МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ИАОУ ФГАОУ ВО
«ЮУрГУ (НИУ)», к.т.н.

_____ А.А. Шинкарев

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка сервиса для генерации изображений фасадов зданий по выделенной маске

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1499.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ М.В. Сухов

Автор работы,
студент группы КЭ-228
_____ М.А. Сипатов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта

студенту группы КЭ-228
Сипатову Максиму Александровичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка сервиса для генерации изображений фасадов зданий
по выделенной маске.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Radford A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. // 2016 International Conference on Learning Representations, 2016. – 7–15 pp.
3.2. Vayadande K., Bhemde S. AI-Based Image Generator Web Application using OpenAI's DALL-E System. // 2023 International Conference on Recent Advances in Science and Engineering Technology (ICRASET), 2023. – 8–13 pp.
3.3. Korkmaz C., Tekalp M., Dogan Z. Training Generative Image Super-Resolution Models by Wavelet-Domain Losses Enables Better Control of Artifacts. // 2024 Computer Vision and Pattern Recognition Conference, 2024. – 22–30 pp.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.
4.2. Провести обзор научной литературы.
4.3. Выполнить проектирование архитектуры приложения.
4.4. Разработать инфраструктуру для приложения.
4.5. Выбрать модель генеративной нейронной сети.

4.6. Реализовать приложение.

4.7. Протестировать приложение.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.т.н. М.В. Сухов

Задание принял к исполнению М.А. Сипатов

ГЛОССАРИЙ

1. *Docker* – это программная платформа, которая позволяет разработчикам быстро создавать, тестировать и развертывать контейнерные приложения [1].

2. *Docker container* – это формат пакетирования, который позволяет упаковать весь код и зависимости в стандартный формат [2].

3. *Helm–chart* – это пакет Helm. Он содержит описания ресурсов, необходимые для запуска приложения внутри кластера Kubernetes [3].

4. *K8s* – это открытое программное обеспечение для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями [4].

5. *Автокодировщик* – это тип нейросети, который используется для извлечения наиболее значимых признаков из входных данных [5].

6. *Балансировщик нагрузки* – сервис для распределения сетевого трафика и задач между сетевыми устройствами [6].

7. *Пайплайн* – это поток автоматической интеграции и доставки (или развертывания) приложений [7].

8. *Под* – это абстрактный объект Kubernetes, представляющий собой группу из одного или нескольких контейнеров приложения [8].

9. *Трансформер* – архитектура глубоких нейронных сетей, основанная на механизме внимания без использования рекуррентных нейронных сетей [9].

10. *Узел* – виртуальная или физическая машина для запуска подов [10].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном информационном обществе, где визуальный контент играет ключевую роль, создание уникальных и качественных изображений становится все более важной задачей. Разработка сервиса для генерации изображений представляет собой актуальное и перспективное направление в области информационных технологий. С учетом быстрого развития технологий и повышенного спроса на визуальные материалы, необходимость в инструментах, способных автоматизировать и упростить процесс создания графического контента, становится неотъемлемой частью современного программного обеспечения.

Данная работа посвящена разработке сервиса для генерации изображений фасадов зданий по выделенной маске.

Основной акцент делается на создании эффективного и удобного в использовании инструмента, способного удовлетворить потребности широкого круга пользователей, начиная от дизайнеров и архитекторов, и заканчивая обычными пользователями.

Постановка задачи

Целью выпускной квалификационной работы является разработка сервиса для генерации изображений фасадов зданий по выделенной маске. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) провести обзор научной литературы;
3) выполнить проектирование архитектуры приложения;
4) разработать инфраструктуру для приложения;
5) выбрать модель генеративной нейронной сети;
6) реализовать приложение;
7) протестировать приложение.

Структура и содержание работы

Работа состоит из введения, глоссария, пяти глав, заключения и списка литературы. Объем работы составляет 67 страниц, объем списка литературы – 46 источников.

Первая глава посвящена описанию теоретических сведений. В ней приводятся основные теоретические сведения по генеративно-состязательным и диффузионным нейронным сетям.

Во второй главе проводится обзор предметной области. В ней описываются основные архитектуры нейронных сетей в задаче компьютерного зрения.

В третьей главе описывается архитектура приложения. В ней приведены требования к системе, показана основная архитектура приложения и спроектированы диаграмма вариантов использования и диаграмма компонентов.

В четвертой главе описывается реализация приложения. В ней показана разработка инфраструктуры, описана реализация основных модулей приложения.

В пятой главе описывается тестирование приложения. В ней описывается тестирование приложения по функциональным требованиям.

В приложении А приведены листинги, описывающие код, который был написан во время реализации инфраструктуры. В приложении Б приведен код, описывающий реализацию пользовательского интерфейса. В приложении В показан код для реализации сервиса основного api, который используется для работы с изображениями. В приложении Г приведен код для реализации сервиса получения результатов. В приложении Д описан код для реализации сервиса для обработки изображений.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Теоретические сведения по генеративным нейронным сетям

В данной главе описаны основные теоретические сведения о генеративных состязательных нейронных сетях и диффузионных нейронных сетях, чтобы предоставить основные особенности обеих архитектур.

В последние годы генеративные нейронные сети (ГНС) [11] стали ключевым элементом в сфере генерации визуального контента. Эти сети представляют собой важный класс искусственного интеллекта, разработанный для создания новых данных, которые могут быть статистически схожи с обучающим набором. В частности, в области генерации изображений, ГНС демонстрируют впечатляющую способность создавать реалистичные и убедительные визуальные эффекты. На рисунке 1 представлена основная архитектура ГНС.

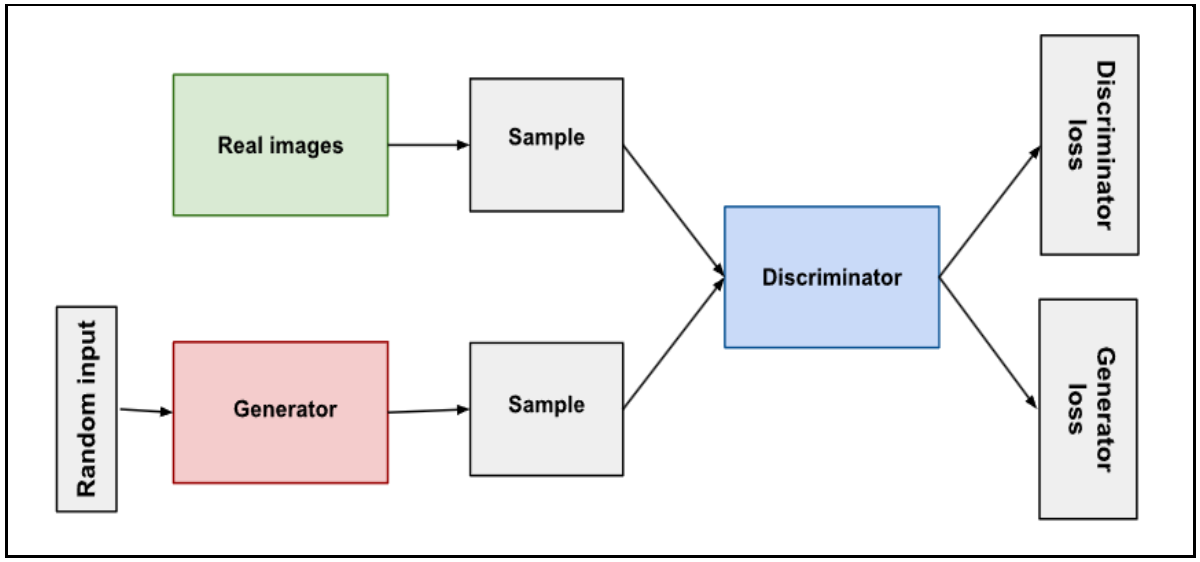

Рисунок 1 – Архитектура ГНС

Генератор является центральным элементом ГНС. Это нейронная сеть, которая принимает на вход случайный шум или другие представления и трансформирует их в изображения. Задача генератора – научиться создавать данные, которые неотличимы от реальных, обучившись структурным и стилистическим характеристикам обучающего набора. Дискриминатор является контролирующим элементом, который оценивает реалистичность

сгенерированных данных. Он обучается различать между реальными и искусственно созданными изображениями. В процессе тренировки, дискриминатор стремится стать все более точным в выявлении различий между реальными и генерированными данными. Функция потерь определяет, насколько сгенерированные изображения соответствуют реальным данным. В процессе обучения, ГНС стремятся минимизировать функцию потерь, что приводит к улучшению качества генерации [12].

Генератор и дискриминатор взаимодействуют в процессе тренировки. Генератор создает изображения, стремясь обмануть дискриминатор, который в свою очередь улучшает свои навыки различения между реальными и сгенерированными данными. Этот итеративный процесс приводит к постоянному улучшению их работы. После завершения тренировки, ГНС могут быть использованы для генерации новых изображений. Путем предоставления на вход случайного шума или других представлений, генератор создает соответствующие изображения.

ГНС способны создавать изображения с новыми стилями и визуальными характеристиками. Также они могут извлекать стили из одних изображений и применять их к другим, обогащая визуальный контент. Такие сети позволяют генерировать изображения, основываясь на текстовых описаниях, что полезно в сферах создания и редактирования контента. Их активно применяют для генерации данных в ситуациях, когда обучающий набор ограничен или неполон.

1.2. Теоретические сведения по диффузионным нейронным сетям

В последние годы диффузионные нейронные сети (ДНС) [13] выделились как мощный инструмент в области генерации изображений. ДНС представляют собой класс моделей, основанных на принципах диффузии, что делает их уникальными по сравнению с традиционными генеративными сетями. В этом контексте, ДНС демонстрируют способность генерировать высококачественные и непревзойденно реалистичные изображения.

Основной принцип ДНС основан на математической модели диффузии, представляющей собой распространение случайного процесса в пространстве. В контексте генерации изображений, диффузионный процесс используется для постепенного распределения интенсивности пикселей от начального состояния к целевому [14]. Диффузионные нейронные сети моделируют диффузионную динамику внутри изображения. Процесс диффузии происходит пошагово, при этом каждый шаг представляет собой изменение интенсивности пикселей на изображении. Уникальной особенностью ДНС является применение градиентного спуска в обратном направлении. Это позволяет модели учиться относительно заданного распределения данных, исходя из последовательного процесса диффузии.

Генеративная модель в ДНС ответственна за процесс диффузии и создание высококачественных изображений. Она учится параметрам диффузионного процесса, таким образом, чтобы пошагово распределять интенсивность пикселей и создавать реалистичные изображения. Параметры диффузии определяют скорость изменения интенсивности пикселей на каждом шаге диффузии. Их обучение позволяет модели адаптироваться к различным структурам данных.

Вывод по первой главе

Для решения задачи генерации изображения подходят и генеративно-состязательные сети, и диффузионные нейронные сети. Когда как ГНС могут предоставить для пользователя нечто новое, ДНС, наоборот, помогают достичь реалистичности изображения на выходе. В этой работе будет применятся ДНС, так как необходимо получать наиболее реалистичное изображение после генерации.

2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1. Обзор научной литературы

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks [15]

В данной статье представлена архитектура сверточной генеративно-состязательной нейронной сети DCGAN (Deep Convolutional Generative Adversarial Network). В работе представлено, что нейронные сети GAN могут использоваться не только для генерации изображений, но и для изучения более низкоуровневых признаков и представлений, что может быть важным для различных задач обработки изображений.

В статье авторы предлагают следующие инновации, которые делают DCGAN более устойчивыми и способными к обучению.

1. Использование сверток без пулинга в генераторе и дискриминаторе.
2. Использование нормализации в обеих сетях.
3. Удаление полносвязных слоев из глубокой сверточной нейронной сети.
4. Использование активации ReLU в скрытых слоях генератора, кроме выходного слоя, который использует активацию tanh.
5. Использование однородного распределения на скрытых переменных генератора.

С помощью этих улучшений DCGAN способен генерировать изображения, которые выглядят похожими на обучающие данные, а также изучать интересные представления обучающих данных. Это открывает путь к использованию GAN для неразмеченного обучения и изучения признаковых представлений в различных задачах компьютерного зрения. На рисунке 2 представлено изображение с результатами сравнения различных методов машинного обучения для классификации изображений на наборе данных CIFAR-10. DCGAN применялась для извлечения признаков из данных, затем эти признаки использовала модель L2-SVM для предсказаний. DCGAN

не был предварительно обучен для набора данных CIFAR-10, но ансамбль методов показывает хороший результат в классификации.

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

Рисунок 2 – DPGAN на тестовом наборе CIFAR-10

На рисунке 3 представлена таблица сравнения процента ошибки различных методов машинного обучения в классификации изображений номеров на домах из набора данных SVHN. В этой задаче DCGAN применялся также для извлечения признаков, в качестве классификатора использовался L2-SVM. Пара этих методов получила самый маленький процент ошибки на тестовом наборе среди других методов машинного обучения.

Model	error rate
KNN	77.93%
TSVM	66.55%
M1+KNN	65.63%
M1+TSVM	54.33%
M1+M2	36.02%
SWWAE without dropout	27.83%
SWWAE with dropout	23.56%
DCGAN (ours) + L2-SVM	22.48%
Supervised CNN with the same architecture	28.87% (validation)

Рисунок 3 – DCGAN на тестовом наборе SVHN

High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs [16]

В данной статье представлена архитектура Conditional GAN для генерации изображений высокого разрешения. Для достижения этого объединяются два ключевых компонента: генератор и дискриминатор.

Генератор в CGAN обучается создавать изображения, которые могут быть семантически использованы с высоким разрешением. Генератор включает в себя блоки, основанные на архитектуре ResNet, которые помогают

сохранить детали изображений при генерации изображений высокого разрешения.

Дискриминатор в CGAN обучается различать реальные изображения от сгенерированных. Он также принимает на вход семантическую информацию, которая помогает ему более точно оценивать качество и реалистичность сгенерированных изображений.

Важным аспектом работы является способность CGAN к семантической манипуляции с изображениями. Это означает, что при наличии определенных семантических признаков, таких как наличие объектов на изображении, CGAN может изменять эти признаки в результирующем изображении.

В экспериментах авторы показывают, что CGAN способен генерировать изображения высокого разрешения с высоким уровнем детализации и качества, а также успешно выполнять семантическую манипуляцию с этими изображениями. На рисунке 4 представлено сравнение различных архитектур генеративных сетей в задаче семантической сегментации изображений из набора данных Cityscapes. CGAN показывает лучшую точность выделения пикселей на изображениях, чем аналоги.

	U-Net	CRN	Our generator
Pixel acc (%)	77.86	78.96	**83.78**
Mean IoU	0.3905	0.3994	**0.6389**

Рисунок 4 – Сравнение точности генерации CGAN

На рисунке 5 представлено сравнение нескольких архитектур в задаче генерации изображений из набора данных NYU. CGAN генерирует более красочные и реалистичные изображения.

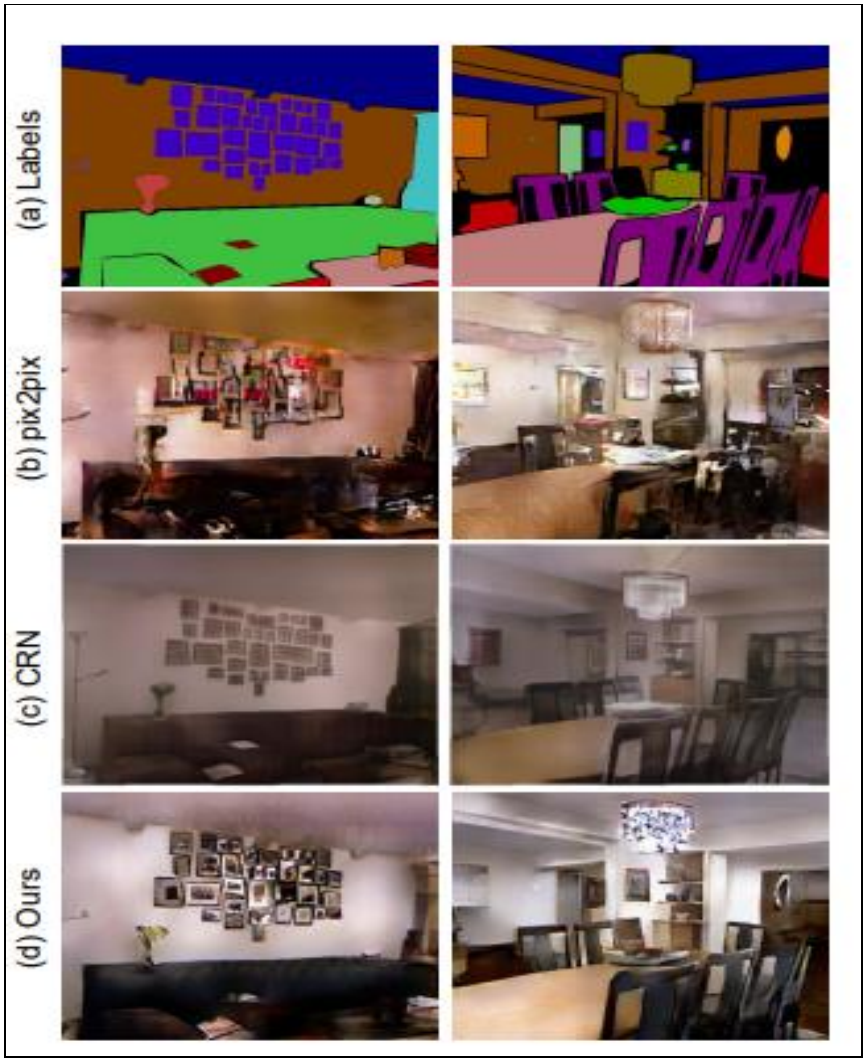

Рисунок 5 – Сравнение качества генерации изображений

Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis [17]

В данной статье представлена архитектура GAN, называемая Global and Local Perception GAN (GLPGAN), для синтеза фронтальных изображений лиц. Главная особенность GLPGAN заключается в том, что она учитывает как глобальные, так и локальные аспекты лицевых признаков при генерации изображений.

Глобальный аспект включает в себя общие характеристики лица, такие как форма глаз, носа и рта, а также расположение их относительно друг друга. Локальный аспект охватывает более мелкие детали, такие как текстура кожи, морщины и прочие детали лицевых черт.

GLPGAN состоит из двух основных компонентов: генератора и дискриминатора. Генератор обучается создавать фронтальные изображения лиц на основе заданных атрибутов, в то время как дискриминатор оценивает реалистичность и сохранение идентичности сгенерированных изображений.

Основное преимущество GLPGAN заключается в том, что она способна сохранять идентичность лица, даже при значительных изменениях в позе и освещении. Это позволяет создавать фотореалистичные фронтальные изображения лиц, которые сохраняют уникальные черты каждого человека.

В экспериментах авторы показывают, что GLPGAN превосходит предыдущие методы синтеза лиц по нескольким метрикам качества, включая реалистичность, сохранение идентичности и стабильность обучения.

Генератор GLPGAN состоит из нескольких сверточных блоков, которые последовательно преобразуют входное изображение (обычно в формате проекции лица) в фронтальное изображение.

Для учета глобальных и локальных аспектов лица генератор включает в себя две ветви. Глобальная ветвь отвечает за учет общих характеристик лица, таких как форма глаз, носа и рта. Она содержит несколько слоев свертки и пулинга для извлечения высокоуровневых признаков.

Локальная ветвь фокусируется на более мелких деталях лица, таких как текстура кожи и морщины. Она также содержит несколько слоев свертки и пулинга, но с более мелкими ядрами свертки для извлечения низкоуровневых признаков.

Дискриминатор в GLPGAN обучается оценивать реалистичность и сохранение идентичности сгенерированных изображений.

Он принимает на вход как реальные фронтальные изображения, так и сгенерированные изображения, и пытается отличить их друг от друга.

Для улучшения стабильности обучения дискриминатор включает в себя несколько слоев свертки и пулинга, а также слоев нормализации.

В процессе обучения GLPGAN также использует механизм восприятия, который оценивает сходство между сгенерированными и реальными изображениями на основе предварительно обученной нейронной сети VGG.

Этот механизм помогает улучшить качество сгенерированных изображений, обеспечивая более высокую реалистичность и сохранение идентичности.

Архитектура GLPGAN представляет собой совокупность глубоких сверточных нейронных сетей, способных учитывать как глобальные, так и локальные аспекты лицевых признаков при генерации фронтальных изображений. Это позволяет ей достигать высокого качества синтеза лиц и сохранение их уникальной идентичности.

A Style-Based Generator Architecture for Generative Adversarial Networks [18]

В статье предлагается архитектура генератора, основанная на стилевых представлениях, которая позволяет контролировать различные аспекты изображения, такие как стиль, разрешение, детализация. Традиционные GAN-архитектуры часто ограничены в контроле над этими аспектами, но StyleGAN предлагает более гибкий и мощный подход.

StyleGAN представляет собой эволюцию архитектуры генеративных состязательных сетей (GAN), которая позволяет более гибко контролировать генерацию изображений. В отличие от предыдущих архитектур, где генератор принимал только случайный шум, StyleGAN использует стилевые векторы, которые помогают контролировать различные аспекты изображения, такие как стиль, детализация и разрешение.

Генератор StyleGAN состоит из нескольких блоков, каждый из которых отвечает за генерацию изображения на определенном уровне разрешения. На каждом уровне разрешения генератор принимает на вход вектор шума и преобразует его в изображение. Однако ключевая особенность StyleGAN заключается в том, что он также принимает на вход стилевые векторы, которые управляют аспектами изображения, такими как форма лица

и распределение цветов. Эти стилевые векторы подаются в модуль стиля, который преобразует грубый вывод генератора в конечное изображение с учетом указанных стилевых характеристик.

StyleGAN предлагает ряд преимуществ по сравнению с другими архитектурами генеративных моделей. Он позволяет генерировать изображения высокого качества с большим разнообразием стилей и детализации. Кроме того, благодаря стилевым векторам, StyleGAN обеспечивает более гибкий контроль над генерацией изображений, что позволяет создавать более реалистичные и разнообразные изображения.

Denoising Diffusion Probabilistic Models [19]

В данной статье представлена концепция диффузионных вероятностных моделей (DDPM), основным задачей этого подхода является создание качественных изображений, используя диффузионные процессы.

Принцип работы модели заключается в применении процесса диффузии к начальному шумовому изображению. На каждом шаге диффузии изображение постепенно улучшается, происходит удаление шума и приближение к оригинальному изображению. Этот процесс достигается с помощью декодера, который на каждом шаге обрабатывает изображение, учитывая предыдущие шаги диффузии.

Для обучения модели используются пары изображений – шумное и оригинальное. Модель минимизирует разницу между сгенерированным и оригинальным изображением, используя метод максимального правдоподобия. Это позволяет модели выучивать правильные признаки для генерации высококачественных изображений.

Модель также показывает гибкость, поскольку ее можно использовать не только для генерации изображений, но и для решения других задач, таких как удаление шума, улучшение качества изображений и обучение нейронных сетей. На рисунке 6 представлена таблица сравнения качества сгенерированных изображений различными архитектурами нейронный сетей.

DDPM показывает лучший результат в качестве генерации новых изображений.

Model	IS	FID	NLL Test (Train)
Conditional			
EBM [11]	8.30	37.9	
JEM [17]	8.76	38.4	
BigGAN [3]	9.22	14.73	
StyleGAN2 + ADA (v1) [29]	**10.06**	**2.67**	
Unconditional			
Diffusion (original) [53]			≤ 5.40
Gated PixelCNN [59]	4.60	65.93	3.03 (2.90)
Sparse Transformer [7]			**2.80**
PixelIQN [43]	5.29	49.46	
EBM [11]	6.78	38.2	
NCSNv2 [56]		31.75	
NCSN [55]	8.87 ± 0.12	25.32	
SNGAN [39]	8.22 ± 0.05	21.7	
SNGAN-DDLS [4]	9.09 ± 0.10	15.42	
StyleGAN2 + ADA (v1) [29]	$\mathbf{9.74} \pm 0.05$	3.26	
Ours (L, fixed isotropic $\mathbf{\Sigma}$)	7.67 ± 0.13	13.51	≤ 3.70 (3.69)
Ours (L_{simple})	9.46 ± 0.11	**3.17**	≤ 3.75 (3.72)

Рисунок 6 – Сравнение качества сгенерированных изображений по метрикам FID и IS

High-Resolution Image Synthesis with Latent Diffusion Models [20]

В статье представлен новый подход к синтезу изображений, использующий модели латентной диффузии (LD). Основная идея модели LD состоит в том, чтобы использовать процесс диффузии в латентном пространстве для генерации изображений. В начале процесса изображение представлено в виде зашумленного состояния в латентном пространстве. Затем с каждым шагом диффузии оно постепенно очищается и приходит к реалистичному состоянию. Этот процесс основывается на обновлении латентного вектора в направлении градиента, оцененного относительно распределения данных. Модель LD обучается на парах изображений оригинальных и соответствующих зашумленных версий. Она минимизирует разницу между синтезированным и оригинальным изображением, используя метод максимального правдоподобия. Кроме того, авторы применяют технику обучения с

подкреплением для дополнительного улучшения качества генерируемых изображений. Результаты экспериментов показывают, что модель LD демонстрирует высокую способность создавать изображения высокого разрешения с высоким качеством. Это достигается благодаря эффективному использованию процесса диффузии и обучению с использованием различных техник.

Одной из ключевых особенностей этой работы является ее способность генерировать изображения с разрешением 1024x1024 и выше, что ранее было сложной задачей для многих методов генеративного моделирования. На рисунках 7 и 8 представлены результаты генерации изображений с помощью модели LD, обученной на наборе данных LAION.

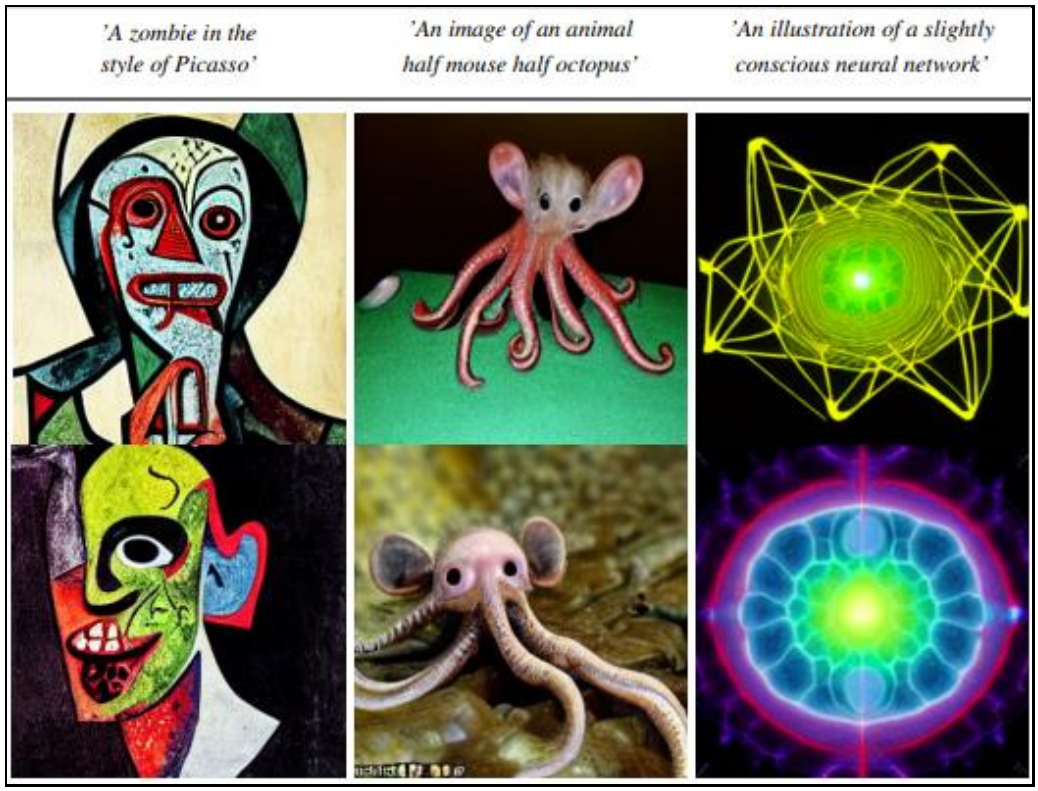

Рисунок 7 – Генерация изображений по тексту

Рисунок 8 – Пример генерации изображений

На представленных выше изображениях можно отметить высокую детализацию сгенерированных изображений, они также отражают текст, использованный для их генерации.

Для генерации изображений без текста модель обучалась на таких наборах как CelebA-HQ [21], FFHQ [22], LSUN-Churches [23] и Bedrooms [24]. Результаты обучения представлены на рисунках 9–12.

CelebA-HQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
DC-VAE [63]	15.8	-	-
VQGAN+T. [23] (k=400)	10.2	-	-
PGGAN [39]	8.0	-	-
LSGM [93]	7.22	-	-
UDM [43]	7.16	-	-
LDM-4 (ours, 500-s†)	**5.11**	0.72	0.49

Рисунок 9 – Сравнение на наборе CelebA-HQ

FFHQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
ImageBART [21]	9.57	-	-
U-Net GAN (+aug) [77]	10.9 (7.6)	-	-
UDM [43]	5.54	-	-
StyleGAN [41]	4.16	0.71	0.46
ProjectedGAN [76]	**3.08**	0.65	0.46
LDM-4 (ours, 200-s)	4.98	**0.73**	**0.50**

Рисунок 10 – Сравнение на наборе FFHQ

На наборе FFHQ модель показывает не последний результат по метрике FID и лучшие показатели по метрикам precision и recall.

LSUN-Churches 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
DDPM [30]	7.89	-	-
ImageBART [21]	7.32	-	-
PGGAN [39]	6.42	-	-
StyleGAN [41]	4.21	-	-
StyleGAN2 [42]	3.86	-	-
ProjectedGAN [76]	**1.59**	0.61	0.44
*LDM-8** (ours, 200-s)	4.02	**0.64**	**0.52**

Рисунок 11 – Сравнение на наборе LSUN

На наборе LSUN модель показывает такие же результаты, как и на предыдущем наборе.

LSUN-Bedrooms 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
ImageBART [21]	5.51	-	-
DDPM [30]	4.9	-	-
UDM [43]	4.57	-	-
StyleGAN [41]	2.35	0.59	0.48
ADM [15]	1.90	**0.66**	**0.51**
ProjectedGAN [76]	**1.52**	0.61	0.34
LDM-4 (ours, 200-s)	2.95	**0.66**	0.48

Рисунок 12 – Сравнение на наборе Bedrooms

По результатам обучения на наборах данных модель показывает хорошие показатели по метрикам в сравнении с другими архитектурами. Также стоит отметить высокую детализацию сгенерированных изображений.

Вывод по второй главе

В этой главе были приведены примеры различных ГНС и ДНС. Для создания изображений фасадов зданий на основе заданной маски отлично подходят диффузионные нейронные сети, которые на вход принимают не только исходное изображение, но и текстовые данные. Такие нейронные сети показывают хороший результат в обработке и постобработке изображений, они обеспечивают высокое качество генерации с точностью, необходимой для успешного выполнения задачи обработки.

3. ПРОЕКТИРОВАНИЕ

3.1. Требования

Было решено решать задачу очистки фасадов зданий посредством веб-приложения, где пользователь может загрузить нужное изображение и получить изображение с очищенным фасадом здания с заданными параметрами для обработки. В данной работе будет представлена реализация сервиса, который включает в себя нейронную сеть, генерирующую новые изображения с заданными параметрами, а также может обмениваться информацией с другими сервисами системы.

Функциональные требования

Функциональное требование – это заявление о том, как должна вести себя система. Он определяет, что система должна делать, чтобы удовлетворить потребности или ожидания пользователя. Функциональные требования можно рассматривать как функции, которые обнаруживает пользователь. Для приложения были выделены следующие функциональные требования.

1. Пользователь может загрузить в приложение изображение и получить новое сгенерированное изображение.
2. Обработка изображений различных форматов (PNG, JPG и т.д.).
3. Возможность загружать изображения размером до 100 мегабайт.
4. Возможность скачивания изображений на устройство пользователя.
5. Текущая сессия пользователя должна сохраняться.
6. Пользовательский интерфейс должен быть написан на языке JavaScript.
7. Серверная часть приложения должна быть написана на языке C#.
8. Сервис обработки изображений должен быть написан на языке Python.

Нефункциональные требования

Нефункциональные требования – это требования, определяющие свойства, которые система должна демонстрировать, или ограничения, которые она должна соблюдать, не относящиеся к поведению системы. Были выделены следующие нефункциональные требования.

1. Система должна быть расширяемой.
2. Доставка и развертывание приложения должны быть реализованы с помощью CI/CD.
3. У окружения, в котором развернуто приложение, должна быть возможность предоставить отказоустойчивость.
4. Приложение должно быть публично доступно.
5. Должна быть возможность миграции приложения из одного окружения в другое.

3.2. Архитектура приложения

В ходе разработки решения была предварительно разработана архитектура всей системы, включающая все модули и основное взаимодействие между ними (рисунок 13).

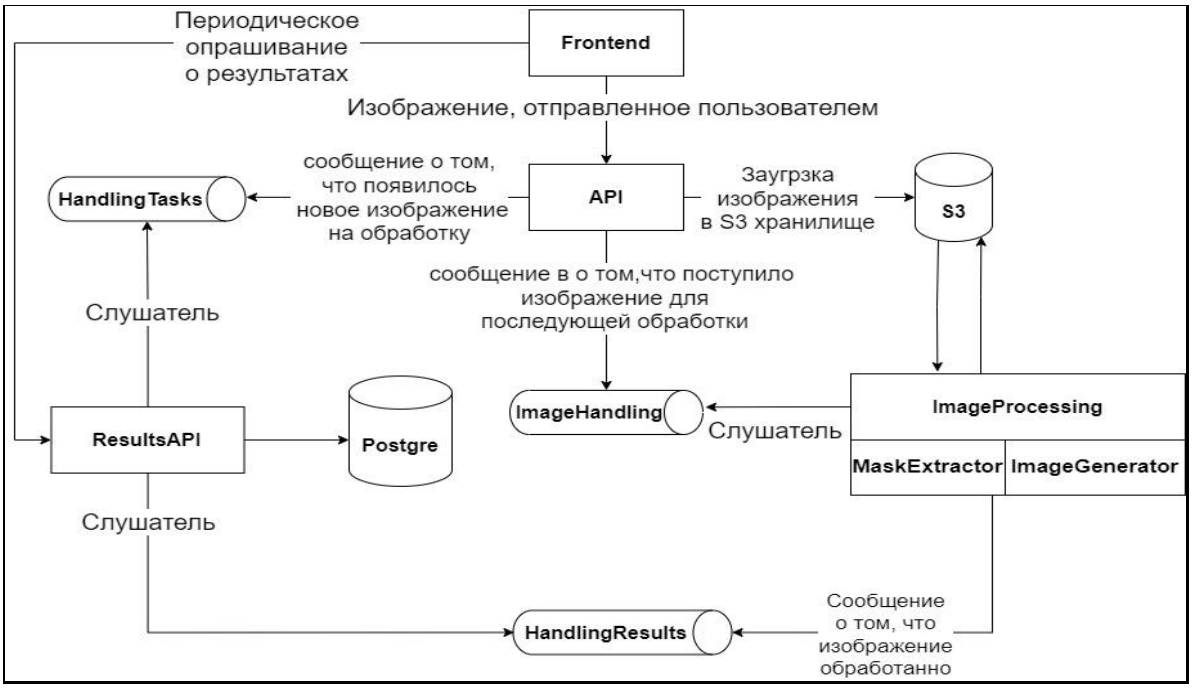

Рисунок 13 – Архитектура приложения

Сервис frontend отвечает за предоставление пользователю графического интерфейса для работы с приложением. После загрузки нового изображения пользователем сервис периодически опрашивает сервис ResultsAPI о том, завершилась ли обработка изображения для текущей сессии пользователя.

Сервис API предоставляет для frontend’а запросы, которые управляют приложением.

S3 – файловое хранилище для хранения изображений, их загрузку и выгрузку [25].

ImageProcessing – сервис для обработки изображений, включает в себя две нейросетевые модели.

MaskExtractor – сервис, предоставляющий интерфейс для использования модели сегментации изображения для извлечения маски из изображения, необходим для выделения фасада на изображении здания.

ImageGenerator – модель для генерации нового изображения на основе маски, предоставляемой моделью MaskExtractor.

Также сервис ImageProcessing взаимодействует с другими сервисами с помощью очередей.

ImageHandling – очередь, которая необходима для общения между сервисом API и ImageProcessing. API отправляет сообщение в очередь, когда пользователь загружает новое изображение.

HandlingResults – очередь, отвечающая за обмен сообщениями между сервисом ImageHandling и ResultsAPI. ResultsAPI слушает очередь, когда поступает сообщение о том, что обработанно новое изображение, ResultsAPI отправляет сообщение в frontend о появлении результата, чтобы отобразить его для пользователя. ImageHandling отправляет в очередь сообщения о завершении обработки изображения.

ResultsAPI – отвечат за отправку в сервис frontend сообщения, что изображение обработанно для текущей сессии пользователя, также он слушает очереди HandlingResults и HandlingTasks.

HandlingTasks – очередь, в которую пишет сообщение API о появлении нового изображения на обработку.

Очереди реализованы в брокере сообщений RabbitMQ [26].

Когда пользователь загружает изображение, запрос на стороне API сохраняет это изображение в файловое хранилище. Затем API отправляет сообщение в очередь, которую слушает сервис обработки изображений. Далее сервис ImageProcessing отправляет запрос на сегментацию изображения в сервис MaskExtractor. После получения маски из сервиса сегментации начинается генерация нового изображения на основе исходного и маски, после завершения обработки итоговое изображение сохраняется в файловое хранилище, и ImageProcessing отправляет сообщение о завершении обработки в очередь, которую слушает сервис ResultsAPI. Сервис результатов отправляет в frontend статус завершения обработки изображения и ссылку на это изображение.

3.3. Диаграмма вариантов использования системы

Диаграмма прецедентов или диаграмма вариантов использования – это графическое представление, отображающее отношения между акторами и прецедентами (вариантами использования) и являющееся составной частью модели прецедентов, которая позволяет описать систему на концептуальном уровне. Диаграмма вариантов использования играет ключевую роль в моделировании и анализе требований к системе, предоставляя высокоуровневое представление о функциях системы и их взаимодействии с внешними сущностями. В рамках разрабатываемого приложения была спроектирована диаграмма вариантов использования приложения, представленная на рисунке 14.

Пользователь – это основной актер, который взаимодействует с системой. На диаграмме отраженно, какие варианты использования он может совершить. Актер может загрузить изображение в систему, изображение сохраняется в облачном хранилище данных, далее пользователь может

обработать загруженное изображение. Тогда со стороны приложения у изображения выделяется маска, затем по выделенной маске генерируется новое изображение, оно сохраняется в хранилище, и пользователь может увидеть это изображение и загрузить.

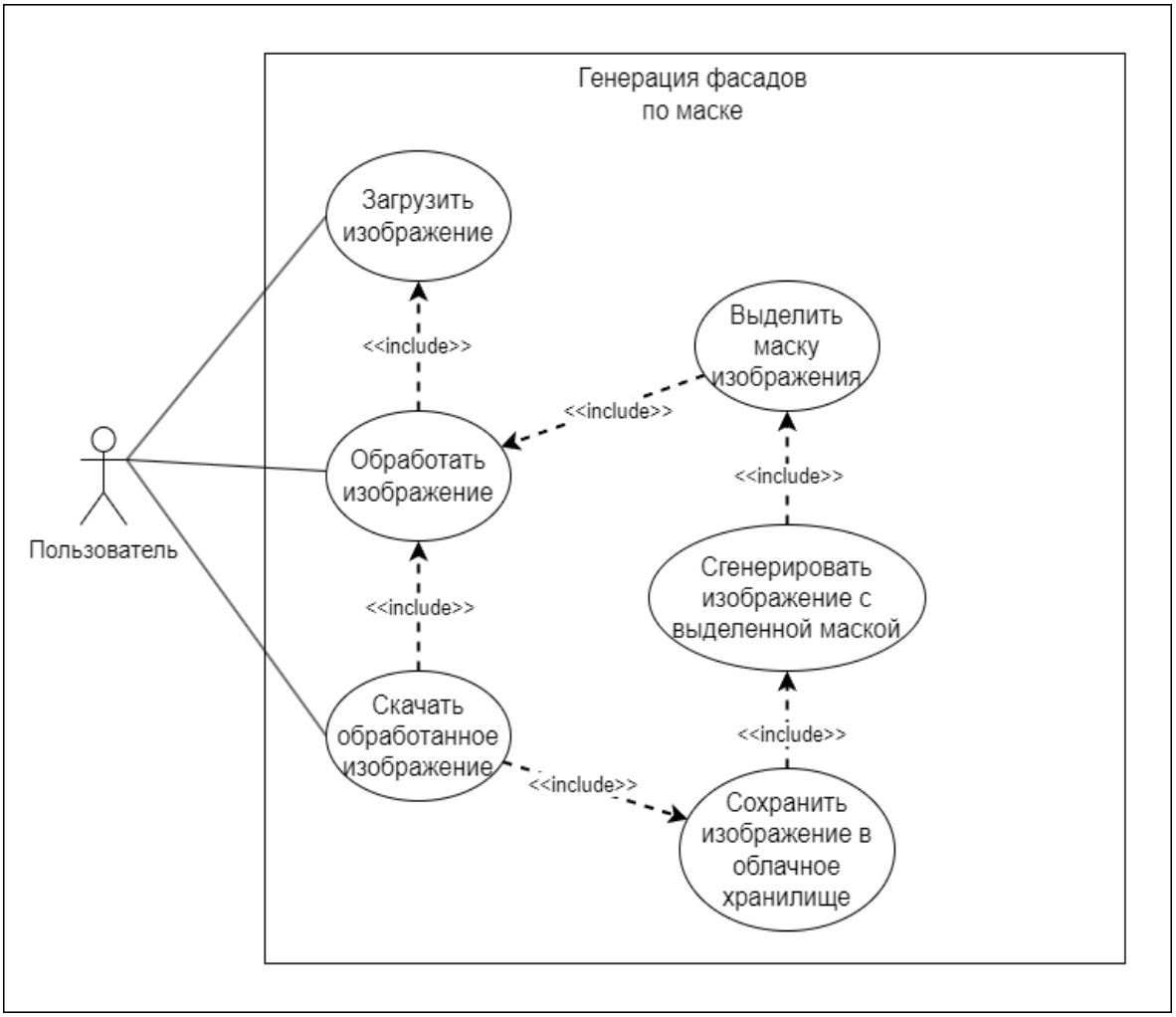

Рисунок 14 – Диаграмма вариантов использования

3.4. Диаграмма компонентов

Диаграмма компонентов – это статическая, структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. Для приложения была разработана диаграмма компонентов, она представлена на рисунке 15.

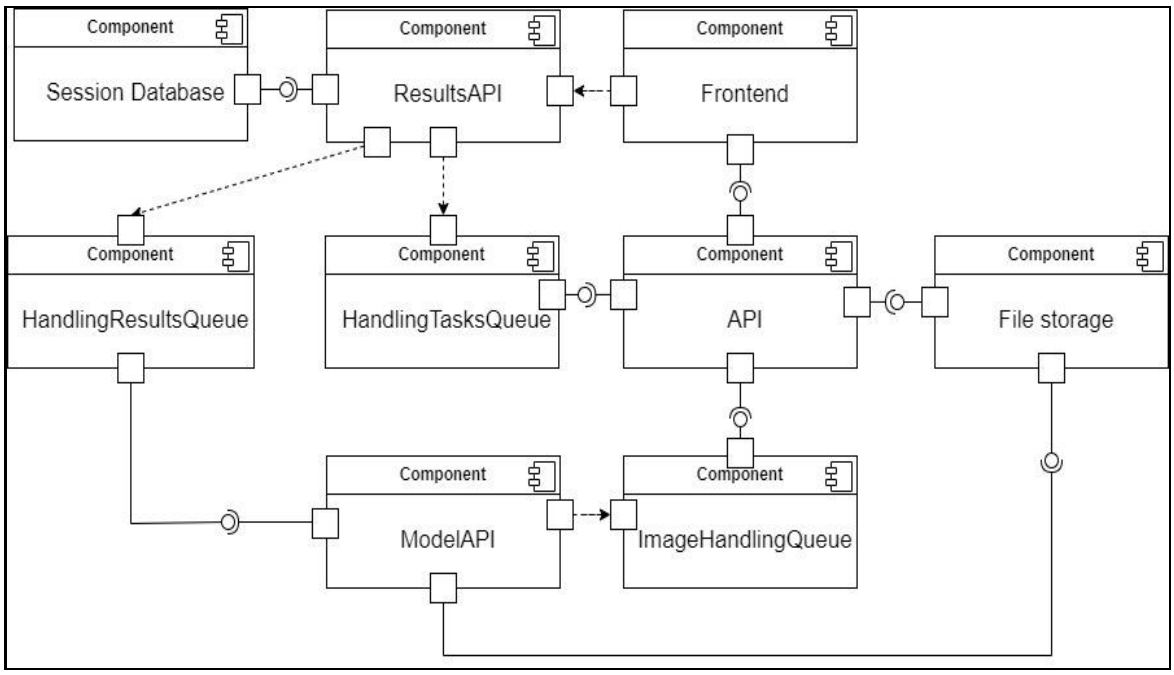

Рисунок 15 – Диаграмма компонентов

Диаграмма состоит из следующих компонентов.

1. Frontend – визуальный интерфейс для пользователя, который для взаимодействия с системой использует API.

2. API – элемент системы, отвечающий за основные запросы по загрузке и выгрузке изображений. Пишет сообщения в очередь ImageHandlingQueue и HandlingTasksQueue.

3. FileStorage – компонент, ответственный за хранение изображений.

4. ImageHandlingQueue – очередь, в которой появляются сообщения о поступлении нового изображения на обработку.

5. HandlingTasksQueue – компонент системы, представляющий собой очередь, в которой появляются сообщения о том, что поступившее изображение обрабатывается.

6. HandlingResultsQueue – очередь, в которой пишутся сообщения о том, что изображение прошло обработку.

7. ModelAPI – компонент системы, который отвечает за обработку изображений, выделения в них нужной маски, и генерацию нового изображения на основе выделенной маски. Пишет о результатах обработки

изображения в очередь HandlingResultsQueue, слушает очередь ImageHandlingQueue на наличие новых запросов для обработки. Также загружает поступившее на обработку изображение из файлового хранилища – FileStorage.

8. Session Database – компонент системы, отвечающий за хранение текущей сессии пользователя.

9. ResultsAPI – компонент системы, отвечающий за уведомление компонента Frontend о начале и окончании обработки поступившего изображения.

Вывод по третьей главе

В этой главе были определены функциональные и нефункциональные требования к системе, что позволило точно описать, какие задачи должно выполнять приложение, и как оно должно себя вести в различных условиях. Также в этой главе была спроектирована архитектура приложения, определены основные компоненты и их взаимодействие. Архитектура описывает, как различные модули и сервисы приложения будут взаимодействовать друг с другом, обеспечивая выполнение всех функциональных и нефункциональных требований. Спроектированная архитектура создает прочную основу для разработки приложения, обеспечивая его структурированность, модульность и возможность дальнейшего расширения и модификации.

4. РЕАЛИЗАЦИЯ

4.1. Программные средства реализации

Для разработки программной части приложения были выбран языки программирования Python 3.8, C# 12, JavaScript ES 6 с фреймворком React 18. Visual Studio Code.

Для настройки всех зависимостей в Python использовалась библиотека poetry версии 1.2.2 [27].

Серверная часть приложения была реализована с использованием библиотеки .NET 8.0 [28].

Окружение было развернуто на ОС Ubuntu 22.04 в оркестраторе виртуальных машин Opennebula 6.4.0 в k8s 1.27.

База данных PostgreSQL 15 [29].

Брокер сообщений RabbitMQ 3.13.

Облачный сервис для хранения файлов S3 2024.5.1.

В качестве балансировщика нагрузки использовался nginx 1.25.5 [30]

Для работы с k8s были использованы следующие пакеты: kubectl 1.29.1 [31], Helm 3.14.1 [32], Docker 25.0.3 [33].

Для работы с нейронными сетями были использованы следующие библиотеки: PyTorch версии 2.2.2 [34], torchvision версии 0.17.2 [35], pillow версии 10.3.0 [36], numpy версии 1.19.5 [37], albumentations версии 1.4.3 [38], scikit-learn версии 1.4.2 [39], pandas версии 2.2.2 [40].

Для автоматизации проведения экспериментов и запуска этапов обучения, и тестирования нейронной сети использовался механизм GitLab CI/CD.

4.2. Модель для выделения маски из изображения

Семантическая сегментация включает классификацию каждого пикселя изображения, присваивая его определенному классу объектов. Это позволяет точно определить границы объектов на изображении, что полезно для таких задач, как автоматическое распознавание и анализ изображений.

В отличие от детектирования объектов, семантическая сегментация требует значительных вычислительных мощностей, поскольку необходимо обработать и классифицировать каждый отдельный пиксель.

Основная идея PSPNet [41] заключается в использовании пирамидальной агрегации контекста, при которой изображение разбивается на несколько областей разного размера, и для каждой из них вычисляется статистика, отражающая ее контекст (рисунок 16). Ключевая особенность PSPNet это применение пирамидального модуля (pyramid pooling module), который позволяет сети адаптироваться к объектам различных размеров в сцене. Этот модуль помогает сети учитывать контекстную информацию на разных уровнях детализации, что важно для точной сегментации объектов различных размеров.

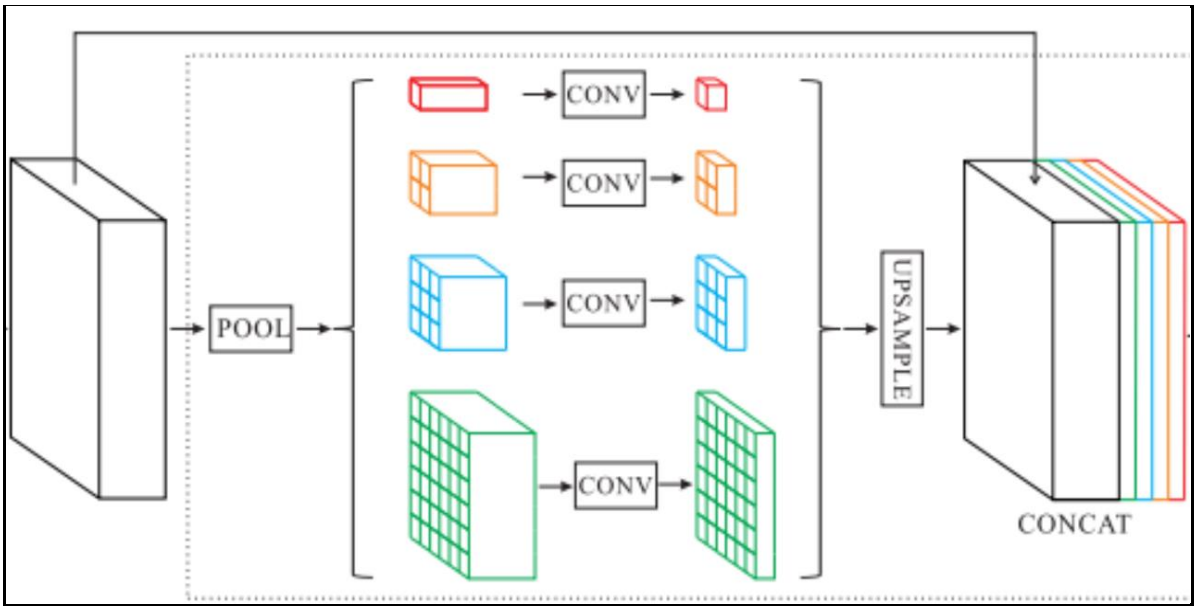

Рисунок 16 – Архитектура сегментатора

4.3. Латентная диффузионная нейросетевая модель

В данной работе применялась латентная диффузионная модель [20]. Основная особенность такой модели в том, что на этапе удаления шума из изображения модель обучается на скрытом пространстве небольшой размерности, нежели на многомерном пиксельном пространстве. Такой подход позволяет ускорить обучение, позволяя быстрее сходиться в нужную

область, также в таком пространстве модель легче находит новые признаки из изображения. На рисунке 17 представлена архитектура модели.

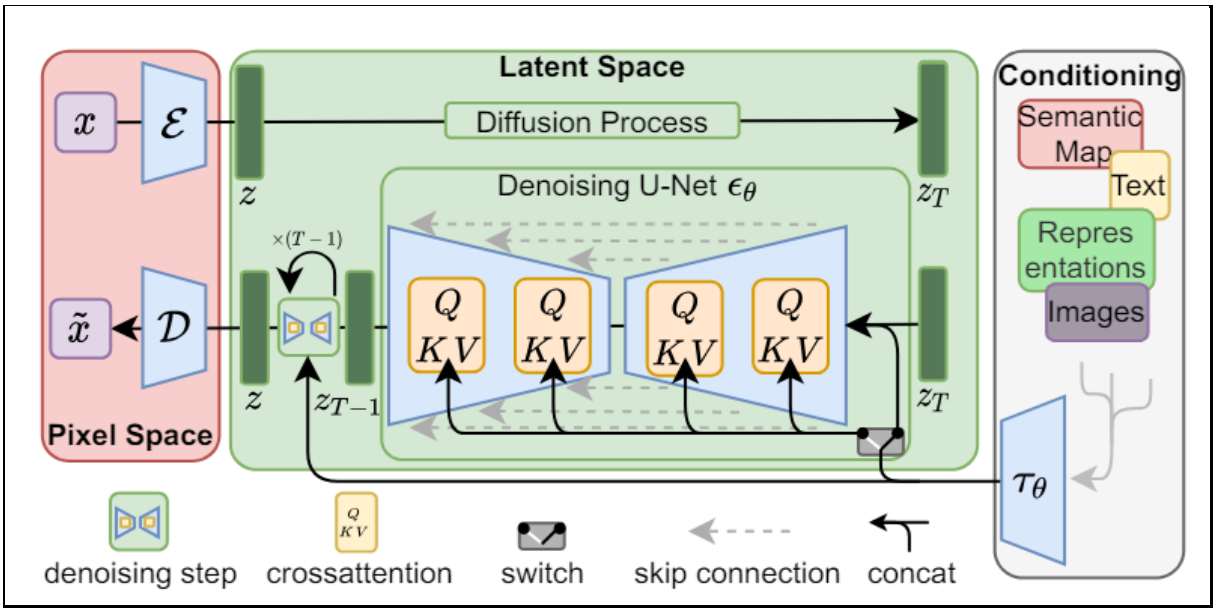

Рисунок 17 – Архитектура генератора

Диффузионные модели состоят из двух этапов. На первом этапе данные объединяются с шумом, путем постепенного внесения помех в исходную информацию, это достигается с помощью случайного процесса, который начинается с исходного образца и итеративно генерирует зашумленные экземпляры, используя простое гауссовое ядро диффузии, такой процесс применяется только во время обучения. На втором этапе происходит процесс отката диффузии и итеративного удаления шума. Он представляет из себя синтез данных и обучен генерировать информацию путем преобразования случайного шума в реалистичные данные.

Эти два этапа требуют много вычислительных ресурсов, поэтому в данной модели обучение проводилось не на исходных данных, а на преобразованных с помощью автокодировщика с коэффициентом редукции 8. То есть исходное изображение уменьшается в 8 раз.

Архитектура этой модели состоит из трех частей.

1. Автокодировщик. На вход модель получает случайный шум, далее образец редуцируется в латентное пространство до меньшего размера. Для

этого использовалась архитектура VAE [42]. Она состоит из двух частей кодировщика и декодировщика. Кодировщик используется во время обучения и отвечает за уменьшение исходного изображение в пространство меньшей размерности. Декодировщик преобразует, прошедшие этап удаления шума, редуцированные изображения в их исходный размер. На рисунке 18 представлена архитектура VAE.

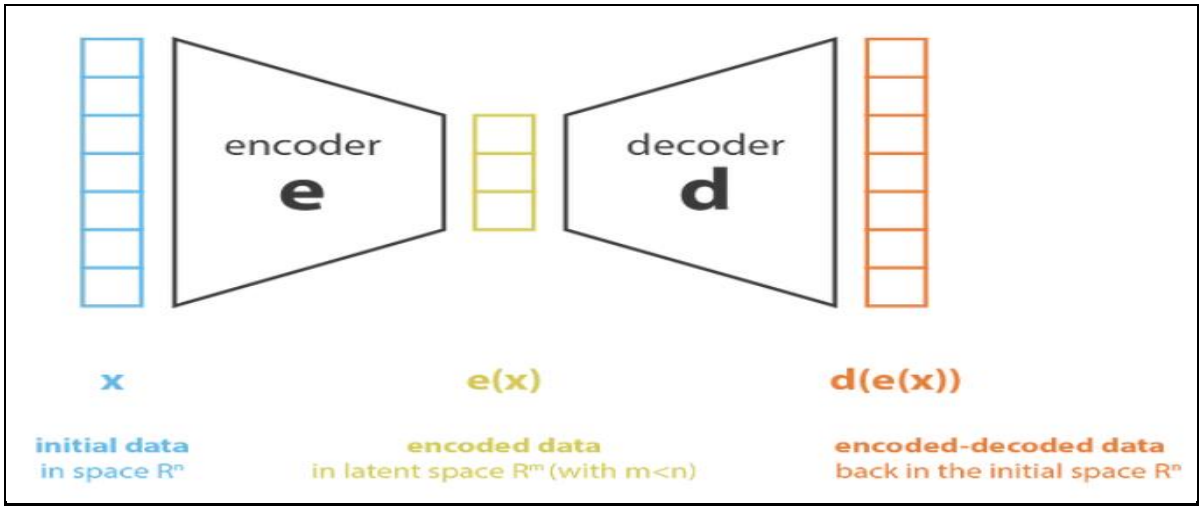

Рисунок 18 – Архитектура VAE

2. Блок U–Net [43]. Он состоит из ResNet [44], получает зашумленный образец в латентном пространстве и удаляет шум из него, затем данные передаются в декодировщик, чтобы преобразовать изображение в исходную размерность.

3. Кодировщик текста преобразует входные текстовые данные для их последующего встраивания в слои блока U–Net. В этой модели применялся CLIP ViT-L/14 [45].

Обучение таких моделей все же занимает много вычислительных ресурсов и времени. Сбор наборов данных для обучения также довольно трудозатратный процесс, поэтому было принято решение использовать уже предобученную модель.

4.4. Настройка окружения для развертывания

Для данного веб приложения было принято решение разворачивать его в k8s. Это позволит легко масштабировать систему, отслеживать ее состояние. Написанные конфигурационные файлы позволят мигрировать все приложение в любое облако, где предоставляется k8s, как сервис. Хост сервер, где будет установлен k8s работает на ОС Ubuntu 22.04.

Nginx

Весь трафик нужный для приложения будет проходить через установленный на хост сервере nginx, и направляться на нужные виртуальные машины в opennebula. Содержимое файла конфигурации nginx.conf представлено в листинге 1 приложения А.

В листинге представлены виртуальные машины, на которые будет направляться трафик. Сервер для RabbitMQ развернут отдельно. Как видно в конфигурационном файле, весь http/https трафик направляется напрямую в виртуальные машины k8s.

Opennebula

Openebula – провайдер облачных вычислений. Он позволяет настраивать приватную инфраструктуру со своей сетью и виртуальными машинами. После установки OpenNebula создается приватная сеть по умолчанию для виртуальных машин. Ее нужно использовать при создании виртуальных машин. Такая сеть будет доступна только для сервера хоста, на котором развернуты виртуальные машины этой сети. С помощью nginx только нужный трафик будет направляться на виртуальные машины. Создавать виртуальные машины можно через ui – интерфейс OpenNebula. Для данной работы были созданы следующие виртуальные машины.

1. K8s–master – узел, который отвечает за обработку api запросов в k8s.
2. K8s–slave – узел, на котором будут разворачиваться основные сервисы приложения.

3. K8s–slave–large – узел, с большим количеством ресурсов хоста, на котором будут разворачиваться требовательные по вычислительной нагрузке сервисы.

4. RabbitMQ – сервер, на котором развернут брокер очередей.

Установка вспомогательных сервисов для кластера

После настройки сетей, разворачивания необходимых виртуальных машин, установки на них k8s необходимо установить дополнительные инструменты в кластер для того, чтобы была возможность разворачивать в нем приложение.

Ingress-nginx

Сущность ingress-nginx представляет собой объект k8s, который отвечает за балансировку нагрузки и направление трафика на нужные сервисы кластера. Код представлен в листинге 2 приложения А.

После установки этого компонента в кластер, на каждом узле появится новый под, который слушает указанные в конфигурационном файле порты для направления трафика в нужные сущности, которые называются ingress. Каждый сервис приложения будет развернут вместе с ingress.

Cert-manager

Эта сущность отвечает за выписывание ssl сертификатов для доменов, которые использует приложение. После его установки необходимо также создать сущность ClusterIssuer, который и будет заниматься выписыванием сертификатов, полученных с сервера letsencrypt. Конфигурационный файл для создания этой сущности представлен в листинге 1.

Листинг 1 – Код letsencrypt.yaml

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt
  namespace: cert-manager
spec:
  acme:
    # The ACME server URL
    server: https://acme-v02.api.letsencrypt.org/directory
    # Email address used for ACME registration
    email: <use emal for notifications>
    # Name of a secret used to store the ACME account private key
```

```
    privateKeySecretRef:
      name: letsencrypt
    # Enable the HTTP-01 challenge provider
    solvers:
    - http01:
        ingress:
          class: nginx
```

Разворачивание S3

В данной работе S3 используется как файловое хранилище, к которому будут иметь доступ все приложения системы. Для S3 также существует helm-chart, позволяющий развернуть этот сервис в k8s. В листинге 2 представлен конфигурационный файл для разворачивания этого сервиса.

Листинг 2 – Код s3.yaml

```
auth:
  rootUser: <admin user>
  rootPassword: <admin user password>

defaultBuckets: <buckets to create>

provisioning:
  enabled: true
  users:
    - username: <key to api access>
      password: <secret key to api access>
      disabled: false
      policies:
        - readwrite
ingress:
  enabled: true
  ingressClassName: "nginx"
  hostname: <domain name for ui>
  path: /
apiIngress:
  enabled: true
  ingressClassName: "nginx"
  hostname: <domaon name for api>
  path: /

persistence:
  size: <persistent volume size in GB>
```

После разворачивания сервиса, он будет доступен по указанному в конфигурационном файле доменному имени. На рисунке 19 представлено окно аутентификации в сервис.

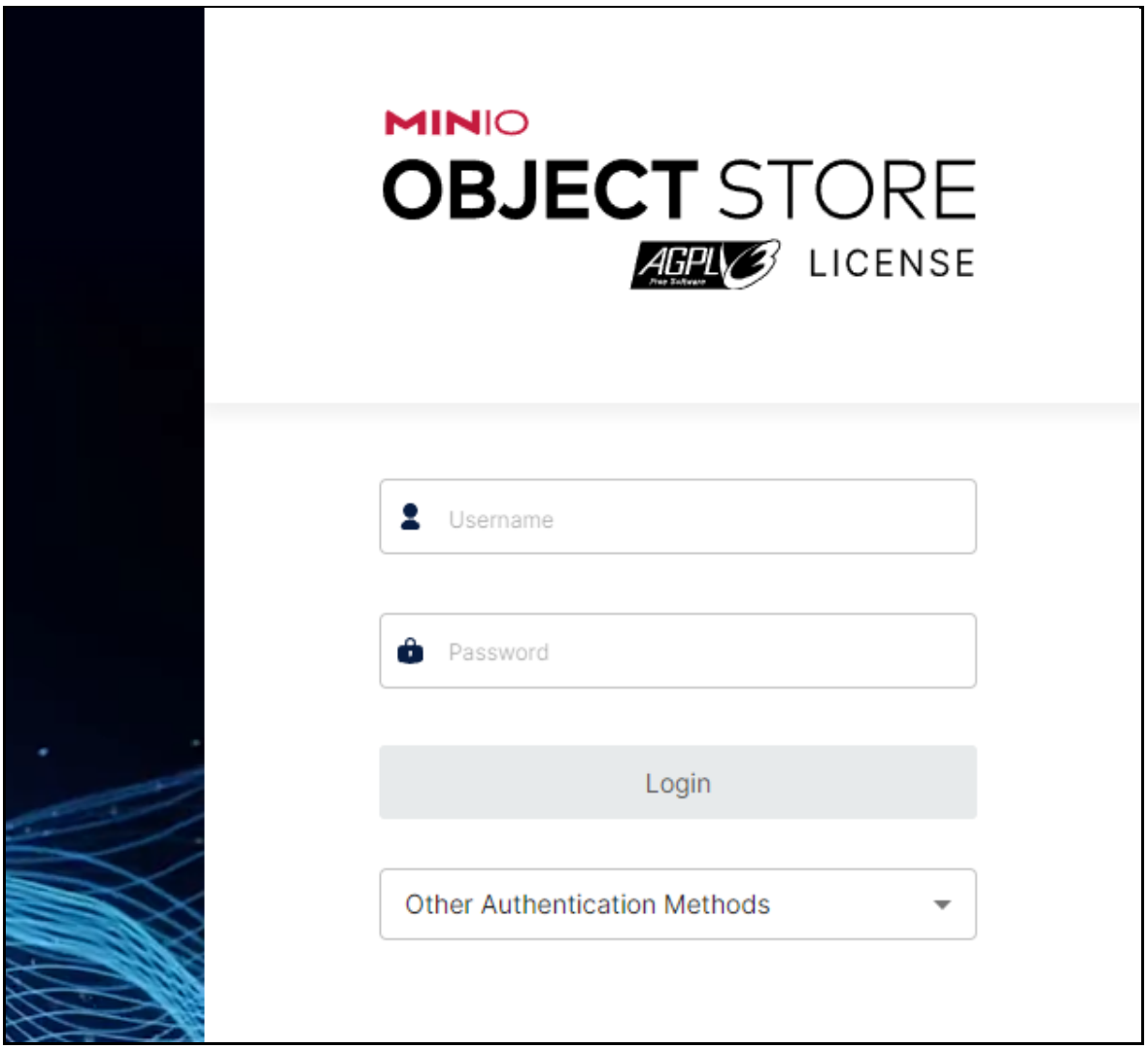

Рисунок 19 – Страница аутентификации в S3

Разворачивание PostgreSQL

В работе база данных PostgreSQL также развернута в кластере, чтобы сервер базы данных находился в одной сети с сервисами, которым необходим доступ к ней. В листинге 3 представлен конфигурационный файл для разворачивания postgres.

Листинг 3 – Код postgre.yaml

```
global:
    postgresql:
        auth:
            postgresPassword: <postgre password>
            username: <postgre user name>
            password: <postgre user password>
            database: <data base name>

primary:
    service:
```

```
        type: NodePort
        nodePorts:
            postgresql: <openned postgre port>

    persistence:
        size: <persistent volume size in GB>
```

После установки сервиса postgres он станет доступен на открытом порту. На рисунке 20 представлено подключение в базе данных из приложения DBeaver[43].

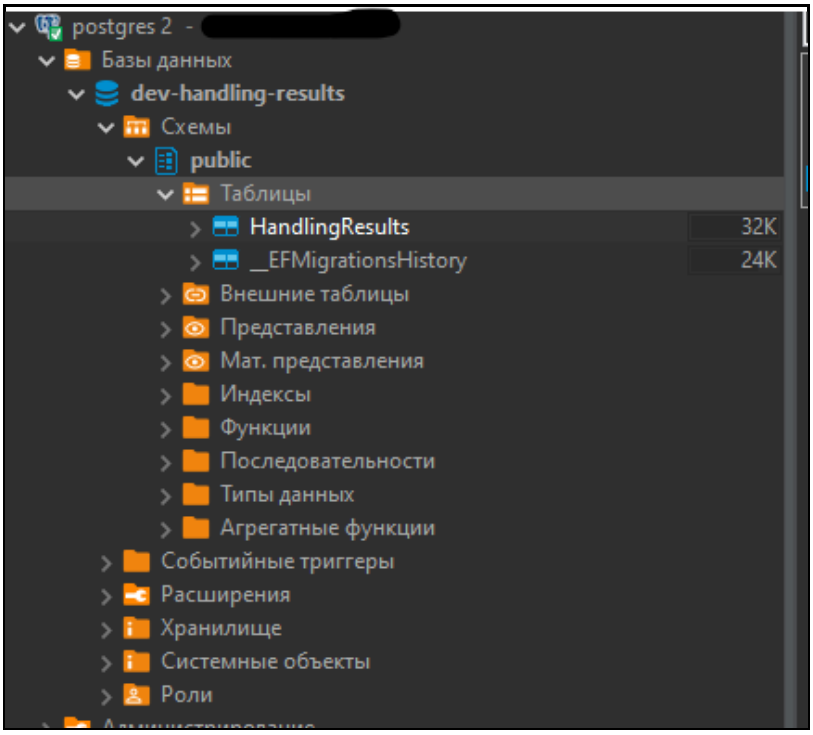

Рисунок 20 – Пример подключения к БД

Разворачивание сервисов приложения

Все сервисы системы также развернуты в кластере, они используют один helm-chart. В листинге 4 представлен файл значений для этих сервисов.

Листинг 4 – Код values.yaml

```
image:
    tag: latest

    pullSecrets:
        - <name of service>-nginx-docker-secret
    pullCredentials:
        username: <name for registry user>
        password: <password for registry user>

ingress:
    enabled: true
```

```
ingressClassName: "nginx"
hostname: <service domain name>
path: <path for service endpoint>
tls: true
```

Разворачивание сервисов проходит автоматически в gitlab CI/CD, используя конфигурационный файл, представленный выше.

4.5. Настройка CI/CD

Для автоматической доставки кода в кластер и сохранения истории изменений использовался gitlab CI/CD. Кластеру k8s необходим docker образ приложения. Для ui был написан такой Dockerfile, его код представлен в листинге 5.

Листинг 5 – Код Dockerfile для ui

```
FROM node:18.14.1 as build
ENV PATH /node_modules/.bin:$PATH
COPY package.json ./
COPY package-lock.json ./
RUN npm ci
COPY . ./
RUN npm run build

FROM nginx:1.16.1-alpine
COPY /ci/nginx.conf /data/conf/nginx.conf
COPY --from=build /dist /usr/share/nginx/html
EXPOSE 80

WORKDIR /usr/share/nginx/html
COPY ./ci/env.sh .
COPY .config-keys .
RUN apk add --no-cache bash
RUN chmod +x /usr/share/nginx/html/env.sh

CMD /bin/bash -c "/usr/share/nginx/html/env.sh" && nginx -g "daemon off;"
-c "/data/conf/nginx.conf"
```

Из такого Dockerfile можно собрать образ приложения, которое потом будет разворачиваться в k8s.

Для api сервиса, написанном на языке C#, код Dockerfile представлен в листинге 6.

Листинг 6 – Код Dockerfile для api

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
```

```
WORKDIR /src
COPY ["API/API.csproj", "API/"]
COPY ["Infrastructure.RabbitMq/Infrastructure.RabbitMq.csproj", "Infra-
structure.RabbitMq/"]
RUN dotnet restore "API/API.csproj"
COPY . .
WORKDIR "/src/API"
RUN dotnet build "API.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "API.csproj" -c Release -o /app/publish
/p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "API.dll"]
```

Это многослойный образ, который впоследствии можно оптимизировать и ускорить итоговую сборку образа в CI/CD.

Для сервиса ModelAPI, написанном на языке python, код Dockerfile представлен в листинге 7.

Листинг 7 – Код Dockerfile для modelApi

```
FROM python:3.8.10-slim as base

ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONFAULTHANDLER 1
ENV PYTHONUNBUFFERED 1

RUN python -m pip install --upgrade pip

ENV POETRY_VERSION=1.2.2
ENV POETRY_HOME='/opt/poetry'
ENV POETRY_VENV='/opt/poetry-venv'
ENV POETRY_CACHE_DIR='/opt/.cache'
ENV PYTHONPATH="${PYTHONPATH}:/app-workspace"

WORKDIR /app-workspace

COPY poetry.lock /app-workspace
COPY pyproject.toml /app-workspace
RUN python -m venv $POETRY_VENV \
    && $POETRY_VENV/bin/pip install -U pip setuptools \
    && $POETRY_VENV/bin/pip install poetry==${POETRY_VERSION}

ENV PATH="$POETRY_VENV/bin:$PATH"

RUN poetry config virtualenvs.create false
RUN poetry install --no-interaction --no-ansi

COPY . .

CMD poetry run python application.py
```

Для настройки gitlab CI/CD необходимо написать gitlab-ci.yaml файл, в котором будет описан пайплайн для разворачивания приложения в кластере. Код пайплайна представлен в листинге 3 приложения А.

Пайплайн состоит из двух этапов. Первый этап собирает образ сервиса и загружает его в хранилище образов. Второй этап отправляет запрос в кластер на разворачивание этого сервиса с нужными ему переменными среды. На рисунке 21 представлено, как выглядит этот пайплайн в gitlab.

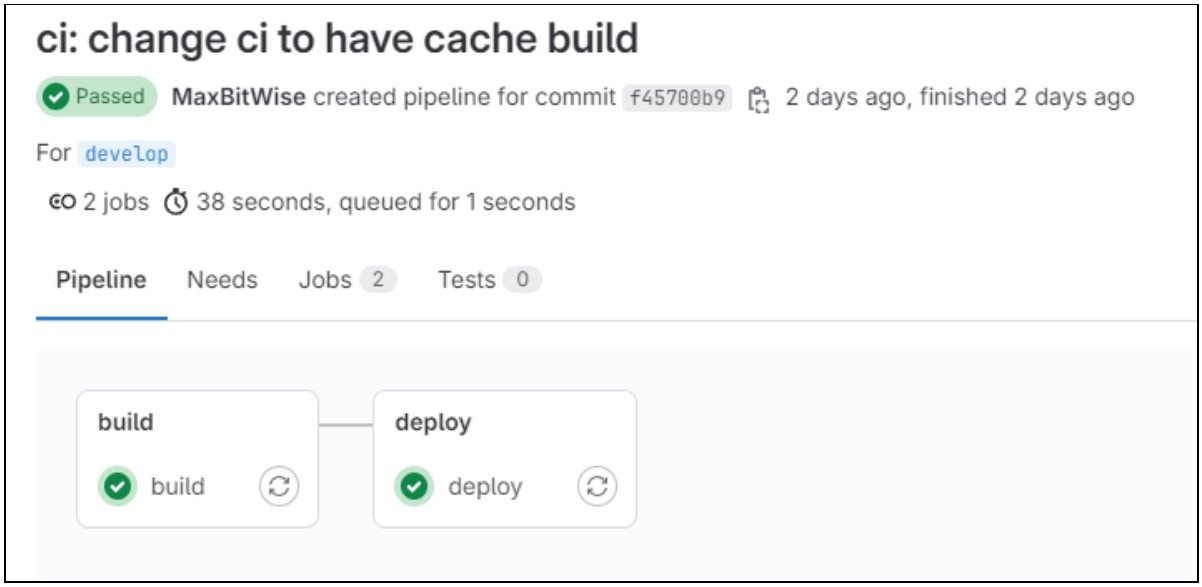

Рисунок 21 – Страница gitlab с примером пайплайна

Таким образом, реализуется автоматическая доставка сервиса приложения в кластер, а также версионирование этого сервиса, так что, по необходимости, можно будет вернуть предыдущую версию сервиса в кластере по нажатию пары кнопок.

4.6. Разработка клиентской части приложения

Frontend состоит из трех основных компонентов.

1. Форма для загрузки изображений. В нее загружается изображение, затем после нажатия на кнопку, запрос с изображением отправляется в api. На рисунке 22 показан этот компонент.

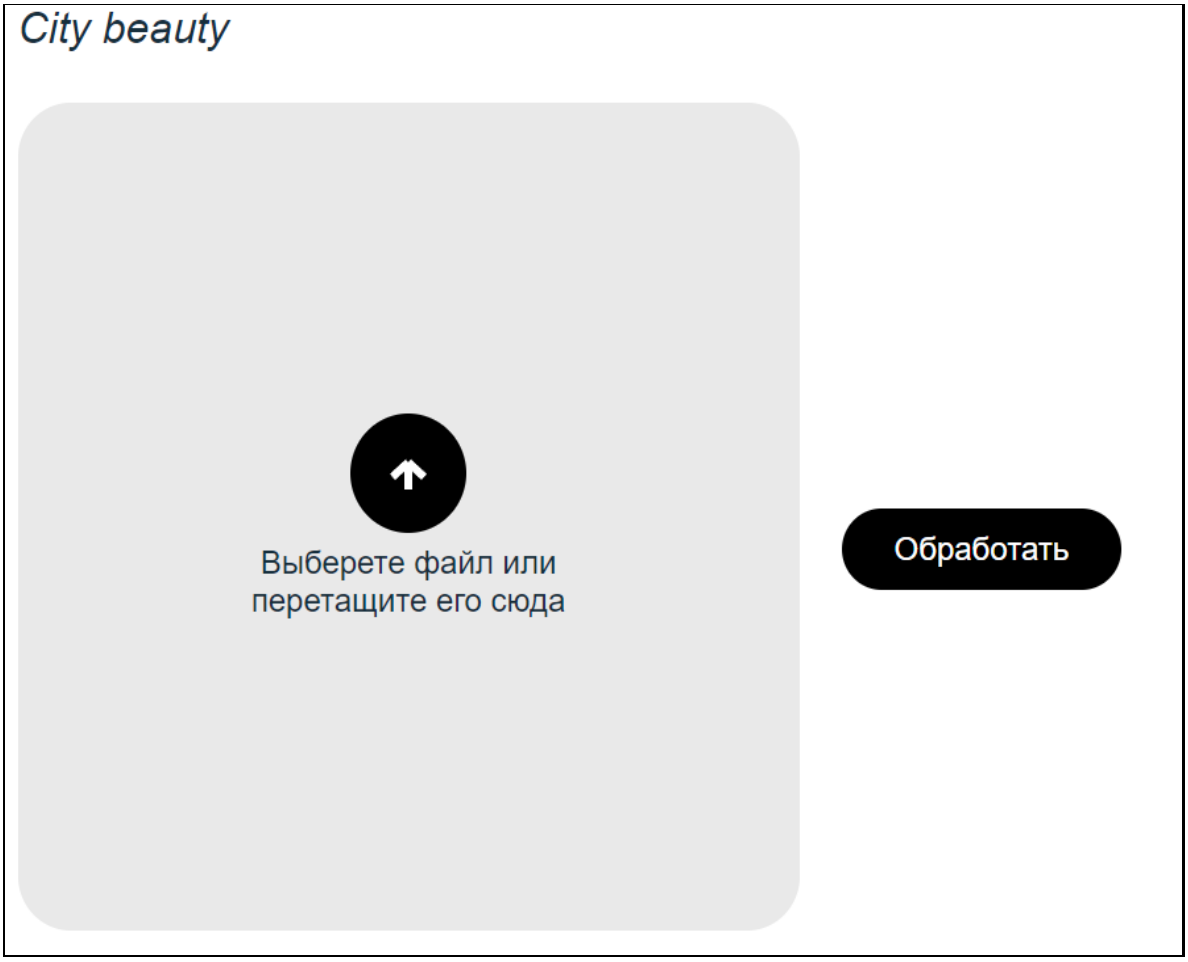

Рисунок 22 – Форма загрузки изображений

2. Галерея обработанных изображений. Представляет собой список изображений, прошедших обработку. На рисунке 23 изображен этот компонент.

Рисунок 23 – Компонент галереи

3. Форма обработанного изображения. После завершения обработки на сервере появляется эта форма, на ней можно скачать новое изображение

после обработки, а также вернуться в форму для загрузки изображений после нажатия на кнопку. На рисунках 24 и 25 изображен этот компонент.

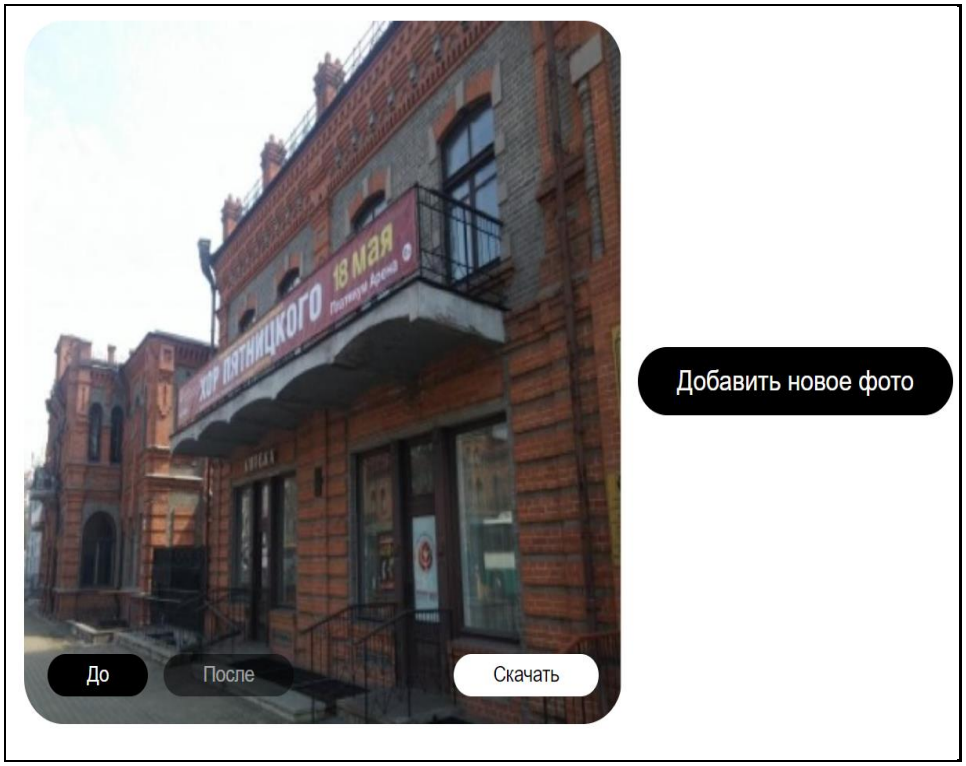

Рисунок 24 – Форма обработанного изображения до обработки

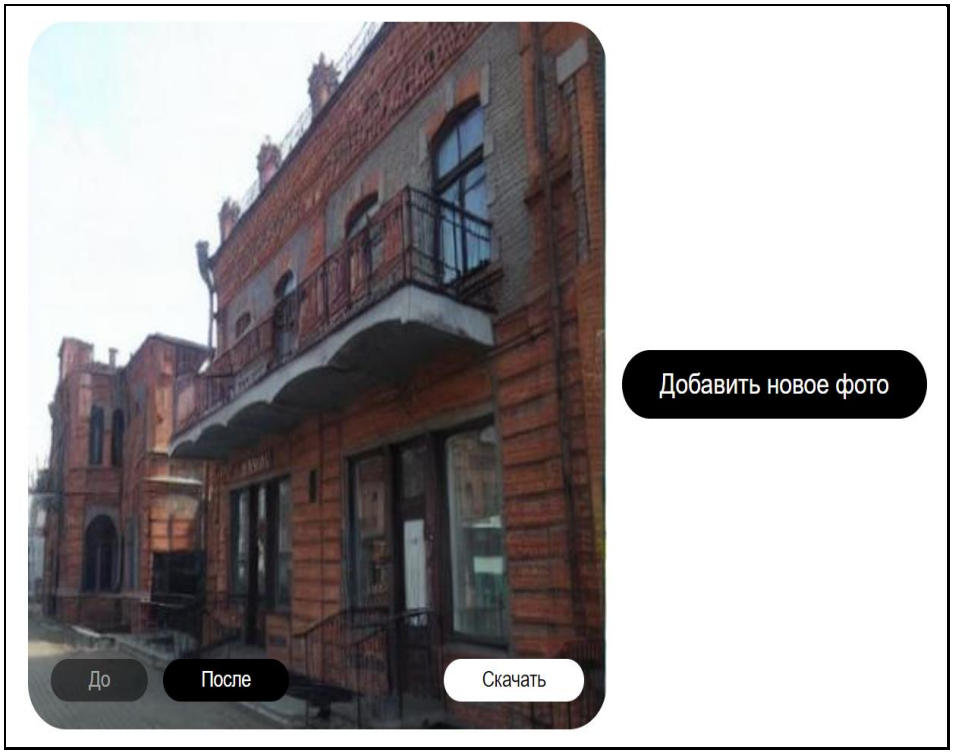

Рисунок 25 – Форма изображения после обработки

Также появляется всплывающее окно после отправки запроса на обработку изображения. Оно существует до тех пор, пока frontend не получит со стороны сервера статус об обработанном изображении. На рисунке 26 показано это окно. Код клиентской части представлен в листинге 4 приложения Б.

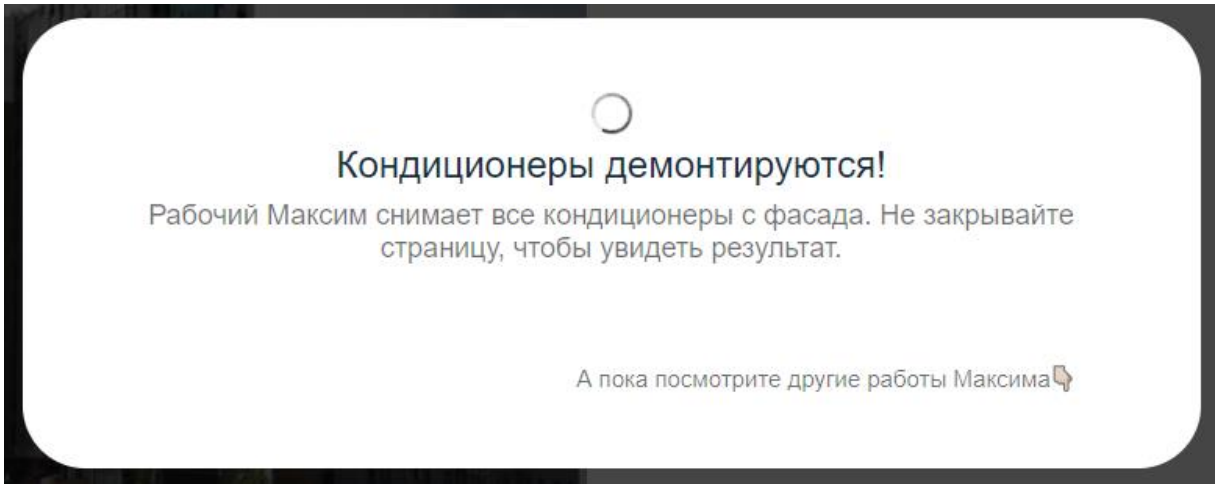

Рисунок 26 – Всплывающее окно ожидания обработки

4.7. Разработка серверной части приложения

Сервис api предоставляет для клиентской части конечную точку для загрузки изображений в файловое хранилище S3. Также предоставляет конечную точку по получению списка ссылок изображений для галереи. После получения нового изображения из frontend сервис отправляет новое сообщение с названием изображения и номера сессии в очереди ImageHandling и в HandlingTasks. Код сервиса api, представлен в листинге 5 приложения В.

Сервис ResultsApi предоставляет информацию для клиентской части о статусе обработки изображения. У него одна конечная точка, которая возвращает статус обработки. Таже он слушает HandlingTasks, если в ней появляется сообщение, он записывает в базу данных строку о новом изображении с статусом – ожидает обработку. Очередь HandlingResults сервис прослушивает на наличие сообщения о завершении обработки изображения, когда появляется сообщение об окончании обработки, сервис меняет статус

строки нужного изображения на статус – завершен. Код сервиса представлен в листинге 6 приложения Г.

Сервис ModelApi не предоставляет никаких конечных точек. Его задача получить маску и обработать поступившее изображение, и уведомить о результатах обработки в соответствующую очередь. Он слушает очередь ImageHandling на наличие новых изображений для обработки. После появления новой задачи, он сначала загружает изображение из S3. Потом сегментирует полученное изображение и по выделенной маске и исходному изображению генерирует новое изображение. Результат загружается в хранилище S3, и отправляется сообщение об окончании обработки исходного изображения в очередь HandlingResults. Код сервиса представлен в листинге 7 приложения Д.

Вывод по четвертой главе

В данной главе успешно разработана инфраструктура, способная поддерживать развертывание приложения. Кроме того, сервисы приложения были реализованы в соответствии с предъявленными требованиями, гарантируя выполнение своих функций. Особое внимание было уделено автоматизации процесса развертывания приложения. Для этого были разработаны пайплайны, позволяющие автоматически разворачивать приложение в кластере. Этот подход упрощает и ускоряет процесс развертывания, а также обеспечивает гибкость в случае необходимости миграции приложения в другие среды, поддерживающие k8s в качестве сервиса.

5. ТЕСТИРОВАНИЕ

Функциональное тестирование выполняется для подтверждения соответствия системы заявленным функциональным требованиям, которые были подробно изложены в третьей главе данной работы. Для оценки выполнения функциональных аспектов системы был создан специализированный набор тестов. Результаты тестирования представлены в таблице 1, что позволяет наглядно оценить соответствие разработанной системы предъявленным к ней требованиям.

Таблица 1 – Функциональное тестирование

Название теста	**Шаги**	**Ожидаемый результат**	**Тест пройден?**
Заполнил форму для отправки изображения.	1. Перейти на страницу приложения. 2. Заполнить форму отправки изображения. 3. Получить сгенерированное изображение	Отобразиться новый компонент с предыдущим изображением и сгенерированным.	Да
Загрузить изображения различных форматов.	1. Перейти на страницу приложения. 2. Заполнить форму отправки изображения, загружая изображения разных форматов.	Изображение успешно обработается и появится компонент с обработанным изображением.	Да
Скачивание изображения.	1. Дождаться обработки изображения 2. Нажать на кнопку скачивания изображения.	Скачивается обработанное изображение.	Да
Загрузить изображение не поддерживаемого формата	1. Перейти на страницу приложения. 2. Заполнить форму отправки изображения, загружая изображение не поддерживаемого формата.	Пользователь получит сообщение о том, что такой формат изображений не поддерживается	Да

Окончание таблицы 1

Произошла ошибка на стороне сервера	1. Ожидание обработки изображения 2. Обработка прервалась 3. Получить сообщение об ошибке	Пользователь получит сообщение о том, что произошла ошибка на сервере и текущая обработка прервана	Да
Загрузка изображения размером больше 100 мегабайт	1. Заполнить форму обработки изображения 2. Отправить изображение на обработку 3. Получить сообщение о том, что изображение превышает размер в 100 мегабайт	Пользователь получит сообщение о том, что изображение превышает допустимый размер для обработки	Да

Вывод по пятой главе

В данной главе подробно описаны результаты тестирования приложения, которые подтверждают его соответствие заявленным требованиям и способность предоставить необходимый функционал пользователям. Результаты тестирования подтвердили успешную реализацию всех функциональных требований, при этом система работает стабильно и предоставляет ожидаемый функционал. Кроме того, все сервисы приложения проявили высокую стабильность и способность взаимодействовать друг с другом без перебоев, что гарантирует бесперебойную работу системы в целом. Результаты тестирования уверенно подтверждают готовность приложения к запуску и использованию конечным пользователем.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано приложение для генерации изображений на основе предоставленной маски с использованием диффузионных нейронных сетей. По ходу выполнения работы были выполнены следующие задачи.

1. Произведен обзор литературы и существующих приложений по предметной области.
2. Выполнено проектирование архитектуры приложения.
3. Разработана инфраструктура для приложения.
4. Выбрана модель диффузионной нейронной сети.
5. Разработано приложение для генерации изображений с использованием диффузионных нейронных сетей.
6. Проведено тестирование приложения.

Это приложение может помочь дизайнерам в разработке новых макетов дизайна городского ландшафта, ускорить их работу.

Также остаются задачи, которые улучшат приложение.

1. Разработать логику аутентификации пользователя.
2. Разработать персональную галерею для каждого пользователя.
3. Ускорить время ответа сервиса обработки изображений.
4. Добавить возможность пользователю вводить текстовую информацию для модели генерации.

ЛИТЕРАТУРА

1. Docker. [Электронный ресурс] URL: https://aws.amazon.com/ru/docker/ (дата обращения: 05.05.2024 г.).
2. Docker container. [Электронный ресурс] URL: https://www.oracle.com/cis/cloud/cloud-native/container-registry/what-is-docker/ (дата обращения: 05.05.2024 г.).
3. Helm–chart. [Электронный ресурс] URL: https://helm.sh/ru/docs/intro/using_helm/ (дата обращения: 05.05.2024 г.).
4. K8s. [Электронный ресурс] URL: https://kubernetes.io/ru/ (дата обращения: 05.05.2024 г.).
5. Автокодировщик. [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Автокодировщик (дата обращения: 05.05.2024 г.).
6. Балансировщик нагрузки. [Электронный ресурс] URL: https://selectel.ru/blog/load-balancer-review/ (дата обращения: 05.05.2024 г.).
7. Пайплайн. [Электронный ресурс] URL: https://yandex.cloud/ru/blog/posts/2023/04/ci-cd-serverless (дата обращения: 05.05.2024 г.).
8. K8s Pod. [Электронный ресурс] URL: https://kubernetes.io/ru/docs/tutorials/kubernetes-basics/explore/explore-intro/ (дата обращения: 05.05.2024 г.).
9. Трансформер. [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Трансформер (дата обращения: 05.05.2024 г.).
10. Узел. [Электронный ресурс] URL: https://kubernetes.io/ru/docs/concepts/architecture/nodes/ (дата обращения: 05.05.2024 г.).
11. Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y. // Journal of Design Sciences and Applied Arts, 2014. – 37–50 pp.

12. Song Y., Ermon S. Improved Techniques for Training Score-Based Generative Models. // Generative model appliance, 2014. – 12–26 pp.

13. Hanna D. The Use of Artificial Intelligence Art Generator «Midjourney» in Artistic and Advertising Creativity. // Journal of Design Sciences and Applied Arts, 2023. – 42–58 pp.

14. Vayadande K., Bhemde S. AI-Based Image Generator Web Application using OpenAI's DALL-E System. // International Conference on Recent Advances in Science and Engineering Technology (ICRASET), 2023 – 6–15 pp.

15. Radford A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. // International Conference on Learning Representations, 2016. – 24–40 pp.

16. Wang T., Liu M., Tao A., Kautz J., Catanzaro B. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. // Computer Vision and Pattern Recognition Conference, 2018. – 12–26 pp.

17. Huang R., Zhang S., He R. Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis. // International Conference on Computer Vision, 2017. – 40–51 pp.

18. Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks. // Computer Vision and Pattern Recognition Conference, 2019. – 32–50 pp.

19. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. // Computer Vision and Pattern Recognition Conference, 2020. – 12–20 pp.

20. Rombach R., Blattmann A., Lorenz D., Esser P., Ommer B. High-Resolution Image Synthesis with Latent Diffusion Models. // Computer Vision and Pattern Recognition, 2022. – 11–25 pp.

21. CelebA-HQ. [Электронный ресурс] URL: https://paperswithcode.com/dataset/celeba-hq (дата обращения: 20.02.2024 г.).

22. FFHQ. [Электронный ресурс] URL: https://paperswithcode.com/dataset/ffhq (дата обращения: 20.02.2024 г.).

23. LSUN-Churches. [Электронный ресурс] URL: https://paperswithcode.com/sota/image-generation-on-lsun-churches-256-x-256 (дата обращения: 20.02.2024 г.).

24. Bedrooms. [Электронный ресурс] URL: https://paperswithcode.com/dataset/lsun (дата обращения: 20.02.2024 г.).

25. Minio S3. [Электронный ресурс] URL: https://min.io/ (дата обращения: 20.02.2024 г.).

26. RabbitMQ. [Электронный ресурс] URL: https://www.rabbitmq.com/ (дата обращения: 20.02.2024 г.).

27. Poetry. [Электронный ресурс] URL: https://python-poetry.org/ (дата обращения: 20.02.2024 г.).

28. .NET. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/dotnet/core/whats-new/dotnet-8/overview (дата обращения: 20.02.2024 г.).

29. PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org/ (дата обращения: 20.02.2024 г.).

30. Nginx. [Электронный ресурс] URL: https://nginx.org/ru/ (дата обращения: 20.02.2024 г.).

31. Kubectl. [Электронный ресурс] URL: https://kubernetes.io/ru/docs/reference/kubectl/ (дата обращения: 20.02.2024 г.).

32. Helm. [Электронный ресурс] URL: https://helm.sh/ru/ (дата обращения: 20.02.2024 г.).

33. PyTorch. [Электронный ресурс] URL: https://pytorch.org/ (дата обращения: 20.02.2024 г.).

34. Torchvision. [Электронный ресурс] URL: https://pytorch.org/vision/stable/index.html (дата обращения: 20.02.2024 г.).

35. Pillow. [Электронный ресурс] URL: https://pypi.org/project/pillow/ (дата обращения: 20.02.2024 г.).

36. Numpy. [Электронный ресурс] URL: https://numpy.org/doc/stable/ (дата обращения: 20.02.2024 г.).

37. Pillow. [Электронный ресурс] URL: https://pypi.org/project/pillow/ (дата обращения: 20.02.2024 г.).

38. Albumentations. [Электронный ресурс] URL: https://albumentations.ai/ (дата обращения: 20.02.2024 г.).

39. Scikit-learn. [Электронный ресурс] URL: https://scikit-learn.org/ (дата обращения: 20.02.2024 г.).

40. Pandas. [Электронный ресурс] URL: https://pandas.pydata.org/ (дата обращения: 20.02.2024 г.).

41. PSPNet. [Электронный ресурс] URL: https://paperswithcode.com/method/pspnet (дата обращения: 20.02.2024 г.).

42. Lei L., Gai K., Jing Y., Liehuang Z., DiffuseTrace: A Transparent and Flexible Watermarking Scheme for Latent Diffusion Model. // Computer Vision and Pattern Recognition Conference, 2023. – 13–20 pp.

43. U-Net. [Электронный ресурс] URL: https://paperswithcode.com/method/u-net (дата обращения: 20.02.2024 г.).

44. ResNet. [Электронный ресурс] URL: https://paperswithcode.com/method/resnet (дата обращения: 20.02.2024 г.).

45. CLIP ViT-L/14. [Электронный ресурс] URL: https://paperswithcode.com/paper/learning-transferable-visual-models-from/review/ (дата обращения: 20.02.2024 г.).

46. DBeaver. [Электронный ресурс] URL: https://dbeaver.io/ (дата обращения: 20.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Код реализации инфраструктуры

Листинг 1 – Код nginx.conf

```
stream {
    upstream k8s_servers {
        server <master-server-ip>:6443;
    }
    server {
        listen <openned-port-for-kube-api>;
        proxy_pass k3s_servers;
    }

    upstream k8s_http {
        server <master-server-ip>:<ingress-port-for-http>;
        server <slave-server-ip-1>:<ingress-port-for-http>;
        server <slave-server-ip-2>:<ingress-port-for-http>;
    }
    server {
        listen <openned-port-for-http>;
        proxy_pass k8s_http;
    }

    upstream k8s_https {
        server <master-server-ip>:<ingress-port-for-https>;
        server <slave-server-ip-1>:<ingress-port-for-https>;
        server <slave-server-ip-2>:<ingress-port-for-https>;
    }
    server {
        listen <openned-port-for-https>;
        proxy_pass k8s_https;
    }

    upstream rabbitmq {
        server <rabbitmq-server>:<rabbit-ui-port>;
    }
    server {
        listen <openned-port-for-rabbit-ui>;
        proxy_pass rabbitmq;
    }

    upstream rabbitmq_api {
        server <rabbitmq-server>:<rabbit-api-port>;
    }
    server {
        listen <openned-port-for-rabbit-api>;
        proxy_pass rabbitmq_api;
    }

    upstream postgre_api {
        server <master-server-ip>:<nodeport-port-for-postgre>;
        server <slave-server-ip-1>:<ingress-port-for-postgre>;
        server <slave-server-ip-2>:<ingress-port-for-postgre>;
    }
    server {
        listen <openned-port-for-postgre>;
        proxy_pass postgre_api;
    }
}
```

Листинг 2 – Код values-ingress.yaml

```
## nginx configuration
## Ref: https://github.com/kubernetes/ingress-nginx/blob/main/docs/user-
guide/nginx-configuration/index.md
##
## Overrides for generated resource names
# See templates/_helpers.tpl
# nameOverride:
# fullnameOverride:
## Labels to apply to all resources
##
commonLabels: {}
controller:
  name: controller
  image:
    ## Keep false as default for now!
    chroot: false
    registry: registry.k8s.io
    image: ingress-nginx/controller
    ## for backwards compatibility consider setting the full image url
via the repository value below
    ## use *either* current default registry/image or repository format
or installing chart by providing the values.yaml will fail
    ## repository:
    tag: "v1.5.1"
    digest:
sha256:4ba73c697770664c1e00e9f968de14e08f606ff961c76e5d7033a4a9c593c629
    digestChroot:
sha256:c1c091b88a6c936a83bd7b098662760a87868d12452529bad0d178fb36147345

  config:
    use-proxy-protocol: "true"
    real-ip-header: proxy_protocol
    # here use ip range of local subnet area
    set-real-ip-from: 10.0.2.0/24
  service:
    enabled: true
    # -- List of IP families (e.g. IPv4, IPv6) assigned to the service.
This field is usually assigned automatically
    # based on cluster configuration and the ipFamilyPolicy field.
    ## Ref: https://kubernetes.io/docs/concepts/services-networking/dual-
stack/
    ipFamilies:
      - IPv4
    ports:
      http: 80
      https: 443
    targetPorts:
      http: http
      https: https
    type: NodePort
    nodePorts:
      http: "30080"
      https: "30443"
      tcp: {}
      udp: {}
    external:
      enabled: true
```

Листинг 3 – Код gitlab-ci.yaml

```
stages:
  - build
  - deploy

default:
  tags:
    - mux-runner-gpu

variables:
  DOCKER_BUILDKIT: 1
  IMAGE_BASE_TAG: $CI_PRIVATE_REGISTRY/city-beauty
  BASE_CACHE_TAG: $IMAGE_BASE_TAG:deps-cache-$CI_COMMIT_REF_SLUG
  BUILD_CACHE_TAG: $IMAGE_BASE_TAG:build-cache-$CI_COMMIT_REF_SLUG
  PUBLISH_CACHE_TAG: $IMAGE_BASE_TAG:publish-cache-$CI_COMMIT_REF_SLUG
  FINAL_CACHE_TAG: $IMAGE_BASE_TAG:api-cache-$CI_COMMIT_REF_SLUG

.docker-build:
  before_script:
    - docker login -u $CI_PRIVATE_REGISTRY_USER  -p $CI_PRIVATE_REGIS-
TRY_PASSWORD $CI_PRIVATE_REGISTRY

build:
  stage: build
  extends: .docker-build
  script:
    - docker build --pull --build-arg BUILDKIT_INLINE_CACHE=1 --cache-
from $BASE_CACHE_TAG -f API/Dockerfile --target base -t $BASE_CACHE_TAG .
    - docker push $BASE_CACHE_TAG
    - docker build --pull --build-arg BUILDKIT_INLINE_CACHE=1 --cache-
from $BASE_CACHE_TAG --cache-from $BUILD_CACHE_TAG -f API/Dockerfile --
target build -t $BUILD_CACHE_TAG .
    - docker push $BUILD_CACHE_TAG
    - docker build --pull --build-arg BUILDKIT_INLINE_CACHE=1 --cache-
from $BASE_CACHE_TAG --cache-from $BUILD_CACHE_TAG --cache-from $PUB-
LISH_CACHE_TAG -f API/Dockerfile --target publish -t $PUBLISH_CACHE_TAG .
    - docker push $PUBLISH_CACHE_TAG
    - docker build --pull --build-arg BUILDKIT_INLINE_CACHE=1 --cache-
from $BASE_CACHE_TAG --cache-from $BUILD_CACHE_TAG --cache-from $PUB-
LISH_CACHE_TAG --cache-from $FINAL_CACHE_TAG -f API/Dockerfile --target
final -t $FINAL_CACHE_TAG .
    - docker push $FINAL_CACHE_TAG
    - docker push $FINAL_CACHE_TAG:${CI_COMMIT_SHORT_SHA}

  rules:
    - if: $CI_COMMIT_BRANCH == "develop"

deploy:
  stage: deploy
  image: lwolf/helm-kubectl-docker:v1.21.1-v3.6.0
  dependencies:
    - build
  variables:
    KUBECONFIG: /etc/deploy/config
  environment:
    name: Development
  before_script:
    - mkdir -p /etc/deploy
    - echo ${KUBE_CONFIG} | base64 -d > ${KUBECONFIG}
  script:
```

```
    - helm version
    - RELEASE_NAME=api
    - helm repo add bitnami https://charts.bitnami.com/bitnami
    - helm upgrade --install --namespace city-beauty --create-namespace -
-kubeconfig ${KUBECONFIG}
      --values ./API/ci/values-custom.yaml
      --set "image.tag=${CI_COMMIT_SHORT_SHA}"
      --set "ingress.hostname=${HOST}"
      --set "image.registry=${FINAL_CACHE_TAG}"
      --set "image.pullSecrets={${RELEASE_NAME}-aspnet-core-docker-se-
cret}"
      --set "image.pullCredentials.registry=${FINAL_CACHE_TAG}"
      --set "image.pullCredentials.username=${CI_PRIVATE_REGISTRY_USER}"
      --set "image.pullCredentials.password=${CI_PRIVATE_REGISTRY_PASS-
WORD}"
      --set "resources.limits.cpu=${CPU_LIMIT}"
      --set "resources.limits.memory=${MEMORY_LIMIT}"
      --set "resources.requests.cpu=${CPU_REQUEST}"
      --set "resources.requests.memory=${MEMORY_REQUEST}"
      --set "ingress.enabled=true"
      --set "extraConfigMapEnvVars.ASPNETCORE_ENVIRONMENT=Development"
      --set "extraSecretEnvVars.RabbitOptions__HostName=${RAB-
BIT_HOST_NAME}"
      --set "extraSecretEnvVars.RabbitOptions__VirtualHost=${RAB-
BIT_VIRT_HOST}"
      --set "extraSecretEnvVars.RabbitOptions__UserName=${RAB-
BIT_USER_NAME}"
      --set "extraSecretEnvVars.RabbitOptions__Password=${RABBIT_PASS-
WORD}"
      --set "extraSecretEnvVars.RabbitOptions__Port=${RABBIT_PORT}"
      --set "extraSecretEnvVars.S3Options__Endpoint=${S3_ENDPOINT}"
      --set "extraSecretEnvVars.S3Options__AccessKey=${S3_ACCESS_KEY}"
      --set "extraSecretEnvVars.S3Options__SecretKey=${S3_SECRET_KEY}"
      --set "extraSecretEnvVars.S3Options__BucketName=${S3_BUCKET_NAME}"
      "${RELEASE_NAME}"
      bitnami/aspnet-core --version 4.4.7
  rules:
    - if: $CI_COMMIT_BRANCH == "develop"
```

Приложение Б. Код реализации пользовательского интерфейса

Листинг 4 – Реализация пользовательского интерфейса

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import ProcessedPhoto from './components/BuildingElement/ProcessedPhoto';
import { getLocalStorage, removeLocalStorage } from '../../common/utils';
import { observer } from 'mobx-react-lite';
import { useContext } from 'react';
import MainStateContext from '../../states/Main/Main.context';
import { LocalStoreg } from '../../common/const';

function ProcessedForm() {
    const mainState = useContext(MainStateContext);

  return (
    <div className='processed-form'>
        <div className='processed-form__photos-box'>
            <div className='processed-form__image'>
                <ProcessedPhoto
                    innerPhoto={`https://s3.citybeauty.muxhome-
lab.ru/citybeauty-public/${getLocalStorage('fileName')}`}
                    processedPhoto={`https://s3.citybeauty.muxhome-
lab.ru/citybeauty-public/${getLocalStorage('proccessedPhoto')}`}
                />
            </div>
            <div><button title='Добавить новое фото' type="button" class-
Name='header__add' onClick={() => addNewPhoto()}>Добавить новое
фото</button></div>
        </div>
    </div>
  )
    function addNewPhoto() {
        removeLocalStorage(LocalStoreg.SESSION_ID);
        removeLocalStorage(LocalStoreg.STATUS);
        removeLocalStorage(LocalStoreg.PROCCESSED_PHOTO);
        removeLocalStorage(LocalStoreg.FILE_NAME);
        removeLocalStorage(LocalStoreg.FILE_USER);

        mainState.setSessiondId('');
        mainState.setStatus('');
        mainState.setFileName('');
        mainState.newProccessedPhoto('');
        mainState.setFile('');
    }

}

export default observer(ProcessedForm);

import { observer } from "mobx-react-lite";

function Header() {
  return (
    <header className='container header'>
        <h1 className='header__title'>City beauty</h1>
    </header>
  )
}

export default observer(Header);

import { useEffect, useState } from 'react';
```

```
import { api } from '../../common/api';

function Gallery() {
  const [images, setImages] = useState<string[]>([]);

  useEffect(() => {
    getImagesLoad();
  }, []);
  return (
    <div className="gallery">
      <ul className="gallery__list">
        {images.map((image) => (
          <li key={image} className="gallery__item">
            <img className="gallery__img"
src={`https://s3.citybeauty.muxhomelab.ru/citybeauty-public/${image}`}
alt={`Фото ${image}`} />
          </li>
        ))}
      </ul>
    </div>
  )

  async function getImagesLoad() {
    const {data} = await api.get('gallery');
    setImages(data);
  }
}

export default Gallery

/* eslint-disable @typescript-eslint/no-explicit-any */
import  { ChangeEvent, DragEvent, useContext, useEffect } from 'react'
import { api } from '../../common/api';
import { v4 as uuidv4 } from 'uuid';
import { removeLocalStorage, setLocalStorage } from '../../common/utils';
import MainStateContext from '../../states/Main/Main.context';
import { observer } from 'mobx-react-lite';
import axios from 'axios';
import { LocalStoreg, StatusesPhoto } from '../../common/const';
import ArrowSvg from "../../assets/arrow-svg.svg?react";

function Form() {
    const mainState = useContext(MainStateContext);

    const handleFileChange = (event: ChangeEvent<HTMLInputElement>) => {
      const selectedFile = event.target.files && event.target.files[0];

      if (selectedFile) {
        mainState.setFile(selectedFile);
      }
    };

    const handleDragOver = (event: DragEvent<HTMLDivElement>) => {
      event.preventDefault();
      event.dataTransfer.dropEffect = 'copy';
    };

    const handleDrop = (event: DragEvent<HTMLDivElement>) => {
      event.preventDefault();

      const droppedFile = event.dataTransfer.files[0];
```

```
      if (droppedFile) {
        mainState.setFile(droppedFile)
      }
    };

    useEffect(() => {
      let interval: any;

      if (mainState.sessionId && mainState.fileName && mainState.status
!== StatusesPhoto.SUCCESS) {
        interval = setInterval(() => {
          getStatus();
        }, 5000);
      }

      return () => {
        clearInterval(interval);
      }
    }, [mainState.sessionId, mainState.status]);

  return (
    <div className='form' id='formID'>

      {mainState.fileName && mainState.sessionId && mainState.status !==
StatusesPhoto.SUCCESS && (<Modal />)}

    <div className="" onDragOver={handleDragOver} onDrop={handleDrop}>
      {!mainState.file && (<label htmlFor="fileInput" class-
Name="form__label">
        <div>
          <div className='form__svg'>
            <ArrowSvg />
          </div>
          <span className="form__label-text">Выберете файл или перетащите
его сюда</span>
        </div>
        </label>
      )}

      <input
        type="file"
        id="fileInput"
        accept="image/*"
        onChange={handleFileChange}
        style={{ display: 'none' }}
      />
      {mainState.file && <div id="preview" className='form__preview'>
        {mainState.file && (
          <img
            src={URL.createObjectURL(mainState.file)}
            alt="Preview"
          />
        )}
      </div>}
    </div>

    <div className='form__inner'>

        <div className='form__box-submit'>
            <button type='button' className='form__button' onClick={() =>
uploadPhotoAsync()}>Обработать</button>
```

```
        </div>

    </div>
  </div>
  )
  async function uploadPhotoAsync() {
    const sessionid = uuidv4();

    if (mainState.file) {
      const formData = new FormData();
      formData.append('file', mainState.file);

      try {
        const { data }= await api.post(`/s3/upload?sessionId=${ses-
sionid}`, formData);

        setLocalStorage(LocalStoreg.SESSION_ID, sessionid);
        setLocalStorage(LocalStoreg.FILE_NAME, data.buck-
etSourceImageName);

        mainState.setSessiondId(sessionid);
        mainState.setFileName(data.bucketSourceImageName);
        // setIs(true);
      } catch(e) {
        console.log('er', e)
      } finally {
        console.log('fi')
      }
    }
  }

  async function getStatus() {
    try {
      const { data } = await axios.post(`https://citybeauty.muxhome-
lab.ru/results-api/image-handling-results/status`, {
        sessionId : mainState.sessionId,
        bucketSourceImageName : mainState.fileName
      });

      setLocalStorage(LocalStoreg.STATUS, data.status);
      setLocalStorage(LocalStoreg.PROCCESSED_PHOTO, data.buck-
etHandledImageName);
      mainState.setStatus(data.status);
      mainState.newProccessedPhoto(data.bucketHandledImageName)

    } catch {
      removeLocalStorage(LocalStoreg.SESSION_ID);
      removeLocalStorage(LocalStoreg.FILE_NAME);
      removeLocalStorage(LocalStoreg.STATUS);
      removeLocalStorage(LocalStoreg.PROCCESSED_PHOTO);

      mainState.setFileName('');
      mainState.setSessiondId('');
```

Приложение В. Код реализации сервиса api

Листинг 5 – Реализация сервиса api

```
using API.Options;
using Microsoft.Extensions.Options;
using Minio;
using Minio.DataModel.Args;

namespace API.Services;

public class S3Service
{
    private readonly ILogger<S3Service> _logger;
    private readonly S3Options _s3Options;
    private readonly IMinioClient _s3Client;
    private readonly AppFileOptions _fileOptions;

    public S3Service(
        IOptions<S3Options> s3Options,
        ILogger<S3Service> logger,
        IMinioClient s3Client,
        IOptions<AppFileOptions> fileOptions)
    {
        _s3Options = s3Options.Value;
        _logger = logger;
        _s3Client = s3Client;
        _fileOptions = fileOptions.Value;
    }

    public async Task<string> UploadFileAsync(IList<IFormFile> formFiles)
    {
        ValidateNewFile(formFiles);
        var formFile = formFiles[0];
        var fileExtension = Path.GetExtension(formFile.FileName);
        var fileName = $"{Guid.NewGuid()}{fileExtension}";

        var putObjectArgs = new PutObjectArgs()
            .WithBucket(_s3Options.BucketName)
            .WithObject(fileName)
            .WithObjectSize(formFile.Length)
            .WithStreamData(formFile.OpenReadStream())
            .WithContentType("application/octet-stream");

        try
        {
            await _s3Client.PutObjectAsync(putObjectArgs).ConfigureA-
wait(false);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex.Message);
            throw new Exception("Cannot upload file to s3");
        }

        _logger.LogDebug($"The file {fileName} is uploaded success-
fully");
        return fileName;
    }

    public async Task RemoveFileAsync(string fileName)
    {
        await _s3Client.RemoveObjectAsync(new RemoveObjectArgs()
```

```
                .WithBucket(_s3Options.BucketName)
                .WithObject(fileName)
            );

        _logger.LogDebug($"Removed object {fileName}");
    }

    private void ValidateNewFile(IList<IFormFile> formFiles)
    {
        if (formFiles.Count != 1)
        {
            throw new ArgumentException($"Only one file should be send.
You sent {formFiles.Count} files");
        }

        var formFile = formFiles[0];

        if (formFile.Length <= 0)
        {
            throw new ArgumentException("File is empty");
        }

        var fileExtension = Path.GetExtension(formFile.FileName);

        if (!_fileOptions.AvailableExtensions.Contains(fileExtension))
        {
            throw new ArgumentException($"The file extension ['{fileEx-
tension}'] is not allowed");
        }
    }
}
namespace API.RabbitMq;

public class QueueOptions
{
    public string ImagesHandlingQueueName { get; set; }
    public string NewHandlingTasksQueueName { get; set; }
}
using API.RabbitMq;
using Infrastructure.RabbitMq;
using Microsoft.Extensions.Options;

namespace API.RabbitMQ;

public class NewHandlingTaskMessagePublisher
{
    private readonly RabbitMqProducer _rabbitMqProducer;
    private readonly QueueOptions _queueOptions;

    public NewHandlingTaskMessagePublisher(
        RabbitMqProducer rabbitMqProducer,
        IOptions<QueueOptions> queueOptions)
    {
        _rabbitMqProducer = rabbitMqProducer;
        _queueOptions = queueOptions.Value;
    }

    public async Task PublishAsync(string fileName, string sessionId)
```

Приложение Г. Код реализации сервиса получения результатов

Листинг 6 – Код реализации сервиса получения результат

```
namespace Domain;

public static class ImageHandlingStatuses
{
    public const string WaitingForHandling = "waiting for handing";
    public const string Handling = "handling";
    public const string SuccessHandling = "success handling";
    public const string HandlingError = "handling error";
}
namespace Domain;

public class ImageHandlingResult
{
    public string SessionId { get; set; }
    public string BucketSourceImageName { get; set; }
    public string Status { get; set; }
    public string? BucketHandledImageName { get; set; }
    public DateTime? HandlingFinishedAtUtc { get; set; }
    public DateTime? StartedHandlingAtUtc { get; set; }

    public string? Recommendations { get; set; }
}
using Domain;
using Microsoft.EntityFrameworkCore;

namespace Data;

//Use next command in Package Manager Console to update Dev env DB
//PM> $env:ASPNETCORE_ENVIRONMENT = 'Debug'; Update-Database
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions options)
        : base(options)
    {
    }

    public DbSet<ImageHandlingResult> HandlingResults { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.ApplyConfigurationsFromAssembly(GetType().Assembly);
    }

    public static void ConfigureContextOptions(DbContextOptionsBuilder
options, string connection)
    {
        options.EnableSensitiveDataLogging();
    }
}
using Domain;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data.Mappings;

public class ImageHandlingResultMapping : IEntityTypeConfiguration<Image-
HandlingResult>
{
```

```
    public void Configure(EntityTypeBuilder<ImageHandlingResult> builder)
    {
        builder.HasKey(x => new
                {
                    x.SessionId,
                    x.BucketSourceImageName,
                }
            );
    }
}
using Domain;
using Microsoft.EntityFrameworkCore;

namespace Data.Commands;

public class UpdateHandlingResultCommand
{
    private readonly AppDbContext _context;

    public UpdateHandlingResultCommand(AppDbContext context)
    {
        _context = context;
    }

    public async Task ExecuteAsync(
        string sessionId,
        string bucketSourceImageName,
        string? startedHandlingAtUtc,
        string? bucketHandledImageName,
        string? handlingFinishedAtUtc,
        string? recommendations)
    {
        var entry = await _context.HandlingResults.SingleOrDefaultAsync(x
=> x.SessionId == sessionId && x.BucketSourceImageName == buck-
etSourceImageName);

        // TODO: remove code duplicate later
        if (entry == null)
        {
            entry = new ImageHandlingResult
            {
                SessionId = sessionId,
                BucketSourceImageName = bucketSourceImageName,
            };
            Handle(entry, startedHandlingAtUtc, bucketHandledImageName,
handlingFinishedAtUtc, recommendations);

            await _context.AddAsync(entry);
            await _context.SaveChangesAsync();
            return;
        }

        Handle(entry, startedHandlingAtUtc, bucketHandledImageName, han-
dlingFinishedAtUtc, recommendations);
        _context.Update(entry);
        await _context.SaveChangesAsync();
    }

    private static void Handle(
        ImageHandlingResult entry, string? startedHandlingAtUtc,
```

Приложение Д. Код реализации сервиса обработки

Листинг 7 – Код сервиса обработки

```
import time
import cv2
from citybeauty.modules.fasad_analyzer import FasadAnalyzer
from citybeauty.modules.fasad_cleaner import FasadCleaner
from PIL import Image

class ImageHandler:
    def __init__(self):
        pass

    def handle(self, image_path):

        source_name = image_path.split(".")[0]
        source_name_handled = f"{source_name}-handled.jpg"

        image = cv2.imread(image_path)

        print("Image start analizing")
        try:
            fasad_analyzer = FasadAnalyzer()
            fasad_cleaner = FasadCleaner()
        except Exception as e:
            print(repr(e))

        _, mask = fasad_analyzer.get_analytics(image_path)

        cv2.imwrite(f"{source_name}-mask.jpg", mask)

        image = Image.open(image_path).resize((400, 400))
        mask = Image.open("mask.jpg").resize((400, 400))

        print("Image analized")

        print("Image start cleaning")

        try:
            cleaned_image = fasad_cleaner.get_cleaned_fasad(image, mask)
        except Exception as e:
            print(repr(e))

        print("Image cleaned")
        cleaned_image[0].save(source_name_handled)

        return source_name_handled
import numpy as np
import torch
import PIL
from diffusers import StableDiffusionInpaintPipeline


class FasadCleaner:
    def __init__(self):
        device = "cuda"
        model_path = "runwayml/stable-diffusion-inpainting"
        self.pipeline = StableDiffusionInpaintPipeline.from_pre-
trained(model_path, torch_dtype=torch.float16,).to(device)
        self.generator = torch.Generator(device=device).manual_seed(0) #
change the seed to get different results
```

```
    def get_cleaned_fasad(self, image, mask_image):
        prompt = "Complete the areas on the building facades where the
advertisements were cut out. "
        guidance_scale = 5
        cleaned_image = self.pipeline(
            prompt=prompt,
            image=image,
            mask_image=mask_image,
            height=image.height,
            width=image.width,
            guidance_scale=guidance_scale,
            generator=self.generator,
            num_images_per_prompt=1,
        ).images

        return cleaned_image

import requests
from PIL import Image
import torch
from transformers import OwlViTProcessor, OwlViTForObjectDetection
import matplotlib.pyplot as plt
from transformers.image_utils import ImageFeatureExtractionMixin
import matplotlib.pyplot as plt
import math
import numpy as np
import cv2
from collections import Counter
from sklearn.cluster import KMeans


def get_image(image_path,dsize=(224,224)):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image,
                         cv2.COLOR_BGR2RGB)
    image = cv2.resize(image,dsize)

    return image

def get_modified_img(image):
    modified_img = cv2.resize(image,
                              (600, 400),
                              interpolation = cv2.INTER_AREA)
    modified_img = modified_img.reshape(modified_img.shape[0] * modi-
fied_img.shape[1], 3)
    return modified_img


def get_base_colors(image_path, number_of_colours,):
    image = get_image(image_path)
    image=get_modified_img(image)

    clf = KMeans(n_clusters = number_of_colours)
    labels = clf.fit_predict(image)

    counts = Counter(labels)
    counts = dict(sorted(counts.items()))

    center_colours = clf.cluster_centers_
    ordered_colours = [center_colours[i] for i in counts.keys()]
```

```
    rgb_colors = [ordered_colours[i].astype(int) for i in counts.keys()]

    return rgb_colors


def save_image(image, image_path):
    cv2.imwrite(image_path, mask)


class FasadAnalyzer:
    def __init__(self):
        self.processor = OwlViTProcessor.from_pretrained("google/owlvit-
base-patch32")
        self.model = OwlViTForObjectDetection.from_pre-
trained("google/owlvit-base-patch32")

        self.rules = { 'Вывески': {
                            'signboard':'Вывески засоряет фасад',
                            'words':'Лучше избегать надписей',
                            'images':'Яркие изображения засоряют ',
                                    },
                       'Адресные таблички': {
                            'adress sign':'На данном здании имеются адрес-
ные таблички, лучше согласовать их с гостом',
                                    },
                     }

    def get_boxes(self, image_path, texts):
        image = Image.open(image_path)
        inputs = self.processor(text=texts, images=image, return_ten-
sors="pt")
        outputs = self.model(**inputs)
        target_sizes = torch.Tensor([image.size[::-1]])
        threshold = 0.1
        if texts == [["adress sign"]]:
             threshold = 0.06
        results = self.processor.post_process_object_detection(out-
puts=outputs, target_sizes=target_sizes, threshold=threshold)
        boxes = results[0]["boxes"]
        return boxes

    def drow_mask(self, image, mask, boxes):
        for box in boxes:
            x1, y1, x2, y2 = box.detach().numpy()
            roi_corners = np.array([[(x1, y1),(x1, y2),(x2, y2),(x2,
y1)]], dtype=np.int32)
            channel_count = image.shape[2]
            ignore_mask_color = (255,)*channel_count
            cv2.fillPoly(mask, roi_corners, ignore_mask_color)
        return mask

    def cleaner_analytics(self, analytics):


        return analytics

    def get_analytics(self, image_path):
        image = cv2.imread(image_path)
        mask = np.zeros(image.shape, dtype=np.uint8)
```