

Лабораторная работа 3

Задача: Вам нужно протестировать класс `AuthManager`, который управляет пользователями, их аутентификацией, а также предоставляет функциональность для подсчета пользователей по странам и перевода средств между ними. В тестах вам нужно продемонстрировать несколько видов тестов: базовые(3 штуки), параметризованные(3 штуки), тестирование исключений(2 штуки), использование фикстур(базы данных) и меток(минимум 2).

Листинг

```
import pytest

class AuthManager:
    def __init__(self, db):
        self.db = db
        self.users = []

    def register_user(self, username, password, country, balance):
        if ";" in username or ";" in password:
            raise ValueError("Invalid input")
        self.users.append({
            "username": username,
            "password": password,
            "country": country,
            "balance": balance
        })

    def authenticate_user(self, username, password):
        for user in self.users:
            if user["username"] == username and user["password"] == password:
                return user
        return None

    def transfer_balance(self, from_user, to_user, amount):
        if from_user['balance'] < amount:
            raise ValueError("Insufficient funds")
        from_user['balance'] -= amount
        to_user['balance'] += amount

    def user_count_by_country(self, country):
        return sum(1 for user in self.users if user["country"] == country)

@pytest.fixture
def db():
    return []

@pytest.fixture
def auth_manager(db):
    return AuthManager(db)

# Базовые тесты
def test_sql_injection_user_registration(auth_manager):
```

```

    with pytest.raises(ValueError, match="Invalid input"):
        auth_manager.register_user("user1; DROP TABLE users", "password123",
                                   "Country", 1000)

def test_sql_injection_user_authentication(auth_manager):
    auth_manager.register_user("user1", "password123", "Country", 1000)
    user = auth_manager.authenticate_user("user1; DROP TABLE users",
                                          "password123")
    assert user is None

def test_user_count_by_country(auth_manager):
    auth_manager.register_user("user1", "password123", "USA", 1000)
    auth_manager.register_user("user2", "password456", "USA", 500)
    auth_manager.register_user("user3", "password789", "Canada", 300)
    count = auth_manager.user_count_by_country("USA")
    assert count == 2

# Параметризованные тесты
@pytest.mark.parametrize("username, password, expected", [
    ("user1", "password123", True),
    ("user2", "wrongpassword", False),
    ("user3", "password123", True)
])
def test_parametrized_authentication(auth_manager, username, password,
                                     expected):
    if expected:
        auth_manager.register_user(username, "password123", "Country", 1000)
        user = auth_manager.authenticate_user(username, "password123")
        assert user is not None
    else:
        user = auth_manager.authenticate_user(username, password)
        assert user is None

@pytest.mark.parametrize("username, country, balance", [
    ("user1", "USA", 1000),
    ("user2", "Canada", 500),
    ("user3", "Mexico", 300)
])
def test_parametrized_user_registration(auth_manager, username, country,
                                       balance):
    auth_manager.register_user(username, "password123", country, balance)
    user = auth_manager.authenticate_user(username, "password123")
    assert user is not None
    assert user["country"] == country
    assert user["balance"] == balance

@pytest.mark.parametrize("from_user_balance, to_user_balance,
                          transfer_amount, expected_from, expected_to", [
    (100, 50, 30, 70, 80),
    (200, 100, 150, 50, 250),
    (300, 200, 100, 200, 300)
])
def test_parametrized_balance_transfer(auth_manager, from_user_balance,
                                       to_user_balance, transfer_amount, expected_from, expected_to):

```

```

    auth_manager.register_user("user1", "password123", "CountryA",
from_user_balance)
    auth_manager.register_user("user2", "password456", "CountryB",
to_user_balance)
    auth_manager.transfer_balance(auth_manager.users[0],
auth_manager.users[1], transfer_amount)
    assert auth_manager.users[0]['balance'] == expected_from
    assert auth_manager.users[1]['balance'] == expected_to

# Тестирование исключений
def test_insufficient_funds_exception(auth_manager):
    auth_manager.register_user("user1", "password123", "CountryA", 100)
    auth_manager.register_user("user2", "password123", "CountryB", 50)
    with pytest.raises(ValueError, match="Insufficient funds"):
        auth_manager.transfer_balance(auth_manager.users[0],
auth_manager.users[1], 200)

def test_nonexistent_user_authentication(auth_manager):
    user = auth_manager.authenticate_user("nonexistent_user", "any_password")
    assert user is None

# Метки
@pytest.mark.slow
def test_slow_operation(auth_manager):
    auth_manager.register_user("user1", "password123", "Country", 1000)
    user = auth_manager.authenticate_user("user1", "password123")
    assert user is not None

@pytest.mark.injection
def test_sql_injection(auth_manager):
    with pytest.raises(ValueError, match="Invalid input"):
        auth_manager.register_user("user1; DROP TABLE users", "password123",
"Country", 1000)

```

Тест 1 – Базовый (test_sql_injection_user_registration(auth_manager))

```

def test_sql_injection_user_registration(auth_manager):
    with pytest.raises(ValueError, match="Invalid input"):
        auth_manager.register_user("user1; DROP TABLE users", "password123",
"Country", 1000)

```

Тест проверяет, что при попытке зарегистрировать пользователя с именем или паролем, содержащими символ ;, возникает исключение ValueError. Тест Успешно проверяет защиту от SQL-инъекций. Исключение ValueError вызывается, когда имя или пароль содержат символ ;.

Тест 2 – параметризированный (test_parametrized_authentication(auth_manager, username, password, expected))

```

@pytest.mark.parametrize("username, password, expected", [

```

```

    ("user1", "password123", True),
    ("user2", "wrongpassword", False),
    ("user3", "password123", True)
])
def test_parametrized_authentication(auth_manager, username, password,
expected):
    if expected:
        auth_manager.register_user(username, "password123", "Country", 1000)
        user = auth_manager.authenticate_user(username, "password123")
        assert user is not None
    else:
        user = auth_manager.authenticate_user(username, password)
        assert user is None

```

Тест проверяет аутентификацию пользователей с различными именами и паролями. Использует параметризацию для проверки нескольких случаев. Если ожидается успешная аутентификация, пользователь должен быть найден; если нет, возвращается None. В ходе теста успешно проверяются различные сценарии аутентификации, включая как успешные, так и неудачные попытки

Тест 3 – Тестирование исключений
(test_insufficient_funds_exception(auth_manager))

```

def test_insufficient_funds_exception(auth_manager):
    auth_manager.register_user("user1", "password123", "CountryA", 100)
    auth_manager.register_user("user2", "password123", "CountryB", 50)
    with pytest.raises(ValueError, match="Insufficient funds"):
        auth_manager.transfer_balance(auth_manager.users[0],
auth_manager.users[1], 200)

```

Тест проверяет, что при попытке перевести средства, когда у отправителя недостаточно средств, возникает исключение ValueError. Исключение должно быть вызвано с сообщением "Insufficient funds". В ходе теста исключение ValueError возникает при попытке перевести средства без достаточного баланса.

Тест 4 – тест меток Тест на SQL-инъекции (@pytest.mark.injection
def test_sql_injection(auth_manager))

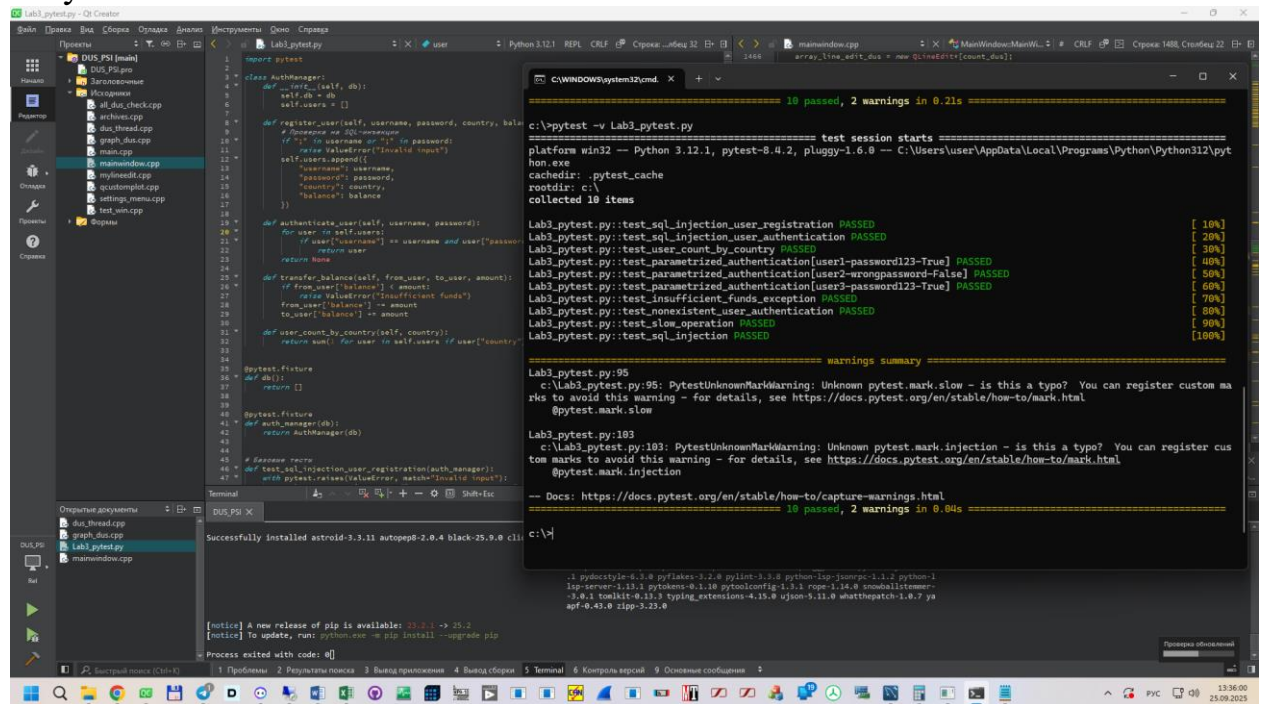
```

@pytest.mark.injection
def test_sql_injection(auth_manager):
    with pytest.raises(ValueError, match="Invalid input"):
        auth_manager.register_user("user1; DROP TABLE users", "password123",
"Country", 1000)

```

Данный тест проверяет, что при попытке зарегистрировать пользователя с именем, содержащим SQL-инъекцию, возникает исключение ValueError. Тест Успешно вызывает исключение ValueError при регистрации пользователя с SQL-инъекцией.

Результаты тестов:



The screenshot shows the Qt Creator IDE with a C++ project named 'DUS_PSI'. The main window displays the source code of 'Lab3_pytest.py', which includes a 'AuthManager' class and several test functions. The terminal window shows the execution of 'pytest' on 'Lab3_pytest.py', resulting in 10 passed tests and 2 warnings. The warnings are related to 'PytestUnknownMarkWarning' and 'PytestUnknownMarkWarning'.

```
1 import pytest
2
3 class AuthManager:
4     def __init__(self, db):
5         self.db = db
6         self.users = []
7
8     def register_user(self, username, password, country, balance):
9         # Проверка на SQL-инъекцию
10        if " " in username or " " in password:
11            raise ValueError("Invalid input")
12        self.users.append({
13            "username": username,
14            "password": password,
15            "country": country,
16            "balance": balance
17        })
18
19    def authenticate_user(self, username, password):
20        for user in self.users:
21            if user["username"] == username and user["password"] == password:
22                return user
23        return None
24
25    def transfer_balance(self, from_user, to_user, amount):
26        if from_user["balance"] < amount:
27            raise ValueError("Insufficient funds")
28        from_user["balance"] -= amount
29        to_user["balance"] += amount
30
31    def user_count_by_country(self, country):
32        return sum(1 for user in self.users if user["country"] == country)
33
34
35 @pytest.fixture
36 def db():
37     return []
38
39 @pytest.fixture
40 def auth_manager(db):
41     return AuthManager(db)
42
43 # Exception raise
44 def test_sql_injection_user_registration(auth_manager):
45     with pytest.raises(ValueError, match="Invalid input"):
46         auth_manager.register_user(" ", " ", " ", " ")
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
10 passed, 2 warnings in 0.21s
===== test session starts =====
platform win32 -- Python 3.12.1, pytest-8.4.2, pluggy-1.6.0 -- C:\Users\user\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: c:\
collected 10 items

Lab3_pytest.py::test_sql_injection_user_registration PASSED [ 10%]
Lab3_pytest.py::test_sql_injection_user_authentication PASSED [ 20%]
Lab3_pytest.py::test_user_count_by_country PASSED [ 30%]
Lab3_pytest.py::test_parametrized_authentication[user1-password123-True] PASSED [ 40%]
Lab3_pytest.py::test_parametrized_authentication[user2-wrongpassword-False] PASSED [ 50%]
Lab3_pytest.py::test_parametrized_authentication[user3-password123-True] PASSED [ 60%]
Lab3_pytest.py::test_insufficient_funds_exception PASSED [ 70%]
Lab3_pytest.py::test_nonexistent_user_authentication PASSED [ 80%]
Lab3_pytest.py::test_slow_operation PASSED [ 90%]
Lab3_pytest.py::test_sql_injection PASSED [100%]

===== warnings summary =====

Lab3_pytest.py:95
c:\Lab3_pytest.py:95: PytestUnknownMarkWarning: Unknown pytest.mark.slow - is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-to/mark.html
@pytest.mark.slow

Lab3_pytest.py:103
c:\Lab3_pytest.py:103: PytestUnknownMarkWarning: Unknown pytest.mark.injection - is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-to/mark.html
@pytest.mark.injection

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 10 passed, 2 warnings in 0.04s =====

.c:\python\python312\python.exe
.pydocstyle-6.3.0 pyflakes-3.2.0 pylint-3.3.0 python-lsp-jsonrpc-1.1.2 python-lsp-server-1.13.1 pytokens-0.1.18 pytoolconfig-1.3.3 rope-1.14.0 unobalister-1.0.1 toolbelt-0.11.0 typing_extensions-4.13.0 ujson-5.11.0 whatthepatch-1.0.7 ya-apf-0.43.0 zipp-3.23.0

[notice] A new release of pip is available: 23.2.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
Process exited with code: 0
```