Лабораторная работа №5: Форматирование и линтинг проекта. Сборка образа проекта

ФИО: Ляхова Елизавета Олеговна
Группа: РИМ-150950
Цель: познакомится со способами поддержки качества кода и сборки образа приложения.

Ход работы:

1.Применение линтеров и форматеров при локальной разработке
Установим в проект pre-commit, pylint, black и isort.

В корне проекта настраиваем прекоммит .pre-commit-config.yaml

Файл pre-commit-config.yaml:

```yaml
fail_fast: false

repos:
 - repo: local
   hooks:
    - id: black
      name: Black
      entry: python -m black
      language: system
      types: [python]
      pass_filenames: true


    - id: isort
      name: isort
```

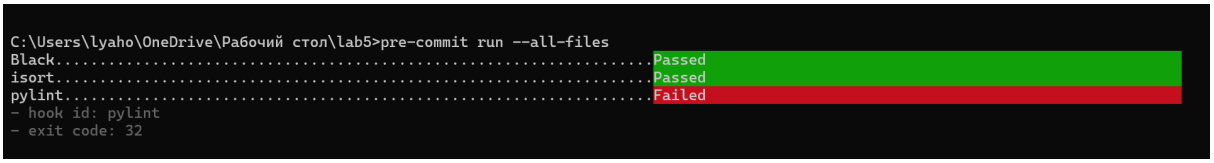
```
      entry: python -m isort
      language: system
      types: [python]
      pass_filenames: true
      args: ["--profile", "black"]

    - id: pylint
      name: pylint
      entry: python -m pylint
      language: system
      types: [python]
      pass_filenames: true
      args: []
      # Убрали --rcfile=.pylintrc, пусть использует настройки по умолчанию
```

Файл .pylintrc:

```
[MASTER]
jobs=1

[MESSAGES CONTROL]
disable=
    missing-docstring,
    too-few-public-methods,
    import-error,
    fixme,
    broad-except
```

```
[FORMAT]
max-line-length=88

[TYPECHECK]
generated-members=request,response,session,logger,app

[DESIGN]
max-locals=15

[FORMAT]
max-line-length=88

[MESSAGES CONTROL]
disable=duplicate-code,missing-docstring,too-few-public-methods[MESSAGES
CONTROL]
disable=duplicate-code,missing-docstring,too-few-public-methods
```

3. Выполнение команды pre-commit install и pre-commit run –all-files



4. Сборка образа проекта

1) Создание Dockerfile:

```
FROM python:3.13
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
# Запускаем напрямую Python
CMD ["python", "run.py"]
```

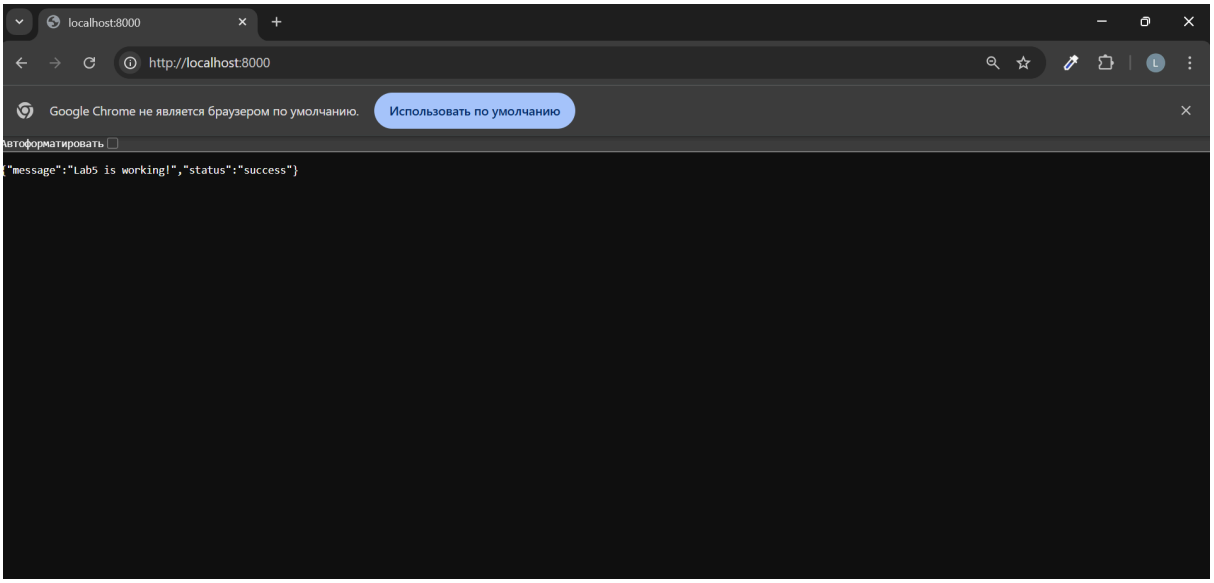
2) Добавление Docker-compose.yaml

```
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: app_db
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    ports:
      - "5432:5432"

  app:
    build: .
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      DATABASE_URL: postgresql://user:pass@db:5432/app_db
```

3) Запуск docker build -t lab5-app . и docker run -p 8000:8000 lab5-app

```
C:\Users\lyaho\OneDrive\Рабочий стол\lab5> docker build -t lab5-app .
[+] Building 1.0s (10/10) FINISHED                          docker:desktop-linux
 => [internal] load build definition from dockerfile                        0.0s
 => => transferring dockerfile: 228B                                        0.0s
 => [internal] load metadata for docker.io/library/python:3.13              0.5s
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [1/5] FROM docker.io/library/python:3.13@sha256:52ed248783130e635b76b2  0.0s
 => => resolve docker.io/library/python:3.13@sha256:52ed248783130e635b76b2  0.0s
 => [internal] load build context                                           0.0s
 => => transferring context: 3.57kB                                         0.0s
 => CACHED [2/5] WORKDIR /app                                               0.0s
 => CACHED [3/5] COPY requirements.txt .                                    0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt         0.0s
 => [5/5] COPY . .                                                          0.0s
 => exporting to image                                                      0.2s
 => => exporting layers                                                     0.1s
 => => exporting manifest sha256:c59f2183b891d5dab03ec2cd70929868a385e2c3e  0.0s
 => => exporting config sha256:56dfdfb8592f403b2ed4dc2fe6d5d095dc1f800cb86  0.0s
 => => exporting attestation manifest sha256:4eb2d7c5b988ea033c68bc276983d  0.0s
 => => exporting manifest list sha256:e149703f5cf8314b1aa9a5a4bf35a5ec6f2e  0.0s
 => => naming to docker.io/library/lab5-app:latest                          0.0s
 => => unpacking to docker.io/library/lab5-app:latest                       0.0s

C:\Users\lyaho\OneDrive\Рабочий стол\lab5>docker run -p 8000:8000 lab5-app
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     172.17.0.1:59776 - "GET / HTTP/1.1" 200 OK
INFO:     172.17.0.1:59776 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO:     172.17.0.1:38046 - "GET / HTTP/1.1" 200 OK
INFO:     172.17.0.1:38046 - "GET /favicon.ico HTTP/1.1" 404 Not Found
```

5) Проверка порта http://localhost:8000



Ответы на вопросы

1. Docker-контейнер vs виртуальная машина
   Контейнер — это легковесный изолированный процесс, работающий поверх ядра ОС. Виртуальная машина — полная эмуляция ОС со своим ядром. Контейнеры быстрее, легче и эффективнее используют ресурсы.
2. Кеширование слоев в Docker
   Каждая инструкция в Dockerfile создает слой. Docker кеширует слои, и если они не изменились, использует кеш. Это ускоряет сборку образов.
3. Инструкция `depends_on`
   Указывает зависимость сервисов в docker-compose. Контейнер `app` начнет запуск только после запуска контейнера `db`.
4. Миграции в entrypoint.sh, а не при сборке
   Миграции зависят от состояния БД, которое неизвестно при сборке образа. Их нужно выполнять при запуске контейнера, когда БД уже доступна.
5. Ошибка миграций при запуске
   Контейнер завершится с ошибкой, приложение не запустится. Это обеспечивает безопасность и предотвращает работу с неактуальной схемой БД.
6. Линтеры vs форматеры
   Линтеры (pylint, flake8) анализируют код на ошибки, стиль, сложность. Форматеры (black, isort) автоматически форматируют код по единым правилам.
7. pre-commit хуки
   Автоматически запускают проверки перед коммитом, предотвращая попадание некорректного кода в репозиторий.

Вывод:

В ходе лабораторной работы были освоены инструменты для поддержания качества кода (pylint, black, isort) и автоматизации проверок через

pre-commit. Была реализована сборка Docker-образа приложения с настройкой миграций БД и оркестрацией через docker-compose. Полученные навыки позволяют обеспечивать стандартизацию кода и эффективное развертывание приложений в изолированных средах.