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: «МОДЕЛИ ПАЙДАНТИКА»

Выполнил: Бородин Г.
студент группы ИУ8-13М

Проверил: Зотов М.

г. Москва, 2025 г.

1. ЦЕЛЬ РАБОТЫ

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

2. ПОСТАНОВКА ЗАДАЧИ

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

3. ХОД РАБОТЫ

ИСПОЛЬЗУЕТСЯ В ДАННОЙ РАБОТЕ: язык Python; библиотека Pydantic для описания моделей и валидации данных; библиотека PyYAML для чтения конфигурации из YAML; модуль logging для вывода результатов; модуль typing для задания типов (List, Union, Literal). Для проверки русскоязычных строк применено регулярное выражение RUS_RE, а для запрета лишних полей в YAML использована настройка extra="forbid".

ШАГ 1. Анализ требований.

Определены требования к структуре входных данных интернет-магазина: сущности пользователя (UserSpec, ProfileSpec), товара/услуги (ItemSpec, ServiceSpec) и заказов (OrderLineSpec, OrderSpec, OrdersSpec). Уточнены правила валидации: обязательность полей, допустимые значения (status), ограничения на типы, а также запрет дополнительных атрибутов.

ШАГ 2. Реализация моделей Pydantic.

Созданы модели на базе BaseModel. Для строковых полей, содержащих русские символы, задано ограничение pattern=RUS_RE. Для email использован тип EmailStr, для url использован тип HttpUrl. Для цен и количества применены числовые ограничения gt=0. Для всех моделей включён запрет лишних полей через ConfigDict(extra="forbid").

ШАГ 3. Реализация бизнес-валидации.

В модели OrderLineSpec добавлена проверка согласованности стоимости строки заказа: поле line_price должно быть равно произведению quantity * item_line.price. Проверка выполнена через @model_validator(mode='after'), при несоответствии генерируется исключение ValueError.

ШАГ 4. Загрузка YAML и создание экземпляров.

Реализована функция get_data_from_yaml(), которая читает YAML-файл через yaml.safe_load(), выполняет валидацию с помощью OrdersSpec.model_validate(), а затем выводит результат через logging.info() в формате JSON (model_dump_json(indent=2, ensure_ascii=False)).

РЕЗУЛЬТАТ: получена система, которая валидирует структуру и содержимое входного YAML, запрещает лишние поля, проверяет корректность email/url, а также контролирует вычисляемую стоимость строк заказа.

4. ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были изучены возможности библиотеки Pydantic для типизации и валидации данных. Реализован набор моделей предметной области интернет-магазина (пользователь, профиль, товар, услуга, строка заказа, заказ и список заказов) с ограничениями на типы и значения полей. Настроен запрет дополнительных атрибутов, реализована проверка корректности вычисляемого поля line_price, а также выполнена инициализация объектов из конфигурации YAML с последующим выводом структуры данных через журналирование. Полученная реализация позволяет автоматически обнаруживать ошибки структуры и содержания входных данных на этапе загрузки конфигурации.

5. ПРИЛОЖЕНИЕ

Реализованный код располагается по следующей ссылке: https://github.com/zezOtik/bmstu--iu8--python/pull/52#pullrequestreview-3444287497

Листинг 1 – Реализация Pydantic-моделей и загрузка данных из YAML с валидацией и выводом через logging.

```
import logging
import yaml
from pydantic import BaseModel, Field, ConfigDict, model_validator
from pydantic import EmailStr, HttpUrl
from typing import Literal, Union, List

logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')
RUS_RE = r'^[а-яА-ЯёЁ\s]+$'



class UserSpec(BaseModel):
    user_id:        int
    username:       str = Field(..., pattern=RUS_RE)
    surname:        str = Field(..., pattern=RUS_RE)
    second_name:    str = Field(None, pattern=RUS_RE)
    email:          EmailStr
    status:         Literal['active', 'non-active']
    model_config = ConfigDict(extra="forbid")

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

```
    model_config = ConfigDict(extra="forbid")


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


    @model_validator(mode='after')
    def check_line_prices(self):
        check = self.quantity * self.item_line.price
        if self.line_price != check:
            raise ValueError(f'line_price{self.line_price} != '
```

```
                    f'quantity * item_line.price{self.quantity *
self.item_line.price}')
        return self

    model_config = ConfigDict(extra="forbid")

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

    model_config = ConfigDict(extra="forbid")

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

    model_config = ConfigDict(extra="forbid")

def get_data_from_yaml(yaml_path: str) -> OrdersSpec:
    """
    Загружает и валидирует данные из YAML-файла с использованием
Pydantic.
    """
    try:
        with open(yaml_path, 'r', encoding='utf-8') as f:
            data = yaml.safe_load(f)
        orders = OrdersSpec.model_validate(data)
        logging.info("Данные успешно загружены и валидированы.")
        logging.info(orders.model_dump_json(indent=2, ensure_ascii=False))
```

```
        return orders
    except yaml.YAMLError as e:
        logging.error(f"Ошибка при чтении YAML: {e}")
        raise
    except Exception as e:
        logging.error(f"Ошибка валидации данных: {e}")
        raise


orders = get_data_from_yaml("students_folder/Borodin/LabaMore1/data.yaml")
```