Практическая 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).

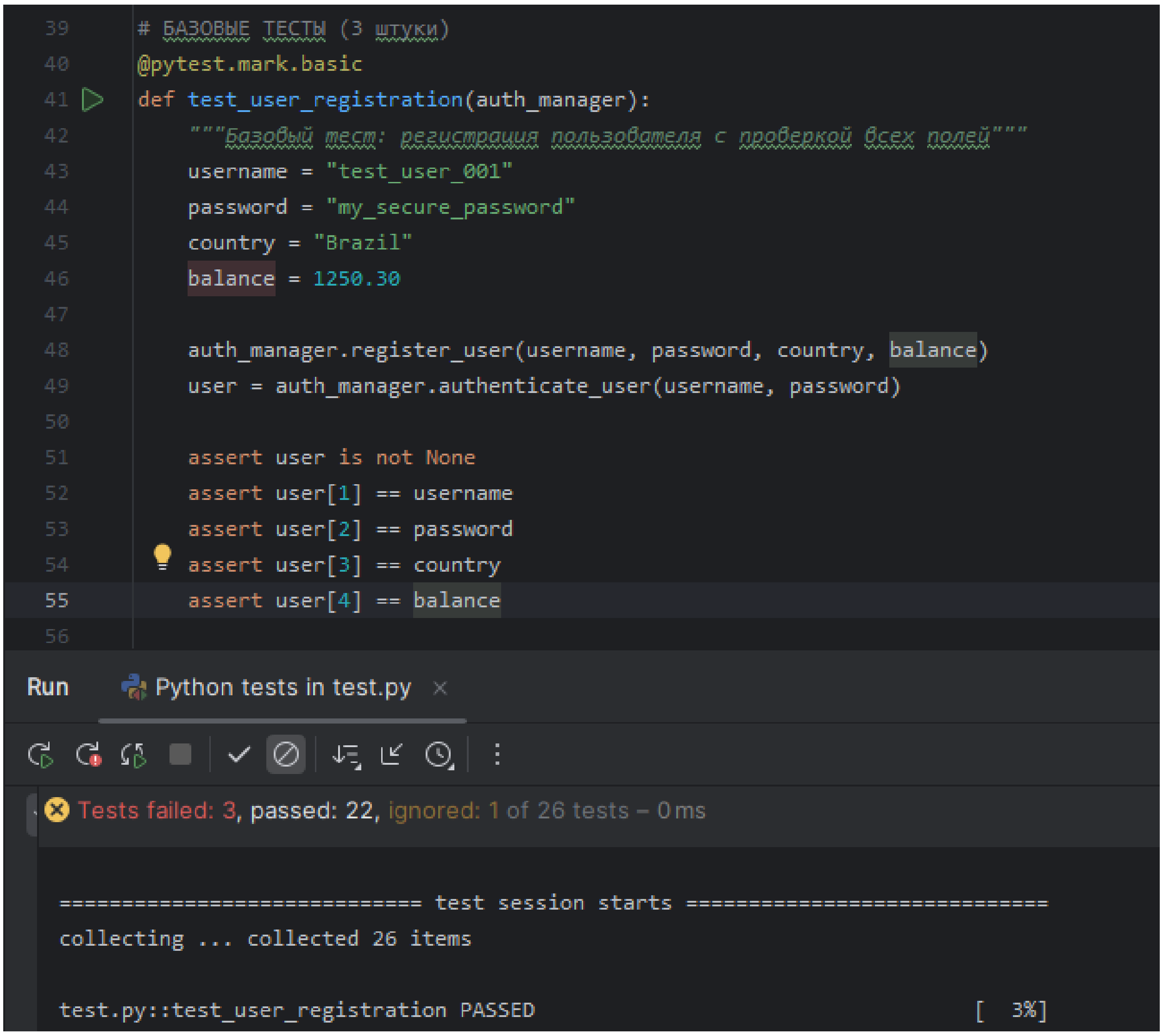
Код класса для тестирования:

```
import sqlite3


class AuthManager:
    def __init__(self, connection):
        self.connection = connection
        self.create_tables()


    def create_tables(self):
        """Создание таблицы пользователей"""
        with self.connection:
            self.connection.execute("""
                CREATE TABLE IF NOT EXISTS users (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT NOT NULL UNIQUE,
                    password TEXT NOT NULL,
                    country TEXT NOT NULL,
                    balance REAL NOT NULL
                )
        """)

def register_user(self, username, password, country, balance):
    """Регистрация нового пользователя"""
    with self.connection:
        self.connection.execute(f"""
            INSERT INTO users (username, password, country, balance)
            VALUES ('{username}', '{password}', '{country}', {balance})
        """)

def authenticate_user(self, username, password):
    """Аутентификация пользователя"""
    cursor = self.connection.cursor()
    cursor.execute(f"""
        SELECT * FROM users
        WHERE username = '{username}' AND password = '{password}'
    """)
    return cursor.fetchone()

def delete_user(self, user_id):
    """Удаление пользователя по ID"""
    with self.connection:
        self.connection.execute(f"""
            DELETE FROM users WHERE id = {user_id}
        """)

def get_user_by_id(self, user_id):
    """Получение пользователя по ID"""
    cursor = self.connection.cursor()
```

```
        cursor.execute(f"""
            SELECT * FROM users WHERE id = {user_id}
        """)
        return cursor.fetchone()

    def count_users_by_country(self, country):
        """Подсчет пользователей по стране"""
        cursor = self.connection.cursor()
        cursor.execute(f"""
            SELECT COUNT(*) FROM users WHERE country = '{country}'
        """)
        return cursor.fetchone()[0]

    def transfer_balance(self, from_user_id, to_user_id, amount):
        """Перевод средств между пользователями"""
        with self.connection:
            # Проверяем достаточность средств
            cursor = self.connection.cursor()
            cursor.execute(f"SELECT balance FROM users WHERE id =
{from_user_id}")
            from_balance = cursor.fetchone()[0]

            if from_balance < amount:
                raise ValueError("Недостаточно средств для перевода")

            # Выполняем перевод
            self.connection.execute(f"""
                UPDATE users SET balance = balance - {amount} WHERE id =
{from_user_id}
```

```
            """)
            self.connection.execute(f"""
                UPDATE users SET balance = balance + {amount} WHERE id = 
{to_user_id}
            """)

    def get_all_users(self):
        """Получение всех пользователей (для тестирования)"""
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM users")
        return cursor.fetchall()

    def update_user_balance(self, user_id, new_balance):
        """Обновление баланса пользователя"""
        with self.connection:
            self.connection.execute(f"""
                UPDATE users SET balance = {new_balance} WHERE id = {user_id}
            """)
```

1.Базовые тесты

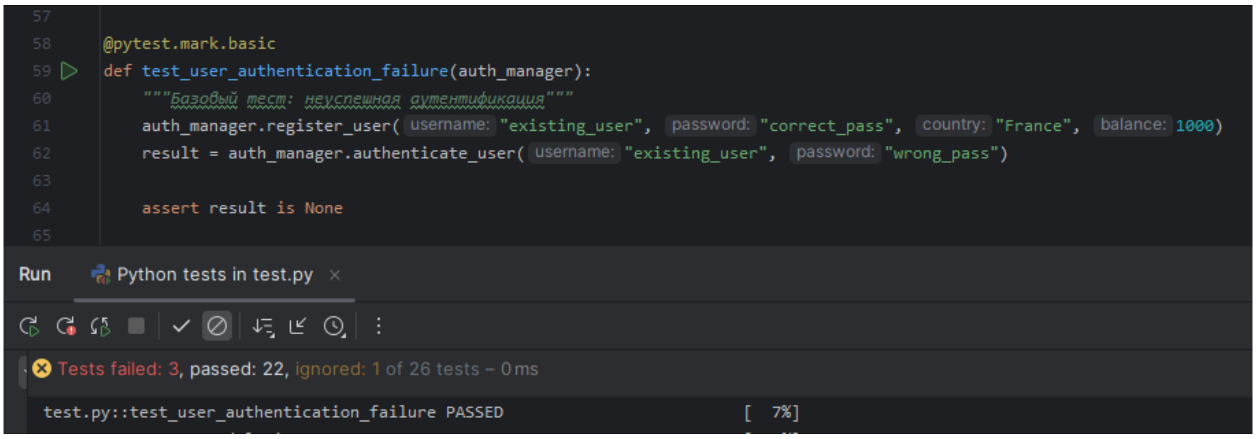


```python
# БАЗОВЫЕ ТЕСТЫ (3 штуки)
@pytest.mark.basic
def test_user_registration(auth_manager):
    """Базовый тест: регистрация пользователя с проверкой всех полей"""
    username = "test_user_001"
    password = "my_secure_password"
    country = "Brazil"
    balance = 1250.30

    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[2] == password
    assert user[3] == country
    assert user[4] == balance
```

Run   Python tests in test.py ×

```
Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms


============================= test session starts ==============================
collecting ... collected 26 items

test.py::test_user_registration PASSED                                   [  3%]
```

1

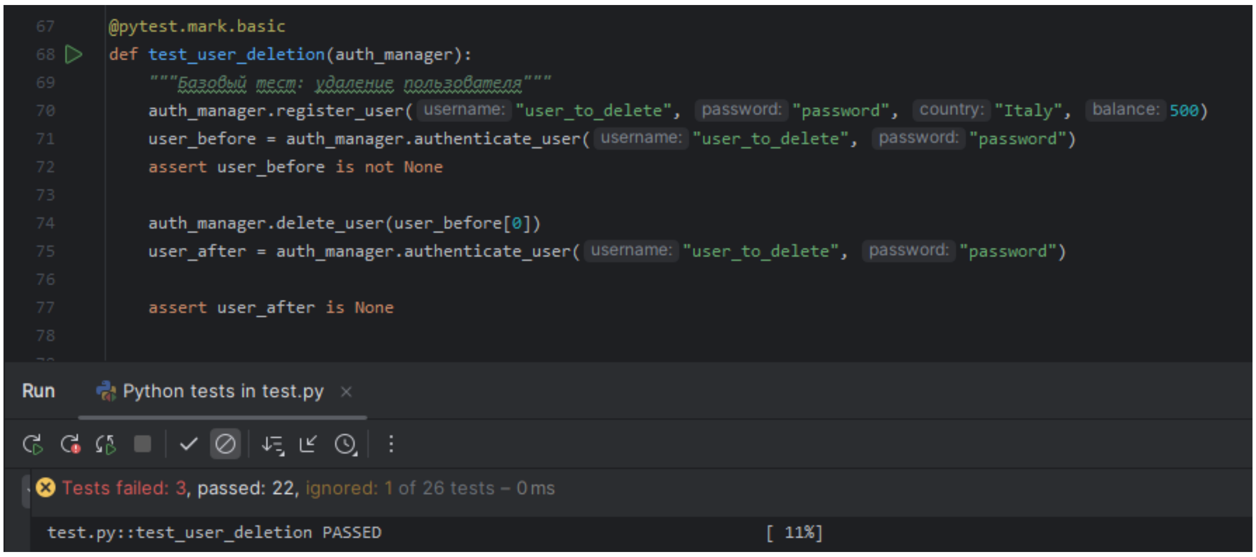


```python
@pytest.mark.basic
def test_user_authentication_failure(auth_manager):
    """Базовый тест: неуспешная аутентификация"""
    auth_manager.register_user( username: "existing_user",  password: "correct_pass",  country: "France",  balance: 1000)
    result = auth_manager.authenticate_user( username: "existing_user",  password: "wrong_pass")

    assert result is None
```

Run   Python tests in test.py ×

```
Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms

test.py::test_user_authentication_failure PASSED                         [  7%]
```

2

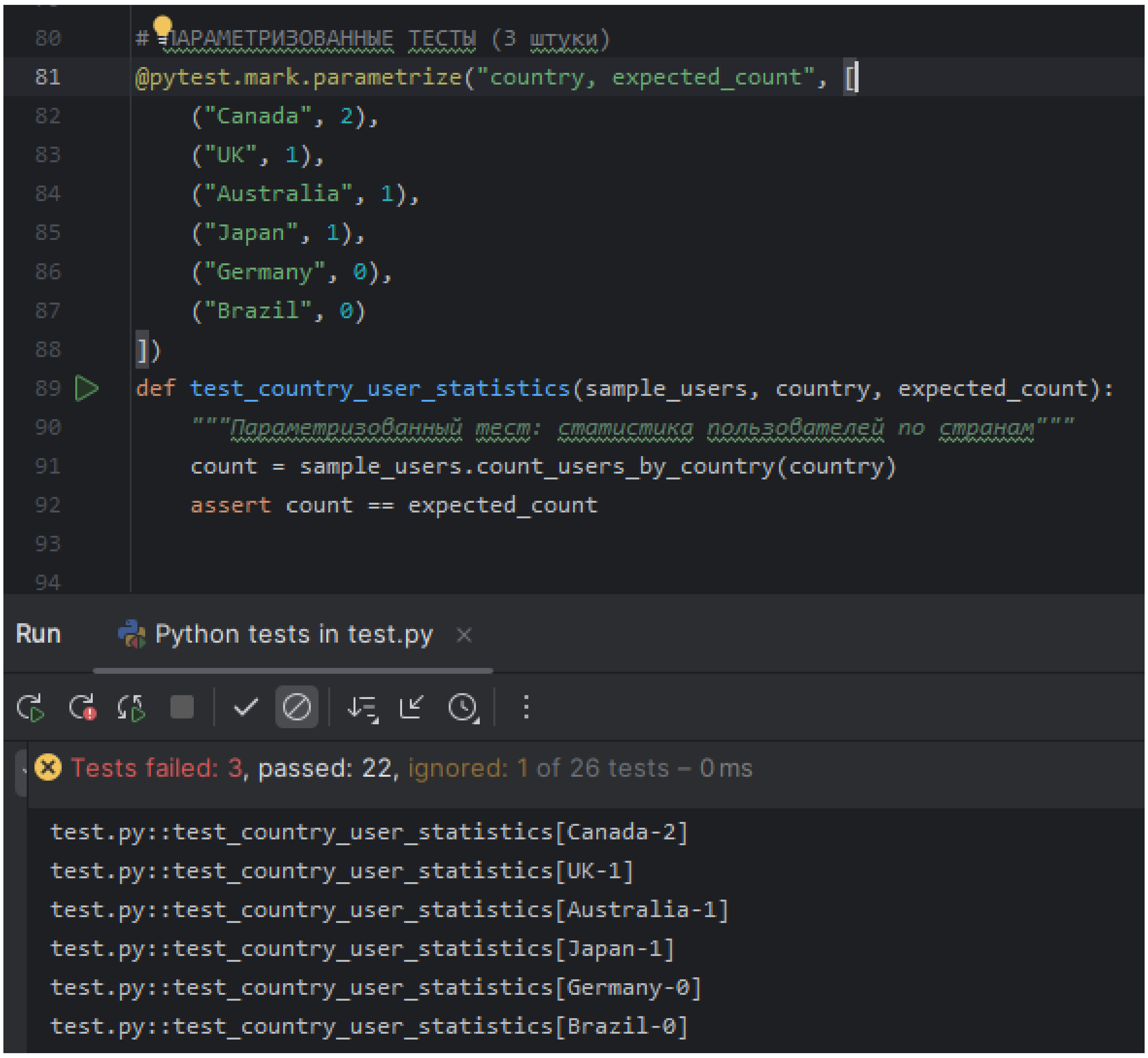


```
@pytest.mark.basic
def test_user_deletion(auth_manager):
    """Базовый тест: удаление пользователя"""
    auth_manager.register_user( username: "user_to_delete",  password: "password",  country: "Italy",  balance: 500)
    user_before = auth_manager.authenticate_user( username: "user_to_delete",  password: "password")
    assert user_before is not None

    auth_manager.delete_user(user_before[0])
    user_after = auth_manager.authenticate_user( username: "user_to_delete",  password: "password")

    assert user_after is None

```

```
Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms
test.py::test_user_deletion PASSED                                        [ 11%]
```

3

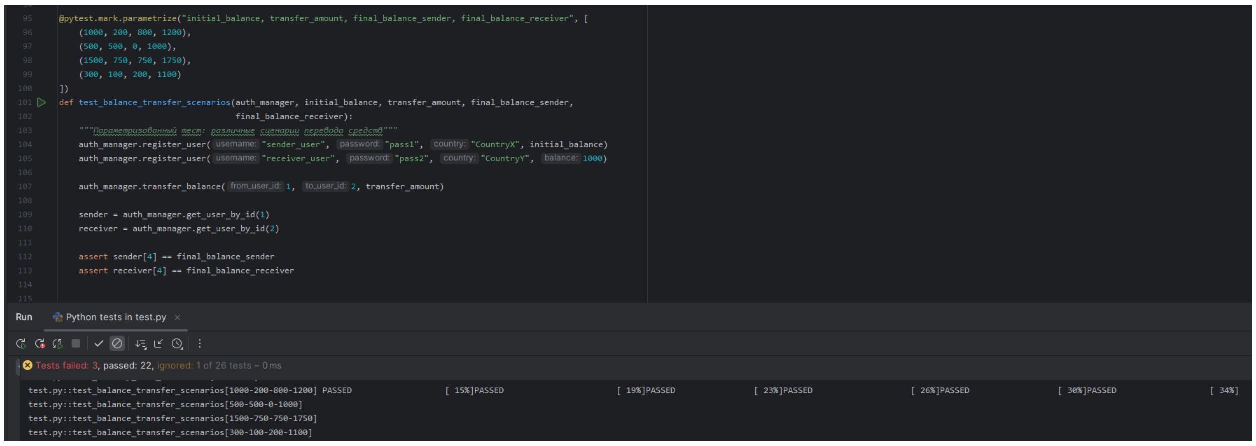
2.Параметризованные тесты



```python
# ПАРАМЕТРИЗОВАННЫЕ ТЕСТЫ (3 штуки)
@pytest.mark.parametrize("country, expected_count", [
    ("Canada", 2),
    ("UK", 1),
    ("Australia", 1),
    ("Japan", 1),
    ("Germany", 0),
    ("Brazil", 0)
])
def test_country_user_statistics(sample_users, country, expected_count):
    """Параметризованный тест: статистика пользователей по странам"""
    count = sample_users.count_users_by_country(country)
    assert count == expected_count
```

```
Run    Python tests in test.py ×

Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms

test.py::test_country_user_statistics[Canada-2]
test.py::test_country_user_statistics[UK-1]
test.py::test_country_user_statistics[Australia-1]
test.py::test_country_user_statistics[Japan-1]
test.py::test_country_user_statistics[Germany-0]
test.py::test_country_user_statistics[Brazil-0]
```

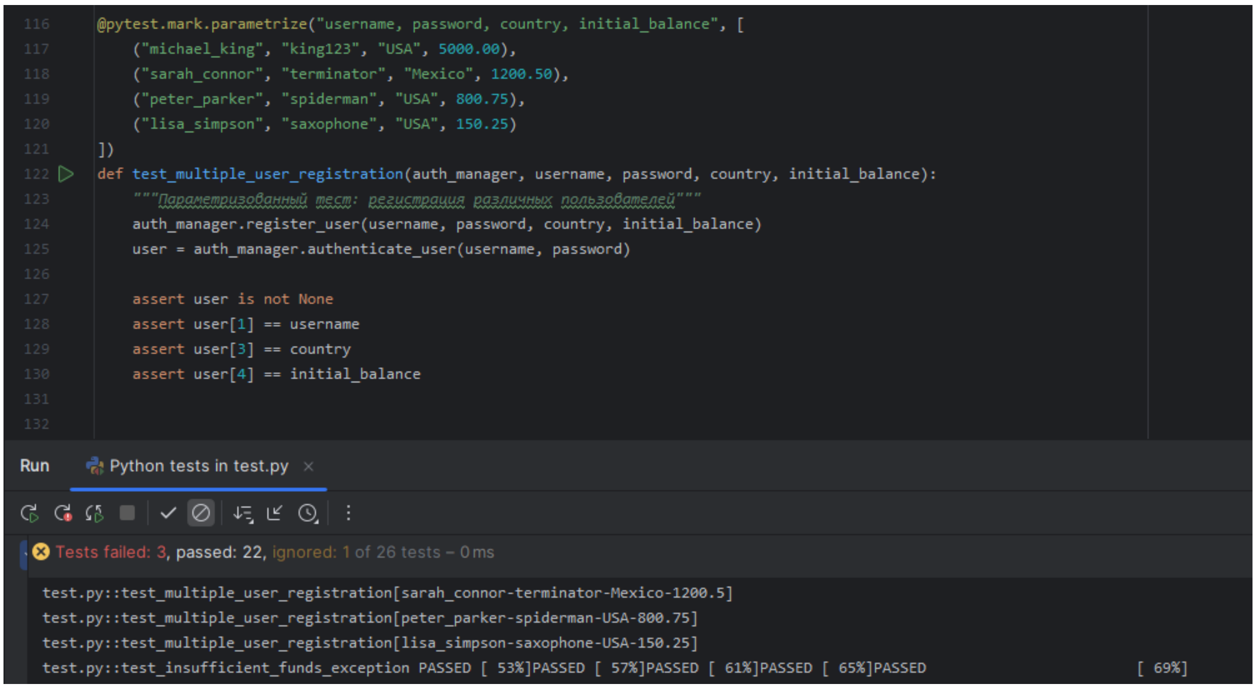
1



```python
@pytest.mark.parametrize("initial_balance, transfer_amount, final_balance_sender, final_balance_receiver", [
    (1000, 200, 800, 1200),
    (500, 500, 0, 1000),
    (1500, 750, 750, 1750),
    (300, 100, 200, 1100)
])
def test_balance_transfer_scenarios(auth_manager, initial_balance, transfer_amount, final_balance_sender,
                                    final_balance_receiver):
    """Параметризованный тест: различные сценарии перевода средств"""
    auth_manager.register_user( username: "sender_user",  password: "pass1",  country: "CountryX", initial_balance)
    auth_manager.register_user( username: "receiver_user",  password: "pass2",  country: "CountryY",  balance: 1000)

    auth_manager.transfer_balance( from_user_id: 1,  to_user_id: 2, transfer_amount)

    sender = auth_manager.get_user_by_id(1)
    receiver = auth_manager.get_user_by_id(2)

    assert sender[4] == final_balance_sender
    assert receiver[4] == final_balance_receiver
```

```
Run    Python tests in test.py ×

Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms

test.py::test_balance_transfer_scenarios[1000-200-800-1200] PASSED            [ 15%]PASSED            [ 19%]PASSED        [ 23%]PASSED          [ 26%]PASSED         [ 30%]PASSED         [ 34%]
test.py::test_balance_transfer_scenarios[500-500-0-1000]
test.py::test_balance_transfer_scenarios[1500-750-750-1750]
test.py::test_balance_transfer_scenarios[300-100-200-1100]
```

2



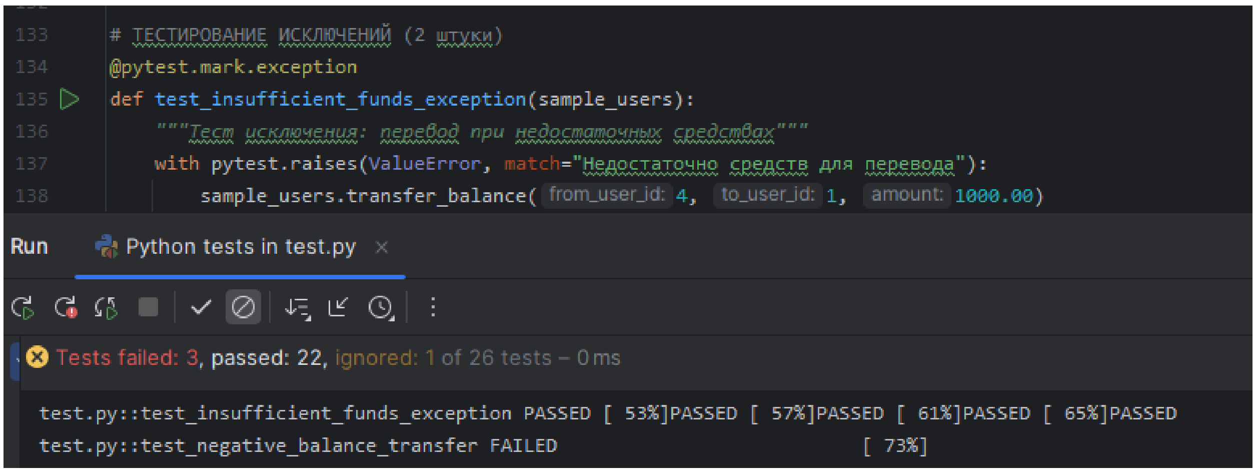
```
@pytest.mark.parametrize("username, password, country, initial_balance", [
    ("michael_king", "king123", "USA", 5000.00),
    ("sarah_connor", "terminator", "Mexico", 1200.50),
    ("peter_parker", "spiderman", "USA", 800.75),
    ("lisa_simpson", "saxophone", "USA", 150.25)
])
def test_multiple_user_registration(auth_manager, username, password, country, initial_balance):
    """Параметризованный тест: регистрация различных пользователей"""
    auth_manager.register_user(username, password, country, initial_balance)
    user = auth_manager.authenticate_user(username, password)

    assert user is not None
    assert user[1] == username
    assert user[3] == country
    assert user[4] == initial_balance
```

```
Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms
```

```
test.py::test_multiple_user_registration[sarah_connor-terminator-Mexico-1200.5]
test.py::test_multiple_user_registration[peter_parker-spiderman-USA-800.75]
test.py::test_multiple_user_registration[lisa_simpson-saxophone-USA-150.25]
test.py::test_insufficient_funds_exception PASSED [ 53%]PASSED [ 57%]PASSED [ 61%]PASSED [ 65%]PASSED                        [ 69%]
```

3

3.Тестирование исключений

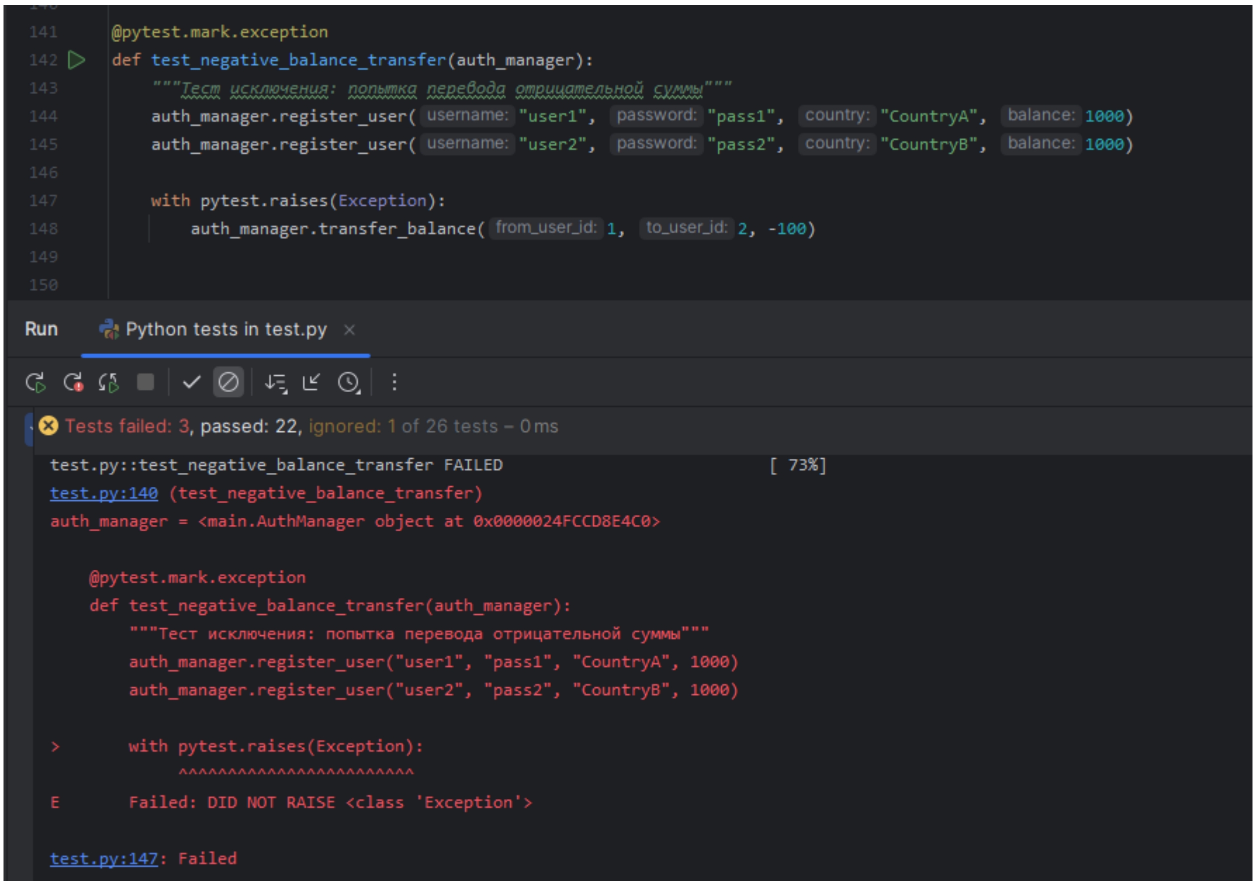


```
# ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ (2 штуки)
@pytest.mark.exception
def test_insufficient_funds_exception(sample_users):
    """Тест исключения: перевод при недостаточных средствах"""
    with pytest.raises(ValueError, match="Недостаточно средств для перевода"):
        sample_users.transfer_balance( from_user_id: 4,  to_user_id: 1,  amount: 1000.00)
```

```
Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms
```

```
test.py::test_insufficient_funds_exception PASSED [ 53%]PASSED [ 57%]PASSED [ 61%]PASSED [ 65%]PASSED
test.py::test_negative_balance_transfer FAILED                                        [ 73%]
```

1

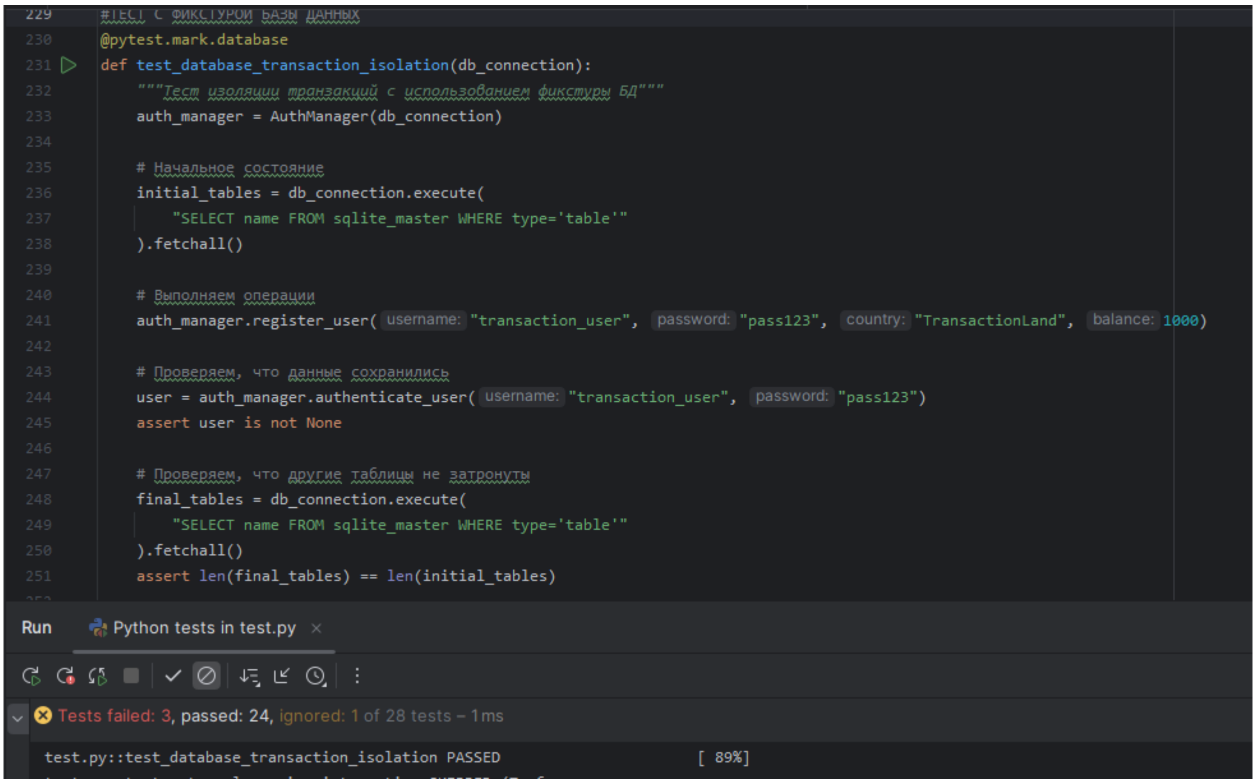


```
@pytest.mark.exception
def test_negative_balance_transfer(auth_manager):
    """Тест исключения: попытка перевода отрицательной суммы"""
    auth_manager.register_user( username: "user1",  password: "pass1",  country: "CountryA",  balance: 1000)
    auth_manager.register_user( username: "user2",  password: "pass2",  country: "CountryB",  balance: 1000)

    with pytest.raises(Exception):
        auth_manager.transfer_balance( from_user_id: 1,  to_user_id: 2, -100)
```

```
Tests failed: 3, passed: 22, ignored: 1 of 26 tests – 0 ms
test.py::test_negative_balance_transfer FAILED                           [ 73%]
test.py:140 (test_negative_balance_transfer)
auth_manager = <main.AuthManager object at 0x0000024FCCD8E4C0>

    @pytest.mark.exception
    def test_negative_balance_transfer(auth_manager):
        """Тест исключения: попытка перевода отрицательной суммы"""
        auth_manager.register_user("user1", "pass1", "CountryA", 1000)
        auth_manager.register_user("user2", "pass2", "CountryB", 1000)

>       with pytest.raises(Exception):
             ^^^^^^^^^^^^^^^^^^^^^^^^
E       Failed: DID NOT RAISE <class 'Exception'>

test.py:147: Failed
```

2

4.Тесты с использованием фикстур базы данных:

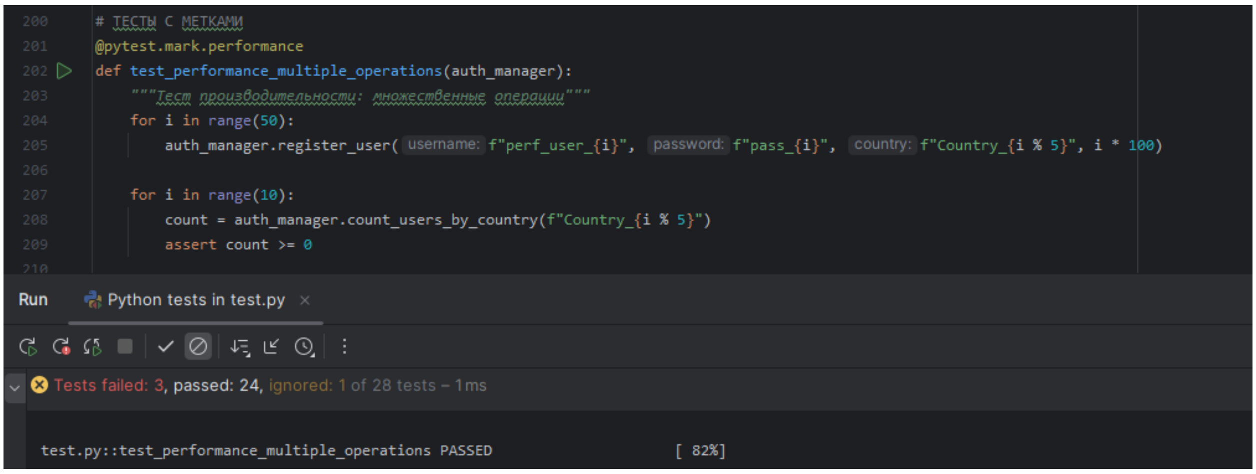


```
#ТЕСТ С ФИКСТУРОЙ БАЗЫ ДАННЫХ
@pytest.mark.database
def test_database_transaction_isolation(db_connection):
    """Тест изоляции транзакций с использованием фикстуры БД"""
    auth_manager = AuthManager(db_connection)

    # Начальное состояние
    initial_tables = db_connection.execute(
        "SELECT name FROM sqlite_master WHERE type='table'"
    ).fetchall()

    # Выполняем операции
    auth_manager.register_user( username: "transaction_user",  password: "pass123",  country: "TransactionLand",  balance: 1000)

    # Проверяем, что данные сохранились
    user = auth_manager.authenticate_user( username: "transaction_user",  password: "pass123")
    assert user is not None

    # Проверяем, что другие таблицы не затронуты
    final_tables = db_connection.execute(
        "SELECT name FROM sqlite_master WHERE type='table'"
    ).fetchall()
    assert len(final_tables) == len(initial_tables)
```

```
Tests failed: 3, passed: 24, ignored: 1 of 28 tests – 1 ms

test.py::test_database_transaction_isolation PASSED                    [ 89%]
```

1

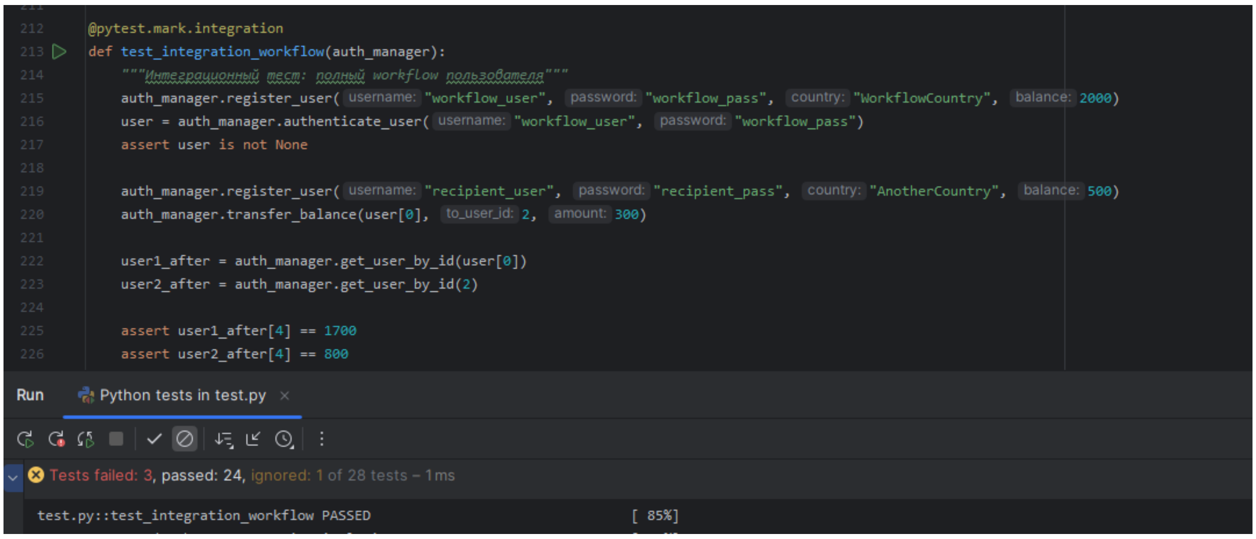
5.Тесты с метками



```
# ТЕСТЫ С МЕТКАМИ
@pytest.mark.performance
def test_performance_multiple_operations(auth_manager):
    """Тест производительности: множественные операции"""
    for i in range(50):
        auth_manager.register_user( username: f"perf_user_{i}",  password: f"pass_{i}",  country: f"Country_{i % 5}", i * 100)

    for i in range(10):
        count = auth_manager.count_users_by_country(f"Country_{i % 5}")
        assert count >= 0
```

```
Tests failed: 3, passed: 24, ignored: 1 of 28 tests – 1 ms

test.py::test_performance_multiple_operations PASSED                    [ 82%]
```

1



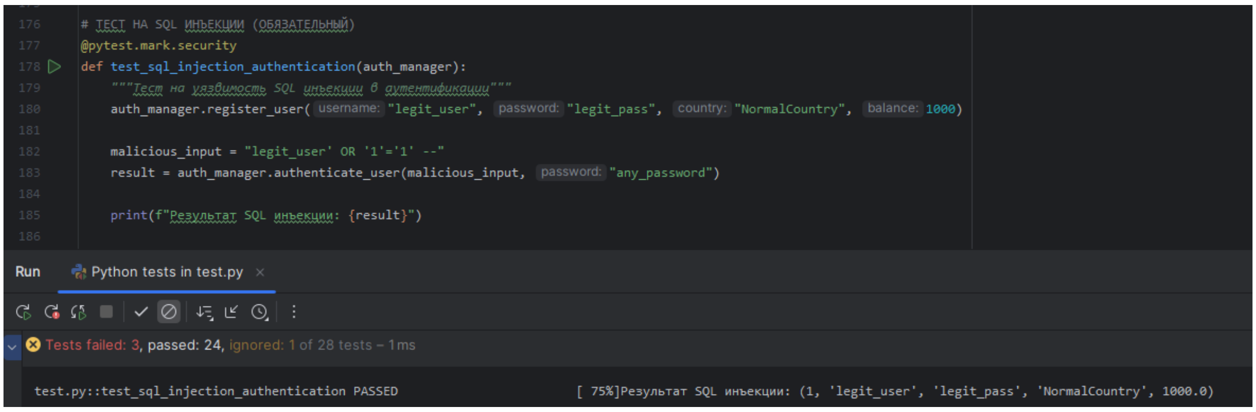
```
@pytest.mark.integration
def test_integration_workflow(auth_manager):
    """Интеграционный тест: полный workflow пользователя"""
    auth_manager.register_user( username: "workflow_user", password: "workflow_pass", country: "WorkflowCountry", balance: 2000)
    user = auth_manager.authenticate_user( username: "workflow_user", password: "workflow_pass")
    assert user is not None

    auth_manager.register_user( username: "recipient_user", password: "recipient_pass", country: "AnotherCountry", balance: 500)
    auth_manager.transfer_balance(user[0], to_user_id: 2, amount: 300)

    user1_after = auth_manager.get_user_by_id(user[0])
    user2_after = auth_manager.get_user_by_id(2)

    assert user1_after[4] == 1700
    assert user2_after[4] == 800
```

Run Python tests in test.py

```
Tests failed: 3, passed: 24, ignored: 1 of 28 tests – 1 ms

test.py::test_integration_workflow PASSED                    [ 85%]
```

2

6.Тестирование различных векторов SQL-инъекций

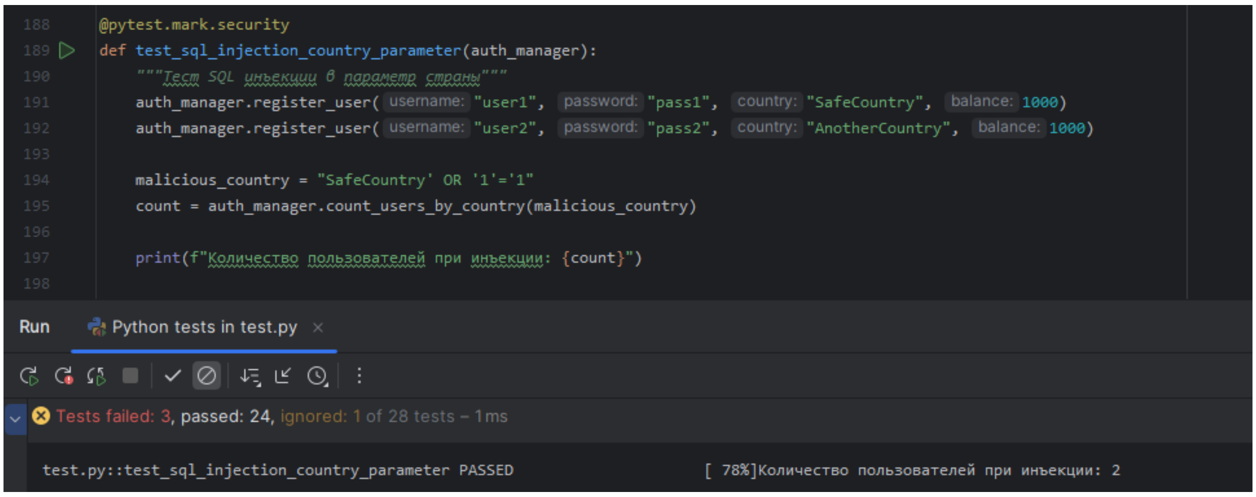


```
# ТЕСТ НА SQL ИНЪЕКЦИИ (ОБЯЗАТЕЛЬНЫЙ)
@pytest.mark.security
def test_sql_injection_authentication(auth_manager):
    """Тест на уязвимость SQL инъекции в аутентификации"""
    auth_manager.register_user( username: "legit_user", password: "legit_pass", country: "NormalCountry", balance: 1000)

    malicious_input = "legit_user' OR '1'='1' --"
    result = auth_manager.authenticate_user(malicious_input, password: "any_password")

    print(f"Результат SQL инъекции: {result}")
```

Run Python tests in test.py

```
Tests failed: 3, passed: 24, ignored: 1 of 28 tests – 1 ms

test.py::test_sql_injection_authentication PASSED          [ 75%]Результат SQL инъекции: (1, 'legit_user', 'legit_pass', 'NormalCountry', 1000.0)
```

1



```
@pytest.mark.security
def test_sql_injection_country_parameter(auth_manager):
    """Тест SQL инъекции в параметр страны"""
    auth_manager.register_user( username: "user1", password: "pass1", country: "SafeCountry", balance: 1000)
    auth_manager.register_user( username: "user2", password: "pass2", country: "AnotherCountry", balance: 1000)

    malicious_country = "SafeCountry' OR '1'='1"
    count = auth_manager.count_users_by_country(malicious_country)

    print(f"Количество пользователей при инъекции: {count}")
```

Run Python tests in test.py

```
Tests failed: 3, passed: 24, ignored: 1 of 28 tests – 1 ms

test.py::test_sql_injection_country_parameter PASSED          [ 78%]Количество пользователей при инъекции: 2
```

2