МИНОБР НАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Р.Е. Алексеева»

АРЗАМАССКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

Отчёт по лабораторной работе №3

по предмету:

«Тестирование ПО»

Выполнили студенты группы АЗИС 22-2

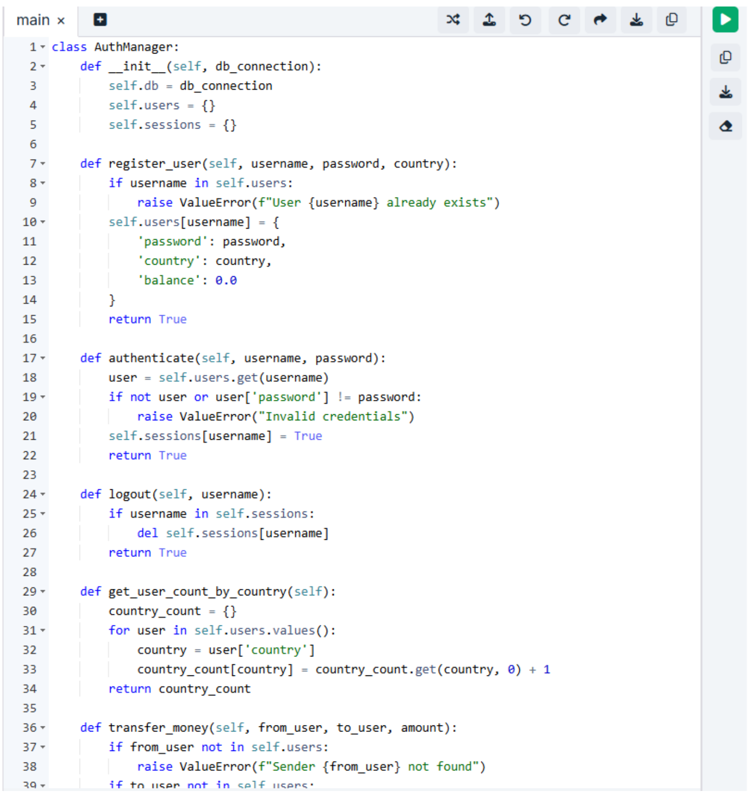
Кривоногова Е. А.

Паничкина В.Н.

Проверил: Комаров А.О.

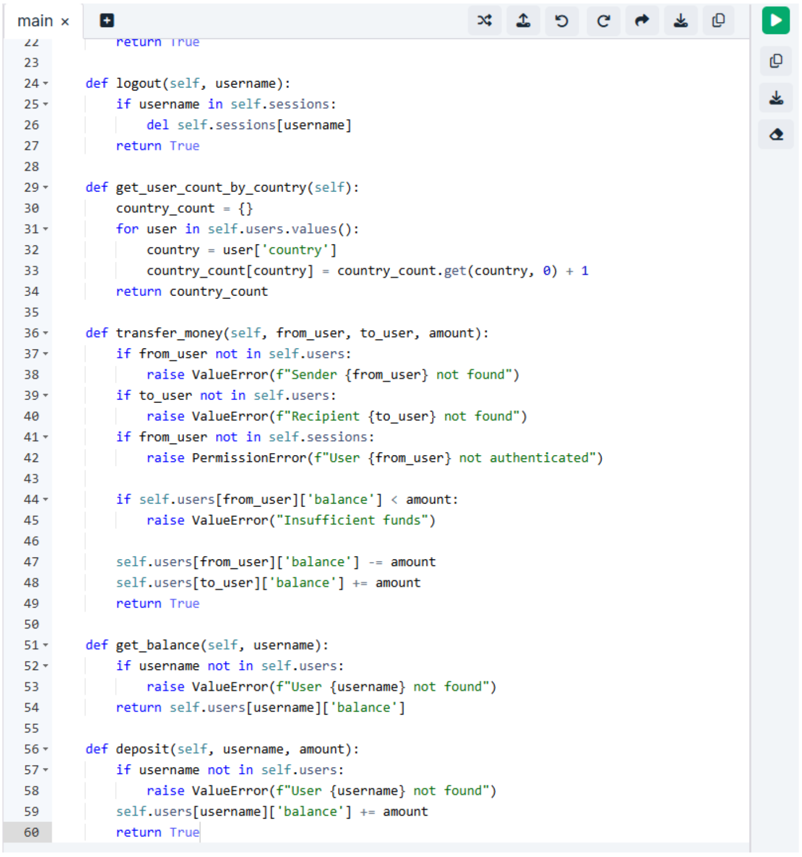
Арзамас 2025

Класс AuthManager для тестирования



```python
class AuthManager:
    def __init__(self, db_connection):
        self.db = db_connection
        self.users = {}
        self.sessions = {}

    def register_user(self, username, password, country):
        if username in self.users:
            raise ValueError(f"User {username} already exists")
        self.users[username] = {
            'password': password,
            'country': country,
            'balance': 0.0
        }
        return True

    def authenticate(self, username, password):
        user = self.users.get(username)
        if not user or user['password'] != password:
            raise ValueError("Invalid credentials")
        self.sessions[username] = True
        return True

    def logout(self, username):
        if username in self.sessions:
            del self.sessions[username]
        return True

    def get_user_count_by_country(self):
        country_count = {}
        for user in self.users.values():
            country = user['country']
            country_count[country] = country_count.get(country, 0) + 1
        return country_count

    def transfer_money(self, from_user, to_user, amount):
        if from_user not in self.users:
            raise ValueError(f"Sender {from_user} not found")
        if to_user not in self.users:
```

```
# Code execution complete.
```

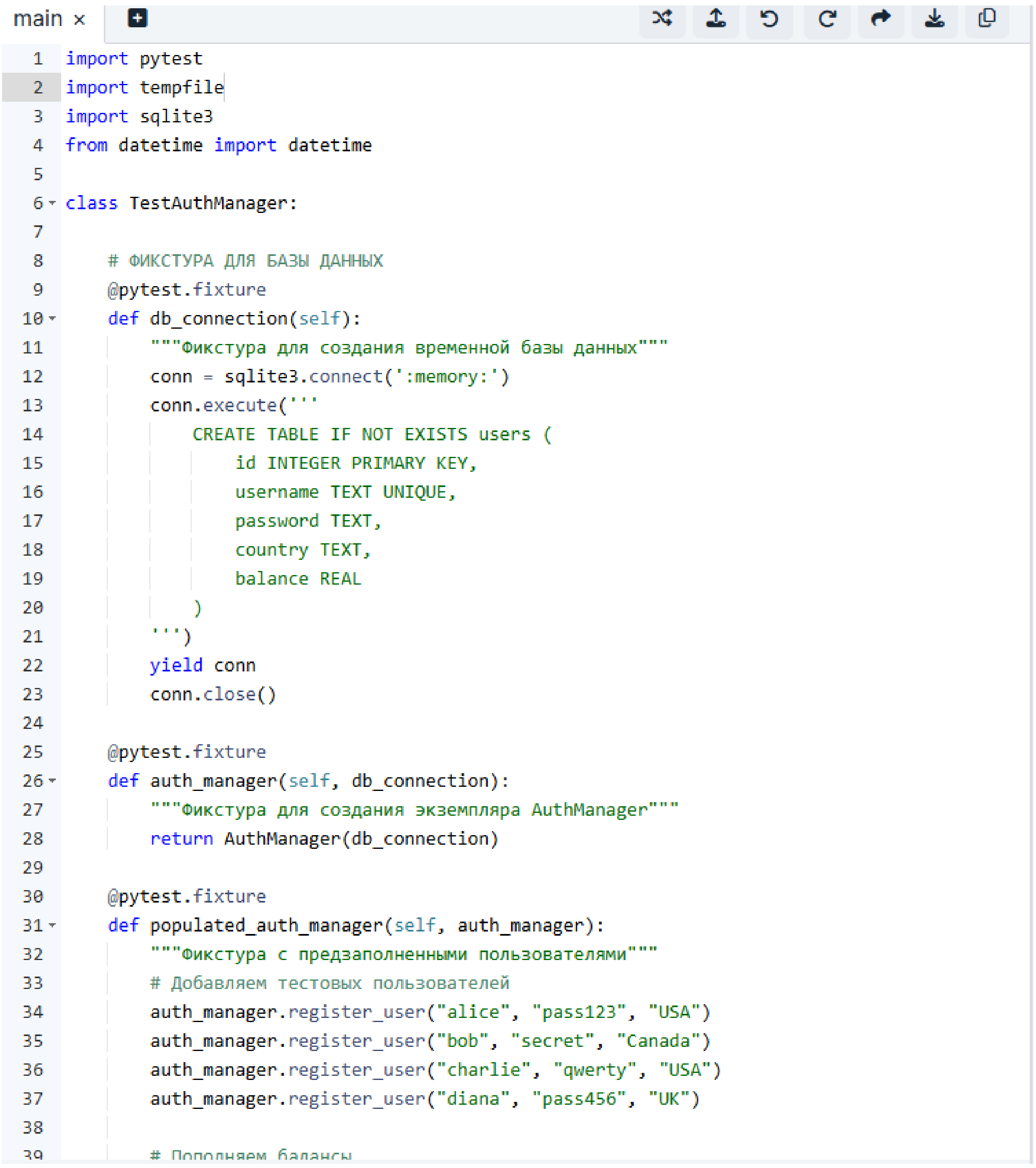


```python
        return True

    def logout(self, username):
        if username in self.sessions:
            del self.sessions[username]
        return True

    def get_user_count_by_country(self):
        country_count = {}
        for user in self.users.values():
            country = user['country']
            country_count[country] = country_count.get(country, 0) + 1
        return country_count

    def transfer_money(self, from_user, to_user, amount):
        if from_user not in self.users:
            raise ValueError(f"Sender {from_user} not found")
```

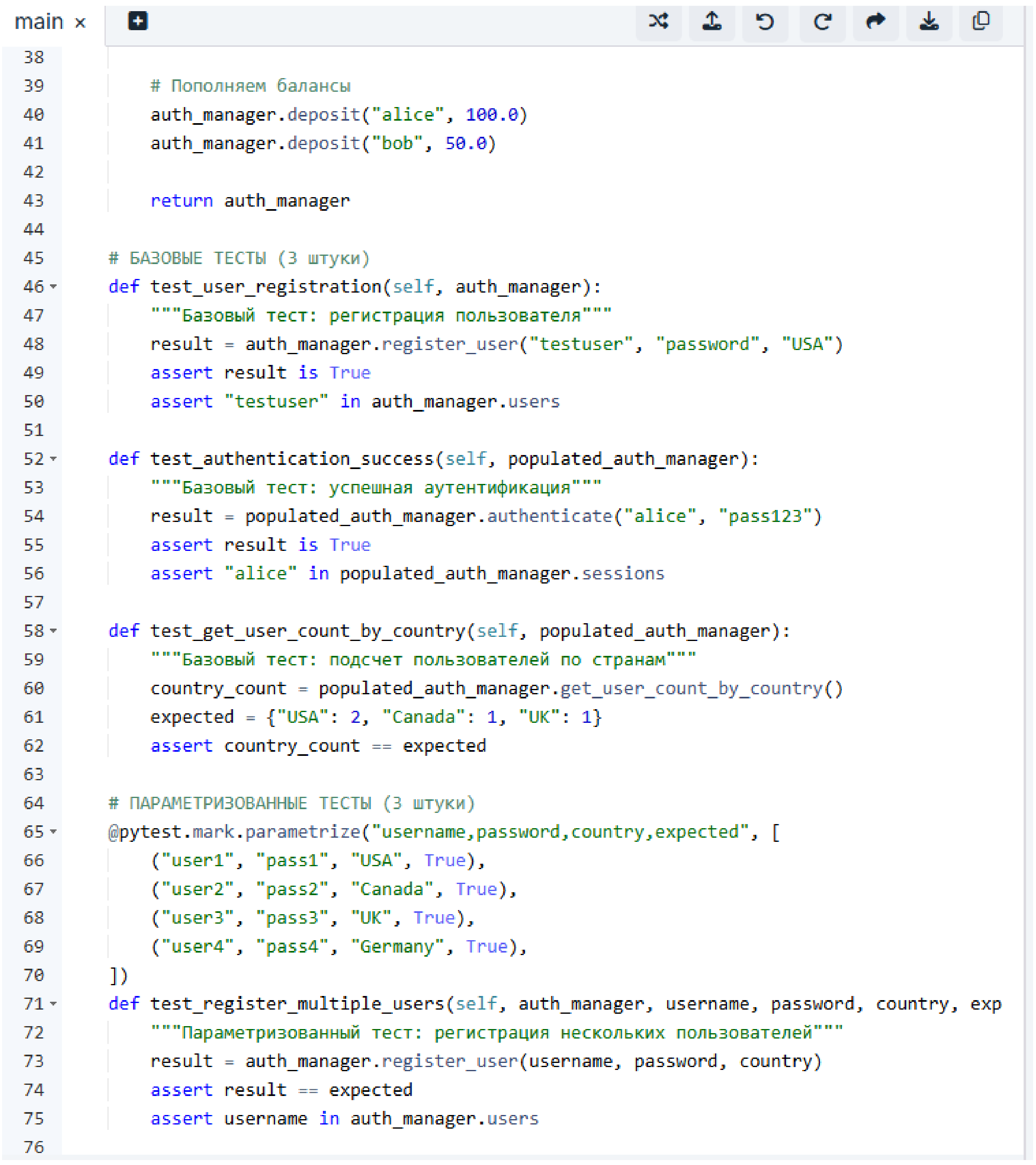
```python
        if to_user not in self.users:
            raise ValueError(f"Recipient {to_user} not found")
        if from_user not in self.sessions:
            raise PermissionError(f"User {from_user} not authenticated")

        if self.users[from_user]['balance'] < amount:
            raise ValueError("Insufficient funds")

        self.users[from_user]['balance'] -= amount
        self.users[to_user]['balance'] += amount
        return True

    def get_balance(self, username):
        if username not in self.users:
            raise ValueError(f"User {username} not found")
        return self.users[username]['balance']

    def deposit(self, username, amount):
        if username not in self.users:
            raise ValueError(f"User {username} not found")
        self.users[username]['balance'] += amount
        return True
```

```
# Code execution complete.
```

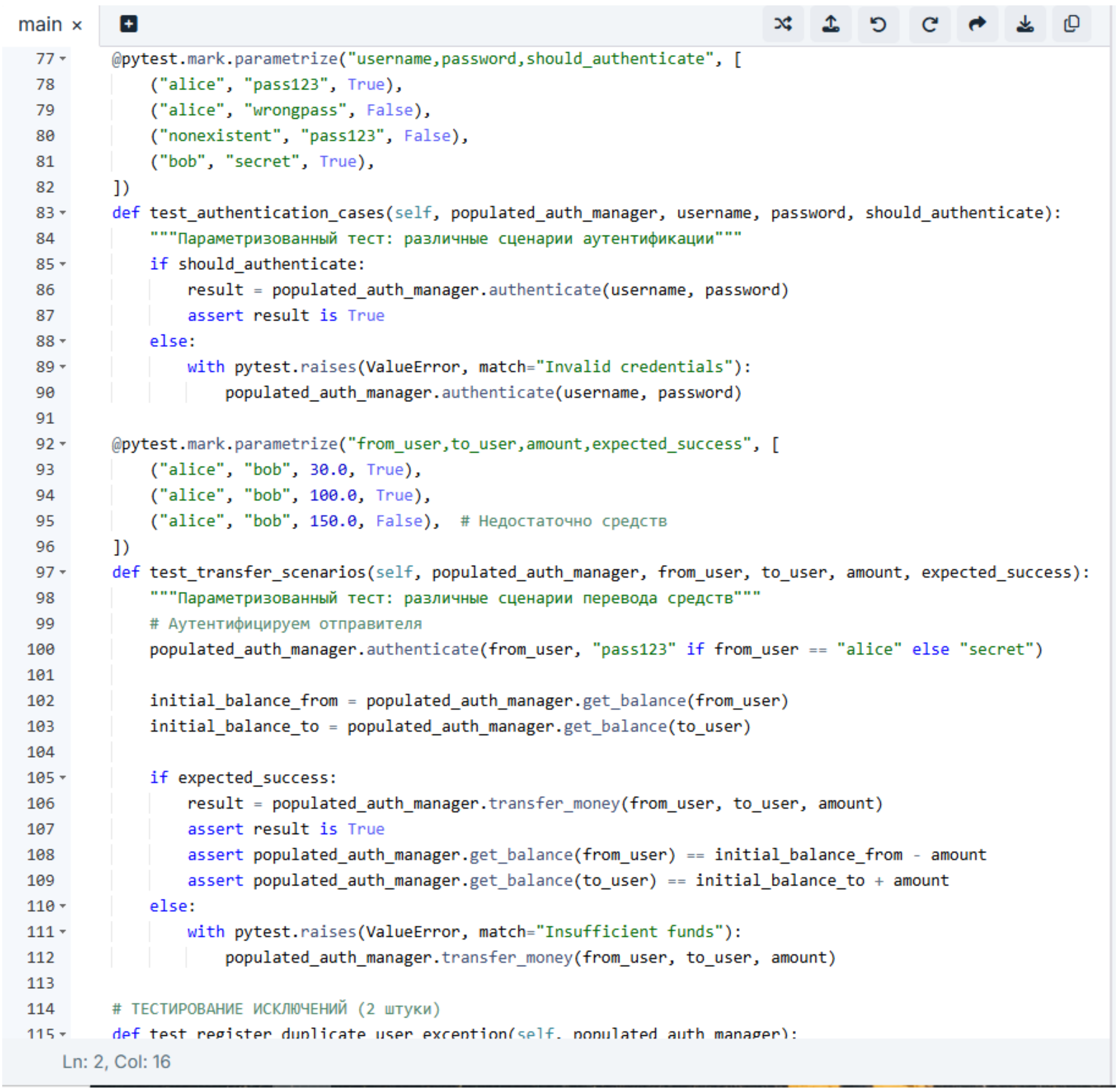
Тесты для AuthManager



```python
import pytest
import tempfile
import sqlite3
from datetime import datetime

class TestAuthManager:

    # ФИКСТУРА ДЛЯ БАЗЫ ДАННЫХ
    @pytest.fixture
    def db_connection(self):
        """Фикстура для создания временной базы данных"""
        conn = sqlite3.connect(':memory:')
        conn.execute('''
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY,
                username TEXT UNIQUE,
                password TEXT,
                country TEXT,
                balance REAL
            )
        ''')
        yield conn
        conn.close()

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

    @pytest.fixture
    def populated_auth_manager(self, auth_manager):
        """Фикстура с предзаполненными пользователями"""
        # Добавляем тестовых пользователей
        auth_manager.register_user("alice", "pass123", "USA")
        auth_manager.register_user("bob", "secret", "Canada")
        auth_manager.register_user("charlie", "qwerty", "USA")
        auth_manager.register_user("diana", "pass456", "UK")

        # Пополняем балансы
```



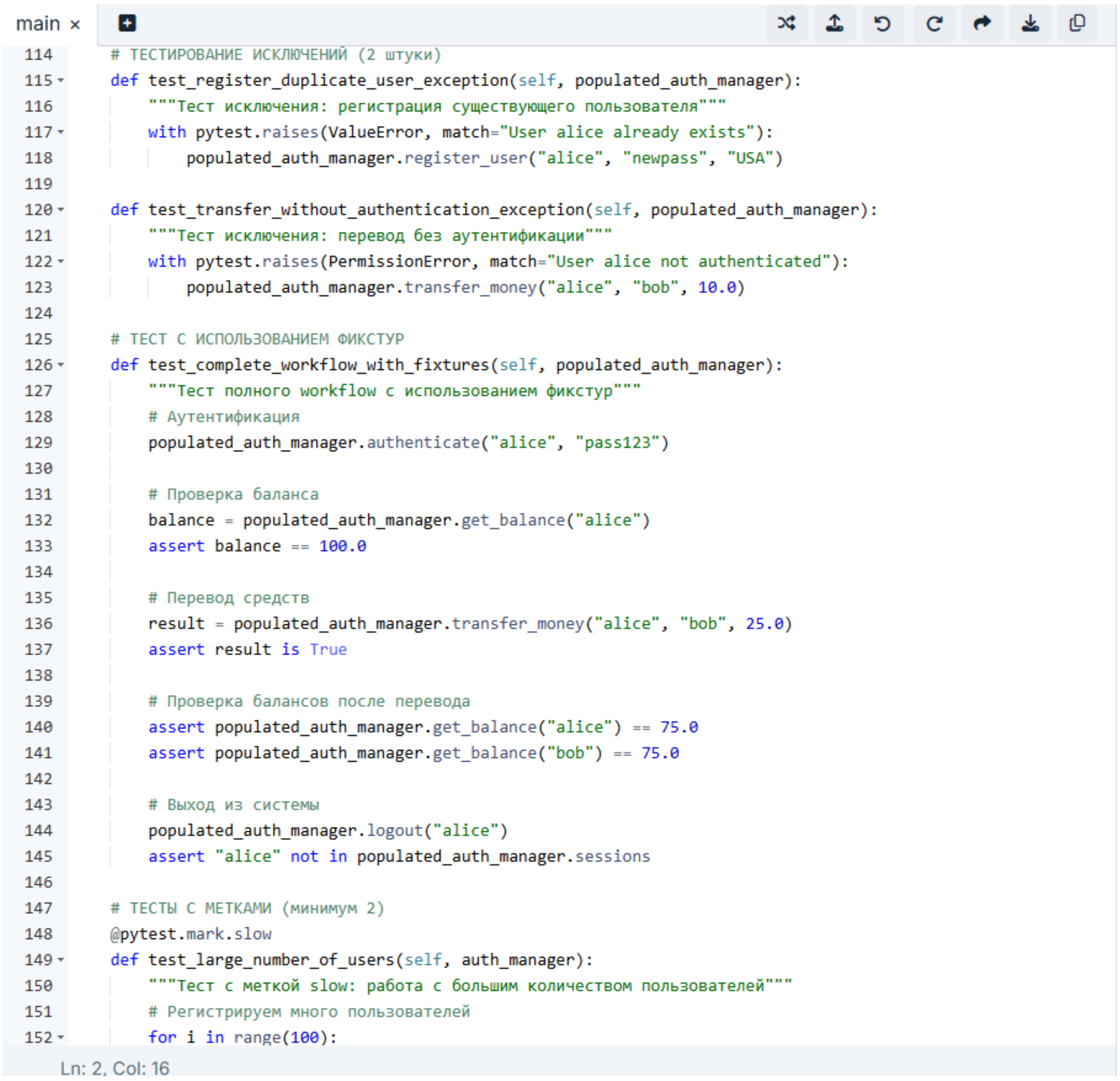
```python

    # Пополняем балансы
    auth_manager.deposit("alice", 100.0)
    auth_manager.deposit("bob", 50.0)

    return auth_manager

# БАЗОВЫЕ ТЕСТЫ (3 штуки)
def test_user_registration(self, auth_manager):
    """Базовый тест: регистрация пользователя"""
    result = auth_manager.register_user("testuser", "password", "USA")
    assert result is True
    assert "testuser" in auth_manager.users

def test_authentication_success(self, populated_auth_manager):
    """Базовый тест: успешная аутентификация"""
    result = populated_auth_manager.authenticate("alice", "pass123")
    assert result is True
    assert "alice" in populated_auth_manager.sessions

def test_get_user_count_by_country(self, populated_auth_manager):
    """Базовый тест: подсчет пользователей по странам"""
    country_count = populated_auth_manager.get_user_count_by_country()
    expected = {"USA": 2, "Canada": 1, "UK": 1}
    assert country_count == expected

# ПАРАМЕТРИЗОВАННЫЕ ТЕСТЫ (3 штуки)
@pytest.mark.parametrize("username,password,country,expected", [
    ("user1", "pass1", "USA", True),
    ("user2", "pass2", "Canada", True),
    ("user3", "pass3", "UK", True),
    ("user4", "pass4", "Germany", True),
])
def test_register_multiple_users(self, auth_manager, username, password, country, exp
    """Параметризованный тест: регистрация нескольких пользователей"""
    result = auth_manager.register_user(username, password, country)
    assert result == expected
    assert username in auth_manager.users

```



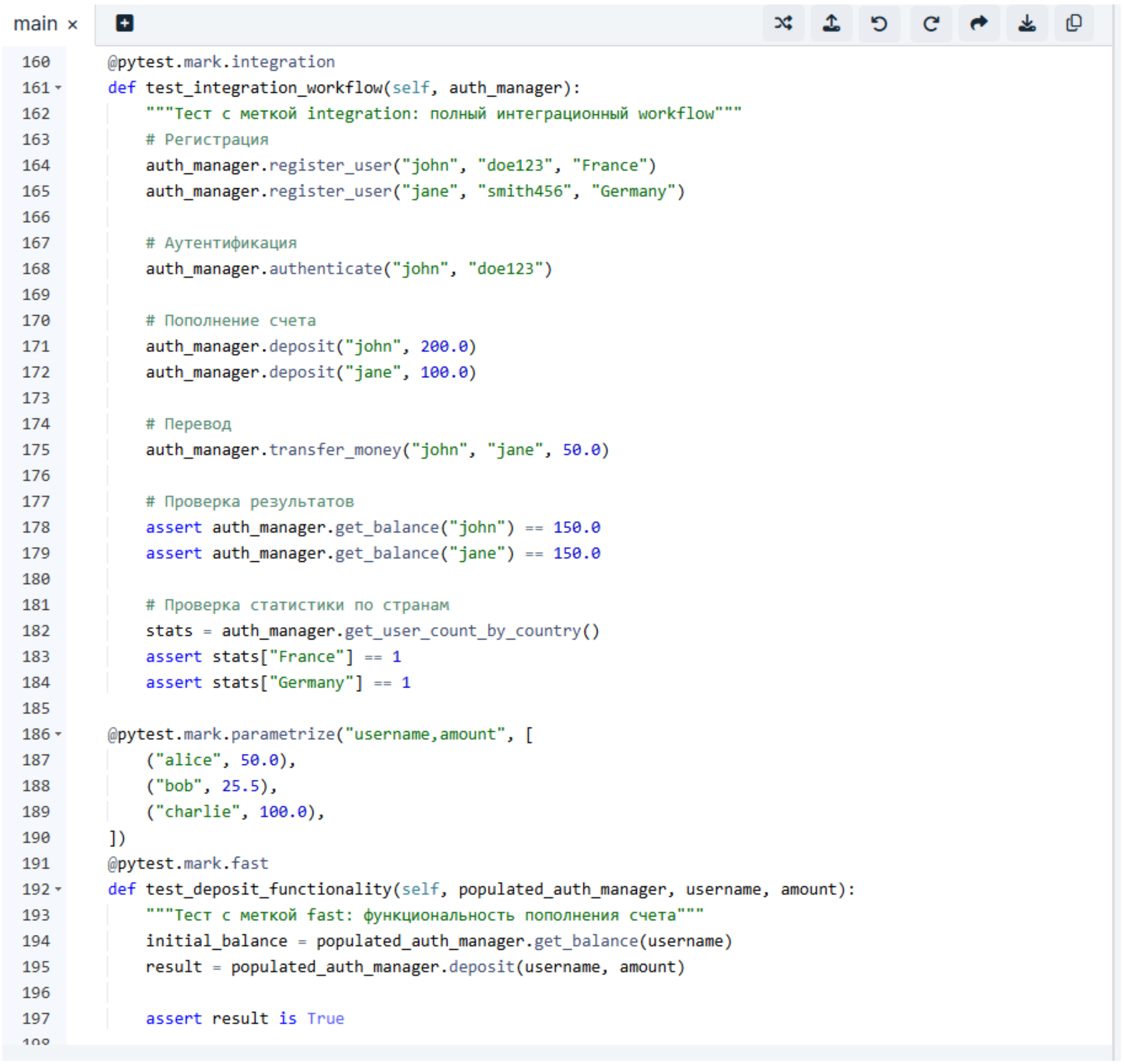
main ×

```python
@pytest.mark.parametrize("username,password,should_authenticate", [
    ("alice", "pass123", True),
    ("alice", "wrongpass", False),
    ("nonexistent", "pass123", False),
    ("bob", "secret", True),
])
def test_authentication_cases(self, populated_auth_manager, username, password, should_authenticate):
    """Параметризованный тест: различные сценарии аутентификации"""
    if should_authenticate:
        result = populated_auth_manager.authenticate(username, password)
        assert result is True
    else:
        with pytest.raises(ValueError, match="Invalid credentials"):
            populated_auth_manager.authenticate(username, password)

@pytest.mark.parametrize("from_user,to_user,amount,expected_success", [
    ("alice", "bob", 30.0, True),
    ("alice", "bob", 100.0, True),
    ("alice", "bob", 150.0, False),  # Недостаточно средств
])
def test_transfer_scenarios(self, populated_auth_manager, from_user, to_user, amount, expected_success):
    """Параметризованный тест: различные сценарии перевода средств"""
    # Аутентифицируем отправителя
    populated_auth_manager.authenticate(from_user, "pass123" if from_user == "alice" else "secret")

    initial_balance_from = populated_auth_manager.get_balance(from_user)
    initial_balance_to = populated_auth_manager.get_balance(to_user)

    if expected_success:
        result = populated_auth_manager.transfer_money(from_user, to_user, amount)
        assert result is True
        assert populated_auth_manager.get_balance(from_user) == initial_balance_from - amount
        assert populated_auth_manager.get_balance(to_user) == initial_balance_to + amount
    else:
        with pytest.raises(ValueError, match="Insufficient funds"):
            populated_auth_manager.transfer_money(from_user, to_user, amount)

# ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ (2 штуки)
def test_register_duplicate_user_exception(self, populated_auth_manager):
```

Ln: 2, Col: 16



```python
    # ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ (2 штуки)
    def test_register_duplicate_user_exception(self, populated_auth_manager):
        """Тест исключения: регистрация существующего пользователя"""
        with pytest.raises(ValueError, match="User alice already exists"):
            populated_auth_manager.register_user("alice", "newpass", "USA")

    def test_transfer_without_authentication_exception(self, populated_auth_manager):
        """Тест исключения: перевод без аутентификации"""
        with pytest.raises(PermissionError, match="User alice not authenticated"):
            populated_auth_manager.transfer_money("alice", "bob", 10.0)

    # ТЕСТ С ИСПОЛЬЗОВАНИЕМ ФИКСТУР
    def test_complete_workflow_with_fixtures(self, populated_auth_manager):
        """Тест полного workflow с использованием фикстур"""
        # Аутентификация
        populated_auth_manager.authenticate("alice", "pass123")

        # Проверка баланса
        balance = populated_auth_manager.get_balance("alice")
        assert balance == 100.0

        # Перевод средств
        result = populated_auth_manager.transfer_money("alice", "bob", 25.0)
        assert result is True

        # Проверка балансов после перевода
        assert populated_auth_manager.get_balance("alice") == 75.0
        assert populated_auth_manager.get_balance("bob") == 75.0

        # Выход из системы
        populated_auth_manager.logout("alice")
        assert "alice" not in populated_auth_manager.sessions

    # ТЕСТЫ С МЕТКАМИ (минимум 2)
    @pytest.mark.slow
    def test_large_number_of_users(self, auth_manager):
        """Тест с меткой slow: работа с большим количеством пользователей"""
        # Регистрируем много пользователей
        for i in range(100):
```



```python
@pytest.mark.integration
def test_integration_workflow(self, auth_manager):
    """Тест с меткой integration: полный интеграционный workflow"""
    # Регистрация
    auth_manager.register_user("john", "doe123", "France")
    auth_manager.register_user("jane", "smith456", "Germany")

    # Аутентификация
    auth_manager.authenticate("john", "doe123")

    # Пополнение счета
    auth_manager.deposit("john", 200.0)
    auth_manager.deposit("jane", 100.0)

    # Перевод
    auth_manager.transfer_money("john", "jane", 50.0)

    # Проверка результатов
    assert auth_manager.get_balance("john") == 150.0
    assert auth_manager.get_balance("jane") == 150.0

    # Проверка статистики по странам
    stats = auth_manager.get_user_count_by_country()
    assert stats["France"] == 1
    assert stats["Germany"] == 1

@pytest.mark.parametrize("username,amount", [
    ("alice", 50.0),
    ("bob", 25.5),
    ("charlie", 100.0),
])
@pytest.mark.fast
def test_deposit_functionality(self, populated_auth_manager, username, amount):
    """Тест с меткой fast: функциональность пополнения счета"""
    initial_balance = populated_auth_manager.get_balance(username)
    result = populated_auth_manager.deposit(username, amount)

    assert result is True
```