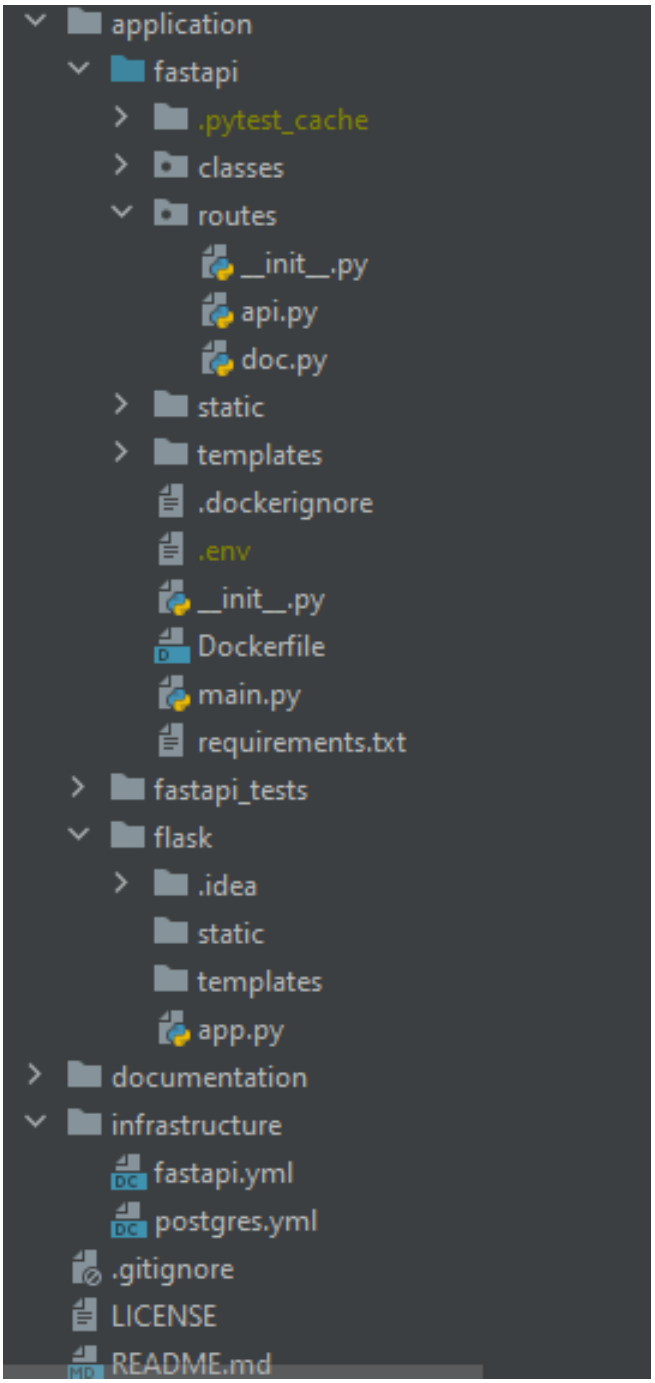
Documentação de API para armazenamento de arquivos em banco de dados relacional

Arquitetura

Para o desenvolvimento desta solução foi utilizada a versão 3.8 do python, o framework fastapi e o banco de dados relacional postgres.

Folder Structure



a aplicação é dividida em 3 pastas principais sendo:

- infraestructure: contém os arquivos referentes ao build e automação. Nesta pasta estão contidos os arquivos para montagem do banco de dados relacional postgres(postgres.yml) e para o build da aplicação (fastapi.yml).
- documentation: contém arquivos e documentos passados e utilizados durante o desenvolvimento da aplicação.
- application: contém todos os códigos referentes a montagem e execução do código. Este diretório contem todos os arquivos do projeto divididos entre cada módulo isolado de build ao qual sera utilizado.
    - fastapi: com ponto de montagem deste diretório o arquivo main contem a chamada run a qual coloca o servidor no ar. Foi utilizado o factory builder pattern para evitar o problema de importação cíclica e para a adição ou edição dos endpoints referentes as rotas deve-se acessar a pasta routes.
    - fastapi_test: alguns testes simples para avaliar se os endpoints estão funcionais(Obs.: os testes apresentados são bem simples, queria ter mais tempo para montar algo mais complexo com avaliação de tempo e resposta).

Execução

Após rodar a aplicação, acessar o endpoint http://localhost:8080/ escolher o arquivo ao qual sera armazenado no banco de dados e enviar para o servidor.

Caso exista qualquer dúvida referente ao uso pode-se acessar o endpoint http://localhost:8080/docs ou http://localhost:8080/redocs ao qual possuem 2 versões distintas de documentação montadas a partir do swagger.

Funcional

A aplicação cria uma tabela usando o nome do arquivo como base para o nome e adiciona os campos os quais serão inseridos por cada linha do arquivo.

Ao final da execução sera persistido no banco de dados todas as linhas do arquivo.

Decisões Funcionais

- Sobre a modelagem do banco foi evitado o uso de “joins” para acelerar buscas executadas apos a inserção, logo foi criado apenas uma tabela contendo todos os campos e aplicadas as validações durante a criação.
- Sobre campos de chave, em um primeiro momento pensei ser CPF, porem depois de alguns testes percebi conterem outras informações na mesma coluna, referentes a outra documentação, logo adotei a tática de deixar o campo mais livre sendo adicionado apenas a “string” sem nenhuma higienização no campo(Essa tática ocupa um pouco mais de espaço porem, me permite ter toda a informação caso seja necessária depois para outros métodos de validação). Foi adotado como chave um “id” referente ao número da linha por não se ter nenhuma garantia que a primeira coluna não possui itens repetidos.
-

Testes e tempo de execução

Existe dentro do diretorio documantation um arquivo postman colection contendo uma coleção com testes para os endpoints.

Existe o diretório de testes da aplicação(./application/fastapi_tests) o qual roda alguns testes funcionais a partir do comando “python -m pytest”

De forma geral os testes em minha maquina com o docker limitado a 4GB de ram executaram em cerca de 58s. Sendo pelos testes ~7s para criação e validação dos itens e ~51s para inserção de 49998 registros, sendo ocupado 6.8 mB de memória para armazenamento do arquivo.

Melhorias e TODO

Para melhorar o tempo de execução uma das opções seria o uso de um do ORM marshmellow optimizada para esse serviço, porém acredito que este recurso não traria tantos ganhos.

Como proposta para melhoria seria o uso de um framework de paralelização para executar esse serviço de inserção como exemplo spark.