ООО «ЭЛЕТЕК»

Руководство эксплуатации программного обеспечения «ЛАН. Процессор изображений»

2025

1 Общие сведения

Программное обеспечение «ЛАН. Процессор изображений» разработано для обработки и анализ изображений и видео.

Для оказания технической поддержки с установкой, настройкой или запуском выделен единый номер технической поддержки 8 (999) 524-04-71.

2 Установка и запуск

2.1 Модуль обработки изображений

Для установки модуля обработки изображений требуются следующие компоненты:

- архивы image.tar, minio.tar, содержащие соответствующие Docker-образы;
- архив с дополнительными ресурсами image.zip;
- конфигурационные файлы image.yml и minio.yml;
- сервер с установленной ОС Ubuntu\Astra Linux\Alt Linux, с доступом по протоколу SSH.

Для развертывания сервисов обработки изображений требуется выполнить следующие действия:

1. Передать на сервер архивы image.tar, minio.tar, image.zip, а также файлы image.yml и minio.yml.

2. Подключиться к серверу по протоколу SSH, используя логин и пароль пользователя ОС.

3. Загрузить Docker-образы из архивов командами «docker load –i image.tar» и «docker load –i minio.tar». После завершения загрузки в терминале отобразятся имена и теги образов, которые могут понадобиться для настройки docker-compose.

4. Распаковать архив с дополнительными ресурсами командой «unzip image.zip».

5. Указать абсолютный путь к месту распакованной папки MINIO в конфигурационном файле minio.yml, используя поле volumes.

6. Запустить сервис minio командой «docker compose –f minio.yml up –d».

7. Указать корректный IP-адрес, где развернут сервис S3 Minio, для всех полей ENDPOINT в конфигурационном файле image.yml.

8. Запустить сервисы командой «docker compose –f image.yml up –d».

Для удаления сервисов обработки изображений требуется выполнить следующие действия:

1. Остановить запущенные сервисы командами «docker compose –f image.yml down» и «docker compose –f minio.yml down».

2. Удалить ранее загруженные Docker-образы командой «docker rmi {имя_образа:тег}».

3. Удалить архивы image.tar, minio.tar, image.zip.

4. Удалить папку MINIO и файл request.txt.

5. Удалить конфигурационные файлы image.yml и minio.yml.

Для проверки работоспособности сервисов обработки изображений требуется выполнить следующие действия:

1. Открыть в браузере сервис по адресу http://localhost:7101/docs.

2. Использовать метод POST /tasks для запуска задачи обработки с телом запроса, указанным в файле request.txt, который был распакован из архива image.zip.

3. Использовать метод GET /tasks/{task} с полученным UUID задачи в теле ответа из предыдущего шага для отслеживания статуса обработки и получения результатов обработки задачи.

4. Использовать метод DELETE /tasks/{task} с UUID задачи для удаления задачи из системы.

2.2 Модуль обработки видео

Для установки модуля обработки видео требуются следующие компоненты:

- архивы video.tar, minio.tar, содержащие соответствующие Docker-образы;
- архив с дополнительными ресурсами video.zip;
- конфигурационные файлы video.yml и minio.yml;
- сервер с установленной ОС Ubuntu\Astra Linux\Alt Linux, с доступом по протоколу SSH.

Для развертывания сервисов обработки видео требуется выполнить следующие действия:

1. Передать на сервер архивы video.tar, minio.tar, video.zip, а также файлы video.yml и minio.yml.

2. Подключиться к серверу по протоколу SSH, используя логин и пароль пользователя ОС.

3. Загрузить Docker-образы из архивов командами «docker load –i video.tar» и «docker load –i minio.tar». После завершения загрузки в терминале отобразятся имена и теги образов, которые могут понадобиться для настройки docker-compose.

4. Распаковать архив с дополнительными ресурсами командой «unzip video.zip».

5. Указать абсолютный путь к месту распакованной папки MINIO в конфигурационном файле minio.yml, используя поле volumes.

6. Запустить сервис minio командой «docker compose –f minio.yml up –d».

7. Указать корректный IP-адрес, где развернут сервис S3 Minio, для всех полей ENDPOINT в конфигурационном файле video.yml.

8. Запустить сервисы командой «docker compose –f video.yml up –d».

Для удаления сервисов обработки видео требуется выполнить следующие действия:

1. Остановить запущенные сервисы командами «docker compose –f video.yml down» и «docker compose –f minio.yml down».

2. Удалить ранее загруженные Docker-образы командой «docker rmi {имя_образа:тег}».

3. Удалить архивы video.tar, minio.tar, video.zip.

4. Удалить папки MINIO и VIDEO.

5. Удалить конфигурационные файлы video.yml и minio.yml.

Для проверки работоспособности сервисов обработки видео требуется выполнить следующие действия:

1. Открыть в браузере сервис по адресу http://localhost:6101/docs.

2. Использовать метод POST /tasks для запуска задачи обработки с телом запроса:

```
{
  "task": null,
  "pipelines": [
    "video_scene",
    "video_emblem",
    "video_military_vehicle",
    "video_face",
    "video_text"
  ],
  "source": "Командование и военнослужащие РВСН приняли участие в акции
Елка.mp4"
}
```

3. Использовать метод GET /tasks/{task} с полученным UUID задачи в теле ответа из предыдущего шага для отслеживания статуса обработки и получения результатов обработки задачи.

4. Использовать метод DELETE /tasks/{task} с UUID задачи для удаления задачи из системы.

2.3 Модуль хранения и индексации архивов изображений

Для установки модуля хранения и индексации архивов изображений требуются следующие компоненты:

- архивы search.tar, postgres.tar, содержащие соответствующие Docker-образы;
- архив с дополнительными ресурсами search.zip;
- конфигурационные файлы search.yml и postgres.yml;
- сервер с установленной ОС Ubuntu\Astra Linux\Alt Linux, с доступом по протоколу SSH.

Для развертывания сервисов хранения и индексации архивов изображений требуется выполнить следующие действия:

1. Передать на сервер архивы search.tar, postgres.tar, search.zip, а также файлы search.yml и postgres.yml.

2. Подключиться к серверу по протоколу SSH, используя логин и пароль пользователя ОС.

3. Загрузить Docker-образы из архивов командами «docker load –i search.tar» и «docker load –i postgres.tar». После завершения загрузки в терминале отобразятся имена и теги образов, которые могут понадобиться для настройки docker-compose.

4. Распаковать архив с дополнительными ресурсами командой «unzip search.zip».

5. Указать абсолютный путь к месту распакованной папки PGDATA в конфигурационном файле postgres.yml, используя поле volumes.

6. Запустить сервис postgres командой «docker compose –f postgres.yml up –d».

7. Указать корректный IP-адрес, где развернут сервис PostgreSQL, для всех полей POSTGRESQL_URL в конфигурационном файле search.yml, а также абсолютный путь к ранее распакованной папке с названием 00000000-0000-0000-0000-000000000000 для всех полей volumes.

8. Запустить сервисы командой «docker compose –f search.yml up –d».

Для удаления сервисов хранения и индексации архивов изображений требуется выполнить следующие действия:

1. Остановить запущенные сервисы командами «docker compose –f search.yml down» и «docker compose –f postgres.yml down».

2. Удалить ранее загруженные Docker-образы командой «docker rmi {имя_образа:тег}».

3. Удалить архивы search.tar, postgres.tar, searcrh.zip.

4. Удалить папки PGDATA и 00000000-0000-0000-0000-000000000000.

5. Удалить конфигурационные файлы search.yml и postgres.yml.

Для проверки работоспособности сервисов хранения и индексации архивов изображений требуется выполнить следующие действия:

1. Открыть в браузере сервис по адресу http://localhost:8101/docs.

2. Использовать метод POST /indexes/search для проведения поиска по индексу с телом запроса:

```
{
  "indexes": [
```

```
    "7aa9d013-9169-4b7b-a838-36f803466f9c"
  ],
  "model": "a9c0f70a-50e4-427b-ae9a-21f6bf0b7079",
  "top_k": 1,
  "vectors": [
"XWOrOyeA+LyN4Iq8uMutPK/3HLzMopi8KO3RuxaH2LyEr289GDD1uotL4L2LJTA8ceFAvftJ8zuK
5Cq9F852PRf3xzskO9q7loBGvU9BBr7hZyY9J/jnvBtD5zx5lSW9qGtePXCOKz0wbfo7ylybvCMtH
z0V1Eg9irnJPOPlLj1FGvO7iesuPENa/DwApFW9wem/vP7QyrpzO7+9Gh2nPQqfczzMHzC8HT7sPF
74kbr78IA8DYKQOyA8H72bpF49N47MvUwFob3noZS9IjwsvFJART3oFLm8kcaQvZbDdT33NpY9NS7
7PFwpiLxrsoE99SjwvXlMhTyn1Rw8pDM7vDSfij3L8TE91T1cOGEojTzS0uM8yDQWvU63mz0dqz85
CAFkO6S5lD2hsg+6qYdPvPevB71XRIS81tOQPN2A87zOCYa9dMU2vCH7HTxqC089kqc9PS2sDT1wf
8k73AQbPYrT+DwqRDK9pR2FPZZEmTx2/Ei915SHvKePUb3l5l+9tz1ePcrw1DxVnRG960TfPLeQVr
1QULE8Vg4MPUP8tr2mj+U8VkHPPEolWDv3VR08Gq+HPTn1OryhS6i8FzdjPWSG0Dzor1y9uG1XvTW
krr2gB2u9mP0WPEspgr30R/u8ecHvPOb9H7yzrQK8mBOlPFn8pbzO1js7eExcvfElEbsVxj49LLMb
PTiTVjvL/7m9PFy0vKdpEz3yx0w9TjXguonaCT2//pU7tRsDvajNWr2qccO8DFMAPGCTTTuiA1q9v
htxvZYBe7x4T7i7WD9qPb6kRzytD5K8ar4dvfrY1b0VGqu9bQLhPCYWgz1/0rk7lFJwPFWGYT1Mp8
U9961yvdXzXz0c4n88F3OIPKliv7yoLCi9BT+aO+ayujxRuiy9Q1pTPZDfDT2Q6Iw8x7vYvM8JPT1
GS5o8A3qpPKKHjz3nW/G9yNUDOz7Ahb1dCYi8JXmQPXTMm7zCY9q8v/feOwQma70ikso8b6GtPTDx
tD294I088/qhO3egNLwSSLk7p+MBPQBY8TykVp28+2k7vVG7jryxl448rub7PGt8dr3Y2gm9K5w2v
UU3mj1l4GU9kN8bvZWUlD2AUs+8AGrQvCtFED2FL4O8U54gvZaWUryr5iM90Ps+vULUI7xrAWm6I9
qFPVyWyr2+Awk+AXMEPXkxGj0o3um82TwUvCGh3TvGetC7WPEcvXoVkD3dM8m898DCvCwbNrzCIYM
8cFg0PJbYQ72D1QA999YnPd8ewDr7Uga9/K0ZPd+ZTLwG3ww9RXkGvYloej2FY4O9BVe6vNsLQLww
EsM8OpfRPDLqgzptFDC6bseqvCM1RbzqsiK9kZbrOcOecb3TbbG9zy12vDvgWTt5tFw9ZoWSPIC1r
rzvwwC8kRcWvRhdqzuzA3Y8yfMGPUceyjxYG6M9/0pyPXAxdjwLAKW7bllLvG7sNb01vtQ8ucUMvB
m5fr0G0Cq94I6SPUWmgjxmZkQ9JWXCPK5ZQDj+u3+9YdTDPNvE1TvC5DG5LlrxPGzJlDxVIYe9PlS
xvGL7gbyUQ+a9BMTbO3agRj3t74S9KfTePf4rjrxXTS67GtYZu65Alz0Q0Sg9hA9muUkoCL2j3B49
riSpu+1GAL3vRy+9o02EvRZReD2xrTu9ca/QPMw9zrwFEIo8yJB9ve3HIz3JW589PmkRPJX3ur3Vz
yQ91N6LukkW1T1hTIo9kk4TO+TXQb0TC4q6vMYYvee54zsyHwu9U1zau6yVOb2Z6lo8KModPR2+/j
x18Q29LzEfPkbdDT1mPcm9+87yvG2a0rzpLZY9wajJvBB2u73gK6+7ttVzvbcl47vI/by9h+gRvXN
YuzzHaR+917QCug+R4bz9VpI8hUoGvaFXMLx8Y547XMsUvak6N70biHM9EgGhvM7egD3y6YI9G+Q2
PfvLWj0BEII7QqNhPF87AL3lqqi95TLcPAVyory8mjc8vTYjvWo5vrzfoAK+pbO0vHWIADxZxgG7V
ra1vOwvtj3jSwq9vOqvPDmu6Dxm9Ru83ZmWPPDsKjyYtfK89GzFPQYqSbvfecu8qeopPKGvpzwkLq
A85+lQOUrm4zwAD8+9fEmSvB96iDt4xYk8XNLWPJx2Ej068qI9vJvpvBTMMT0TI4W85YLjvGVinL3
YH3+8/+PdPDgYR72TShC9PuCbPLnPSzzvcIw9LFXnvHKr6bvQMvy84YVFvJkyT72RZ5c6S4wzPEFJ
h7pph528Sa8APgQ+qbxcqsY8k1mqPdQEA70PTbw9pAhPvFth2D3YSgM944SnPWjFQD1M7qm8aB/qP
J6rgz3mPeM7hTe/vLAewTxnEis8b3PZufpWiLzoAwa95o2pPNrgAj7UvnQ80SsDPc5XpD3JYxY9Xs
NCPACzsDxph6w8gpCdOyZYrbwmLYq8PNu/u54kwbza5bq8C6n0PPqBfD0+DsY9wSUAvDjziTzm5bo
8ktYpvQihLDwAiyI9XHyTPHVFxzxL2/k8iTJrPUzRIr1pwpq6+AQ4vJzvWb3eKhK9YVzvPPiuGbuG
JeK8xDcGPUsAMr1YPvu8HVD8vN+OCb1XIaE8sdFQvUhCab1ig8063zxjPRnWyLyYNMk8uQR6vDOGf
7wv3la80GPuO2pGSbwyrcm9xDPIOzpIMjwEbQe9cPZOPPmb7bxqtvi7U4eSvEGtZruSQWo92bxhPX
xFAz3Mrn89RjaDPJ3ZHzyyTIY9vajSOzQrczw="
  ]
}
```


3. Использовать метод GET /indexes/{index} с UUID индекса 7aa9d013-9169-4b7b-a838-36f803466f9c для получения информации об индексе.

4. Использовать метод GET /indexes/{index}/objects с UUID индекса 7aa9d013-9169-4b7b-a838-36f803466f9c для получения списка объектов в индексе.

2.4 Модуль обработки потоковых видео

Для установки модуля обработки потоковых видео требуются следующие компоненты:

- архив stream.tar, содержащий соответствующий Docker-образ;

- конфигурационный файл stream.yml;
- сервер с установленной ОС Ubuntu\Astra Linux\Alt Linux, с доступом по протоколу SSH.

Для развертывания сервисов обработки потоковых видео требуется выполнить следующие действия:

1. Передать на сервер архив stream.tar, а также файл stream.yml.

2. Подключиться к серверу по протоколу SSH, используя логин и пароль пользователя ОС.

3. Загрузить Docker-образ из архива командой «docker load –i stream.tar». После завершения загрузки в терминале отобразится имя и тег образа, которые могут понадобиться для настройки docker-compose.

4. Запустить сервисы командой «docker compose –f stream.yml up –d».

Для удаления сервисов обработки потоковых видео требуется выполнить следующие действия:

1. Остановить запущенные сервисы командой «docker compose –f stream.yml down».

2. Удалить ранее загруженный Docker-образ командой «docker rmi {имя_образа:тег}».

3. Удалить архив stream.tar.

4. Удалить конфигурационный файл stream.yml.

Для проверки работоспособности сервисов обработки потоковых видео требуется выполнить следующие действия:

1. Открыть в браузере сервис по адресу http://localhost:4101/docs.

2. Использовать метод POST /tasks для запуска задачи обработки с телом запроса:

```
{
  "task": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "index": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "camera": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
}
```

3. Использовать метод GET /tasks/{task} с UUID задачи 3fa85f64-5717-4562-b3fc-2c963f66afa6 для отслеживания статуса обработки задачи.

4. Использовать метод DELETE /tasks/{task} с UUID задачи 3fa85f64-5717-4562-b3fc-2c963f66afa6 для удаления задачи из системы.

3 Настройка

3.1 Настройка модуля обработки изображений

Настройка модуля обработки изображений осуществляется через конфигурационные файлы docker-compose.yaml и minio.yaml:

1. Отправной точкой является выбор вида обработки сервиса:

- распознавание лиц – image_face;
- распознавание эмблем – image_emblem;
- распознавание военной техники – image_military_vehicle;
- распознавание известных мест – image_scene;
- распознавание надписей – image_text;
- распределение задач между сервисами – image_manager;
- обработка шаблонных изображений в S3 Minio – template_minio.

Выбранный вид обработки указывается в секции environment соответствующего сервиса в файле docker-compose.yaml через параметр PIPELINE.

2. Порт внутри контейнера, на котором работает сервис, по умолчанию – 80. Для указания порта на хосте используется ports в конфигурационном файле для каждого сервиса в формате:

```
ports:
  - "<порт хоста>:80"
```

3. Название сервиса должно соответствовать имени контейнера, указанному под ключом container_name, и иметь формат «{PIPELINE}.{порт_хоста}», где PIPELINE – значение параметра вида обработки, а порт_хоста – внешний порт, указанный в параметре ports.

4. Для сервиса с видом обработки image_manager необходимо указать переменную окружения SERVICES в секции environment. В этом параметре должен быть перечислен весь список подконтрольных сервисов, в формате, соответствующем значениям, указанным в поле container_name этих сервисов через пробел.

5. Для сервисов, работающих с шаблонными изображениями, необходимо указать следующие переменные окружения в секции environment файла docker-compose.yaml:

- ENDPOINT – в формате «<ip>:<порт>», должен соответствовать адресу доступа к хранилищу S3 Minio;
- ACCESS_KEY – значение должно совпадать с MINIO_ROOT_USER, указанным в конфигурации minio.yaml;
- SECRET_KEY – значение должно совпадать с MINIO_ROOT_PASSWORD, указанным в конфигурации minio.yaml;
- BUCKET_NAME – значение должно быть private-templates.

Пример конфигурационного файла docker-compose.yaml:

```yaml
version: '3.8'

networks:
  image:
    driver: bridge

services:
  template_minio:
    image: image:0.1
    init: true
    container_name: template_minio
    environment:
      PIPELINE: template_minio
      ENDPOINT: 10.239.16.89:9090
      ACCESS_KEY: root
      SECRET_KEY: password
      PUBLIC_BUCKET_NAME: public-templates
      PRIVATE_BUCKET_NAME: private-templates
    deploy:
      resources:
        limits:
          cpus: '2'
        reservations:
          cpus: '2'
          devices:
            - driver: nvidia
              device_ids: ['0']
              capabilities: [gpu]

  image_manager.7001:
    image: image:0.1
    init: true
    container_name: image_manager.7001
    ports: ['7001:80']
    networks: [image]
    restart: always
    environment:
      PIPELINE: image_manager
      SERVICES:
        image_face.7002
```

```
      image_emblem.7003
  depends_on:
    image_face.7002:
      condition: service_healthy
    image_emblem.7003:
      condition: service_healthy
  healthcheck:
    test: curl -f http://localhost:80/health
    start_period: 1h
  deploy:
    resources:
      limits:
        cpus: '2'
      reservations:
        cpus: '2'

image_face.7002:
  image: image:0.1
  init: true
  container_name: image_face.7002
  ports: ['7002:80']
  networks: [image]
  restart: always
  environment:
    PIPELINE: image_face
  healthcheck:
    test: curl -f http://localhost:80/health
    start_period: 1h
  deploy:
    resources:
      limits:
        cpus: '2'
      reservations:
        cpus: '2'
        devices:
          - driver: nvidia
            device_ids: ['0']
            capabilities: [gpu]

image_emblem.7003:
  image: image:0.1
  init: true
  container_name: image_emblem.7003
  ports: ['7003:80']
  networks: [image]
  restart: always
  environment:
    PIPELINE: image_emblem
    ENDPOINT: 10.239.16.89:9090
    ACCESS_KEY: root
    SECRET_KEY: password
    BUCKET_NAME: private-templates
  healthcheck:
    test: curl -f http://localhost:80/health
    start_period: 1h
  deploy:
    resources:
      limits:
        cpus: '2'
      reservations:
        cpus: '2'
        devices:
          - driver: nvidia
            device_ids: ['0']
```

```
capabilities: [gpu]
```

Примечание: раздел `deploy` необходим только при использовании видеокарт. Для использования контейнера на CPU необходимо убрать данный раздел и в `environment` указать : `PROVIDERS: CPUExecutionProvider.`

3.2 Настройка модуля обработки видео

Настройка модуля обработки видео осуществляется через конфигурационные файлы docker-compose.yaml и minio.yaml:

1. Отправной точкой является выбор вида обработки сервиса:

- распознавание лиц – video_face;
- распознавание эмблем – video_emblem;
- распознавание военной техники – video_military_vehicle;
- распознавание известных мест – video_scene;
- распознавание надписей – video_text;
- распределение задач между сервисами – video_manager;
- обработка шаблонных изображений в S3 Minio – template_minio.

Выбранный вид обработки указывается в секции environment соответствующего сервиса в файле docker-compose.yaml через параметр PIPELINE.

2. Порт внутри контейнера, на котором работает сервис, по умолчанию – 80. Для указания порта на хосте используется ports в конфигурационном файле для каждого сервиса в формате:

```
ports:
  - "<порт_хоста>:80"
```

3. Название сервиса должно соответствовать имени контейнера, указанному под ключом container_name, и иметь формат «{PIPELINE}.{порт_хоста}», где PIPELINE – значение параметра вида обработки, а порт_хоста – внешний порт, указанный в параметре ports.

4. Для сервиса с видом обработки video_manager необходимо указать переменную окружения SERVICES в секции environment. В этом параметре должен быть перечислен весь список подконтрольных сервисов, в формате,

соответствующем значениям, указанным в поле container_name этих сервисов через пробел.

5. Для сервисов, работающих с шаблонными изображениями, необходимо указать следующие переменные окружения в секции environment файла docker-compose.yaml:

- ENDPOINT – в формате «<ip>:<порт>», должен соответствовать адресу доступа к хранилищу S3 Minio;
- ACCESS_KEY – значение должно совпадать с MINIO_ROOT_USER, указанным в конфигурации minio.yaml;
- SECRET_KEY – значение должно совпадать с MINIO_ROOT_PASSWORD, указанным в конфигурации minio.yaml;
- BUCKET_NAME – значение должно быть private-templates.

6. Для сервисов необходимо корректно указать директорию, откуда будут поступать видеофайлы для последующего распознавания. Настройка осуществляется через параметр volumes в файле docker-compose.yaml в следующем формате «<путь_на_хосте>:/mnt», где путь_на_хосте – это абсолютный путь к целевой директории на сервере, содержащей видео.

Пример конфигурационного файла docker-compose.yaml:

```
version: '3.8'

networks:
  video:
    driver: bridge

services:
  template_minio:
    image: video:0.1
    init: true
    container_name: template_minio
    environment:
      PIPELINE: template_minio
      ENDPOINT: 10.239.16.89:9090
      ACCESS_KEY: root
      SECRET_KEY: password
      PUBLIC_BUCKET_NAME: public-templates
      PRIVATE_BUCKET_NAME: private-templates
    deploy:
      resources:
        limits:
          cpus: '2'
        reservations:
          cpus: '2'
          devices:
```

```
          - driver: nvidia
            device_ids: ['0']
            capabilities: [gpu]

video_manager.6001:
  image: video:0.1
  init: true
  container_name: video_manager.6001
  ports: ['6001:80']
  networks: [video]
  environment:
    PIPELINE: video_manager
    SERVICES:
      video_face.6002
      video_emblem.6003
  depends_on:
    video_face.6002:
      condition: service_healthy
    video_emblem.6003:
      condition: service_healthy
  healthcheck:
    test: curl -f http://localhost:80/health
    start_period: 1h
  deploy:
    resources:
      limits:
        cpus: '2'
      reservations:
        cpus: '2'

video_face.6002:
  image: video:0.1
  init: true
  container_name: video_face.6002
  ports: ['6002:80']
  networks: [video]
  volumes: [/storage/video:/mnt]
  environment:
    PIPELINE: video_face
  healthcheck:
    test: curl -f http://localhost:80/health
    start_period: 1h
  deploy:
    resources:
      limits:
        cpus: '2'
      reservations:
        cpus: '2'
        devices:
          - driver: nvidia
            device_ids: ['0']
            capabilities: [gpu]


video_emblem.6003:
  image: video:0.1
  init: true
  container_name: video_emblem.6003
  ports: ['6003:80']
  networks: [video]
  volumes: [/storage/video:/mnt]
  environment:
    PIPELINE: video_emblem
    ENDPOINT: 10.239.16.89:9090
```

```
      ACCESS_KEY: root
      SECRET_KEY: password
      BUCKET_NAME: private-templates
    healthcheck:
      test: curl -f http://localhost:80/health
      start_period: 1h
    deploy:
      resources:
        limits:
          cpus: '2'
        reservations:
          cpus: '2'
          devices:
            - driver: nvidia
              device_ids: ['0']
              capabilities: [gpu]
```

Примечание: раздел `deploy` необходим только при использовании видеокарт. Для использования контейнера на CPU необходимо убрать данный раздел и в `environment` указать : `PROVIDERS: CPUExecutionProvider.`

3.3 Настройка модуля хранения и индексации архивов изображений

Настройка модуля хранения и индексации архивов изображений осуществляется через конфигурационные файлы docker-compose.yaml и postgres.yaml:

1. Отправной точкой является выбор вида обработки сервиса:

- распознавание лиц – image_face_nano;
- матричный индекс – search_matrix;
- графовый индекс – search_diskann;
- гибридный индекс – search_hybrid;
- распределение запросов между сервисами – search_manager.

Выбранный вид обработки указывается в секции environment соответствующего сервиса в файле docker-compose.yaml через параметр PIPELINE.

2. Порт внутри контейнера, на котором работает сервис, по умолчанию – 80. Для указания порта на хосте используется ports в конфигурационном файле для каждого сервиса в формате:

```
ports:
  - "<порт_хоста>:80"
```

3. Название сервиса должно соответствовать имени контейнера, указанному под ключом container_name, и иметь формат «{PIPELINE}.{порт_хоста}», где PIPELINE – значение параметра вида обработки, а порт_хоста – внешний порт, указанный в параметре ports.

4. Для сервиса с видом обработки search_manager необходимо указать переменную окружения SERVICES в секции environment. В этом параметре должен быть перечислен весь список подконтрольных сервисов, в формате, соответствующем значениям, указанным в поле container_name этих сервисов через пробел.

5. Сервисы с видами обработок search_matrix, search_diskann и search_hybrid поддерживают следующие настройки, указываемые в секции environment:

- POSTGRESQL_URL – строка подключения к PostgreSQL в формате «postgresql://<логин>:<пароль>@<ip>:<порт>/<название_бд>»;
- DIRECTORY – определяет директорию на сервере, используемую для хранения индексов (должна совпадать с UUID-папки, смонтированной через volumes);
- MAX_CAPACITY – задает максимальную вместимость индекса.

6. Дополнительно, сервисы с видами обработок search_diskann и search_hybrid поддерживают следующие настройки, указываемые в секции environment:

- SEARCH_MEMORY_MAXIMUM – максимальный объём оперативной памяти (в GB), который может быть использован при выполнении поиска;
- BUILD_MEMORY_MAXIMUM – максимальный объём оперативной памяти (в GB), допустимый при построении индекса.

7. Дополнительно, в сервисе с видом обработки search_hybrid можно указать параметр MATRIX_CAPACITY — максимальный размер внутренней матрицы, используемой при гибридном поиске.

Пример конфигурационного файла docker-compose.yaml:

```yaml
version: '3.8'

networks:
  search:
    driver: bridge

services:
  image_face_nano.7775:
    image: search:0.1
    init: true
    container_name: image_face_nano.7775
    ports: ['7775:80']
    restart: always
    environment:
      PIPELINE: image_face_nano
    healthcheck:
      test: curl -f http://localhost:80/health
      start_period: 1h
    deploy:
      resources:
        limits:
          cpus: '2'
        reservations:
          cpus: '2'
          devices:
            - driver: nvidia
              device_ids: ['0']
              capabilities: [gpu]

  search_manager.8001:
    image: search:0.1
    init: true
    container_name: search_manager.8001
    ports: ['8001:80']
    networks: [search]
    restart: always
    environment:
      PIPELINE: search_manager
      POSTGRESQL_URL: postgres://USER:PASSWORD@10.239.16.89:15432/DB
      SERVICES:
        search_matrix.8002
        search_diskann.8003
    depends_on:
      search_matrix.8002:
        condition: service_healthy
      search_diskann.8003:
        condition: service_healthy
    healthcheck:
      test: curl -f http://localhost:80/health
      start_period: 24h
    deploy:
      resources:
        limits:
          cpus: '2'
        reservations:
          cpus: '2'

  search_matrix.8002:
    image: search:0.1
    init: true
    container_name: search_matrix.8002
    ports: ['8002:80']
    networks: [search]
```

```
    volumes: [/storage/2093999f-033d-4367-b578-8ca5f735411f:/2093999f-033d-
4367-b578-8ca5f735411f]
    restart: always
    environment:
      PIPELINE: search_matrix
      POSTGRESQL_URL: postgresql://USER:PASSWORD@10.239.16.89:15432/DB
      DIRECTORY: 2093999f-033d-4367-b578-8ca5f735411f
    healthcheck:
      test: curl -f http://localhost:80/health
      start_period: 24h
    deploy:
      resources:
        limits:
          cpus: '8'
        reservations:
          cpus: '8'

  search_diskann.8003:
    image: search:0.1
    init: true
    container_name: search_diskann.8003
    ports: ['8003:80']
    networks: [search]
    volumes: [/storage/26af8925-1d25-453f-b555-0ab0b3a54567:/26af8925-1d25-
453f-b555-0ab0b3a54567]
    restart: always
    environment:
      PIPELINE: search_diskann
      POSTGRESQL_URL: postgresql://USER:PASSWORD@10.239.16.89:15432/DB
      DIRECTORY: 26af8925-1d25-453f-b555-0ab0b3a54567
    healthcheck:
      test: curl -f http://localhost:80/health
      start_period: 24h
    deploy:
      resources:
        limits:
          cpus: '8'
        reservations:
          cpus: '8'
```

Примечание: раздел `deploy` необходим только при использовании видеокарт. Для использования контейнера на CPU необходимо убрать данный раздел и в `environment` указать : `PROVIDERS: CPUExecutionProvider.`

3.4 Настройка модуля обработки потоковых видео

Настройка модуля обработки потоковых видео осуществляется через конфигурационный файл docker-compose.yaml:

1. Отправной точкой является выбор вида обработки сервиса:

- построение векторов лиц – crop_face;

- распознавание лиц на потоковых видео в реальном времени – stream_realtime_face;

- распределение задач между сервисами распознавания лиц на потоковых видео в реальном времени – search_realtime_manager;

- распознавание лиц на архивных потоковых видео – stream_archive_face;

- распределение задач между сервисами распознавания лиц на архивных потоковых видео – search_archive_manager.

Выбранный вид обработки указывается в секции environment соответствующего сервиса в файле docker-compose.yaml через параметр PIPELINE.

2. Порт внутри контейнера, на котором работает сервис, по умолчанию – 80. Для указания порта на хосте используется ports в конфигурационном файле для каждого сервиса в формате:

```
ports:
  - "<порт_хоста>:80"
```

3. Название сервиса должно соответствовать имени контейнера, указанному под ключом container_name, и иметь формат «{PIPELINE}.{порт_хоста}», где PIPELINE – значение параметра вида обработки, а порт_хоста – внешний порт, указанный в параметре ports.

4. Для сервисов распределения задач stream_realtime_manager и stream_archive_manager необходимо указать переменную окружения SERVICES в секции environment. В этом параметре должен быть перечислен весь список подконтрольных сервисов, в формате, соответствующем значениям, указанным в поле container_name этих сервисов через пробел.

5. Дополнительно, для сервисов stream_realtime_manager и stream_archive_manager необходимо указать переменную окружения TARGET_PIPELINE в секции environment со значениями:

для stream_realtime_manager – stream_realtime_face;

для stream_archive_manager – stream_archive_face.

6. Сервисы с видами обработок stream_realtime_face и stream_archive_face поддерживают следующие настройки, указываемые в секции environment:

- API_URL – URL-адрес доступа к сервису получения ссылок на потоковые видео;

- API_USERNAME – логин для доступа к сервису получения ссылок на потоковые видео;

- API_PASSOWRD – пароль для доступа к сервису получения ссылок на потоковые видео;

- FPS – количество кадров в секунду, подлежащих обработке;

- PROVIDERS – наименование провайдера обработки моделей;

- MULTI_EVENT_MODE – флаг активации режима сбора нескольких событий по одному треку объекта;

- FIRST_EVENT_DELAY – время задержки при фиксации первого события по объекту;

- SUB_EVENT_DELAY – время задержки при фиксации второго и последующего событий по тому же объекту;

- EXIT_EVENT – признак необходимости фиксации события при выходе объекта из кадра;

- PORTAL_URL – URL-адрес доступа к ресурсу поратла WebFace;

- PORTAL_USERNAME – логин для доступа к порталу WebFace;

- PORTAL_PASSWORD – пароль для доступа к порталу WebFace.

7. Дополнительно, для сервиса с видом обработки stream_archive_face можно указать параметр SPEED_FACTOR – он определяет степень ускорения воспроизведения архивного потокового видео.

Пример конфигурационного файла docker-compose.yaml:

```
version: '3.8'

networks:
  stream_realtime:
    driver: bridge
    name: stream_realtime

services:
  crop_face.5501:
    image: stream:0.1
    init: true
    container_name: crop_face.5501
    ports: ['5501:80']
    restart: always
    environment:
```

```
    PIPELINE: crop_face
  healthcheck:
    test: curl -f http://localhost:80/health
    start_period: 1h
  deploy:
    resources:
      limits:
        cpus: '2'
      reservations:
        cpus: '2'
        devices:
          - driver: nvidia
            device_ids: ['0']
            capabilities: [gpu]

stream_realtime_manager.4002:
  image: stream:0.1
  init: true
  container_name: stream_realtime_manager.4002
  ports: ['4002:80']
  networks: [stream_realtime]
  restart: always
  environment:
    PIPELINE: stream_realtime_manager
    TARGET_PIPELINE: stream_realtime_face
    SERVICES:
      stream_realtime_face.4003
      stream_realtime_face.4004

stream_realtime_face.4003:
  image: stream:0.1
  init: true
  container_name: stream_realtime_face.4003
  ports: ['4003:80']
  networks: [stream_realtime]
  restart: always
  environment:
    PIPELINE: stream_realtime_face
    API_URL: https://api-test.enpv.ru/cctv/api/camera/url
    API_USERNAME: username
    API_PASSWORD: password
    PROVIDERS: CPUExecutionProvider
    PORTAL_URL: http://10.7.102.102:56000
    PORTAL_USERNAME: username
    PORTAL_PASSWORD: password

stream_realtime_face.4004:
  image: stream:0.1
  init: true
  container_name: stream_realtime_face.4004
  ports: ['4004:80']
  networks: [stream_realtime]
  restart: always
  environment:
    PIPELINE: stream_realtime_face
    API_URL: https://api-test.enpv.ru/cctv/api/camera/url
    API_USERNAME: username
    API_PASSWORD: password
    PROVIDERS: CPUExecutionProvider
    PORTAL_URL: http://10.7.102.102:56000
    PORTAL_USERNAME: username
    PORTAL_PASSWORD: password
```

Примечание: раздел `deploy` необходим только при использовании видеокарт. Для использования контейнера на CPU необходимо убрать данный раздел и в `environment` указать : `PROVIDERS: CPUExecutionProvider.`

4 Интерфейсы

4.1 Интерфейс обработки изображений

Входные и выходные данные запросов представляют собой JSON объекты.

Для постановки задачи обработки изображений необходимо отправить POST-запрос «/tasks» со структурой, представленной в таблице 1. Структура тела ответа на запрос представлена в таблице 2.

Таблица 1 – структура объекта, передаваемого в запросе на создание задачи

Поле	Описание
task	UUID задачи (если не указать, то сгенерируется автоматически)
pipelines	Список видов обработок, к примеру: 1. image_face 2. image_emblem 3. image_military_vehicle 4. image_scene 5. image_text
images	Список изображений, представленных в формате base64

Пример запроса POST /tasks:

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "pipelines": [
        "image_face",
        "image_emblem"
    ],
    "images": [
        "<изображение в base64>",
        "<изображение в base64>"
        "<изображение в base64>"
    ]
}
```

Таблица 2 – структура объекта, передаваемого в теле ответа на создание задачи

Поле	Описание
task	UUID задачи

Пример тела ответа (код состояния – 201):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c"
}
```

Для получения списка задач необходимо отправить GET-запрос «/tasks». Структура тела ответа на запрос представлена в таблице 3.

Таблица 3 – структура объекта, передаваемого в теле ответа на получение списка задач

Поле	Описание
tasks	Список UUID задач

Пример тела ответа (код состояния – 200):

```
{
    "tasks": [
        "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
        "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ]
}
```

Таблица 4 – структура объекта, передаваемого в теле ответа на получение результатов задачи

Поле	Описание
task	UUID задачи
status	Статус задачи с возможными значениями: 1. pending – задача находится в режиме ожидания 2. in_progress – задача находится в обработке 3. completed – задача завершена успешно 4. failed – задача завершена с ошибкой
progress	Прогресс выполнения задачи в процентах (от 0 до 100)
message	Сообщение с описанием состояния задачи в случае завершения с ошибкой
pipelines	Список видов обработок, которые задача должна выполнить
failed_pipelines	Список видов обработок, которые задача не смогла выполнить
images	Список изображений с результатами распознавания

Таблица 5 – структура объекта, описывающая список изображений с результатами распознавания

Поле	Описание
objects	Список распознанных объектов на изображении
tags	Список распознанных тегов на изображении
success	Флаг успеха обработки конкретного изображения (true или false)
error	Сообщение с ошибкой в случае отсутствия успеха обработки конкретного изображения

Таблица 6 – структура объекта, описывающая список распознанных объектов на изображении

Поле	Описание
classifier_uuid	UUID классификатора
classifier_name	Название классификатора
class_uuids	Список UUID, описывающий составной класс
class_names	Список названий классов, описывающий составной класс
template	Файловый путь шаблона, к которому относится объект
score	Оценка принадлежности объекта к классу (от 0 до 1)
has_quality	Оценка качества объекта (true или false)
crop	Вырезанный фрагмент объекта в base64
vector	Вектор объекта в base64
model	UUID модели используемой для построения вектора объекта
bbox	Координаты выделенной области объекта

Таблица 7 – структура объекта, описывающая координаты выделенной области объекта

Поле	Описание
x	Координата X левого верхнего угла области
y	Координата Y левого верхнего угла области
width	Ширина области в пикселях
height	Высота области в пикселях

Таблица 8 – структура объекта, описывающая список распознанных тегов на изображении

Поле	Описание
classifier_uuid	UUID классификатора
classifier_name	Название классификатора
class_uuid	UUID класса
class_name	Название класса
score	Оценка принадлежности объекта к классу (от 0 до 1)

Пример тела ответа (код состояния – 200):


```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "status": "completed",
    "progress": 100,
    "message": null,
    "pipelines": [
        "image_face",
        "image_emblem"
    ],
    "failed_pipelines": [
        "image_face"
    ],
    "images": [
        {
```

```
            "objects": [],
            "tags": [],
            "success": false,
            "error": "ImageDecodeError"
        },
            {
            "objects": [
                {
                    "classifier_uuid": "a322bf47-7a32-43ce-95c0-
017d491cbaed",
                    "classifier_name": "Классификатор",
                    "class_uuids": [
                        "703307e0-1b49-440c-b557-70d95c48dc9b",
                        "27054b71-63fd-40b0-8002-5bef4d97f990"
                    ],
                    "class_names": [
                        "Класс1",
                        "Класс2"
                    ],
                    "template": "Класс1/Класс2/Эмблема_001.jpg",
                    "score": 0.875311023,
                    "has_quality": true,
                    "crop": "<вырезанный фрагмент объекта в base64>",
                    "vector": "<вектор фрагмента объекта в base64>",
                    "model": "39187191-7436-4cbd-8989-d0666e9d45c7",
                    "bbox": {
                        "x": 12,
                        "y": 154,
                        "width": 272,
                        "height": 349
                    }
                }
            ]
            "tags": [],
            "success": true,
            "error": null
        }
    ]
}
```


Для удаления конкретной задачи необходимо отправить DELETE-запрос «/tasks/{task}», где {task} – UUID задачи. В случае успешного удаления задачи в ответ придет код состояния 204 без тела ответа.

Таблица 9 – структура объекта, передаваемого в теле ответа на просмотр списка доступных видов обработок

Поле	Описание
pipelines	Список доступных видов обработок

Пример тела ответа (код состояния – 200):


```
{
    "pipelines": [
        "image_face",
        "image_emblem"
    ]
}
```

4.2 Интерфейс обработки видео

Входные и выходные данные запросов представляют собой JSON объекты.

Для постановки задачи обработки видео необходимо отправить POST-запрос «/tasks» со структурой, представленной в таблице 10. Структура тела ответа на запрос представлена в таблице 11.

Таблица 10 – структура объекта, передаваемого в запросе на создание задачи

Поле	Описание
task	UUID задачи (если не указать, то сгенерируется автоматически)
pipelines	Список видов обработок, к примеру: 1. video_face 2. video_emblem 3. video_military_vehicle 4. video_scene 5. video_text
filename	Название файла

Пример запроса POST /tasks:

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "pipelines": [
        "video_face",
        "video_emblem"
    ],
    "filename": "video.mp4"
}
```

Таблица 11 – структура объекта, передаваемого в теле ответа на создание задачи

Поле	Описание
task	UUID задачи

Пример тела ответа (код состояния – 201):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c"
}
```

Для получения списка задач необходимо отправить GET-запрос «/tasks». Структура тела ответа на запрос представлена в таблице 12.

Таблица 12 – структура объекта, передаваемого в теле ответа на получение списка задач

Поле	Описание
tasks	Список UUID задач

Пример тела ответа (код состояния – 200):

```
{
    "tasks": [
        "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
        "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ]
}
```

Таблица 13 – структура объекта, передаваемого в теле ответа на получение результатов задачи

Поле	Описание
task	UUID задачи
status	Статус задачи с возможными значениями: 1. pending – задача находится в режиме ожидания 2. in_progress – задача находится в обработке 3. completed – задача завершена успешно 4. failed – задача завершена с ошибкой
progress	Прогресс выполнения задачи в процентах (от 0 до 100)
message	Сообщение с описанием состояния задачи в случае завершения с ошибкой
pipelines	Список видов обработок, которые задача должна выполнить
failed_pipelines	Список видов обработок, которые задача не смогла выполнить
filename	Название файла
objects	Список распознанных объектов на видео
tags	Список распознанных тегов на видео

Таблица 14 – структура объекта, описывающая список распознанных объектов на видео

Поле	Описание
classifier_uuid	UUID классификатора
classifier_name	Название классификатора
class_uuids	Список UUID, описывающий составной класс
class_names	Список названий классов, описывающий составной класс
template	Файловый путь шаблона, к которому относится объект
score	Оценка принадлежности объекта к классу (от 0 до 1)
has_quality	Оценка качества объекта (true или false)
crop	Вырезанный фрагмент объекта в base64
vector	Вектор объекта в base64
model	UUID модели используемой для построения вектора объекта
intervals	Список временных интервалов объекта

Таблица 15 – структура объекта, описывающая список временных интервалов объекта

Поле	Описание
start_frame	Индекс начального кадра
end_frame	Индекс конечного кадра
start_time	Время начала (мс)
end_time	Время окончания (мс)
start_box	Координаты выделенной области объекта на начальном кадре
end_box	Координаты выделенной области объекта на конечном кадре

Таблица 16 – структура объекта, описывающая координаты выделенной области объекта

Поле	Описание
x	Координата X левого верхнего угла области
y	Координата Y левого верхнего угла области
width	Ширина области в пикселях
height	Высота области в пикселях

Таблица 17 – структура объекта, описывающая список распознанных тегов на видео

Поле	Описание
classifier_uuid	UUID классификатора
classifier_name	Название классификатора
class_uuid	UUID класса
class_name	Название класса
score	Оценка принадлежности объекта к классу (от 0 до 1)
intervals	Список временных интервалов тега

Таблица 18 – структура объекта, описывающая список временных интервалов тега

Поле	Описание
start_frame	Индекс начального кадра
end_frame	Индекс конечного кадра
start_time	Время начала (мс)
end_time	Время окончания (мс)

Пример тела ответа (код состояния – 200):


```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "status": "completed",
    "progress": 100,
```

```
    "message": null,
    "pipelines": [
        "video_face",
        "video_emblem"
    ],
    "failed_pipelines": [
        "video_face"
    ],
    "filename": "video.mp4",
    "objects": [
        {
            "classifier_uuid": "a322bf47-7a32-43ce-95c0-017d491cbaed",
            "classifier_name": "Классификатор",
            "class_uuids": [
                "703307e0-1b49-440c-b557-70d95c48dc9b",
                "27054b71-63fd-40b0-8002-5bef4d97f990"
            ],
            "class_names": [
                "Класс1",
                "Класс2"
            ],
            "template": "Класс1/Класс2/Эмблема_001.jpg",
            "score": 0.875311023,
            "has_quality": true,
            "crop": "<вырезанный фрагмент объекта в base64>",
            "vector": "<вектор фрагмента объекта в base64>",
            "model": "39187191-7436-4cbd-8989-d0666e9d45c7",
            "intervals": [
                {
                    "start_frame": 0,
                    "end_frame": 24,
                    "start_time": 0,
                    "end_time": 1000,
                    "start_box": {
                        "x": 12,
                        "y": 154,
                        "width": 272,
                        "height": 349
                    },
                    "end_box": {
                        "x": 14,
                        "y": 162,
                        "width": 320,
                        "height": 386
                    }
                }
            ]
        }
    ],
    "tags": []
}
```


Для удаления конкретной задачи необходимо отправить DELETE-запрос «/tasks/{task}», где {task} – UUID задачи. В случае успешного удаления задачи в ответ придет код состояния 204 без тела ответа.

Таблица 19 – структура объекта, передаваемого в теле ответа на просмотр списка доступных видов обработок

Поле	Описание
pipelines	Список доступных видов обработок

Пример тела ответа (код состояния – 200):

```
{
      "pipelines": [
            "video_face",
            "video_emblem"
      ]
}
```

4.3 Интерфейс построения векторов

Входные и выходные данные запросов представляют собой JSON объекты.

Для постановки задачи построения векторов необходимо отправить POST-запрос «/tasks» со структурой, представленной в таблице 20. Структура тела ответа на запрос представлена в таблице 21.

Таблица 20 – структура объекта, передаваемого в запросе на создание задачи

Поле	Описание
task	UUID задачи (если не указать, то сгенерируется автоматически)
model	UUID модели
crops	Список вырезанных фрагментов, представленных в формате base64

Пример запроса POST /tasks:

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "model": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "crops": [
        "<вырезанный фрагмент в base64>",
        "<вырезанный фрагмент в base64>"
        "<вырезанный фрагмент в base64>"
    ]
}
```

Таблица 21 – структура объекта, передаваемого в теле ответа на создание задачи

Поле	Описание
task	UUID задачи

Пример тела ответа (код состояния – 201):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c"
}
```

Для получения списка задач необходимо отправить GET-запрос «/tasks». Структура тела ответа на запрос представлена в таблице 22.

Таблица 22 – структура объекта, передаваемого в теле ответа на получение списка задач

Поле	Описание
tasks	Список UUID задач

Пример тела ответа (код состояния – 200):

```
{
    "tasks": [
        "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
        "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ]
}
```

Таблица 23 – структура объекта, передаваемого в теле ответа на получение результатов задачи

Поле	Описание
task	UUID задачи
status	Статус задачи с возможными значениями: 1. pending – задача находится в режиме ожидания 2. in_progress – задача находится в обработке 3. completed – задача завершена успешно 4. failed – задача завершена с ошибкой
progress	Прогресс выполнения задачи в процентах (от 0 до 100)
message	Сообщение с описанием состояния задачи в случае завершения с ошибкой
model	UUID модели
crops	Список вырезанных фрагментов с результатами обработки

Таблица 24 – структура объекта, описывающая список вырезанных фрагментов с результатами обработки

Поле	Описание
vector	Построенный вектор в base64
success	Флаг успеха обработки конкретного вырезанного фрагмента (true или false)
error	Сообщение с ошибкой в случае отсутствия успеха обработки конкретного вырезанного фрагмента

Пример тела ответа (код состояния – 200):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "status": "completed",
    "progress": 100,
    "message": null,
    "model": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "crops": [
        {
            "vector": null,
```

```
            "success": false,
            "error": "VectorDecodeError"
        },
        {
            "vector": "<построенный вектор в base64>",
            "success": true,
            "error": null
        }
    ]
}
```

Для удаления конкретной задачи необходимо отправить DELETE-запрос «/tasks/{task}», где {task} – UUID задачи. В случае успешного удаления задачи в ответ придет код состояния 204 без тела ответа.

Таблица 25 – структура объекта, передаваемого в теле ответа на просмотр списка доступных моделей

Поле	Описание
models	Список доступных моделей

Пример тела ответа (код состояния – 200):

```
{
    "models": [
        "27054b71-63fd-40b0-8002-5bef4d97f990",
        "703307e0-1b49-440c-b557-70d95c48dc9b"
    ]
}
```

4.4 Интерфейс обработки потоковых видео в реальном времени

Входные и выходные данные запросов представляют собой JSON объекты.

Для постановки задачи обработки потокового видео в реальном времени необходимо отправить POST-запрос «/tasks» со структурой, представленной в таблице 26. Структура тела ответа на запрос представлена в таблице 27.

Таблица 26 – структура объекта, передаваемого в запросе на создание задачи

Поле	Описание
task	UUID задачи
index	UUID индекса
camera	UUID камеры

Пример запроса POST /tasks:

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "index": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "camera": "a322bf47-7a32-43ce-95c0-017d491cbaed"
```

```
}
```

Таблица 27 – структура объекта, передаваемого в теле ответа на создание задачи

Поле	Описание
task	UUID задачи

Пример тела ответа (код состояния – 201):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c"
}
```

Для получения списка задач необходимо отправить GET-запрос «/tasks». Структура тела ответа на запрос представлена в таблице 28.

Таблица 28 – структура объекта, передаваемого в теле ответа на получение списка задач

Поле	Описание
tasks	Список UUID задач

Пример тела ответа (код состояния – 200):

```
{
    "tasks": [
        "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
        "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ]
}
```

Для получения результатов конкретной задачи необходимо отправить GET-запрос «/tasks/{task}», где {task} – UUID задачи. Структура тела ответа на запрос представлена в таблице 29.

Таблица 29 – структура объекта, передаваемого в теле ответа на получение результатов задачи

Поле	Описание
task	UUID задачи
index	UUID индекса
camera	UUID камеры
status	Статус задачи с возможными значениями: 1. in_progress – задача находится в обработке 2. failed – задача завершена с ошибкой

Пример тела ответа (код состояния – 200):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "index": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "camera": "a322bf47-7a32-43ce-95c0-017d491cbaed",
    "status": "in_progress"
```

```
}
```

Для удаления конкретной задачи необходимо отправить DELETE-запрос «/tasks/{task}», где {task} – UUID задачи. В случае успешного удаления задачи в ответ придет код состояния 204 без тела ответа.

Для добавления дополнительных сервисов необходимо отправить POST-запрос «/services» со структурой, представленной в таблице 30. Структура тела ответа на запрос представлена в таблице 31.

Таблица 30 – структура объекта, передаваемого в запросе на добавление сервиса

Поле	Описание
pipeline	Название вида обработки
port	Порт сервиса (от 1 до 65535)

Пример запроса POST /services:

```
{
    "pipeline": "stream_realtime_face",
    "port": 8000
}
```

Таблица 31 – структура объекта, передаваемого в теле ответа на добавление сервиса

Поле	Описание
service	Название сервиса

Пример тела ответа (код состояния – 201):

```
{
    "service": "stream_realtime_face.8000"
}
```

Таблица 32 – структура объекта, передаваемого в теле ответа на просмотр списка доступных сервисов

Поле	Описание
services	Список доступных сервисов

Пример тела ответа (код состояния – 200):

```
{
    "services": [
        "stream_realtime_face.7000",
        "stream_realtime_face.7001"
    ]
}
```

4.5 Интерфейс обработки архивных потоковых видео

Входные и выходные данные запросов представляют собой JSON объекты.

Для постановки задачи обработки архивного потокового видео необходимо отправить POST-запрос «/tasks» со структурой, представленной в таблице 33. Структура тела ответа на запрос представлена в таблице 34.

Таблица 33 – структура объекта, передаваемого в запросе на создание задачи

Поле	Описание
task	UUID задачи
index	UUID индекса
camera	UUID камеры
from_timestamp_ms	Начальная граница индексирования в архиве
to_timestamp_ms	Конечная граница индексирования в архиве

Пример запроса POST /tasks:

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "index": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "camera": "a322bf47-7a32-43ce-95c0-017d491cbaed",
    "from_timestamp_ms": "1716826136132",
    "to_timestamp_ms": "1716826170784"
}
```

Таблица 34 – структура объекта, передаваемого в теле ответа на создание задачи

Поле	Описание
task	UUID задачи

Пример тела ответа (код состояния – 201):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c"
}
```

Для получения списка задач необходимо отправить GET-запрос «/tasks». Структура тела ответа на запрос представлена в таблице 35.

Таблица 35 – структура объекта, передаваемого в теле ответа на получение списка задач

Поле	Описание
tasks	Список UUID задач

Пример тела ответа (код состояния – 200):

```
{
    "tasks": [
        "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
        "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ]
}
```

Для получения результатов конкретной задачи необходимо отправить GET-запрос «/tasks/{task}», где {task} – UUID задачи. Структура тела ответа на запрос представлена в таблице 36.

Таблица 36 – структура объекта, передаваемого в теле ответа на получение результатов задачи

Поле	Описание
task	UUID задачи
index	UUID индекса
camera	UUID камеры
status	Статус задачи с возможными значениями: 1. in_progress – задача находится в обработке 2. completed – задача завершена успешно 3. failed – задача завершена с ошибкой
progress	Прогресс выполнения задачи в процентах (от 0 до 100)
estimated_time	Оценка времени завершения задачи в секундах

Пример тела ответа (код состояния – 200):

```
{
    "task": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "index": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "camera": "a322bf47-7a32-43ce-95c0-017d491cbaed",
    "status": "in_progress",
    "progress": 87,
    "estimated_time": 1072
}
```

Для удаления конкретной задачи необходимо отправить DELETE-запрос «/tasks/{task}», где {task} – UUID задачи. В случае успешного удаления задачи в ответ придет код состояния 204 без тела ответа.

Для добавления дополнительных сервисов необходимо отправить POST-запрос «/services» со структурой, представленной в таблице 37. Структура тела ответа на запрос представлена в таблице 38.

Таблица 37 – структура объекта, передаваемого в запросе на добавление сервиса

Поле	Описание
pipeline	Название вида обработки
port	Порт сервиса (от 1 до 65535)

Пример запроса POST /services:

```
{
    "pipeline": "stream_archive_face",
    "port": 8000
}
```

Таблица 38 – структура объекта, передаваемого в теле ответа на добавление сервиса

Поле	Описание
service	Название сервиса

Пример тела ответа (код состояния – 201):

```
{
    "service": "stream_archive_face.8000"
}
```

Таблица 39 – структура объекта, передаваемого в теле ответа на просмотр списка доступных сервисов

Поле	Описание
services	Список доступных сервисов

Пример тела ответа (код состояния – 200):

```
{
    "services": [
        "stream_archive_face.7000",
        "stream_archive_face.7001"
    ]
}
```

4.6 Интерфейс работы с поисковыми индексами

Входные и выходные данные запросов представляют собой JSON объекты.

Для создания индекса необходимо отправить POST-запрос «/indexes» со структурой, представленной в таблице 40. Структура тела ответа на запрос представлена в таблице 41.

Таблица 40 – структура объекта, передаваемого в запросе на создание индекса

Поле	Описание
index	UUID индекса (если не указать, то сгенерируется автоматически)
pipeline	Тип индекса с возможными значениями: 1. search_matrix – матричный индекс 2. search_diskann – графовый индекс 3. search_hybrid – гибридный индекс
model	UUID модели

Пример запроса POST /indexes:

```
{
    "index": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "pipeline": "search_matrix",
    "model": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5"
}
```

Таблица 41 – структура объекта, передаваемого в теле ответа на создание индекса

Поле	Описание
index	UUID индекса

Пример тела ответа (код состояния – 201):

```
{
    "index": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c"
}
```

Для получения списка индексов необходимо отправить GET-запрос «/indexes». Структура тела ответа на запрос представлена в таблице 42.

Таблица 42 – структура объекта, передаваемого в теле ответа на получение списка индексов

Поле	Описание
indexes	Список UUID индексов

Пример тела ответа (код состояния – 200):

```
{
    "indexes": [
        "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
        "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ]
}
```

Для получения состояния конкретного индекса необходимо отправить GET-запрос «/indexes/{index}», где {index} – UUID индекса. Структура тела ответа на запрос представлена в таблице 43.

Таблица 43 – структура объекта, передаваемого в теле ответа на получение состояния индекса

Поле	Описание
index	UUID индекса
pipeline	Тип индекса
model	UUID модели
directory	UUID директории
status	Статус индекса с возможными значениями: 1. not_built – индекс не построен 2. building – индекс строится

	3. built – индекс построен 4. build_failed – ошибка построения индекса 5. build_not_supported – индекс не поддерживает построение

Пример тела ответа (код состояния – 200):

```
{
    "index": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
    "pipeline": "search_matrix",
    "model": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "directory": "a9c0f70a-50e4-427b-ae9a-21f6bf0b7079",
    "status": "building"
}
```

Для удаления конкретного индекса необходимо отправить DELETE-запрос «/indexes/{index}», где {index} – UUID индекса. В случае успешного удаления индекса в ответ придет код состояния 204 без тела ответа.

Для добавления объектов в индекс необходимо отправить POST-запрос «/indexes/{index}/objects», где {index} – UUID индекса, со структурой, представленной в таблицах 44-45. Структура тела ответа на запрос представлена в таблицах 46-47.

Таблица 44 – структура объекта, передаваемого в запросе на добавление объектов в индекс

Поле	Описание
model	UUID модели
objects	Список объектов с векторами

Таблица 45 – структура объекта, описывающая список объектов с векторами

Поле	Описание
object	UUID объекта (если не указать, то сгенерируется автоматически)
vector	Вектор объекта в base64

Пример запроса POST /indexes/{index}/objects:

```
{
    "model": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "objects": [
        {
            "object": null,
            "vector": "<вектор объекта в base64>"
        },
        {
            "object": null,
            "vector": "<вектор объекта в base64>"
```

```
        }
}
```

Таблица 46 – структура объекта, передаваемого в теле ответа на добавление объектов в индекс

Поле	Описание
objects	Список объектов с флагами успешности добавления

Таблица 47 – структура объекта, описывающая список объектов с флагами успешности добавления

Поле	Описание
object	UUID объекта в случае успешного добавления
success	Флаг успеха добавления объекта в индекс (true или false)
error	Сообщение с ошибкой в случае отсутствия успеха добавления объекта в индекс

Пример тела ответа (код состояния – 200):

```
{
    "objects": [
        {
            "object": "0777e1a3-1201-4ba1-9d22-8b5a8eae8649",
            "success": true,
            "error": null
        },
        {
            "object": null,
            "success": false,
            "error": "VectorDecodeError"
        }
}
```

Для получения списка объектов в конкретном индексе необходимо отправить GET-запрос «/indexes/{index}/objects», где {index} – UUID индекса. Структура тела ответа на запрос представлена в таблице 48.

Таблица 48 – структура объекта, передаваемого в теле ответа на получение списка объектов в индексе

Поле	Описание
objects	Список UUID объектов в индексе

Пример тела ответа (код состояния – 200):

```
{
    "objects": [
        "0777e1a3-1201-4ba1-9d22-8b5a8eae8649",
        "39187191-7436-4cbd-8989-d0666e9d45c7"
    ]
}
```

Для удаления объектов из индекса необходимо отправить DELETE-запрос «/indexes/{index}/objects», где {index} – UUID индекса, со структурой, представленной в таблице 49. Структура тела ответа на запрос представлена в таблицах 50-51.

Таблица 49 – структура объекта, передаваемого в запросе на удаление объектов из индекса

Поле	Описание
objects	Список UUID объектов

Пример запроса DELETE /indexes/{index}/objects:

```
{
	"objects": [
		"0777e1a3-1201-4ba1-9d22-8b5a8eae8649",
		"703307e0-1b49-440c-b557-70d95c48dc9b"
	]
}
```

Таблица 50 – структура объекта, передаваемого в теле ответа на удаление объектов из индекса

Поле	Описание
objects	Список объектов с флагами успешности удаления

Таблица 51 – структура объекта, описывающая список объектов с флагами успешности удаления

Поле	Описание
success	Флаг успеха удаления объекта из индекса (true или false)
error	Сообщение с ошибкой в случае отсутствия успеха удаления объекта из индекса

Пример тела ответа (код состояния – 200):

```
{
    "objects": [
        {
            "success": true,
            "error": null
        },
        {
            "success": false,
            "error": "ObjectNotFoundError"
        }
}
```

Таблица 52 – структура объекта, передаваемого в теле ответа на запуск процесса построения индекса

Поле	Описание
index	UUID индекса

Пример тела ответа (код состояния – 202):

```
{
    "index": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c"
}
```

Для проведения поиска похожих объектов по индексам необходимо отправить POST-запрос «/indexes/search» со структурой, представленной в таблице 53. Структура тела ответа на запрос представлена в таблице 54-56.

Таблица 53 – структура объекта, передаваемого в запросе на поиск похожих объектов по индексам

Поле	Описание
indexes	Список UUID индексов
model	UUID модели
top_k	Количество похожих объектов, которое необходимо вывести (от 1 и более)
vectors	Список векторов в base64

Пример запроса POST /indexes/search:

```
{
    "indexes": [
        "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
        "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ],
    "model": "ccc9a533-38cd-46bd-8bd5-dec1d4893ba5",
    "top_k": 2,
    "vectors": [
        "<вектор в base64>",
        "<вектор в base64>",
        "<вектор в base64>"
    ]
}
```

Таблица 54 – структура объекта, передаваемого в теле ответа на поиск похожих объектов по индексам

Поле	Описание
vectors	Список векторов с флагами успешности использования в поиске
objects	Список похожих объектов

Таблица 55 – структура объекта, описывающая список векторов с флагами успешности использования в поиске

Поле	Описание
success	Флаг успеха использования вектора в поиске (true или false)
error	Сообщение с ошибкой в случае отсутствия успеха использования вектора в поиске

Таблица 56 – структура объекта, описывающая список похожих объектов

Поле	Описание
index	UUID индекса

object	UUID объекта
similarity	Оценка сходства
normalized_similarity	Нормализованная оценка сходства (от 0 до 1)
similarity_level	Уровень сходства с возможными значениями: 1. low – низкий уровень 2. medium – средний уровень 3. high – высокий уровень

Пример тела ответа (код состояния – 200):


```
{
    "vectors": [
        {
            "success": true,
            "error": null
        },
        {
            "success": false,
            "error": "VectorDecodeError"
        },
        {
            "success": true,
            "error": null
        }
    ],
    "objects": [
        {
            "index": "6ecacf5f-e40f-49f0-9b95-6e07cd70515c",
            "object": "27054b71-63fd-40b0-8002-5bef4d97f990",
            "similarity": 0.863123,
            "normalized_similarity": 1.0,
            "similarity_level": "high"
        },
        {
            "index": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
            "object": "703307e0-1b49-440c-b557-70d95c48dc9b",
            "similarity": 0.743841,
            "normalized_similarity": 0.987682,
            "similarity_level": "medium"
        }
    ]
}
```