Интеллектуальные информационные системы

Практическое задание 6

Кафедра информатики
Институт кибербезопасности и цифровых технологий
РТУ МИРЭА

Добавление тестов к API сервису «Библиотека»

В продолжении предыдущий практики мы создадим тесты для проверки работоспособности нашего API.

Мы будем использовать библиотеку Pytest вот ссылка на текущую версию документации https://docs.pytest.org/en/stable/

1. Зайдите в консоле в папку проекта с прошлой практической работы и запустите виртуальное окружение.

2. Откройте проект в Visual studio Code.

3. Для примера работы с фикстурами и понимания базовой логики работы с тестами создайте в папке **test** файл **example_test.py** со следующим содержанием:

```python
import pytest
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.sql import text

# класс для хранения состояние (любых данных) передающихся между тестами
class SharedData():
    def __init__(self, db):
        self.session = db # сохраняем сессия к базе данных

    def get_session(self):
        return self.session

# Создаём фикстуру для базы данных
@pytest.fixture(scope="class")
def fixture_create_db(request):
    engine = create_engine("sqlite:///:memory:") #используем бд в памяти
    SessionLocal = sessionmaker(bind=engine)

    session = SessionLocal()

    session.execute(text("CREATE TABLE test (id INTEGER PRIMARY KEY, name TEXT)"))
    # Передаём сессию в тест
    request.cls.testData = SharedData(session)

    yield session

    session.close()  # Очистка после теста

@pytest.mark.usefixtures("fixture_create_db")
class TestSuite():
    def test_check_record(self):
        print( "start test")
        # Используем фикстуру db_session
        session = self.testData.get_session()
        session.execute(text("INSERT INTO test (name) VALUES ('Alice')"))

        result = session.execute(text("SELECT * FROM test")).fetchall()
        assert len(result) == 1
        assert result[0][1] == 'Alice'
```

Добавление тестов к API сервису «Библиотека»

4. Изучите содержимое теста и логику работы фикстуры.

5. Запустите тест в консоли командой **pytest -vs app\tests\example_test.py**

6. Добавьте дополнительный тестовый метод в тестовый класс

```
def test_count_records(self):
    session = self.testData.get_session()
    result = session.execute(text("SELECT * FROM test")).fetchall()
    assert len(result) == 3
```

7. Повторно запустите тест, изучите что изменилось и почему.

Добавление тестов к API сервису «Библиотека»

8. Добавьте библиотеку **httpx** в файл **requirements.txt**

9. Выполните заново команду **pip install -r requirements.txt**

10. Создайте в папке tests файл test_readers.py

11. Мы будем использовать тестового клиента из FastApi. Добавьте тестовый метод

```
from fastapi.testclient import TestClient
from app.main import app

client = TestClient(app)

def test_read_items():
    response = client.get("/reader/1/")
    assert response.status_code == 200
```

12. Проверьте что тест работает. Если у вас уже были данные в о читателях в вашей бд с id =1 то тест будет пройден (зеленый), в противном случае тест упадет (будет красный).

13. Подменим через Dependency injection работу с базой данных для того чтобы тесты могли запускаться на своей базе данных (изолированно)

14. Запустите тест еще раз, поскольку база для работы с тестам теперь новая (при каждом запуске создается новый экземпляр бд с таблицами в памяти), то тест должен быть красным.

```python
from fastapi.testclient import TestClient
from app.main import app


from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker


from app.db.session import get_db
from app.db.models import Base

def override_get_db():
    engine = create_engine("sqlite:///:memory:")
    Base.metadata.create_all(bind=engine)

    SessionLocal = sessionmaker(bind=engine)
    db = SessionLocal()  # Используем тестовую базу
    try:
        yield db
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db


client = TestClient(app)

def test_read_items():
    response = client.get("/reader/1/")
    assert response.status_code == 200
```

Добавление тестов к API сервису «Библиотека»

- 15. Можете удалить метод test_read_items() он больше не понадобиться и добавьте тесты создания и получения читателей:

```python
def test_create_reader():
    print ("создание")
    response = client.post("/reader/",json={"name": "Марина"})
    assert response.status_code == 200
    data = response.json()
    assert data["name"] == "Марина"

    #TODO добавьте код проверяющий что читаль действительно создался с нужными параметрами


def test_get_reader():
    print ("чтение")

    #TODO добавьте код создания читателя, чтобы тест мог стать зеленым

    reader_id = 1
    # Получаем читателя
    response = client.get(f"/reader/{reader_id}/")
    assert response.status_code == 200
    data = response.json()
    assert data["id"] == reader_id
    assert data["name"] == "Иван"
```

Добавление тестов к API сервису «Библиотека»

16. Запустите тесты. Убедитесь что тест создания читателей зеленый, а тест получения красный. Это происходит потому, что метод override_get_db вызывается для каждого тестового метода.

17. Добавьте в тестовые методы код за место маркеров #TODO двумя способами:
    - Используя client (get/post запросы)
    - Через базу данных (для этого используете пример из пункта 3) учтите, что должна быть создана фикстура с отдельным выделением сессии соединения с бд, то есть будет 2 сессии в override_get_db и в фикстуре. При этом надо чтобы они ссылались на одну бд (база в памяти уже не подойдет, надо указывать файл тестовый бд).

18. Создайте тестовые методы проверки на удаления читателя и на получения всех читателей.

19. Создайте аналогичные файлы и методы в них для тестирования сущности “книг” и “выдачи”.