ПРИЛОЖЕНИЕ Б. СЦЕНАРИЙ И РЕЗУЛЬТАТЫ ТЕСТОВЫХ ИСПЫТАНИЙ

АННОТАЦИЯ

1. Сценарий тестов

В данном разделе представлен сценарий тестов

2. Тесты jmetr

В данном разделе представлены тесты jmetr

3. Ручное тестирование

В данном разделе представлено ручное тестирование

4. Unit тесты

В данном разделе представлены unit тесты

4.1 Результат Unit тестов

В данном разделе представлен результат unit тестов

4.2 Unit тесты исходный код

В данном разделе представлен исходный код unit тестов

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данном разделе документации описаны тесты программы l_admin

1.Сценарий тестов

Рисунок 1-схема тестирования

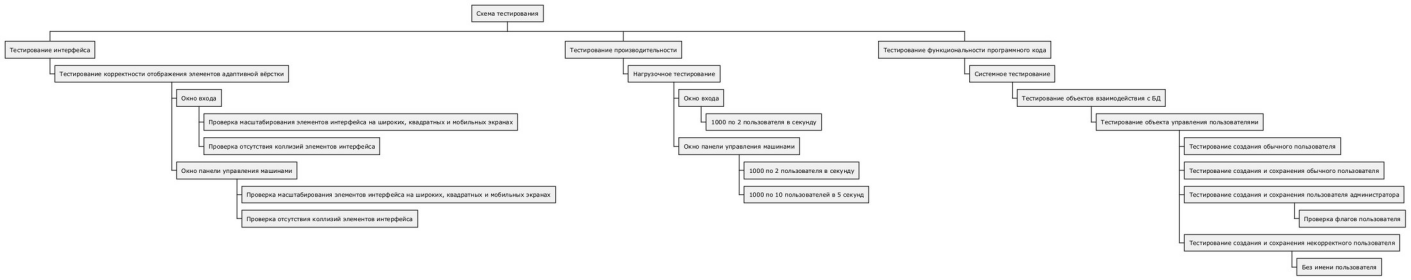

Рисунок 1-схема тестирования

2.Тесты jmetr

Тесты jmetr находятся l_admin/test/result/index.html

3.Ручное тестирование

№	Предпосылки	Приоритетность	Шаги тестирования	Данные тестирования	Ожидаемый результат	Фактический результат
1	Проверка отображения элементов окно входа	C	Открытие приложения по url /, изменение размера окна	Не предусмотрены	Интерфейс отображается корректно	Интерфейс отображается корректно
2	Проверка отображения элементов окно панели управления	C	Открытие приложения по url /login, изменение размера окна	Не предусмотрены	Данные успешно изменены и отображаются корректно	Данные успешно изменены и отображаются корректно

4.Unit тесты

4.1 Результат Unit тестов

```
(env) [we@wePC l_admin]$ python manage.py test API
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
Ran 5 tests in 0.807s

OK
Destroying test database for alias 'default'...
(env) [we@wePC l_admin]$ python manage.py test
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
Ran 5 tests in 0.821s

OK
Destroying test database for alias 'default'...
```

4.2 Unit тесты исходный код

l_admin/API/tests.py

```
from django.test import TestCase

from django.core.exceptions import ObjectDoesNotExist

from django.contrib.auth.hashers import make_password

from .models import User

class TC_UserManager(TestCase):
    def setUp(self):
        self.userManager = User.objects

    def test_CreateUser_regular_login_myuser_password_123(self):
        expected_login = "myuser"
        expected_password = "123"
        expected_is_staff = False
        expected_is_superuser = False

        self.userManager.create_user(login=expected_login, password=expected_password)
        user = self.userManager.get(login=expected_login)
        salt = user.password.split('$')[2]
        actual_login = user.login
        actual_password = user.password
        actual_is_staff = user.is_staff
```

```
    actual_is_superuser = user.is_superuser

    self.assertEqual(expected_login, actual_login)
    self.assertEqual(make_password(expected_password, salt=salt), actual_password)
    self.assertEqual(expected_is_staff, actual_is_staff)
    self.assertEqual(expected_is_superuser, actual_is_superuser)

def test_CreateUser_must_have_a_login(self):
    expected_password = "123"
    expected_exception_message = "Users must have a login"

    with self.assertRaises(ValueError) as catch:
        self.userManager.create_user(login=None, password=expected_password)

    actual_exception_message = str(catch.exception)
    self.assertEqual(expected_exception_message, actual_exception_message)

def test_CreateUser_when_commit_is_false_is_not_commited(self):
    expected_exception_message = "User matching query does not exist."

    login = "myuser"
    password = "123"
    self.userManager.create_user(login=login, password=password, commit=False)
    with self.assertRaises(ObjectDoesNotExist) as catch:
        self.userManager.get(login=login)
    actual_exception_message = str(catch.exception)
```

```
        self.assertEqual(expected_exception_message, actual_exception_message)

    def test_CreateSuperuser_superuser_login_mysuperuser_password_123(self):
        expected_login = "mysuperuser"
        expected_password = "123"
        expected_is_staff = True
        expected_is_superuser = True

        self.userManager.create_superuser(login=expected_login,
password=expected_password)
        superuser = self.userManager.get(login=expected_login)
        salt = superuser.password.split('$')[2]
        actual_login = superuser.login
        actual_password = superuser.password
        actual_is_staff = superuser.is_staff
        actual_is_superuser = superuser.is_superuser

        self.assertEqual(expected_login, actual_login)
        self.assertEqual(make_password(expected_password, salt=salt), actual_password)
        self.assertEqual(expected_is_staff, actual_is_staff)
        self.assertEqual(expected_is_superuser, actual_is_superuser)

    def test_CreateSuperuser_must_have_a_login(self):
        expected_password = "123"
        expected_exception_message = "Users must have a login"

        with self.assertRaises(ValueError) as catch:
```

```
            self.userManager.create_superuser(login=None, password=expected_password)

        actual_exception_message = str(catch.exception)
        self.assertEqual(expected_exception_message, actual_exception_message)


class TC_User(TestCase):
    pass
```