integer																																											

7.8. Modelagem do cadastro e login:



Nesse diagrama está representada a estrutura de dados necessária para implementar funcionalidades de login, cadastro de usuários e registro de acessos, conforme os requisitos RF01 a RF05.

Tabela: Usuario

Armazena os dados dos usuários cadastrados no sistema.

id: Identificador único (chave primária)

nome: Nome completo do usuário.

email: Endereço de e-mail único (usado para login).

senha: Senha criptografada.

created_at / updated_at: Datas para controle de criação e atualização.

Relacionamento:

Associada com LoginLog via usuario_id.

Tabela: LoginLog

Registra tentativas de login feitas no sistema.

id: Identificador da tentativa de login.

usuario_id: FK para Usuario.id, indicando quem tentou o acesso.

data_login: Data e hora da tentativa.

sucesso: Booleano indicando se o login foi bem-sucedido.

Objetivo:

Auxilia na auditoria e segurança, permitindo rastrear logins e falhas (RF01).

Tabela: CadastroSolicitacao

Opcional, usada para armazenar dados temporários de cadastro antes da criação efetiva de um usuário.

nome, email, senha_hash, senha_confirmacao_hash: Dados enviados pelo formulário de cadastro.

data_solicitacao: Registro de quando o pedido foi feito.

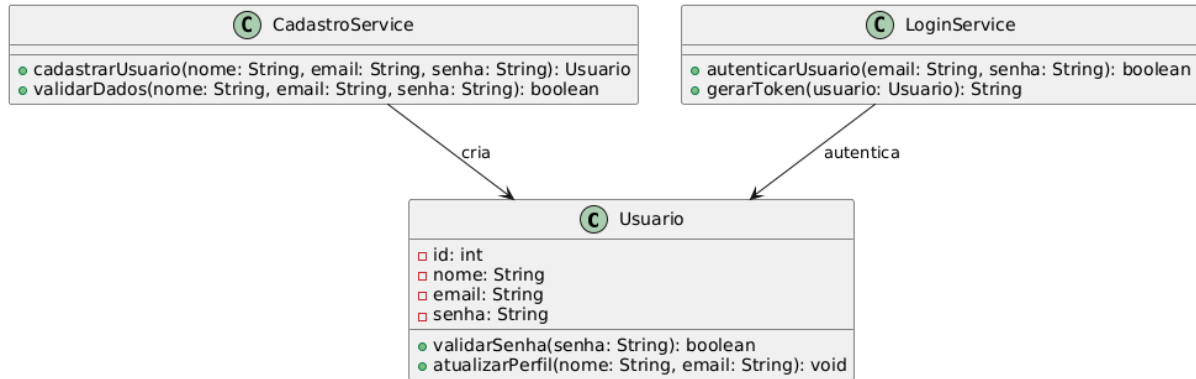
validado: Booleano indicando se a validação foi bem-sucedida (ex: senhas coincidem, e-mail válido etc.).

Relevância:

Suporta o processo de validação de campos e lógica de cadastro do RF04.

Relacionamentos

LoginLog.usuario_id → Usuario.id: Relacionamento de muitos para um, onde múltiplas tentativas de login podem ser atribuídas a um único usuário.



Descrição das principais classes:

Usuario: Entidade que armazena dados do usuário e ações relacionadas, como validar senha e atualizar perfil.

CadastroService: Serviço responsável pelo cadastro e validação dos dados do usuário.

LoginService: Serviço que gerencia a autenticação do usuário e geração de tokens de acesso.

8. Regras de Negócio Implementadas

O sistema aplica regras de negócio para garantir consistência nos cálculos financeiros, integridade dos dados inseridos e uma visualização clara das finanças pessoais do usuário. Abaixo estão descritas as principais regras:

8.1. Cálculo de Saldo Total

O saldo total do usuário é calculado com base na diferença entre o total de receitas e o total de despesas registradas. Este valor é exibido no painel principal como resumo da situação financeira.

Fórmula: $\text{saldo} = \text{total_receitas} - \text{total_despesas}$
 (Função associada: `bar_valores()`)

8.2. Geração do Gráfico de Pizza

Para facilitar a análise visual dos gastos, o sistema agrupa as despesas por categoria e soma os valores correspondentes. O resultado é representado por um gráfico de pizza, exibindo a distribuição proporcional dos gastos.

(Função associada: `pie_valores()`)

8.3. Cálculo da Porcentagem Gasta

O sistema calcula o percentual da receita total que já foi utilizada com despesas. Esse valor é útil para o usuário entender quanto do seu orçamento já foi comprometido.

Fórmula: $\text{porcentagem} = (\text{total_despesas} / \text{total_receitas}) \times 100$
(Função associada: `porcentagem_valor()`)

8.4. Validações de Cadastro

Durante o cadastro de novos usuários, o sistema realiza as seguintes validações obrigatórias:

- Todos os campos devem ser preenchidos.
- A senha e a confirmação de senha devem ser idênticas.
- O nome de usuário e a senha devem conter no mínimo 4 caracteres cada.

9. Tratamento de Erros e Feedback ao Usuário

A aplicação implementa mecanismos de tratamento de erros e exibição de mensagens para garantir uma experiência mais segura e compreensível ao usuário. Esses tratamentos evitam falhas críticas no sistema e informam o usuário de forma clara sobre qualquer problema que ocorra. Abaixo estão os principais tipos de tratamento utilizados:

- **Erro de Tipo (ValueError)**
 Ao inserir valores em campos numéricos, como "quantia", o sistema valida o tipo informado. Caso o valor não seja numérico, a exceção `ValueError` é capturada via `try-except` e uma mensagem de erro é exibida ao usuário.
- **Ação sem Seleção (IndexError)**
 Se o usuário tentar realizar uma ação (como exclusão ou edição) sem selecionar previamente um item na tabela, a exceção `IndexError` é tratada para evitar travamentos. Uma mensagem de aviso é exibida alertando sobre a necessidade da seleção.
- **Campos Vazios**
 O sistema verifica se os campos obrigatórios foram preenchidos antes de prosseguir com qualquer operação (como cadastro, receita ou despesa). Caso contrário, o usuário é informado por meio de mensagens específicas.
- **Mensagens ao Usuário**
 A biblioteca `messagebox` (do Tkinter) é utilizada para fornecer retorno visual ao usuário por meio de três tipos principais de mensagem:
 - `showerror`: para erros críticos ou inserções inválidas.
 - `showinfo`: para confirmações e informações gerais.
 - `showwarning`: para alertas e avisos de preenchimento incompleto.

- **Erros Inesperados (Genéricos)**

Blocos genéricos de `try-except` são utilizados em pontos críticos da aplicação para capturar exceções não previstas. Isso garante que a aplicação continue funcionando mesmo diante de falhas ocasionais, exibindo mensagens de erro ao invés de encerrar o programa abruptamente.

10. Protótipo de alta fidelidade no Figma

- **Link para o Projeto:** [Protótipo de Alta Fidelidade - Figma](#)

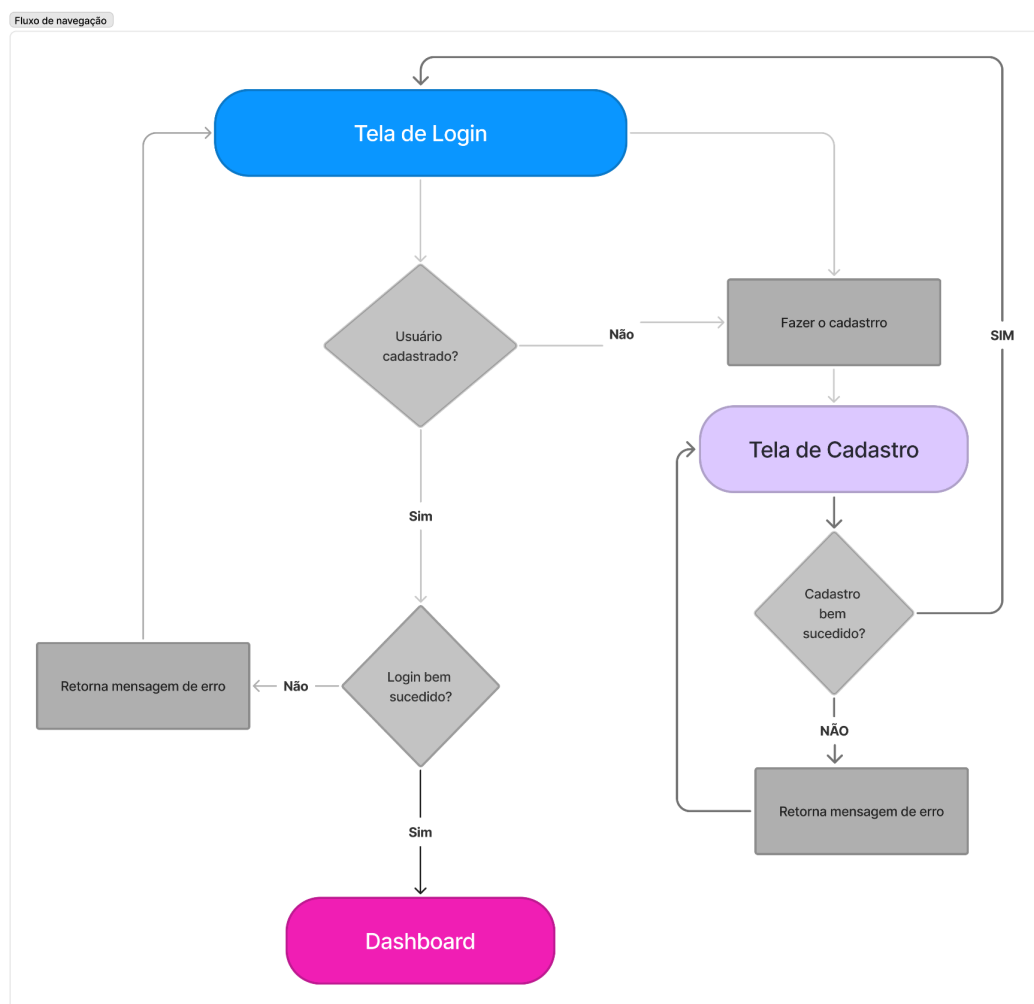
10.1. Telas Desenvolvidas

- **Login:** com exibição de erro
- **Cadastro:** campos obrigatórios com botão de visualizar senha
- **Dashboard:**
 - Cards de saldo, receita e despesas
 - Gráficos (pizza e barras)
 - Tabela de registros
 - Formulários para inserir e deletar registros

10.2. Fluxo de Navegação

1. Login → Dashboard
2. Login → Cadastro
3. Cadastro → Login
4. Dashboard → Ações locais

11.2.1. Representação do fluxo entre telas



11.3. Responsividade

O protótipo foi desenvolvido com foco na visualização em ambientes desktop, priorizando uma interface horizontal que acomoda múltiplos componentes simultaneamente na tela. A disposição dos elementos busca otimizar o uso do espaço e facilitar a leitura e a interação em telas largas.

11.4. Elementos Reutilizáveis

Para garantir consistência visual e eficiência no desenvolvimento, diversos componentes foram projetados como reutilizáveis:

- **Botões:** Seguem um padrão de cor, forma e tipografia, mantendo coerência visual em toda a interface.

- **Campos de entrada (inputs):** Utilizam uma estrutura uniforme com foco na clareza e na usabilidade.
- **Cards e gráficos:** Apresentam layout padronizado no dashboard, permitindo fácil leitura e comparabilidade dos dados.
- **Ícones funcionais:** Elementos como o ícone de "mostrar/ocultar senha" e os indicadores de erro são replicados de forma consistente entre telas.

11.5. Alinhamento com o Front-End

A implementação front-end segue fielmente a estrutura proposta no Figma, garantindo que os elementos visuais estejam de acordo com o protótipo:

- Componentes como inputs, botões, espaçamentos e cores foram adotados conforme especificado.
- A padronização visual contribui para uma transição fluida entre prototipação e desenvolvimento.

11.6. Padrões Visuais

O protótipo adota uma identidade visual consistente, que se mantém coesa em todas as telas:

- **Tipografia:** Fonte sem serifa aplicada uniformemente em títulos, campos e botões. Textos explicativos seguem alinhamento à esquerda ou centralizado, com uso de tamanhos menores para conteúdo secundário.
- **Cores:**
 - Verde escuro: Ações afirmativas e confirmatórias (ex.: cadastro).
 - Azul escuro: Ações principais (ex.: login).
 - Cinza escuro: Base neutra, garantindo aparência limpa e moderna.
 - O dashboard utiliza tons de azul, cinza e verde, reforçando a continuidade visual em relação às telas iniciais.

11.7. Anexos - imagens das telas

Imagem 1 - Tela de Login



Faça seu login ou cadastre-se aqui para acessar os nossos serviços


Poupaê

Faça seu Login

Seu nome de usuário..

Sua senha..

☒ Clique para ver a senha

FAZER LOGIN

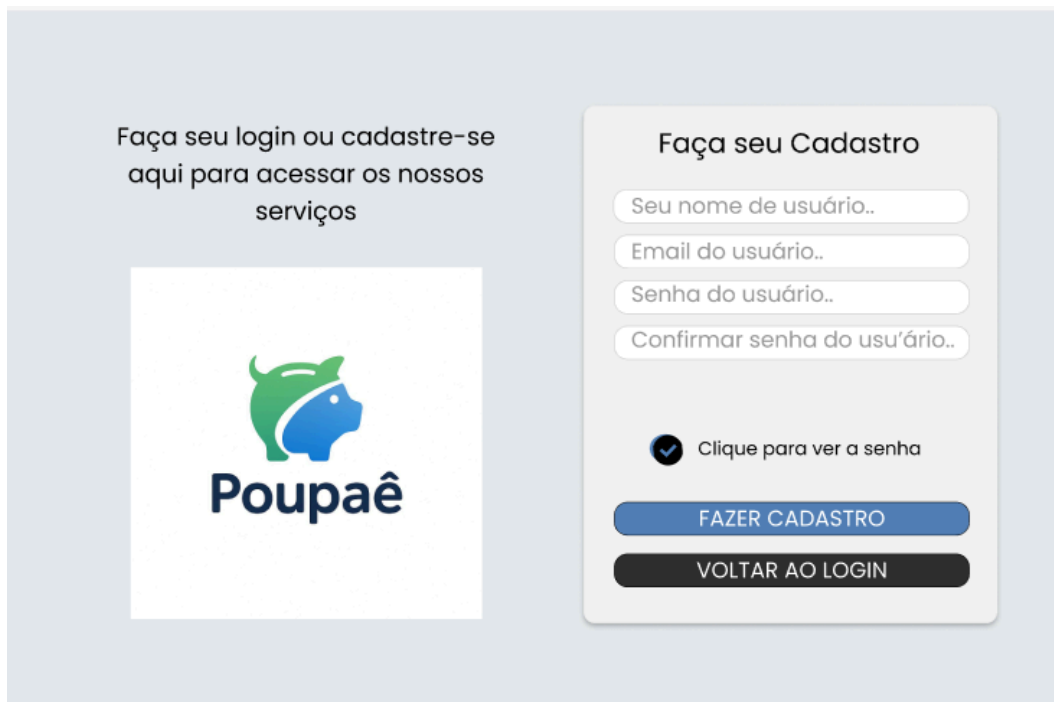
Se não tem conta, clique no botão abaixo para poder se cadastrar no nosso sistema!

FAZER CADASTRO

Tela de Login – Identidade Visual e Ações Primárias

- A tela utiliza tipografia sem serifa uniforme, com tamanhos distintos para títulos, campos e mensagens secundárias.
- O botão FAZER LOGIN aplica a cor azul escuro #1A237E, representando a ação principal.
- Já o botão FAZER CADASTRO usa verde escuro #1B5E20, indicando ação afirmativa.
- Os campos de entrada contam com placeholders em cinza claro #BDBDBD, com design minimalista e cantos arredondados.
- O layout mantém alinhamento centralizado e espaçamentos coerentes, refletindo uma aparência limpa e moderna.

Imagem 2 - Tela de Cadastro

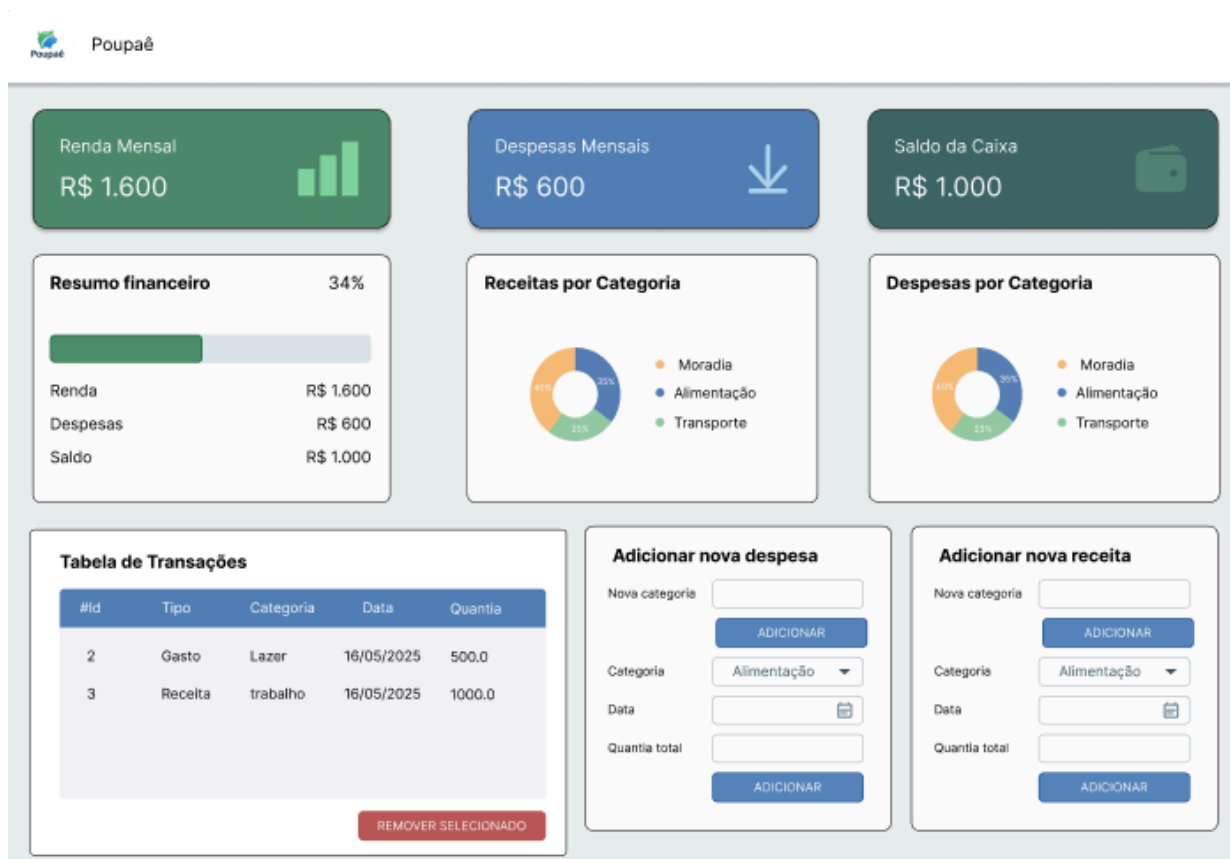


A tela de cadastro do Poupaê apresenta um layout limpo e moderno. No topo, o texto "Faça seu login ou cadastre-se aqui para acessar os nossos serviços" orienta o usuário. À esquerda, o logo do Poupaê, composto por uma piggy bank verde e azul com o nome "Poupaê" em azul escuro, é exibido. À direita, o formulário de cadastro, intitulado "Faça seu Cadastro", contém quatro campos de entrada com placeholders: "Seu nome de usuário..", "Email do usuário..", "Senha do usuário..", e "Confirmar senha do usuário..". Abaixo dos campos, há uma opção "Clique para ver a senha" com um ícone de olho. No final do formulário, dois botões são apresentados: "FAZER CADASTRO" em azul escuro e "VOLTAR AO LOGIN" em cinza escuro.

Tela de Cadastro – Continuidade Visual e Hierarquia de Ações

- Esta tela segue o mesmo padrão visual da tela de login, utilizando tipografia sem serifa e organização vertical clara.
- O botão FAZER CADASTRO utiliza o azul escuro #5381B6 para reforçar a importância da ação de cadastro, enquanto o botão VOLTAR AO LOGIN adota cinza escuro #2D2D2D como ação secundária.
- Os campos de entrada possuem placeholders em tom cinza claro #BDBDBD com bordas arredondadas, promovendo uma aparência limpa e moderna.
- O layout mantém alinhamentos centralizados e espaçamento consistente entre os elementos, reforçando a coesão entre as telas.

Imagem 3 - Tela Dashboard



Dashboard – Continuidade Visual e Hierarquia Informacional

- A tela de dashboard mantém a identidade visual com o uso consistente de tipografia sem serifa, cores funcionais e layout modular em grade.
- Os indicadores utilizam verde escuro #1B5E20 para representar saldo positivo e renda mensal, e azul escuro #5581B6 para despesas e receitas.
- Os gráficos de pizza seguem uma padronização cromática por categoria: moradia (laranja), alimentação (verde), transporte (azul claro), reforçando a compreensão visual.
- Os botões de ação crítica, como REMOVER SELECIONADO, utilizam vermelho #B95855 para alertar o usuário.
- Todos os componentes seguem espaçamento e proporções consistentes, com borda arredondada e sombreamento leve, reforçando a estética limpa e moderna.

12. Teste de Software

O objetivo do Teste de Software é validar a funcionalidade, a usabilidade e a estabilidade do sistema, garantindo que ele atenda aos requisitos e que a experiência do usuário seja fluida e

livre de erros.

Os testes cobrirão todos os principais módulos da aplicação, incluindo:

- Autenticação (Login e Cadastro de Usuários)
- Dashboard Financeiro (Visualização de dados)
- Gerenciamento de Transações (Receitas e Despesas)
- Gerenciamento de Categorias
- Persistência de Dados

12.1. Ambiente de Teste

- **Sistema Operacional:** Windows 10/11
- **Versão do Python:** 3.8 ou superior
- **Dependências:** customtkinter, pandas, matplotlib, tkcalendar, pillow

12.2 Casos de Teste

A seguir, são detalhados os casos de teste a serem executados.

12.2.1. Módulo 1: Autenticação (app.py)

ID do Teste	Descrição do Teste	Passos para Execução	Resultado Esperado	Status
CAD-001	Cadastro de usuário com sucesso	1. Abrir o app. 2. Clicar em "Fazer Cadastro". 3. Preencher todos os campos com dados válidos (senhas iguais). 4. Clicar em "Fazer Cadastro".	O usuário é cadastrado com sucesso e uma mensagem de confirmação é exibida. Os campos são limpos.	Aprovado
CAD-002	Cadastro com campos vazios	1. Abrir o app. 2. Clicar em "Fazer Cadastro". 3. Deixar um ou mais campos em branco. 4. Clicar em "Fazer Cadastro".	Uma mensagem de erro informando que todos os campos devem ser preenchidos é exibida.	Aprovado
CAD-003	Cadastro com senhas divergentes	1. Abrir o app. 2. Clicar em "Fazer Cadastro". 3. Preencher os campos de senha e confirmação com	Uma mensagem de erro informando que as senhas não coincidem é exibida.	Aprovado

		valores diferentes. 4. Clicar em "Fazer Cadastro".		
LOGIN-001	Login com sucesso	1. Abrir o app. 2. Inserir um nome de usuário e senha válidos e existentes. 3. Clicar em "Fazer Login".	A janela de login é fechada e o Dashboard principal é aberto.	Aprovado
LOGIN-002	Login com credenciais inválidas	1. Abrir o app. 2. Inserir um nome de usuário ou senha incorretos. 3. Clicar em "Fazer Login".	Uma mensagem de erro informando que as credenciais são inválidas é exibida.	Aprovado
LOGIN-003	Login com campos vazios	1. Abrir o app. 2. Deixar os campos de usuário e/ou senha em branco. 3. Clicar em "Fazer Login".	Uma mensagem de erro informando que todos os campos devem ser preenchidos é exibida.	Aprovado

12.2.2. Módulo 2: Dashboard e Transações (main.py e view.py)

ID do Teste	Descrição do Teste	Passos para Execução	Resultado Esperado	Status
CAT-001	Adicionar nova categoria de receita	1. Logar no sistema. 2. No painel de receitas, digitar um nome para a nova categoria. 3. Clicar em "SALVAR".	A categoria é salva e aparece na lista suspensa de categorias de receita. Uma mensagem de sucesso é exibida.	Aprovado
CAT-002	Adicionar nova categoria de despesa	1. Logar no sistema. 2. No painel de despesas, digitar um nome para a nova categoria. 3.	A categoria é salva e aparece na lista suspensa de categorias de despesa. Uma mensagem de sucesso é exibida	Aprovado

		Clicar em "SALVAR".		
TRANS-001	Inserir nova receita	1. Logar no sistema. 2. Selecionar uma categoria, data e inserir um valor válido. 3. Clicar em "ADICIONAR" no painel de receitas.	A receita é adicionada. Os cards de "Renda" e "Saldo", os gráficos e a tabela são atualizados.	Aprovado
TRANS-002	Inserir nova despesa	1. Logar no sistema. 2. Selecionar uma categoria, data e inserir um valor válido. 3. Clicar em "ADICIONAR" no painel de despesas.	A despesa é adicionada. Os cards de "Despesas" e "Saldo", os gráficos e a tabela são atualizados.	Aprovado
TRANS-003	Inserir transação com valor inválido	1. Logar no sistema. 2. Tentar adicionar uma receita ou despesa com um texto no campo de valor (ex: "abc").	Uma mensagem de erro informando que o valor deve ser numérico é exibida. A transação não é salva.	Aprovado
TRANS-004	Deletar uma transação	1. Logar no sistema. 2. Selecionar uma transação na tabela. 3. Clicar no botão "DELETAR SELECIONADO"	A transação é removida da tabela e do banco de dados. Todos os valores e gráficos do dashboard são atualizados.	Aprovado
TRANS-005	Tentar deletar sem selecionar	1. Logar no sistema. 2. Clicar no botão "DELETAR SELECIONADO"	Uma mensagem de erro informando que um item deve ser selecionado é exibida.	Aprovado

		" sem ter selecionado um item na tabela.		
--	--	------------------------------------------	--	--

12.2.3. Módulo 3: Persistência de Dados e Isolamento

ID do Teste	Descrição do Teste	Passos para Execução	Resultado Esperado	Status
DATA-001	Persistência de dados após fechar	1. Logar como Usuário A. 2. Adicionar/deletar categorias e transações. 3. Fechar o aplicativo. 4. Abrir o aplicativo novamente e logar como Usuário A.	Todas as alterações feitas pelo Usuário A (categorias, receitas, despesas) devem estar exatamente como foram deixadas.	Aprovado
DATA-002	Isolamento de dados entre usuários	1. Cadastrar e logar como Usuário A. Adicionar transações. 2. Fazer logout (fechar e reabrir o app). 3. Cadastrar e logar como Usuário B.	O dashboard do Usuário B deve estar completamente vazio, sem nenhuma informação do Usuário A.	Aprovado

12.3 Anexos



Figura 1: print do cadastro realizado com sucesso



Figura 2: print do cadastro mostrando falha, pois os campos não foram preenchidos

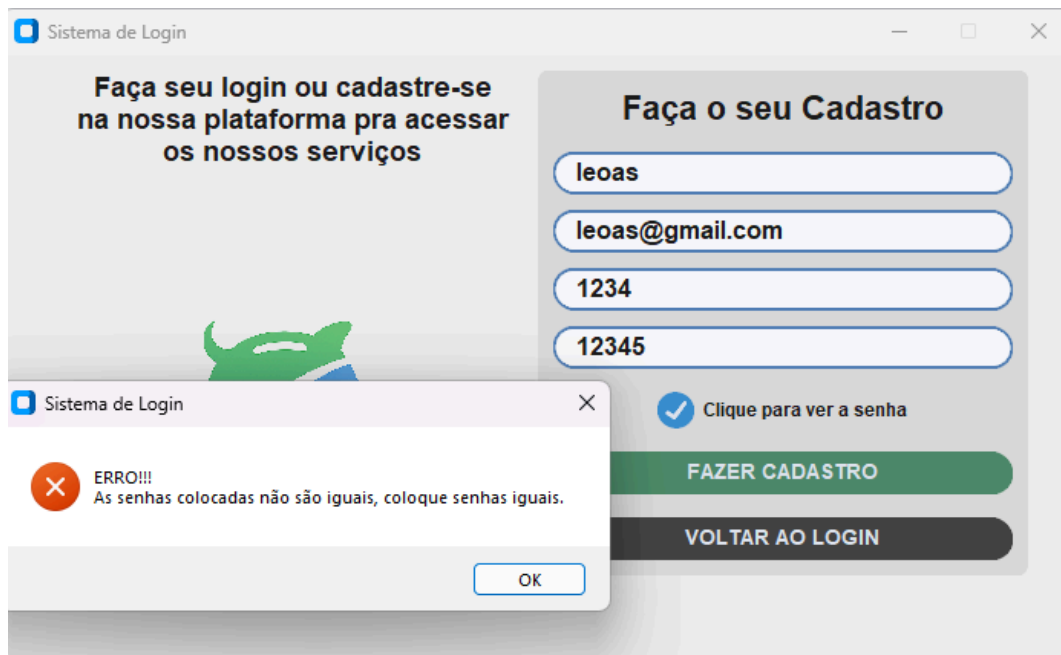


Figura 3: print do cadastro mostrando falha, pois as senhas são diferentes

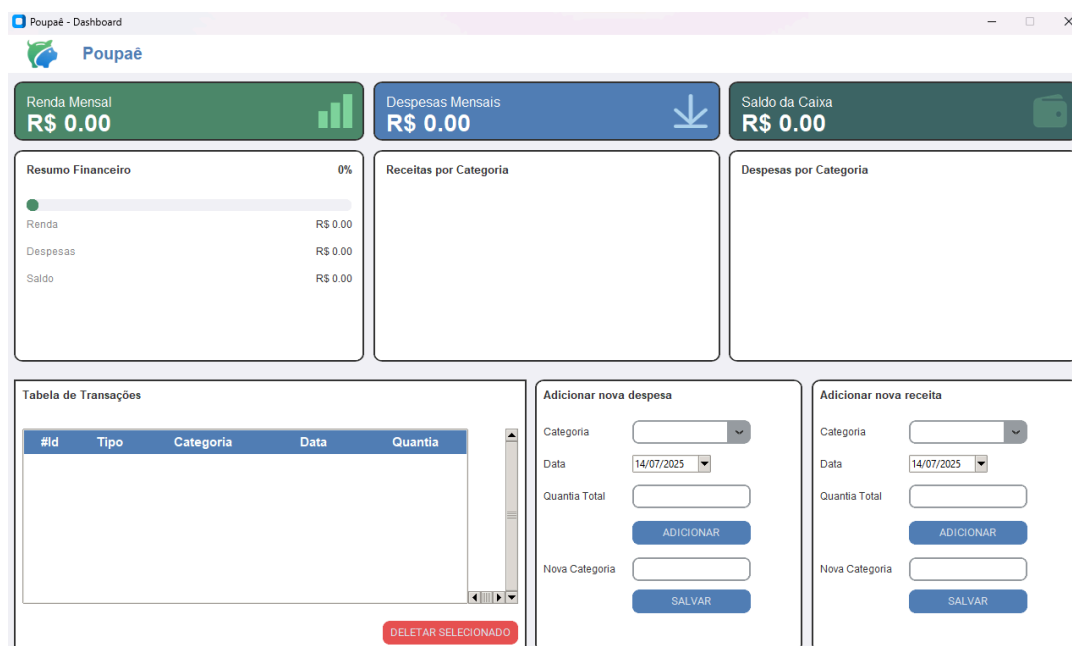


Figura 4: print do cadastro feito e login feito, fechando a tela de login e abrindo o dashboard

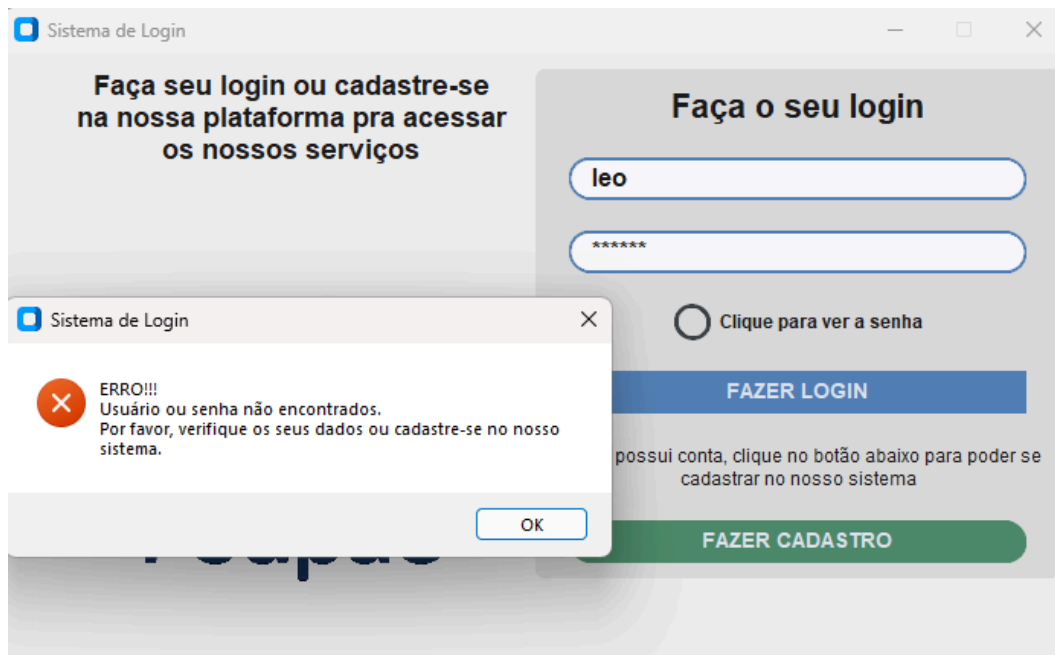


Figura 5: print do login sem sucesso, pois as credenciais estão inválidas



Figura 6: print do login sem sucesso, pois os campos não foram preenchidos

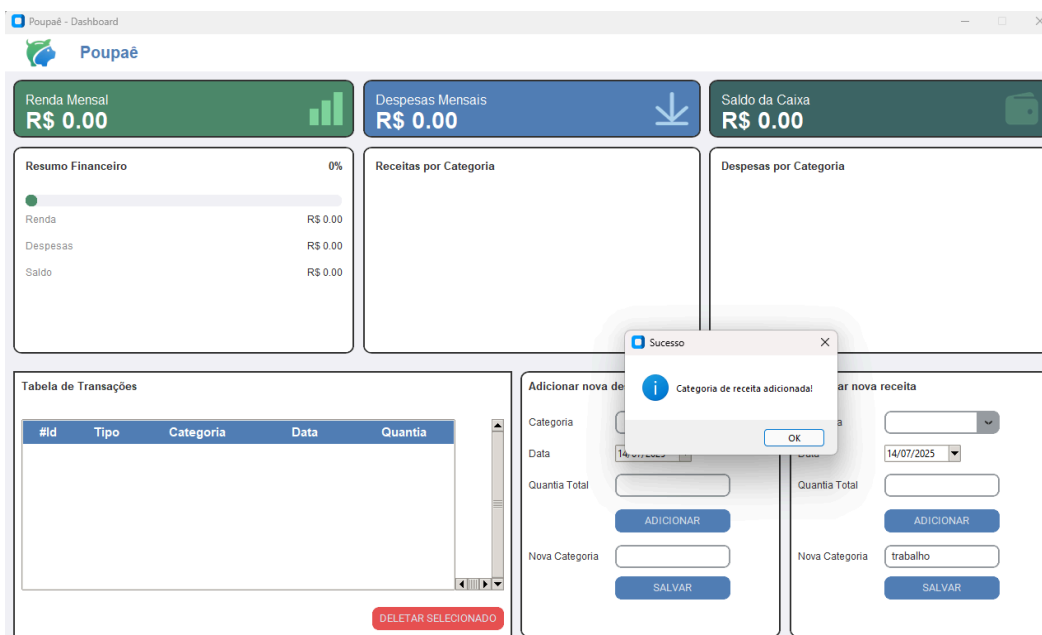


Figura 7: print da receita salva com sucesso

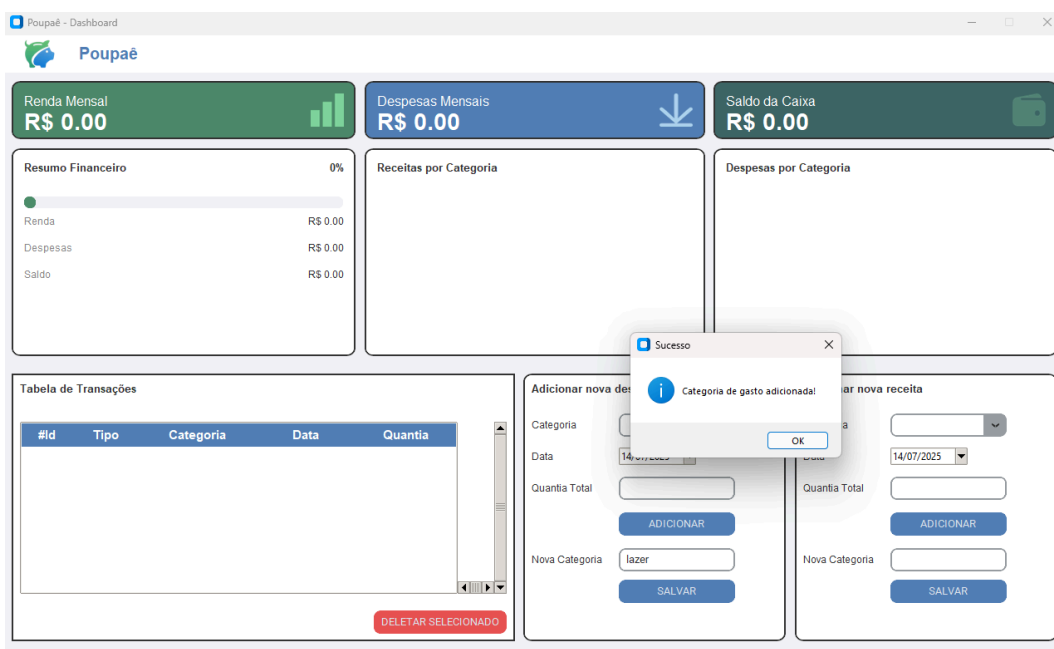


Figura 8: print da despesa salva com sucesso

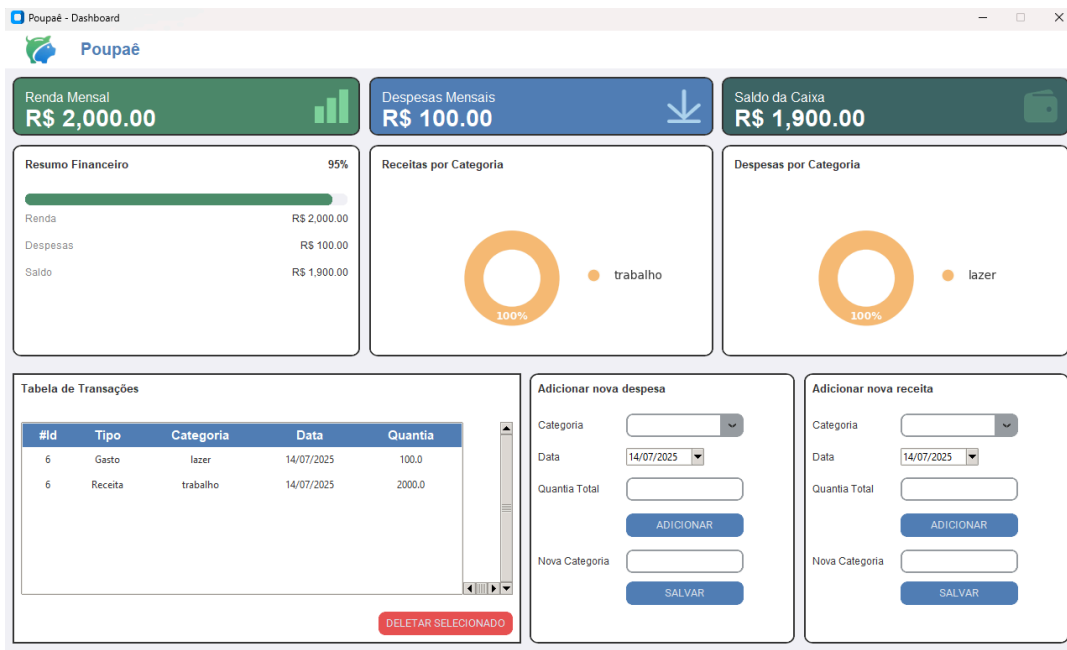


Figura 9: print da despesa e receita adicionadas com sucesso

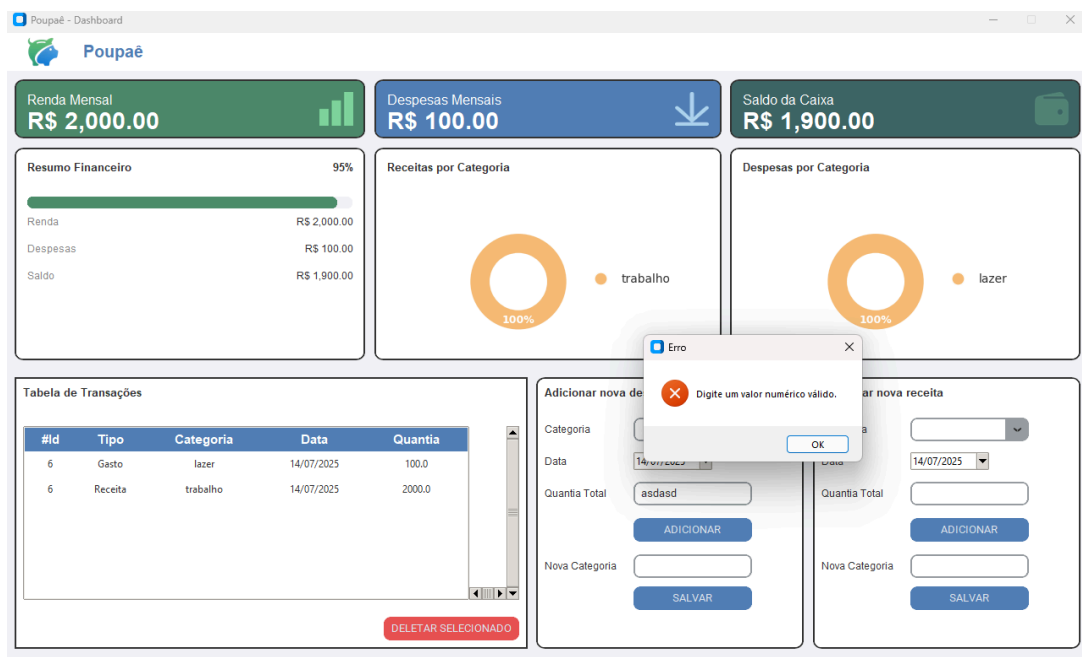


Figura 10: print da tentativa de inserir letras no campo de “quantia”

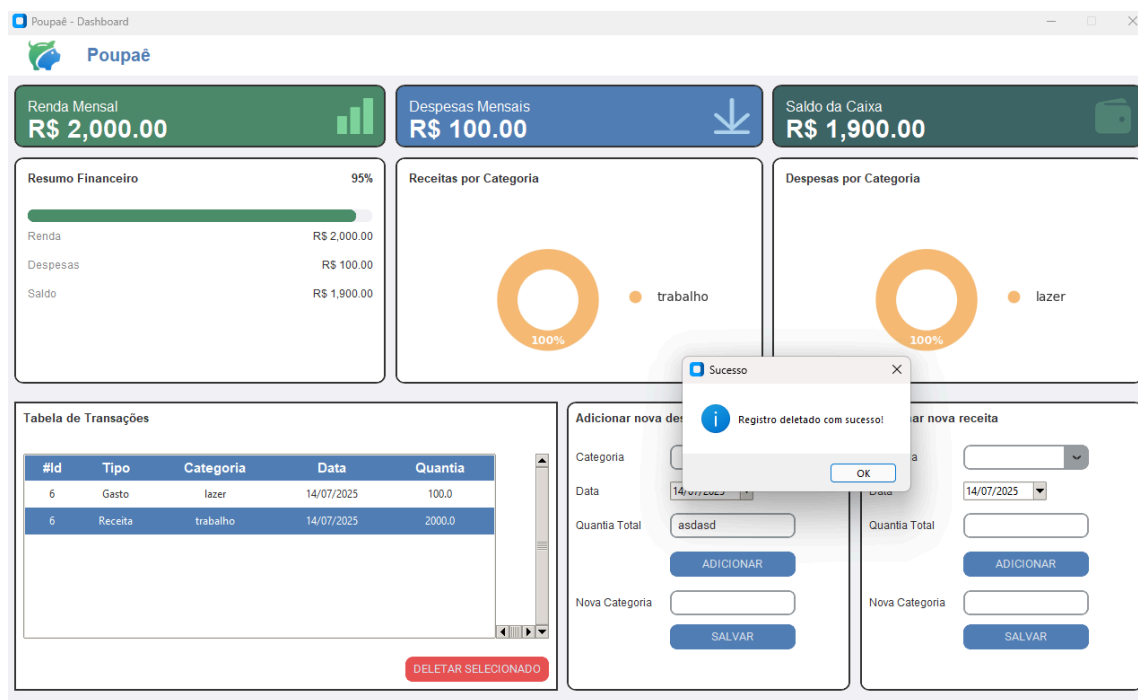


Figura 11: print da receita “trabalho” excluída com sucesso

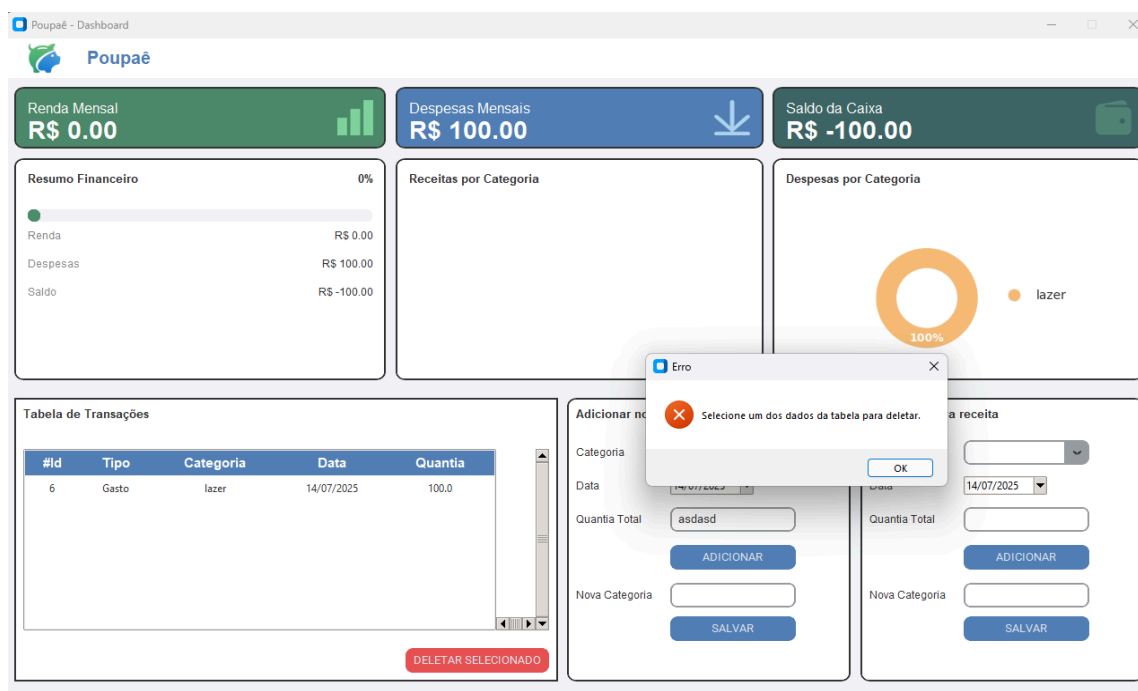


Figura 12: print da tentativa de excluir uma transação sem selecionar

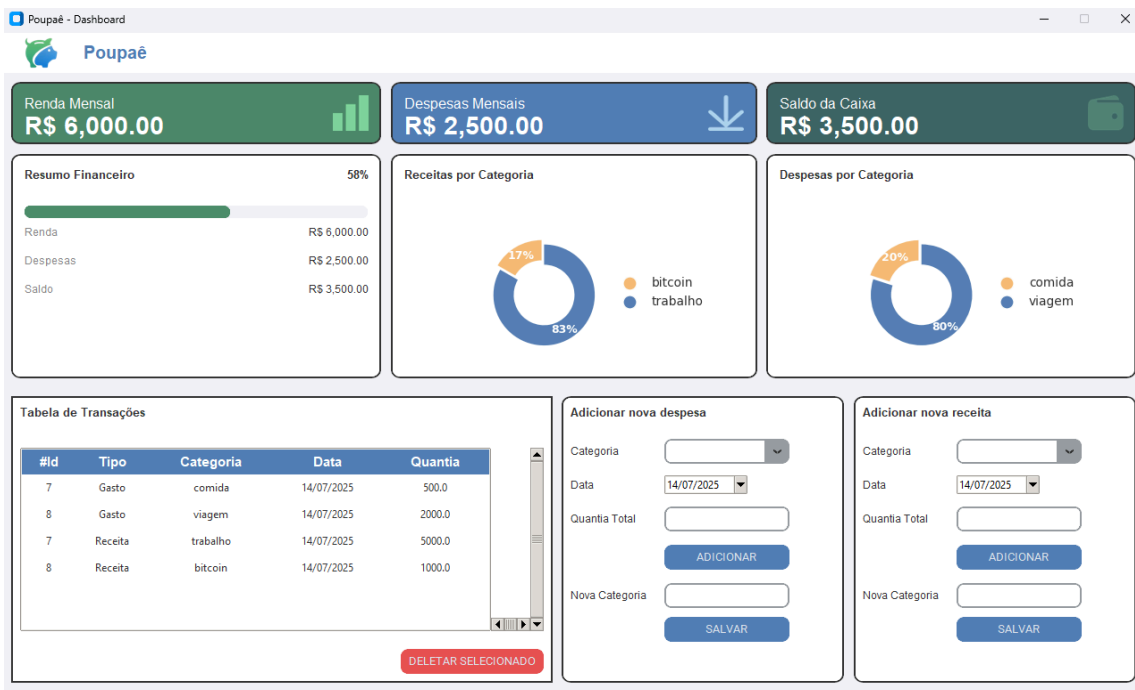


Figura 13: print da criação de um novo usuário, feito o login e adicionado 2 receitas e 2 despesas, programa funcionando por completo

13. Guia de geração do executável (build)

Esta seção detalha o processo passo a passo para gerar um arquivo executável (.exe) autocontido para o sistema de finanças Poupaê, utilizando Python e a ferramenta PyInstaller em um ambiente Windows.

- **Pré-requisitos**

- Python (versão 3.8 ou superior).
- Git (opcional, para clonar o repositório).
- O código-fonte completo do projeto.

- **Passo a Passo para a Geração**

Seguir essas etapas abaixo a partir de um terminal (como o PowerShell) aberto na pasta raiz do projeto.

1. Criação do Ambiente Virtual

Para isolar as dependências do projeto e evitar conflitos, foi fundamental criar um ambiente virtual (venv).

No PowerShell

```
# Criar a pasta do ambiente virtual chamada 'venv'
python -m venv venv
```

2. *Ativação do Ambiente Virtual*

Ativar o ambiente para que todos os comandos de instalação funcionem dentro dele.

PowerShell

```
# Ativa o venv no PowerShell
```

```
.\venv\Scripts\Activate.ps1
```

O terminal agora exibiu (venv) no início da linha.

3. *Instalação das Dependências*

Instalar todas as bibliotecas Python necessárias para o projeto, listadas no arquivo requirements.txt.

PowerShell

```
pip install -r requirements.txt
```

4. *Instalação do PyInstaller*

Instalar o PyInstaller, que irá empacotar o projeto em um executável.

PowerShell

```
pip install pyinstaller
```

5. *Preparação dos Ativos*

O ícone do executável precisa estar no formato .ico.

Converter o arquivo logo.png para logo.ico usando um conversor online.

Salvar o arquivo logo.ico na pasta raiz do projeto.

6. *Execução do Comando de Build*

Foi o comando principal que leu o código, encontrou todas as dependências e as empacotou em um único arquivo .exe. Copiar o comando da seção abaixo e executar no terminal (com o venv ainda ativo).

Comando de Build do PyInstaller

O comando foi executado como uma única linha.

PowerShell

```
pyinstaller --name PoupaeApp --onefile --windowed --icon="logo.ico"
--add-data="C:\Users\victo\OneDrive\Área de
Trabalho\python\venv\Lib\site-packages\customtkinter;customtkinter"
--add-data="logo.png;." --add-data="porco.png;." --add-data="rendaIcon.png;."
--add-data="despesasIcon.png;." --add-data="walletIcon.png;." --collect-all matplotlib
app.py
```

Atenção: Como encontrar o CAMINHO_DO_CUSTOMTKINTER:

O caminho para a biblioteca customtkinter é específico para cada computador. Para encontrá-lo, é necessário executar o seguinte comando no terminal (com o venv ativo):

PowerShell

```
python -c "import customtkinter; print(customtkinter.__file__)"
```

Copiar o resultado e remover o `__init__.py` do final para obter o caminho da pasta. Substituir o `CAMINHO_DO_CUSTOMTKINTER` no comando de build por este caminho.

Exemplo:

Se a saída for `C:\Projetos\Poupae\venv\Lib\site-packages\customtkinter__init__.py`

O caminho a ser usado será `C:\Projetos\Poupae\venv\Lib\site-packages\customtkinter`

7. Resultado Final

Após a execução bem-sucedida do comando, as duas novas pastas foram criadas: build e dist. O programa final e distribuível, PoupaeApp.exe, ficou localizado dentro da pasta dist. Agora ele ficou fácil de compartilhar com qualquer usuário. Ao ser executado, ele criará uma pasta data ao seu lado para armazenar o banco de dados.

14. Formulário sobre a avaliação do software

Foi realizado um pequeno formulário com 05 perguntas para diversos usuários, alunos da Universidade Federal do Ceará - Campus Crateús, obtivemos 12 respostas ao total. O formulário foi feito com sigilo de identidade.

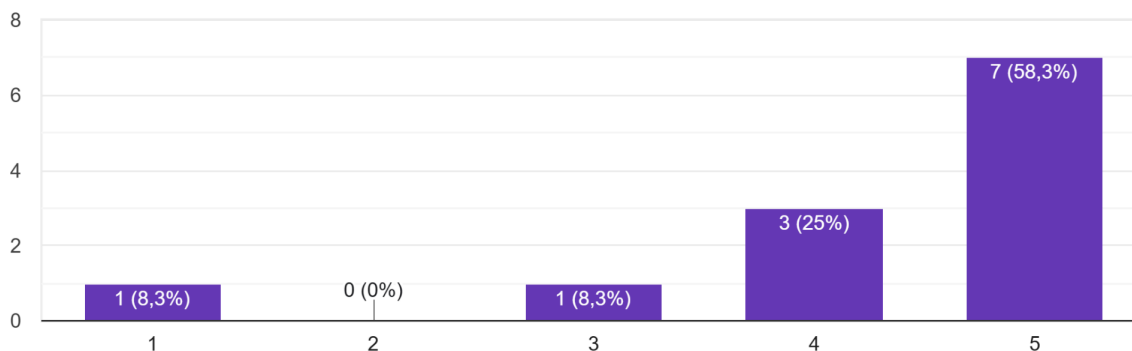
Segue abaixo as avaliações dos participantes e o link a seguir:

<https://docs.google.com/forms/d/e/1FAIpQLSeJ4gH0dMYgRCNxURXu2qxzpSaPbxA0wZwAEFozxK1k7eNdrw/viewform>:

1.

Em uma escala de 1 a 5, quão fácil e intuitivo você achou o aplicativo? (1 = Muito Difícil, 5 = Muito Fácil)

12 respostas



2.

O que você achou do visual do aplicativo (cores, ícones, layout)?

10 respostas

Os ícones, cores e o próprio layout foram bem escolhidos e projetados, são telas que não cansa muito a visão de quem está olhando, e sempre mantém um padrão de cores e de ícones em suas telas, o que facilita a navegação de quem usa.

Muito bonito e prático

Intuitivo

Muito bonito

Muito bom e intuitivo

Boa visualização e compreensão.

Visual extremamente agradável e didático para o usuário

Achei bem intuitivo, as cores são agradáveis, e os ícones ajudam a entender cada função

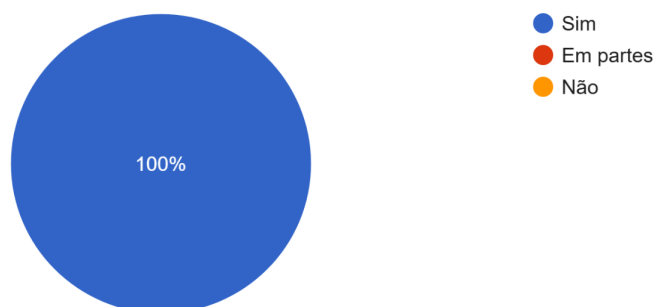
Harmônica

Achei as cores bem adequadas, remetem bem ao tema, muito bonito mesmo!

3.

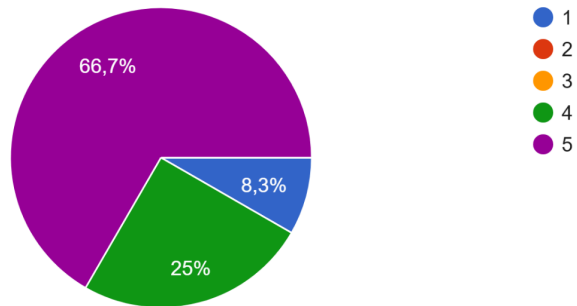
As funções de adicionar, deletar e visualizar suas finanças (gráficos, tabelas) foram úteis e funcionaram como esperado?

12 respostas

**4.**

Qual o seu nível de satisfação geral com o aplicativo "Poupaê"? (1 = Muito insatisfeito, 5 = Muito satisfeito)

12 respostas



5.

Quais sugestões você daria para melhorar o app ?

8 respostas

Mais funções

Uma versão mobile

Login direto após fazer o cadastro

Colocaria a cor vermelha nas despesas.

Melhoria no campo onde preencho o dados de despesas

Aumentar o tamanho da tela.

Nada, está perfeito, gostaria muito de poder usar no meu dia a dia

Renda passiva e renda ativa

15. Hospedagem do Código

A hospedagem foi feita no github, segue o link: [GitHub - Poupaê](#)



Documentação Técnica - Projeto Integrador I

Projeto e modelagem de um sistema de controle financeiro pessoal: Poupaê
Crateús-Ce | 2025