SmartBudget

---Controle suas finanças sem complicação.---

Desenvolvido Por:
João Moraes <jpam>
Marcello Menezes <meam>

Sumário

Sobre o Projeto

SmartBudget é uma aplicação desktop leve para monitoramento de despesas, desenvolvida em **C++**. O objetivo é fornecer uma ferramenta simples, eficiente e acessível para auxiliar no gerenciamento financeiro pessoal, com mínimo esforço do usuário.

Estrutura do Projeto

```
C:.
├─────.vscode
├─────docs
├─────lib                          //Bibliotecas externas (ImGui, GLFW)
└─────SmartBudget
    ├─────include                  // Arquivos de  cabeçalho(*.hpp)
    └─────src                      // Implementações em C++
```

Arquitetura Orientada a Objetos

Durante o desenvolvimento, buscamos aplicar de forma prática os principais pilares da programação orientada a objetos, garantindo um código mais organizado, reutilizável e de fácil manutenção. Os conceitos aplicados foram:

Encapsulamento

Cada classe foi projetada para ser responsável por um único conjunto de funcionalidades. Por exemplo, a classe `Transaction` cuida exclusivamente dos dados de uma transação, enquanto a `TransactionManager` lida apenas com a manipulação das mesmas. Isso evita que diferentes partes do código precisem conhecer os detalhes internos de como uma transação funciona, tornando o sistema mais protegido contra erros e alterações inesperadas.

Abstração

O usuário interage com uma interface gráfica simples e intuitiva, sem precisar entender como os dados são processados internamente. Essa separação entre o que é visível e o que acontece "por trás das cortinas" é garantida pelas abstrações oferecidas pelas classes.

Por exemplo, ao clicar em “Adicionar Transação”, a interface chama os métodos do `TransactionManager`, que realiza a lógica sem expor detalhes ao usuário.

Modularidade

O projeto foi dividido em módulos claros, como lógica de negócio, interface com o usuário e gerenciamento de arquivos. Isso permite que partes do sistema possam ser modificadas ou até substituídas sem afetar o restante. Caso decidíssemos, por exemplo, trocar a interface gráfica por outra biblioteca, o restante do sistema continuaria funcionando com poucas alterações.

Coesão e baixo acoplamento

As classes têm uma responsabilidade bem definida (alta coesão) e interagem entre si de forma controlada (baixo acoplamento). Isso torna o código mais limpo e flexível. A AppUI, por exemplo, não gerencia transações diretamente, mas delega isso ao TransactionManager, com quem se comunica através de métodos bem definidos.

Classes

`Transaction`

Representa uma transação financeira. Armazena os dados valor, tipo (renda ou despesa), categoria, data e descrição. Além de fornecer métodos de acesso a essas informações, é responsável pela representação dos seus dados, a partir da função `toCSV()`. É a estrutura base do sistema, servindo como unidade de dados para todas as operações.

*A função print() foi uma funcionalidade importante durante o protótipo do projeto, sendo mantida até hoje caso o usuário escolha pela versão em terminal.

`TransactionManager`

É responsável por gerenciar a lista de transações. Controla a adição e remoção de elementos, bem como a aplicação de filtros. Atua como uma camada intermediária entre os dados e as operações da interface gráfica.

`BudgetAnalyzer`

Realiza os cálculos financeiros com base nas transações disponíveis. Fornece o saldo atual, o total de rendas e despesas, e ainda permite a análise de totais agrupados por categoria. É essencial para a funcionalidade analítica do sistema.

FileManager

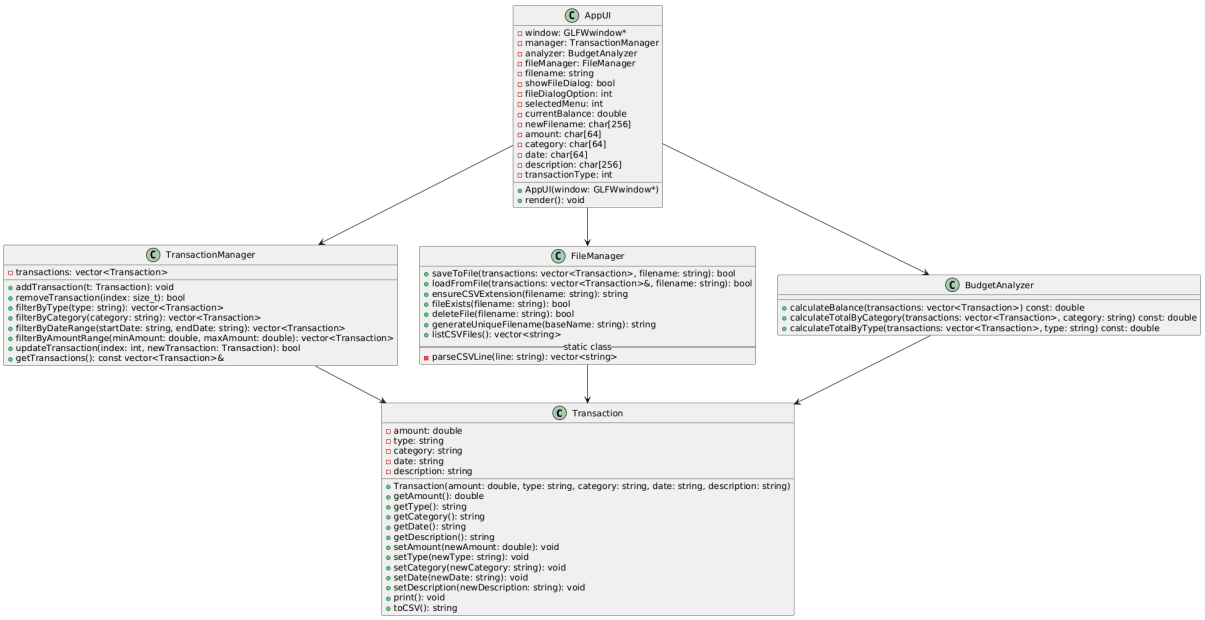
Cuida da persistência dos dados, salvando toda transação em um arquivo .csv criados pelo usuário.É possível ler transações salvas e armazenar novas transações em disco, garantindo que o estado da aplicação seja mantido entre execuções. Também verifica duplicidade de nomes.

AppUI

Responsável por toda a interface gráfica do sistema, utilizando a biblioteca ImGui. Organiza e desenha as janelas e todos os elementos. Além disso, gerencia os estados da interface e encaminha as interações do usuário para as classes apropriadas.

*Existe a AppUI2, que junto com a main2, representa a versão em terminal do projeto, que permaneceu salva no repositório.

Diagrama de Classes



Uso de Bibliotecas Externas

Dear ImGui

Utilizamos esta biblioteca por ser uma biblioteca em C++ voltada para a construção de interfaces gráficas. É uma excelente ferramenta para programas que pedem bom desempenho e facilidade de uso. Toda a lógica visual e interação do usuário com o sistema foi construída com o ImGui.

GLFW

Para que a interface pudesse ser exibida em uma janela própria, utilizamos esta biblioteca, que é responsável por criar e gerenciar a janela principal da aplicação, bem como o contexto do OpenGL. Ela também gerencia a entrada do teclado e do mouse.

OpenGL

O OpenGL foi utilizado como a base gráfica para a renderização dos componentes do ImGui. Ele atua como o motor por trás da parte visual do programa, sendo responsável por desenhar todos os elementos gráficos na tela.

Responsabilidades dos Membros

João Moraes

Responsável pela arquitetura geral do sistema e implementação do núcleo lógico. Estrutura das classes principais, garantindo a separação adequada de responsabilidades, a modularidade e a clareza do código. Integração entre os componentes internos, testou os fluxos da aplicação.

Marcello Menezes

Responsável pelo desenvolvimento da interface gráfica do projeto utilizando a biblioteca Dear ImGui. Criação de todas as telas e menus, incluindo os módulos de transações, relatórios e manipulação de arquivos, garantindo uma experiência de usuário fluida, responsiva e intuitiva. Organização visual do sistema, alinhando os elementos de interface com as funcionalidades internas do código.