Лабораторная работа 3

Листинг файла:

```python
import pytest
import sqlite3
from auth_manager import AuthManager

# Регистрируем кастомные метки
def pytest_configure(config):
    config.addinivalue_line("markers", "basic: Базовые тесты")
    config.addinivalue_line("markers", "parametrized: Параметризованные тесты")
    config.addinivalue_line("markers", "exception: Тесты исключений")
    config.addinivalue_line("markers", "security: Тесты безопасности")
    config.addinivalue_line("markers", "database: Тесты работы с БД")
    config.addinivalue_line("markers", "integration: Интеграционные тесты")

# ФИКСТУРЫ
@pytest.fixture
def db_connection():
    """Создает временную базу данных в памяти."""
    conn = sqlite3.connect(':memory:')
    yield conn
    conn.close()

@pytest.fixture
def auth_manager(db_connection):
    """Создает экземпляр AuthManager с подключением к временной БД."""
    return AuthManager(db_connection)

@pytest.fixture
def populated_auth_manager(auth_manager):
    """Фикстура с предзаполненными данными."""
    test_users = [
        ("user1", "pass1", "USA", 1000.0),
        ("user2", "pass2", "USA", 2000.0),
        ("user3", "pass3", "Canada", 1500.0),
        ("user4", "pass4", "Germany", 3000.0),
        ("user5", "pass5", "France", 500.0)
    ]

    for username, password, country, balance in test_users:
        auth_manager.register_user(username, password, country, balance)

    return auth_manager

# БАЗОВЫЕ ТЕСТЫ (6 штук)
@pytest.mark.basic
@pytest.mark.database
def test_register_user(auth_manager):
    """Тест регистрации пользователя."""
```

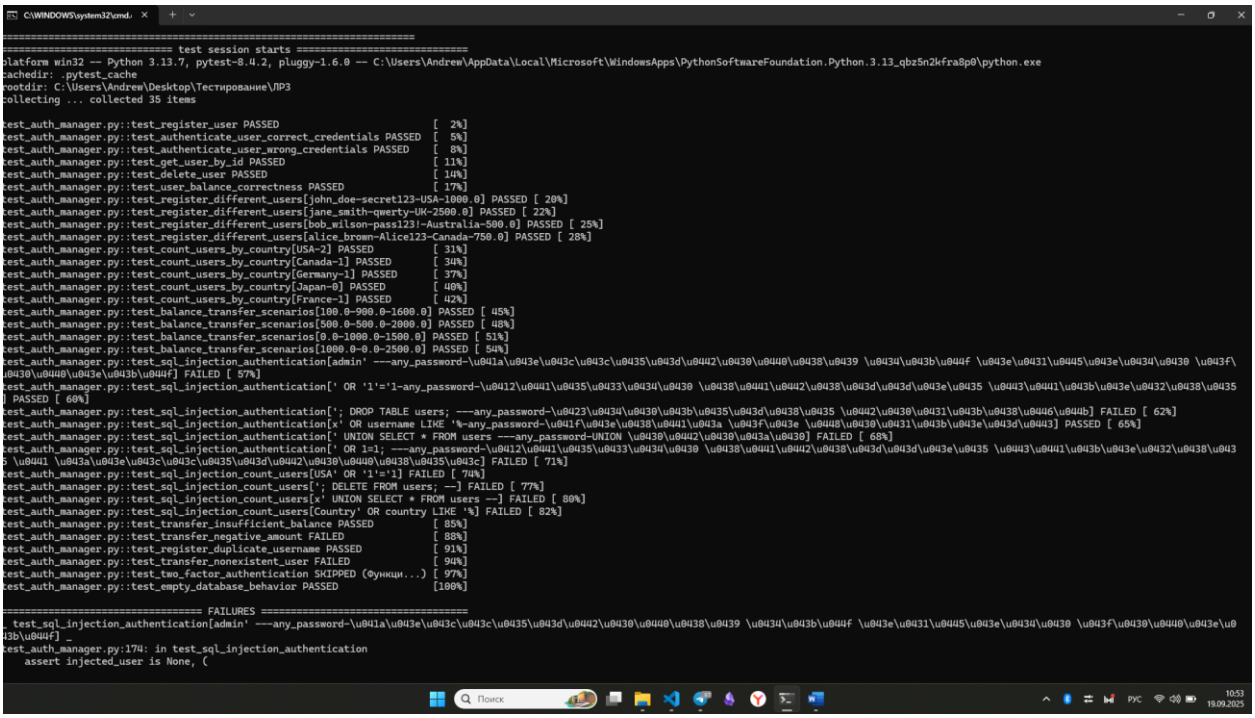
```python
    auth_manager.register_user("test_user", "password123", "Testland", 100.0)
    user = auth_manager.authenticate_user("test_user", "password123")
    assert user is not None
    assert user[1] == "test_user"

@pytest.mark.basic
@pytest.mark.database
def test_authenticate_user_correct_credentials(auth_manager):
    """Тест успешной аутентификации."""
    auth_manager.register_user("user1", "pass1", "CountryA", 100.0)
    user = auth_manager.authenticate_user("user1", "pass1")
    assert user is not None
    assert user[1] == "user1"

@pytest.mark.basic
@pytest.mark.database
def test_authenticate_user_wrong_credentials(auth_manager):
    """Тест неуспешной аутентификации."""
    auth_manager.register_user("user1", "pass1", "CountryA", 100.0)
    user = auth_manager.authenticate_user("user1", "wrong_pass")
    assert user is None

@pytest.mark.basic
@pytest.mark.database
def test_get_user_by_id(auth_manager):
    """Тест получения пользователя по ID."""
    auth_manager.register_user("user1", "pass1", "CountryA", 500.0)
    user = auth_manager.get_user_by_id(1)
    assert user is not None
    assert user[1] == "user1"
    assert user[4] == 500.0  # balance

@pytest.mark.basic
@pytest.mark.database
def test_delete_user(auth_manager):
    """Тест удаления пользователя."""
    auth_manager.register_user("to_delete", "pass", "CountryA", 100.0)
    user_before = auth_manager.authenticate_user("to_delete", "pass")
    assert user_before is not None

    auth_manager.delete_user(user_before[0])
    user_after = auth_manager.authenticate_user("to_delete", "pass")
    assert user_after is None

@pytest.mark.basic
@pytest.mark.database
def test_user_balance_correctness(auth_manager):
    """Тест корректности баланса пользователя."""
    auth_manager.register_user("rich_user", "pass", "Monaco", 9999.99)
    user = auth_manager.authenticate_user("rich_user", "pass")
    assert user[4] == 9999.99  # balance
```

```python
# ПАРАМЕТРИЗОВАННЫЕ ТЕСТЫ (6 штук)
@pytest.mark.parametrize("username, password, country, balance", [
    ("john_doe", "secret123", "USA", 1000.0),
    ("jane_smith", "qwerty", "UK", 2500.0),
    ("bob_wilson", "pass123!", "Australia", 500.0),
    ("alice_brown", "Alice123", "Canada", 750.0),
])
@pytest.mark.parametrized
@pytest.mark.database
def test_register_different_users(auth_manager, username, password, country,
balance):
    """Параметризованный тест регистрации различных пользователей."""
    auth_manager.register_user(username, password, country, balance)
    user = auth_manager.authenticate_user(username, password)
    assert user is not None
    assert user[1] == username
    assert user[3] == country
    assert user[4] == balance

@pytest.mark.parametrize("country, expected_count", [
    ("USA", 2),
    ("Canada", 1),
    ("Germany", 1),
    ("Japan", 0),
    ("France", 1),
])
@pytest.mark.parametrized
@pytest.mark.database
def test_count_users_by_country(populated_auth_manager, country, expected_count):
    """Параметризованный тест подсчета пользователей по странам."""
    count = populated_auth_manager.count_users_by_country(country)
    assert count == expected_count

@pytest.mark.parametrize("transfer_amount, expected_from_balance,
expected_to_balance", [
    (100.0, 900.0, 1600.0),   # Обычный перевод
    (500.0, 500.0, 2000.0),   # Большой перевод
    (0.0, 1000.0, 1500.0),    # Перевод нуля
    (1000.0, 0.0, 2500.0),    # Перевод всего баланса
])
@pytest.mark.parametrized
@pytest.mark.integration
def test_balance_transfer_scenarios(populated_auth_manager, transfer_amount,
expected_from_balance, expected_to_balance):
    """Параметризованный тест различных сценариев перевода."""
    # user1 (USA, 1000) -> user3 (Canada, 1500)
    populated_auth_manager.transfer_balance(1, 3, transfer_amount)

    from_balance = populated_auth_manager.get_user_by_id(1)[4]
    to_balance = populated_auth_manager.get_user_by_id(3)[4]
```

```
    assert from_balance == expected_from_balance
    assert to_balance == expected_to_balance

@pytest.mark.parametrize("malicious_username, malicious_password, description", [
    ("admin' --", "any_password", "Комментарий для обхода пароля"),
    ("' OR '1'='1", "any_password", "Всегда истинное условие"),
    ("'; DROP TABLE users; --", "any_password", "Удаление таблицы"),
    ("x' OR username LIKE '%", "any_password", "Поиск по шаблону"),
    ("' UNION SELECT * FROM users --", "any_password", "UNION атака"),
    ("' OR 1=1; --", "any_password", "Всегда истинное условие с комментарием"),
])
@pytest.mark.parametrized
@pytest.mark.security
def test_sql_injection_authentication(auth_manager, malicious_username,
malicious_password, description):
    """
    Тест SQL инъекции в аутентификации.
    Должен ПРОВАЛИТЬСЯ для доказательства уязвимости.
    """
    # Создаем тестового пользователя
    auth_manager.register_user("admin", "admin123", "Adminland", 9999.0)

    # Пытаемся аутентифицироваться с помощью инъекции
    injected_user = auth_manager.authenticate_user(malicious_username,
malicious_password)

    # Утверждаем, что инъекция НЕ должна работать (но она работает из-за
уязвимости)
    # Этот assert должен УПАСТЬ для всех инъекций
    assert injected_user is None, (
        f"SQL Injection vulnerability found! "
        f"Injection: username='{malicious_username}',
password='{malicious_password}'. "
        f"Description: {description}. "
        f"Returned user: {injected_user}"
    )

@pytest.mark.parametrize("malicious_country", [
    "USA' OR '1'='1",
    "'; DELETE FROM users; --",
    "x' UNION SELECT * FROM users --",
    "Country' OR country LIKE '%",
])
@pytest.mark.parametrized
@pytest.mark.security
def test_sql_injection_count_users(auth_manager, malicious_country):
    """Тест SQL инъекции в подсчете пользователей."""
    auth_manager.register_user("user1", "pass1", "USA", 100.0)
    auth_manager.register_user("user2", "pass2", "Canada", 200.0)
```

```
    # Должно вернуть 0, но из-за уязвимости может вернуть другое значение
    count = auth_manager.count_users_by_country(malicious_country)
    # Этот assert может упасть, доказывая уязвимость
    assert count == 0, f"SQL Injection vulnerability with: {malicious_country}"

# ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ (4 штуки)
@pytest.mark.exception
def test_transfer_insufficient_balance(populated_auth_manager):
    """Тест исключения при недостаточном балансе."""
    with pytest.raises(ValueError, match="Insufficient funds"):
        populated_auth_manager.transfer_balance(1, 2, 5000.0)  # user1 имеет
только 1000

@pytest.mark.exception
def test_transfer_negative_amount(populated_auth_manager):
    """Тест исключения при отрицательной сумме перевода."""
    with pytest.raises(ValueError):
        populated_auth_manager.transfer_balance(1, 2, -100.0)

@pytest.mark.exception
def test_register_duplicate_username(auth_manager):
    """Тест исключения при дубликате username."""
    auth_manager.register_user("duplicate", "pass1", "CountryA", 100.0)
    with pytest.raises(sqlite3.IntegrityError):
        auth_manager.register_user("duplicate", "pass2", "CountryB", 200.0)

@pytest.mark.exception
def test_transfer_nonexistent_user(populated_auth_manager):
    """Тест исключения при переводе несуществующему пользователю."""
    with pytest.raises(sqlite3.OperationalError):
        populated_auth_manager.transfer_balance(1, 999, 100.0)  # user 999 не
существует

# ПРОПУСКАЕМЫЙ ТЕСТ
@pytest.mark.skip(reason="Функционал двухфакторной аутентификации еще в
разработке")
def test_two_factor_authentication(auth_manager):
    """Тест для будущего функционала."""
    assert False

# ТЕСТ НА ПОВЕДЕНИЕ ПУСТОЙ БАЗЫ
@pytest.mark.database
def test_empty_database_behavior(auth_manager):
    """Тест поведения с пустой базой данных."""
    assert auth_manager.count_users_by_country("USA") == 0
    assert auth_manager.authenticate_user("nonexistent", "pass") is None
    assert auth_manager.get_user_by_id(1) is None
```

Результат:


C:\WINDOWS\system32\cmd.
test session starts
platform win32 -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0 -- C:\Users\Andrew\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Andrew\Desktop\Тестирование\ЛР3
collecting ... collected 35 items
test_auth_manager.py::test_register_user PASSED [2%]
test_auth_manager.py::test_authenticate_user_correct_credentials PASSED [5%]
test_auth_manager.py::test_authenticate_user_wrong_credentials PASSED [8%]
test_auth_manager.py::test_get_user_by_id PASSED [11%]
test_auth_manager.py::test_delete_user PASSED [14%]
test_auth_manager.py::test_user_balance_correctness PASSED [17%]
test_auth_manager.py::test_register_different_users[john_doe-secret123-USA-1000.0] PASSED [20%]
test_auth_manager.py::test_register_different_users[jane_smith-qwerty-UK-2500.0] PASSED [22%]
test_auth_manager.py::test_register_different_users[bob_wilson-pass123!-Australia-500.0] PASSED [25%]
test_auth_manager.py::test_register_different_users[alice_brown-Alice123-Canada-750.0] PASSED [28%]
test_auth_manager.py::test_count_users_by_country[USA-2] PASSED [31%]
test_auth_manager.py::test_count_users_by_country[Canada-1] PASSED [34%]
test_auth_manager.py::test_count_users_by_country[Germany-1] PASSED [37%]
test_auth_manager.py::test_count_users_by_country[Japan-0] PASSED [40%]
test_auth_manager.py::test_count_users_by_country[France-1] PASSED [42%]
test_auth_manager.py::test_balance_transfer_scenarios[100.0-900.0-1600.0] PASSED [45%]
test_auth_manager.py::test_balance_transfer_scenarios[500.0-500.0-2000.0] PASSED [48%]
test_auth_manager.py::test_balance_transfer_scenarios[0.0-1000.0-1500.0] PASSED [51%]
test_auth_manager.py::test_balance_transfer_scenarios[1000.0-0.0-2500.0] PASSED [54%]
test_auth_manager.py::test_sql_injection_authentication[admin' --any_password-\u041a\u043e\u043c\u043c\u0435\u043d\u0442\u0430\u0440\u0438\u0439 \u0434\u043b\u044f \u043e\u0431\u0445\u043e\u0434\u0430 \u043f\u0430\u0440\u043e\u043b\u044f] FAILED [57%]
test_auth_manager.py::test_sql_injection_authentication[' OR '1'='1-any_password-\u0412\u0441\u0435\u0433\u0434\u0430 \u0438\u0441\u0442\u0438\u043d\u043d\u043e\u0435 \u0443\u0441\u043b\u043e\u0432\u0438\u0435] PASSED [60%]
test_auth_manager.py::test_sql_injection_authentication['; DROP TABLE users; --any_password-\u0423\u0434\u0430\u043b\u0435\u043d\u0438\u0435 \u0442\u0430\u0431\u043b\u0438\u0446\u044b] FAILED [62%]
test_auth_manager.py::test_sql_injection_authentication[x' OR username LIKE '%-any_password-\u041f\u043e\u0438\u0441\u043a \u043f\u043e \u0448\u0430\u0431\u043b\u043e\u043d\u0443] PASSED [65%]
test_auth_manager.py::test_sql_injection_authentication[' UNION SELECT * FROM users --any_password-UNION \u0438\u0442\u0430\u043a\u0430] FAILED [68%]
test_auth_manager.py::test_sql_injection_authentication[' OR 1=1; --any_password-\u0412\u0441\u0435\u0433\u0434\u0430 \u0438\u0441\u0442\u0438\u043d\u043d\u043e\u0435 \u0443\u0441\u043b\u043e\u0432\u0438\u0435] FAILED [71%]
test_auth_manager.py::test_sql_injection_count_users[USA' OR '1'='1] FAILED [74%]
test_auth_manager.py::test_sql_injection_count_users['; DELETE FROM users; --] FAILED [77%]
test_auth_manager.py::test_sql_injection_count_users[x' UNION SELECT * FROM users --] FAILED [80%]
test_auth_manager.py::test_sql_injection_count_users[Country' OR country LIKE '%] FAILED [82%]
test_auth_manager.py::test_transfer_insufficient_balance PASSED [85%]
test_auth_manager.py::test_transfer_negative_amount FAILED [88%]
test_auth_manager.py::test_register_duplicate_username PASSED [91%]
test_auth_manager.py::test_transfer_nonexistent_user FAILED [94%]
test_auth_manager.py::test_two_factor_authentication SKIPPED (Функци...) [97%]
test_auth_manager.py::test_empty_database_behavior PASSED [100%]
FAILURES
_ test_sql_injection_authentication[admin' --any_password-\u041a\u043e\u043c\u043c\u0435\u043d\u0442\u0430\u0440\u0438\u0439 \u0434\u043b\u044f \u043e\u0431\u0445\u043e\u0434\u0430 \u043f\u0430\u0440\u043e\u043b\u044f] _
test_auth_manager.py:174: in test_sql_injection_authentication
assert injected_user is None, (
Поиск
РУС
10:53
19.09.2025