|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное автономное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 2

по дисциплине «Программирование на Python»

**Тема: «Валидация данных с Pydantic»**

Выполнил: Жукова Мария Андреевна,

студент группы ИУ8-13М

Проверил: Зотов Михаил Владиславович.

г. Москва, 2025 г.

1. Цель работы

Реализовать систему валидации входных данных в формате YAML(data.yaml) с использованием Pydantic.

2. Постановка задачи

1. Реализовать классы для интернет-магазина на основе задания;
2. Инициализировать экземпляры классы на основе YAML config;
3. Вывести экземпляры классов через logger.info.

3. Ход работы

Основной используемый в данной практической работе инструмент – библиотека pydantic. Для определения значений атрибутов классов была использована библиотека typing. Также, для логирования использовался модуль logging.

1. ***Разбор задачи***

В приведенном yaml файле описывается структура заказа в интернет-магазине. В одном заказе присутствуют основные сущности, такие как пользователь (и соответствующая информация о нем), товар в заказе (или товары) и, собственно, сам заказ.

Описание предоставленных для заполнения классов:

1. Классы Пользователя:
   1. UserSpec – основная информация по пользователю;
   2. ProfileSpec (наследует UserSpec) – профиль пользователя;
2. Классы Товара:
   1. ItemSpec – основная информация по товару;
   2. ServiceSpec - основная информация по товару, идентичен предыдущему классу;
3. Классы заказа:
   1. OrderLineSpec – Описание товара именно в заказе, содержит класс товара в атрибуте;
   2. OrderSpec – описание 1 заказа, содержит класс профиля пользователя и класс описания товара в заказе;
   3. OrdersSpec – описание всех заказов, содержит описание каждого заказа в отдельности.
4. ***Шаги решения задачи***
5. Инициализировать атрибуты для каждого класса в соответствии с их заданным типом данных с использованием pydantic и typing. Реализовать заданные ограничения на атрибуты;
6. Прописать конструкции для валидации значений полей атрибутов. Первыми прописываются конструкции, которые проверяют только локальные значения внутри одного класса. Далее, прописываются конструкции, затрагивающие валидацию нескольких классов;
7. Добавление логирования вывода и тестирование кода на конструкции в файле с расширением yaml.
8. ***Реализация шагов из хода решения***
9. Инициализация атрибутов и ограничений на них

Для инициализации большинства атрибутов использовались встроенные классы языка Python, такие как *int*, *str* и *float*. Исключения:

* Класс *UserSpec*, атрибут *second\_name* - опциональное назначение, для его инициализации использовался метод Optional[str] = None из библиотеки typing;
* Класс *UserSpec*, атрибут *email* – должен содержать @ и “.”, для его инициализации использовался метод EmailStr из библиотеки pydantic;
* Класс *ProfileSpec*, атрибут *url* – должен содержать “://”, для его инициализации использовался метод HttpUrl из библиотеки pydantic;
* Класс *OrderLineSpec*, атрибут *item\_line* – может содержать объект класса ItemSpec или объект класса ServiceSpec, для его инициализации использовался метод Union из библиотеки typing;
* Класс *OrderSpec*, атрибутb *user\_info* – содержит описание 1 конкретного заказа, для его инициализации использовался класс ProfileSpec;
* Класс *OrderSpec*, атрибут *items\_line* – должен содержать список объектов OrderLineSpec;

Для всех классов устанавливается ограничение на добавление дополнительных атрибутов. Данное ограничение реализуется через задание настроек модели с помощью атрибута model\_config с использованием метода ConfigDict из pydantic:

*model\_config = ConfigDict(extra="forbid")*

1. Конструкции валидации значений атрибутов

Для валидации использовались декораторы *field\_validator* – для валидации значений конкретных полей и *model\_validator* – для валидации выполнения условий значениями нескольких полей. В обеих конструкциях использовались режимы after - валидатор вызывается после стандартной валидации и приведения типов.

В случае, если значение атрибута не прошло валидацию, в поток ошибок выводится соответствующая ошибка *ValueError* с поясняющим комментарием.

При задании атрибутов классов использовалось несколько типов валидации ввода:

1. Строковая валидация:

* Класс *UserSpec*, атрибут *username* - проверка на допустимый алфавит;
* Класс *UserSpec*, атрибут *surname* - проверка на допустимый алфавит;
* Класс *UserSpec*, атрибут *status* - проверка на одно из двух допустимых значений;
* Класс *ProfileSpec*, атрибут *bio* – проверка на допустимый алфавит;
* Класс *ItemSpec*, атрибут *name* - проверка на допустимый алфавит;
* Класс *ItemSpec*, атрибут *desc* - проверка на допустимый алфавит;
* Класс *ServiceSpec*, атрибут *name* - проверка на допустимый алфавит;
* Класс *ServiceSpec*, атрибут *desc* - проверка на допустимый алфавит;

Проверка на допустимый алфавит реализована с помощью фильтрации строки по регулярному выражению с использованием модуля re:

*re.match(r'^[А-Яа-яЁё\s\-]+$', value)*

1. Числовая валидация:

* Класс *ItemSpec*, атрибут *price* - проверка на положительное значение вещественного числа;
* Класс *OrderLineSpec*, атрибут *quantity* - проверка на положительное значение вещественного числа;
* Класс *OrderLineSpec*, атрибут *line\_price* - проверка на положительное значение вещественного числа;
* Класс *OrderLineSpec*, атрибут *line\_price* - проверка на соответствие числа ожидаемому значению;
* Класс *OrderSpec*, атрибут *order\_line\_id* - проверка на уникальность целочисленного значения;
* Класс OrdersSpec:
  1. Класс *UserSpec*, атрибут *user\_id* - проверка на уникальность целочисленного значения;
  2. Класс *ItemSpec*, атрибут *item\_id* - проверка на уникальность целочисленного значения;
  3. Класс *ServiceSpec*, атрибут *item\_id* - проверка на уникальность целочисленного значения;
  4. Класс *ServiceSpec*, атрибут *price* - проверка на положительное значение вещественного числа;
  5. Класс *OrderLineSpec*, атрибут *order\_id* - проверка на уникальность целочисленного значения;
  6. Класс *OrderLineSpec*, атрибут *order\_line\_id* - проверка на уникальность целочисленного значения;

Важно отметить, что валидация (в данном случае, конкретно проверка на уникальность) таких значений, как *user\_id*, *item\_id*, *service\_id*, *order\_id* невозможна внутри соответствующих классов. Поэтому, валидация таких параметров вынесена в класс *OrdersSpec*, который объединяет все заданные классы и может сравнивать атрибуты разных объектов одних классов.

1. Логирование и тестирование

Тестирование модели из классов осуществляется с помощью конструкции в yaml-файле:

*market\_place\_orders:*

*- order\_id: 123321*

*user\_info:*

*user\_id: 1000*

*username: Тест*

*surname: Петров*

*second\_name: null*

*email: test@mail.ru*

*status: active*

*bio: студент*

*url: https://folder.ru*

*items\_line:*

*- order\_id: 123321*

*order\_line\_id: 1*

*item\_line:*

*service\_id: 101*

*name: "Доставка"*

*desc: "Клиентская доставка"*

*price: 359*

*quantity: 1*

*line\_price: 359*

*- order\_id: 123321*

*order\_line\_id: 2*

*item\_line:*

*service\_id: 9999*

*name: "Лампа"*

*desc: "Настольная лампа"*

*price: 990*

*quantity: 2*

*line\_price: 1980*

В данной конструкции представлен 1 заказ на 2 товара, сделанный 1 пользователем.

Конструкция из yaml-файла считывается при помощи конструкции безопасной работы с файлами из python:

*with open("./data.yaml", "r", encoding="utf-8") as f:*

*data = yaml.safe\_load(f)*

Данная конструкция открывает и закрывает его. Перед закрытием, содержимое файла необходимо записать в python-объект. Для этого используется библиотека yaml и метод safe\_load, который не исполняет произвольный код внутри yaml-файла. Соответственно, yaml преобразуется в стандартные Python-структуры. В данном конкретном случае содержимое yaml файла десериализуется в словарь.

После закрытия файла создается объект класса OrdersSpec путем распаковки считанного словаря из yaml в аргументы. Следовательно, pydantic проверяет что:

* Ключи соответствуют полям модели;
* Для каждого заказа создаёт объект *OrderSpec*;
* Внутри *OrderSpec* проверяет *user\_info* → превращает его в *ProfileSpec*;
* Для *items*\_*line* создаёт список объектов *OrderLineSpec*;
* В каждом *OrderLineSpec*.*item\_line* пытается распознать, это *ItemSpec* или *ServiceSpec*;
* Применяет все валидаторы (@*field\_validator*, @*model*\_*validator*);
* Возвращает готовый объект *OrdersSpec* (при условии, что все проверки пройдены).

Логирование операций осуществляется с использованием модуля logger. Логгер инициализируется в заголовке файла, после подключения библиотек и зависимостей. При выполнении данного задания логгер настраивался на уровень логирования INFO:

*logging.basicConfig(level=logging.INFO);*

*logger = logging.getLogger(\_\_name\_\_);*

Далее, логгер активируется в функции main по экземпляру класса OrdersSpec, в который считываются данные из yaml-файла и сериализует модель в json-строку:

*logger.info(orders.model\_dump\_json(indent=2));*

Результат логирования записи корректной конструкции из yaml в python-объект:

INFO:\_\_main\_\_:{

"market\_place\_orders": [

{

"order\_id": 123321,

"user\_info": {

"user\_id": 1000,

"username": "Тест",

"surname": "Петров",

"second\_name": null,

"email": "test@mail.ru",

"status": "active",

"bio": "студент",

"url": "https://folder.ru/"

},

"items\_line": [

{

"order\_id": 123321,

"order\_line\_id": 1,

"item\_line": {

"service\_id": 101,

"name": "Доставка",

"desc": "Клиентская доставка",

"price": 359.0

},

"quantity": 1.0,

"line\_price": 359.0

},

{

"order\_id": 123321,

"order\_line\_id": 2,

"item\_line": {

"service\_id": 9999,

"name": "Лампа",

"desc": "Настольная лампа",

"price": 990.0

},

"quantity": 2.0,

"line\_price": 1980.0

}]}]}

При изменении одного параметра, например, при дублировании order\_line\_id во втором товаре в заказе, получаем следующую ошибку:

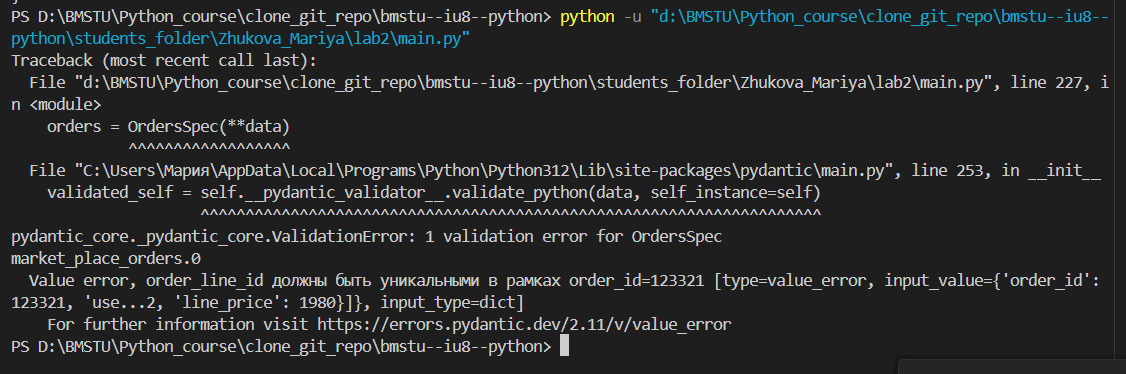


Рисунок 1 – ошибка при валидации не уникальных значений параметра order\_line\_id

4. Заключение

В ходе работы были изучены библиотеки pydantic, typing и logging. Также, применялись такие методы как считывание данных из yaml и проверка на соответствие регулярным выражениям. Была реализована программа, позволяющая проверить на корректность и считать конструкцию из yaml-файла в python-объект и посмотреть на лог в json.

5. ПРИЛОЖЕНИЕ

Листинг 1 – Программа, позволяющая проверить на корректность и считать конструкцию из yaml-файла в python-объект и посмотреть на лог в json

|  |
| --- |
| from pydantic import BaseModel, field\_validator, \      EmailStr, HttpUrl, model\_validator, ConfigDict  from typing import Optional, Union, List  import re  import logging  import yaml  # Логгер  logging.basicConfig(level=logging.INFO)  logger = logging.getLogger(\_\_name\_\_)  class UserSpec(BaseModel):      user\_id: int      username: str      surname: str      second\_name: Optional[str] = None      email: EmailStr      status: str      model\_config = ConfigDict(extra="forbid")  # Запрещаем лишние поля      # Проверка поля username на содержание только русского языка      @field\_validator("username", mode="after")      @classmethod      def check\_russian\_in\_name(cls, value: str) -> str:          if not re.match(r'^[А-Яа-яЁё\s\-]+$', value):              raise ValueError("Поле username должно \                              содержать только русский язык и быть не пустым")          return value      # Проверка поля surname на содержание только русского языка      @field\_validator("surname", mode="after")      @classmethod      def check\_russian\_in\_surname(cls, value: str) -> str:          if not re.match(r'^[А-Яа-яЁё\s\-]+$', value):              raise ValueError("Поле surname должно содержать \                              только русский язык и быть не пустым")          return value      # Проверка поля статус на допустимые значения      @field\_validator("status", mode="after")      @classmethod      def check\_status(cls, value: str) -> str:          if value not in ["active", "non-active"]:              raise ValueError("Поле status должно быть \                              'active' или 'non-active'")          return value  class ProfileSpec(UserSpec):      bio: str      url: HttpUrl  # url: Текстовое поле, не пустой, однозначно должны быть ://      model\_config = ConfigDict(extra="forbid")  # Запрещаем лишние поля      # Проверка поля bio на содержание только русского языка      @field\_validator("bio", mode="after")      @classmethod      def check\_russian\_in\_bio(cls, value: str) -> str:          if not re.match(r'^[А-Яа-яЁё\s\-]+$', value):              raise ValueError("Поле username должно содержать \                              только русский язык и быть не пустым")          return value  class ItemSpec(BaseModel):      item\_id: int      name: str   # Русский алфавит      desc: str   # Русский алфавит      price: float  # >0      model\_config = ConfigDict(extra="forbid")  # Запрещаем лишние поля      # Проверка поля name на содержание только русского языка      @field\_validator("name", mode="after")      @classmethod      def check\_russian\_in\_name(cls, value: str) -> str:          if not re.match(r'^[А-Яа-яЁё\s\-]+$', value):              raise ValueError("Поле name должно содержать \                              только русский язык и быть не пустым")          return value      # Проверка поля desc на содержание только русского языка      @field\_validator("desc", mode="after")      @classmethod      def check\_russian\_in\_desc(cls, value: str) -> str:          if not re.match(r'^[А-Яа-яЁё\s\-]+$', value):              raise ValueError("Поле desc должно содержать \                              только русский язык и быть не пустым")          return value      # Проверка поля price на допустимые значения      @field\_validator("price", mode="after")      @classmethod      def check\_price(cls, value: float) -> float:          if value <= 0:              raise ValueError("Значение в поле price \                              должно быть строго больше 0")          return value  class ServiceSpec(BaseModel):      service\_id: int      name: str  # Русский алфавит      desc: str  # Русский алфавит      price: float  # >0      model\_config = ConfigDict(extra="forbid")  # Запрещаем лишние поля      # Проверка поля name на содержание только русского языка      @field\_validator("name", mode="after")      @classmethod      def check\_russian\_in\_name(cls, value: str) -> str:          if not re.match(r'^[А-Яа-яЁё\s\-]+$', value):              raise ValueError("Поле name должно содержать \                              только русский язык и быть не пустым")          return value      # Проверка поля desc на содержание только русского языка      @field\_validator("desc", mode="after")      @classmethod      def check\_russian\_in\_desc(cls, value: str) -> str:          if not re.match(r'^[А-Яа-яЁё\s\-]+$', value):              raise ValueError("Поле desc должно содержать \                              только русский язык и быть не пустым")          return value      # Проверка поля price на допустимые значения      @field\_validator("price", mode="after")      @classmethod      def check\_price(cls, value: float) -> float:          if value <= 0:              raise ValueError("Значение в поле price \                              Должно быть строго больше 0")          return value  class OrderLineSpec(BaseModel):      order\_id: int      order\_line\_id: int  # Уникальное значение в рамках одного order\_id      item\_line: Union[ItemSpec, ServiceSpec]      quantity: float      line\_price: float      model\_config = ConfigDict(extra="forbid")      @field\_validator("quantity", mode="after")      @classmethod      def check\_quantity(cls, value: float) -> float:          if value <= 0:              raise ValueError("Значение в поле quantity \                              должно быть строго больше 0")          return value      @model\_validator(mode="after")      def check\_line\_price(self):          if self.line\_price <= 0:              raise ValueError("Значение в поле line\_price \                              должно быть строго больше 0")          expected\_price = self.item\_line.price \* self.quantity          if abs(expected\_price - self.line\_price) > 0:              raise ValueError(f"line\_price должно быть равно {expected\_price}")          return self  class OrderSpec(BaseModel):      order\_id: int      user\_info: ProfileSpec      items\_line: List[OrderLineSpec]      model\_config = ConfigDict(extra="forbid")  # Запрещаем лишние поля        @model\_validator(mode="after")      def check\_unique\_order\_lines(self):          order\_line\_ids = [line.order\_line\_id for line in self.items\_line]          if len(order\_line\_ids) != len(set(order\_line\_ids)):              raise ValueError(f"order\_line\_id должны быть уникальными в рамках order\_id={self.order\_id}")          return self  class OrdersSpec(BaseModel):      market\_place\_orders: List[OrderSpec]      model\_config = ConfigDict(extra="forbid")  # Запрещаем лишние поля      @model\_validator(mode="after")      def check\_global\_uniques(self):          user\_ids = []          item\_ids = []          service\_ids = []          order\_ids = []          for order in self.market\_place\_orders:              order\_ids.append(order.order\_id)              # user\_id (из ProfileSpec)              user\_ids.append(order.user\_info.user\_id)              # item\_id / service\_id              for line in order.items\_line:                  item\_line = line.item\_line                  if isinstance(item\_line, ItemSpec):                      item\_ids.append(item\_line.item\_id)                  elif isinstance(item\_line, ServiceSpec):                      service\_ids.append(item\_line.service\_id)          # Проверки          if len(user\_ids) != len(set(user\_ids)):              raise ValueError("user\_id должны быть уникальными")          if len(order\_ids) != len(set(order\_ids)):              raise ValueError("order\_id должны быть уникальными")          if len(item\_ids) != len(set(item\_ids)):              raise ValueError("item\_id должны быть уникальными")          if len(service\_ids) != len(set(service\_ids)):              raise ValueError("service\_id должны быть уникальными")          return self  if \_\_name\_\_ == "\_\_main\_\_":      with open("D:/BMSTU/Python\_course/clone\_git\_repo/bmstu--iu8--python/students\_folder/Zhukova\_Mariya/lab2/data.yaml", "r", encoding="utf-8") as f:          data = yaml.safe\_load(f)      orders = OrdersSpec(\*\*data)      logger.info(orders.model\_dump\_json(indent=2)) |