1. Класс для тестирования:

auth_manager.py

```python
class AuthManager:  10 usages
    def __init__(self, connection):
        self.connection = connection
        self.create_tables()

    def create_tables(self):  1 usage
        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):  25 usages
        with self.connection:
            self.connection.execute("""
            INSERT INTO users (username, password, country, balance)
            VALUES ('{}', '{}', '{}', {})
            """.format( *args: username, password, country, balance))

    def authenticate_user(self, username, password):  10 usages
        cursor = self.connection.cursor()
        cursor.execute("""
        SELECT * FROM users
        WHERE username = '{}' AND password = '{}'
        """.format( *args: username, password))
        return cursor.fetchone()

    def delete_user(self, user_id):  1 usage
        with self.connection:
            self.connection.execute("""
            DELETE FROM users WHERE id = {}
            """.format(user_id))

    def get_user_by_id(self, user_id):  14 usages
        cursor = self.connection.cursor()
        cursor.execute("""
```

auth_manager.py

```
1   class AuthManager:  10 usages
33      def delete_user(self, user_id):  1 usage
36              DELETE FROM users WHERE id = {}
37              """.format(user_id))
38
39      def get_user_by_id(self, user_id):  14 usages
40          cursor = self.connection.cursor()
41          cursor.execute("""
42          SELECT * FROM users WHERE id = {}
43          """.format(user_id))
44          return cursor.fetchone()
45
46      def count_users_by_country(self, country):  1 usage
47          cursor = self.connection.cursor()
48          cursor.execute("""
49          SELECT COUNT(*) FROM users WHERE country = '{}'
50          """.format(country))
51          return cursor.fetchone()[0]
52
53      def transfer_balance(self, from_user_id, to_user_id, amount):  3 usages
54          with self.connection:
55              # Проверяем, достаточно ли средств
56              cursor = self.connection.cursor()
57              cursor.execute("SELECT balance FROM users WHERE id ={}".format(from_user_id))
58              from_balance = cursor.fetchone()[0]
59
60              if from_balance < amount:
61                  raise ValueError("Insufficient funds")
62              # Выполняем перевод
63              self.connection.execute("""
64              UPDATE users SET balance = balance - {} WHERE id = {}
65              """.format( *args: amount, from_user_id))
66              self.connection.execute("""
67              UPDATE users SET balance = balance + {} WHERE id = {}
68              """.format( *args: amount, to_user_id))
69
```

2. Файл базовых тестов:

test_base.py

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


@pytest.fixture  2 usages
def db():
    connection = sqlite3.connect(":memory:")
    yield connection
    connection.close()


@pytest.fixture  21 usages
def auth_manager(db):
    return AuthManager(db)


def test_register_user(auth_manager):
    auth_manager.register_user( username: "test_user",  password: "password123",  country: "Russia",  balance: 100)
    user = auth_manager.get_user_by_id(1)
    assert user is not None, "Пользователь должен быть зарегистрирован"
    assert user[1] == "test_user", "Имя пользователя должно совпадать"
    assert user[2] == "password123", "Пароль пользователя должен совпадать"
    assert user[3] == "Russia", "Страна пользователя должна совпадать"
    assert user[4] == 100, "Баланс должен быть корректным"


def test_authenticate_user(auth_manager):
    auth_manager.register_user( username: "test_user",  password: "password123",  country: "Russia",  balance: 100)
    user = auth_manager.authenticate_user( username: "test_user",  password: "password123")
    assert user is not None, "Аутентификация должна быть успешной"
    assert user[1] == "test_user", "Имя пользователя должно совпадать"
    assert user[2] == "password123", "Пароль должен совпадать"


def test_get_user_by_id_existing_user(auth_manager):
    """Тест получения существующего пользователя"""
    auth_manager.register_user( username: "test_user",  password: "password123",  country: "Russia",  balance: 100)
    user = auth_manager.get_user_by_id(1)
    assert user is not None
    assert user[0] == 1  # ID
```

test_base.py

```python

def test_get_user_by_id_existing_user(auth_manager):
    """Тест получения существующего пользователя"""
    auth_manager.register_user( username: "test_user",  password: "password123",  country: "Russia",  balance: 100)
    user = auth_manager.get_user_by_id(1)
    assert user is not None
    assert user[0] == 1  # ID


def test_get_user_by_id_non_existing_user(auth_manager):
    """Тест получения несуществующего пользователя"""
    user = auth_manager.get_user_by_id(999)
    assert user is None


def test_get_user_by_id_invalid_input(auth_manager):
    """Тест с некорректным входным параметром"""
    with pytest.raises(Exception):
        auth_manager.get_user_by_id('invalid_id')


def test_transfer_balance(auth_manager):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 100)
    auth_manager.register_user( username: "user2",  password: "password123",  country: "CountryB",  balance: 500)
    user1 = auth_manager.authenticate_user( username: "user1",  password: "password123")
    user2 = auth_manager.authenticate_user( username: "user2",  password: "password123")
    try:
        auth_manager.transfer_balance(user1[0], user2[0],  amount: 200)
    except ValueError as e:
        assert str(
            e) == "Insufficient funds", "Exception message should be 'Insufficient funds'"
        update_user1 = auth_manager.get_user_by_id(user1[0])
        update_user2 = auth_manager.get_user_by_id(user2[0])
        assert update_user1[4] == 100
        assert update_user2[4] == 500


```

2.1. Удачные тесты:

```
(.venv) PS C:\...\практика_3> pytest test_base.py
============================= test session starts =============================
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\...\практика_3
configfile: pytest.ini
collected 6 items

test_base.py ......                                                      [100%]

============================== 6 passed in 0.06s ==============================
(.venv) PS C:\...\практика_3>
```

2.2. Неудачные тесты:

```
============================= test session starts =============================
platform win32 -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\...\практика_3
configfile: pytest.ini
collected 6 items

test_base.py F.....                                                      [100%]

================================== FAILURES ===================================
_____________________________ test_register_user ______________________________

auth_manager = <auth_manager.AuthManager object at 0x00000246ADBFB890>

    def test_register_user(auth_manager):
        auth_manager.register_user("test_user", "password123", "Russia", -100)
        user = auth_manager.get_user_by_id(1)
        assert user is not None, "Пользователь должен быть зарегистрирован"
        assert user[1] == "test_user", "Имя пользователя должно совпадать"
        assert user[2] == "password123", "Пароль пользователя должен совпадать"
        assert user[3] == "Russia", "Страна пользователя должна совпадать"
>       assert user[4] == 100, "Баланс должен быть корректным"
E       AssertionError: Баланс должен быть корректным
E       assert -100.0 == 100

test_base.py:25: AssertionError
=========================== short test summary info ===========================
FAILED test_base.py::test_register_user - AssertionError: Баланс должен быть корректным
========================= 1 failed, 5 passed in 0.29s =========================
```

3. Файл параметризованных тестов:

test_param.py

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


@pytest.fixture  2 usages
def db():
    connection = sqlite3.connect(":memory:")
    yield connection
    connection.close()


@pytest.fixture  28 usages
def auth_manager(db):
    return AuthManager(db)


@pytest.mark.parametrize( "username, password, country, balance",
                          [ ("user1", "pass1", "CountryA", 100),
                            ("user2", "pass2", "CountryB", 200),
                            ("user3", "pass3", "CountryC", 300), ] )
def test_register_user_param(auth_manager, username, password, country, balance):
    auth_manager.register_user(username, password, country, balance)
    user = auth_manager.authenticate_user(username, password)
    assert user is not None, f"Пользователь {username} должен быть зарегистрирован"
    assert user[1] == username, f"Имя пользователя должно быть {username}"
    assert user[3] == country, f"Страна пользователя должна быть {country}"
    assert user[4] == balance, f"Баланс пользователя должен быть {balance}"


@pytest.mark.parametrize( "username, password, expected_result", [
                        ("user1", "password123", True),
                        ("user2", "wrongpassword", False),
                        ("nonexistent", "password123", False),] )
def test_authenticate_user_param(auth_manager, username, password, expected_result):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 100)
    auth_manager.register_user( username: "user2",  password: "password456",  country: "CountryB",  balance: 200)
    user = auth_manager.authenticate_user(username, password)
    if expected_result:
        assert user is not None, f"Пользователь {username} должен быть аутентифицирован"
```

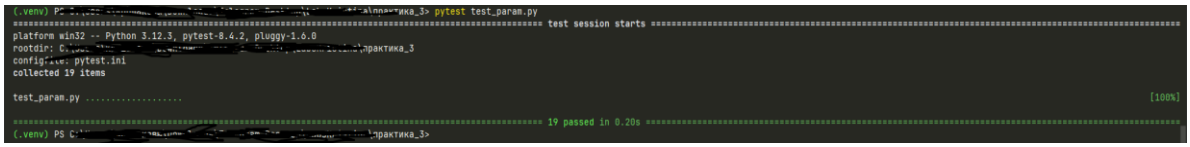
test_param.py

```python
def test_authenticate_user_param(auth_manager, username, password, expected_result):
    user = auth_manager.authenticate_user(username, password)
    if expected_result:
        assert user is not None, f"Пользователь {username} должен быть аутентифицирован"
    else:
        assert user is None, f"Пользователь {username} не должен быть аутентифицирован"


@pytest.mark.parametrize( "from_balance, to_balance, transfer_amount, expected_from_balance, expected_to_balance",
                          [ (100, 50, 50, 50, 100),
                            (200, 100, 150, 50, 250),
                            (100, 100, 200, 100, 100),] )
def test_transfer_balance_param(auth_manager, from_balance, to_balance, transfer_amount, expected_from_balance, expected_to_balance):
    # Регистрация пользователей
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA", from_balance)
    auth_manager.register_user( username: "user2",  password: "password123",  country: "CountryA", to_balance)
    from_user_id = auth_manager.authenticate_user( username: "user1",  password: "password123")[0]
    to_user_id = auth_manager.authenticate_user( username: "user2",  password: "password123")[0]
    if from_balance >= transfer_amount:
        auth_manager.transfer_balance(from_user_id, to_user_id, transfer_amount)
        from_user = auth_manager.get_user_by_id(from_user_id)
        to_user = auth_manager.get_user_by_id(to_user_id)
        assert from_user[4] == expected_from_balance, f"У отправителя должно остаться {expected_from_balance} единиц"
        assert to_user[4] == expected_to_balance, f"У получателя должно быть {expected_to_balance} единиц"


@pytest.mark.parametrize("country, expected_count", [("CountryA", 2), ("CountryB", 1), ("CountryC", 0)])
def test_count_users_by_country(auth_manager, country, expected_count):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 1000)
    auth_manager.register_user( username: "user2",  password: "password123",  country: "CountryA",  balance: 1000)
    auth_manager.register_user( username: "user3",  password: "password123",  country: "CountryB",  balance: 1000)
    count = auth_manager.count_users_by_country(country)
    assert count == expected_count


@pytest.mark.parametrize("user_id, expected", [
    (1, (1, "user1", "password123", "CountryA", 1000)),
    (2, (2, "user2", "password123", "CountryB", 1000)),
    (3, None), # Несуществующий пользователь
    (0, None), # Несуществующий пользователь
```

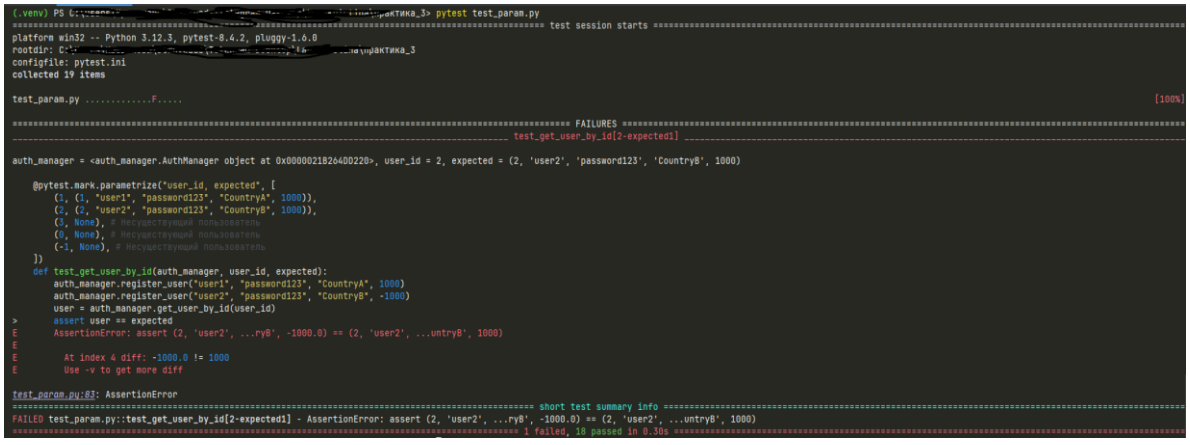
test_param.py

```python

@pytest.mark.parametrize("country, expected_count", [("CountryA", 2), ("CountryB", 1), ("CountryC", 0)])
def test_count_users_by_country(auth_manager, country, expected_count):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 1000)
    auth_manager.register_user( username: "user2",  password: "password123",  country: "CountryA",  balance: 1000)
    auth_manager.register_user( username: "user3",  password: "password123",  country: "CountryB",  balance: 1000)
    count = auth_manager.count_users_by_country(country)
    assert count == expected_count


@pytest.mark.parametrize("user_id, expected", [
    (1, (1, "user1", "password123", "CountryA", 1000)),
    (2, (2, "user2", "password123", "CountryB", 1000)),
    (3, None), # Несуществующий пользователь
    (0, None), # Несуществующий пользователь
    (-1, None), # Несуществующий пользователь
])
def test_get_user_by_id(auth_manager, user_id, expected):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 1000)
    auth_manager.register_user( username: "user2",  password: "password123",  country: "CountryB",  balance: 1000)
    user = auth_manager.get_user_by_id(user_id)
    assert user == expected


@pytest.mark.parametrize("user_id, expected", [
    (1, None),
    (2, None),
])
def test_delete_user(auth_manager, user_id, expected):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 1000)
    auth_manager.register_user( username: "user2",  password: "password123",  country: "CountryB",  balance: 1000)
    user = auth_manager.delete_user(user_id)
    assert user == expected

```

3.1. Удачные тесты:



3.2. Неудачные тесты:

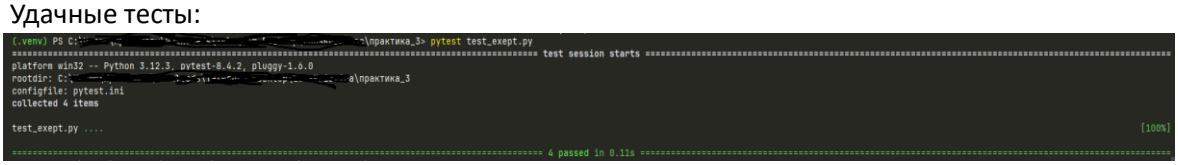


4. Файл тестирования исключений:

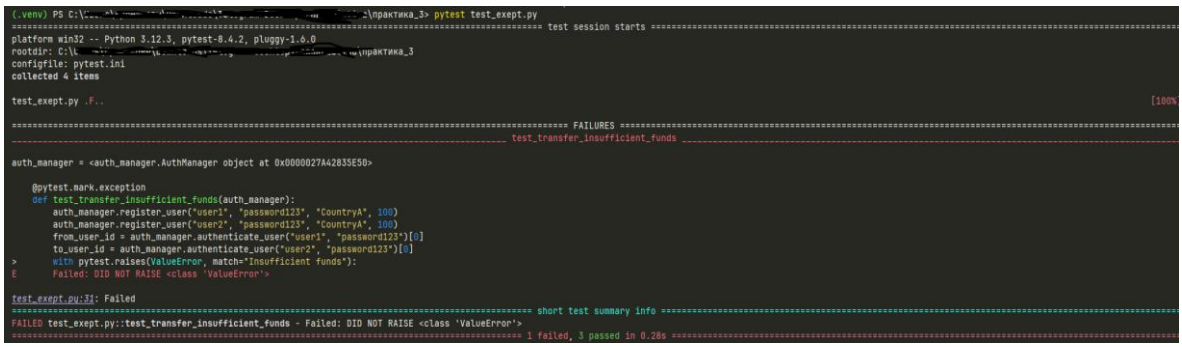
test_exept.py

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


@pytest.fixture  2 usages
def db():
    connection = sqlite3.connect(":memory:")
    yield connection
    connection.close()


@pytest.fixture  12 usages
def auth_manager(db):
    return AuthManager(db)


@pytest.mark.exception
def test_user_not_found(auth_manager):
    non_existent_user_id = 999
    user = auth_manager.get_user_by_id(non_existent_user_id)
    assert user is None


@pytest.mark.exception
def test_transfer_insufficient_funds(auth_manager):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 100)
    auth_manager.register_user( username: "user2",  password: "password123",  country: "CountryA",  balance: 100)
    from_user_id = auth_manager.authenticate_user( username: "user1",  password: "password123")[0]
    to_user_id = auth_manager.authenticate_user( username: "user2",  password: "password123")[0]
    with pytest.raises(ValueError, match="Insufficient funds"):
        auth_manager.transfer_balance(from_user_id, to_user_id,  amount: 200)


@pytest.mark.exception
def test_dict_type_raises_exception(auth_manager):
    """Тест передачи словаря"""
    service = auth_manager
    with pytest.raises((TypeError, sqlite3.Error)):
        service.get_user_by_id({"id": 1})


@pytest.mark.exception
def test_list_type_raises_exception(auth_manager):
        """Тест передачи списка"""
        service = auth_manager
        with pytest.raises((TypeError, sqlite3.Error)):
            service.get_user_by_id([1, 2, 3])
```

4.1. Удачные тесты:



4.2. Неудачные тесты:



5. Файл тестов с использованием меток:

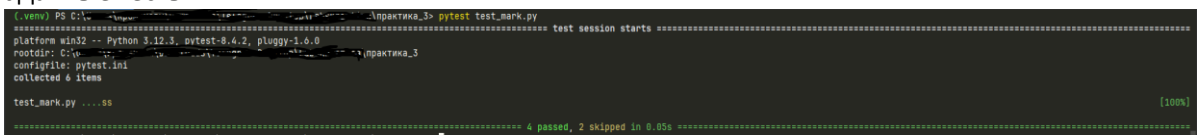
test_mark.py

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


@pytest.fixture  2 usages
def db():
    connection = sqlite3.connect(":memory:")
    yield connection
    connection.close()


@pytest.fixture  15 usages
def auth_manager(db):
    return AuthManager(db)


@pytest.mark.exception
def test_register_user_with_existing_username(auth_manager):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 100)
    with pytest.raises(sqlite3.IntegrityError, match="UNIQUE constraint failed: users.username"):
        auth_manager.register_user( username: "user1",  password: "password456",  country: "CountryB",  balance: 200)


@pytest.mark.exception
def test_authenticate_user_with_wrong_password(auth_manager):
    auth_manager.register_user( username: "user1",  password: "password123",  country: "CountryA",  balance: 100)
    user = auth_manager.authenticate_user( username: "user1",  password: "wrongpassword")
    assert user is None, "Аутентификация должна вернуть None для неверного пароля"


@pytest.mark.basic
def test_create_tables(auth_manager):
    """Тест создания таблиц"""
    # Проверяем что таблица создалась
    cursor = auth_manager.connection.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
    result = cursor.fetchone()
    assert result is not None
    assert result[0] == 'users'
```

test_mark.py

```python
    def test_create_tables(auth_manager):
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users'")
        result = cursor.fetchone()
        assert result is not None
        assert result[0] == 'users'


@pytest.mark.basic
def test_register_user_success(auth_manager):
    """Тест успешной регистрации пользователя"""
    # Регистрируем пользователя
    auth_manager.register_user( username: "testuser",  password: "password123",  country: "Russia",  balance: 100.0)

    # Проверяем что пользователь добавился
    user = auth_manager.get_user_by_id(1)
    assert user is not None
    assert user[1] == "testuser"  # username
    assert user[2] == "password123"  # password
    assert user[3] == "Russia"  # country
    assert user[4] == 100.0  # balance


@pytest.mark.skip(reason="Функция верификации пароля еще не реализована")
def test_password_strength_verification(self, auth_manager):
    """Тест проверки сложности пароля - НЕ РЕАЛИЗОВАНО"""
    # Этот тест будет пропущен, так как функциональность еще не готова
    with pytest.raises(ValueError, match="Password is too weak"):
        auth_manager.register_user( username: "user1",  password: "123",  country: "Country",  balance: 100.0)


@pytest.mark.skip(reason="API для проверки страны временно недоступно")
def test_country_validation(self, auth_manager):
    """Тест валидации страны"""
    with pytest.raises(ValueError, match="Invalid country"):
        auth_manager.register_user( username: "user1",  password: "password",  country: "NonexistentCountry",  balance: 100.0)

```

5.1. Удачные тесты:



5.2. Неудачные тесты:

6. Файл теста на SQL-инъекцию:

test_injection.py

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


@pytest.fixture  2 usages
def db():
    connection = sqlite3.connect(":memory:")
    yield connection
    connection.close()


@pytest.fixture  2 usages
def auth_manager(db):
    return AuthManager(db)


@pytest.mark.exception
def test_sql_injection_drop_table(auth_manager):
    """Тест SQL инъекции с DROP TABLE"""
    with pytest.raises(sqlite3.Error) as exc_info:
        auth_manager.get_user_by_id("DROP TABLE users")
    # Проверяем что это ошибка SQL синтаксиса
    error_message = str(exc_info.value).lower()
    assert any(word in error_message for word in ['syntax', 'near', 'error'])

```

6.1. Удачный тест: