Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

ITMO University

ЛАБОРАТОРНАЯ РАБОТА №2

По дисциплине Тестирование программного обеспечения

Тема работы Интеграционное тестирование

Обучающийся: Белисов Глеб Андреевич

Факультет ПИН

Направление подготовки 11.03.02 Инфокоммуникационные технологии и системы связи

Образовательная программа Программирование в инфокоммуникационных системах

2025

Описание проекта

Для выполнения работы был выбран открытый проект FastAPI RealWorld Example App — полнофункциональное backend-приложение на FastAPI с поддержкой регистрации пользователей, аутентификации, CRUD-операций, взаимодействия с базой данных и формированием JWT-токенов. Архитектура проекта модульная, что делает его подходящим для интеграционного тестирования.

Архитектура приложения включает:

- API-уровень (FastAPI роуты)
- Репозитории (Data Access Layer)
- Модели домена и схемы валидации
- База данных PostgreSQL
- Сервис JWT для токенизации
- Конфигурационный модуль с окружениями

Тесты

1. **test_register_user — тест регистрации пользователя:**

Проверяет корректность работы маршрута /api/users. Взаимодействие API, service users, repository и БД. Формирование и валидацию входных данных, а также успешную вставку пользователя в таблицу users.

```python
@pytest.mark.asyncio
async def test_register_user():
    async with LifespanManager(app):
        async with AsyncClient(base_url="http://test", app=app) as ac:
            response = await ac.post("/api/users", json={
                "user": {
                    "username": "testuser",
                    "email": "testuser@example.com",
                    "password": "password123"
                }
            })
            assert response.status_code == 201
            data = response.json()
            assert data["user"]["username"] == "testuser"
            assert data["user"]["email"] == "testuser@example.com"
```

2. **test_login_user — тест входа пользователя**

Проверяет корректность генерации JWT токена, работу цепочки API - сервис авторизации - репозиторий - проверка пароля – JWT. Работу проверки пароля (bcrypt). Это базовый сценарий для всех последующих тестов, так как для публикации статей, фида и лайков нужен токен.

```python
@pytest.mark.anyio
async def test_login_user():
    async with LifespanManager(app):
        async with AsyncClient(app=app, base_url="http://test") as ac:

            await ac.post("/api/users", json={
                "user": {
                    "username": "testlogin",
                    "email": "login@example.com",
                    "password": "password123"
                }
            })

            response = await ac.post("/api/users/login", json={
                "user": {
                    "email": "login@example.com",
                    "password": "password123"
                }
            })
            assert response.status_code == 200
            data = response.json()
            assert "token" in data["user"]
```

3. **test_create_article — создание статьи**

Проверяет весь путь публикации статьи API - сервис статей - slug-генерация - проверка уникальности - SQL-вставка - возврат данных. Работу middleware авторизации (проверку токена) и связь между таблицами users и articles. Это критически важная интеграция, поскольку ошибка в создании статей делает проект нефункциональным.

```python
@pytest.mark.anyio
async def test_create_article():
    async with LifespanManager(app):
        async with AsyncClient(app=app, base_url="http://test") as ac:

            await ac.post("/api/users", json={
                "user": {"username": "author", "email": "author@example.com", "password": "password123"}
            })
            login_resp = await ac.post("/api/users/login", json={
                "user": {"email": "author@example.com", "password": "password123"}
            })
            token = login_resp.json()["user"]["token"]

            response = await ac.post(
                "/api/articles",
                json={
                    "article": {"title": "Test Article", "description": "Desc", "body": "Content"}
                },
                headers={"Authorization": f"Token {token}"}
            )
            assert response.status_code == 201
            data = response.json()
            assert data["article"]["title"] == "Test Article"
```

4. **test_feed — получение фида статей**

Проверяет работу SQL-запроса, который получает статьи авторов, на которых подписан пользователь. Корректность join-ов в SQL и репозитории. Возврат списка в правильном формате.

```python
@pytest.mark.anyio
async def test_feed():
    async with LifespanManager(app):
        async with AsyncClient(app=app, base_url="http://test") as ac:

            await ac.post("/api/users", json={"user": {"username": "feeduser", "email": "feed@example.com", "password": "password123"}})
            login_resp = await ac.post("/api/users/login", json={"user": {"email": "feed@example.com", "password": "password123"}})
            token = login_resp.json()["user"]["token"]

            response = await ac.get("/api/articles/feed", headers={"Authorization": f"Token {token}"})
            assert response.status_code == 200
            assert "articles" in response.json()
```

5. **test_favorite_article — избранные статьи**

Проверяет добавление статьи в избранное, работу связывающей таблицы favorites, генерацию корректного ответа с флагом favorited = True, также то, что повторный запрос на статью возвращает изменённое состояние.

```python
@pytest.mark.anyio
async def test_favorite_article():
    async with LifespanManager(app):
        async with AsyncClient(app=app, base_url="http://test") as ac:

            await ac.post("/api/users", json={"user": {"username": "favuser", "email": "fav@example.com", "password": "password123"}})
            login_resp = await ac.post("/api/users/login", json={"user": {"email": "fav@example.com", "password": "password123"}})
            token = login_resp.json()["user"]["token"]

            create_resp = await ac.post(
                "/api/articles",
                json={"article": {"title": "Fav Article", "description": "Desc", "body": "Content"}},
                headers={"Authorization": f"Token {token}"}
            )
            slug = create_resp.json()["article"]["slug"]

            fav_resp = await ac.post(f"/api/articles/{slug}/favorite", headers={"Authorization": f"Token {token}"})
            assert fav_resp.status_code == 200
            assert fav_resp.json()["article"]["favorited"] is True
```

Результаты

```
(venv) PS C:\Users\belga\Documents\Testt\fastapi-realworld-example-app> python -m pytest -v -n0
============================================= test session starts =============================================
platform win32 -- Python 3.13.2, pytest-9.0.2, pluggy-1.6.0 -- C:\Users\belga\Documents\Testt\fastapi-realworld-example-app\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\belga\Documents\Testt\fastapi-realworld-example-app
configfile: pyproject.toml
testpaths: tests
plugins: anyio-4.12.0, asyncio-1.3.0, cov-7.0.0, xdist-3.8.0
asyncio: mode=Mode.AUTO, debug=False, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 5 items

tests/test_integration.py::test_register_user PASSED                                                  [ 20%]
tests/test_integration.py::test_login_user PASSED                                                     [ 40%]
tests/test_integration.py::test_create_article PASSED                                                 [ 60%]
tests/test_integration.py::test_feed PASSED                                                           [ 80%]
tests/test_integration.py::test_favorite_article PASSED                                               [100%]
```

Тесты покрывают всего ~4% кода и все проходят успешно. В ходе работы была выполнена интеграция FastAPI, asyncpg, PostgreSQL, пользовательских репозиториев и JWT-сервиса. Интеграционные тесты подтвердили корректность взаимодействия модулей и обеспечили уверенность в том, что модули корректно взаимодействуют.