Relatório Completo do Desenvolvimento e Resolução de Problemas

Configuração Inicial e Estrutura do Projeto

- Estrutura de Pastas e Arquivos:
  - Criei uma estrutura clara e organizada com uma pasta principal (`app`) para configurações do projeto e uma pasta específica da aplicação (`stock_app`) para modelos, views, serializers e outros arquivos relevantes. Isso ajuda na organização e facilita a compreensão do projeto por outros desenvolvedores.

- Configuração do Django e Django Rest Framework:
  - Adotei o Django devido à sua eficiência e ao Django Rest Framework para facilitar a criação de APIs, aproveitando recursos como serialização e autenticação.

- Modelos:
  - Implementei modelos utilizando o ORM do Django para `Vendedor`, `Comprador`, `Produto`, `Venda`, `Compra`, e `Estoque`, permitindo uma manipulação segura e eficiente do banco de dados.

- Serializers:
  - Configurei serializers para converter os dados dos modelos em JSON, simplificando a interação com a API.

- Views e Roteamento:
  - As views foram configuradas com o Django Rest Framework para gerenciar as requisições HTTP. O roteamento foi cuidadosamente mapeado para facilitar o acesso aos endpoints da API.

- Administração do Django:
  - Ativei o painel de administração do Django para permitir um gerenciamento fácil dos dados durante o desenvolvimento.

- Banco de Dados MySQL:
  - Escolhi o MySQL pela sua robustez e capacidade de gerenciar grandes volumes de dados e transações, configurado para trabalhar harmoniosamente com o Django.

Desafios e Soluções

1. Configuração do Ambiente de Desenvolvimento
   - Enfrentei a necessidade de preparar o ambiente com todas as dependências necessárias, o que foi resolvido com a instalação e configuração do Python, Django, Django Rest Framework, e MySQL.

2. Migrações e Banco de Dados
   - Tive um problema onde as tabelas necessárias não foram criadas, pois as migrações não foram aplicadas. Isso foi solucionado verificando e aplicando as migrações corretamente.

3. Endpoints de Relatório

- Implementei endpoints para geração de relatórios em PDF e Excel. Enfrentei um problema com a geração de Excel devido a um erro de método em `XlsxWriter`. Resolvi ajustando o método de fechamento do arquivo de Excel para usar `close()` dentro de um contexto gerenciado.

4. Testes
- Escrevi testes unitários para validar a funcionalidade básica dos modelos e endpoints. Também configurei testes de integração para verificar a interação entre diferentes partes do sistema, como a atualização de estoque após uma venda.

5. Desempenho e CI/CD
- Propos testes de desempenho para garantir que o sistema possa lidar com altas cargas de requisições. Além disso, iniciei a configuração de uma pipeline de CI/CD usando GitHub Actions para automatizar a execução dos testes sempre que mudanças são feitas no código.

Funcionalidades Desenvolvidas

- Desenvolver uma API Rest de controle de estoque com os seguintes módulos:
  - Venda
  - Vendedor
  - Compra
  - Comprador
  - Estoque

- Criar um endpoint para gerar um relatório das vendas efetuadas, filtrar por data ou por vendedor ou por cliente e exportar em pdf e para excel.
  - Implementei endpoints `/vendas/report-pdf/` e `/vendas/report-excel/` para gerar relatórios de vendas.

- Criar teste unitário e de integração em um dos módulos a sua escolha.
  - Escrevi testes unitários e de integração para os modelos de `Venda`, `Compra`, `Produto`, `Vendedor`, e `Comprador`.

- Consumir produtos da API externa (https://github.com/keikaavousi/fakestore-api)
  - Adicionei um endpoint `/importar-produtos/` que consome produtos da API externa `https://fakestoreapi.com/products` e os armazena no banco de dados.

- Criar forma de personalizar o preço do produto na venda e/ou compra
  - Atualizei os modelos, serializers e views para incluir o campo `preco_final` permitindo a personalização do preço durante a criação de vendas e compras.

- Demonstrar conhecimentos em Docker, orquestração de containers utilizando: Kubernetes, Docker Swarm, Rancher ou outros, e utilizando serviços de computação em nuvem como: GCP, Aws, Oracle (pode ser em vídeo explicativo, arquivo de configuração, outros).
  - Criei um `Dockerfile` para a aplicação e uma configuração básica de Kubernetes (`deployment.yaml`) para demonstrar a implantação da aplicação.

Passo a Passo do Projeto

1. Configuração Inicial:
   - Instalei Django e Django Rest Framework.
   - Configurei o banco de dados MySQL.

2. Desenvolvimento dos Modelos:
   - Criei modelos para `Vendedor`, `Comprador`, `Produto`, `Venda`, `Compra`, e `Estoque`.

3. Configuração dos Serializers:
   - Configurei serializers para converter instâncias de modelos em JSON.

4. Implementação das Views:
   - Configurei views para gerenciar as requisições HTTP e operações CRUD.

5. Criação dos Endpoints de Relatório:
   - Implementei endpoints para geração de relatórios em PDF e Excel.

6. Criação e Execução dos Testes:
   - Escrevi testes unitários e de integração para garantir a funcionalidade dos endpoints e modelos.

7. Integração com API Externa:
   - Adicionei a funcionalidade para consumir produtos da API externa `fakestoreapi`.

8. Personalização de Preço na Venda e Compra:
   - Adicionei lógica para permitir a personalização do preço final durante a criação de vendas e compras.

9. Configuração de Docker e Kubernetes:
   - Criei um `Dockerfile` e uma configuração básica de Kubernetes para implantação da aplicação.

Ferramentas e Tecnologias Utilizadas

- Django: Framework web de alto nível para desenvolvimento rápido e design limpo.
- Django Rest Framework: Conjunto de ferramentas para criar APIs web em Django.
- MySQL: Sistema de gerenciamento de banco de dados relacional robusto.
- Factory Boy: Ferramenta para criar dados de teste no banco de dados.
- Postman: Ferramenta para testar APIs HTTP.
- pytest: Framework de testes para Python.
- Git: Sistema de controle de versão.
- GitHub Actions: Plataforma de CI/CD para automação de fluxos de trabalho.
- Docker: Plataforma para desenvolvimento e implantação de aplicações em containers.