Тестовое задание
Техническая записка и скриншоты

Table of Content

1 Repositories: 2
2 Техническое задание: 2
2.1 Дополнительные плюсы: 3
3 Техническая записка 3
3.1 Версии SW 3
3.2 Варианты решения задачи 3
3.3 Подключение БД 3
3.4 Тестовые данные 4
3.5 Реализованные механизмы 4
3.6 Варианты запуска программы 5
4 Скриншоты 6
4.1 Развертывание приложения в Docker 6
4.2 Работа приложения в Docker 7
4.3 Ответ по запросу http://127.0.0.1:8000/?category=train&category=bus - Image8.jpg 8
4.4 Следующий ответ по запросу http://127.0.0.1:8000/?category=train&category=bus - Image9.jpg 9
4.5 Ответ по запросу если категория не найдена 10

1 Repositories:

https://github.com/ecrvmal/RTSoft - for native implementation
https://github.com/ecrvmal/RTSoft/tree/Docker_Implementation - for docker implementation

2 Техническое задание:

Задача - сделать сервис просмотра контента.

Он должен представлять из себя простой веб-сервис написанный на python 3 с использованием любых показавшихся необходимыми библиотек.

Есть не изменяемый файл конфигурации в CSV, который содержит данные вида:

Image_URL;needed_amount_of_shows;category1;category2;category3; … ;category N

Количество записей в файле не более 1000.

Файл содержит только символы.

Необходимое количество показов - сколько раз картинку нужно показать.

Категорий у одной картинки может быть от 1 до 10 штук, записанных в соответствующих колонках.

Пример:

http://localhost:8080/static/image1.jpg;500;flight;airlplane
http://localhost:8080/static/image2.jpg;3300;show;britain;bennyhill;sketches;tv
http://localhost:8080/static/image3.jpg;1500;games;minecraft;blocks;sandbox
http://localhost:8080/static/image4.jpg;120;onlycategory

При запуске сервис считывает конфигурационный файл и начинает слушать HTTP обращения.

Клиент HTTP GET запросом сообщает категории, которые он готов принять, к примеру:

http://localhost:8080/?category[]=auto&category[]=trains

Количество запрошенных категорий может быть от 0 до 10 штук.

В ответ на него сервис должен выдавать простую HTML обертку с изображением картинки, которая совпадает минимум по одной категории.

Категории картинки произвольны если в запросе категорий не указано.

Каждый показ вычитается из количества необходимых к показу.

По характеру запросов считаем, что:

1. не будут запрашиваться категории, которых нет в конфигурационном файле.
2. количество запрошенных категорий равновероятно в рамках 0-10.

3. появление любой из категорий в запросе равновероятно.

2.1 Дополнительные плюсы:

1. Код соответствует PEP8.
2. Механизм, который уменьшает вероятность выдачи одной и той же картинки несколько раз подряд
3. Механизм, позволяющий минимизировать вероятность возникновения случаев, когда подходящие картинки уже исчерпали свой лимит и ответить на запрос нечем.

3 Техническая записка

3.1 Версии SW

Для выплнения задачи была использована библиотека Django версии 3.2.23,
Версия интерпретатора Python 3.10
Версии используемых модулей приведены в репозитории в файле requirements.txt

3.2 Варианты решения задачи

Поставленную задачу можно было выполнить 2-мя способами:

1. без создания БД, в этом случае файл CSV нужно подгружать каждый раз снова, результаты прошлой сессии не сохраняются, и
2. с созданием БД- в этом случае файл CSV загружается 1 раз командами

   ```
   python manage import_categs.py  <filename.csv>
   python manage import_data.py  <filename.csv>
   ```

   и в дальнейшем код работает с данными, загруженными в БД. При перезагрузке программы данные не теряются.

В проекте реализован 2-й вариант.

3.3 Подключение БД

В качестве БД в проекте используется Postgres. Перед запуском монолитного варианта Django для работы с БД нужно обеспечить связь программы и БД, для этого сделать следующее:

- поправить строки в config/settings.py (в части user/password/name)

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql_psycopg2",
        "NAME": "RTSoft",
        "USER": "prj_user",
        "PASSWORD": "prj_user",
        "HOST": "127.0.0.1",
        "PORT": "5432",
    }
}
```

- Также установить БД Postgres
- Создать в Postgres базу данных с названием "RTSoft".

3.4 Тестовые данные

Для отладки программы были созданы тестовые изображения и тестовый CSV файл.

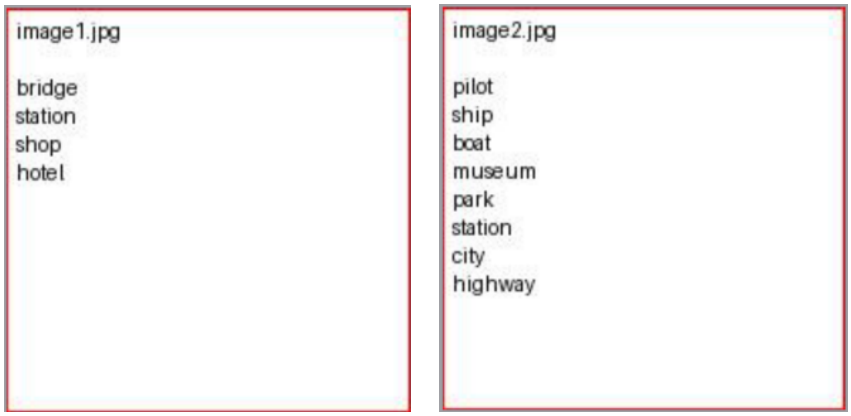
Данные созданы с помощью скрипта расположенного в директории /gen_csv/gen_csv.py

Тестовый файл SCV

```
image1.jpg;2106;bridge;station;shop;hotel
image2.jpg;2677;pilot;ship;boat;museum;park;station;city;highway
image3.jpg;2199;tower;ticket;pilot;street
image4.jpg;2439;tower;park;highway;bridge;parking;hotel
image5.jpg;132;station;bus;bridge;boat;ship
image6.jpg;1536;bike;parking;boat;city;station;airport;train;highway
image7.jpg;419;station;bike
image8.jpg;1967;bus;bike;shop;city
image9.jpg;1931;hotel;city;flight;street;train
```

Screenshot 1

Примеры тестовых изображений (jpg формат) :



Screenshot 2

Тестовые данные можно восстановить из фикстуры командой (Вместо загрузки данных командами) :

```
python manage.py loaddata mainapp/fixtures/001_dumpdata.json
```

Тестовые изобрвжения загружены в фолдер /static/ , тестовый файл сохранен как /gen_csv/input_data.csv

3.5 Реализованные механизмы

В процессе написания кода реализован механизм, который уменьшает вероятность выдачи одной и той же картинки несколько раз подряд. Это обеспечивается тем, что из списка изображений выбирается изображение с изображение с максимальным количеством оставшихся показов, если оно совпадает с ранее показанным – выбирается следующее изображение из списка, отсортированного по количеству показов.

Также реализован механизм, позволяющий минимизировать вероятность возникновения случаев, когда подходящие картинки уже исчерпали свой лимит и ответить на запрос нечем : в начале работы программы из набора возможных изображений выбирается

изображение с максимальным количеством оставшихся показов, при выравнивании оставшегося количества показов картинки показываются равномерно.

Реализован контроль количества показа изображений, при исчерпании лимита показов – показывается изображение «по умолчанию» и выводится надпись
« Items not found, Dummy image shown »

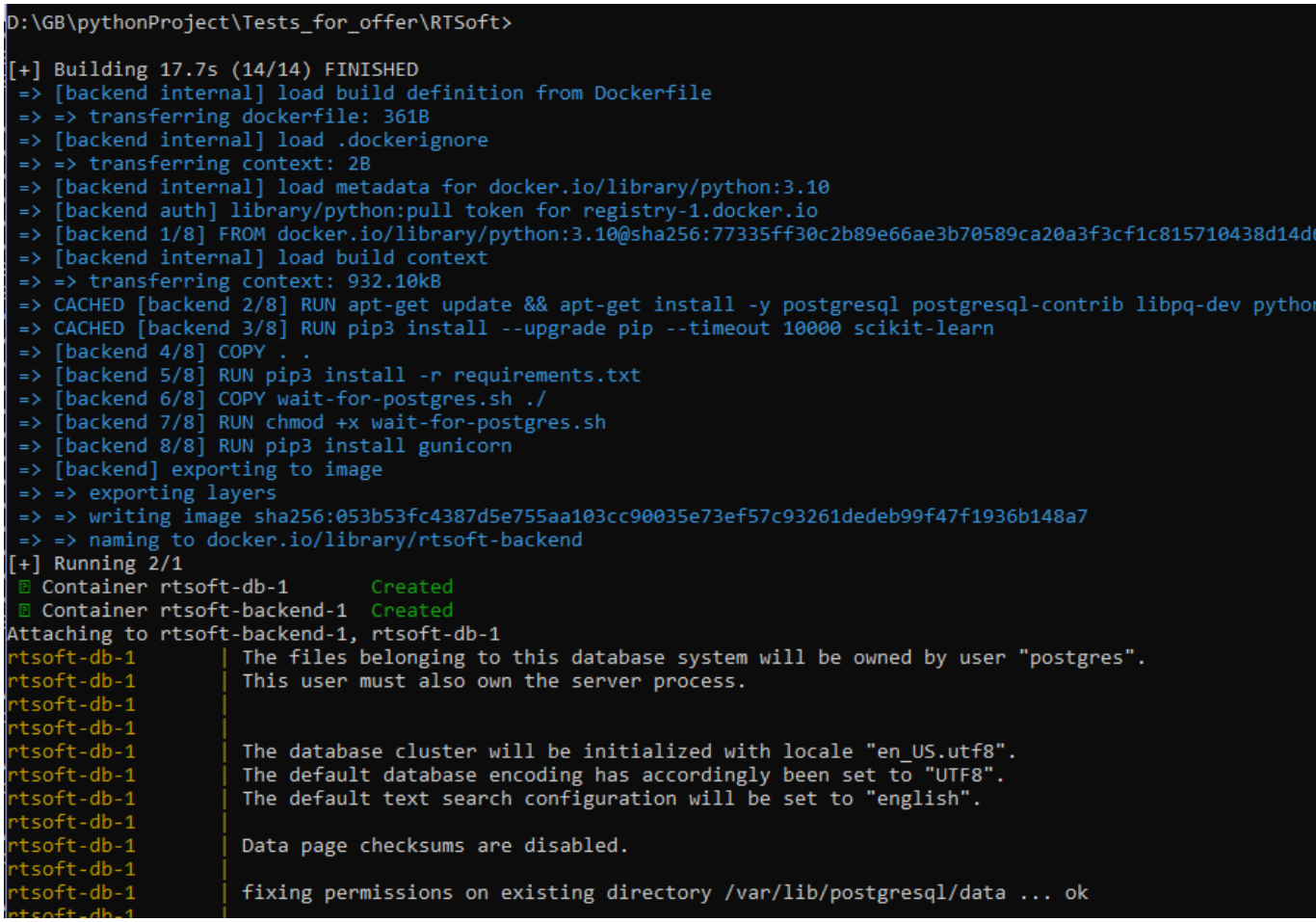
3.6 Варианты запуска программы

Возможен запуск программы как в монолитном варианте, так и в среде Docker. Соответствующие файлы и настройки загружены в репозиторий.

При запуске в среде Docker для ускорения разработки был использован втроенный wsgi сервер, для коммерческого использования необходимо доработать программу, использовать сервер Gunicorn и сервер NGINX frontend для раздачи статики.

4 Скриншоты

4.1 Развертывание приложения в Docker



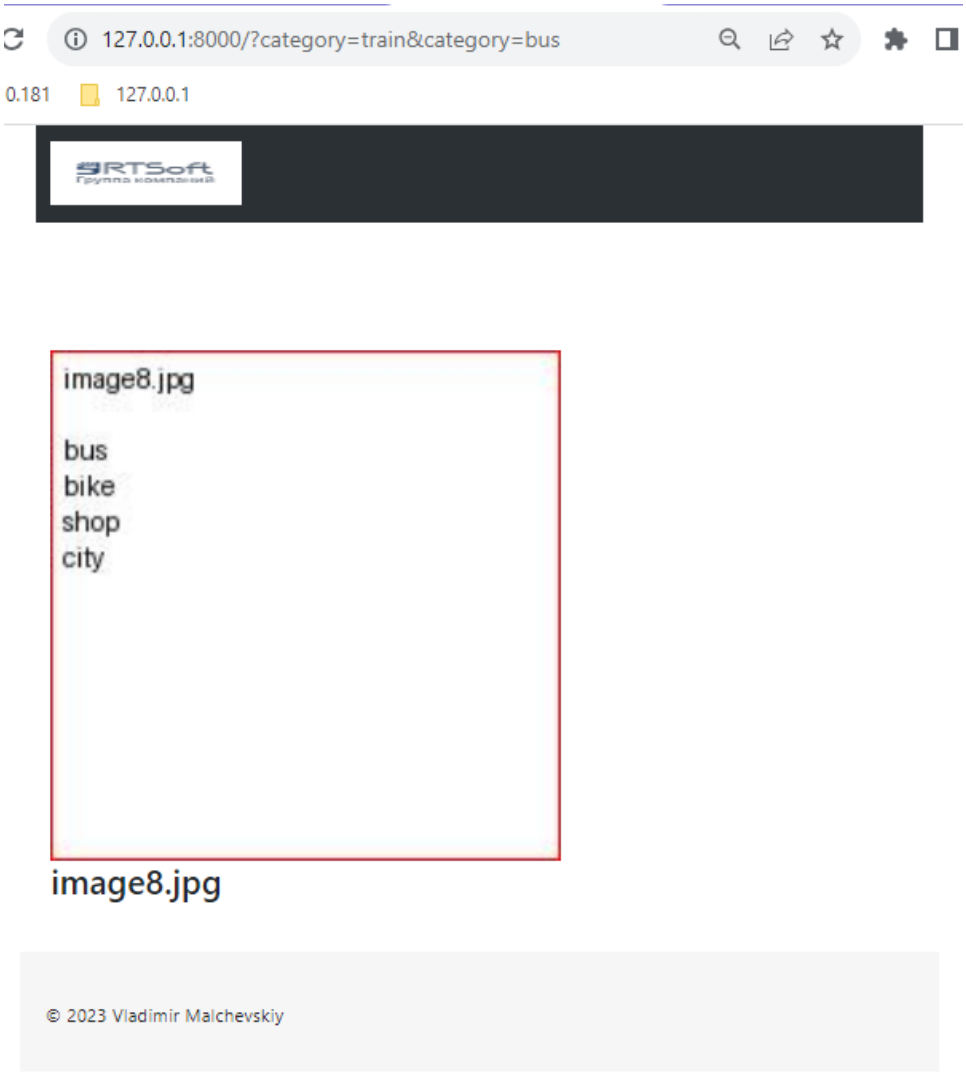
Screenshot 3

4.2 Работа приложения в Docker

```
rtsoft-backend-1  | Watching for file changes with StatReloader
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /?category=train HTTP/1.1" 200 1853
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/css/stars.css HTTP/1.1" 200 572
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/css/style.css HTTP/1.1" 200 1272
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/css/brands.min.css HTTP/1.1" 200 18860
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/css/solid.min.css HTTP/1.1" 200 577
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/img/image9.jpg HTTP/1.1" 200 4595
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/css/fontawesome.min.css HTTP/1.1" 200 80831
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/js/bootstrap.bundle.min.js HTTP/1.1" 200 80429
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/css/bootstrap.min.css HTTP/1.1" 200 232918
rtsoft-backend-1  | [12/Dec/2023 22:13:51] "GET /static/img/logo.png HTTP/1.1" 200 43674
rtsoft-db-1       | 2023-12-12 22:18:40.761 UTC [62] LOG:  checkpoint starting: time
rtsoft-db-1       | 2023-12-12 22:19:01.604 UTC [62] LOG:  checkpoint complete: wrote 210 buffers (1.3%); 0
=20.806 s, sync=0.026 s, total=20.844 s; sync files=131, longest=0.003 s, average=0.001 s; distance=950 kB,
B38
rtsoft-backend-1  | [12/Dec/2023 22:20:24] "GET /?category=train HTTP/1.1" 200 1853
rtsoft-backend-1  | [12/Dec/2023 22:20:24] "GET /static/img/image6.jpg HTTP/1.1" 200 5425
rtsoft-db-1       | 2023-12-12 22:23:40.611 UTC [62] LOG:  checkpoint starting: time
rtsoft-db-1       | 2023-12-12 22:23:40.723 UTC [62] LOG:  checkpoint complete: wrote 2 buffers (0.0%); 0 W
.102 s, sync=0.004 s, total=0.113 s; sync files=2, longest=0.004 s, average=0.002 s; distance=0 kB, estimat
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /?category=blog HTTP/1.1" 200 1893
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/css/bootstrap.min.css HTTP/1.1" 304 0
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/css/style.css HTTP/1.1" 304 0
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/css/stars.css HTTP/1.1" 304 0
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/js/bootstrap.bundle.min.js HTTP/1.1" 304 0
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/css/brands.min.css HTTP/1.1" 304 0
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/css/fontawesome.min.css HTTP/1.1" 304 0
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/css/solid.min.css HTTP/1.1" 304 0
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/img/image1.jpg HTTP/1.1" 200 4495
rtsoft-backend-1  | [12/Dec/2023 23:08:44] "GET /static/img/logo.png HTTP/1.1" 304 0
```

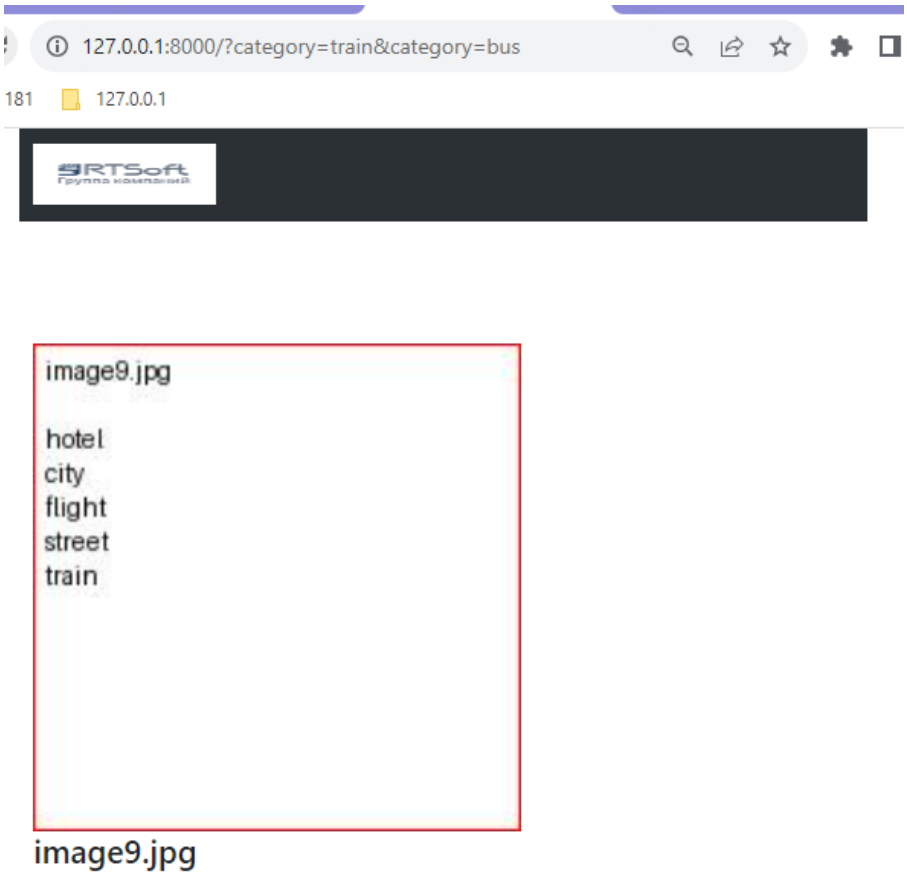
Screenshot 4

4.3 Ответ по запросу http://127.0.0.1:8000/?category=train&category=bus - Image8.jpg



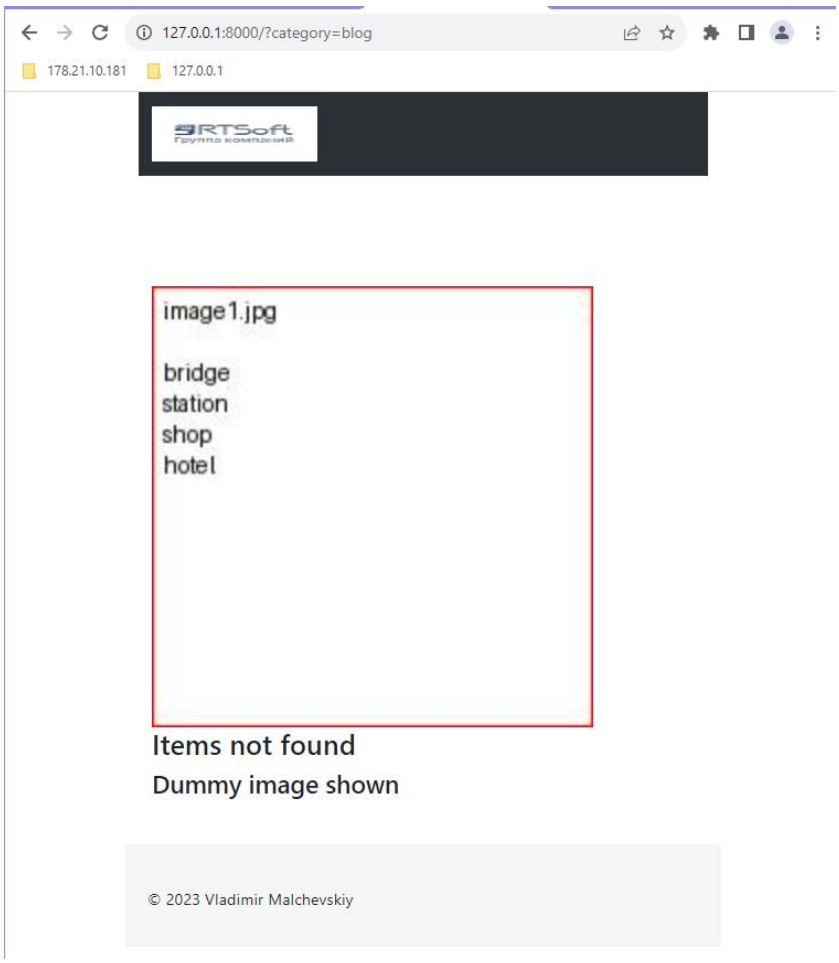
Screenshot 5

4.4 Следующий ответ по запросу http://127.0.0.1:8000/?category=train&category=bus - Image9.jpg



Screenshot 6

4.5 Ответ по запросу если категория не найдена



Screenshot 7