Relatório 03 - Prática: Validação de dados com Pydantic (I)

Lucas Augusto Nunes de Barros

Descrição da atividade

O card traz um curso de sistemas multiagentes desenvolvido pela DeepLearning.AI em parceria com a crewAI. O objetivo central é demonstrar como orquestrar a colaboração entre agentes para resolver problemas complexos, desde a pesquisa técnica até a automação de processos de negócios.

Outro conteúdo presente no card é um vídeo sobre sobre a biblioteca Pydantic, essencial para validação e serialização de dados. O autor introduz a ferramenta como uma solução para os problemas comuns causados pela tipagem dinâmica do Python, permitindo que desenvolvedores tenham mais controle e facilidade ao lidar com dados de fontes externas, como APIs e entradas de usuários.

1. Curso Multiagentes da DeepLearning.AI e crewAI

O curso sobre sistemas multiagente aborda o uso da inteligência artificial, focada em lidar com incertezas e lógica complexa, permitindo a criação de ecossistemas onde agentes cooperam em tempo real, como demonstrado no fluxo automatizado de qualificação de vendas que os vídeos iniciais do curso abordam. Essa integração corrobora com o ponto citado por Andrew Ng sobre a capacidade da IA de não apenas processar informações, mas agir de forma integrada para impulsionar vendas e gerar valor de negócio através de aplicações inteligentes.

A apresentação sobre o CrewAI, plataforma para criar equipes multiagente com LLMs e nuvem, deixa claro como a orquestração de múltiplos agentes supera a arquitetura de um único agente. Ao dividir tarefas complexas em partes menores, os LLMs ganham eficiência e precisão. O framework permite configurar agentes com diferentes personas, cada um tendo suas funções, objetivos e tarefas específicas.

A estrutura básica para implementações utilizando a plataforma se baseia no seguintes componentes:

- 1. Agents: São os membros que compõem a equipe (crew), devem ter papéis, ferramentas, objetivos e histórias bem definidas

- 2. Tasks: São as tarefas que cada agente deve cumprir, devem ser configuradas de acordo com o objetivo, formando um roteiro bem escrito para atingi-lo.

- 3. Crews: São as equipes de agentes. Onde ocorre a supervisão e manutenção do fluxo de trabalho entre os agentes membros. Os agentes devem operar de forma orquestrada dentro dos times para alcançar o objetivo final.

- 4. Processes: É quem define e o gerencia o fluxo de trabalho dentro das equipes. Os processos determinam as tarefas de cada agente e como será a colaboração entre eles.

2. Pydantic

2.1 Tutorial

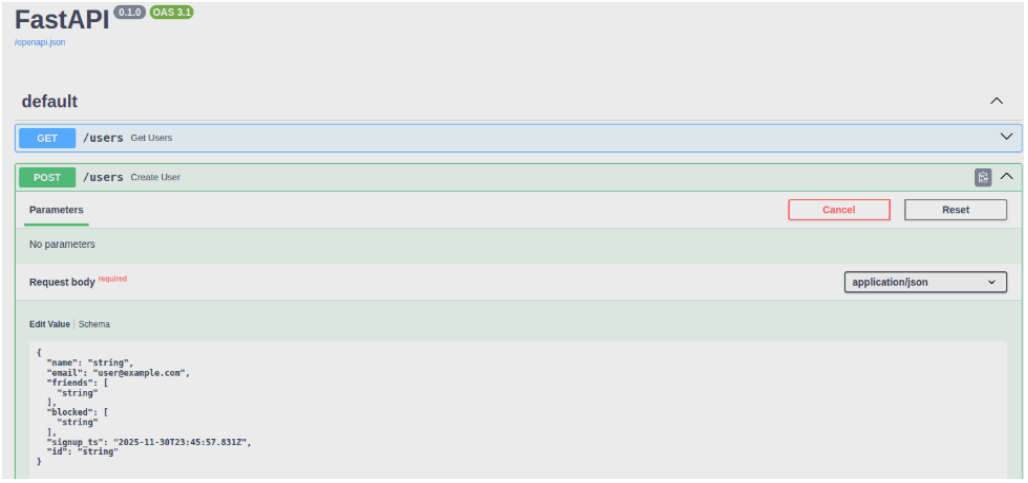
O autor inicia demonstrando algumas vantagens do Pydantic em relação às classes de dados nativas do Python quando é necessário fazer uma validação mais robusta. Ele ilustra o uso prático das ferramentas da biblioteca através da criação de modelos, utilizando tipos específicos como EmailStr para e-mails e SecretStr para senhas. Ele aprofunda os exemplos com a criação de validadores personalizados (field_validator e model_validator) para criar regras de negócio mais complexas e explica o processo de serialização, mostrando como converter objetos para JSON e excluir campos sensíveis. Além disso, o autor explica como o Pydantic é importante no ecossistema web, exemplificando sua integração nativa e eficiente com o framework FastAPI.

É possível concluir que a biblioteca Pydantic é uma ferramenta muito importante para um desenvolvedor de APIs e de sistemas de processamento de dados. Oferecendo controle sobre a estrutura e os tipos de dados do Python, a biblioteca aumenta significativamente a segurança e a robustez do software, facilitando a manutenção e a prevenção de falhas de segurança.

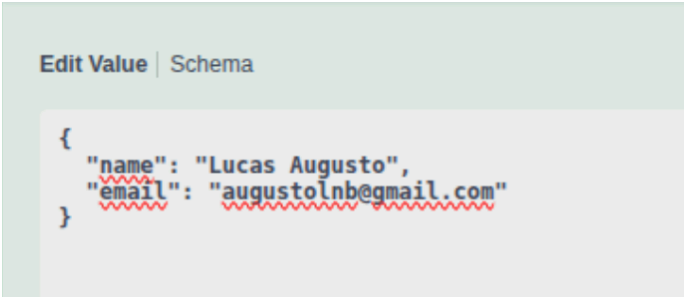
2.2 Prática

Para testar as funcionalidade da biblioteca, desenvolvi uma pequena aplicação usando FastAPI e tentei fazer a inserção de dados manualmente. Como as estruturas da API foram declaradas e validadas usando o Pydantic, sempre que as informações estavam no formato correto, a inserção dos dados no “banco de dados” simulado (__user__) ocorria normalmente. Já nos casos de dados incorretos, não declarados e outras situações que poderiam ser problemáticas para o administrador da API, eles eram tratados pela biblioteca, gerando erros autoexplicativos e evitando a inserção desses dados incoerentes.

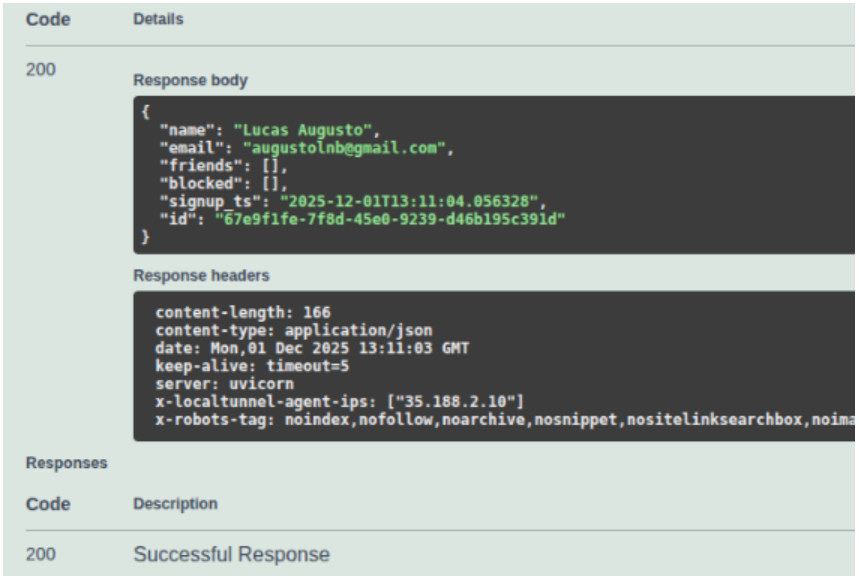
Interface de teste



Adicionando novos dados



Código de retorno 200 indica sucesso na recepção da requisição

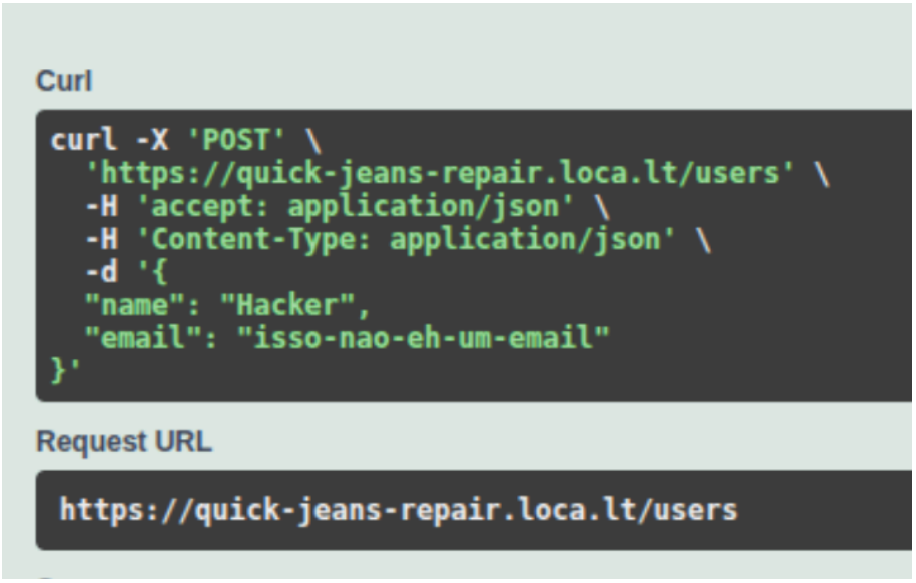


Com a inserção correta a API gera automaticamente um link para os dados adicionados no “banco de dados” simulado (__user__), ao acessar o link a API retorna a seguinte informação:

Dados recebidos com sucesso pela API

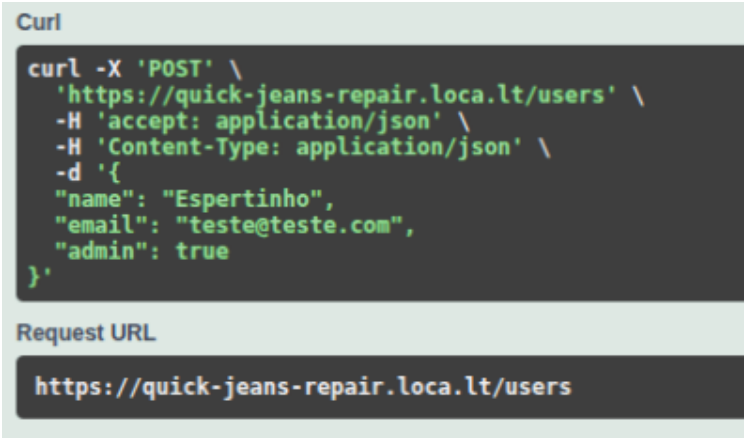
O que comprova a correta inserção dos dados, uma vez que possuem a estrutura necessária e a formatação correta dos dados. Já ao tentar adicionar informações inválidas, como emails incorretos, ou ainda enviar à API dados não declarados na estrutura da API, a biblioteca automaticamente gera um erro informando quais problemas encontrados na solicitação.

API recebendo email com formato inválido



O Pydantic identifica a incoerência e gera uma mensagem de erro

API recebendo dados não declarados na estrutura



Pydantic identifica a inserção de campos extra e gera um erro

Server response

Code	Details
422	Error: Unprocessable Entity

Response body

```
{
  "detail": [
    {
      "type": "extra_forbidden",
      "loc": [
        "body",
        "admin"
      ],
      "msg": "Extra inputs are not permitted",
      "input": true
    }
  ]
}
```

Dificuldades

Durante a implementação dos exemplos com o Pydantic houve uma pequena divergência entre a biblioteca usada no vídeo (provavelmente V2) e a nova versão disponível (V3). O parâmetro *max_items* foi removido e substituído pelo parâmetro *max_lenght*.

Erro retornado pelo terminal

```
⚠ atualização pydantic

/tmp/ipython-input-4221860055.py:8: PydanticDeprecatedSince20: max_items is deprecated and will be removed, use max_length instead. Deprecated in Pydantic V2.0 to be removed in V3.0. See Pydantic V2 Migration Guide at https://errors.pydantic.dev/2.12/migration/ friends: list[UUID4] = Field(
```

Trecho de código atualizado

```python
__users__ = []
name: str = Field(..., description="Name of the user")
email: EmailStr = Field(..., description="Email address of the user")
friends: list[UUID4] = Field(
    default_factory=list, max_length=500, description="List of friends"
)
blocked: list[UUID4] = Field(
    default_factory=list, max_length=500, description="List of blocked users"
)
```

Conclusões

A arquitetura multiagente da plataforma crewAI permite superar as limitações de modelos isolados ao simular fluxos de trabalho humanos baseados em hierarquia, funções e objetivos. Em conjunto com a validação fornecida pela biblioteca Pydantic, esse fluxo operacional se torna mais seguro e viável para aplicações reais, diminuindo a chance de erros e alucinações através do tratamento robusto dos dados que adentram a aplicação. Combinando essas tecnologias é possível criar sistemas que não apenas raciocinam e colaboram autonomamente, de forma otimizada através do paradigma multiagentes, mas também torna os limites da aplicação (APIs e demais entradas de usuário) seguros e menos propensos a erros e ataques mal intencionados, criando um ambiente propício para unir o melhor dos agente inteligentes com a alta disponibilidade e segurança que as aplicações necessitam.

Referências

[1] LEARN.DEEPLEARNING.AI. **Multi AI Agent Systems with crewAI**

Disponível em: <https://learn.deeplearning.ai/courses/multi-ai-agent-systems-with-crewai/lesson/eusjw/create-agents-to-research-and-write-an-article>

Acesso em 30 de novembro de 2025

[2] YOUTUBE.COM. **Why You Should Use Pydantic in 2024 | Tutorial**

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

Acesso em 01 de dezembro de 2025

[3] DOCS.PYDANTIC.DEV. **Pydantic Documentation**

Disponível em: <https://docs.pydantic.dev/latest/>

Acesso em 01 de dezembro de 2025

[4] DOCS.PYDANTIC.DEV. **BaseModel - Pydantic Validation**

Disponível em: <https://docs.pydantic.dev/latest/api/base_model/>

Acesso em 01 de dezembro de 2025