A proposta deste desafio é o desenvolvimento de uma API (backend) para a ferramenta denominada ***insights***, onde a mesma irá conter informações sobre cards. Seguindo a descrição do desafio, foi implementado funcionalidades CRUD (Create, Read, Update, Delete) para as entidades ***Card*** e ***Tag***.

Um card é composto pelas seguintes informações:

- id
- texto
- data_criacao
- data_modificacao
- tags

Uma tag é composta por:

- id
- name

Decidi utilizar um banco de dados relacional (**MySQL**) para armazenar essas informações, e logo de cara podemos identificar um relacionamento de muitos para muitos (many-to-many) entre as tabelas, visto que um card pode ter várias tags e uma tag pode estar presente em vários cards. Diante disso, foi necessário realizar a normalização destas tabelas, gerando outra tabela chamada ***card_has_tag***, onde a mesma armazena os ids da tabela card e da tabela tag, onde tais ids compõem a chave primária desta tabela. Porém, poderia também utilizar um banco de dados NoSQL ou não relacional, como, o popular **MongoDB**, onde seria de certa forma mais simples mitigar esse problema de relacionamento, pois seria possível simplesmente armazenar um campo de tags em forma de Array na coleção (tabela) de cards contendo os ids das tags que pertencem aquele card.

O desenvolvimento da API foi feito na linguagem Python, utilizando o framework FastAPI. Utilizei também o SQLAlchemy como ORM, Alembic como gerenciador de versões das migrations do banco de dados, o Poetry como gerenciador de pacotes, e por fim, utilizei o docker para criar um container para a API. Basicamente as rotas da API são um CRUD, exceto de uma rota para listar os cards passando uma tag como filtro. Quando se tenta criar um card, é preciso passar um objeto contendo essas informações, esse objeto será serializado pelo **Pydantic**, usado para construir os schemas. Se é passado uma tag ainda não cadastrada, essa mesma tag além de ser vinculada ao card, será adicionada ao banco, e caso o usuário passe uma tag já existente, a mesma será somente vinculada ao card, pois o campo *name* da tabela tag é um índice único.

Como parte do desafio, também se pede que se crie uma CLI que possibilita importar cards para a ferramenta, onde os dados dos cards estão em arquivo csv. Utilizei os mesmos métodos utilizados na API para interagir com o banco de dados, usando a biblioteca *pandas* para ler o csv em formato de DataFrame, em seguida itero sobre cada card (linha do dataframe), e salvo as informações no schema do card, então, armazeno o card no banco de dados.

Para facilitar os testes para quem fazer o clone do projeto, subi para este repositório um arquivo sqlite que será o nosso banco de dados fictício, porém localmente utilizei um servidor mysql, caso deseje testar, é só usar o método *connect_to_db()* do arquivo presente em *src/db/conn.py*, esse arquivo tem os métodos de conexão com o banco, no caso da conexão com o mysql, é preciso passar as credenciais de conexão com o servidor mysql instalado em sua máquina em um arquivo *.env*, o template desse arquivo está no em *.env.example.*

Os testes das rotas com funcionalidade CRUD também foram implementados, basicamente estes testes verificam os códigos HTTP de retorno das chamadas as rotas da API. Para realizar os testes, no terminal na raiz do projeto, digite *pytest*, e ele irá realizar os testes.

As instruções de execução estão no Readme do projeto.