

Практическая работа №3

Задача: 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 нужно продемонстрировать несколько видов тестов: базовые(3 штуки), параметризованные(3 штуки), тестирование исключений(2 штуки), использование фикстур(базы данных) и меток(минимум 2). Так же должен присутствовать тест на SQL инъекции.

Решение:

Создадим код, в котором будут все тесты.

Код класса для тестирования:

```
import pytest
import sqlite3
from auth_manager import AuthManager

# Фикстуры для базы данных
@pytest.fixture
def db_connection():
    """Инициализация базы данных в памяти"""
    conn = sqlite3.connect(":memory:")
    yield conn
    conn.close()

@pytest.fixture
def auth_system(db_connection):
    """Создание системы аутентификации"""
    return AuthManager(db_connection)

@pytest.fixture
def populated_system(auth_system):
    """Система с тестовыми пользователями"""
    # Добавляем разнообразных пользователей
    test_users = [
```

```
    ("ivanov", "qwerty123", "Russia", 7500.0),  
    ("petrov", "password456", "Poland", 4200.0),  
    ("sidorova", "secret789", "Russia", 9800.0),  
    ("smith", "john2024", "USA", 3500.0),  
    ("muller", "germany1", "Germany", 6100.0)  
]
```

```
for username, password, country, balance in test_users:  
    auth_system.register_user(username, password, country, balance)  
  
return auth_system
```

Базовые тесты (3 штуки)

```
def test_database_initialization(auth_system):  
    """Проверка инициализации структуры БД"""  
    cursor = auth_system.connection.cursor()  
    cursor.execute("""  
        SELECT name FROM sqlite_master  
        WHERE type='table' AND name='users'  
    """)  
    result = cursor.fetchone()  
  
    assert result is not None, "Таблица users не создана"  
    assert result[0] == 'users', "Некорректное имя таблицы"
```

```
def test_new_user_creation(auth_system):  
    """Создание и проверка нового пользователя"""  
    auth_system.register_user("new_user", "secure_pass", "Poland", 2500.0)  
  
    cursor = auth_system.connection.cursor()  
    cursor.execute("SELECT username, country, balance FROM users WHERE  
    username = 'new_user'")  
    user = cursor.fetchone()  
  
    assert user is not None, "Пользователь не создан"  
    assert user[0] == "new_user", "Неверное имя пользователя"  
    assert user[1] == "Poland", "Неверная страна"  
    assert user[2] == 2500.0, "Неверный баланс"
```

```
def test_valid_credentials_check(auth_system):
    """Проверка валидации корректных учетных данных"""
    auth_system.register_user("valid_user", "correct_password", "France",
5000.0)

    authenticated_user = auth_system.authenticate_user("valid_user",
"correct_password")

    assert authenticated_user is not None, "Валидный пользователь не
аутентифицирован"
    assert authenticated_user[1] == 'valid_user', "Неверные данные
пользователя"

# Параметризованные тесты (3 штуки)
@pytest.mark.parametrize("login, passw, location, money", [
    ("user_alpha", "alpha123", "Sweden", 8800.0),
    ("user_beta", "beta456", "Norway", 7200.0),
    ("user_gamma", "gamma789", "Finland", 9300.0),
])
def test_bulk_user_creation(auth_system, login, passw, location, money):
    """Массовое создание пользователей с различными параметрами"""
    auth_system.register_user(login, passw, location, money)
    verification = auth_system.authenticate_user(login, passw)

    assert verification is not None, f"Пользователь {login} не создан"
    assert verification[1] == login, "Несоответствие логина"
    assert verification[3] == location, "Несоответствие страны"
    assert verification[4] == money, "Несоответствие баланса"

@pytest.mark.parametrize("input_login, input_password, expected_result", [
    ("nonexistent_user", "random_pass", None),
    ("ivanov", "wrong_password", None),
    ("", "some_pass", None),
    ("test_user", "", None),
])
def test_invalid_login_scenarios(populated_system, input_login, input_password,
expected_result):
    """Сценарии неудачных попыток входа"""
    auth_attempt = populated_system.authenticate_user(input_login,
input_password)
```

```
assert auth_attempt == expected_result, "Неверный результат  
аутентификации"
```

```
@pytest.mark.parametrize("territory, expected_number", [  
    ("Russia", 2),  
    ("Poland", 1),  
    ("USA", 1),  
    ("Germany", 1),  
    ("Japan", 0),  
    ("China", 0),  
])
```

```
def test_territorial_user_statistics(populated_system, territory,  
expected_number):  
    """Статистика пользователей по территориям"""  
    user_count = populated_system.count_users_by_country(territory)  
    assert user_count == expected_number, f"Неверное количество  
пользователей для {territory}"
```

```
# Тестирование исключений (2 штуки)
```

```
def test_funds_transfer_validation(populated_system):  
    """Валидация достаточности средств при переводе"""  
    with pytest.raises(ValueError, match="Insufficient funds"):  
        populated_system.transfer_balance(3, 1, 15000.0)
```

```
def test_invalid_recipient_handling(populated_system):  
    """Обработка несуществующего получателя"""  
    try:  
        populated_system.transfer_balance(1, 1000, 500.0)  
        pytest.fail("Ожидалась ошибка для невалидного получателя")  
    except Exception as e:  
        # Логируем тип исключения для анализа  
        print(f"Поймано исключение: {type(e).__name__}")
```

```
# Тесты с использованием фикстур
```

```
def test_account_deletion_process(populated_system):  
    """Процесс удаления учетной записи"""  
    # Проверяем наличие пользователя  
    existing_user = populated_system.get_user_by_id(2)
```

```

assert existing_user is not None, "Пользователь не существует"

# Удаляем пользователя
populated_system.delete_user(2)

# Проверяем удаление
deleted_user = populated_system.get_user_by_id(2)
assert deleted_user is None, "Пользователь не удален"

# Тесты с метками (минимум 2)
@pytest.mark.load
def test_concurrent_user_registration(auth_system):
    """Тестирование регистрации при высокой нагрузке"""
    registration_count = 150

    for index in range(registration_count):
        auth_system.register_user(
            f"stress_user_{index}",
            f"pass_{index}",
            f"country_{index % 20}",
            2000 + index * 10
        )

    country_count = auth_system.count_users_by_country("country_0")
    expected_minimum = registration_count // 20
    assert country_count >= expected_minimum, "Неверное распределение по странам"

@pytest.mark.security
def test_sql_injection_registration_vulnerability(auth_system):
    """Проверка уязвимости к SQL инъекциям при регистрации"""
    try:
        auth_system.register_user(
            "test' OR '1'='1'; DROP TABLE users; --",
            "hacked",
            "Testland",
            1000
        )
    except Exception as e:
        print(f"Ошибка при инъекции: {e}")

```

```

# Проверяем сохранность структуры БД
cursor = auth_system.connection.cursor()
cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
tables = [table[0] for table in cursor.fetchall()]

assert 'users' in tables, "Таблица users была скомпрометирована"

@pytest.mark.security
def test_authentication_bypass_attempt(auth_system):
    """Попытка обхода аутентификации через SQL инъекцию"""
    auth_system.register_user("normal_user", "normal_pass", "Canada",
4000.0)

    # Попытка инъекции для обхода пароля
    injected_result = auth_system.authenticate_user("normal_user' --",
"any_password")
    standard_result = auth_system.authenticate_user("normal_user",
"normal_pass")

    # Оба запроса должны работать из-за уязвимости
    assert injected_result is not None, "SQL инъекция не сработала"
    assert standard_result is not None, "Стандартная аутентификация не
работает"

# Пропускаемый тест
@pytest.mark.skip(reason="Требуется внешнего API")
def test_external_integration():
    """Тест интеграции с внешними системами"""
    assert False

# Дополнительные тесты безопасности
@pytest.mark.security
def test_multiple_injection_techniques(auth_system):
    """Тестирование различных техник SQL инъекций"""
    injection_methods = [
        "admin' OR '1'='1' --",
        "user'; INSERT INTO users VALUES (100, 'hacker', 'pass', 'US', 10000) --
",

```

```
"test" UNION SELECT 1, 'admin', 'pass', 'US', 10000 --"  
]
```

for method in injection_methods:

try:

auth_system.register_user(method, "test", "Test", 1000)

Базовая проверка целостности

cursor = auth_system.connection.cursor()

cursor.execute("SELECT COUNT() FROM users")*

count = cursor.fetchone()[0]

assert count >= 0, "База данных повреждена"

except Exception as ex:

print(f"Метод {method} вызвал ошибку: {ex}")

Базовые тесты(3 штуки):

```
40 ▶ def test_database_initialization(auth_system):  
41     """Проверка инициализации структуры БД"""  
42     cursor = auth_system.connection.cursor()  
43     cursor.execute("""  
44         SELECT name FROM sqlite_master  
45         WHERE type='table' AND name='users'  
46     """)  
47     result = cursor.fetchone()  
48  
49     assert result is not None, "Таблица users не создана"  
50     assert result[0] == 'users', "Некорректное имя таблицы"  
51  
52
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

3.py::test_database_initialization PASSED

```
53 ▶ def test_new_user_creation(auth_system):
54     """Создание и проверка нового пользователя"""
55     auth_system.register_user( username: "new_user", password: "secure_pass", country: "PoLand", balance: 2500.0)
56
57     cursor = auth_system.connection.cursor()
58     cursor.execute("SELECT username, country, balance FROM users WHERE username = 'new_user'")
59     user = cursor.fetchone()
60
61     assert user is not None, "Пользователь не создан"
62     assert user[0] == "new_user", "Неверное имя пользователя"
63     assert user[1] == "PoLand", "Неверная страна"
64     assert user[2] == 2500.0, "Неверный баланс"
65
66
Run Python tests in 3.py x
Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms
3.py::test_new_user_creation PASSED [ 8%]
```

2

```
67 ▶ def test_valid_credentials_check(auth_system):
68     """Проверка валидации корректных учетных данных"""
69     auth_system.register_user( username: "valid_user", password: "correct_password", country: "France", balance: 5000.0)
70
71     authenticated_user = auth_system.authenticate_user( username: "valid_user", password: "correct_password")
72
73     assert authenticated_user is not None, "Валидный пользователь не аутентифицирован"
74     assert authenticated_user[1] == 'valid_user', "Неверные данные пользователя"
75
76
Run Python tests in 3.py x
Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms
3.py::test_valid_credentials_check PASSED [ 12%]
```

2

Параметризованные тесты(3 штуки):

```
78 @pytest.mark.parametrize("login, passw, location, money", [
79     ("user_alpha", "alpha123", "Sweden", 8800.0),
80     ("user_beta", "beta456", "Norway", 7200.0),
81     ("user_gamma", "gamma789", "Finland", 9300.0),
82 ])
83 def test_bulk_user_creation(auth_system, login, passw, location, money):
84     """Массовое создание пользователей с различными параметрами"""
85     auth_system.register_user(login, passw, location, money)
86     verification = auth_system.authenticate_user(login, passw)
87
88     assert verification is not None, f"Пользователь {login} не создан"
89     assert verification[1] == login, "Несоответствие логина"
90     assert verification[3] == location, "Несоответствие страны"
91     assert verification[4] == money, "Несоответствие баланса"
92
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

3.py::test_bulk_user_creation[user_alpha-alpha123-Sweden-8800.0]
3.py::test_bulk_user_creation[user_beta-beta456-Norway-7200.0]
3.py::test_bulk_user_creation[user_gamma-gamma789-Finland-9300.0]

1

```
94 @pytest.mark.parametrize("input_login, input_password, expected_result", [
95     ("nonexistent_user", "random_pass", None),
96     ("ivanov", "wrong_password", None),
97     ("", "some_pass", None),
98     ("test_user", "", None),
99 ])
100 def test_invalid_login_scenarios(populated_system, input_login, input_password, expected_result):
101     """Сценарии неудачных попыток входа"""
102     auth_attempt = populated_system.authenticate_user(input_login, input_password)
103     assert auth_attempt == expected_result, "Неверный результат аутентификации"
104
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

3.py::test_invalid_login_scenarios[nonexistent_user-random_pass-None] PASSED [16%]PASSED [20%]PASSED [25%]
3.py::test_invalid_login_scenarios[ivanov-wrong_password-None]
3.py::test_invalid_login_scenarios[-some_pass-None]
3.py::test_invalid_login_scenarios[test_user--None]

2

```
106  @pytest.mark.parametrize("territory, expected_number", [
107      ("Russia", 2),
108      ("Poland", 1),
109      ("USA", 1),
110      ("Germany", 1),
111      ("Japan", 0),
112      ("China", 0),
113  ])
114  def test_territorial_user_statistics(populated_system, territory, expected_number):
115      """Статистика пользователей по территориям"""
116      user_count = populated_system.count_users_by_country(territory)
117      assert user_count == expected_number, f"Неверное количество пользователей для {territory}"
118
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

| | | | | | | | | |
|---|--------|--------|--------|--------|--------|--------|--------|--------|
| 3.py::test_territorial_user_statistics[Russia-2] | PASSED | [29%] | PASSED | [33%] | PASSED | [37%] | PASSED | [41%] |
| 3.py::test_territorial_user_statistics[Poland-1] | | | | | | | | |
| 3.py::test_territorial_user_statistics[USA-1] | | | | | | | | |
| 3.py::test_territorial_user_statistics[Germany-1] | | | | | | | | |
| 3.py::test_territorial_user_statistics[Japan-0] | | | | | | | | |
| 3.py::test_territorial_user_statistics[China-0] | | | | | | | | |

3

Тестирование исключений(2 штуки):

```
121  def test_funds_transfer_validation(populated_system):
122      """Валидация достаточности средств при переводе"""
123      with pytest.raises(ValueError, match="Insufficient funds"):
124          populated_system.transfer_balance(from_user_id=3, to_user_id=1, amount=15000.0)
125
126
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

| | | | | | | | | |
|---|--------|--------|--------|--------|--------|--------|--------|--|
| 3.py::test_territorial_user_statistics[China-0] | | | | | | | | |
| 3.py::test_funds_transfer_validation | PASSED | [45%] | PASSED | [50%] | PASSED | [54%] | PASSED | |

1

```
127 def test_invalid_recipient_handling(populated_system):
128     """Обработка несуществующего получателя"""
129     try:
130         populated_system.transfer_balance(from_user_id=1, to_user_id=1000, amount=500.0)
131         pytest.fail("Ожидалась ошибка для невалидного получателя")
132     except Exception as e:
133         # Логируем тип исключения для анализа
134         print(f"Поймано исключение: {type(e).__name__}")
135
Run Python tests in 3.py x
Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms
3.py::test_invalid_recipient_handling FAILED [ 75%]
3.py:126 (test_invalid_recipient_handling)
populated_system = <auth_manager.AuthManager object at 0x000001FB992E2780>

def test_invalid_recipient_handling(populated_system):
    """Обработка несуществующего получателя"""
    try:
        populated_system.transfer_balance(1, 1000, 500.0)
>       pytest.fail("Ожидалась ошибка для невалидного получателя")
E       Failed: Ожидалась ошибка для невалидного получателя

3.py:131: Failed
```

2

Тесты с использованием фикстур базы данных(1 штука):

```
138 def test_account_deletion_process(populated_system):
139     """Процесс удаления учетной записи"""
140     # Проверяем наличие пользователя
141     existing_user = populated_system.get_user_by_id(2)
142     assert existing_user is not None, "Пользователь не существует"
143
144     # Удаляем пользователя
145     populated_system.delete_user(2)
146
147     # Проверяем удаление
148     deleted_user = populated_system.get_user_by_id(2)
149     assert deleted_user is None, "Пользователь не удален"
150
Run Python tests in 3.py x
Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms
3.py::test_account_deletion_process PASSED [ 79%]
```

1

Тесты с метками(2 штуки):

```
153 @pytest.mark.load
154 def test_concurrent_user_registration(auth_system):
155     """Тестирование регистрации при высокой нагрузке"""
156     registration_count = 150
157
158     for index in range(registration_count):
159         auth_system.register_user(
160             username: f"stress_user_{index}",
161             password: f"pass_{index}",
162             country: f"country_{index % 20}",
163             2000 + index * 10
164         )
165
166     country_count = auth_system.count_users_by_country("country_0")
167     expected_minimum = registration_count // 20
168     assert country_count >= expected_minimum, "Неверное распределение по странам"
169
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

3.py::test_concurrent_user_registration PASSED [83%]

1

```
172 def test_sql_injection_registration_vulnerability(auth_system):
173     """Проверка уязвимости к SQL инъекциям при регистрации"""
174     try:
175         auth_system.register_user(
176             username: "test' OR '1'='1'; DROP TABLE users; --",
177             password: "hacked",
178             country: "Testland",
179             balance: 1000
180         )
181     except Exception as e:
182         print(f"Ошибка при инъекции: {e}")
183
184     # Проверяем сохранность структуры БД
185     cursor = auth_system.connection.cursor()
186     cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
187     tables = [table[0] for table in cursor.fetchall()]
188
189     assert 'users' in tables, "Таблица users была скомпрометирована"
190
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

3.py::test_sql_injection_registration_vulnerability PASSED [87%]Ошибка при инъекции: near ";": syntax error

2

Тестирование различных векторов SQL-инъекций(1 штука):

```
192 @pytest.mark.security
193 def test_authentication_bypass_attempt(auth_system):
194     """Попытка обхода аутентификации через SQL инъекцию"""
195     auth_system.register_user( username= "normal_user", password= "normal_pass", country= "Canada", balance= 4000.0)
196
197     # Попытка инъекции для обхода пароля
198     injected_result = auth_system.authenticate_user( username= "normal_user' --", password= "any_password")
199     standard_result = auth_system.authenticate_user( username= "normal_user", password= "normal_pass")
200
201     # Оба запроса должны работать из-за уязвимости
202     assert injected_result is not None, "SQL инъекция не сработала"
203     assert standard_result is not None, "Стандартная аутентификация не работает"
204
205
```

Run Python tests in 3.py x

Tests failed: 1, passed: 22, ignored: 1 of 24 tests – 5 ms

3.py::test_authentication_bypass_attempt PASSED [91%]