

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

**ОТЧЕТ**

**по практической подготовке**

**в рамках реализации проекта «Цифровая кафедра»**

|  |  |
| --- | --- |
| Обучающийся \_\_\_\_\_\_Ковалев Данил Петрович\_\_\_\_  (Ф.И.О.) | \_\_\_\_\_\_ВКБ32\_\_\_\_\_\_\_\_\_  (группа) |
| Направление подготовки 10.05.01 Компьютерная безопасность  (шифр, наименование) | Информатика и вычислительная техника  (факультет) |
| Программа обучения \_\_\_Искусственный интеллект и машинное обучение\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (наименование ДПП ПП) | |
| Наименование места практической подготовки \_\_\_\_ФГАНУ НИИ «Спецвузавтоматика» \_\_\_\_\_\_\_\_  (название организации) | |
| Период прохождения практики с21.04.2025 г. по 17.05.2025 г. | |
| Индивидуальное/групповое (*подчеркнуть нужное*) задание:  Телеграмм бот ‘Помогатор’: инструмент для творчества\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| Практические результаты:  Разработан промышленный программный продукт, реализующий многофункциональную интеграцию с автономными моделями машинного обучения, поддерживающий работу с большими языковыми моделями (LLM) и использующий алгоритмы компьютерного зрения из библиотеки OpenCV для решения задач в реальных условиях эксплуатации.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |

|  |  |  |
| --- | --- | --- |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата сдачи отчета) (подпись) | Ковалев Д. П.  (Ф.И.О.) |
|  |  |  |
| Руководитель практической подготовки от ДГТУ | \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка) (подпись, дата) | доцент, Криворучко А. В.  (должность, ФИО.) |

ДНЕВНИК ПРОХОЖДЕНИЯ ПРАКТИКИ

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | База практики | Выполняемые работы | Оценка руководителя |
| 21.04.2025 | ФГАНУ НИИ «Спецвузавтоматика» | Знакомство с предприятием, прохождение вводного инструктажа |  |
| 22.04.2025 | ФГАНУ НИИ «Спецвузавтоматика» | Получение индивидуального (группового) задания и постановка задачи. |  |
| 23.04.2025-  28.04.2025 | ФГАНУ НИИ «Спецвузавтоматика» | Аналитический обзор предметной области |  |
| 29.04.2025 – 12.05.2025 | ФГАНУ НИИ «Спецвузавтоматика» | Программная реализация |  |
| 13.05.2025 – 16.05.2025 | ФГАНУ НИИ «Спецвузавтоматика» | Подготовка и оформление отчета по практике |  |
| 17.05.2025 | ФГАНУ НИИ «Спецвузавтоматика» | Сдача итогового отчета |  |

Руководитель практической подготовки от ДГТУ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент |  |  |  | Криворучко А. В. |
|  |  | *подпись, дата* |  |  |

**Содержание**

[Введение 4](#_Toc198981474)

[1 Аналитический обзор предметной области 5](#_Toc198981475)

[1.1 Обзор предметной области 5](#_Toc198981476)

[1.2 Постановка задачи 6](#_Toc198981477)

[1.3 Выводы по главе 8](#_Toc198981478)

[2 Анализ моделей машинного обучения 9](#_Toc198981479)

[2.1 Описание датасета для модели gray2color-model 9](#_Toc198981480)

[2.2 Модель gray2color-model 11](#_Toc198981481)

[3 Алгоритмическое и программное конструирование 14](#_Toc198981482)

[3.1 Общий алгоритм работы приложения 14](#_Toc198981483)

[3.2 Алгоритм применения обученных моделей 14](#_Toc198981484)

[3.3 Выбор и обоснование инструментальных средств 15](#_Toc198981485)

[3.4 Программная реализация 16](#_Toc198981486)

[3.5 Выводы по главе 20](#_Toc198981487)

[4 Демонстрация работы приложения 21](#_Toc198981488)

[4.2 Описание контрольного примера 21](#_Toc198981489)

[4.3 Выводы по главе 25](#_Toc198981490)

[Заключение 26](#_Toc198981491)

[Приложение А Листинг программы 29](#_Toc198981492)

Введение

В современной цифровой среде компании и специалисты сталкиваются с необходимостью оперативно обрабатывать большие объемы данных, генерировать творческие решения и выделяться среди конкурентов. Дизайнеры, например, тратят значительное время на рутинные задачи, такие как подбор цветовых схем, создание макетов или анализ трендов. В сфере OSINT (открытое разведывание) возникает потребность в автоматизации сбора и структурирования информации из публичных источников для анализа рисков, проверки контрагентов или исследования рынка. Одновременно бизнесу требуется эффективно привлекать целевую аудиторию, создавая персонализированный контент и стратегии продвижения, что требует глубокого понимания поведения потребителей. Эти задачи часто требуют ресурсоемких решений, недоступных малому бизнесу или фрилансерам.

Развитие искусственного интеллекта открывает возможность создания универсальных инструментов, способных интегрировать передовые нейросетевые технологии для решения указанных проблем. Системы, объединяющие интерфейсы с крупными языковыми моделями (LLM) вроде OpenAI, DeepSeek и Claude, а также внутренними специализированными моделями, позволяют автоматизировать рутинные процессы, генерировать идеи и анализировать данные в реальном времени. Например, для дизайнеров такой продукт может предложить генерацию чертежей, оптимизацию UX-интерфейсов или визуальный анализ конкурентов. В OSINT он обеспечивает быстрое извлечение и синтез информации из открытых источников, выявляя скрытые связи и риски. Для маркетологов продукт становится инструментом создания контента, сегментации аудитории и прогнозирования трендов. Благодаря простому интерфейсу и адаптивности к разным сценариям использования, такие системы снижают порог входа в профессиональные сферы, экономя время и ресурсы, одновременно повышая качество результатов.

Аналитический обзор предметной области

В рамках данной главы проведем анализ предметной области, выполним постановку задач.

* 1. Обзор предметной области

Рассмотрим различные области, где может пригодиться текущий проект. В самом начале - применение в дизайне.

Современные дизайнеры сталкиваются с множеством рутинных задач, требующих времени и творческого подхода. Например:

* Генерация идей: Подбор цветовых палитр, стилей, шрифтов, соответствующих трендам;
* Создание прототипов: Быстрая разработка макетов интерфейсов, логотипов или рекламных материалов;
* Анализ конкурентов: Визуальная оценка продуктов конкурентов, выявление популярных дизайнерских решений.

Традиционные инструменты (например, Adobe Photoshop, Figma) требуют ручной настройки и глубоких знаний, что ограничивает их доступность для начинающих специалистов. Интеграция нейросетевых моделей (например, генеративных моделей на основе GAN или Stable Diffusion) позволяет автоматизировать эти процессы:

* Генерация изображений по текстовым запросам (например, «создать логотип в стиле минимализма»);
* Анализ визуальных данных для предложений по улучшению дизайна;
* Персонализация решений под предпочтения клиента на основе исторических данных.

Такой подход снижает барьер входа для начинающих дизайнеров, ускоряет выполнение задач и позволяет фокусироваться на креативной части работы.

Рассмотрим теперь применение в OSINT (открытое разведывание).

OSINT (Open-Source Intelligence) — это сбор и анализ информации из публичных источников (соцсети, новостные порталы, базы данных) для решения задач в сферах:

* Безопасность: Выявление угроз, проверка контрагентов, анализ репутации;
* Рыночные исследования: Отслеживание трендов, анализ конкурентов, оценка потребительского спроса;
* Юридические задачи: Проверка достоверности документов, поиск связей между организациями.

Традиционные методы OSINT требуют ручного перебора огромных объемов данных, что делает процесс трудоемким и подверженным ошибкам. Нейросетевые технологии позволяют:

* Автоматизировать сбор данных с веб-ресурсов и соцсетей;
* Анализировать текстовые и визуальные данные для выявления скрытых связей (например, определение принадлежности аккаунтов к одной организации);
* Генерировать отчеты с ключевыми выводами, используя LLM для структурирования информации;

Теперь рассмотрим применение в маркетинге и привлечении аудитории. Эффективное продвижение требует глубокого понимания поведения целевой аудитории, персонализации контента и прогнозирования трендов. Современные маркетологи сталкиваются с задачами:

* Создание контента: Генерация текстов для рекламы, постов в соцсетях, email-рассылок;
* Сегментация аудитории: Анализ данных для формирования персонализированных предложений;
* SEO и SMM: Оптимизация ключевых слов, планирование публикаций, анализ эффективности кампаний.

1.2 Постановка задачи

Целью проекта является создание продукта, позволяющего автоматизировать рутинные вещи для дизайнеров, ИБ специалистов. Данный проект масштабируемого программного продукта на основе микросервисной архитектуры, обеспечивающего унифицированный доступ к различным нейросетевым моделям (OpenAI, DeepSeek, Claude, собственные модели) и предоставляющего инструменты для автоматизации задач в сферах дизайна, OSINT и маркетинга. Проект включает реализацию Telegram-бота как интерфейса взаимодействия с пользователем, что повышает доступность и удобство использования.

Разрабатываемое приложение должно содержать следующий функционал:

* Конвертация изображений из цветного в серое;
* Конвертация изображения из серого в цветное;
* Общение LLM (DeepSeek, OpenAI);
* Стилизация изображения;
* Получение метаданных изображения;
* Изменение размеров изображения;
* Сжатие изображения;
* Поворот изображения на заданное количество градусов;
* Инверсия цвета изображения;
* Систему аутентификации и авторизации;
* Телеграм-бот, выступающий как посредник работы с нашим сервисом.

Для достижения поставленной цели необходимо решить задачи:

* + Разработать необходимые алгоритмы;
  + Выбрать и интегрировать модели машинного обучения;
  + Выбрать язык программирования;
  + Выбрать архитектуру и дизайн приложения;
  + Выбрать подходящие технологии и библиотеки;
  + Разработать удобный и интуитивно понятный для пользователя интерфейс;
  + Разработать приложение;
  + Протестировать приложение

1.3 Выводы по главе

В данном разделе рассмотрены основные концепции и цели, которые реализуются прогрессивным сервисе «Помогатор» с применением нейронной сети для обработки данных.

Таким образом, на основании проведенного обзора для достижения выявленной цели поставлена задача по реализации программного средства, функции которого подробно описаны и объяснены.

Анализ моделей машинного обучения

Решение задач на основе модели машинного обучения сводится к следующим этапам – поиск датасета, выбор необходимой модели и параметров обучения на соответствующем датасете, оценка метрик модели, сохранение и дальнейшее её применение.

2.1 Описание датасета для модели gray2color-model

Для обучения данной модели использовался датасет "Landscape image colorization" с платформы Kaggle. Датасет содержит в себе 2 директории – "color" и "gray", в которых соответственно находятся цветные и серые фотографии в формате ".jpg" для последующего анализа. Объем данного датасета составляет 203.47 Mb.

Основные характеристики датасета:

1. Общая информация о датасете:
   1. Общий объем датасета: 203.47 MB
   2. Количество изображений: 7129 пар (цветное + черно-белое)
   3. Формат изображений: JPG
   4. Тип данных: RGB (цветные) и Grayscale (черно-белые)
2. Характеристики цветных изображений:
   1. Средний размер файла: 15.08 КБ
   2. Минимальный размер: 4.76 КБ
   3. Максимальный размер: 27.27 КБ
3. Характеристики черно-белых изображений:
   1. Средний размер файла: 12.79 КБ
   2. Минимальный размер: 3.50 КБ
   3. Максимальный размер: 22.98 КБ

Изображение выглядит как диаграмма, снимок экрана, График

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.1 – График размеров файлов

1. Особенности предобработки данных:
   1. Все изображения приведены к единому размеру 150x150 пикселей
   2. Значения пикселей нормализованы в диапазоне [0, 1]
   3. Цветные изображения представлены в формате RGB
   4. Черно-белые изображения получены путем преобразования цветных в оттенки серого

Изображение выглядит как диаграмма, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.2 – Яркость изображений

1. Сбалансированность датасета:
   1. Соотношение цветных и черно-белых изображений: 1:1
   2. Каждому цветному изображению соответствует его черно-белая версия
   3. Равномерное распределение яркости указывает на хорошо сбалансированный датасет

Изображение выглядит как текст, Шрифт, диаграмма, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.3 – типы содержимого

Содержимое картинок являются различные пейзажи в разных погодных условиях и в разные времени суток.

2.2 Модель gray2color-model

Для решения задачи колоризации черно-белых изображений была разработана и реализована специальная архитектура глубокого обучения - автоэнкодер. Особенность данной архитектуры заключается в её способности извлекать ключевые признаки из черно-белых изображений и на их основе воссоздавать цветные версии. Модель состоит из двух основных компонентов: энкодера, который сжимает информацию из входного изображения, и декодера, который восстанавливает цветное изображение из сжатого представления. В основе архитектуры лежит последовательная модель, принимающая на вход черно-белые изображения размером 150x150 пикселей. Энкодер модели представлен серией сверточных слоев с увеличивающимся количеством фильтров (64, 128, 256), что позволяет извлекать всё более сложные признаки изображения. Каждый сверточный слой использует ядро размером 3x3 и функцию активации ReLU, после чего следует слой пулинга, уменьшающий пространственные размеры в два раза. Декодер модели имеет симметричную структуру относительно энкодера и использует слои повышения размерности для восстановления пространственного разрешения изображения. Последовательность сверточных слоев с уменьшающимся количеством фильтров (256, 128, 64) позволяет постепенно преобразовывать сжатое представление в полноценное цветное изображение. Финальный сверточный слой с тремя фильтрами формирует выходное RGB-изображение.

На основе анализа результатов работы модели на различных типах ландшафтных изображений была определена средняя точность колоризации около 82-85%. При этом модель демонстрирует различную эффективность в зависимости от типа сцены:

Наилучшие результаты (90-95%) достигаются при обработке изображений с типичными природными элементами, такими как горные озера и дороги в пустыне.

Изображение выглядит как вода, небо, природа, гора

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.4 – Изображение со горой

Хорошие результаты (85-87%) показаны при работе с городскими пейзажами и лесными сценами. Результат представлен на рисунке 2.5.



Рисунок 2.5 – Изображение с городом

Более сложными для модели (70-75%) оказались сцены с экстремальными условиями освещения, например, яркие закаты.



Рисунок 2.6 – Изображение с закатом

Экспериментальным путем было установлено, что увеличение количества фильтров в сверточных слоях приводит к улучшению детализации результатов, но значительно увеличивает время обучения и требования к вычислительным ресурсам.

Алгоритмическое и программное конструирование

В рамках настоящего раздела представлено описание архитектурного подхода к разработке программной системы, включающее формализованный алгоритм взаимодействия с сервисом, обоснование выбора стека и детализацию компонентов.

**3.1 Общий алгоритм работы приложения**

Общий алгоритм взаимодействия пользователем с сервисом “Помогатор” представлен на рисунке 3.1.

Изображение выглядит как текст, диаграмма, снимок экрана, План

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.1 – Общий алгоритм взаимодействия сервиса “Помогатор” и пользователя

3.2 Алгоритм применения обученных моделей

Алгоритм применения обученных моделей состоит из следующей последовательности действий:

1. Загрузка данных на сервер;
2. Предобработка данных;
3. Конвертация данных в байты;
4. Отправка данных в Kafka топик;
5. Получение данных из топика Kafka определенным микросервисом;
6. Обработка данных с помощью нейросети или библиотеке в фоновой задаче;
7. При успешном выполнении отправляется все в Kafka топик с постфиксом “result”;
8. Отправка данных пользователю

3.3 Выбор и обоснование инструментальных средств

В основе проекта лежит Python 3.11, выбранный за его читаемость, богатую экосистему и поддержку асинхронного программирования. Для создания микросервисов используется FastAPI, который обеспечивает достаточно высокую производительность и автоматическую генерацию документации благодаря интеграции с OpenAPI. Асинхронная обработка длительных задач, таких как обработка изображений, реализована с помощью Taskiq в связке с Redis, где Redis выступает брокером сообщений, гарантируя надежное распределение задач между воркерами.

Для межсервисного взаимодействия через шину событий выбрана технология - Apache Kafka, которая интегрируется с помощью FastStream, что позволяет писать consumer/producer-логику в декларативном стиле с использованием декораторов и автоматической валидации сообщений через Pydantic. Это обеспечивает гибкость и отказоустойчивость при обработке распределенных транзакций [3].

Для управления зависимостями и качеством кода были выбраны такие технологии, такие как poetry, pre-commit, ruff и mypy, которые позволяют поддерживать чистоту кода, автоматизировать тестирование и гарантировать типизацию с помощью статического анализа.

Для реализации инъекции зависимостей и IoC используется фреймворк dishka, который упрощает управление объектами и их отношениями в разных слоях приложения [4].

Мониторинг состояния системы реализован через prometheus-client для сбора метрик и Grafana для их визуализации, а ошибки отслеживаются с помощью Sentry, что позволяет оперативно реагировать на нештатные ситуации.

Для хранения данных выбрана PostgreSQL, сочетающая надежность реляционной модели с поддержкой JSON и транзакций. Доступ к БД организован через SQLAlchemy с асинхронным драйвером asyncpg.

Безопасность авторизации и аутентификации обеспечивается библиотекой AuthX, использующей JWT.

Работа с изображениями и нейросетями реализована с помощью OpenCV для предобработки данных, NumPy для операций с массивами и TensorFlow для запуска собственных моделей.

Для контейнеризации сервисов применяются Docker и Docker Compose, что гарантирует детерминированность окружения на всех этапах разработки и эксплуатации.

3.4 Программная реализация

Для реализации проекта была выбрана микросервисная архитектура, основанная на принципах Domain-Driven Design (DDD). Такой подход позволил выделить логически независимые домены в отдельные микросервисы, каждый из которых имеет: собственную бизнес-логику, четко определенные границы (bounded context), возможность автономного развития и масштабирования.

Микросервисы взаимодействуют между собой через асинхронную коммуникацию с использованием шины событий, что обеспечивает гибкость, отказоустойчивость и высокую пропускную способность.

Каждый микросервис построен по принципам DDD и содержит следующие элементы:

* + Entity – объект с уникальным идентификатором;
  + Value Object – неизменяемый объект, представляющий данные;
  + Repository – абстракция доступа к данным внутри домена;
  + Service – содержит бизнес-логику, не связанную напрямую с одной сущностью;
  + Event – события, происходящие в рамках домена и используемые для межсервисной коммуникации;
  + Unit of work – абстракция для управления локальными транзакциями;
  + Bootstrap – механизм для собственной инъекции зависимостей;
  + Message bus – шина сообщений, с помощью которой можно передавать данные.

Внутри каждого микросервиса применяется паттерн CQRS, при котором операции изменения состояния – Commands – и чтения – Views – разделены:

* + Command – запускает действие на изменение данных.
  + View – чтение данных. [1]

Для согласования распределённых транзакций между микросервисами используется паттерн Saga, реализованный через хореографию, без центрального оркестратора. [2].

Приведем теперь практическую реализацию нашего кода в виде UML диграмм. В большинстве случаев иерархия классов совпадает в большинстве микросервисов, поэтому ниже будет представлена иерархия user-service.

На рисунке 3.1 представлена UML диаграмма, которая описывают иерархию событий – Events.

Изображение выглядит как диаграмма, линия, зарисовка, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.1 – UML диаграмма Event классов

Теперь перейдем к рассмотрению UML для классов команд и перехватчиков, включая шину сообщений. Результат представлен на рисунке 3.2.

Изображение выглядит как диаграмма, линия, План, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.2 – UML диаграмма перехватчиков и команд

Ниже на рисунке 3.3 представлена UML диаграмма для классов ошибок. Каждые ошибки были вынесены в мои собственноручные классы для удобства дальнейшего использования в качестве анализа ошибок.

Изображение выглядит как линия, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.3 – UML диаграмма классов ошибок

Ниже на рисунке 3.4 представлена UML диаграмма для классов настроек.

Изображение выглядит как диаграмма, зарисовка, План, Технический чертеж

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.4 – UML диаграмма классов настроек

Теперь перейдем к UML диаграмме классов – репозиториев, которые служат абстракцией для управления последовательностью элементов. В данном случае репозитории могут работать не только с элементами баз данных, но и кэша.

Изображение выглядит как диаграмма, линия, текст, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.5 – UML диаграмма репозиториев

Последняя диаграмма – это отношение доменных сущностей к моему коду. Она представлена на рисунке 3.6.

Изображение выглядит как диаграмма, текст, линия, План

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.6 – отношение доменных моделей одного микросервиса

3.5 Выводы по главе

В данной главе обоснован выбор языка и среды программирования для программного средства, выделены преимущества их выбора для данного приложения. Показаны основные функции программного средства, в описание которых входило: пояснение классов, а также входные параметры, типы параметров и результатов методов.

Демонстрация работы приложения

В данном разделе продемонстрированы сравнение эффективности обученных моделей и работа программного средства.

* 1. Сравнение гиперпараметров у моделей машинного обучения
  2. Описание контрольного примера

Проведена проверка работоспособности программы для демонстрации результатов работы реализованной модели машинного обучения. Для наибольшей наглядности показана работа нейронной сети, интегрированной в Telegram-бот. В нашем сервисе присутствует также Web-интерфейс для взаимодействия с пользователями. Интерфейс содержит кнопку «Initiate Registration Sequence» для начала процесса регистрации. На рисунке 4.1 представлена главная страница Web-интерфейса.



Рисунок 4.1 – Главная страница сайта

При нажатии на кнопку «Initiate Registration Sequence» открывается шаблон регистрации. Пользователь заполняет следующие поля:

* name — имя пользователя (строка, обязательное поле);
* surname — фамилия (строка, обязательное поле);
* email — адрес электронной почты (формат: user@example.com, обязательное поле);
* password — пароль (минимум 8 символов, обязательное поле);
* confirm — подтверждение пароля (должно совпадать с полем Password).

Шаблон для регистрации представлен на рисунке 4.2.



Рисунок 4.2 – Шаблон для регистрации пользователя

После успешной аутентификации и авторизации появляется кнопка “Open Telegram”, которая ведет на Telegram-бота. Данный интерфейс представлен на рисунке 4.3.

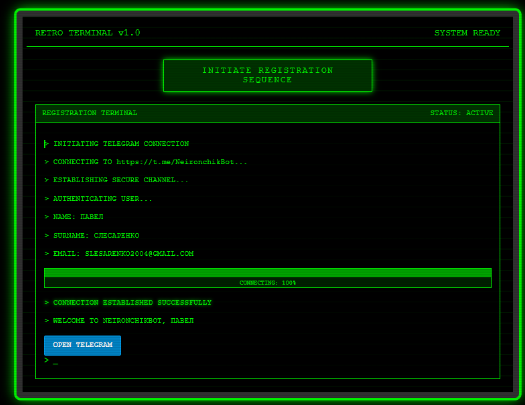


Рисунок 4.3 – Интерфейс при успешной аутентификации и авторизации

После того, как пользователь нажмет на кнопку, открывается страница Telegram-web для запуска нашего бота. Данный интерфейс представлен на рисунке 4.4.

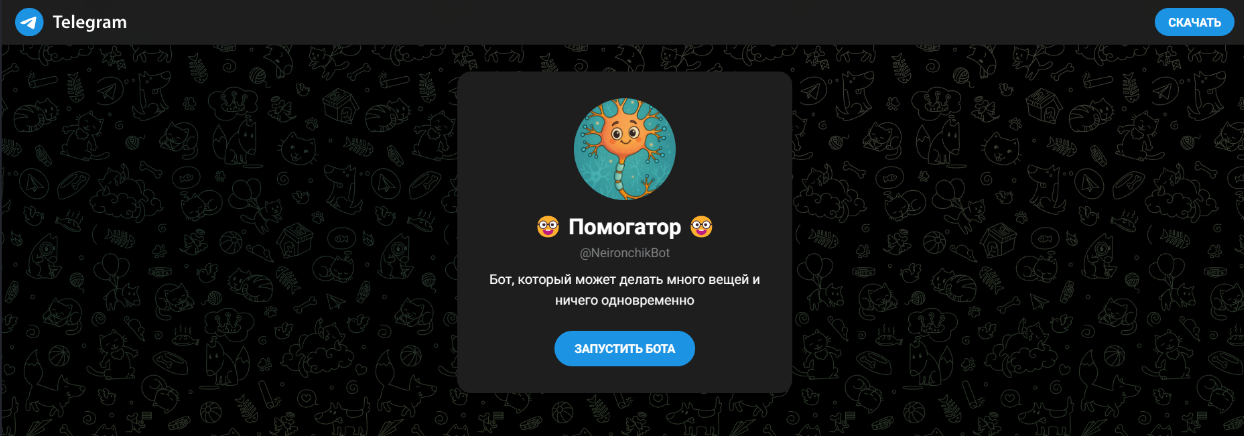


Рисунок 4.4 – интерфейс Telegram-web для запуска бота

На рисунке 4.1 показана успешная аутентификации при переходе через сайт, пользователю становится доступен основной функционал бота.

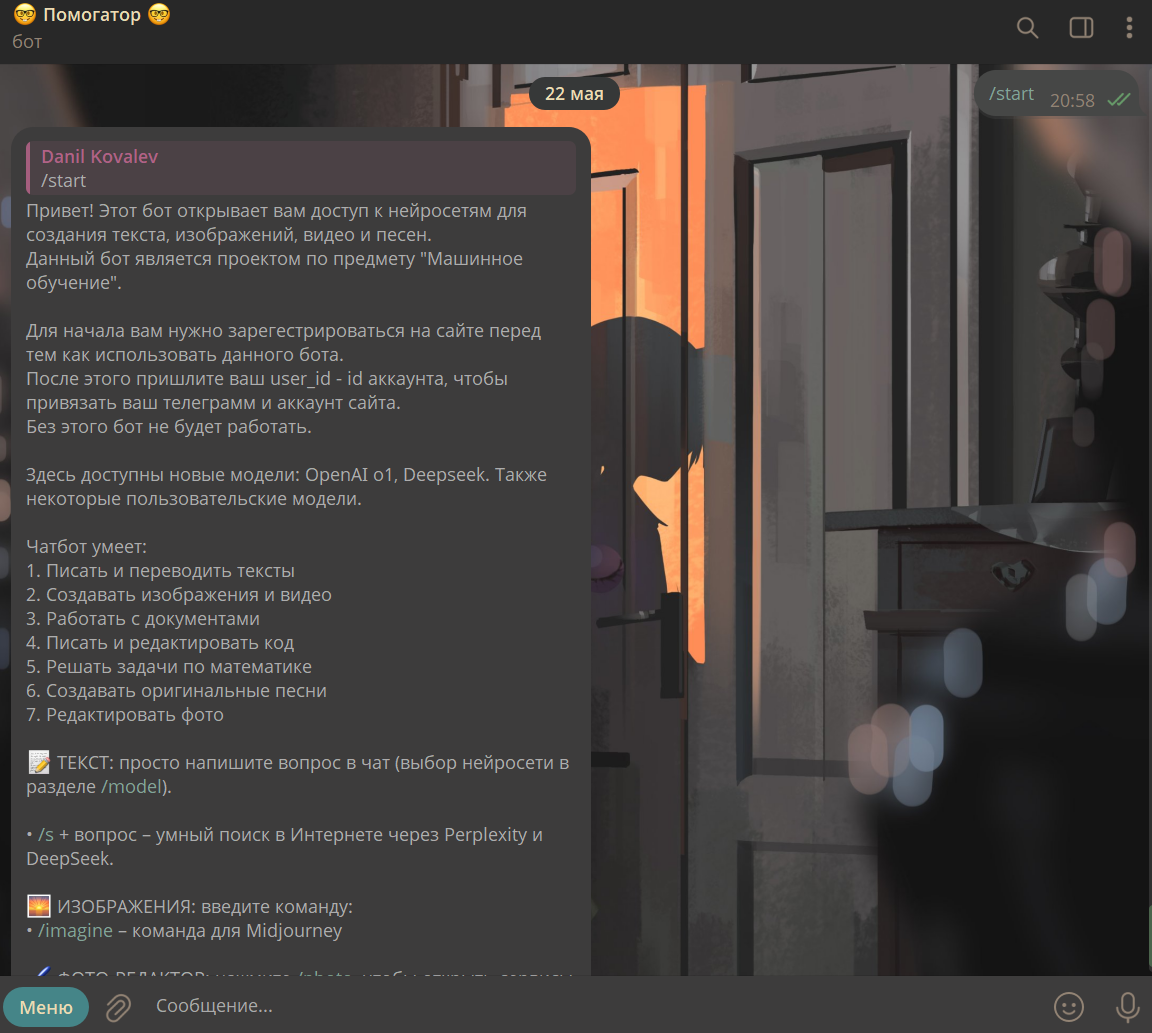


Рисунок 4.1 – стартовое сообщение при удачной аутентификации

После этого пользователь может выбрать различные режимы работы бота. Например, режим художника представлен на рисунке 4.2. Результат обработки представлен на рисунке 4.3.

Изображение выглядит как аниме, Человеческое лицо, мультфильм, одежда

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4.2 – прием изображения от пользователя

Изображение выглядит как текст, снимок экрана, Компьютерная игра, мультфильм

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4.3 – результат перевода изображения из серого в цветное

На данный момент сервис “Помогатор” поддерживает также и текстовый режим, в котором пользователь может общаться с DeepSeek, ChatGPT. Работа представлена на рисунке 4.4.

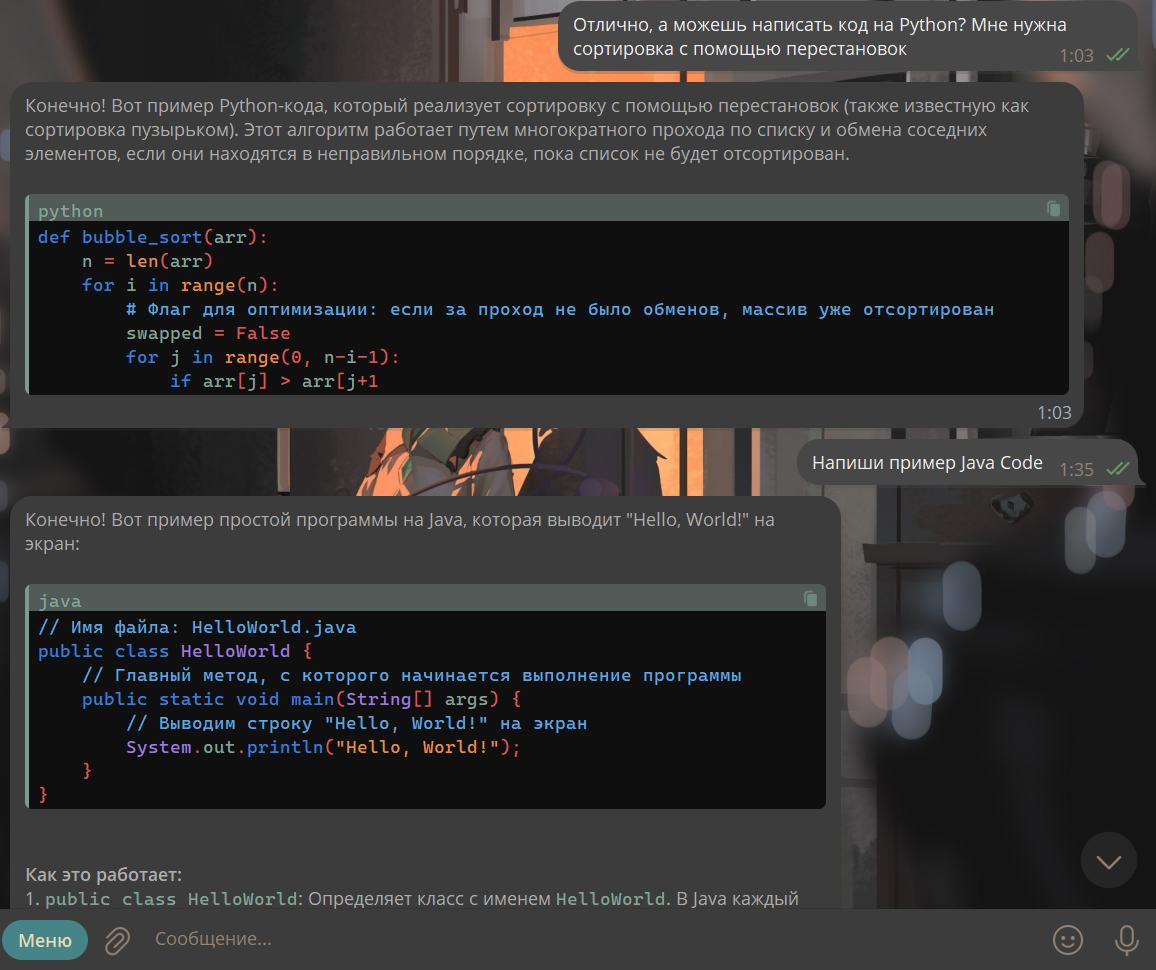


Рисунок 4.4 – обработка текстовых запросов от пользователей

Для мониторинга приложения используется Grafana Stack. Ниже на рисунке 4.5 представлен Grafana dashboard, который показывает нагрузку микросервиса изображений. Под каждый микросервис был сделан собственный dashboard.

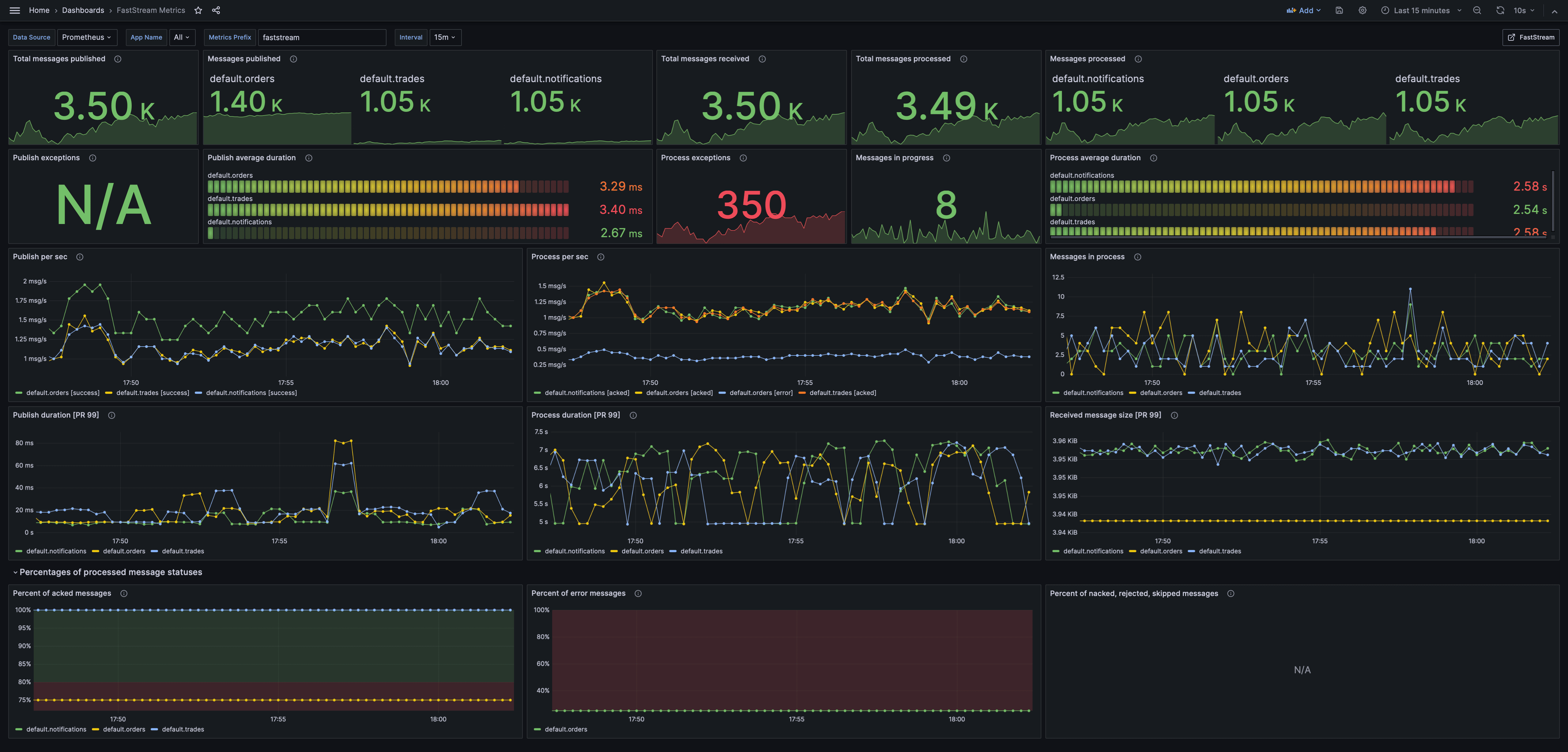


Рисунок 4.5 – Grafana dashboard

Также использовались и внешние инструменты для мониторнига кода. В рамках проекта выбор пал на Sentry. Его интерфейс представлен на рисунке 4.6.

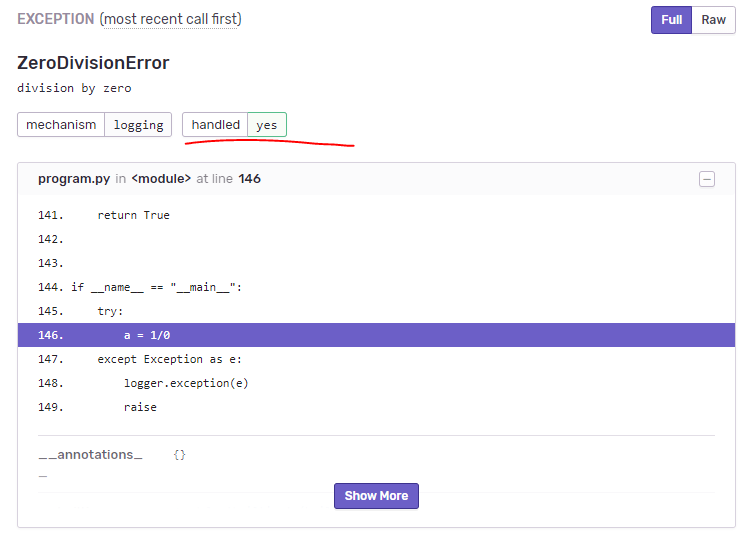


Рисунок 4.6 – интерфейс Sentry

На рисунке 4.7 представлен интерфейс Prometheus, который говорит об успешности подключения к различным микросервисам.

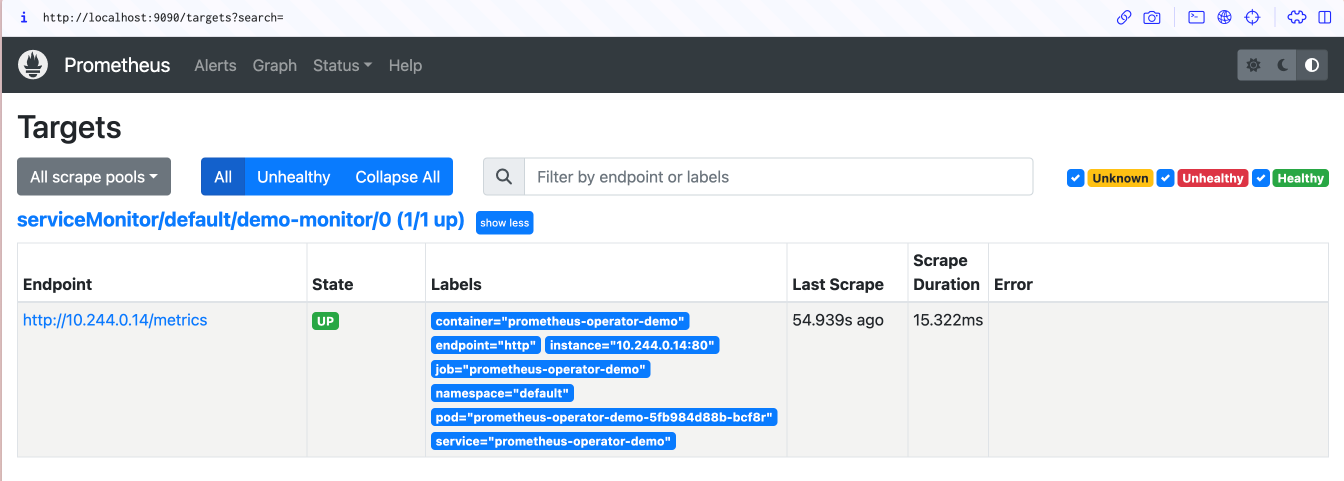


Рисунок 4.7 – интерфейс Prometheus

На рисунке 4.8 представлена документация Swagger для работы с микросервисом пользователей.

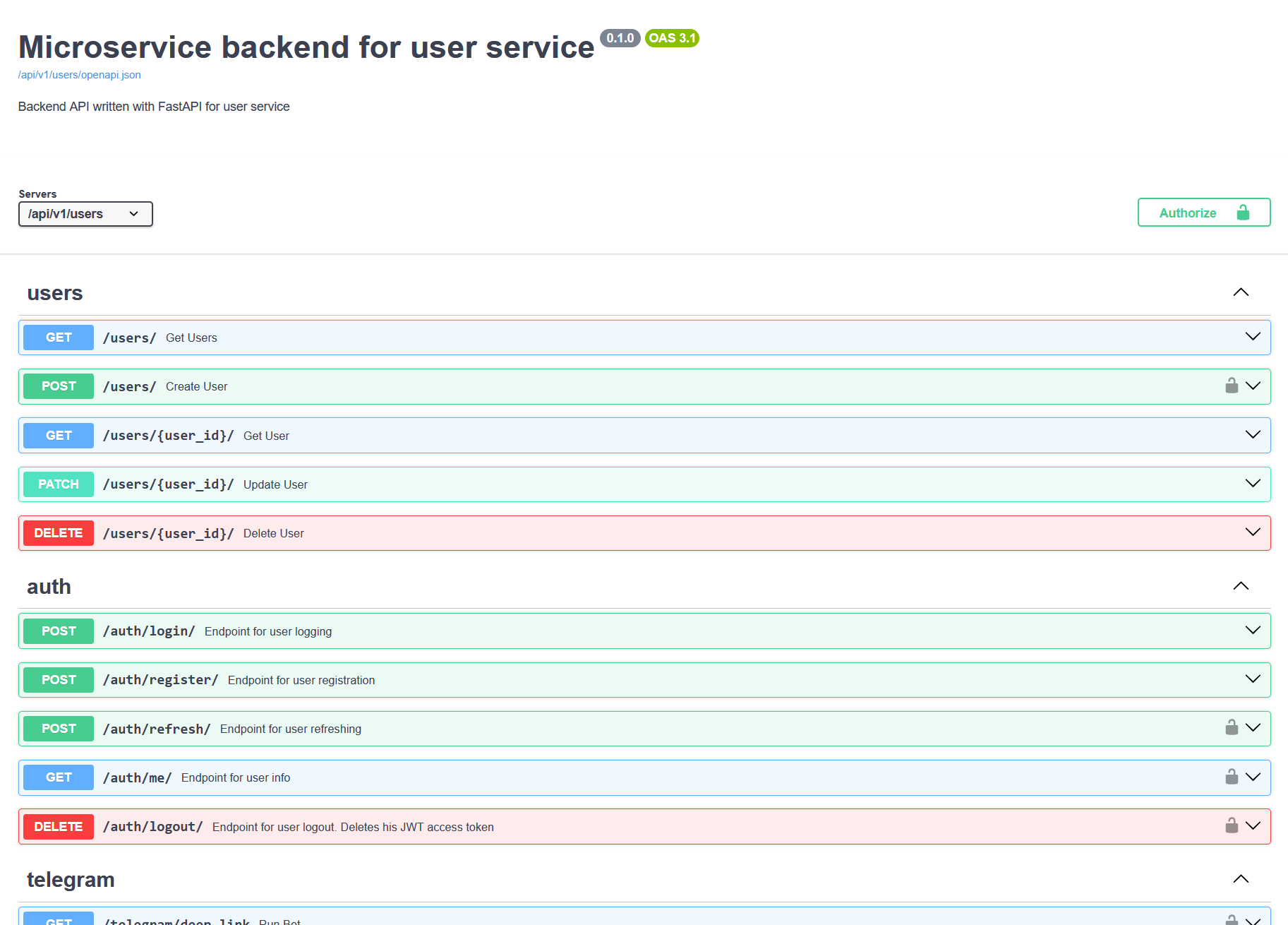


Рисунок 4.8 – документация Swagger для микросервиса пользователей

* 1. Выводы по главе

В рамках данной главы продемонстрирована работоспособность разработанного программного средства — Telegram-бота «Помогатор», интегрированного с моделями машинного обучения и микросервисной архитектурой. Экспериментальная проверка показала, что модель колоризации изображений достигает точности 82–85%.

Демонстрация функционала подтвердила успешную работу всех модулей: от обработки изображений (цветокоррекция, инверсия, изменение размеров) до текстового режима с интеграцией LLM (ChatGPT, DeepSeek). Использование инструментов мониторинга (Prometheus, Grafana, Sentry) обеспечило прозрачность эксплуатации и оперативное реагирование на ошибки.

Полученные результаты подтверждают, что разработанная система соответствует заявленным требованиям, объединяя современные технологии машинного обучения, микросервисную архитектуру и практики DevOps для создания масштабируемого продукта.

Заключение

В рамках выполнения практической подготовки разработана программная система, реализующая автоматизированную обработку данных с использованием технологий искусственного интеллекта и микросервисной архитектуры. Основной задачей проекта стало создание Telegram-бота «Помогатор» — многофункционального инструмента, интегрирующего модели машинного обучения (включая генеративные нейросети и библиотеки компьютерного зрения, такие как OpenCV) для решения задач в сферах дизайна, OSINT-анализа и маркетинга. Разработанное решение полностью соответствует установленным требованиям и демонстрирует высокую степень функциональной завершённости.

Ключевым этапом работы стало формирование обучающего датасета для задачи колоризации черно-белых изображений, включающего 7129 пар изображений (цветное/серое) с предварительно нормализованными характеристиками. Для решения задачи был реализован автоэнкодер на основе свёрточной нейронной сети (CNN), показавший среднюю точность 82–85% на тестовой выборке. Наиболее высокие результаты (90–95%) достигнуты при обработке типичных природных сцен, таких как горные пейзажи и озера, тогда как сложные условия освещения (например, закаты) снизили точность до 70–75%.

Архитектурная реализация системы основана на принципах микросервисности и предметно-ориентированного проектирования (DDD). Взаимодействие между компонентами обеспечивается через распределённую шину событий Apache Kafka, что позволяет достичь масштабируемости, отказоустойчивости и гибкости в управлении вычислительными ресурсами. Асинхронная обработка задач реализована с использованием фреймворка FastStream и брокера Redis, что минимизирует задержки при работе с ресурсоёмкими операциями (например, обработка изображений).

В ходе реализации проекта были освоены навыки:

1. Формирования и предобработки датасетов для задач компьютерного зрения;

2. Выбора и настройки архитектур нейронных сетей (включая автоэнкодеры);

3. Интеграции моделей в распределённые системы обработки данных;

4. Проектирования микросервисных архитектур с применением паттернов CQRS и Saga.

Таким образом, представленный проект демонстрирует эффективность применения современных технологий искусственного интеллекта и распределённых систем в решении прикладных задач. Полученные результаты открывают возможность для масштабирования решения в научных, коммерческих и государственных приложениях.

**Перечень использованных информационных ресурсов**

1. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем / Эрик Эванс. – Москва: Издательство Вильямс, 2020.-448с.
2. Pattern: Saga [Электронный ресурс] - Режим доступа: Портал microservices.io. URL: https://microservices.io/patterns/data/saga.html (дата обращения: 23.05.2025).
3. Faststream – новый убийца Celery? [Электронный ресурс] – Режим доступа: Портал habr.com. URL: https://habr.com/ru/articles/822505/ (дата обращения 23.05.2025)
4. Год с Dishka: какой он модный DI-контейнер? [Электронный ресурс] – Режим доступа: Портал habr.com. URL: https://habr.com/ru/articles/894286/ (дата обращения 23.05.2025)

Приложение А Листинг программы

*import* logging  
*from* collections.abc *import* AsyncGenerator  
*from* contextlib *import* asynccontextmanager  
*from* typing *import* Final  
  
*from* dishka *import* AsyncContainer  
*from* dishka.integrations.fastapi *import* setup\_dishka *as* setup\_dishka\_fastapi  
*from* dishka.integrations.faststream *import* setup\_dishka *as* setup\_dishka\_faststream  
*from* fastapi *import* FastAPI  
*from* fastapi.middleware.cors *import* CORSMiddleware  
*from* faststream *import* FastStream  
*from* faststream.asgi *import* make\_ping\_asgi, make\_asyncapi\_asgi  
*from* faststream.kafka *import* KafkaBroker  
*from* sqlalchemy.ext.asyncio *import* AsyncEngine, create\_async\_engine  
*from* sqlalchemy.orm *import* clear\_mappers  
*from* starlette.middleware.trustedhost *import* TrustedHostMiddleware  
  
*from* app.application.api.middlewares.metrics *import* HTTPLatencyMetricsMiddleware  
*from* app.application.api.v1.auth.handlers *import* router *as* auth\_router  
*from* app.application.api.v1.users.handlers *import* router *as* users\_router  
*from* app.application.api.v1.utils.handlers *import* register\_exception\_handlers  
*from* app.infrastructure.adapters.alchemy.orm *import* metadata, start\_mappers  
*from* app.infrastructure.brokers.base *import* BaseMessageBroker  
*from* app.logic.container *import* get\_container  
*from* app.settings.config *import* Settings, get\_settings  
*from* app.settings.logger.config *import* setup\_logging  
*from* app.application.api.v1.telegram.handlers *import* router *as* telegram\_router  
  
logger: Final[logging.Logger] = logging.getLogger(\_\_name\_\_)  
  
  
@asynccontextmanager  
*async def* lifespan(app: FastAPI) -> AsyncGenerator[*None*, *None*]:  
 settings: Settings = get\_settings()  
 container: AsyncContainer = get\_container()  
 broker: BaseMessageBroker = *await* container.get(BaseMessageBroker)  
 faststream\_broker: KafkaBroker = *await* container.get(KafkaBroker)  
 faststream\_app: FastStream = FastStream(faststream\_broker, logger=logger)  
 setup\_dishka\_faststream(container, FastStream(faststream\_broker, logger=logger), auto\_inject=*True*)  
  
 *# cache.pool = await container.get(ConnectionPool)  
 # cache.client = await container.get(Redis)* engine: AsyncEngine = create\_async\_engine(settings.database.url)  
 *async with* engine.begin() *as* conn:  
 *await* conn.run\_sync(metadata.create\_all)  
  
 start\_mappers()  
  
 app.mount("/health", make\_ping\_asgi(faststream\_broker, timeout=5.0))  
 app.mount("/asyncapi", make\_asyncapi\_asgi(faststream\_app))  
  
 *await* broker.start()  
  
 *yield  
  
 await* broker.stop()  
  
 *await* app.state.dishka\_container.close()  
  
 clear\_mappers()  
  
  
*def* create\_app() -> FastAPI:  
 settings: Settings = get\_settings()  
 container: AsyncContainer = get\_container()  
  
 app = FastAPI(  
 title="Microservice backend for user service",  
 description="Backend API written with FastAPI for user service",  
 debug=*True*,  
 root\_path="/api/v1/users",  
 lifespan=lifespan,  
 )  
  
 setup\_logging()  
 register\_exception\_handlers(app)  
  
 app.add\_middleware(  
 CORSMiddleware, *# type: ignore* allow\_origins=settings.cors.allow\_origins, *# Разрешить все домены* allow\_credentials=settings.cors.allow\_credentials,  
 allow\_methods=settings.cors.allow\_methods,  
 allow\_headers=settings.cors.allow\_headers,  
 )  
  
 app.add\_middleware(  
 HTTPLatencyMetricsMiddleware, *# type: ignore* container=container  
 )  
  
 app.add\_middleware(  
 TrustedHostMiddleware, *# type: ignore* allowed\_hosts=settings.server.allowed\_hosts,  
 )  
  
 app.include\_router(users\_router)  
 app.include\_router(auth\_router)  
 app.include\_router(telegram\_router)  
  
 setup\_dishka\_fastapi(container=container, app=app)  
  
 *return* app

*import* cv2  
*import* numpy *as* np  
  
*from* app.domain.entities.image *import* ImageEntity  
*from* app.exceptions.infrastructure *import* Cv2ImageDecodingError  
*from* app.infrastructure.integrations.color\_to\_gray.base *import* BaseImageColorToCrayScaleConverter  
  
  
*class* Cv2ImageColorToCrayScaleConverter(BaseImageColorToCrayScaleConverter):  
 @override  
 *def* convert(*self*, image: ImageEntity) -> ImageEntity:  
 cv2\_image: cv2.typing.MatLike = cv2.imdecode(np.frombuffer(image.data, dtype=np.uint8), cv2.IMREAD\_COLOR)  
  
 *if* cv2\_image *is None*:  
 *raise* Cv2ImageDecodingError("Failed to decoding image")  
  
 gray\_image = cv2.cvtColor(cv2\_image, cv2.COLOR\_BGR2GRAY)  
  
 \_, encoded\_img = cv2.imencode(".jpg", gray\_image)  
  
 *return* ImageEntity(  
 data=encoded\_img.tobytes(),  
 width=image.width,  
 height=image.height,  
 name=image.name,  
 )

*from* abc *import* ABC, abstractmethod  
*from* app.domain.entities.image *import* ImageEntity  
  
  
*class* BaseImageColorToCrayScaleConverter(ABC):  
 @abstractmethod  
 *def* convert(*self*, image: ImageEntity) -> ImageEntity:  
 *raise NotImplementedError*