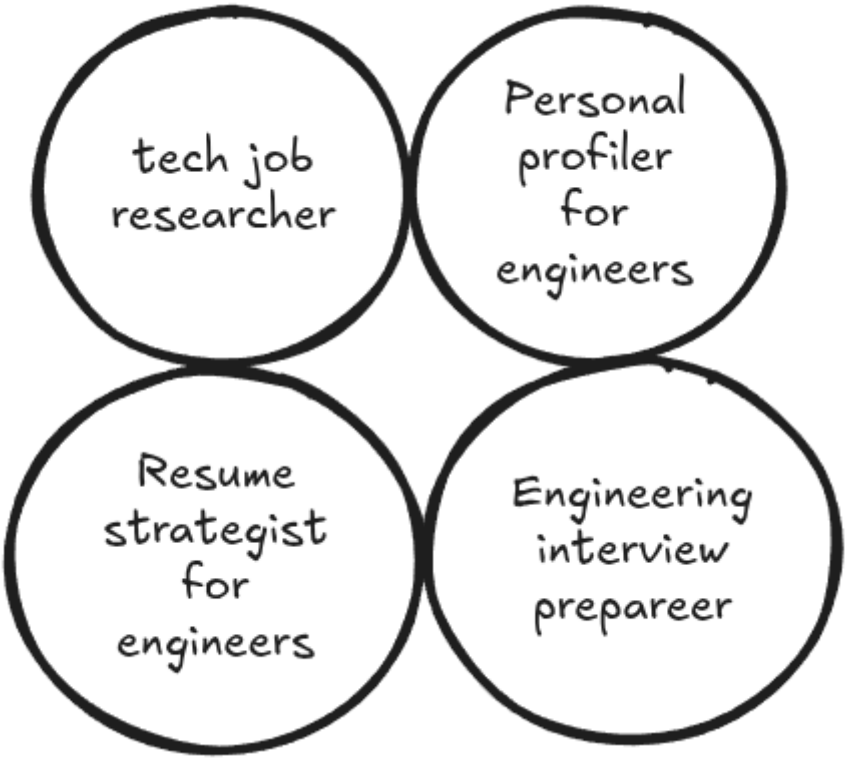
Relatório 3 - Validação de dados com Pydantic

Marcelo do Nascimento Oliveira Soares

Descrição da atividade

Em Overview, recebemos um exemplo básico de um time de agentes de IA, constituído por:



Ele apresenta isso em uma analogia em que Noah precisa se aplicar para uma vaga de trabalho e suas habilidades precisam corresponder às habilidades que a vaga precisa. No processo de se aplicar a uma vaga, além das habilidades, é necessário saber apresentá-las, o time de agentes trabalha nesse sentido, acessando as especificações da vaga, analisando as habilidades de quem está se aplicando e reescrevendo da melhor forma uma apresentação que contemple as habilidades do interessado e os requisitos técnicos da vaga.

Além disso, no contexto de automação ele fala sobre o tema fuzzy inputs. Antigamente os tipos de dados que seriam usados na entrada por algum usuário eram claros, poderiam ser strings ou dados numéricos, de qualquer forma era algo planejado, mas hoje, no contexto da inteligência artificial, é possível fazer um processamento de dados diferente que permite vários tipos de entradas. Como exemplo, temos o Chat-Gpt, que pode receber tabelas, imagens, números, códigos, textos e outros formatos.

Em seguida temos acesso ao vídeo que fala sobre porque devemos usar Pydantic e a resposta é, validar dados de forma rápida e eficiente. No vídeo ArjanCodes apresenta a classe User, que herda de BaseModel, um dos módulos do pydantic, formas de validar as entradas que a classe vai receber. Por exemplo, no campo email a entrada recebe o atributo EmailStr, que garante que esse campo será preenchido com algum tipo de “example@arjancodes.com” e caso a entrada seja diferente, como “examplearjacondes.com” ou “example@arjancodes”, o módulo irá sinalizar um erro nessa entrada.

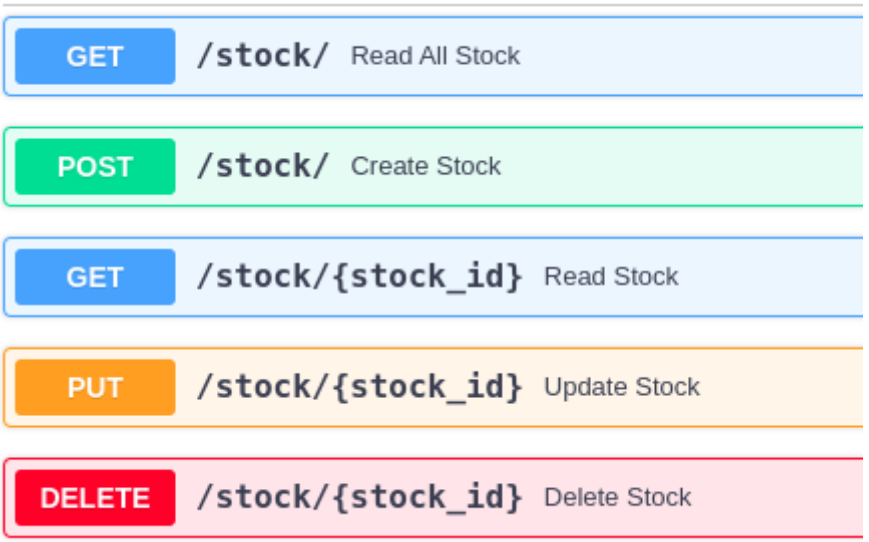
Além dessa classe, o autor cria a função validate, que vai herdar da classe User as entradas e a capacidade de validar dados, então com ela, ele consegue fornecer exemplos de entradas corretas e incorretas. Por fim, ele faz um teste em uma função main, que contém um exemplo válido e outro inválido e a execução da função validade. No primeiro exemplo a saída são os

valores corretos e no segundo a saída é “User is invalid” com o detalhamento do erro. Abaixo deixo imagens referentes à classe User e à função validate:

```python
#Classe User
class User(BaseModel):
    #O nome, email, senha e role, serão inseridos como como entrada na instância da classe User.
    #Note que o conteúdo dos campos é definido pelos módulos importados de pydantic.
    #Eles fazem uma "pré-validação" dos dados que serão inseridos.
    name : str = Field(examples= ["Arjan"])
    #O email deve conter a estrutura de um email "abc@def.com", caso contrário haverá um erro na instanciação
    email : EmailStr = Field(
        examples = ["exampleam@arjancodes.com"],
        description = "The email adress of the user",
        frozen = True,
    )
```

```python
#validade recebe um dicionário que contém os valores da classe Usuário e faz a validação deles.
def validate(data: dict[str, Any]) -> None:
    try:
        user = User.model_validate(data)
        print(user)
    except ValidationError as e:
        #No caso de uma entrada inválida, a saída irá conter os erros de forma a possibilitar
        #  de forma simples a correção da falha.
        print("User is invalid")
        for error in e.errors():
            print(error)
```

Na parte prática resolvi utilizar o exemplo apresentado no vídeo só que em algum caso particular. Encontrei no Medium um artigo que falava sobre o uso das bibliotecas Pydantic e FastAPI, então resolvi utilizar elas para validar entradas em uma API para gestão do estoque de um bar ou restaurante.



,

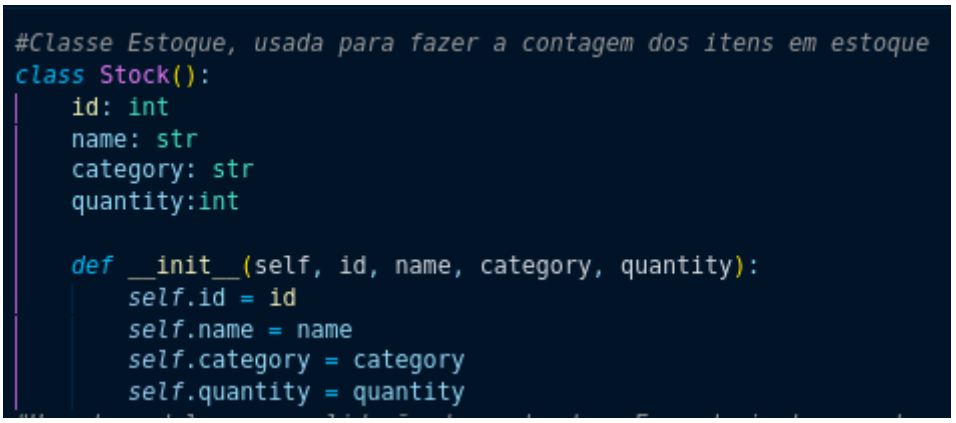
O primeiro método é o GET, ele faz a leitura e mostra ao usuário todos os itens registrados. Em um primeiro momento, em que não há adições feitas, tudo que ele vai mostrar são os exemplos inseridos na lista que armazena os dados. O segundo método utilizado é o POST, que permite ao usuário adicionar qualquer item à lista, basta que ele preencha todos os campos necessários como id, nome, categoria e quantidade. O terceiro método é o GET para ler um item específico a partir de seu id, é útil para visualizar um item específico antes de atualizá-lo ou apagá-lo. O penúltimo método é o PUT, utilizado para editar ou atualizar registros anteriores, o sistema faz a busca a partir do id especificado e permite ao usuário alterar todos os atributos com exceção do id, que permanece fixo. Por último, o método Delete simplesmente deleta o item especificado.

Abaixo deixo imagens referentes às saídas da função que visualiza todos os itens e da função que permite o usuário a inserção de algum item:

Método GET(Visualizar)

Método POST(Inserir):

Abaixo irei apresentar o código da minha proposta:



Na classe Stock eu defino o padrão que será utilizado, as entradas para uma instância são o id, o nome, a categoria e a quantidade, cada um com um respectivo tipo de entrada. Essa classe ainda não herda nenhuma validação fornecida pelo pydantic, a seguinte irá herdar de BaseModel. A classe seguinte é a Stock_Request, que será utilizada pela FastAPI para validar os dados que entrarão na lista que irá armazenar todas as informações. O campo name deverá ter um mínimo de 3 caracteres, o campo categoria também, enquanto o campo quantidade e id deverão ter o tamanho mínimo igual ou superior a 0, para garantir que não haverão entradas negativas. O Pydantic garante que essa regras serão seguidas e caso não sejam, haverá um sinal de ValidationError, impedindo que a instância seja criada normalmente. Além dessas informações e validações de entrada, criei a classe Config, dentro da classe Stock_request, com a finalidade de definir uma estrutura json que irá ser utilizada pela API em /docs

Após isso iniciei a app com o FastAPI e criei os endpoints referentes aos métodos HTTP já citados anteriormente. Acredito que o único endpoint relativamente complexo foi o do método PUT, que contou com um endereçamento dinâmico e um loop que busca e altera um item.

Abaixo deixo as últimas imagens desse relatório. A primeira imagem mostra a classe Stock_Request e sua estrutura, enquanto a segunda mostra a configuração dos endpoints utilizados pela API:

Classe Stock_Request

```python
class Stock_Request(BaseModel):
    id: int = Field(ge=0,
        description="É necessário fornecer ao sistema o identificador referente ao produto",
        )
    #Nome do produto
    name: str = Field(
        min_length=3,
        description= "nome do produto",
        examples=["Absolut Vodka", "Tanqueray London Dry Gin"]
    )
    #Categoria, sendo algo como bebida, alimento...
    category: str = Field(
        min_length=3,
        description= "Tipo de produto, bebida, equipamento ou insumo",
        examples=["Laranja","Vodka","Gin","Whisky","Energético,","Refrigerante"]
    )
    #Quantidade atual do produto
    quantity: int = Field(ge=0)
```

Endpoints:

```python
#Método post para adicionar item a lista
@app.post('/stock/', status_code=status.HTTP_201_CREATED)
async def create_stock(stock_request: Stock_Request):
    new_stock = Stock(**stock_request.model_dump())
    Lista.append(new_stock)
    return new_stock
#método get para visualizar os itens da lista
@app.get('/stock/', status_code=status.HTTP_201_CREATED)
async def read_all_stock():
    return Lista
#método get para pesquisar um item único na lista
@app.get('/stock/{stock_id}', status_code= status.HTTP_200_OK)
async def read_stock(stock_id: int = Path(gt=0)):
    for stock in Lista:
        if stock.id == stock_id:
            return stock
    raise HTTPException(status_code=404, detail='Stock not found')
#método put, para atualizar itens da lista
@app.put('/stock/{stock_id}', status_code=status.HTTP_200_OK)
async def update_stock(stock_id:int, stock_request:Stock_Request):
    for i, stock in enumerate(Lista): #procura item e quando acha faz a atualiação
        if stock.id == stock_id:
            updated_stock = Stock(      #tive que instanciar manualmente,
                #pois (id = stock_id, **stock_request.model_dump()) estava dando conflito
                id = stock_id,
                name = stock_request.name,
                category = stock_request.category,
                quantity = stock_request.quantity
            )
            Lista[i] = updated_stock
            return updated_stock
    raise HTTPException(status_code= 404, detail= 'Stock not found')

@app.delete('/stock/{stock_id}', status_code= status.HTTP_204_NO_CONTENT)
async def delete_stock(stock_id:int):
    for i, stock in enumerate(Lista): #procura item da lista e apaga ele com .pop
        if stock.id == stock_id:
            Lista.pop(i)
            return
    raise HTTPException(status_code=404, detail= 'Stock not found')
```

Dificuldades

Acompanhei o vídeo seguindo o passo a passo do autor, buscando entender a necessidade do uso de bibliotecas para validação de dados. Em um certo momento ele afirmou que era muito útil para casos de uso como com o FastAPI. A partir daí, olhei o site oficial da FastAPI, da Pydantic e depois do artigo no Medium, não foi tão difícil usar a biblioteca Pydantic, difícil foi unir com a FastAPI e desenvolver algo simples e funcional. Estou começando no mundo da programação e me habituando às necessidades, mas no momento ainda tenho dificuldade com o “ato de programar”.

Conclusões

No curso da DeepLearning.AI a aula Overview dá uma ótima perspectiva a respeito das possibilidades que os agentes representam, além de contextualizar o estado da arte dos agentes em relação aos antigos usos em aplicações.

Na aula do ArjanCodes tivemos uma ótima introdução a respeito da biblioteca Pydantic e do tratamento de dados. A respeito disso, um dos pontos que mais me interessou foi, o tratamento de dados é necessário, porque protege sistemas contra diversos tipos de falhas, ou seja, não é apenas uma questão de organização, é questão de segurança.

Referências

https://learn.deeplearning.ai/courses/multi-ai-agent-systems-with-crewai/lesson/ddys8/overview

https://docs.pydantic.dev/latest/#pydantic-examples

https://www.youtube.com/watch?v=502XOB0u8OY&t=46s

https://fastapi.tiangolo.com/

https://medium.com/codenx/fastapi-pydantic-d809e046007f