Практика 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).

Определение класса AuthManager:

```
class User:
    def__init__(self, username, country):
        self.username = username
        self.country = country
        self.balance = 0

classAuthManager:
    def__init__(self):
        self.users = {}

    defregister_user(self, username, country):
        if username inself.users:
            raiseValueError("Пользователь уже зарегистрирован")
        self.users[username] = User(username, country)

    defauthenticate_user(self, username):
        if username notinself.users:
            raiseValueError("Пользователь не найден")
        returnself.users[username]

    defcount_users_by_country(self, country):
        returnsum(1for user inself.users.values() ifuser.country == country)

    deftransfer_funds(self, from_user, to_user, amount):
        iffrom_user.balance< amount:
            raiseValueError("Недостаточносредств")
        from_user.balance -= amount
        to_user.balance += amount

    defregister_sql_injection(self, username):
        if"'"inusername:
            raiseValueError("Недопустимый символ в имени пользователя")
```

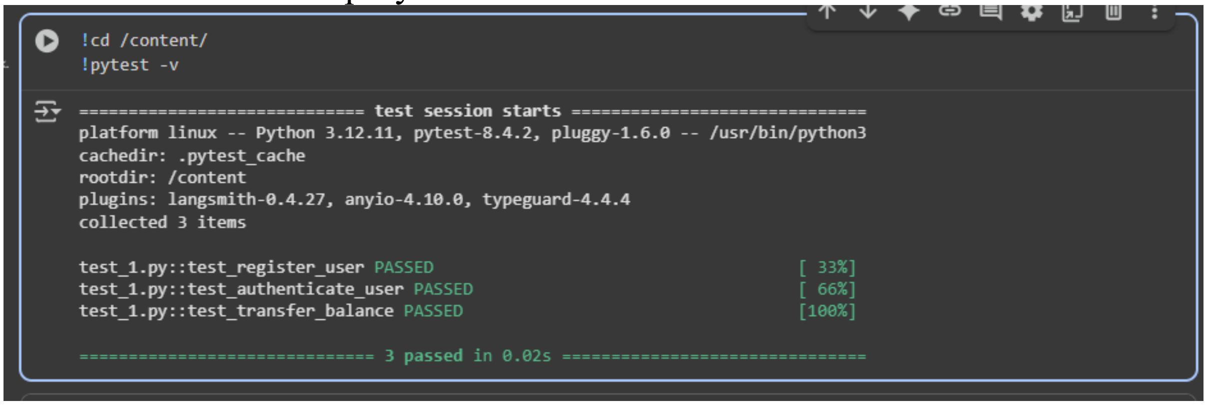
Листинг файла:

```
import sqlite3
class AuthManager:
    def __init__(self, connection):
        self.connection = connection
        self.create_tables()
    def create_tables(self):
        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):
        with self.connection:
            self.connection.execute("""
            INSERT INTO users (username, password, country, balance)
            VALUES ('{}', '{}', '{}', {})
            """.format(username, password, country, balance))
    def authenticate_user(self, username, password):
        cursor = self.connection.cursor()
        cursor.execute("""
        SELECT * FROM users
        WHERE username = '{}' AND password = '{}'
        """.format(username, password))
        return cursor.fetchone()
    def delete_user(self, user_id):
        with self.connection:
            self.connection.execute("""
            DELETE FROM users WHERE id = {}
            """.format(user_id))
    def get_user_by_id(self, user_id):
        cursor = self.connection.cursor()
        cursor.execute("""
        SELECT * FROM users WHERE id = {}
        """.format(user_id))
        return cursor.fetchone()
    def count_users_by_country(self, country):
        cursor = self.connection.cursor()
        cursor.execute("""
        SELECT COUNT(*) FROM users WHERE country = '{}'
        """.format(country))
        return cursor.fetchone()[0]
    def transfer_balance(self, from_user_id, to_user_id, amount):
        with self.connection:
# Проверяем, достаточно ли средств
            cursor = self.connection.cursor()
            cursor.execute("SELECT balance FROM users WHERE id
={}".format(from_user_id))
            from_balance = cursor.fetchone()[0]
            if from_balance < amount:
                raise ValueError("Insufficient funds")
# Выполняем перевод
            self.connection.execute("""
            UPDATE users SET balance = balance - {} WHERE id = {}
            """.format(amount, from_user_id))
            self.connection.execute("""
            UPDATE users SET balance = balance + {} WHERE id = {}
            """.format(amount, to_user_id))
```

Базовые тесты:

```
import pytest
import sqlite3
from auth_manager import AuthManager  # Предполагается, что класс
находится в файле auth_manager.py

# Фикстура для создания временной базы данных
@pytest.fixture
def db_connection():
    connection = sqlite3.connect(":memory:")  # Используем SQLite базу
данных в памяти
    yield connection
    connection.close()

# Фикстура для создания экземпляра AuthManager
@pytest.fixture
def auth_manager(db_connection):
    return AuthManager(db_connection)

# 1. Тест на регистрацию пользователя
def test_register_user(db_connection):
 auth_manager = AuthManager(db_connection)
 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"
# 2. Тест на аутентификацию пользователя
def test_authenticate_user(db_connection):
 auth_manager = AuthManager(db_connection)
 auth_manager.register_user("test_user", "password123", "Russia", 100)
 user = auth_manager.authenticate_user("test_user", "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"
# 3. Тест на перевод средств
def test_transfer_balance(auth_manager):
 # Регистрация двух пользователей
 auth_manager.register_user("user1", "password123", "CountryA", 100)
 auth_manager.register_user("user2", "password123", "CountryB", 500)
 user1 = auth_manager.authenticate_user("user1", "password123")
 user2 = auth_manager.authenticate_user("user2", "password123")
 try:
  auth_manager.transfer_balance(user1[0], user2[0], 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
```

Положительный результат:



```
!cd /content/
!pytest -v
```

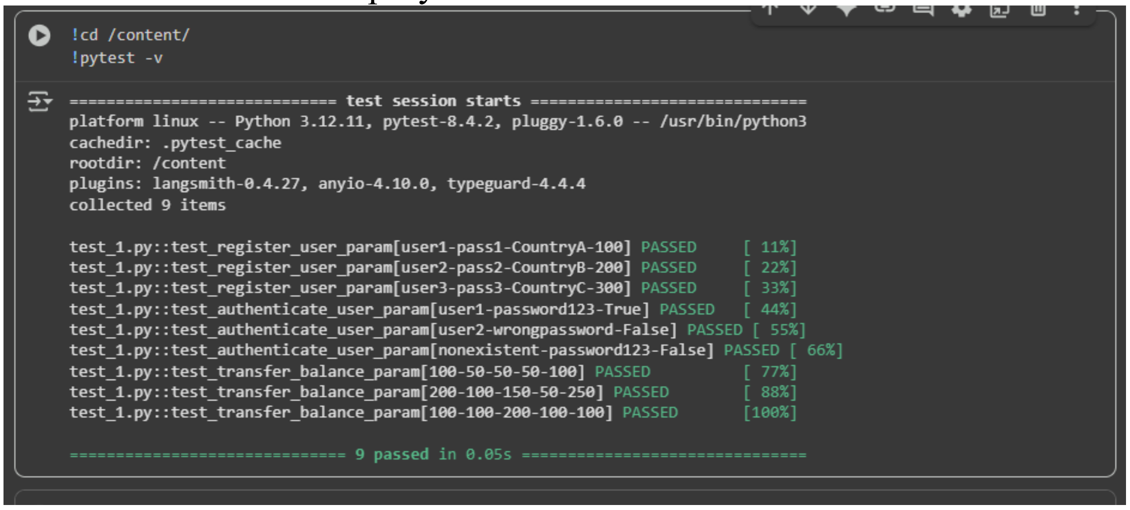
```
============================= test session starts ==============================
platform linux -- Python 3.12.11, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: langsmith-0.4.27, anyio-4.10.0, typeguard-4.4.4
collected 3 items

test_1.py::test_register_user PASSED                                     [ 33%]
test_1.py::test_authenticate_user PASSED                                 [ 66%]
test_1.py::test_transfer_balance PASSED                                  [100%]

============================== 3 passed in 0.02s ===============================
```

Параметризованные тесты:

```
import pytest
import sqlite3
from auth_manager import AuthManager  # Предполагается, что класс
находится в файле auth_manager.py

# ------------------------------
# Фикстура для создания временной базы данных
# ------------------------------
@pytest.fixture
def db():
    connection = sqlite3.connect(":memory:")  # База данных в памяти
    connection.row_factory = sqlite3.Row        # Чтобы можно было
использовать именованные поля
    yield connection
    connection.close()

# -----------------------------
# Фикстура для создания экземпляра AuthManager
# -----------------------------
@pytest.fixture
def auth_manager(db):
    return AuthManager(db)

# -----------------------------
# 1. Параметризованный тест на регистрацию пользователей
# -----------------------------
@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["username"] == username, f"Имя пользователя должно быть
{username}"
    assert user["country"] == country, f"Страна пользователя должна быть
{country}"
    assert user["balance"] == balance, f"Баланс пользователя должен быть
{balance}"

# -----------------------------
# 2. Параметризованный тест на аутентификацию пользователей
# -----------------------------
@pytest.mark.parametrize(
    "username, password, expected_result",
    [
        ("user1", "password123", True),        # Успешная аутентификация
        ("user2", "wrongpassword", False),     # Неверный пароль
        ("nonexistent", "password123", False) # Не существует пользователь
    ]
)
def test_authenticate_user_param(auth_manager, username, password,
expected_result):
    # Регистрируем пользователей для теста
    auth_manager.register_user("user1", "password123", "CountryA", 100)
    auth_manager.register_user("user2", "password456", "CountryB", 200)

    # Аутентификация
    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"

# -----------------------------
# 3. Параметризованный тест на перевод средств
# -----------------------------
@pytest.mark.parametrize(
    "from_balance, to_balance, transfer_amount, expected_from_balance,
expected_to_balance",
    [
        (100, 50, 50, 50, 100),    # Успешный перевод
        (200, 100, 150, 50, 250),  # Успешный перевод
        (100, 100, 200, 100, 100), # Неудачный перевод (недостаточно
средств)
    ]
)
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("user1", "password123", "CountryA",
from_balance)
    auth_manager.register_user("user2", "password123", "CountryA",
to_balance)

    # Аутентификация пользователей
    from_user_id = auth_manager.authenticate_user("user1",
"password123")["id"]
    to_user_id = auth_manager.authenticate_user("user2",
"password123")["id"]

    # Выполняем перевод, если хватает средств
    if from_balance >= transfer_amount:
        auth_manager.transfer_balance(from_user_id, to_user_id,
transfer_amount)

    # Проверяем баланс
    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["balance"] == expected_from_balance, f"У отправителя
должно остаться {expected_from_balance} единиц"
```

```
    assert to_user["balance"] == expected_to_balance, f"У получателя
должно быть {expected_to_balance} единиц"
```

Положительный результат:

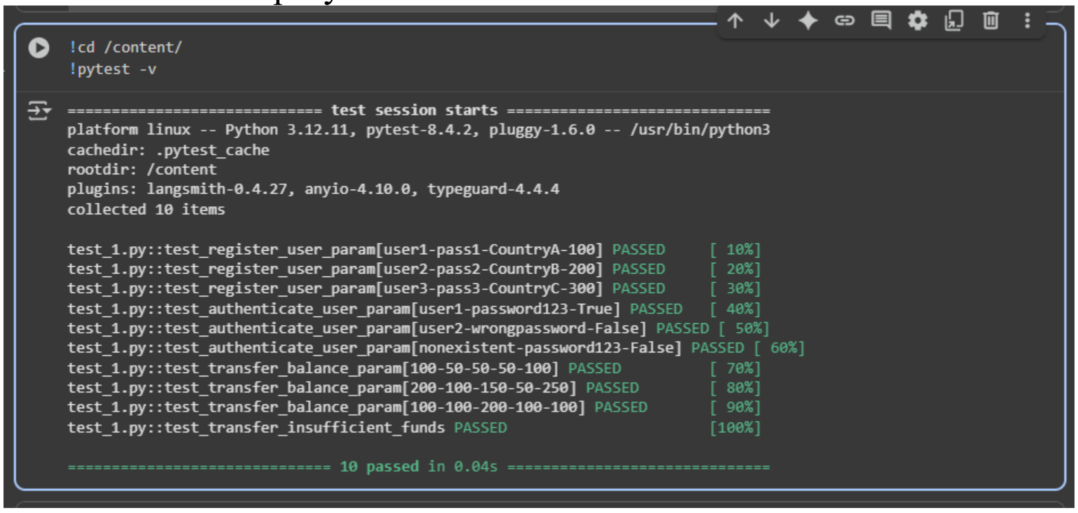


```
!cd /content/
!pytest -v

============================ test session starts ============================
platform linux -- Python 3.12.11, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: langsmith-0.4.27, anyio-4.10.0, typeguard-4.4.4
collected 9 items

test_1.py::test_register_user_param[user1-pass1-CountryA-100] PASSED     [ 11%]
test_1.py::test_register_user_param[user2-pass2-CountryB-200] PASSED     [ 22%]
test_1.py::test_register_user_param[user3-pass3-CountryC-300] PASSED     [ 33%]
test_1.py::test_authenticate_user_param[user1-password123-True] PASSED   [ 44%]
test_1.py::test_authenticate_user_param[user2-wrongpassword-False] PASSED [ 55%]
test_1.py::test_authenticate_user_param[nonexistent-password123-False] PASSED [ 66%]
test_1.py::test_transfer_balance_param[100-50-50-50-100] PASSED          [ 77%]
test_1.py::test_transfer_balance_param[200-100-150-50-250] PASSED        [ 88%]
test_1.py::test_transfer_balance_param[100-100-200-100-100] PASSED       [100%]

============================== 9 passed in 0.05s ==============================
```

Тестирование исключений:

```
def test_transfer_insufficient_funds(auth_manager):
    auth_manager.register_user("user1", "password123", "CountryA", 100)
    auth_manager.register_user("user2", "password123", "CountryA", 100)

    from_user_id = auth_manager.authenticate_user("user1",
"password123")["id"]
    to_user_id = auth_manager.authenticate_user("user2",
"password123")["id"]

    # Проверяем, что при недостатке средств выбрасывается исключение
    with pytest.raises(ValueError, match="Insufficient funds"):
        auth_manager.transfer_balance(from_user_id, to_user_id, 200)
```

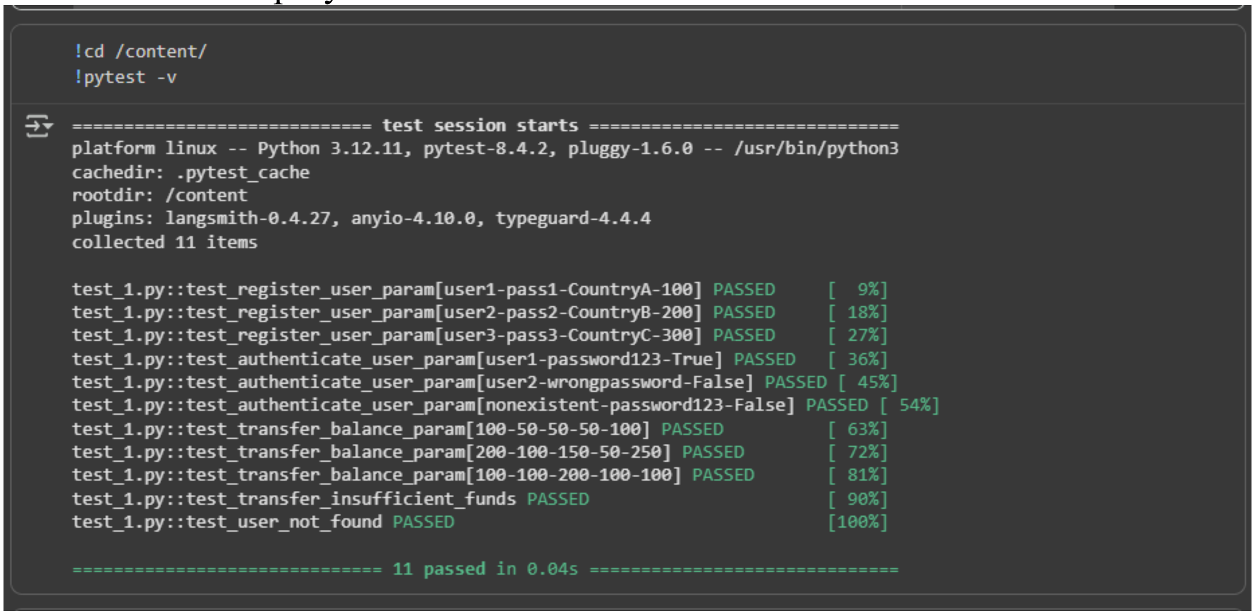
Положительный результат:



```
!cd /content/
!pytest -v

============================ test session starts ============================
platform linux -- Python 3.12.11, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: langsmith-0.4.27, anyio-4.10.0, typeguard-4.4.4
collected 10 items

test_1.py::test_register_user_param[user1-pass1-CountryA-100] PASSED     [ 10%]
test_1.py::test_register_user_param[user2-pass2-CountryB-200] PASSED     [ 20%]
test_1.py::test_register_user_param[user3-pass3-CountryC-300] PASSED     [ 30%]
test_1.py::test_authenticate_user_param[user1-password123-True] PASSED   [ 40%]
test_1.py::test_authenticate_user_param[user2-wrongpassword-False] PASSED [ 50%]
test_1.py::test_authenticate_user_param[nonexistent-password123-False] PASSED [ 60%]
test_1.py::test_transfer_balance_param[100-50-50-50-100] PASSED          [ 70%]
test_1.py::test_transfer_balance_param[200-100-150-50-250] PASSED        [ 80%]
test_1.py::test_transfer_balance_param[100-100-200-100-100] PASSED       [ 90%]
test_1.py::test_transfer_insufficient_funds PASSED                       [100%]

============================== 10 passed in 0.04s ==============================
```

```python
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
```

Положительный результат:



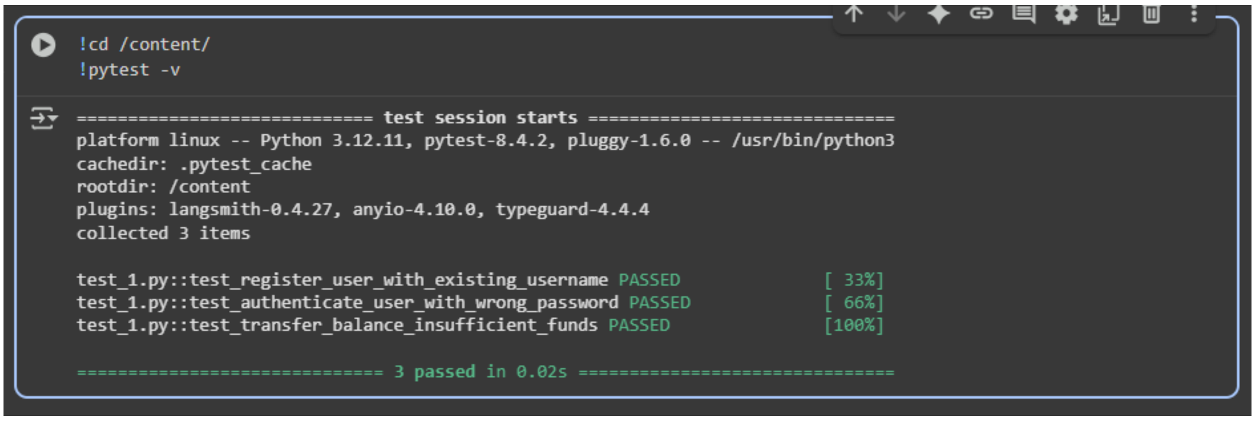
```
!cd /content/
!pytest -v

============================= test session starts ==============================
platform linux -- Python 3.12.11, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: langsmith-0.4.27, anyio-4.10.0, typeguard-4.4.4
collected 11 items

test_1.py::test_register_user_param[user1-pass1-CountryA-100] PASSED     [  9%]
test_1.py::test_register_user_param[user2-pass2-CountryB-200] PASSED     [ 18%]
test_1.py::test_register_user_param[user3-pass3-CountryC-300] PASSED     [ 27%]
test_1.py::test_authenticate_user_param[user1-password123-True] PASSED   [ 36%]
test_1.py::test_authenticate_user_param[user2-wrongpassword-False] PASSED [ 45%]
test_1.py::test_authenticate_user_param[nonexistent-password123-False] PASSED [ 54%]
test_1.py::test_transfer_balance_param[100-50-50-50-100] PASSED          [ 63%]
test_1.py::test_transfer_balance_param[200-100-150-50-250] PASSED        [ 72%]
test_1.py::test_transfer_balance_param[100-100-200-100-100] PASSED       [ 81%]
test_1.py::test_transfer_insufficient_funds PASSED                       [ 90%]
test_1.py::test_user_not_found PASSED                                    [100%]

============================== 11 passed in 0.04s ==============================
```

Использование меток:

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

# -----------------------------
# Фикстура для базы данных
# -----------------------------
@pytest.fixture
def db():
    connection = sqlite3.connect(":memory:")
    connection.row_factory = sqlite3.Row
    cursor = connection.cursor()

    # Создаём таблицу users с уникальным username
    cursor.execute("""
    CREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT,
        country TEXT,
        balance INTEGER
    )
    """)
    connection.commit()
    yield connection
    connection.close()
```

```python
# ------------------------------
# Фикстура для AuthManager
# ------------------------------
@pytest.fixture
def auth_manager(db):
    return AuthManager(db)

# ------------------------------
# 1. Тест: регистрация с уже существующим именем
# ------------------------------
def test_register_user_with_existing_username(auth_manager):
    auth_manager.register_user("user1", "password123", "CountryA", 100)
    with pytest.raises(sqlite3.IntegrityError, match="UNIQUE constraint
failed: users.username"):
        auth_manager.register_user("user1", "password456", "CountryB",
200)

# ------------------------------
# 2. Тест: аутентификация с неверным паролем
# ------------------------------
def test_authenticate_user_with_wrong_password(auth_manager):
    auth_manager.register_user("user1", "password123", "CountryA", 100)
    user = auth_manager.authenticate_user("user1", "wrongpassword")
    assert user is None, "Аутентификация должна вернуть None для неверного
пароля"

# ------------------------------
# 3. Тест: перевод средств с недостаточным балансом
# ------------------------------
def test_transfer_balance_insufficient_funds(auth_manager):
    # Регистрация пользователей
    auth_manager.register_user("user1", "password123", "CountryA", 100)
    auth_manager.register_user("user2", "password123", "CountryA", 100)

    # Аутентификация пользователей
    from_user_id = auth_manager.authenticate_user("user1",
"password123")["id"]
    to_user_id = auth_manager.authenticate_user("user2",
"password123")["id"]

    # Проверка исключения при недостатке средств
    with pytest.raises(ValueError, match="Insufficient funds"):
        auth_manager.transfer_balance(from_user_id, to_user_id, 200)
```

Положительный результат:



```
!cd /content/
!pytest -v
```

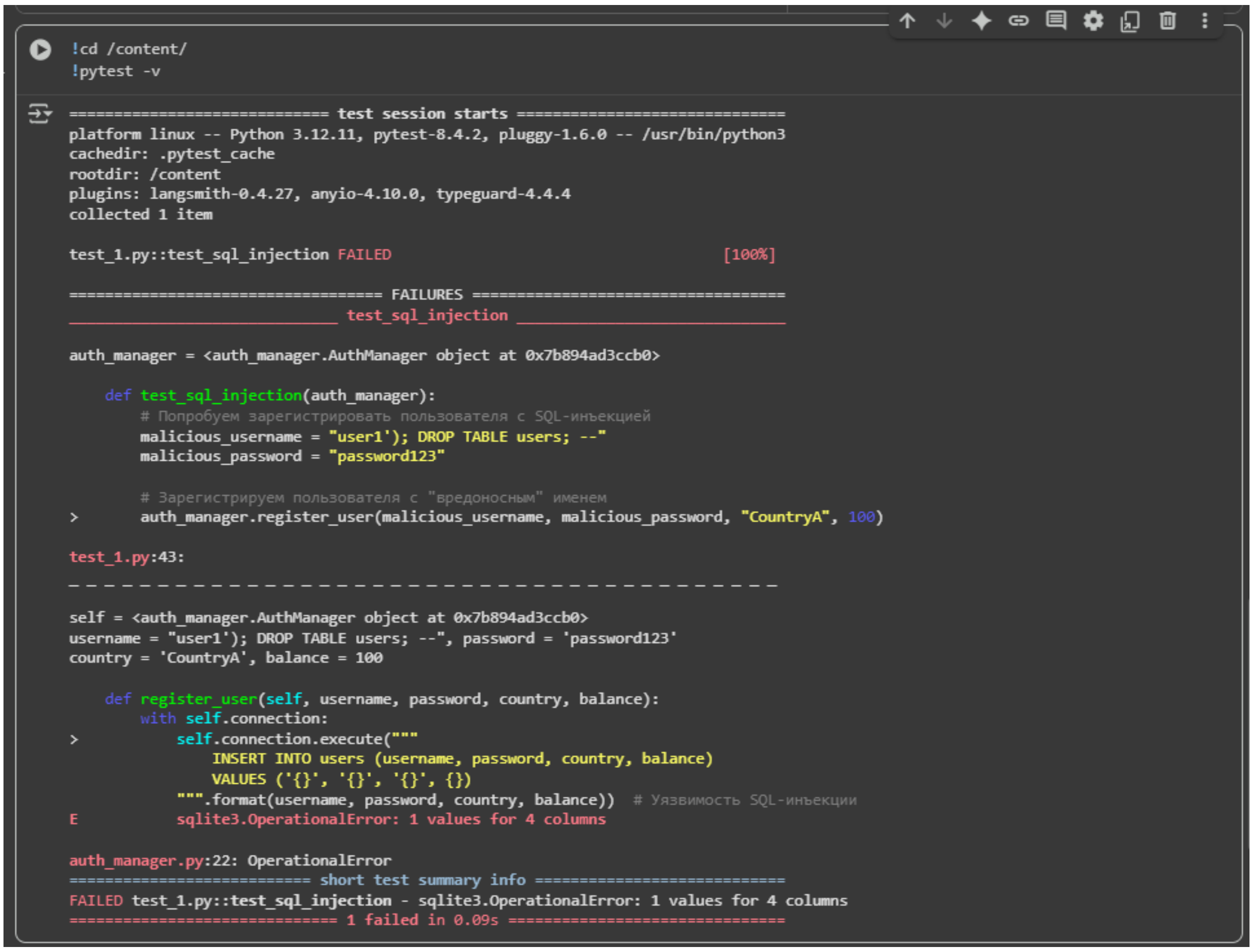
```
============================= test session starts ==============================
platform linux -- Python 3.12.11, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: langsmith-0.4.27, anyio-4.10.0, typeguard-4.4.4
collected 3 items

test_1.py::test_register_user_with_existing_username PASSED              [ 33%]
test_1.py::test_authenticate_user_with_wrong_password PASSED             [ 66%]
test_1.py::test_transfer_balance_insufficient_funds PASSED               [100%]

============================== 3 passed in 0.02s ===============================
```

Тест на SQL-инъекцию

Попытка SQL-инъекции: При регистрации пользователя используется потенциально опасное имя пользователя, которое могло бы попытаться удалить таблицу users.

```
import pytest
import sqlite3
from auth_manager import AuthManager  # Предполагается, что класс
находится в файле auth_manager.py

# ------------------------------
# Фикстура для создания временной базы данных
# ------------------------------
@pytest.fixture
def db():
    connection = sqlite3.connect(":memory:")  # База данных в памяти
    connection.row_factory = sqlite3.Row
    cursor = connection.cursor()
    # Создаём таблицу users с уникальным username
    cursor.execute("""
    CREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT,
        country TEXT,
        balance INTEGER
    )
    """)
    connection.commit()
    yield connection
    connection.close()

# ------------------------------
# Фикстура для создания экземпляра AuthManager
# ------------------------------
@pytest.fixture
def auth_manager(db):
    return AuthManager(db)

# ------------------------------
# Тест на SQL-инъекцию
# ------------------------------
def test_sql_injection(auth_manager):
    # Попробуем зарегистрировать пользователя с SQL-инъекцией
    malicious_username = "user1'); DROP TABLE users; --"
    malicious_password = "password123"

    # Зарегистрируем пользователя с "вредоносным" именем
    auth_manager.register_user(malicious_username, malicious_password,
"CountryA", 100)

    # Проверяем, существует ли таблица users
    cursor = auth_manager.db.cursor()  # Используем фикстурную базу через
атрибут db
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND
name='users';")
    table_exists = cursor.fetchone()

    # Если таблица users существует, SQL-инъекция не удалась
    assert table_exists is not None, "Таблица 'users' должна существовать.
SQL-инъекция могла быть успешной."
```

Положительный результат:



```
!cd /content/
!pytest -v
```

```
============================ test session starts =============================
platform linux -- Python 3.12.11, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: langsmith-0.4.27, anyio-4.10.0, typeguard-4.4.4
collected 1 item

test_1.py::test_sql_injection FAILED                                     [100%]

================================== FAILURES ==================================
______________________________ test_sql_injection ______________________________

auth_manager = <auth_manager.AuthManager object at 0x7b894ad3ccb0>

    def test_sql_injection(auth_manager):
        # Попробуем зарегистрировать пользователя с SQL-инъекцией
        malicious_username = "user1'); DROP TABLE users; --"
        malicious_password = "password123"

        # Зарегистрируем пользователя с "вредоносным" именем
>       auth_manager.register_user(malicious_username, malicious_password, "CountryA", 100)

test_1.py:43:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <auth_manager.AuthManager object at 0x7b894ad3ccb0>
username = "user1'); DROP TABLE users; --", password = 'password123'
country = 'CountryA', balance = 100

    def register_user(self, username, password, country, balance):
        with self.connection:
>           self.connection.execute("""
                INSERT INTO users (username, password, country, balance)
                VALUES ('{}', '{}', '{}', {})
            """.format(username, password, country, balance))  # Уязвимость SQL-инъекции
E           sqlite3.OperationalError: 1 values for 4 columns

auth_manager.py:22: OperationalError
========================== short test summary info ===========================
FAILED test_1.py::test_sql_injection - sqlite3.OperationalError: 1 values for 4 columns
============================== 1 failed in 0.09s ===============================
```