Projeto Integrador I

TIME BACKEND

Informações técnicas do software

1. **Link para o projeto no GitHub**

https://github.com/wictorsousa/Poupae.git

2. **Quais tecnologias e linguagens vocês estão usando no back-end?**

A linguagem usada é Python. Utilizamos também a biblioteca nativa sqlite3, que implementa um banco de dados SQLite, é simples, fácil de usar e não exige um servidor separado para funcionar.

3. **O sistema está usando banco de dados? Qual tipo?**

Sim, SQLite. É um banco de dados relacional embarcado, ou seja, ele opera diretamente a partir de um único arquivo (dados.db no nosso projeto), sem a necessidade de um servidor de banco de dados separado.

4. **Quais são as principais tabelas ou coleções do banco de dados?**

   O banco de dados possui quatro tabelas principais:

   - Usuarios: Guarda os registros de cada usuário, com Id, Username, Email, Senha, Confirma_Senha.
   - Categoria_Receita: Armazena as categorias para as receitas, guardando id e nome (ex: “Salário”, “Vendas”, “Rendimentos”).
   - Categoria_Gasto: Armazena as categorias para os gastos, guardando id e nome (ex: “Contas”, “Lazer”, “Transporte”).
   - Receitas: Guarda os registros de cada entrada de dinheiro, com id, categoria, adicionado_em e valor.
   - Gastos: Guarda os registros de cada saída de dinheiro, com id, categoria, retirado_em e valor.

5. **Tem alguma regra importante de negócio que vocês implementaram no back-end?**

   Sim, embora a verificação ocorra na camada de interface (app.py), ela implementa regras de negócio claras antes de enviar os dados para o back-end: As principais são:

- Cálculo de Saldo Total: A função bar_valores() calcula o total de receitas e o total de gastos para entregar o saldo atual.
- Agrupamento de Despesas: A função pie_valores() agrupa todos os gastos por categoria e soma seus valores para exibir o gráfico de pizza. Isso permite ao usuário ver para onde seus gastos estão indo.
- Cálculo Percentual: A função porcentagem_valor() calcula o percentual do total de receitas que foi gasto, fornecendo um indicador de saúde financeira.
- Campos Obrigatórios: Todos os campos (nome de usuário, e-mail, senha e confirmação de senha) devem ser preenchidos.
- Tamanho Mínimo do Usuário: O nome de usuário deve ter no mínimo 4 caracteres.
- Tamanho Mínimo da Senha: A senha deve ter no mínimo 4 caracteres.
- Confirmação de Senha: A senha e a confirmação de senha devem ser idênticas.

6. **Vocês criaram APIs para o front-end se conectar?**

   Não, a arquitetura do nosso projeto é uma aplicação monolítica de desktop, significa que o front-end (desenvolvido com CustomTkinter) e o back-end (a classe BackEnd com a lógica de banco de dados) rodam no mesmo processo. A interface gráfica chama diretamente as funções Python (localizadas no módulo view.py) que executam a lógica de negócio e as operações no banco de dados.

7. **Como vocês estão tratando erros?**
   - Tratamento de Entradas Inválidas: Nas funções inserir_receitas_b e inserir_despesas_b, usamos um bloco try-except ValueError para capturar um erro caso o usuário digite um texto no campo “Quantia”, que espera um número. Caso isso aconteça, uma messagebox.showerror é exibida.
   - Tratamento de Ações sem Seleção: Na função deletar_dados, usamos um try-except IndexError para tratar caso o usuário clique no botão “DELETAR” sem ter selecionado um item na tabela, exibindo mensagem de erro.
   - Validação de Campos Vazios: As funções de inserção também verificam se os campos essenciais foram preenchidos antes de tentar salvar os dados.
   - Validações de Negócio: Usamos estruturas if/elif para verificar as regras de negócio (campos vazios, senhas diferentes, etc.) e exibimos mensagens específicas para o usuário através do tkinter.messagebox.
   - Erros Inesperados: Utilizamos blocos try...except para capturar falhas genéricas que possam ocorrer durante as operações de banco de dados ou processamento. Em caso de uma exceção, uma mensagem de erro genérica é mostrada ao usuário, também via tkinter.messagebox.

8. **Estão fazendo testes no back-end? Se sim, como?**

   Os testes estão sendo realizados de forma manual, através da execução da aplicação e verificação do comportamento:

   - Os testes no back-end estão sendo realizados de forma manual. Não há um framework de testes automatizados configurado no projeto.
   - Inserir receitas e despesas e confirmar se os valores nos cards e gráficos são atualizados corretamente.
   - Tentar inserir dados inválidos para garantir que as mensagens de erro apareçam.
   - Adicionar e deletar registros e verificar se a tabela é atualizada como esperado

9. **Onde o código do back-end está hospedado (repositório)?**

   O código está hospedado em um repositório no GitHub e também é mantido localmente.

ESPECIFICAÇÕES / TABELA BANCO DE DADOS

- ➔ **Tabela: Usuarios**

Id INTEGER PRIMARY KEY AUTOINCREMENT,

Username TEXT NOT NULL,

Email TEXT NOT NULL UNIQUE,

Senha TEXT NOT NULL,

Confirma_Senha TEXT NOT NULL

Id - Identificador Único: Um número inteiro que identifica cada usuário de forma única. A gestão deste campo (geração e incremento) é feita automaticamente pelo banco de dados a cada novo registro.

Username - Nome de Usuário: O nome que o usuário escolherá para fazer login no sistema. O campo é de preenchimento obrigatório (NOT NULL).

Email - Email do Usuário: O endereço de e-mail do usuário. Usado para comunicação, recuperação de senha e, em alguns sistemas, como alternativa de login. O campo é obrigatório (NOT NULL).

Senha - Senha do Usuário: A senha de acesso do usuário. O campo é obrigatório (NOT NULL).

Confirma_senha - Confirmação da Senha: Campo utilizado durante o cadastro para garantir que o usuário digitou a senha corretamente (NOT NULL).

- ➔ **Tabela: Categoria_Receita**

Id INTEGER PRIMARY KEY AUTOINCREMENT,

nome TEXT,

user_id INTEGER

Id - Identificador Único: Um número inteiro que identifica cada usuário de forma única. A gestão deste campo (geração e incremento) é feita automaticamente pelo banco de dados a cada novo registro.

nome - Nome da Categoria: Um campo de texto para armazenar o nome da categoria de receita (ex: "Salário", "Vendas", "Rendimentos", "Freelance"). É este nome que será exibido para o usuário na interface do sistema.

user_id - Identificador do Usuário: Um campo do tipo inteiro que estabelece a ligação entre o registro do gasto e um usuário específico. Ele funciona como uma "chave estrangeira", conectando-se ao ID da tabela de usuários para identificar quem realizou a transação.

➔ **Tabela: Categoria_Gasto**

Id INTEGER PRIMARY KEY AUTOINCREMENT,

nome TEXT,

user_id INTEGER

Id - Identificador Único: Um número inteiro que identifica cada usuário de forma única. A gestão deste campo (geração e incremento) é feita automaticamente pelo banco de dados a cada novo registro.

nome - Nome da Categoria: Um campo de texto para armazenar o nome da categoria de gasto (ex: "Aluguel", "Supermercado", "Transporte", "Lazer").

user_id - Identificador do Usuário: Um campo do tipo inteiro que estabelece a ligação entre o registro do gasto e um usuário específico. Ele funciona como uma "chave estrangeira", conectando-se ao ID da tabela de usuários para identificar quem realizou a transação.

➔ **Tabela: Receitas**

Id INTEGER PRIMARY KEY AUTOINCREMENT,

categoria TEXT,

adicionado_em DATE,

valor REAL,

user_id INTEGER

Id - Identificador Único: Um número inteiro que identifica cada usuário de forma única. A gestão deste campo (geração e incremento) é feita automaticamente pelo banco de dados a cada novo registro.

categoria - Categoria da Receita: Um campo de texto que associa a receita a uma das categorias definidas na Tabela Categoria_Receita. Por exemplo, "Salário".

adicionado_em - Data da Transação: Um campo do tipo data que registra o dia em que a receita foi creditada. Essencial para gerar relatórios financeiros por período (mensal, anual, etc.).

valor - Valor da Receita: Um campo numérico de ponto flutuante (REAL) para armazenar o montante monetário da receita, permitindo valores com casas decimais (ex: 1800,40).

user_id - Identificador do Usuário: Um campo do tipo inteiro que estabelece a ligação entre o registro do gasto e um usuário específico. Ele funciona como uma "chave estrangeira", conectando-se ao ID da tabela de usuários para identificar quem realizou a transação.

➔ **Tabela: Gastos**

Id INTEGER PRIMARY KEY AUTOINCREMENT,

categoria TEXT,

retirado_em DATE,

valor REAL,

user_id INTEGER

Id - Identificador Único: Um número inteiro que identifica cada usuário de forma única. A gestão deste campo (geração e incremento) é feita automaticamente pelo banco de dados a cada novo registro.

categoria - Categoria do Gasto: Um campo de texto que associa o gasto a uma das categorias definidas na Tabela Categoria_Gasto. Por exemplo, "Supermercado".

retirado_em - Data da Transação: Um campo do tipo data que registra o dia em que o gasto foi realizado.

valor - Valor do Gasto: Um campo numérico de ponto flutuante (REAL) para armazenar o valor monetário do gasto.

user_id - Identificador do Usuário: Um campo do tipo inteiro que estabelece a ligação entre o registro do gasto e um usuário específico. Ele funciona como uma "chave estrangeira", conectando-se ao ID da tabela de usuários para identificar quem realizou a transação.

Projeto Integrador I

TIME FRONTEND

Informações técnicas do software

- **Quais tecnologias e ferramentas estão usando no front-end?**
  O front-end é uma aplicação de desktop desenvolvida em Python. Estamos utilizando as bibliotecas:
  - CustomTkinter: É uma versão moderna e customizável da biblioteca padrão do Python, a Tkinter.
  - Tkinter: Biblioteca base sobre a qual o CustomTkinter opera, usada também para os messagebox.
  - Pillow (PIL): Para manipulação e exibição de imagens como logos e ícones.
  - Matplotlib: Para a geração dos gráficos de pizza e de barras exibidos no dashboard.
  - Tkinter Treelist (ttk.Treeview): Para a criação da tabela de transações.
  - tkcalendar: Para o componente de seleção de datas (DateEntry).

- **Como está organizada a navegação do sistema?**
  Tela de Login/Cadastro: O usuário inicia na tela de login (do código app.py). A partir dela, pode navegar para a tela de cadastro e depois voltar.

  Tela Principal (Dashboard): Após um login bem-sucedido, a janela de login é destruída e o usuário é direcionado para a tela principal do dashboard (main.py), que funciona como a janela principal da aplicação.

  Ações no Dashboard: Dentro da tela principal, o usuário pode interagir com os formulários para inserir ou excluir dados, sem sair desta tela.

- **Como vocês estão se conectando com o back-end?**
  Chamadas de método diretas. Os componentes da interface (front-end) invocam funções Python importadas do módulo view.py (que atua como a camada de lógica/back-end) para realizar ações como verificar_login, cadastrar_novo_usuario, inserir_gastos e outros.

- **Tem alguma validação de dados no front-end?**
  Sim. Existem várias validações de dados diretamente no código do front-end:
  - Verificação se os campos de login e cadastro estão preenchidos.
  - Validação do comprimento mínimo para nome de usuário e senha.
  - Verificação se a senha e a confirmação de senha são iguais.
  - Validação para garantir que o valor inserido para receitas/despesas é um número válido, usando um bloco try-except para tratar o ValueError.
- **Como está organizada a estrutura do código?**
  Interface do Usuário (app.py, main.py):
  - app.py é a tela inicial de login e cadastro.
  - main.py é o painel principal (dashboard) que o usuário vê após fazer login, com todos os gráficos e informações financeiras.

  Lógica Central (view.py):
  - Este arquivo funciona como a central, o "cérebro" do sistema. Ele conecta a interface ao banco de dados, contendo as regras de negócio e as funções para calcular totais, salvar e deletar transações.

  Banco de Dados (bd.py):
  - É o código que cria a estrutura do banco de dados e todas as tabelas (Usuarios, Receitas, Gastos, e outros).
- **Estão tratando mensagens de erro no front-end? Como?**
  Sim. O tratamento de erros e o feedback ao usuário são feitos através de janelas de diálogo (messagebox) da biblioteca Tkinter. São usados diferentes tipos de messagebox para diferentes situações:
  - messagebox.showerror(): Para erros críticos (ex: senhas não conferem, falha no cadastro).
  - messagebox.showwarning(): Para avisos (ex: campo de senha muito curto).
  - messagebox.showinfo(): Para mensagens informativas (ex: cadastro realizado com sucesso).
- **Estão fazendo algum tipo de teste?**
  Sim, os testes continuam sendo feitos de forma manual.

- **Onde está o código do front-end?**
  O código está hospedado em um repositório no GitHub e também é mantido localmente.

FIGMA

Informações técnicas do protótipo

1. **Link para o projeto no Figma**

    - Link para acesso do projeto no figma https://www.figma.com/design/SNbwae5dRxa4F8iikiHnmK/Poupa%C3%AA?node-id=0-1&t=SXUSIY3vQ9ztk34o-1

2. **Descrição dos protótipos criados**

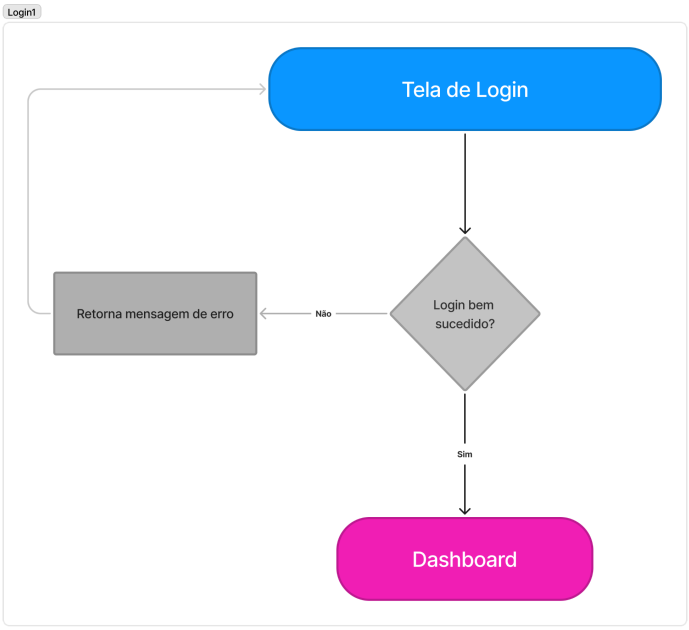
    - **Tela de Login:** Com campos para nome de usuário e senha, botão para visualizar a senha, e botões de "Fazer Login" e "Fazer Cadastro". Há também uma versão com mensagem de erro quando o login falha.
    - **Tela de Cadastro:** Campos para nome de usuário, e-mail, senha e confirmação de senha. Inclui botão de visualização de senha, botão de cadastro e botão para voltar ao login. Versão com mensagem de erro quando o cadastro falha.
    - **Tela de Dashboard (Orçamento Pessoal):** Mostra o saldo mensal, despesas e saldo em caixa. Inclui gráficos (barra e pizza), tabela de receitas/despesas e formulários para adicionar nova receita e despesa. Também há uma versão com mensagem de erro, visível ao tentar adicionar dados com falha.

    - Neste momento o protótipo é só visual.

3. **Fluxo de navegação entre telas**

    - 1. **Login → Dashboard**: após preencher os dados corretamente e clicar em "Fazer Login".
    - 2. **Login → Cadastro**: ao clicar em "Fazer Cadastro".
    - 3. **Cadastro → Login**: ao clicar em "Voltar ao Login".
    - 4. **Dashboard**: é acessada após login bem-sucedido.

    - Pode incluir um diagrama ou até uma imagem do próprio Figma com as setas de navegação. **(OBS: ainda vai ser colocado isso no doc).**



4. **Responsividade**

    - As telas foram desenhadas para desktop, com foco em uma interface mais ampla e detalhada para uso em computadores.
    - Os elementos estão distribuídos de forma horizontal e com aproveitamento do espaço da tela.
    - O dashboard, por exemplo, apresenta múltiplos lançamentos, gráficos, e formulários de entrada lado a lado — algo que seria difícil de adaptar diretamente para dispositivos móveis sem reorganização.

5. **Elementos reutilizáveis / Componentes**

    - Os botões são padronizados com cor, forma e tipografia consistentes.
    - Inputs de texto: com estrutura visual uniforme.
    - Cards e gráficos no dashboard seguem uma mesma estrutura.
    - Ícones como o de “mostrar senha” e indicadores de erro são reutilizados.

6. **Alinhamento com o front-end**

    - O time de front-end segue a mesma estrutura do figma, o que traz uma facilidade para ambos.
    - Elementos estão bem definidos (inputs, botões, espaçamentos, cores).

- Em alguns casos no front foi observado que seria melhor outras cores, mas que já foi atualizado no figma, e o front segue o projeto do figma.

7. **Padrões visuais**

   - O protótipo segue uma identidade visual e coerente em todas as telas.
   - Tipografia: Fonte sem serifa, utilizado de forma consistente em títulos, campos e botões. Os textos explicativos e de apoio estão centralizados ou alinhados à esquerda, com fonte menor.
   - Cores: **Verde escuro** para ações positivas, como fazer cadastro. **Azul escuro** para ações principais, como fazer Login. **Cinza escuro,** deixando com uma aparência limpa e moderna. Dashboard usa tons de **azul**, **cinza** e **verde**, mantendo coerência com as telas iniciais.