Практическое задание №3

Задание:

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).

ОБЯЗАТЕЛЬНО!!!: должен присутствовать тест на SQL инъекции.

Решение:

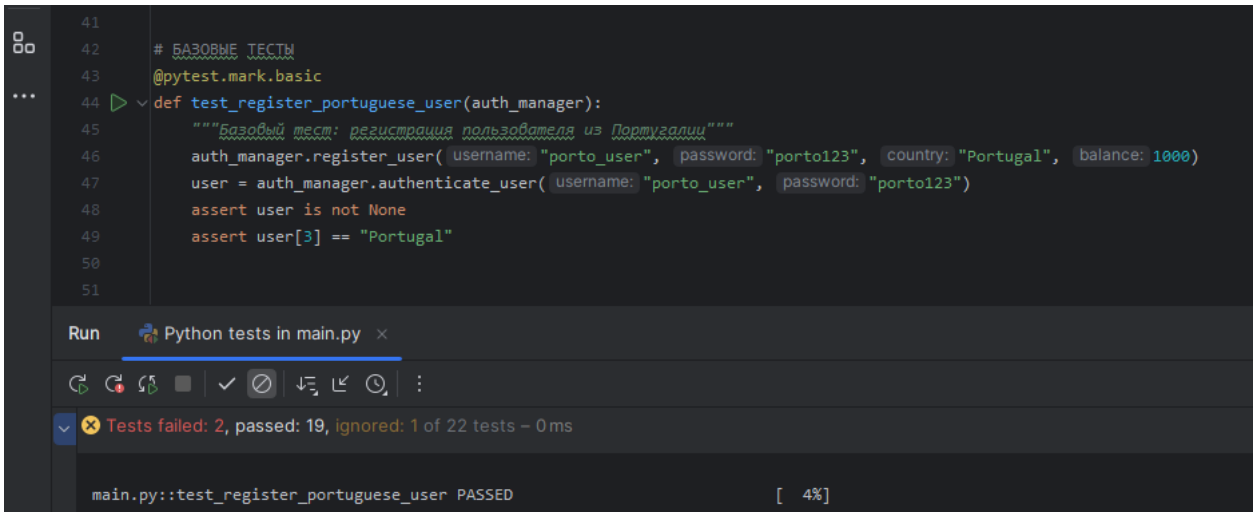
Код класса для тестирования

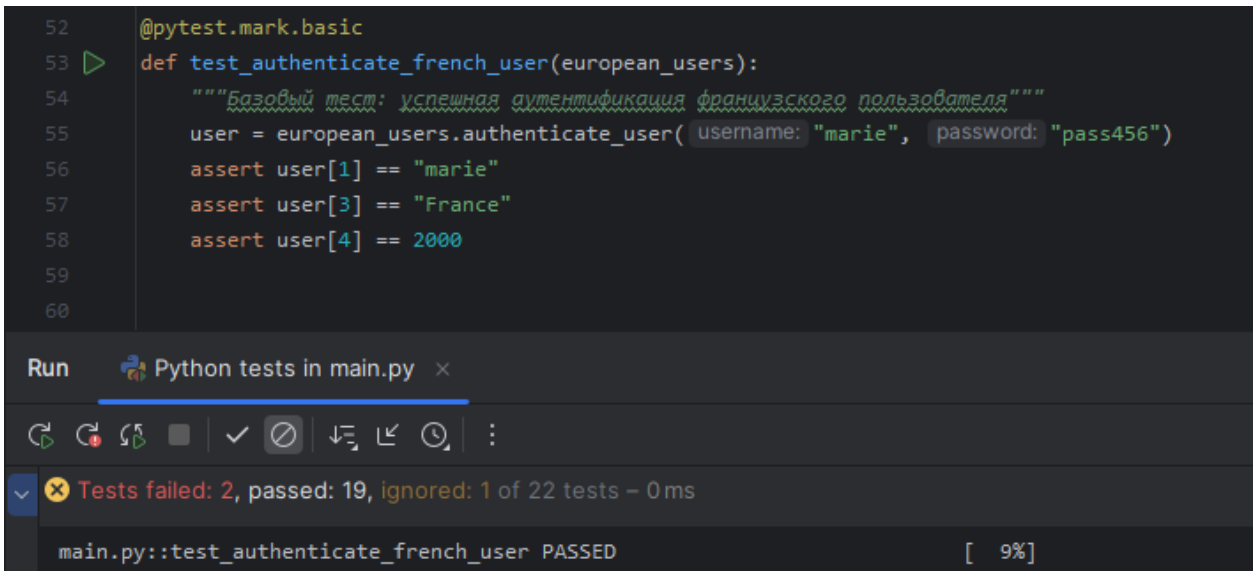
```
import sqlite3

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"""
```

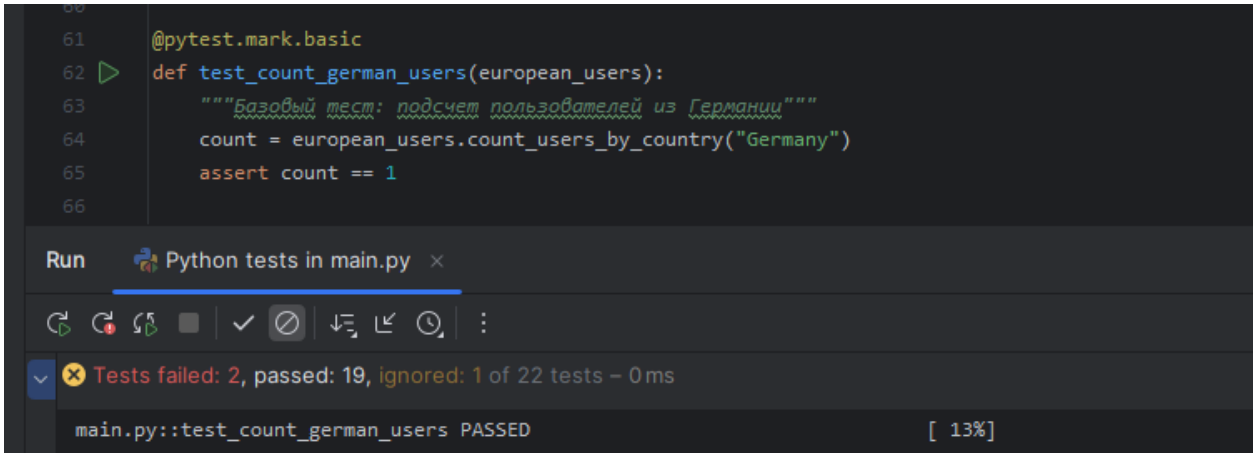
```
            UPDATE users SET balance = balance - {amount} WHERE id =
{from_user_id}
        """)
        self.connection.execute(f"""
            UPDATE users SET balance = balance + {amount} WHERE id =
{to_user_id}
        """)

    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(3 штуки)**




@pytest.mark.basic
def test_authenticate_french_user(european_users):
"""Базовый тест: успешная аутентификация французского пользователя"""
user = european_users.authenticate_user(username: "marie", password: "pass456")
assert user[1] == "marie"
assert user[3] == "France"
assert user[4] == 2000
Run
Python tests in main.py
Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms
main.py::test_authenticate_french_user PASSED [9%]

2


@pytest.mark.basic
def test_count_german_users(european_users):
"""Базовый тест: подсчет пользователей из Германии"""
count = european_users.count_users_by_country("Germany")
assert count == 1
Run
Python tests in main.py
Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms
main.py::test_count_german_users PASSED [13%]

3

2. Параметризованные тесты(3 штуки)

```python
# ПАРАМЕТРИЗОВАННЫЕ ТЕСТЫ
@pytest.mark.parametrize("country,expected_count", [
    ("Portugal", 2),
    ("France", 1),
    ("Denmark", 1),
    ("Czechia", 1),
    ("Poland", 0),
    ("Norway", 0)
])
def test_count_european_users(auth_manager, country, expected_count):
    """Параметризованный тест: подсчет пользователей по европейским странам"""
    # Подготовка данных для Европы
    auth_manager.register_user( username: "user_pt",  password: "pass1",  country: "Portugal",  balance: 1000)
    auth_manager.register_user( username: "user_fr",  password: "pass2",  country: "France",  balance: 1500)
    auth_manager.register_user( username: "user_dk",  password: "pass3",  country: "Denmark",  balance: 1200)
    auth_manager.register_user( username: "user_cz",  password: "pass4",  country: "Czechia",  balance: 1800)
    auth_manager.register_user( username: "user_de",  password: "pass5",  country: "Germany",  balance: 2000)
    auth_manager.register_user( username: "user_it",  password: "pass6",  country: "Italy",  balance: 1700)

    count = auth_manager.count_users_by_country(country)
    assert count == expected_count
```

Run Python tests in main.py

Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms

```
main.py::test_count_european_users[Portugal-2]
main.py::test_count_european_users[France-1]
main.py::test_count_european_users[Denmark-1]
main.py::test_count_european_users[Czechia-1]
main.py::test_count_european_users[Poland-0]
main.py::test_count_european_users[Norway-0]
```

PASSED [22%]PASSED [27%]PASSED [31%]PASSED [36%]PASSED [40%]

```python
def test_register_european_users(auth_manager, username, password, country, balance):
    """Параметризованный тест: регистрация пользователей из разных европейских стран"""
    auth_manager.register_user(username, password, country, balance)
    user = auth_manager.authenticate_user(username, password)
    assert user is not None
    assert user[3] == country
    assert user[4] == balance


@pytest.mark.parametrize("from_country,to_country,transfer_amount,expected_from_balance,expected_to_balance", [
    ("Portugal", "France", 300, 1200, 2300),
    ("Denmark", "Czechia", 500, 700, 2300),
    ("Germany", "Italy", 200, 2000, 1900)
])
```

Run Python tests in main.py

Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms

```
main.py::test_register_european_users[paris_user-paris_pass-France-3000]
main.py::test_register_european_users[copenhagen_user-copenhagen_pass-Denmark-1800]
main.py::test_register_european_users[prague_user-prague_pass-Czechia-2200]
```

2

```python
def test_transfer_between_european_countries(european_users, from_country, to_country, transfer_amount,
                                             expected_from_balance, expected_to_balance):
    """Параметризованный тест: перевод средств между европейскими странами"""
    # Находим пользователей по странам
    from_user = None
    to_user = None

    # Простой поиск пользователей по стране (в реальном приложении был бы метод)
    for username in ["antonio", "marie", "lars", "petr", "sophie", "giovanni"]:
        user = european_users.authenticate_user(username, password: f"pass{username[-3:]}")
        if user and user[3] == from_country:
            from_user = user
        if user and user[3] == to_country:
            to_user = user

    if from_user and to_user:
        original_from_balance = from_user[4]
        original_to_balance = to_user[4]

        european_users.transfer_balance(from_user[0], to_user[0], transfer_amount)

        # Проверяем новые балансы
        updated_from = european_users.get_user_by_id(from_user[0])
        updated_to = european_users.get_user_by_id(to_user[0])

        assert updated_from[4] == original_from_balance - transfer_amount
        assert updated_to[4] == original_to_balance + transfer_amount
```

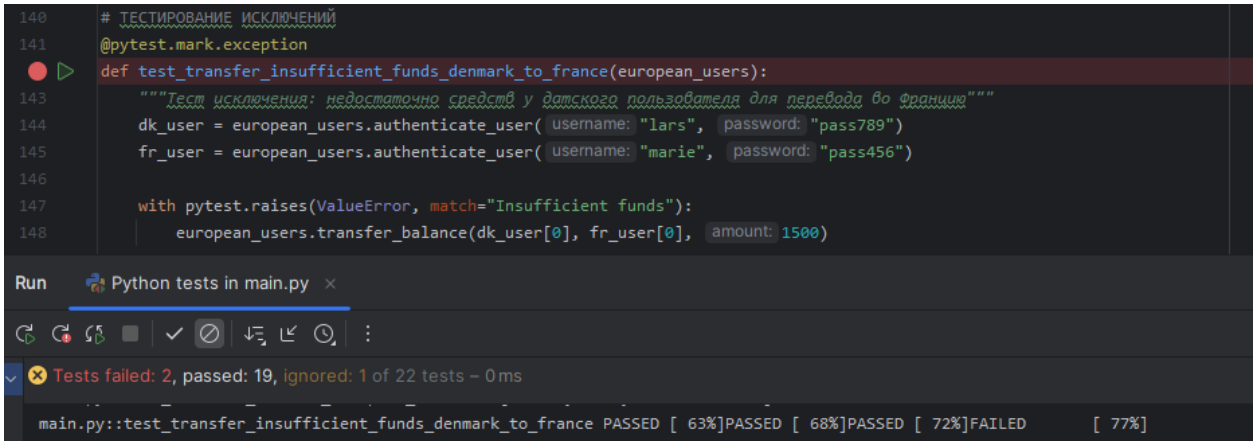
Run Python tests in main.py

Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms

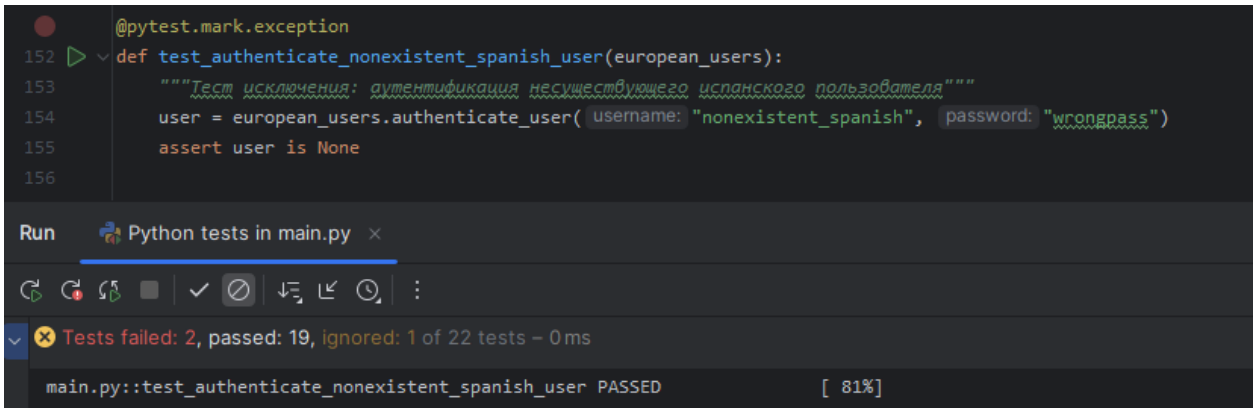
```
main.py::test_transfer_between_european_countries[Portugal-France-300-1200-2300] PASSED [ 45%]PASSED [ 50%]PASSED [ 54%]PASSED [ 59%]
main.py::test_transfer_between_european_countries[Denmark-Czechia-500-700-2300]
main.py::test_transfer_between_european_countries[Germany-Italy-200-2000-1900]
```

3

3. Тестирование исключений(2 штуки)



1



2

4. Тесты с использованием фикстур базы данных(1 штука)

```python
# Фикстура для базы данных в памяти
@pytest.fixture  2 usages
def db_connection():
    conn = sqlite3.connect(':memory:')
    yield conn
    conn.close()


@pytest.fixture  34 usages
def auth_manager(db_connection):
    return AuthManager(db_connection)


# Фикстура с тестовыми данными
@pytest.fixture  17 usages
def european_users(auth_manager):
    """Фикстура с предзаполненными данными пользователей из Европы"""
    users_data = [
        ("antonio", "pass123", "Portugal", 1500),
        ("marie", "pass456", "France", 2000),
        ("lars", "pass789", "Denmark", 1200),
        ("petr", "pass000", "Czechia", 1800),
        ("sophie", "pass111", "Germany", 2200),
        ("giovanni", "pass222", "Italy", 1700),
        ("carlos", "pass333", "Spain", 1900),
        ("anna", "pass444", "Sweden", 1600),
        ("jan", "pass555", "Netherlands", 1400),
        ("eva", "pass666", "Belgium", 1300)
    ]

    for username, password, country, balance in users_data:
        auth_manager.register_user(username, password, country, balance)

    return auth_manager
```

Run Python tests in main.py ×

Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms

```
main.py::test_count_german_users PASSED                    [ 13%]
main.py::test_count_european_users[Portugal-2]
main.py::test_count_european_users[France-1]
main.py::test_count_european_users[Denmark-1]
main.py::test_count_european_users[Czechia-1]
```

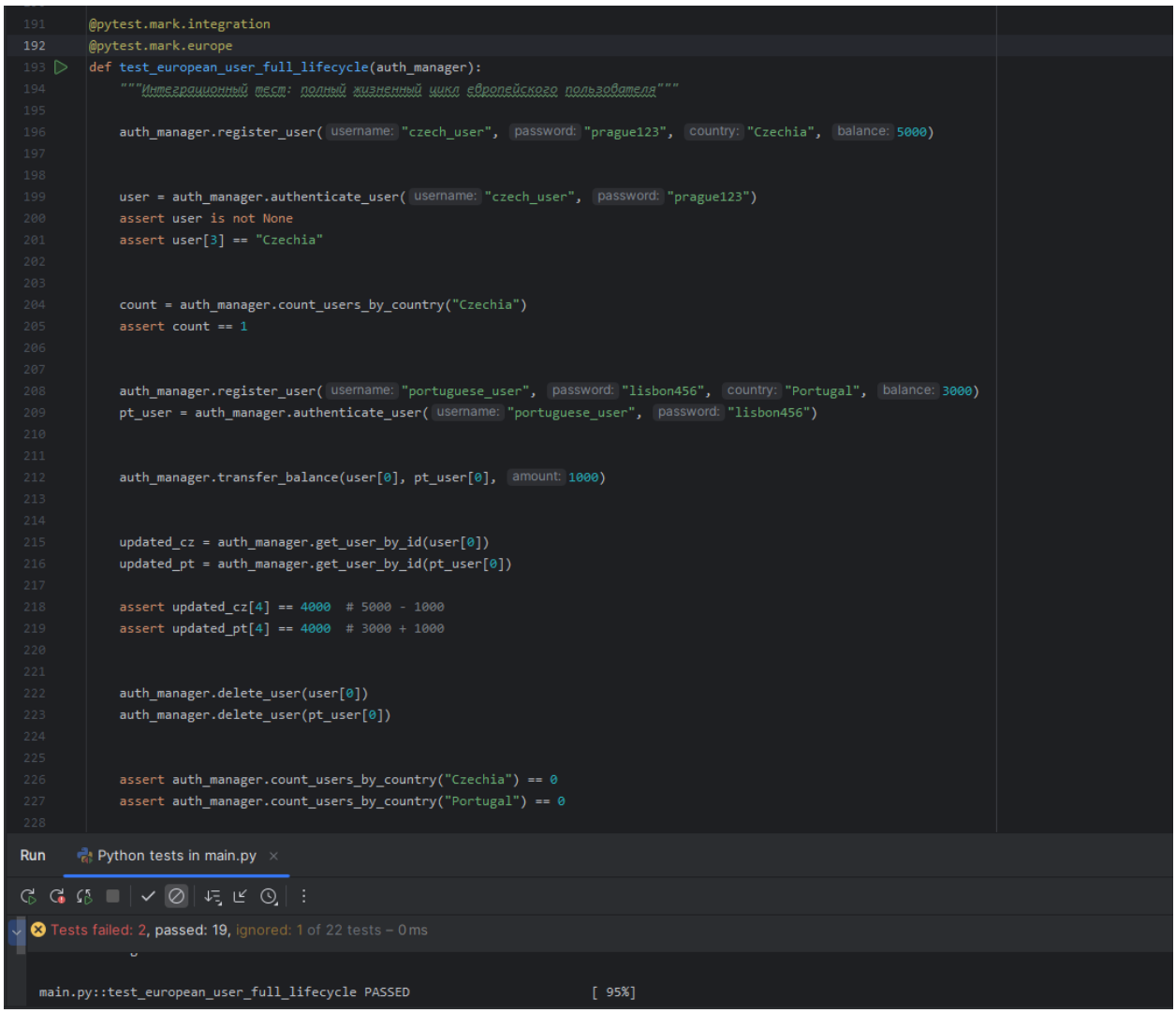
5. Тесты с метками(2 штуки)

```python
# ТЕСТЫ С МЕТКАМИ ДЛЯ ФИЛЬТРАЦИИ
@pytest.mark.slow
@pytest.mark.europe
def test_performance_all_european_countries(european_users):
    """Медленный тест: работа со всеми европейскими странами"""
    european_countries = ["Portugal", "France", "Denmark", "Czechia", "Germany",
                          "Italy", "Spain", "Sweden", "Netherlands", "Belgium"]

    total_users = 0
    for country in european_countries:
        count = european_users.count_users_by_country(country)
        total_users += count
        print(f"Users in {country}: {count}")

    assert total_users == 10  # Всего 10 пользователей в фикстуре

```

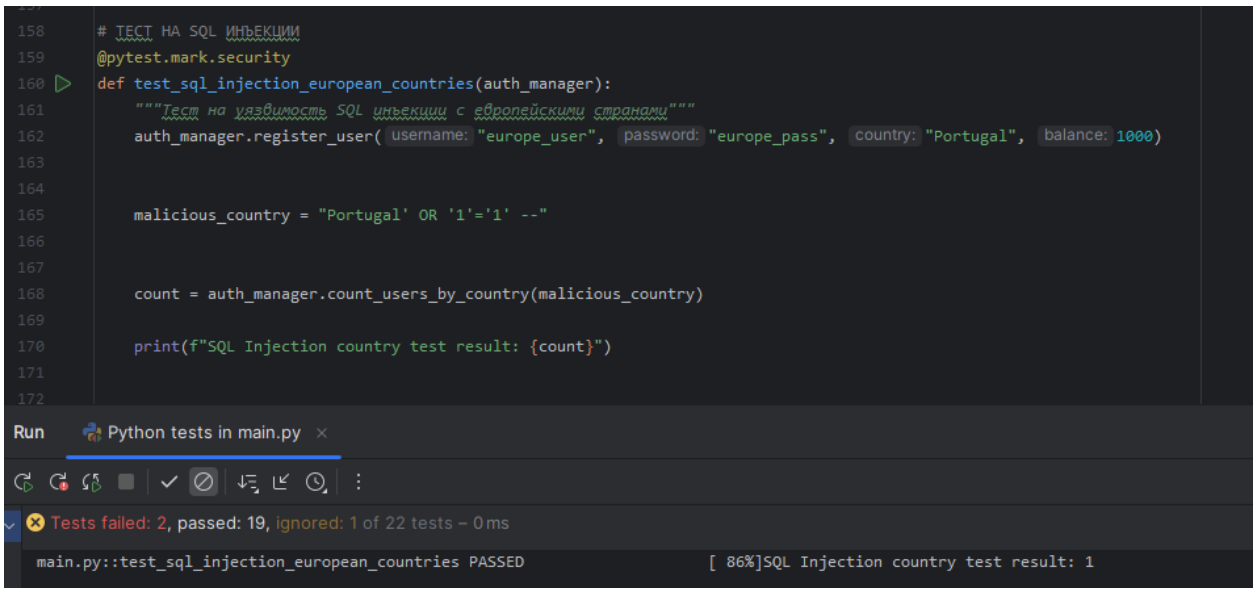
Run Python tests in main.py

Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms

```
main.py::test_performance_all_european_countries PASSED                [ 90%]Users in Portugal: 1
Users in France: 1
Users in Denmark: 1
Users in Czechia: 1
Users in Germany: 1
Users in Italy: 1
Users in Spain: 1
Users in Sweden: 1
Users in Netherlands: 1
Users in Belgium: 1
```



6. Тестирование различных векторов SQL-инъекций(1 штука)


158 # ТЕСТ НА SQL ИНЪЕКЦИИ
159 @pytest.mark.security
160 def test_sql_injection_european_countries(auth_manager):
161 """Тест на уязвимость SQL инъекции с европейскими странами"""
162 auth_manager.register_user(username: "europe_user", password: "europe_pass", country: "Portugal", balance: 1000)
163
164
165 malicious_country = "Portugal' OR '1'='1' --"
166
167
168 count = auth_manager.count_users_by_country(malicious_country)
169
170 print(f"SQL Injection country test result: {count}")
171
172
Run
Python tests in main.py
Tests failed: 2, passed: 19, ignored: 1 of 22 tests – 0 ms
main.py::test_sql_injection_european_countries PASSED [86%]SQL Injection country test result: 1

1