Relatório 3 - Prática: Validação de dados com Pydantic (I)

Enzo Rodrigues Novais Dias

Descrição da atividade

O primeiro vídeo aborda sobre as diferentes maneiras que os multiagentes podem colaborar, citando que pode ser sequencialmente, hierarquicamente e assincronicamente

Figura 1: Maneiras de colaboração dos multiagentes - [1]

Como exemplo, ele mostra um anúncio de uma vaga de engenheiro fullstack e logo após, o perfil de Noah, um jovem que está interessado na vaga. Ele aborda que há muitas coisas que não fazem sentido nenhum com a vaga que ele quer, como machine learning e soluções de IA.

E com isso ele diz que os agentes de IA podem ajudar o Noah, modificando o texto e o adequando para a vaga, mostrando uma das diversas aplicações e qualidade dos multiagentes.

Já o segundo vídeo é uma aula, bastante explicativa, sobre Pydantic, uma biblioteca que permite fazer validação e serialização de dados. Uma ferramenta ideal para ajudar no tratamento de dados, sendo muito útil para trabalhar com fonte externas de dados, como APIs.

Ele faz uma comparação com o DataGlass, dizendo que possuem uma maneira semelhante de funcionamento, sendo a principal diferença o fato do Pydantic adicionar opções de validação e serialização muito poderosas do que os DataGlasses integrados.

Logo após, ele realiza uma série de exemplos demonstrando as funcionalidades e aplicações do Pydantic, no primeiro ele demonstra o uso da biblioteca para definir modelos de dados com validação e como utilizar enumerações para definir papel de usuário. O segundo exemplo amplia o primeiro com validações mais complexas, tendo validações de entrada antes de processamento, fornecendo feedback sobre problemas nos dados. O terceiro exemplo expande ainda mais, demonstrando o controle sobre como os modelos são validados e serializados, sendo essencial para APIs. E por fim, o quarto exemplo implementa uma API simples para gerenciamento de usuários usando FastAPI e Pydantic com casos de uso.

Todos esses exemplos foram cruciais para a minha aplicação prática, que irei discorrer sobre:

Após o fim do conteúdo de aulas, realizei uma implementação de um sistema simples de gerenciamento de biblioteca, que demonstra validação, serialização e relacionamento entre modelos. Criando um modelo para livros, autores, empréstimos e usuários da biblioteca, com validações personalizadas e relacionamentos entre eles

A primeira mudança foi no REGEX, sendo criado um para validar os números ISBN (International Standart Book Number), uma espécie de “RG” dos livros.

```
ISBN_REGEX = re.compile(
    r"^(97[89])?\d{1,5}[- ]?\d{1,7}[- ]?\d{1,7}[- ]?\d{1,7}[- ]?[0-9X]$"
)
```

Logo em seguida, foi criado o BookStatus, que define os possíveis estados de um livro na biblioteca e o BookGenre, onde cada valor representa um gênero de livro

```
class BookStatus(IntEnum):
    AVAILABLE = 1
    BORROWED = 2
    MAINTENANCE = 3
    LOST = 4

class BookGenre(IntEnum):
    FICTION = 1
    NON_FICTION = 2
    SCIENCE = 3
    TECHNOLOGY = 4
    HISTORY = 5
    BIOGRAPHY = 6
    FANTASY = 7
    SCIENCE_FICTION = 8
    MYSTERY = 9
    THRILLER = 10
    ROMANCE = 11
    HORROR = 12
    OTHER = 99
```

Depois veio a criação das classes Autor, Book, LibraryUser e BookLoan. Não printarei todos aqui para não ficar muito desgastante com tantas imagens, porém todas as classes possuem seus devidos atributos, como Autor tendo alguns como nome, data de nascimento e biografia, enquanto Book possui título, ISBN, autores, gêneros e páginas. E claro, todas essas classes com seus validadores e serializadores, deixarei como exemplo um validador da classe BookLoan (empréstimo de livro) que foi bem trabalhoso, visto que verificava muitas coisas para definir se o empréstimo era válido, como se o livro e o usuário existem, se o livro está disponível e se o usuário está ativo e também se as datas são válidas

```python
    @model_validator(mode="after")
    def validate_loan(self) -> "BookLoan":
        if self.book_id not in self._books_map:
            raise ValueError(f"Book with ID {self.book_id} does not exist")
        if self.user_id not in self._users_map:
            raise ValueError(f"User with ID {self.user_id} does not exist")

        book = self._books_map[self.book_id]
        if book.status != BookStatus.AVAILABLE and self.return_date is None:
            raise ValueError(f"Book is not available for loan (current status: {book.status.name})")

        user = self._users_map[self.user_id]
        if not user.is_active:
            raise ValueError(f"User {user.name} is not active")
        if user.membership_valid_until < datetime.now():
            raise ValueError(f"User {user.name}'s membership has expired")

        if self.due_date < self.loan_date:
            raise ValueError("Due date cannot be earlier than loan date")
        if self.return_date and self.return_date < self.loan_date:
            raise ValueError("Return date cannot be earlier than loan date")

        if self.return_date is None and datetime.now() > self.due_date:
            self.is_overdue = True

        return self
```

Após isso, fui para os testes, criei dois autores com seus respectivos livros e características

```python
def test_library_system():
  #criação de dois autores
    author1 = Author(
        name="Rick Riordan",
        birth_year=1964,
        nationality="North-American",
        biography="American author and schoolteacher"
    )
    author2 = Author(
        name="J.K. Rowling",
        birth_year=1965,
        nationality="British",
        website="https://www.jkrowling.com"
    )
```

```python
    book1 = Book(
        title="The Lightning Thief",
        isbn="978-0786838653",
        authors=[author1.id],
        publication_year=2005,
        description="The Lightning Thief follows the story
        genres=[BookGenre.SCIENCE_FICTION], #"mudei" o ge
        pages=384
    )

    book2 = Book(
        title="Harry Potter and the Philosopher's Stone",
        isbn="978-0747532699",
        authors=[author2.id],
        publication_year=1997,
        genres=[BookGenre.FANTASY, BookGenre.FICTION],
        pages=223
    )
```

E também dois usuários, sendo que o segundo com associação expirada

```python
  #criação de dois usuários
    user1 = LibraryUser(
        name="Enzo Rodrigues",
        email="enzo.rodrigues@example.com",
        membership_number="LIB-123456",
        joined_date=datetime.now() - timedelta(days=30)
    )

    user2 = LibraryUser(
        name="Thiago Naves",
        email="thiago.naves@example.com",
        membership_number="LIB-654321",
        joined_date=datetime.now() - timedelta(days=60),
        membership_valid_until=datetime.now() - timedelta(days=10)
    )
```

Com isso, na hora dos empréstimos, o do segundo usuário deveria falhar, visto que sua associação estava expirada.

Também coloquei para tentar ser emprestado um livro que já estava “emprestado”, a devolução do livro e o empréstimo dele novamente, porém agora disponível. E por fim tentei a criação de um livro com um ISBN incorreto.

```python
#tentando emprestar um livro que já está emprestado
try:
    loan3 = BookLoan(
        book_id=book1.id,
        user_id=user1.id
    )
    print("ERROR: Loan for an already borrowed book was created!")
except ValueError as e:
    print(f"Correct error raised: {e}")

#devpolver livro
loan1.complete_return()
print(f"Book '{book1.title}' returned successfully. Status: {book1.status.name}")

#emprestando o livro novamente
book1.status = BookStatus.AVAILABLE  #funciona como um tipo de reset de status
loan4 = BookLoan(
    book_id=book1.id,
    user_id=user1.id,
    due_date=datetime.now() + timedelta(days=7)  #prazo de empréstimo "personalizado"
)
print(f"Book '{book1.title}' borrowed again until {loan4.due_date}")
```

Com isso, irei explicar o output em partes:

- O sistema impediu um empréstimo para o Thiago Naves, que estava com a assinatura expirada

```
Correct error raised: 1 validation error for BookLoan
  Value error, User Thiago Naves's membership has expired [type=value_error, input_value={'book_id': UUID('6f65a21...26, 11,
57, 41, 326062)}, input_type=dict]
```

- Ocorreu outro erro de validação, pois tentaram empréstimo de um livro que já estava emprestado

```
Correct error raised: 1 validation error for BookLoan
  Value error, Book is not available for loan (current status: BORROWED) [type=value_error, input_value={'book_id': UUID('cea
704b...26, 11, 57, 41, 326234)}, input_type=dict]
```

- Aqui houve a devolução do livro, mudando o seu status para Disponível e logo em seguida o seu novo empréstimo, com uma nova data de devolução

```
Book 'The Lightning Thief' returned successfully. Status: AVAILABLE
Book 'The Lightning Thief' borrowed again until 2025-03-19 11:57:41.326409
```

- Mostra a serialização correta dos livros para JSON, incluindo a conversão das UUIDs para strings

```
Books JSON Serialization:
[
  {
    "id": "cea704b4-9e6a-4254-8814-e94b7b8b864a",
    "title": "The Lightning Thief",
    "isbn": "9780786838653",
    "authors": [
      "f4d2fca4-49ac-44f6-8fb5-05471ee860f3"
    ],
    "publication_year": 2005,
    "genres": [
      8
    ],
    "description": "The Lightning Thief follows the story of young Percy Jackson, a troubled 12-year-old boy with a secret un
known even to himself",
    "pages": 384,
    "status": 1,
    "added_at": "2025-03-12T11:57:41.324935"
  },
  {
    "id": "6f65a218-29f6-4227-848e-1bab6f68b871",
    "title": "Harry Potter and the Philosopher's Stone",
    "isbn": "9780747532699",
    "authors": [
      "c077a6bf-4c56-4215-b3a9-b919259fd286"
    ],
    "publication_year": 1997,
    "genres": [
      7,
      1
    ],
    "description": null,
    "pages": 223,
    "status": 1,
    "added_at": "2025-03-12T11:57:41.324972"
  }
```


- O sistema corretamente rejeitou a criação de um livro com ISBN incorreto

```
]
ERROR: Book with invalid ISBN was created!
```

Dificuldades

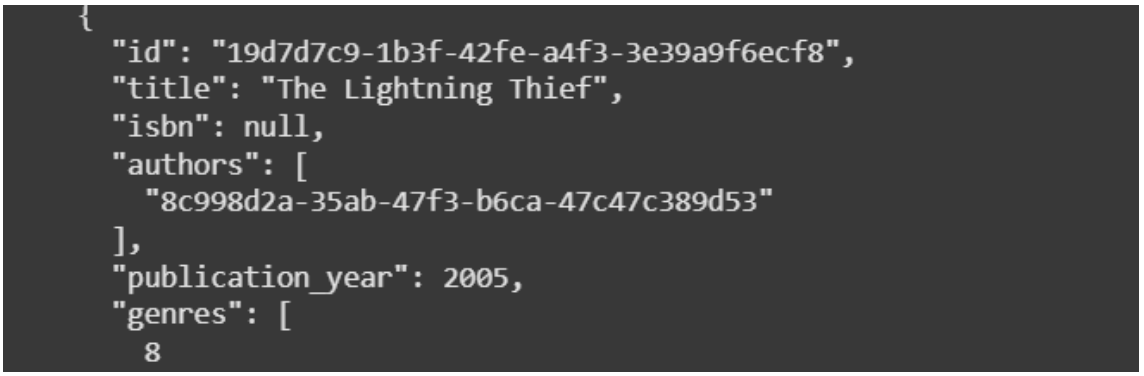
A parte prática foi desafiadora e tive alguns vários erros ao decorrer da atividade

Inicialmente, a função main estava com um erro de formato inválido de ISBN e fiquei em dúvida pois o formato estava correto, e quando eu testava o ISBN sem o “-”, funcionava, foi ai que entendi que provavelmente o erro estava no meu REGEX, e estava mesmo.

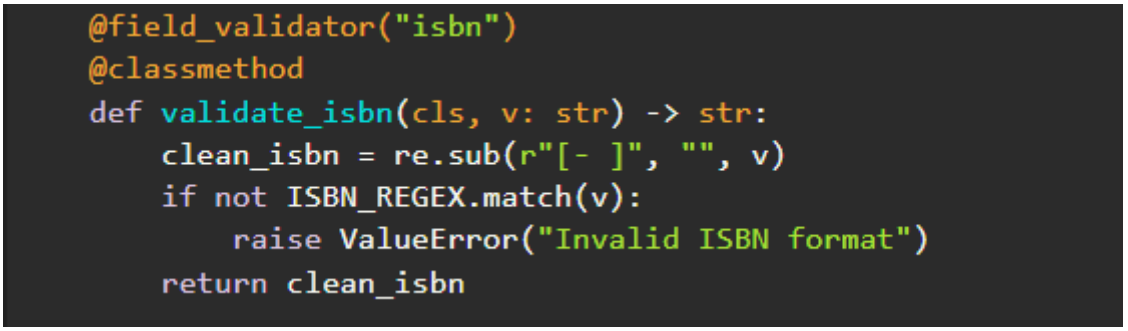
```
1 if __name__ == "__main__":
2     test_library_system()
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
<ipython-input-192-fdf7adb8ba15> in <cell line: 0>()
      1 if __name__ == "__main__":
----> 2     test_library_system()

                                 1 frames
/usr/local/lib/python3.11/dist-packages/pydantic/main.py in __init__(self, **data)
    212         # `__tracebackhide__` tells pytest and some other tools to omit this function from tracebacks
    213         __tracebackhide__ = True
--> 214         validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
    215         if self is not validated_self:
    216             warnings.warn(

ValidationError: 1 validation error for Book
isbn
  Value error, Invalid ISBN format [type=value_error, input_value='978-0786838653', input_type=str]
    For further information visit https://errors.pydantic.dev/2.10/v/value_error
```

Mas depois, no output, o ISBN aparecia como “null”, o que estava incorreto pois ele estava definido como um campo obrigatório na classe Book.



E depois de muito tempo e quebra de cabeça, a correção estava numa simples adição de “clean_isbn” no return



Conclusões

Eu aprendi o quão valioso é a validação dos dados e o quão útil é a serialização, visto que dados é algo que deve ser tratado com bastante delicadeza, pois qualquer dado fora da formatação comum pode comprometer muitas coisas.

Referências

1 - Multi AI Agent Systems with crewAI - DeepLearning.AI

2 - Why You Should Use Pydantic in 2024 | Tutorial