Relatório de Desenvolvimento: Sistema de Gestão de Estoque e Pedidos

Membros da Equipe
- Adailton Lima Santos Segundo – RA 1272219962
- Caio Lucius Nascimento Sales – RA 12722117059
- Gabriel Badaró – RA 12722116786

Link do video demonstração : https://youtu.be/pvGSamY-eTc

Sumário

1. Introdução

1.1 Contextualização e Justificativa

Com base no cenário dos negócios, é visto que a eficiente gestão de estoque e pedidos é vital para o sucesso de qualquer empreendimento. Diante desse desafio, desenvolvemos um Sistema de Gestão de Estoque e Pedidos, com o objetivo de proporcionar às empresas uma ferramenta que otimize seus processos.
Nossa motivação para o desenvolvimento desse sistema veio da necessidade de integração entre controle de estoque, gestão de usuários e processos de pedidos.

1.2 Objetivo do Sistema

O objetivo principal do sistema é fornecer uma plataforma para gerenciar o ciclo de vida de produtos, desde a entrada no estoque até a realização de pedidos pelos clientes.

1.3 Pré-requisitos e Manual de Instalação

O Sistema de Gestão de Estoque e Pedidos foi desenvolvido em Python. Após a instalação do Python e o clone do repositório, a inicialização do ambiente virtual e a configuração do sistema são simplificadas por meio do script "startup.bat". Este script automatiza a instalação de dependências, a inicialização do servidor e a abertura do terminal com o front-end do sistema.

Comandos que podem ser necessários no terminal :

pip install termcolor
pip install fastapi
pip install sqlalchemy
pip install uvicorn
pip install httpie
pip install colorama

2. Tecnologias Utilizadas e Fundamentação Teórica

2.1 Framework Web: FastAPI

O FastAPI foi escolhido não apenas pela sua capacidade de proporcionar alto desempenho, mas também pela sua sintaxe declarativa e intuitiva. As anotações nas rotas não apenas simplificaram o desenvolvimento, mas também contribuíram para a definição das APIs RESTful, facilitando o processo de documentação automática.

2.2 Banco de Dados: SQLite com Alembic para Migrações

A escolha do SQLite como banco de dados foi motivada pela sua natureza leve, tornando-o ideal para aplicações menores. A integração do Alembic para migrações permite a evolução controlada do esquema do banco de dados.

2.3 ORM: SQLAlchemy

O SQLAlchemy, atuando como um mapeador objeto-relacional, simplificou significativamente a interação com o banco de dados. A capacidade de manipular entidades como objetos Python definiu a estrutura do banco de dados de maneira eficiente, e também proporcionou praticidade para operações CRUD.

2.4 Biblioteca de Validação: Pydantic

A biblioteca Pydantic desempenhou um papel crucial no processo de validação de dados nas interações da API. Os modelos Pydantic, responsáveis por definir a estrutura dos dados esperados nas requisições, não apenas ofereceram validação automática, mas também garantiram a integridade e consistência dos dados.

3. Arquitetura do Sistema

3.1 Models

Os models definem a estrutura do banco de dados e são manipulados intuitivamente pelo SQLAlchemy, espelhando a estrutura do banco de dados.

3.2 API FastAPI

A API foi estruturada seguindo os princípios RESTful. As rotas definidas nas funções correspondem às operações do CRUD e aos relatórios, e as anotações são utilizadas pelo FastAPI para a geração automática de documentação.

3.3 Database

O arquivo database.py centraliza a configuração do banco de dados. A função get_db serve como dependência para obter uma sessão do banco de dados sempre que necessário.

3.4 Dependência para Obter uma Sessão do Banco de Dados

A função get_db é utilizada como dependência nas rotas da API, fornecendo uma sessão do banco de dados que é automaticamente encerrada ao final da requisição.

4. Funcionalidades Principais

4.1 CRUD de Itens no Estoque

As operações de CRUD de itens no estoque são gerenciadas pelas rotas correspondentes no arquivo main.py, utilizando o model ItemDB.

4.2 Gestão de Usuários

A gestão de usuários é implementada através das rotas, utilizando o model UsuarioDB para realizar as operações de CRUD associadas aos mesmos.

4.3 Pedidos

A realização de pedidos é gerenciada pela rota add_pedido. A função verifica a disponibilidade em estoque do item solicitado, realiza a atualização do estoque e cria um novo pedido associado ao usuário.

5. Relatórios

5.1 Produtos Mais Vendidos

O relatório get_top_sold_products utiliza uma consulta SQL para identificar os produtos mais vendidos, retornando uma lista de objetos Pydantic do tipo ProductReport.

5.2 Produtos por Cliente

O relatório get_product_by_customer_report realiza uma consulta complexa utilizando joins entre as tabelas ItemDB, PedidoDB, e UsuarioDB. O resultado é uma lista de objetos Pydantic do tipo ProductCustomerReport, contendo informações sobre os produtos comprados por cada cliente.

5.3 Consumo Médio por Cliente

O relatório get_avg_consumption_by_customer_report calcula o consumo médio por cliente, considerando a quantidade e o preço dos produtos em seus pedidos. A função retorna uma lista de objetos Pydantic do tipo AvgConsumptionReport, detalhando o nome do cliente, o produto mais comprado, o consumo médio em quantidade e o consumo médio em preço.

5.4 Produtos em Baixo Estoque

O relatório get_low_stock_products_report identifica os produtos com estoque igual ou abaixo de 3 unidades, retornando uma lista de objetos Pydantic do tipo LowStockProductReport.

6. Projeto de Implementação

6.1 Estrutura do Código

A estrutura do código é organizada em módulos, facilitando a manutenção e expansão do sistema. Os arquivos principais incluem main.py, database.py, e models.py.

6.2 Modelagem e Diagramas

A modelagem inclui os models definidos pelo SQLAlchemy, representando as entidades do sistema. Diagramas como UML e Entidade-Relacionamento foram utilizados para elucidar a estrutura do banco de dados e a relação entre entidades.

7. Considerações Finais

O desenvolvimento do sistema foi conduzido atendendo aos requisitos específicos do projeto. A escolha das tecnologias proporcionou uma base sólida para uma aplicação eficiente e flexível. O sistema apresenta funcionalidades essenciais para o controle do estoque, gestão de usuários e realização de pedidos, além de fornecer relatórios detalhados sobre as operações. A estrutura modular e a documentação automática gerada pelo FastAPI contribuem para a compreensão e manutenção do código. O uso de modelos Pydantic, o padrão ORM do SQLAlchemy e a utilização de migrações com o Alembic garantem a integridade e evolução controlada do banco de dados.

8. Bibliografia

termcolor · PyPI
FastAPI (tiangolo.com)
SQLAlchemy Documentation — SQLAlchemy 2.0 Documentation
Uvicorn
HTTPie – API testing client that flows with you
colorama · PyPI