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 языке Python»

Тема: «Валидация данных с Pydantic»

Выполнил: Никулина Злата Евгеньевна,
студент группы ИУ8-13М

Проверил: Зотов Михаил Владиславович.

г. Москва, 2025 г.

1. ЦЕЛЬ РАБОТЫ

Цель лабораторной работы: реализация системы валидации входных данных в формате YAML (data.yaml) с использованием Pydantic.

2. ПОСТАНОВКА ЗАДАЧИ

Необходимо реализовать классы для интернет-магазина. Обеспечить строгую проверку данных (корректность типов данных, проверку формата строк, контроль уникальности идентификаторов, запрет лишних атрибутов). Организовать загрузку входных данных из файла в формате YAML (data.yaml) с использованием Pydantic создать экземпляры классов, затем вывести экземпляры классов через logger.info с отображением информации о пользователях, заказах, позициях в заказах.

Результат работы должен представлять модель для организации данных интернет-магазина, которая будет выявлять ошибки на начальном этапе.

3. ХОД РАБОТЫ

В данной лабораторной работе используется: язык программирования Python, библиотека Pydantic, библиотека PyYAML для загрузки данных в формате YAML, модуль logging для вывода сообщений.

Ход работы:

Шаг 1. Осознание задачи

Необходимо реализовать систему валидации данных интернет-магазина на основе YAML-файла с использованием Pydantic. Для этого описать классы, содержащие информацию о пользователях и заказах, создать модуль загрузки данных из файла в формате YAML (data.yaml), выполнить проверку и вывести результат.

Шаг 2. Описание действий

1. Были определены регулярные выражения и допустимые значения для полей,
2. Реализованы классы моделей (UserSpec, ProfileSpec, ItemSpec, ServiceSpec, OrderLineSpec, OrderSpec, OrdersSpec),
3. Для каждого класса были заданы ограничения: типы данных, обязательные поля, проверки уникальности и валидаторы,
4. По средствам библиотеки yaml был добавлен входной файл data.yaml,
5. Настроен модуль logging для вывода экземпляров классов.

Шаг 3. Реализация, результат

Была написана программа на языке Python. Листинг программы представлен в приложении А.

Для реализации необходимы классы для интернет-магазина:

- UserSpec – описание пользователя с проверкой корректности email и ограничением допустимых статусов (active, non-active);

- ProfileSpec – данные о профиле пользователя с добавлением биографии и ссылки (с проверкой на наличие :// в URL);
- ItemSpec и ServiceSpec – описание товара и услуги с проверкой на корректность строковых полей и условием, что цена должна быть больше нуля;
- OrderLineSpec – описание позиции заказа, включающее проверку, что итоговая стоимость (line_price) совпадает с произведением цены на количество;
- OrderSpec – описание заказа с валидацией уникальности идентификаторов строк внутри заказа и проверкой связи order_id;
- OrdersSpec – контейнер для списка заказов, проверяющий глобальную уникальность всех идентификаторов (заказов, пользователей, товаров, услуг).

Загрузка данных о заказах осуществлена при помощь функции load. Далее выполнена передача данных, их проверка.

В случае корректных данных создаются экземпляры моделей, которые выводят результат с помощью logger.info. Вывод программы в случае правильных сведений приведен на рисунке 1.

```
[INFO] order_id=145541
[INFO] id=1000, ФИО: Калинин Проба Петрович, email=prob@mail.ru, статус=active, био=студент, url=https://folder.ru
[INFO] строка заказа=1, order_id=145541, количество=1.0, цена=654.0
[INFO] service_id=101, наименование=Доставка, описание=Клиентская доставка, цена=654.0
[INFO] строка заказа=2, order_id=145541, количество=1.0, цена=758.0
[INFO] service_id=8888, наименование=Стол, описание=Деревянный стол, цена=758.0
[INFO] всего заказов = 1
```

Рисунок 1 – Вывод программы при корректных данных

При обнаружении некорректных данных формируется сообщение об этом, где указывается причина ошибки. Пример вывода программы с ошибкой в данных в email показан на рисунке 2.

```
Value error, email должен содержать @ и . [type=value_error, input_value='probmail.ru', input_type=str]
  For further information visit https://errors.pydantic.dev/2.11/v/value_error
```

Рисунок 2 – Вывод программы при некорректных данных

На рисунке 3 выведен результат при несоответствии итоговой стоимости произведению количества объектов на их цену.

```
Value error, line_price не равно quantity * item_line.price (654.0 != 2.0 * 654.0 = 1308.0)
```

Рисунок 3 – Вывод программы с неверной итоговой стоимостью

4. ЗАКЛЮЧЕНИЕ

В ходе работы были изучены возможности библиотеки Pydantic для валидации данных и принципы описания классов. В процессе лабораторной работы применялась библиотека PyYAML для загрузки данных из конфигурационного файла в формате YAML, а также модуль logging для вывода результатов.

Были реализованы наборы классов, описывающих структуру данных интернет-магазина, система проверки корректности входных данных, загрузка данных. Создана система, которая позволяет работать с входными данными, выявлять ошибки на начальном этапе загрузки и формировать корректный вывод.

5. ПРИЛОЖЕНИЕ А

Реализованный код располагается по следующей ссылке: https://github.com/zezOtik/bmstu--iu8--python/tree/Nikulina/students_folder/Nikulina/lab2

Листинг А.1 – Модель формирования заказов

```
from pydantic import BaseModel, Field, field_validator, model_validator
from typing import Optional, List, Union, Set
from typing_extensions import Literal
import logging
import yaml

RUS_RE = r'^[А-Яа-яЁё][А-Яа-яЁё \-]*$'
Status = Literal['active', 'non-active']
logging.basicConfig(level=logging.INFO, format='[%(levelname)s] %(message)s')
logger = logging.getLogger(__name__)

class UserSpec(BaseModel):
    user_id: int
    username: str = Field(..., pattern=RUS_RE)
    surname: str = Field(..., pattern=RUS_RE)
    second_name: Optional[str] = Field(None, pattern=RUS_RE)
    email: str
    status: Status

    class Config:
        extra = 'forbid'

    @field_validator('email')
    @classmethod
    def check_email(cls, v: str) -> str:
        if '@' not in v or '.' not in v:
            raise ValueError('email должен содержать @ и .')
        return v

class ProfileSpec(UserSpec):
    bio: str = Field(..., pattern=RUS_RE)
    url: str
```

```
    @field_validator('url')
    @classmethod
    def check_url(cls, v: str) -> str:
        if '://' not in v:
            raise ValueError("url должен содержать '://'")
        return v

class ItemSpec(BaseModel):
    item_id: int
    name: str = Field(..., pattern=RUS_RE)
    desc: str = Field(..., pattern=RUS_RE)
    price: float = Field(..., gt=0)

    class Config:
        extra = 'forbid'

class ServiceSpec(BaseModel):
    service_id: int
    name: str = Field(..., pattern=RUS_RE)
    desc: str = Field(..., pattern=RUS_RE)
    price: float = Field(..., gt=0)

    class Config:
        extra = 'forbid'

class OrderLineSpec(BaseModel):
    order_id: int
    order_line_id: int
    item_line: Union[ServiceSpec, ItemSpec]
    quantity: float = Field(..., gt=0)
    line_price: float = Field(..., gt=0)

    class Config:
        extra = 'forbid'

    @model_validator(mode='after')
    def check_line_price(self):
        expected = self.quantity * self.item_line.price
        if self.line_price != expected:
            raise ValueError(
                f'line_price не равно quantity * item_line.price '
```

```
                f'({self.line_price} != {self.quantity} * {self.item_line.price} =
{expected})'
            )
        return self

class OrderSpec(BaseModel):
    order_id: int
    user_info: ProfileSpec
    items_line: List[OrderLineSpec]

    class Config:
        extra = 'forbid'

    @model_validator(mode='after')
    def check_lines(self):
        if not self.items_line:
            raise ValueError('items_line пустой')

        seen: Set[int] = set()
        for ln in self.items_line:
            if ln.order_id != self.order_id:
                raise ValueError('order_line.order_id должен совпадать с
order.order_id')
            if ln.order_line_id in seen:
                raise ValueError(f'повтор order_line_id внутри order
{self.order_id}')
            seen.add(ln.order_line_id)
        return self

class OrdersSpec(BaseModel):
    market_place_orders: List[OrderSpec]

    class Config:
        extra = 'forbid'

    @model_validator(mode='after')
    def check_global_uniques(self):
        order_ids: Set[int] = set()
        user_ids: Set[int] = set()
        item_ids: Set[int] = set()
        service_ids: Set[int] = set()
```

```
        for o in self.market_place_orders:
            if o.order_id in order_ids:
                raise ValueError(f'повторяющийся order_id: {o.order_id}')
            order_ids.add(o.order_id)

            uid = o.user_info.user_id
            if uid in user_ids:
                raise ValueError(f'повторяющийся user_id: {uid}')
            user_ids.add(uid)

            for ln in o.items_line:
                il = ln.item_line
                if isinstance(il, ItemSpec):
                    if il.item_id in item_ids:
                        raise ValueError(f'повторяющийся item_id: {il.item_id}')
                    item_ids.add(il.item_id)
                else:
                    if il.service_id in service_ids:
                        raise ValueError(f'повторяющийся service_id:
{il.service_id}')
                    service_ids.add(il.service_id)
        return self

def load(yaml_text: str) -> OrdersSpec:
    data = yaml.safe_load(yaml_text)
    return OrdersSpec(**data)

if __name__ == '__main__':
    with open("data.yaml", "r", encoding="utf-8") as f:
        yaml_text = f.read()
    orders = load(yaml_text)

for o in orders.market_place_orders:
    logger.info(f'order_id={o.order_id}')

    u = o.user_info
    logger.info(
        f'id={u.user_id}, '
        f'ФИО: {u.surname} {u.username} {u.second_name}, '
        f'email={u.email}, статус={u.status}, био={u.bio}, url={u.url}'
```

```
    )

    for ln in o.items_line:
        logger.info(
            f'строка заказа={ln.order_line_id}, '
            f'order_id={ln.order_id}, количество={ln.quantity},
цена={ln.line_price}'
        )

        if isinstance(ln.item_line, ServiceSpec):
            s = ln.item_line
            logger.info(
                f'service_id={s.service_id}, '
                f'наименование={s.name}, описание={s.desc}, цена={s.price}'
            )
        else:
            it = ln.item_line
            logger.info(
                f'item_id={it.item_id}, '
                f'наименование={it.name}, описание={it.desc}, цена={it.price}'
            )

logger.info(f'всего заказов = {len(orders.market_place_orders)}')
```