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»

КАФЕДРА «Информационная безопасность»

ОТЧЕТ

по лабораторной работе № 1

по курсу «Искусственный интеллект»

на тему: «Применение однослойной нейронной сети с линейной функцией активации для прогнозирования временных рядов»

Вариант № 6

Студент ИУ8-13М (Группа) _____ (Подпись, дата) Савватеев А. Э. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Басараб М. А. (И. О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Изучение библиотеки pydantic и применение её для валидации данных. Реализация системы валидации данных в формате YAML с использованием Pydantic.

2 ПОСТАНОВКА ЗАДАЧИ

Изучить возможности библиотеки pydantic для валидации данных и применить её для решения задачи валидации входных данных в приложении. Реализовать систему валидации данных в формате YAML с использованием Pydantic.

3 ХОД РАБОТЫ

Класс UserSpec предназначен для валидации данных пользователя. Он содержит следующие поля:

- *user_id*: целое число, идентификатор пользователя;
- *username*: строка, имя пользователя;
- *surname*: строка, фамилия пользователя;
- *second_name*: необязательное строковое поле, отчество пользователя;
- *email*: строка, адрес электронной почты пользователя;
- *status*: строка, статус пользователя (например, "active "inactive").

Чтобы подчеркнуть, что поле является необязательным используется . Для соблюдения требований к данным, класс UserSpec использует валидаторы:

- *validate_email*: проверяет, что email содержит символ "@"и точку ".";
- *validate_status*: проверяет, что статус является одним из допустимых значений ("active "non-active").

Класс ProfileSpec предназначен для валидации профиля пользователя. Он содержит следующие поля:

- *user_id*: целое число, идентификатор пользователя;
- *username*: строка, имя пользователя;
- *surname*: строка, фамилия пользователя;
- *email*: строка, адрес электронной почты пользователя;
- *status*: строка, статус пользователя (например, "active "inactive");
- *bio*: непустое строковое поле, биография пользователя;
- *url*: непустое строковое поле, URL-адрес пользователя.

Для соблюдения требований к данным, класс ProfileSpec использует валидаторы:

- *validate_email*: проверяет, что email содержит символ "@"и точку ".";
- *validate_status*: проверяет, что статус является одним из допустимых значений ("active "non-active");
- *validate_bio*: проверяет, что биография не пустая и написана на русском языке;
- *validate_url*: проверяет, что URL содержит с "://".

Класс ItemSpec предназначен для валидации данных товара. Он содержит следующие поля:

- *item_id*: целое число, идентификатор товара;
- *name*: строка, название товара;
- *desc*: строка, описание товара;
- *price*: положительное число, цена товара.

Для соблюдения требований к данным, класс ItemSpec использует валидатор:

- *validate_price*: проверяет, что цена товара является положительным числом;
- *validate_desc*: проверяет, что описание товара не пустое;
- *validate_name*: проверяет, что название товара не пустое.

Класс ServiceSpec предназначен для валидации данных услуги. Он содержит следующие поля:

- *service_id*: целое число, идентификатор услуги;
- *name*: строка, название услуги;

- *desc*: строка, описание услуги;
- *price*: положительное число, цена услуги.

Для соблюдения требований к данным, класс ServiceSpec использует валидатор:

- *validate_price*: проверяет, что цена услуги является положительным числом;
- *validate_desc*: проверяет, что описание услуги на русском;
- *validate_name*: проверяет, что название услуги на русском.

Класс OrderLineSpec предназначен для валидации строки заказа. Он содержит следующие поля:

- *order_id*: целое число, идентификатор заказа;
- *line_id*: целое число, идентификатор строки заказа;
- *item*: объект, представляющий товар или услугу (может быть экземпляром класса ItemSpec или ServiceSpec);
- *quantity*: положительное целое число, количество товара или услуги в строке заказа;
- *line_price*: вычисляемое поле, представляющее общую стоимость строки заказа (вычисляется как произведение количества на цену товара или услуги).

Для соблюдения требований к данным, класс OrderLineSpec использует валидатор:

- *validate_quantity*: проверяет, что количество является положительным целым числом;
- *validate_line_id*: проверяет, что *line_id* является положительным целым числом и меньше или равно максимальному значению;

- *compute_line_price*: вычисляет общую стоимость строки заказа.

Класс OrderSpec предназначен для валидации данных заказа. Он содержит следующие поля:

- *order_id*: целое число, идентификатор заказа;
- *user_info*: объект, представляющий информацию о пользователе (экземпляр класса ProfileSpec);
- *items_line*: список объектов, представляющих строки заказа (список экземпляров класса OrderLineSpec).

Класс OrdersSpec предназначен для валидации списка заказов. Он содержит поле *market_place_orders*, которое является списком объектов OrderSpec, представляющих заказы.

4 ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены возможности библиотеки pydantic для валидации данных. Были реализованы классы для валидации различных сущностей, таких как пользователь, профиль пользователя, товар, услуга, строка заказа, заказ и список заказов. Каждый класс содержит необходимые поля и валидаторы для обеспечения корректности данных. Реализованная система валидации данных позволяет эффективно проверять и обрабатывать данные в формате YAML.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Исходный код программы валидации данных с использованием Pydantic

```
from pydantic import BaseModel, model_validator, HttpUrl,
   computed_field, field_validator
from typing import List, Union, Optional

class UserSpec(BaseModel):
    model_config = {
        "extra": "forbid"
    }
    user_id: int
    username: str
    surname: str
    second_name: Optional[str]
    email: str
    status: str

    @field_validator('status', mode='after')
    @classmethod
    def validate_status(cls, value):
        if value not in ['active', 'non-active']:
            raise ValueError("Status must be 'active' or
               'non-active'")
        return value

    @field_validator('email', mode='after')
    def validate_email(cls, value):
        if '@' not in value or '.' not in value:
            raise ValueError("Email must contain '@' and '.'")
        return value


class ProfileSpec(BaseModel):
    model_config = {
        "extra": "forbid"
    }
    user_id: int
    username: str
```

```
35      surname: str
36      email: str
37      status: str
38      bio: str
39      url: HttpUrl
40
41      @field_validator('status', mode='after')
42      def validate_status(cls, value):
43          if value not in ['active', 'non-active']:
44              raise ValueError("Status must be 'active' or
                    'non-active'")
45          return value
46
47      @field_validator('email', mode='after')
48      def validate_email(cls, value):
49          if '@' not in value or '.' not in value:
50              raise ValueError("Email must contain '@' and '.'")
51          return value
52
53      @field_validator('bio', mode='after')
54      def validate_bio(cls, value):
55          isRussian = all('a' <= char <= 'я' or 'A' <= char <= 'Я'
                or char.isspace() for char in value)
56          if not isRussian:
57              raise ValueError("Bio must contain only Russian
                    alphabet characters")
58          return value
59
60      @field_validator('url', mode='after')
61      def validate_url(cls, value):
62          if '://' not in value:
63              raise ValueError("URL must contain '://'")
64          return value
65
66  class ItemSpec(BaseModel):
67      model_config = {
68          "extra": "forbid"
69      }
70      item_id: int
71      name: str
72      desc: str
```

```
73      price: float
74
75      @field_validator('name', mode='after')
76      def validate_name(cls, value):
77          isRussian = all('а' <= char <= 'я' or 'А' <= char <= 'Я'
                or char.isspace() for char in value)
78          if not isRussian:
79              raise ValueError("Field must contain only Russian
                    alphabet characters")
80          return value
81
82      @field_validator('desc', mode='after')
83      def validate_desc(cls, value):
84          isRussian = all('а' <= char <= 'я' or 'А' <= char <= 'Я'
                or char.isspace() for char in value)
85          if not isRussian:
86              raise ValueError("Field must contain only Russian
                    alphabet characters")
87          return value
88
89      @field_validator('price', mode='after')
90      def validate_price(cls, value):
91          if value <= 0:
92              raise ValueError("Price must be greater than 0")
93          return value
94
95  class ServiceSpec(BaseModel):
96      model_config = {
97          "extra": "forbid"
98      }
99      service_id: int
100     name: str
101     desc: str
102     price: float
103
104     @field_validator('name', mode='after')
105     def validate_name(cls, value):
106         isRussian = all('а' <= char <= 'я' or 'А' <= char <= 'Я'
                or char.isspace() for char in value)
107         if not isRussian:
```

```
108             raise ValueError("Field must contain only Russian
                    alphabet characters")
109         return value
110
111     @field_validator('desc', mode='after')
112     def validate_desc(cls, value):
113         isRussian = all('а' <= char <= 'я' or 'А' <= char <= 'Я'
                or char.isspace() for char in value)
114         if not isRussian:
115             raise ValueError("Field must contain only Russian
                    alphabet characters")
116         return value
117
118     @field_validator('price', mode='after')
119     def validate_price(cls, value):
120         if value <= 0:
121             raise ValueError("Price must be greater than 0")
122         return value
123
124 class OrderLineSpec(BaseModel):
125     model_config = {
126         "extra": "forbid"
127     }
128     order_id: int
129     order_line_id: int
130     item_line: Union[ServiceSpec, ItemSpec]
131     quantity: float
132     line_price: float
133
134     @field_validator('quantity', mode='after')
135     def validate_quantity(cls, value):
136         if value <= 0:
137             raise ValueError("Quantity must be greater than 0")
138         return value
139
140     @model_validator(mode='after')
141     def validate_order_line_id(cls):
142         if cls.order_line_id <= 0 or cls.order_line_id >
                cls.order_id:
143             raise ValueError("Order line ID must be greater than
                    0 and less than or equal to order ID")
```

```
144         return cls
145
146     @computed_field
147     def line_price(self) -> float:
148         return self.quantity * self.item_line.price
149
150 class OrderSpec(BaseModel):
151     model_config = {
152         "extra": "forbid"
153     }
154     order_id: int
155     user_info: ProfileSpec
156     items_line: List[OrderLineSpec]
157
158 class OrdersSpec(BaseModel):
159     model_config = {
160         "extra": "forbid"
161     }
162     market_place_orders: List[OrderSpec]
```