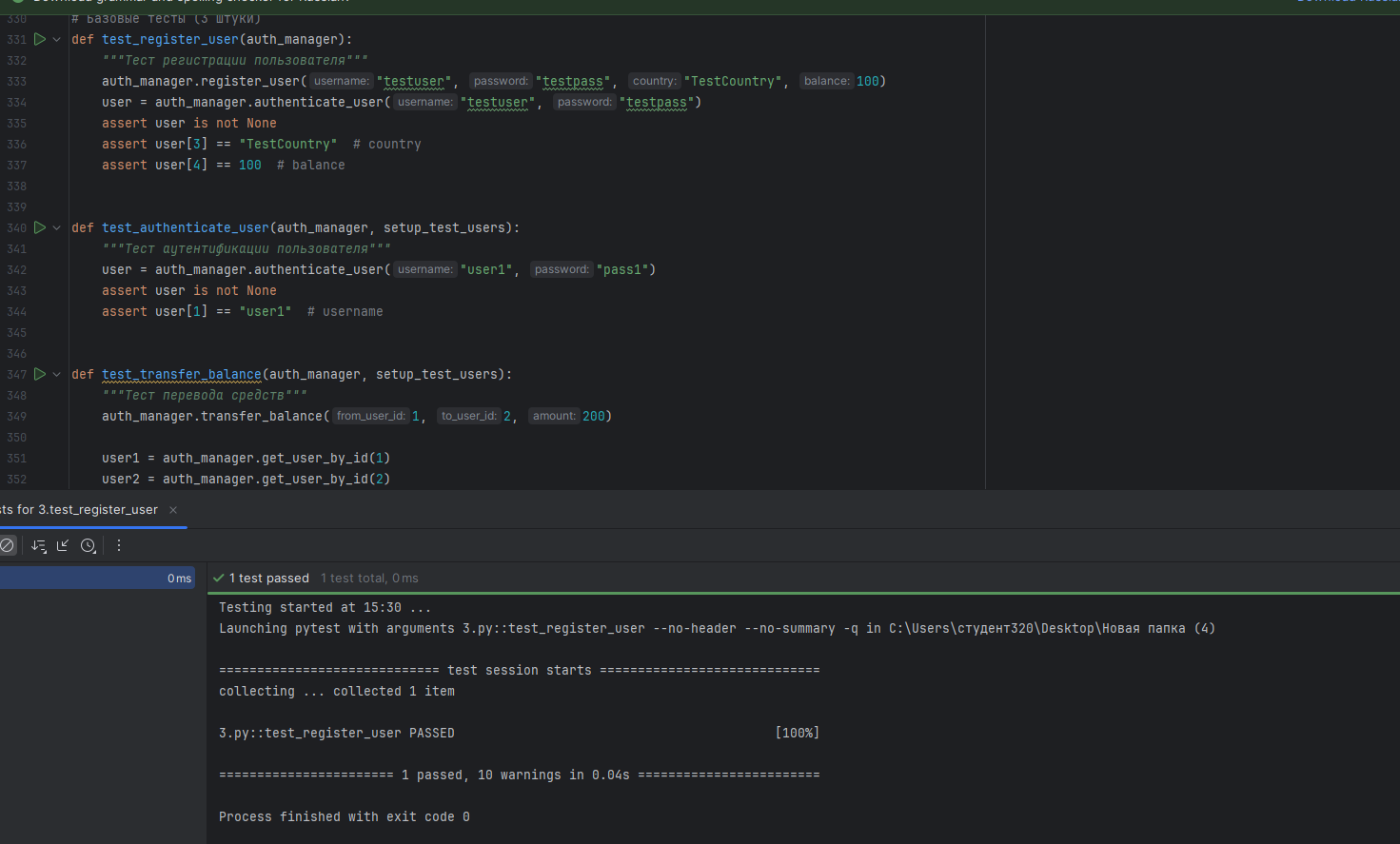
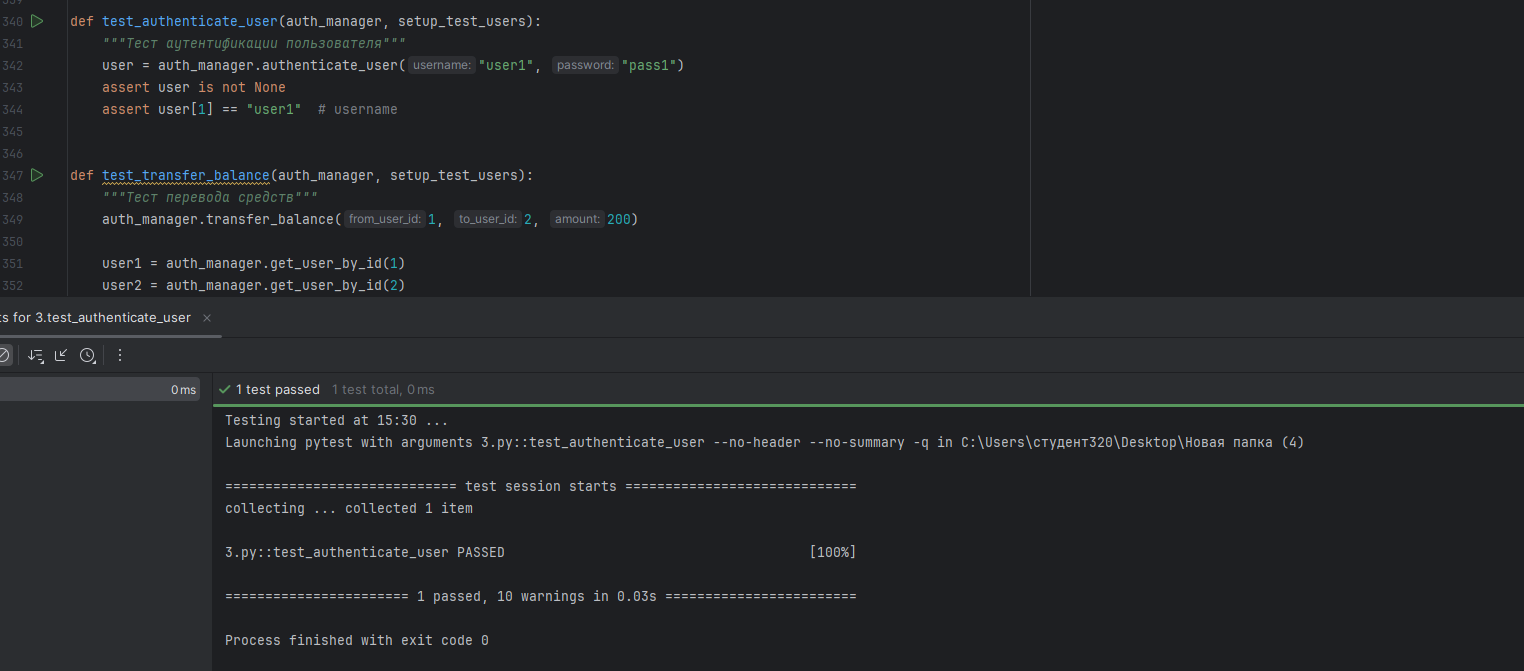
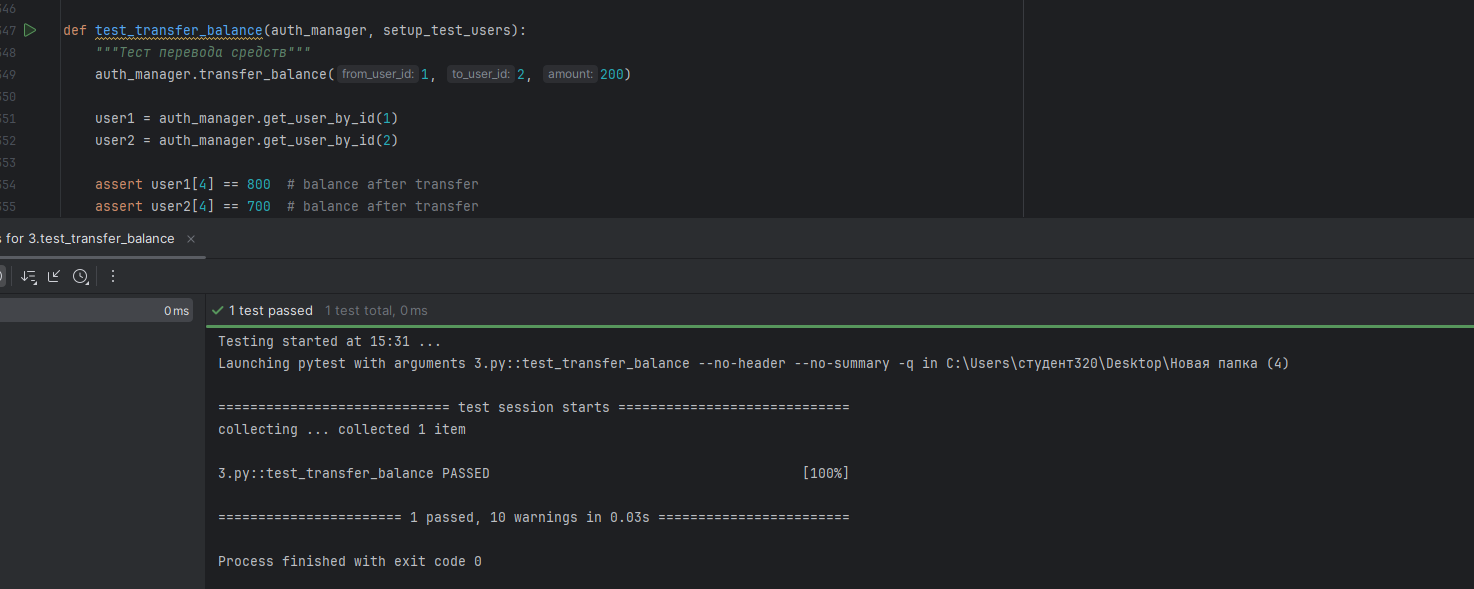
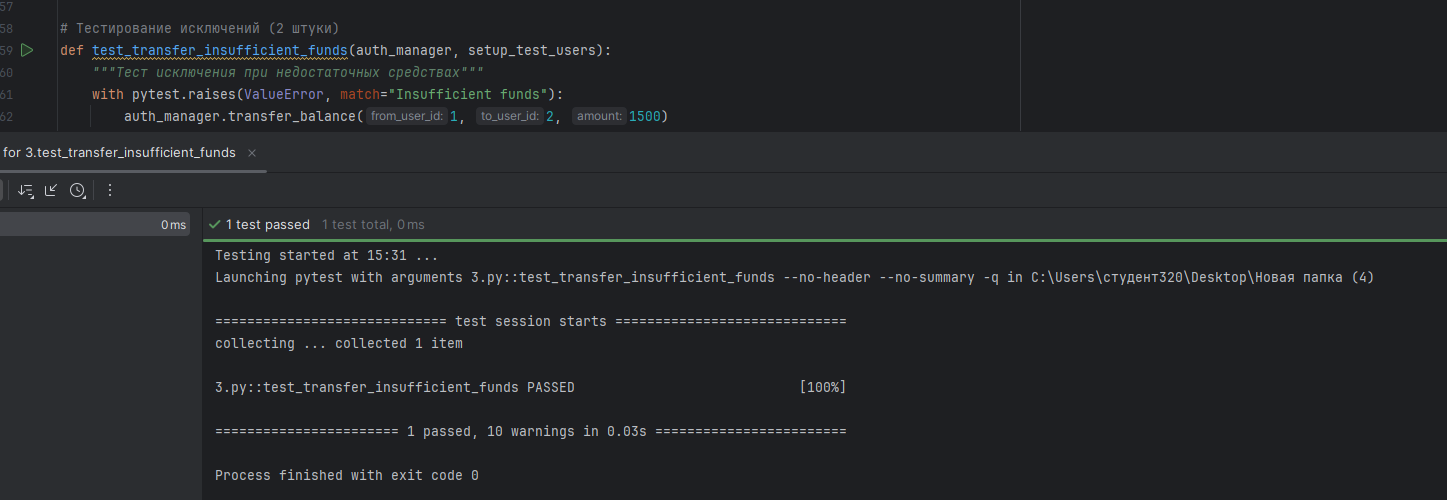
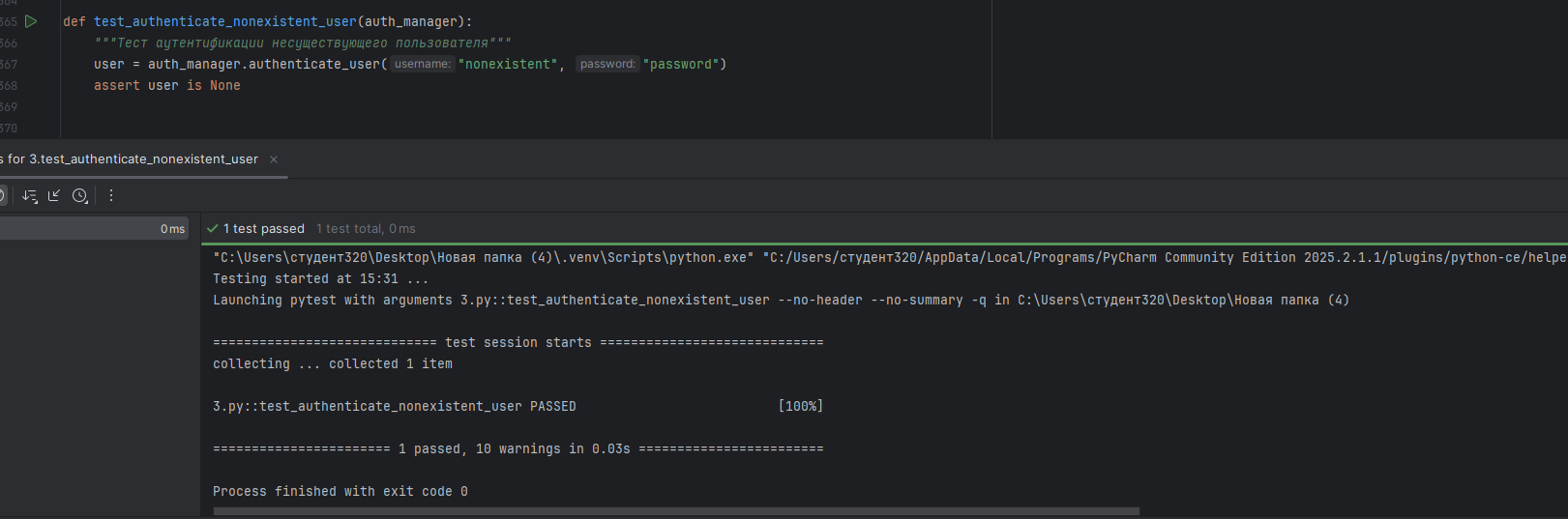
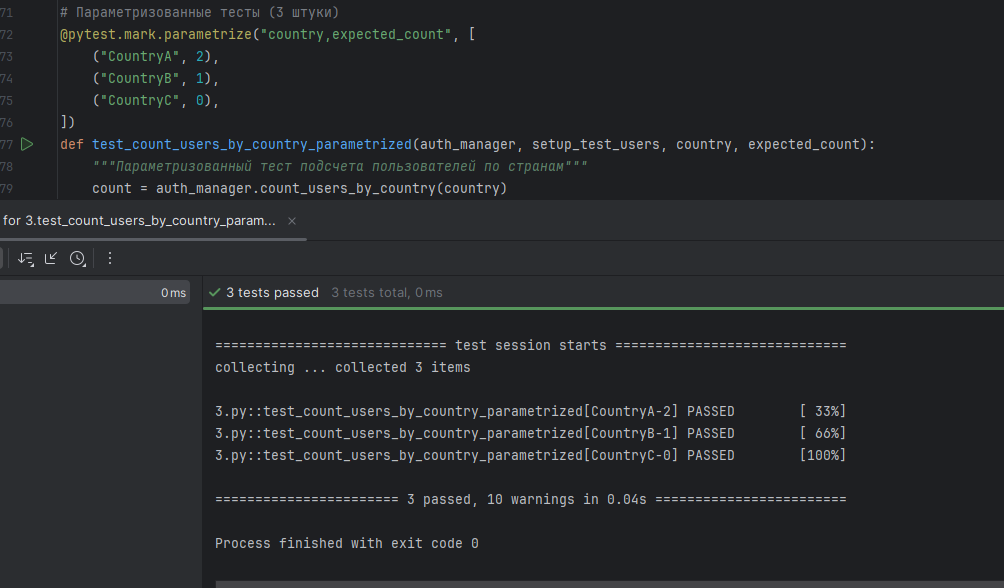
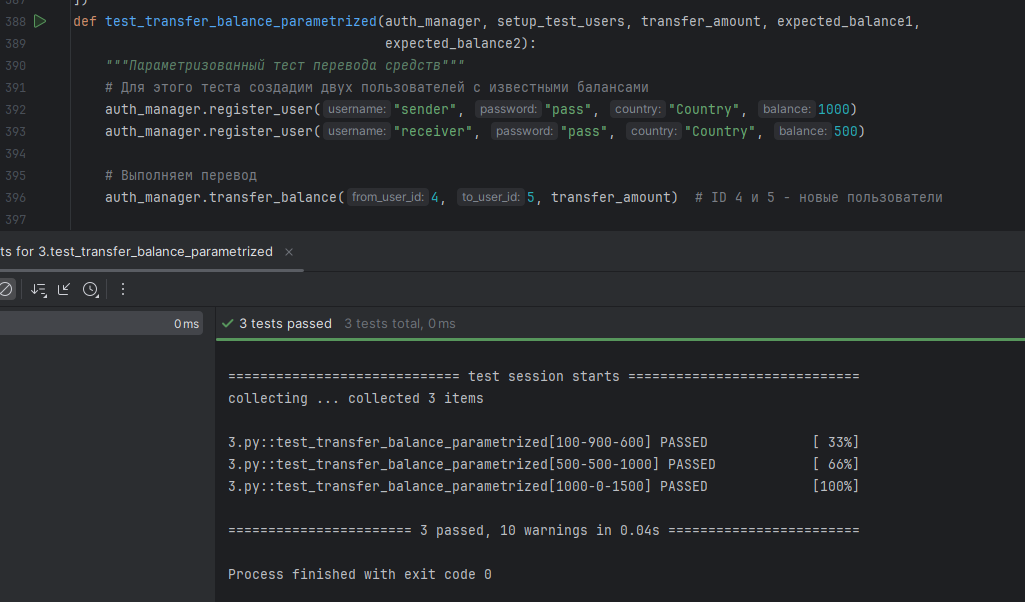
Лабораторная работа №3

Базовые тесты (3 штуки)  
def test\_register\_user(auth\_manager):  
 *"""Тест регистрации пользователя"""* auth\_manager.register\_user("testuser", "testpass", "TestCountry", 100)  
 user = auth\_manager.authenticate\_user("testuser", "testpass")  
 assert user is not None  
 assert user[3] == "TestCountry" # country  
 assert user[4] == 100 # balance  
  
  
def test\_authenticate\_user(auth\_manager, setup\_test\_users):  
 *"""Тест аутентификации пользователя"""* user = auth\_manager.authenticate\_user("user1", "pass1")  
 assert user is not None  
 assert user[1] == "user1" # username  
  
  
def test\_transfer\_balance(auth\_manager, setup\_test\_users):  
 *"""Тест перевода средств"""* auth\_manager.transfer\_balance(1, 2, 200)  
  
 user1 = auth\_manager.get\_user\_by\_id(1)  
 user2 = auth\_manager.get\_user\_by\_id(2)  
  
 assert user1[4] == 800 # balance after transfer  
 assert user2[4] == 700 # balance after transfer

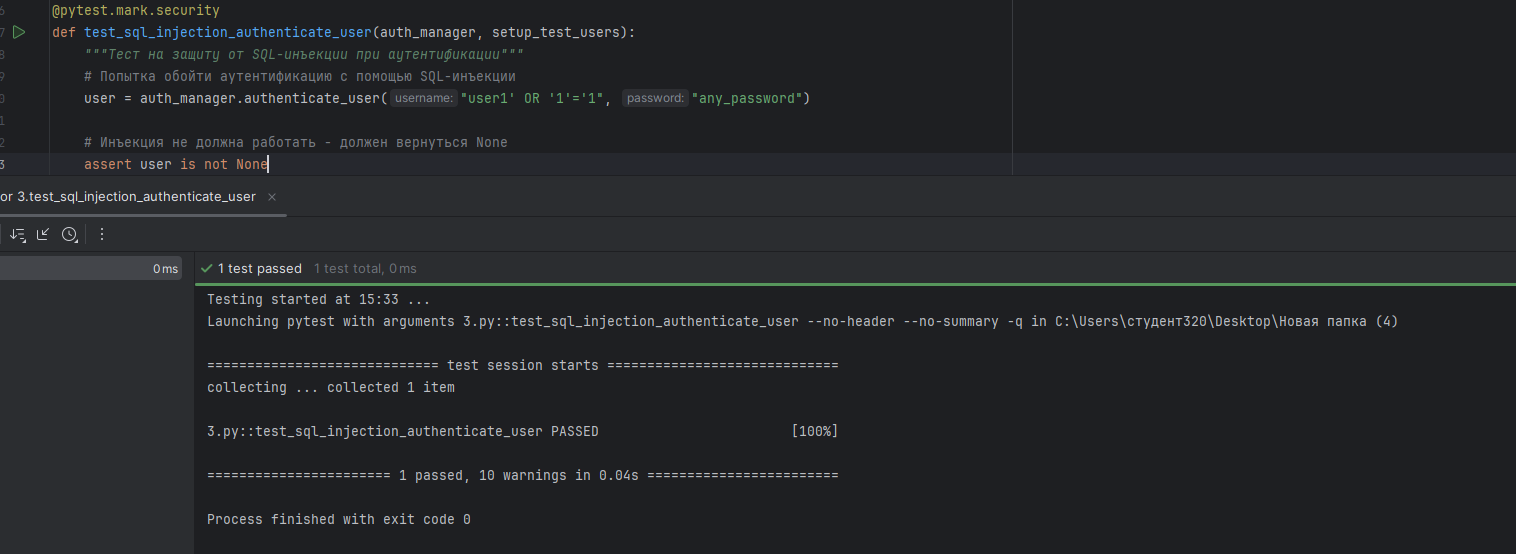


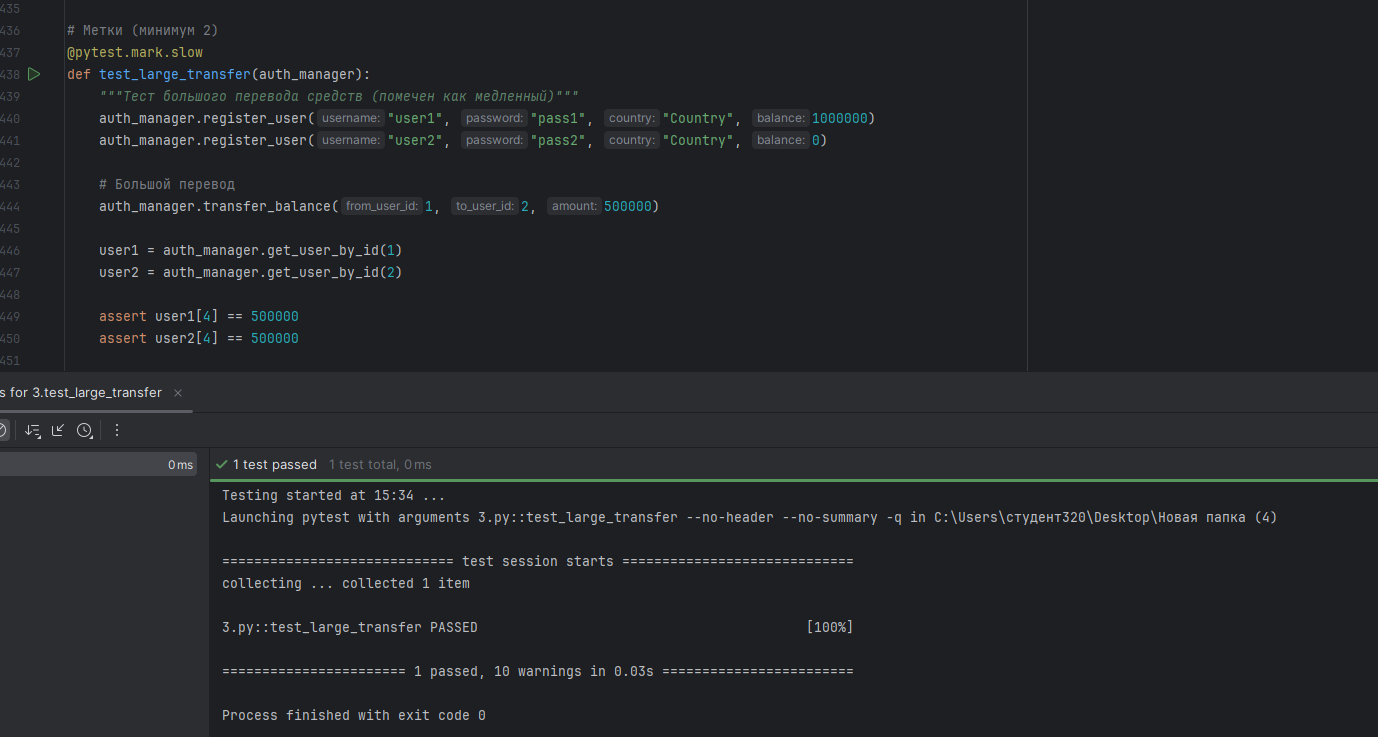
Тестирование исключений (2 штуки)  
def test\_transfer\_insufficient\_funds(auth\_manager, setup\_test\_users):  
 *"""Тест исключения при недостаточных средствах"""* with pytest.raises(ValueError, match="Insufficient funds"):  
 auth\_manager.transfer\_balance(1, 2, 1500)  
  
  
def test\_authenticate\_nonexistent\_user(auth\_manager):  
 *"""Тест аутентификации несуществующего пользователя"""* user = auth\_manager.authenticate\_user("nonexistent", "password")  
 assert user is None  
  
  
# Параметризованные тесты (3 штуки)  
@pytest.mark.parametrize("country,expected\_count", [  
 ("CountryA", 2),  
 ("CountryB", 1),  
 ("CountryC", 0),  
])  
def test\_count\_users\_by\_country\_parametrized(auth\_manager, setup\_test\_users, country, expected\_count):  
 *"""Параметризованный тест подсчета пользователей по странам"""* count = auth\_manager.count\_users\_by\_country(country)  
 assert count == expected\_count  
  
  
@pytest.mark.parametrize("transfer\_amount,expected\_balance1,expected\_balance2", [  
 (100, 900, 600), # Обычный перевод  
 (500, 500, 1000), # Перевод половины баланса  
 (1000, 0, 1500), # Перевод всего баланса  
])

  
def test\_transfer\_balance\_parametrized(auth\_manager, setup\_test\_users, transfer\_amount, expected\_balance1,  
 expected\_balance2):  
 *"""Параметризованный тест перевода средств"""* # Для этого теста создадим двух пользователей с известными балансами  
 auth\_manager.register\_user("sender", "pass", "Country", 1000)  
 auth\_manager.register\_user("receiver", "pass", "Country", 500)  
  
 # Выполняем перевод  
 auth\_manager.transfer\_balance(4, 5, transfer\_amount) # ID 4 и 5 - новые пользователи  
  
 # Проверяем балансы  
 sender = auth\_manager.get\_user\_by\_id(4)  
 receiver = auth\_manager.get\_user\_by\_id(5)  
  
 assert sender[4] == expected\_balance1  
 assert receiver[4] == expected\_balance2



Тесты на SQL-инъекции (должны проходить после исправления)  
@pytest.mark.security  
def test\_sql\_injection\_register\_user(auth\_manager):  
 *"""Тест на защиту от SQL-инъекции при регистрации"""* # Попытка SQL-инъекции через имя пользователя  
 malicious\_username = "testuser'; DROP TABLE users; --"  
  
 # Эта попытка должна быть безопасно обработана  
 auth\_manager.register\_user(malicious\_username, "password123", "Country", 1000)  
  
 # Проверяем, что таблица users всё ещё существует  
 cursor = auth\_manager.connection.cursor()  
 cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='users';")  
 tables = cursor.fetchall()  
  
 # Также проверяем, что пользователь был создан с точным именем (включая инъекцию)  
 user = auth\_manager.authenticate\_user(malicious\_username, "password123")  
 assert user is not None  
 assert user[1] == malicious\_username # username  
  
@pytest.mark.security  
def test\_sql\_injection\_authenticate\_user(auth\_manager, setup\_test\_users):  
 *"""Тест на защиту от SQL-инъекции при аутентификации"""* # Попытка обойти аутентификацию с помощью SQL-инъекции  
 user = auth\_manager.authenticate\_user("user1' OR '1'='1", "any\_password")  
  
 # Инъекция не должна работать - должен вернуться None  
 assert user is None



Метки (минимум 2)  
@pytest.mark.slow  
def test\_large\_transfer(auth\_manager):  
 *"""Тест большого перевода средств (помечен как медленный)"""* auth\_manager.register\_user("user1", "pass1", "Country", 1000000)  
 auth\_manager.register\_user("user2", "pass2", "Country", 0)  
  
 # Большой перевод  
 auth\_manager.transfer\_balance(1, 2, 500000)  
  
 user1 = auth\_manager.get\_user\_by\_id(1)  
 user2 = auth\_manager.get\_user\_by\_id(2)  
  
 assert user1[4] == 500000  
 assert user2[4] == 500000  
  
  
@pytest.mark.security  
def test\_sql\_injection\_country(auth\_manager, setup\_test\_users):  
 *"""Тест на защиту от SQL-инъекции в подсчете по странам"""* # Попытка SQL-инъекции через параметр country  
 count = auth\_manager.count\_users\_by\_country("CountryA' OR '1'='1")  
  
 # Должен вернуть 0, а не общее количество пользователей  
 assert count == 0

