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 № 3
по дисциплине «Программирование на Python»

Тема: «Автотесты в Python»

Выполнил: Бородин Г.
студент группы ИУ8-13М

Проверил: Зотов М.

г. Москва, 2025 г.

1. ЦЕЛЬ РАБОТЫ

Реализовать систему автотестов для своей модели данных с использованием tox+pytest+GithubAction

2. ПОСТАНОВКА ЗАДАЧИ

- Реализовать авто-тесты для вашей модели интернет-магазина
- Добавлен GithubAction, который по пути вашей лабораторный работы, стартует ваши тесты.
- Под каждый класс, метод и т.п. реализовано, как минимум 5 тестовых экземпляров

3. ХОД РАБОТЫ

ИСПОЛЬЗУЕТСЯ В ДАННОЙ РАБОТЕ: язык Python; библиотека pytest для написания и запуска автотестов; tox для запуска тестов в изолированных средах и под разными версиями Python; GitHub Actions для автоматического запуска тестов при push/pull request; библиотека Pydantic (модели предметной области интернет-магазина из лабораторной №2); модуль logging для протоколирования выполнения тестов; YAML-файлы тестовых наборов и фикстура yaml_test_data для загрузки тест-кейсов из конфигураций.

МОЙ ХОД РАБОТЫ:

ШАГ 1. Анализ требований и подготовка структуры проекта.

Определено, что необходимо реализовать автотесты для моделей интернет-магазина (UserSpec, ProfileSpec, ItemSpec, ServiceSpec, OrderLineSpec, OrderSpec, OrdersSpec). Для каждого класса подготовлены тест-наборы (не менее 5 кейсов), включающие как корректные, так и некорректные входные данные.

ШАГ 2. Подготовка тестовых данных в YAML.

Для каждой модели создан отдельный YAML-файл с тест-кейсами (описание кейса, входные данные, ожидаемый результат). Загрузка тест-кейсов осуществляется через фикстуру yaml_test_data, которая читает YAML и возвращает список сценариев для параметризированного запуска.

ШАГ 3. Реализация pytest-тестов.

Для каждой модели реализован отдельный тестовый модуль. Тесты выполняют валидацию через Model.model_validate(value) и сравнивают фактический результат с ожидаемым (успешная валидация или ValidationError). Дополнительно используется logging для вывода статуса каждого тест-кейса. Все тесты помечены маркером @pytest.mark.borodin_lab3 для выборочного запуска.

ШАГ 4. Настройка tox.

Создан файл tox.ini, в котором описаны окружения для нескольких версий Python и зависимости для запуска тестов. В качестве команды запуска используется pytest tests -v, предусмотрен запуск по маркеру лабораторной работы.

ШАГ 5. Настройка GitHub Actions.

Добавлен workflow GitHub Actions, который при push/pull request устанавливает зависимости и запускает тесты (через tox или напрямую pytest). Это обеспечивает автоматическую проверку работоспособности проекта при изменениях в репозитории.

РЕЗУЛЬТАТ: реализован набор автотестов для моделей данных интернет-магазина, обеспечивающий автоматическую проверку корректности валидации (Pydantic) и запуск проверок как локально (pytest/tox), так и в CI (GitHub Actions).

4. ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были изучены принципы построения автотестов на pytest, а также применение tox для запуска тестов в изолированных окружениях и под разными версиями Python. Разработан комплект тестов для моделей интернет-магазина на Pydantic, включающий положительные и отрицательные сценарии (не менее пяти для каждой сущности). Настроена автоматизация проверки через GitHub Actions, обеспечивающая запуск тестов при обновлении кода. Полученная система повышает надёжность проекта и позволяет быстро выявлять ошибки в структуре и правилах валидации данных.

5. ПРИЛОЖЕНИЕ

Реализованный код располагается по следующей ссылке: https://github.com/zezOtik/bmstu--iu8--python/tree/feature/borodin_2/test

Листинг 1 – Реализация pytest-тестов для модели ItemSpec с параметризацией сценариев.

```
import logging


import pytest
from pydantic import ValidationError


from students_folder.Borodin.LabaMore1.lab_1 import ItemSpec


logger = logging.getLogger("test_Item")



@pytest.mark.borodin_lab3
def test_class_profile(yaml_test_data):
    test_cases = yaml_test_data("Borodin/lab_3/ItemSpec.yaml")
    for test_desc, value, answer in test_cases:
        logger.info(f"Testing: {test_desc}")
        try:
            test_class = ItemSpec.model_validate(value)
            # Ожидаем answer = True
            test_answer = True
            assert (
                answer == test_answer
            ), f"Expected validation to fail for {test_desc}, but it passed"
            logger.info(f"{test_desc} PASSED - validation succeeded as expected")
        except ValidationError as e:
```

```
        # Ожидаем answer = False
        test_answer = False
        assert (
            answer == test_answer
        ), f"Expected validation to pass for {test_desc}, but got error: {e}"
        logger.info(f"{test_desc} PASSED - validation failed as expected: {e}")
```

Листинг 2 – Реализация pytest-тестов для модели OrderLineSpec с параметризацией сценариев.

```
import logging


import pytest
from pydantic import ValidationError

from students_folder.Borodin.LabaMore1.lab_1 import OrderLineSpec

logger = logging.getLogger("test_OrderLine")



@pytest.mark.borodin_lab3
def test_class_profile(yaml_test_data):
    test_cases = yaml_test_data("Borodin/lab_3/OrderLineSpec.yaml")
    for test_desc, value, answer in test_cases:
        logger.info(f"Testing: {test_desc}")
        try:
            test_class = OrderLineSpec.model_validate(value)
            # Ожидаем answer = True
            test_answer = True
            assert (
                answer == test_answer
```

```
        ), f"Expected validation to fail for {test_desc}, but it passed"
        logger.info(f"{test_desc} PASSED - validation succeeded as expected")
    except ValidationError as e:
        # Ожидаем answer = False
        test_answer = False
        assert (
            answer == test_answer
        ), f"Expected validation to pass for {test_desc}, but got error: {e}"
        logger.info(f"{test_desc} PASSED - validation failed as expected: {e}")
```

Листинг 3 – Реализация pytest-тестов для модели OrderSpec с параметризацией сценариев.

```
import logging

import pytest
from pydantic import ValidationError

from students_folder.Borodin.LabaMore1.lab_1 import OrderSpec

logger = logging.getLogger("test_Order")



@pytest.mark.borodin_lab3
def test_class_profile(yaml_test_data):
    test_cases = yaml_test_data("Borodin/lab_3/OrderSpec.yaml")
    for test_desc, value, answer in test_cases:
        logger.info(f"Testing: {test_desc}")
        try:
            test_class = OrderSpec.model_validate(value)
            # Ожидаем answer = True
```

```
            test_answer = True
            assert (
                answer == test_answer
            ), f"Expected validation to fail for {test_desc}, but it passed"
            logger.info(f"{test_desc} PASSED - validation succeeded as expected")
        except ValidationError as e:
            # Ожидаем answer = False
            test_answer = False
            assert (
                answer == test_answer
            ), f"Expected validation to pass for {test_desc}, but got error: {e}"
            logger.info(f"{test_desc} PASSED - validation failed as expected: {e}")
```

Листинг 4 – Реализация pytest-тестов для модели OrdersSpec с параметризацией сценариев.

```
import logging

import pytest
from pydantic import ValidationError

from students_folder.Borodin.LabaMore1.lab_1 import OrdersSpec

logger = logging.getLogger("test_Orders")



@pytest.mark.borodin_lab3
def test_class_profile(yaml_test_data):
    test_cases = yaml_test_data("Borodin/lab_3/OrdersSpec.yaml")
    for test_desc, value, answer in test_cases:
        logger.info(f"Testing: {test_desc}")
```

```
        try:
            test_class = OrdersSpec.model_validate(value)
            # Ожидаем answer = True
            test_answer = True
            assert (
                answer == test_answer
            ), f"Expected validation to fail for {test_desc}, but it passed"
            logger.info(f"{test_desc} PASSED - validation succeeded as expected")
        except ValidationError as e:
            # Ожидаем answer = False
            test_answer = False
            assert (
                answer == test_answer
            ), f"Expected validation to pass for {test_desc}, but got error: {e}"
            logger.info(f"{test_desc} PASSED - validation failed as expected: {e}")
```

Листинг 5 – Реализация pytest-тестов для модели ProfileSpec с параметризацией сценариев.

```
import logging


import pytest
from pydantic import ValidationError


from students_folder.Borodin.LabaMore1.lab_1 import ProfileSpec


logger = logging.getLogger("test_Profile")



@pytest.mark.borodin_lab3
def test_class_profile(yaml_test_data):
```

```
    test_cases = yaml_test_data("Borodin/lab_3/ProfileSpec.yaml")
    for test_desc, value, answer in test_cases:
        logger.info(f"Testing: {test_desc}")
        try:
            test_class = ProfileSpec.model_validate(value)
            # Ожидаем answer = True
            test_answer = True
            assert (
                answer == test_answer
            ), f"Expected validation to fail for {test_desc}, but it passed"
            logger.info(f"{test_desc} PASSED - validation succeeded as expected")
        except ValidationError as e:
            # Ожидаем answer = False
            test_answer = False
            assert (
                answer == test_answer
            ), f"Expected validation to pass for {test_desc}, but got error: {e}"
            logger.info(f"{test_desc} PASSED - validation failed as expected: {e}")
```

Листинг 6 – Реализация pytest-тестов для модели ServiceSpec с параметризацией сценариев.

```
import logging


import pytest
from pydantic import ValidationError


from students_folder.Borodin.LabaMore1.lab_1 import ServiceSpec


logger = logging.getLogger("test_Service")
```

```
@pytest.mark.borodin_lab3
def test_class_profile(yaml_test_data):
    test_cases = yaml_test_data("Borodin/lab_3/ServiceSpec.yaml")
    for test_desc, value, answer in test_cases:
        logger.info(f"Testing: {test_desc}")
        try:
            test_class = ServiceSpec.model_validate(value)
            # Ожидаем answer = True
            test_answer = True
            assert (
                answer == test_answer
            ), f"Expected validation to fail for {test_desc}, but it passed"
            logger.info(f"{test_desc} PASSED - validation succeeded as expected")
        except ValidationError as e:
            # Ожидаем answer = False
            test_answer = False
            assert (
                answer == test_answer
            ), f"Expected validation to pass for {test_desc}, but got error: {e}"
            logger.info(f"{test_desc} PASSED - validation failed as expected: {e}")
```

Листинг 7 – Реализация pytest-тестов для модели UserSpec с параметризацией сценариев.

```
import logging

import pytest
from pydantic import ValidationError

from students_folder.Borodin.LabaMore1.lab_1 import UserSpec
```

```
logger = logging.getLogger("test_User")



@pytest.mark.borodin_lab3
def test_class_profile(yaml_test_data):
    test_cases = yaml_test_data("Borodin/lab_3/UserSpec.yaml")
    for test_desc, value, answer in test_cases:
        logger.info(f"Testing: {test_desc}")
        try:
            test_class = UserSpec.model_validate(value)
            # Ожидаем answer = True
            test_answer = True
            assert (
                answer == test_answer
            ), f"Expected validation to fail for {test_desc}, but it passed"
            logger.info(f"{test_desc} PASSED - validation succeeded as expected")
        except ValidationError as e:
            # Ожидаем answer = False
            test_answer = False
            assert (
                answer == test_answer
            ), f"Expected validation to pass for {test_desc}, but got error: {e}"
            logger.info(f"{test_desc} PASSED - validation failed as expected: {e}")
```

Листинг 8 – Конфигурация tox (tox.ini) для запуска pytest в изолированных окружениях.

```
[tox]
envlist = py313, py312, py311, py310
isolated_build = false
```

```
skip_missing_interpreters = true

[testenv]
deps =
    pytest>=8.0.0
    pydantic>=2.0.0
    email-validator>=2.0.0
    pyyaml>=6.0.0
    colorama>=0.4.0

setenv =
    PYTHONPATH = {toxinidir}

commands =
    pytest tests -v

[testenv:all-tests]
setenv =
    PYTHONPATH = {toxinidir}
commands =
    pytest tests/ -v

[testenv:lint]
deps =
    {[testenv]deps}
    flake8
    black
    mypy
setenv =
```

```
    PYTHONPATH = {toxinidir}
commands =
    flake8 students_folder/ tests/
    black --check students_folder/ tests/
    mypy students_folder/ tests/ --ignore-missing-imports

[testenv:format]
deps =
    black
    isort
setenv =
    PYTHONPATH = {toxinidir}
commands =
    black students_folder/ tests/
    isort students_folder/ tests/

[flake8]
max-line-length = 88
extend-ignore = E203, W503
exclude = venv/, .tox/, __pycache__/

# pytest configuration moved to pyproject.toml

[testenv:zmv_lab1]
description = run zmv_lab_1
setenv =
    PYTHONPATH = {toxinidir}
commands =
    pytest -m zmv_lab1 --tb=short
```

```
[testenv:zaa_lab3]
description = run zaa_lab_3
setenv =
    PYTHONPATH = {toxinidir}
deps =
    {[testenv]deps}
    email-validator>=2.0.0
commands =
    pytest -m zaa_lab3 --tb=short

# Zhukova lab 3 environment
[testenv:zhukova_lab3]
description = run zhukova_lab3
deps =
    {[testenv]deps}
    email-validator>=2.0.0
setenv =
    PYTHONPATH = {toxinidir}
commands =
    pytest -m zhukova_lab3 --tb=short

[testenv:nze_lab3]
description = run nze_lab_3
deps =
    {[testenv]deps}
    email-validator>=2.0.0
setenv =
```

```
    PYTHONPATH = {toxinidir}
commands =
    pytest -m nze_lab3 --tb=short

[testenv:sae_lab3]
description = run sae_lab3
deps =
    {[testenv]deps}
    email-validator>=2.0.0
setenv =
    PYTHONPATH = {toxinidir}
commands =
    pytest -m sae_lab3 --tb=short

[testenv:borodin_lab3]
description = run borodin_lab3
deps =
    {[testenv]deps}
    email-validator>=2.0.0
setenv =
    PYTHONPATH = {toxinidir}
commands =
    pytest -m borodin_lab3 --tb=short -v

# ------------------------------------------------
# Документация (Sphinx)
# ------------------------------------------------
[testenv:docs_build]
description = Build Sphinx documentation
```

```
deps =
    sphinx==8.2.3
    sphinx-autodoc-typehints
    sphinx_rtd_theme
    pydantic==2.12.0
setenv =
    PYTHONPATH = {toxinidir}
commands =
    # 1. Автоматическое создание .rst файлов из исходного кода
    sphinx-apidoc -o docs_src/source/ students_folder

    # 2. Сборка документации
    # Исходные файлы: docs_src/source/
    # Вывод: docs/_build
    sphinx-build -M html docs_src/source/ docs/_build -W --keep-going

    # 3. Создание файла .nojekyll для корректной публикации на GitHub Pages
    touch docs/.nojekyll
```