РБПО – Отчёт ДЗ 7-8

Выполнил: Васюков Александр, БПИ235
Проект: "Media Catalog"

---

1. Контейнерная безопасность

В рамках задания была выполнена полная переработка контейнеризации приложения с упором на безопасность и воспроизводимость. Основная цель состояла в том, чтобы получить минимальный и защищённый Docker-образ, который не содержит лишних зависимостей, запускается без root-прав и проходит проверки безопасности.

Что было сделано

1. Создан многоступенчатый Dockerfile на базе `python:3.11-slim`, что позволило сократить размер итогового образа и отделить сборочную среду от среды выполнения.
2. Контейнер переведён на запуск под non-root пользователем, что устраняет риск безопасности — выполнение кода с привилегиями суперпользователя.
3. Добавлены health checks через обращение к эндпоинту `/health`, чтобы оркестратор мог отслеживать состояние сервиса и перезапускать его при сбоях.
4. Интегрирован Hadolint и добавлен файл конфигурации с правилами, а также линтинг Dockerfile включён в CI.
5. Обновлён docker-compose, включены health checks для приложения и для базы данных.
6. Расширен .dockerignore, чтобы сократить build context и ускорить сборку.
7. Добавлен скрипт проверки контейнера (`scripts/check_container.sh`), который автоматизирует локальную валидацию и проверки безопасности.
8. CI дополнен задачами по security scanning, тестированию контейнера и проверке Dockerfile.

```
# Build stage
FROM python:3.11-slim AS build

WORKDIR /app

# Install build dependecies
RUN apt-get update && apt-get install -y --no-install-recommends \
    curl \
    gcc \
    python3-dev \
    libmagic-dev \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements and install dependecies
COPY requirements.txt requirements-dev.txt ./
RUN pip install --no-cache-dir -r requirements.txt -r requirements-dev.txt

# Copy application code for build validation
COPY . .

# Runtime stage
FROM python:3.11-slim

WORKDIR /app

# Install runtime dependencies only
RUN apt-get update && apt-get install -y --no-install-recommends \
    curl \
    libmagic1 \
    file \
    && rm -rf /var/lib/apt/lists/*

# Create non-root user
RUN groupadd -r appuser && useradd -r -g appuser appuser

# Copy installed packages from builder stage
COPY --from=build /usr/local/lib/python3.11/site-packages /usr/local/lib/python3.11/site-packages
COPY --from=build /usr/local/bin /usr/local/bin

# Copy application code
COPY . .

# Set proper permissions
```

```
RUN chown -R appuser:appuser /app
USER appuser

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=40s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

EXPOSE 8000
ENV PYTHONUNBUFFERED=1

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

```
services:
    app:
        build: .
        depends_on:
            db:
                condition: service_healthy
        environment:
            DATABASE_USER: ${DATABASE_USER}
            DATABASE_PASSWORD: ${DATABASE_PASSWORD}
            DATABASE_HOST: db
            DATABASE_PORT: 5432
            DATABASE_NAME: ${DATABASE_NAME}
            PYTHONUNBUFFERED: 1
        ports:
            - "8000:8000"
        healthcheck:
            test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
            interval: 30s
            timeout: 10s
            retries: 3
            start_period: 40s

    db:
        image: postgres:17
        restart: always
        environment:
            POSTGRES_USER: ${DATABASE_USER}
            POSTGRES_PASSWORD: ${DATABASE_PASSWORD}
            POSTGRES_DB: ${DATABASE_NAME}
        ports:
            - "5432:5432"
        volumes:
            - postgres_data:/var/lib/postgresql/data
        healthcheck:
            test: ["CMD-SHELL", "pg_isready -U ${DATABASE_USER} -d ${DATABASE_NAME}"]
            interval: 5s
            timeout: 5s
            retries: 5

volumes:
    postgres_data:
```

Как проверялась работа

- Образ успешно собирался и запускался вручную.
- При запуске контейнера проверялся UID, чтобы убедиться, что приложение работает не от root.
- Приложение корректно запускалось как через `docker run`, так и через `docker compose`.
- Hadolint отрабатывал без ошибок.
- Все тесты и проверки из CI проходили успешно.
- Скрипт `check_container.sh` подтверждал корректность конфигурации.

---

2. CI/CD инфраструктура

Работы была посвящена созданию минимальной CI/CD-системы на GitHub Actions. Основная задача — обеспечить автоматическое тестирование, проверку качества кода, сбор артефактов и демонстрацию простейшего pipeline развёртывания.

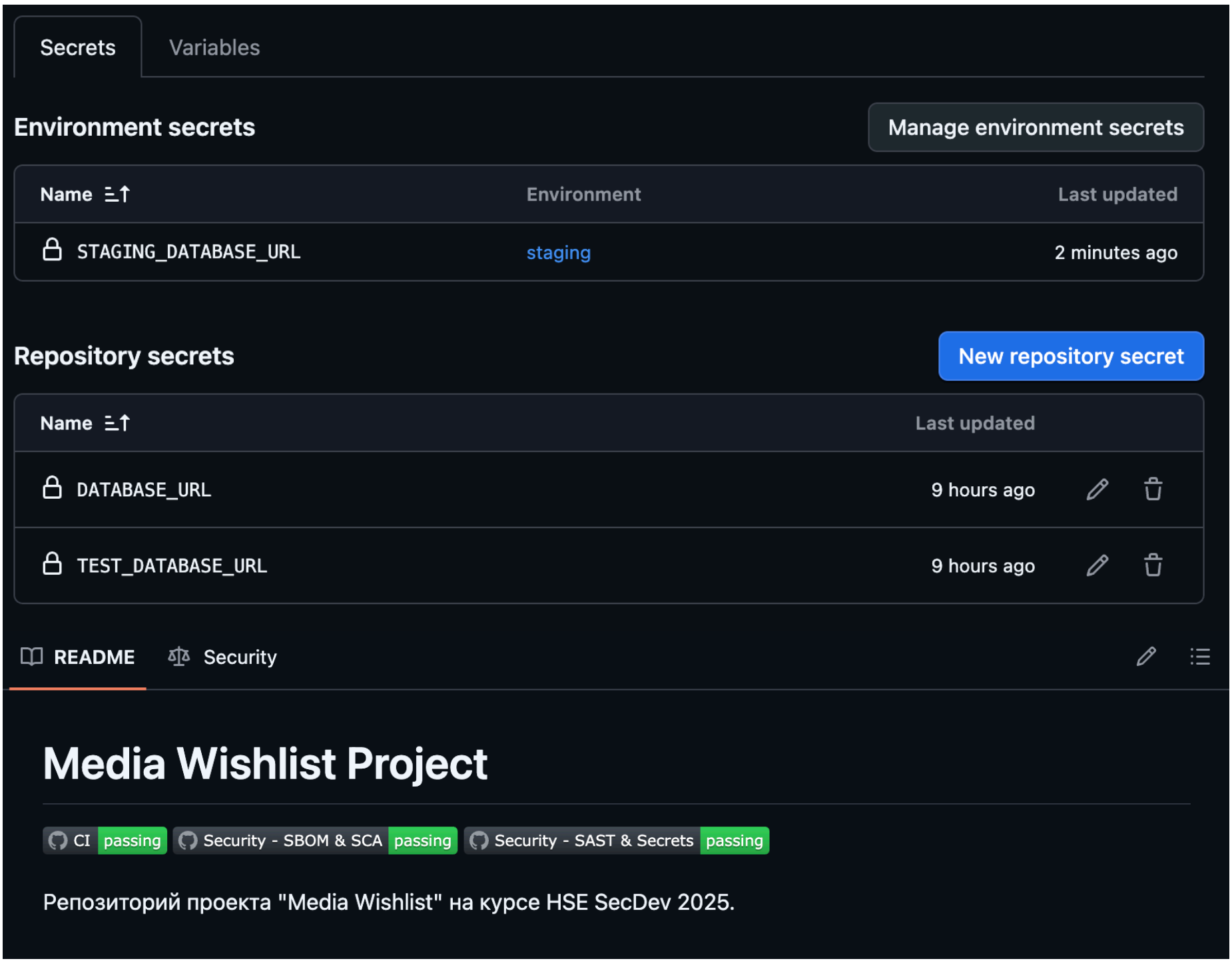
Что было сделано

1. Создан минимальный CI workflow, который запускается на каждом push и pull request.

2. Добавлена настройка Python с кешированием pip.
3. Инструменты проверки качества кода: ruff, black, isort.
4. Настроен запуск pytest, а результаты тестов загружаются как артефакты.
5. В README добавлен статус-бейдж, который отражает состояние CI.
6. Добавлены GitHub Secrets для конфигурации базы и дальнейшего развёртывания.
7. Организована выгрузка расширенных артефактов, включая покрытие тестов, Docker-образ и отчёты сканирования.
8. Реализован простой CD-pipeline.

```yaml
name: CI

on:
  push:
    branches: [ main, develop ]
    paths-ignore:
      - 'docs/**'
      - 'README.md'
      - 'SECURITY.md'
      - '.gitignore'
  pull_request:
    branches: [ main ]
    paths-ignore:
      - 'docs/**'
      - 'README.md'
      - 'SECURITY.md'
      - '.gitignore'

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: ${{ github.event_name == 'pull_request' }}

permissions:
  contents: read

jobs:
  lint-and-test:
    runs-on: ubuntu-latest
    timeout-minutes: 10

    strategy:
      matrix:
        python-version: ["3.12"]

    steps:
    - name: Checkout repository
      uses: actions/checkout@v4
      timeout-minutes: 2

    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v5
      with:
        python-version: ${{ matrix.python-version }}
        cache: 'pip'
      timeout-minutes: 2

    - name: Cache pip dependencies
      uses: actions/cache@v4
      with:
        path: ~/.cache/pip
        key: ${{ runner.os }}-pip-${{ hashFiles('requirements*.txt') }}
        restore-keys: |
          ${{ runner.os }}-pip-

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt -r requirements-dev.txt
      timeout-minutes: 3

    - name: Check code formatting with Black
      run: |
        black --check .
      timeout-minutes: 2

    - name: Check import sorting with isort
      run: |
```

```
        isort --check-only .
      timeout-minutes: 2

    - name: Lint with ruff
      run: |
        ruff check .
      timeout-minutes: 2

    - name: Run tests with pytest
      run: |
        mkdir -p reports
        pytest -q --junitxml=reports/junit.xml --cov=app --cov-report=xml:reports/coverage.xml --cov-
report=html:reports/coverage-html
      timeout-minutes: 5

    - name: Upload test artifacts
      if: always()
      uses: actions/upload-artifact@v4
      with:
        name: test-reports
        path: |
          reports/
        retention-days: 7
      timeout-minutes: 2
```



Как проверялась работа

- Линтеры и форматирование проходили без ошибок.
- Pytest работал стабильно, отчёты корректно сохранялись.
- Pip caching снижал время выполнения workflow.
- Все события корректно запускали нужные пайплайны.
- Статус-бейдж отображал состояние CI.
- Все проверки в CI проходили на зелёный.

Заключение

В ходе выполнения заданий были созданы два ключевых компонента инфраструктуры проекта: защищённая контейнеризация и автоматизированный CI/CD-процесс.

Работы по контейнерной безопасности позволили сформировать минимальный и надёжный Docker-образ, устойчивый к типичным угрозам и полностью соответствующий требованиям безопасной разработки. Были внедрены health checks, отказ от root-пользователя, сканирование уязвимостей и линтинг Dockerfile.

Была разработана CI/CD-система, обеспечивающая непрерывное тестирование, автоматический анализ качества кода, контроль окружений, сохранение артефактов и демонстрацию полного цикла развёртывания. Это создало основу для устойчивой, расширяемой и профессиональной инфраструктуры разработки.

Обе задачи завершены, протестированы и удовлетворяют требованиям.