Sistema de caixa

Nomes: Henrique Neves Braga
Felipe Carvalho

Sobre o programa

É uma aplicação de caixa que pode servir para lojas, supermercados e afins.

A aplicação é operada por um funcionário do estabelecimento e permite cadastro de vendas em um banco de dados que será acessado através de uma API.

A aplicação também permite cadastrar produtos, descontos, clientes (para um sistema de fidelidade) e funcionários.

Existe também a opção de reembolso e de gerar um relatório com as informações das vendas de um determinado mês.

Backend

Usamos FastAPI para construir uma API restful em python, pois ela é simples, requer pouco código e é bem otimizada.

Usamos SQLAlchemy para abstrair a conexão com o banco de dados. Com isso o código se torna mais independente do banco de dados se tornando fácil trocar caso necessário.

Backend

O backend disponibiliza as seguintes funcionalidades

- CRUDs para Funcionários, Clientes, Produtos e Vendas
- Validação de login
- Vendas associadas a um cliente
- Vendas de um determinado mês

Backend: SQLAlchemy

Separação de responsabilidades: O SQLAlchemy separa a lógica de negócios da lógica de acesso ao banco de dados, facilitando a testabilidade e a reutilização do código.

Flexibilidade e escalabilidade: O SQLAlchemy suporta diversos tipos de bancos de dados, permitindo troca de banco de dados no futuro sem precisar modificar significativamente o código.

Backend: FastAPI

O código do backend é separado de forma que o código da requisição, a interação com o banco de dados e o response_model estão em arquivos diferentes.

Isso garante:

- Modularidade
- Manutenibilidade
- Reusabilidade

```python
@app.post("/produtos/", response_model=schemas.ProdutoBase)
def create_produto(produto: schemas.ProdutoCreate, db: Session = Depends(get_db)):
    return crud.create_produto(db=db, produto=produto)
```

```python
def create_produto(db: Session, produto: schemas.ProdutoCreate):
    db_produto = models.Produto(
        nome = produto.nome,
        valor=produto.valor,
        quantidade_estoque=produto.quantidade_estoque
    )
    db.add(db_produto)
    db.commit()
    db.refresh(db_produto)
    return db_produto
```

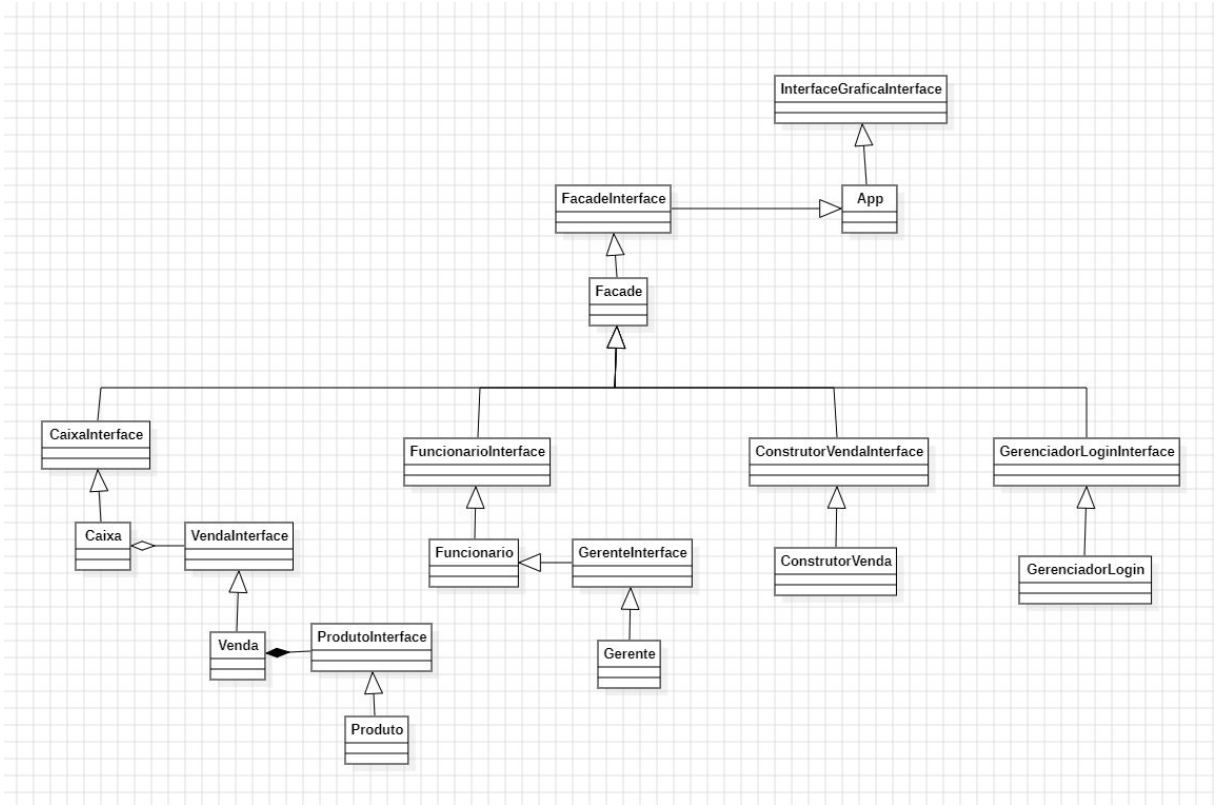
```python
class ProdutoCreate(ProdutoBase):
    valor: float
    quantidade_estoque: int
    nome:str
```

Frontend

Frontend feito em typescript, css e html para garantir portabilidade já que são suportados por quase todos os navegadores.

Tudo funciona em uma página html manipulada por um script que manipula os elementos da interface para dar a impressão de navegar por diferentes páginas.

Diagrama de classes do frontend



Princípios SOLID

Singles Responsibility: Cada classe cuida de uma área específica.

Open-Closed: Novas funcionalidades não precisam modificar métodos existentes.

Liskov-Substitution: Gerente herda funcionário e pode fazer tudo que ele pode.

Interface Segregation: Não há classes que implementam interfaces com métodos que elas não usem.

Inversão de dependência: As classes não dependem de outras classes mas sim de interfaces.

Padrões GRASP

Information expert: Classes como Venda e Produto fornecem informações sobre si mesmos.

Creator: A classe Facade que é a única que utiliza Funcionario/Gerente, Caixa, GerenciadorLogin e ConstrutorVenda é responsável por criá-las.

Baixo acoplamento: Com o uso de interfaces as classes não dependem da interpretação uma das outras.

Alta coesão: As classes possuem responsabilidades relacionadas entre si.

Padrões GoF: Facade

Para fornecer uma interface única e simplificada para o acesso às funcionalidades das classes Caixa, Funcionário e Gerente, ConstrutorVenda e GerenciadorLogin.

Quando um botão é apertado a classe de interface gráfica chama o respectivo método do facade que por sua vez chama as classes adequadas. Dessa maneira não é preciso conhecer os detalhes da implementação interna para criar a interface.

Padrões GoF: Observer

Um objeto da classe ConstrutorVenda é responsável por construir o objeto venda ao mesmo tempo que o usuário adiciona e remove os produtos na interface. Quando o usuário termina de ajeitar os produtos e clica em cadastrar, usando o padrão observer, o objeto construtor de venda avisa ao caixa para que ele possa atualizar o bancos de dados.

Nesse caso o caixa observa o construtor de vendas.

Padrões GoF: Singleton

Para garantir que exista apenas uma instância da classe Caixa em todo o sistema. Isso centraliza a lógica de registro de vendas e geração de relatórios, evitando redundâncias e inconsistências.

Funcionalidades Transacionais

Cadastro de venda: Consulta as informações de produtos e descontos no banco de dados e calcula o valor da venda, após isso verifica se existem produtos suficientes em estoque, em caso afirmativo desconta a quantidade vendida do estoque e cadastra a venda.

Relatório de vendas de um mês: Lê do usuário um mês e um ano e mostra as informações das vendas desse mês em uma tabela.

Evolução para uma LPS - Características mandatórias

Cadastro e login de funcionários

Registro de vendas

Controle de estoque

Sistema de pagamento

Sistema Reembolso

Evolução para uma LPS - Características opcionais

Sistema de fidelidade de clientes

Sistema de desconto para produtos específicos

Gerenciar funcionários, produtos e descontos

Relatório de vendas

Evolução para uma LPS - Características alternativas

Controle de estoque com validade para produtos perecíveis (supermercados, farmácias).

Gerenciamento de mesas e pedidos com divisão de conta para restaurantes e bares.