

Практическая работа №3

1.Код класса для тестирования:

```
import pytest
import sqlite3
from auth_manager import AuthManager

# Фикстуры для базы данных
@pytest.fixture
def database():
    """Создание временной БД в памяти"""
    conn = sqlite3.connect(":memory:")
    yield conn
    conn.close()

@pytest.fixture
def manager(database):
    """Создание менеджера аутентификации"""
    return AuthManager(database)

@pytest.fixture
def preloaded_manager(manager):
    """Менеджер с предустановленными данными"""
    manager.register_user("alex", "secret123", "Canada", 2500.0)
    manager.register_user("maria", "mypass", "Brazil", 1800.0)
    manager.register_user("john", "john123", "Canada", 3200.0)
    manager.register_user("sara", "sara_pass", "Japan", 1500.0)
    return manager
```

```
# Базовые тесты (3 штуки)

def test_table_creation(manager):
    """Проверка создания таблиц в БД"""

    cursor = manager.connection.cursor()

    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")

    table = cursor.fetchone()

    assert table is not None
    assert table[0] == 'users'

def test_user_registration(manager):
    """Тестирование регистрации нового пользователя"""

    manager.register_user("demo_user", "demo_pass", "Australia", 5000.0)

    cursor = manager.connection.cursor()

    cursor.execute("SELECT * FROM users WHERE username = 'demo_user'")

    user_data = cursor.fetchone()

    assert user_data is not None
    assert user_data[1] == "demo_user"
    assert user_data[2] == "demo_pass"
    assert user_data[3] == "Australia"
    assert user_data[4] == 5000.0

def test_successful_login(manager):
    """Проверка успешного входа в систему"""

    pass
```

```
manager.register_user("test_login", "login_pass", "Germany", 1000.0)
user = manager.authenticate_user("test_login", "login_pass")

assert user is not None
assert user[1] == 'test_login'
```

```
# Параметризованные тесты (3 штуки)

@pytest.mark.parametrize("user, pwd, country, funds", [
    ("client1", "cl1_pass", "Italy", 4200.0),
    ("client2", "cl2_pass", "France", 3800.0),
    ("client3", "cl3_pass", "Spain", 2900.0),
])
def test_multiple_registrations(manager, user, pwd, country, funds):
    """Параметризованная проверка регистрации пользователей"""
    manager.register_user(user, pwd, country, funds)
    result = manager.authenticate_user(user, pwd)

    assert result is not None
    assert result[1] == user
    assert result[3] == country
    assert result[4] == funds

@pytest.mark.parametrize("login, secret, result", [
    ("unknown", "wrong", None),
    ("alex", "incorrect", None),
    ("", "", None),
])
def test_failed_authentication(preloaded_manager, login, secret, result):
    """Параметризованная проверка неудачных попыток входа"""
    pass
```

```
auth_result = preloaded_manager.authenticate_user(login, secret)
assert auth_result == result

@pytest.mark.parametrize("nation, expected", [
    ("Canada", 2),
    ("Brazil", 1),
    ("Japan", 1),
    ("China", 0),
])
def test_national_user_count(preloaded_manager, nation, expected):
    """Параметризованный подсчет пользователей по
национальности"""
    count = preloaded_manager.count_users_by_country(nation)
    assert count == expected

# Тестирование исключений (2 штуки)
def test_insufficient_balance_transfer(preloaded_manager):
    """Проверка исключения при недостаточном балансе"""
    with pytest.raises(ValueError, match="Insufficient funds"):
        preloaded_manager.transfer_balance(2, 1, 5000.0)

def test_transfer_to_invalid_user(preloaded_manager):
    """Проверка поведения при переводе несуществующему
пользователю"""
    try:
        preloaded_manager.transfer_balance(1, 999, 300.0)
        pytest.fail("Ожидалось исключение для несуществующего
пользователя")
    except Exception:
        # Ожидаемое поведение из-за SQL уязвимости
```

```
pass
```

Тесты с использованием фикстур

```
def test_user_removal(preloaded_manager):
```

```
    """Тестируем удаление пользователя с использованием  
    фикстур"""
```

```
    # Проверяем существование пользователя
```

```
    user = preloaded_manager.get_user_by_id(1)
```

```
    assert user is not None
```

```
    # Удаляем пользователя
```

```
    preloaded_manager.delete_user(1)
```

```
    # Проверяем удаление
```

```
    removed_user = preloaded_manager.get_user_by_id(1)
```

```
    assert removed_user is None
```

Тесты с метками (минимум 2)

```
@pytest.mark.stress
```

```
def test_high_volume_registrations(manager):
```

```
    """Стресс-тест массовой регистрации пользователей"""
```

```
    for i in range(200):
```

```
        manager.register_user(f"load_user_{i}", f"pass_{i}", f"nation_{i % 15}",  
        1000 + i)
```

```
    total = manager.count_users_by_country("nation_0")
```

```
    assert total >= 13 # 200 пользователей / 15 стран
```

```
@pytest.mark.vulnerability
```

```
def test_sql_injection_in_registration(manager):
    """Тестирование уязвимости SQL инъекции при регистрации"""
    manager.register_user("hacker'; DROP TABLE users; --", "attack",
    "USA", 1000)

    cursor = manager.connection.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table'
AND name='users';")
    tables = cursor.fetchall()

    assert len(tables) == 1 # Таблица должна сохраниться

@pytest.mark.vulnerability
def test_sql_injection_in_authentication(manager):
    """Тестирование уязвимости SQL инъекции при аутентификации"""
    manager.register_user("legit_user", "safe_pass", "UK", 2000.0)

    # SQL инъекция для обхода аутентификации
    user = manager.authenticate_user("legit_user' OR '1'='1",
    "any_password")
    assert user is not None # Инъекция должна сработать

# Пропускаемый тест
@pytest.mark.skip(reason="Требует дополнительной конфигурации
окружения")
def test_environment_specific():
    """Тест, зависящий от специфичного окружения"""
    assert False

# Дополнительные security тесты
```

```
@pytest.mark.vulnerability
def test_sql_injection_vectors(manager):
    """Тестирование различных векторов SQL инъекций"""
    injection_patterns = [
        "test' OR '1='1",
        "admin'; DELETE FROM users; --",
        "user' UNION SELECT * FROM users --",
    ]
    for pattern in injection_patterns:
        try:
            manager.register_user(pattern, "test", "test_country", 1000)
            # Проверяем сохранность таблицы
            cursor = manager.connection.cursor()
            cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
            table_exists = cursor.fetchone() is not None
            assert table_exists
        except Exception:
            # Игнорируем возможные ошибки выполнения
            pass
```

2.Базовые тесты:

2.1.БАЗОВЫЙ-ТЕСТ

```
42 > def test_user_registration(manager):
43     """Тестирование регистрации нового пользователя"""
44     manager.register_user(username="demo_user", password="demo_pass", country="Australia", balance=5000.0)
45
46     cursor = manager.connection.cursor()
47     cursor.execute("SELECT * FROM users WHERE username = 'demo_user'")
48     user_data = cursor.fetchone()
49
50     assert user_data is not None
51     assert user_data[1] == "demo_user"
52     assert user_data[2] == "demo_pass"
53     assert user_data[3] == "Australia"
54     assert user_data[4] == 5000.0
55
56
```

2.2. БАЗОВЫЙ-ТЕСТ

```
57 > def test_successful_login(manager):
58     """Проверка успешного входа в систему"""
59     manager.register_user(username="test_login", password="login_pass", country="Germany", balance=1000.0)
60     user = manager.authenticate_user(username="test_login", password="login_pass")
61
62     assert user is not None
63     assert user[1] == 'test_login'
64
65
un Python tests in 3.py ×
> G ⌂ ✓ ⌂ ⌂ ⓘ : 
✖ Tests failed: 2, passed: 18, ignored: 1 of 21 tests - 3 ms
3.py::test_successful_login PASSED [ 14%]
```

2.3.БАЗОВЫЙ-ТЕСТ

3. Параметризованные тесты:

3.1. ПАРАМЕТРИЗОВАННЫЙ-ТЕСТ

```
83     v @pytest.mark.parametrize("login, secret, result", [
84         ("unknown", "wrong", None),
85         ("alex", "incorrect", None),
86         ("", "", None),
87     ])
88 D  def test_failed_authentication(preloaded_manager, login, secret, result):
89     """Параметризованная проверка неудачных попыток входа"""
90     auth_result = preloaded_manager.authenticate_user(login, secret)
91     assert auth_result == result
92
93
```

Run Python tests in 3.py

Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

3.py::test_failed_authentication[unknown-wrong-None] PASSED	[33%]
3.py::test_failed_authentication[alex-incorrect-None] PASSED	[38%]
3.py::test_failed_authentication[--None] PASSED	[42%]

3.2. ПАРАМЕТРИЗОВАННЫЙ-ТЕСТ

```
94     @pytest.mark.parametrize("nation, expected", [
95         ("Canada", 2),
96         ("Brazil", 1),
97         ("Japan", 1),
98         ("China", 0),
99     ])
100 D def test_national_user_count(preloaded_manager, nation, expected):
101     """Параметризованный подсчет пользователей по национальности"""
102     count = preloaded_manager.count_users_by_country(nation)
103     assert count == expected
104
105
```

Run Python tests in 3.py

Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

3.py::test_national_user_count[Canada-2] PASSED	[47%]
3.py::test_national_user_count[Brazil-1] PASSED	[52%]
3.py::test_national_user_count[Japan-1] PASSED	[57%]
3.py::test_national_user_count[China-0] PASSED	[61%]

3.3. ПАРАМЕТРИЗОВАННЫЙ-ТЕСТ

4. Тестирование исключений:

```
106     # Тестирование исключений (2 штуки)
107 > def test_insufficient_balance_transfer(preloaded_manager):
108     """Проверка исключения при недостаточном балансе"""
109     with pytest.raises(ValueError, match="Insufficient funds"):
110         preloaded_manager.transfer_balance(from_user_id: 2, to_user_id: 1, amount: 5000.0)
111
```

Run Python tests in 3.py



✖ Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

3.py::test_insufficient_balance_transfer PASSED

[66%]

4.1-ТЕСТ-ИСКЛЮЧЕНИЕ

```
113 > def test_transfer_to_invalid_user(preloaded_manager):
114     """Проверка поведения при переводе несуществующему пользователю"""
115     try:
116         preloaded_manager.transfer_balance(from_user_id: 1, to_user_id: 999, amount: 300.0)
117         pytest.fail("Ожидалось исключение для несуществующего пользователя")
118     except Exception:
119         # Ожидаемое поведение из-за SQL уязвимости
120         pass
121
122
```

Run Python tests in 3.py



✖ Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

[71%]

3.py::test_transfer_to_invalid_user FAILED
3.py:112 (test_transfer_to_invalid_user)
preloaded_manager = <auth_manager.AuthManager object at 0x000001D036EB32B0>

```
def test_transfer_to_invalid_user(preloaded_manager):
    """Проверка поведения при переводе несуществующему пользователю"""
    try:
        preloaded_manager.transfer_balance(1, 999, 300.0)
    >       pytest.fail("Ожидалось исключение для несуществующего пользователя")
E       Failed: Ожидалось исключение для несуществующего пользователя
```

3.py:117: Failed

4.2-ТЕСТ-ИСКЛЮЧЕНИЕ

5. Тесты с использованием фикстур базы данных:

```

123 # Тесты с использованием фикстур
124 > def test_user_removal(preloaded_manager):
125     """Тестирование удаления пользователя с использованием фикстур"""
126     # Проверяем существование пользователя
127     user = preloaded_manager.get_user_by_id(1)
128     assert user is not None
129
130     # Удаляем пользователя
131     preloaded_manager.delete_user(1)
132
133     # Проверяем удаление
134     removed_user = preloaded_manager.get_user_by_id(1)
135     assert removed_user is None
136
137

```

Run Python tests in 3.py

Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

3.py::test_user_removal PASSED [76%]

5.1-ТЕСТ-ФИКСТУРА

6. Тесты с метками:

```

138 # Тесты с метками (минимум 2)
139 @pytest.mark.stress
140 > def test_high_volume_registrations(manager):
141     """Стресс-тест массовой регистрации пользователей"""
142     for i in range(200):
143         manager.register_user(username=f"load_user_{i}", password=f"pass_{i}", country=f"nation_{i % 15}", 1000 + i)
144
145     total = manager.count_users_by_country("nation_0")
146     assert total >= 13 # 200 пользователей / 15 стран
147

```

Run Python tests in 3.py

Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms

3.py::test_high_volume_registrations PASSED [80%]

6.1-ТЕСТ-МЕТКА

```

149 @pytest.mark.vulnerability
150 > def test_sql_injection_in_registration(manager):
151     """Тестирование уязвимости SQL инъекции при регистрации"""
152     manager.register_user(username="hacker'; DROP TABLE users; --", password="attack", country="USA", balance=1000)
153
154     cursor = manager.connection.cursor()
155     cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users';")
156     tables = cursor.fetchall()
157
158     assert len(tables) == 1 # Таблица должна сохраниться
159
160
Run Python tests in 3.py ×
⚙️ ⚙️ ⚙️ | ✅ ⚙️ | ⚙️ ⚙️ | ⚙️ | ⚙️ : 
✖️ Tests failed: 2, passed: 18, ignored: 1 of 21 tests – 3 ms
3.py::test_sql_injection_in_registration FAILED [ 85%]

```

```

3.py::test_sql_injection_in_registration FAILED [ 85%]
3.py:148 (test_sql_injection_in_registration)
manager = <auth_manager.AuthManager object at 0x000001D036F96620>

    @pytest.mark.vulnerability
    def test_sql_injection_in_registration(manager):
        """Тестирование уязвимости SQL инъекции при регистрации"""
>       manager.register_user("hacker'; DROP TABLE users; --", "attack", "USA", 1000)

3.py:152:
-----
self = <auth_manager.AuthManager object at 0x000001D036F96620>
username = "hacker'; DROP TABLE users; --", password = 'attack', country = 'USA'
balance = 1000

    def register_user(self, username, password, country, balance):
        with self.connection:
>            self.connection.execute(f"""
            INSERT INTO users (username, password, country, balance)
            VALUES ('{username}', '{password}', '{country}', {balance})
        """)
E             sqlite3.OperationalError: near ";": syntax error
auth_manager.py:22: OperationalError

```

6.2-TECT-METKA

7. Тестирование различных векторов SQL-инъекций:

```
161     @pytest.mark.vulnerability
162     def test_sql_injection_in_authentication(manager):
163         """Тестирование уязвимости SQL инъекции при аутентификации"""
164         manager.register_user(username="legit_user", password="safe_pass", country="UK", balance=2000.0)
165
166         # SQL инъекция для обхода аутентификации
167         user = manager.authenticate_user(username="legit_user' OR '1'='1", password="any_password")
168         assert user is not None # Инъекция должна сработать
169
170
```

Run Python tests in 3.py ×



✖ Tests failed: 2, passed: 18, ignored: 1 of 21 tests - 3 ms

3.py::test_sql_injection_in_authentication PASSED [90%]

7.1-TECT-SQL-ИНЪЕКЦИЙ