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
Руководитель ИП Шинкарев А.А.,
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 веб-сервиса для определения уникальности изображения на основе методов машинного обучения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1485.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ Е.В. Иванова

Автор работы,
студент группы КЭ-228
_____ М.В. Ядрышникова

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
студентке группы КЭ-228
Ядрышниковой Марии Викторовне,
обучающейся 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 веб-сервиса для определения уникальности изображения на основе методов машинного обучения.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**
3.1. What is Content Base Image Retrieval? [Электронный ресурс] URL: https://www.baeldung.com/cs/cbir-tbir (дата обращения: 29.02.2024 г.).
3.2. Сюй А. System Design. // СПб.: Питер, 2023. – 304 с.
3.3. Нейронные сети и компьютерное зрение – обучающий курс на платформе Stepik. [Электронный ресурс] URL: https://stepik.org/course/50352/syllabus (дата обращения: 29.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Рассмотреть понятие уникальности фотографии с точки зрения составляющих фотографии.
4.2. Определить алгоритмы компьютерного зрения, которые помогут в автоматическом определении выявленных составляющих уникальности.
4.3. Определить алгоритмы компьютерного зрения, которые помогут в автоматическом определении плагиата фотографии.

4.4. Разработать метрику оценки уникальности фотографии в пределах сервиса по найденным составляющим фотографий.
4.5. Провести проектирование веб-сервиса, который использует несколько алгоритмов компьютерного зрения для определения уникальности фотографии.
4.6. Реализовать веб-сервис, определяющий уникальность фотографии в пределах сервиса на основе ее составляющих, автоматически определяющихся с помощью алгоритмов машинного обучения.
4.7. Провести функциональное тестирование веб-сервиса.
5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. Е.В. Иванова

Задание принял к исполнению М.В. Ядрышникова

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Визуальный контент – это основа контент-маркетинга. Как правило 95% покупателей услуг и продуктов B2B признают, что рассматривают контент как заслуживающий доверия маркер при оценке бизнеса (Ziflow [1]). Таким образом визуальный контент становится важной составляющей ведения бизнеса. Как правило в данное время количество визуального контента стремительно растет, поэтому для привлечения внимания пользователя, а также для выделения своего продукта из большого потока требуется выделяться и производить уникальный контент.

Визуальная уникальность контента – это сложная, почти неуловимая характеристика, которая базируется на том, что контент явно отличается концептуально от всего остального присутствующего в сервисе. Нельзя сказать, что уникальность – это только отсутствие полного визуального плагиата. Уникальность также можно рассматривать с точки зрения концепции.

Человек за счет визуального восприятия с легкостью может определить уникальную фотографию среди небольшой выборки. Но при увеличении анализируемой выборки, ручной анализ становится практически невозможным. Исходя из этого, появляется потребность в создании инструмента, позволяющего автоматически оценивать уникальность визуального контента.

Наиболее прогрессирующими на данный момент инструментами в сфере визуального контента являются инструменты компьютерного зрения. Компьютерное зрение превосходит другие алгоритмы по различным параметрам (скорость, качество и т.д.). С помощью него можно быстро и эффективно решить задачу определения уникальности фотографии и помочь фотографам и создателям визуального контента определить ценность их продукта для потребителей.

Таким образом, сервис, который может автоматически на основе нейронных сетей определить уникальность фотографии по отношению к другим фотографиям, находящимся в сервисе, имеет высокую ценность.

Постановка задачи

Целью выпускной квалификационной работы является разработка веб-сервиса для определения уникальности изображения на основе методов машинного обучения. Для достижения поставленной цели необходимо решить следующие задачи:

1) рассмотреть понятие уникальности с точки зрения ее составляющих;

2) реализовать веб-сервис, определяющий уникальность фотографии на основе ее составляющих, определяющихся с помощью алгоритмов машинного обучения.

Структура и содержание работы

Работа состоит из введения, одной главы, заключения и списка литературы. Объем работы составляет 57 страниц, объем списка литературы – 21 источник.

В первой главе описываются архитектурные паттерны, которые влияют на проектирование веб-приложения, содержащего несколько алгоритмов машинного обучения.

Во второй главе обозревается понятие уникальности и составляющие фотографии, которые можно оценивать автоматически и учитывать в расчете уникальности.

В третьей главе описываются способы автоматического нахождения составляющих фотографии.

В четвертой главе описываются способы расчета уникальности.

В пятой главе описано проектирование реализуемого веб-сервиса.

В шестой главе описывается реализация веб-сервиса для расчета уникальности фотографии.

В приложении приведены листинги реализации веб-сервиса.

1. ОПИСАНИЕ АРХИТЕКТУРНЫХ ПАТТЕРНОВ

1.1. Многоуровневая архитектура

Многоуровневая архитектура представляет собой шаблон проектирования программного обеспечения, который широко применяется в современной разработке программного обеспечения для облегчения организации, разделения задач и удобства сопровождения приложений [2].

Как правило, многоуровневая архитектура состоит из нескольких уровней, каждый из которых имеет определенный набор обязанностей в контексте приложения. Эти уровни организованы в вертикальную иерархию, в которой верхние уровни зависят от нижних уровней в плане функциональности и услуг. Это известно как отношение зависимости.

В типичной слоистой структуре наблюдаются четыре первичных уровня.

1. Уровень представления: обеспечивает пользовательский интерфейс, отображая данные конечным пользователям и собирая их входные данные.

2. Уровень приложения: реализует и координирует логику обработки и взаимодействия, управляя потоком данных между другими уровнями. Этот уровень, также известный как уровень бизнес-логики, отвечает за управление операциями, правилами и рабочими процессами приложения.

3. Уровень доступа к данным: управляет взаимодействием и связью с системами хранения данных, такими как базы данных и внешние службы, абстрагируя способы получения, хранения и обновления данных от остальной части приложения.

4. Уровень данных: представляет собой хранилище и хранилище информации, включая базы данных, файловые системы и другие источники данных. Этот уровень отвечает за моделирование данных и определение схемы в приложении.

Преимущества принятия многоуровневой архитектуры в контексте разработки программного обеспечения включают в себя следующее.

1. Модульность: разделение задач позволяет разработчикам сосредоточиться на конкретных аспектах приложения, делая базу кода более понятной, организованной и управляемой. Эта модульность также способствует повторному использованию компонентов в различных приложениях.

2. Масштабируемость: многоуровневые архитектуры по своей сути поддерживают масштабируемость, поскольку каждый уровень можно независимо масштабировать и оптимизировать в соответствии с меняющимися требованиями и целевым повышением производительности.

3. Удобство сопровождения: благодаря изоляции функциональных областей и их соответствующих реализаций внутри приложения, многоуровневая архитектура обеспечивает лучшую ремонтопригодность и облегчает внесение изменений или обновлений в компоненты, не оказывая существенного влияния на всю систему.

4. Тестируемость: каждый уровень может быть протестирован независимо, что гарантирует надежность и надежность отдельных компонентов и всего приложения.

5. Функциональная совместимость: многоуровневые архитектуры облегчают интеграцию и связь между различными системами и сервисами.

Недостатки многоуровневой архитектуры включают в себя:

1) наличие уровней снижает производительность: этот шаблон не подходит для высокопроизводительных приложений: проходить несколько уровней архитектуры для выполнения бизнес-запроса – это неэффективно;

2) добавление уровней увеличивает стоимость разработки и усложняет систему.

Таким образом, использовать этот подход следует для небольших, простых приложений и веб-сайтов. Также этот шаблон хорошо подходит, если бюджет и время ограничены. Систему определения уникальности нельзя назвать простой, поскольку она потенциально содержит в себе

несколько нейронных сетей, инференс которых будет занимать немало времени. Замедление приложения с помощью слоев не улучшит ситуацию.

1.2. Клиент-сервер

Клиент-серверная архитектура – это модель организации вычислительных систем, в которой задачи распределены между клиентами и серверами.

В такой архитектуре клиент, обычно являющийся пользователем или программой, запрашивает услуги или ресурсы у сервера, который отвечает на запросы, предоставляя необходимые данные или функциональность.

В большинстве случаев минусы клиент-серверной архитектуры связаны с работоспособностью сервера или базы данных. Решение одних недостатков данной архитектуры приводит к другим, чаще всего к росту стоимости разработки и поддержки.

Преимущества шаблона проектирования «клиент-сервер» описаны в нижеследующих пунктах.

1. Отсутствие дублирования: весь сайт или приложение хранится на одном компьютере-сервере. Это позволяет использовать его с разных устройств, будь то компьютер или мобильный телефон.

2. Минимальные требования к пользователю: от него требуется только наличие программы-клиента. Для работы с сайтами достаточно иметь браузер.

3. Безопасность: данные хранятся в базе данных, и пользователи не могут просматривать данные, которые к ним не относятся, при должном качестве разработки. В таком сценарии данные клиента получаются с помощью запросов к серверной части.

4. Производительность: серверы обычно производительнее, чем компьютеры пользователей. Это позволяет обрабатывать тысячи запросов от сотни разных пользователей одновременно.

К недостаткам данного шаблона можно отнести следующие.

1. Перегрузка сервера: популярные порталы могут получать большое количество запросов одновременно. Например, при десятке миллионов запросов в секунду сервер может не выдержать и отключиться. Этим пользуются хакеры при использовании DDoS-атаки.

2. Выход из строя сервера или базы данных: это сделает сервис недоступным для всех пользователей.

3. Высокая стоимость оборудования: сервер похож на простой компьютер, но его комплектующие должны быть рассчитаны на бесперебойную круглосуточную работу. Такая надежность обеспечивается компонентами со специальными функциями, из-за чего стоимость оборудования повышается.

4. Затраты на поддержку: должен быть специалист, который будет обслуживать сервер и быстро реагировать в случае поломок.

Чтобы избавиться от большинства перечисленных недостатков, разработчики используют кластеры серверов.

Таким образом, для приложения, которое будет оценивать уникальность фотографии, будет сложно применить паттерн проектирования «клиент-сервер», поскольку основной минус в данной ситуации – развертывание приложения на одном сервере. Для приложения с несколькими нейронными сетями предпочтительнее использовать решение, которое можно разворачивать на нескольких серверах, чтобы использовать как можно больше мощностей.

1.3. Модель-представление-контроллер (MVC)

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller).

Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Среди компонентов MVC можно выделить следующие.

1. Модель – этот компонент отвечает за данные, а также определяет структуру приложения.

2. Представление – этот компонент отвечает за взаимодействие с пользователем. То есть код компонента представления определяет внешний вид приложения и способы его использования.

3. Контроллер – этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя.

Модель MVC решает следующие задачи:

1) изменение только пользовательского интерфейса, а не бизнес-логики приложения;

2) использование в одном приложении разных интерфейсов с возможностью выбора;

3) замена реакции приложения на действия пользователя за счет использования другого контроллера.

Среди недостатков паттерна MVC можно выделить следующие.

1. Необходимость использования большего количества ресурсов. Сложность обусловлена тем, что все три фундаментальных блока являются абсолютно независимыми и взаимодействуют между собой исключительно путем передачи данных. Controller должен всегда загрузить (и при необходимости создать) все возможные комбинации переменных и передать их в Model. Model, в свою очередь, должен загрузить все данные для визуализации и передать их во View. Например, в модульном подходе, модуль может напрямую обрабатывать переменные окружения и визуализировать данные без загрузки их в отдельные секции памяти.

2. Усложнен механизм разделения программы на модули. В концепции MVC наличие трех блоков (Model, View, Controller) прописано жестко. Соответственно каждый функциональный модуль должен состоять

из трех блоков, что в свою очередь, несколько усложняет архитектуру функциональных модулей программы.

3. Усложнен процесс расширения функционала. Проблема очень схожа с вышеописанной. Недостаточно просто написать функциональный модуль и подключить его в одном месте программы. Каждый функциональный модуль должен состоять из трех частей, и каждая из этих частей должна быть подключена в соответствующем блоке [3].

Основной минус данного шаблона в рамках приложения определения уникальности – усложнение процесса расширения функционала.

В данном контексте это значит, что при добавлении новой нейронной сети, а значит, и добавлении новой концептуально отличительной особенности фотографии, будет сложно это сделать.

1.4. Управляемая событиями архитектура

Событийно-ориентированная архитектура (EDA) – это парадигма программной архитектуры, способствующая порождению, обнаружению, потреблению событий и реакции на них [4].

Управляемые событиями архитектуры включают пять ключевых компонентов:

1) продюсеры (производитель) событий;
2) начальные события;
3) события обработки;
4) маршрутизаторы событий;
5) потребители событий.

Продюсер публикует начальное событие в маршрутизаторе, который фильтрует и передает событие обработки потребителям.

Преимущества данного паттерна:

1) сервисы продюсера и потребителя разделены, что позволяет масштабировать, обновлять и развертывать их независимо;

2) каждый потребитель событий может масштабироваться отдельно, что обеспечивает точную масштабируемость;

3) высокая производительность за счет асинхронных возможностей: возможность выполнять разделенные, параллельные асинхронные операции перевешивают затраты на постановку в очередь и удаление сообщений из очереди;

4) поскольку компоненты-потребители событий являются одноцелевыми и полностью отделены от других компонентов-потребителей событий, изменения, как правило, ограничиваются одним или несколькими потребителями событий и могут быть выполнены быстро, не затрагивая другие компоненты.

Среди недостатков архитектуры, управляемой событиями, можно выделить следующие.

1. Сложность из-за асинхронного характера шаблона, а также из-за создания контракта и необходимости более сложных условий обработки ошибок в коде для не отвечающих обработчиков событий и отказавших брокеров.

2. Индивидуальное юнит-тестирование не слишком сложно, но для генерации событий требуется какой-то специализированный клиент или инструмент тестирования. Тестирование также осложняется асинхронным характером этого шаблона.

Событийная архитектура использует события для запуска и обмена данными между разделенными службами и является обычным явлением в современных приложениях, созданных с использованием микросервисов [5].

Для сервиса, который определяет уникальность визуального контента на основе нейронных сетей, данный шаблон проектирования подходит больше всего за счет возможности легкого масштабирования и высокой производительности. Недостатки в данном случае не имеют такого веса, как преимущества.

Таким образом, в данной главе был произведен обзор основных шаблонов проектирования архитектуры сервисов. При этом был проведен анализ шаблона проектирования на возможность его использования для сервиса, который сможет определять уникальность визуального контента внутри сервиса.

Наиболее подходящим шаблоном для проектирования сервиса, определяющего уникальность визуального контента, является архитектура, управляемая событиями, поскольку в данном случае наличия нескольких нейронных сетей или алгоритмов машинного обучения гораздо лучшим решением является разбиение всего веб-приложения на независимые микросервисы, которые имеют возможность масштабироваться, могут разрабатываться независимо друг от друга и также с определенной легкостью добавляться в случае нахождения дополнительных составляющих уникальности. Например, после того, как будут проведены эксперименты по определению композиции фотографии и будет принято решение о добавлении новой модели определения композиции фотографии в сервис, это будет легко сделать, поскольку для этого будет необходимо реализовать независимый микросервис, который должен будет подписаться на определенное событие в системе и реагировать на него. Управление событиями также позволяет грамотно логировать приходящие изображения и использовать информацию о них в дальнейших целях анализа работы системы.

2. ПОНЯТИЕ УНИКАЛЬНОСТИ

2.1. Понятие уникальности

Прежде, чем переходить к способам автоматического расчета уникальности фотографии, необходимо дать само определение уникальности. В самых известных словарях русского языка, таких как словарь Ожегова и Шведовой, словаре Даля, словаре Ушакова, нет прямого определения для слова «уникальность», но есть определение для прилагательного слова «уникальный».

По толковому словарю С.И. Ожегова и Н.Ю. Шведовой, уникальный – это единственный в своем роде, неповторимый [6].

По толковому словарю Д.Ю. Ушакова, уникальный – это являющийся уникумом. В свою очередь уникум – это единственный в своем роде, редкий предмет [7].

Из этих определений можно сделать вывод, что уникальность – это единичность, редкость, неповторимость.

Идея этой работы заключается в том, что уникальность – это небинарный признак.

В настоящий момент принято считать, что уникальность может или быть, или не быть вовсе.

В этой работе выдвигается предположение о том, что уникальность может иметь шкалу, и разные фотографии могут иметь разное значение уникальности, не выходящее за пределы этой шкалы.

В качестве шкалы уникальности можно использовать процентную шкалу, при которой 0% означает абсолютно неуникальную фотографию, 100% – абсолютно уникальную.

В данной работе уникальность будет рассчитываться в пределах небольших групп фотографий, присутствующих в разрабатываемом сервисе, поскольку сравнивать фотографию со всем, что есть в сети Интернет – это почти нереальная задача. При этом уникальность фотографии рассчитывается один раз при загрузке изображения и больше не

пересчитывается. Таким образом, фотография, загруженная раньше, будет более уникальной, чем ее дубликат, загруженный после этого.

2.2. Составляющие уникальности

Опора в этой работе идет на интерпретируемость пользователем полученного результата определения уникальности. Польза интерпретируемости при применении машинного обучения состоит в том, что это повышает доверие пользователей к применяемой модели. Если сделать процесс расчета уникальности фотографии настолько прозрачным, насколько это возможно, пользователи приложения смогут объяснять для самих себя, что значит та или иная полученная цифра уникальности и по каким именно критериям уникальность была снижена. Помочь в повышении интерпретируемости определения уникальности может разделение фотографии на составляющие, определение этих составляющих по полученной фотографии и математическая модель расчета конечной уникальности, базирующаяся на полученных составляющих. При этом пользователю необходимо видеть как полученные составляющие, так и формулу, по которой итоговая цифра уникальности получилась.

Среди составляющих уникальности можно выделить следующие:

1) объекты, присутствующие на фотографии;
2) окружение, в котором объекты располагаются;
3) цветовая составляющая фотографии;
4) настроение, которое фотография передает;
5) ассоциации к предыдущим составляющим уникальности.

Все описанные составляющие можно определять автоматически с применением инструментов компьютерного зрения.

3. СПОСОБЫ АВТОМАТИЧЕСКОГО ОПРЕДЕЛЕНИЯ СОСТАВЛЯЮЩИХ УНИКАЛЬНОСТИ

3.1. Определение цветовой составляющей фотографии

Цветовая составляющая изображения является базовой информацией об изображении. Цвет определяет настроение изображения, его концепцию, а также может служить в качестве смыслового акцента.

В цифровом формате изображение представляется набором пикселей. Каждый пиксель описывает цвет в соответствии с цветовой моделью. Существует различные цветовые модели, при этом каждая использует свой подход к описанию цвета. Наиболее популярными цветовыми моделями в цифровом формате являются RGB и HSB, визуальные представления которых отображены на рисунке 1.

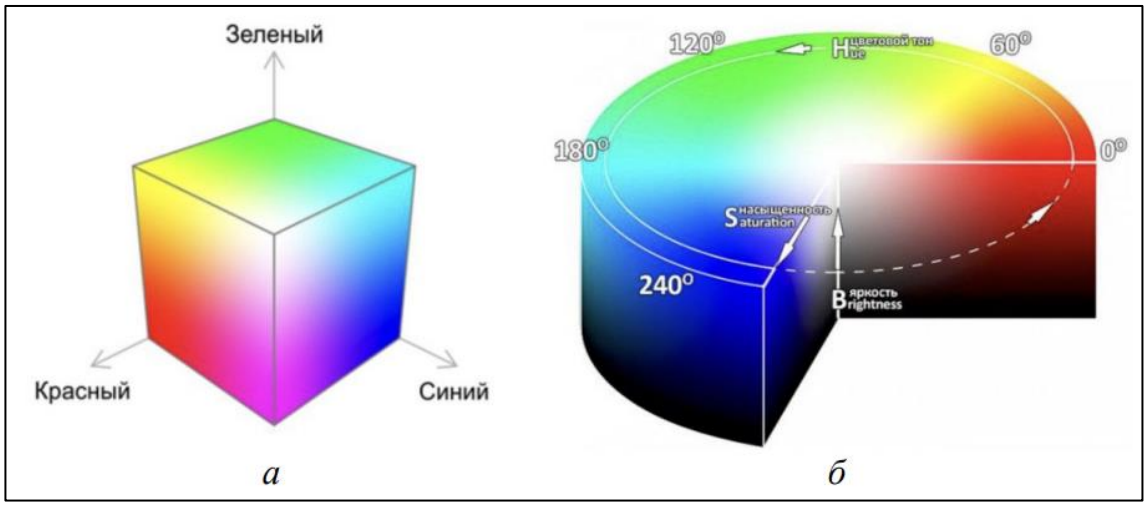

Рисунок 1 – Цветовые модели: а – RGB; б –HSB

Цветовая модель RGB (англ. red – красный, green – зеленый, blue – синий), изображенная на рисунке 1(а), [8] является самой популярной, и во многих областях используется как стандарт «дефакто». Значения каждого цвета варьируются в диапазоне от 0 до 255. При максимальном значении каждого цвета мы получим белый цвет. При разном уровне пропорций базовых цветов сможем получить любой другой цвет. Такое представление может отобразить приблизительно 16,7 млн оттенков.

Цветовая модель HSB (англ. hue – тон, saturation – насыщенность, brightness – яркость), изображенная на рисунке 1(б), основана на показателях цвета: цветовом тоне, насыщенности и яркости [6]. Цветовой тон имеет значение от 0 до 360 (по градусам, которые образуют верхнюю стенку цилиндра), и выбирается из видимого спектра излучения. Значения насыщенности и яркости варьируются от 0 до 100. Насыщенность соответствует интенсивности световой волны. Яркость задает освещенность световой волны. Такое представление может отобразить приблизительно 3,6 млн оттенков.

Дальнейшие описания методов будут проводиться на основе модели RGB, так как этот формат более простой для визуального представления.

Внимание человека может остановиться на цветах, которые занимают большой объем пространства на изображении, или на цветах, которые очень контрастны по отношению к остальным цветам изображения. Первые цвета называются доминирующими, вторые акцентными.

Кластеризация K-средних [9] позволяет выделять желаемое количество доминирующих цветов изображения. Для этого метод разделяет все пиксели изображения на группы и определяет значения центральных пикселей группы как доминирующие цвета изображения. Количество групп напрямую зависит от заранее установленного количества цветов.

Изначально центры групп определяются случайным образом. Все остальные пиксели изображения распределяются по группам на основе вычисленного евклидового расстояния по формуле (1):

$$D(F_1, F_2) = \sqrt{\sum_{i=1}^{n} (F_1(i) - F_2(i))}, \quad (1)$$

где D – расстояние между цветовыми векторами F_1, F_2;

n – длина вектора;

i – позиция значения в векторе.

Пиксели относятся к той группе, до центра которого меньше расстояние по сравнению с центрами всех остальных групп. Процесс определения

центра и распределения пикселей по группам происходит несколько раз и останавливается тогда, когда положение центра в сравнении с предыдущим положением изменяется незначительно.

Для сравнения нескольких наборов доминирующих цветов их нужно представить с помощью единой схемы записи. Для этого набор цветов можно представлять как последовательность определенной длины. Тогда этот набор можно отсортировать внутри вектора определенного цвета по каналам (при использовании цветовой модели RGB необходимо отсортировать значения по каналам цветовой модели), внутри вектора, содержащего несколько полученных цветов, по значениям каналов цветовой модели (сначала сортировка векторов по каналу R, затем сортировка по каналу G и в последнюю очередь сортировка по каналу B). В таком случае после сортировки цвета будут находиться в понятном порядке.

Таким образом, в математическом виде, цветовую составляющую можно представить так, как это описано в формуле (2):

$$F = \{c_1, c_2, \dots, c_n\}, \quad (2)$$

где F – вектор с базовыми цветами, c – вектор базового цвета, описанный в соответствии с каналами цветовой модели;

n – порядок вектора базового цвета в векторе F.

Набор доминирующих цветов имеет высокую интерпретируемость с точки зрения визуального восприятия человека, поскольку конкретный набор цветов можно отнести к определенному изображению.

Выделение доминирующих цветов имеет усредненный характер, поэтому результаты не всегда могут сопоставляться с ожиданиями человека. При небольшом количестве определяемых цветов некоторые акцентные цвета могут пропасть из палитры, поскольку используются в небольшом количестве пикселей.

Например, на рисунке 2 метод K-средних выделил только зеленые цвета, однако голубые глаза кота явно выделяются из общей картины. Голубой цвет глаз в данном случае можно назвать акцентным.

Рисунок 2 – Результат работы метода K-средних

Существуют и иные методы нахождения цветовой составляющей, которые были опробованы, но не будут описаны в данной работе. К таким методам относятся: метод цветовых гистограмм, квантование цветового пространства, нахождение когерентных векторов.

Метод K-средних был выбран в качестве основного метода в данном сервисе, поскольку именно он дал наиболее интерпретируемый результат для пользователя. Все дальнейшие способы сравнения цветовой составляющей будут базироваться именно на методе K-средних и представлении цвета в виде вектора.

Реализация метода K-средних в задаче поиска цветовой составляющей в сервисе определения уникальности изображения представлена в листинге 1 приложения.

3.2. Определение объектов и их окружения на фотографии

Распознавание объектов на фотографии – классическая задача в компьютерном зрении.

Существует несколько задач компьютерного зрения, нацеленных на определение объектов на фотографии. К таким задачам относятся задачи классификации, детекции и сегментации изображений.

При классификации изображений необходимо учитывать, что изображению полностью назначается определенная метка. При таком подходе предполагается, что для каждого изображения существует только один класс. Основной фактор, по которому было решено не использовать классификатор для данной задачи – возможность определения только одной метки для фотографии.

Задача детекции – определить, присутствуют ли объекты заданных классов, если присутствуют, то определить их конкретное положение [10]. Пример работы детектора представлен на рисунке 3.

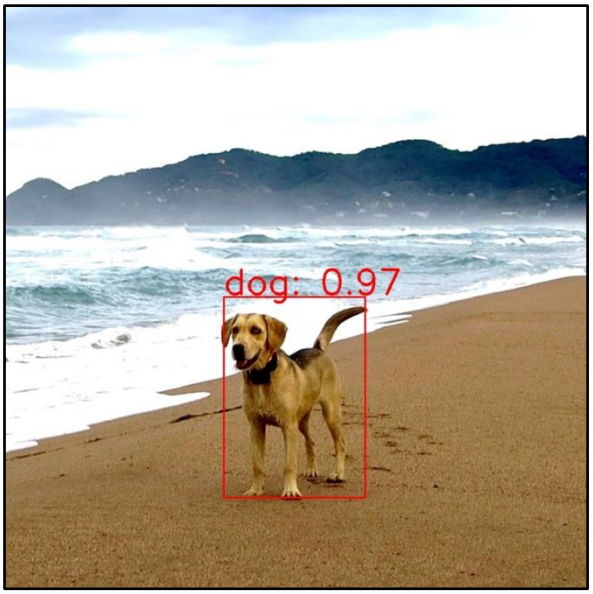

Рисунок 3 – Пример работы детектора

Среди плюсов подхода детекции в данной задаче можно выделить следующие.

1. Скорость работы. В сравнении с сегментаторами, детекторы в среднем работают быстрее. Эксперименты со скоростью работы детекторов и сегментаторов приведены в таблице 1.

Таблица 1 – Сравнение времени работы детекторов и сегментаторов

Тип нейронной сети	**Название архитектуры**	**Датасет, на котором была претренирована нейронная сеть**	**Среднее время работы по результатам экспериментов (в секундах)**
Детекторы	Yolov5	COCO	0,2–0,3
	SSD	COCO	0,3–0,4
	Faster-RCNN	COCO	5–6
Сегментаторы	DeepLabv3	ade20k	13–14
	ResNet50	ade20k	60–65
	ResNet101+ DeepLabv3	COCO	4–6
	ResNet101+FCN	COCO	4–6

2. Возможность фильтрации найденных объектов по уверенности распознавания. Для каждого распознанного объекта на фотографии детектор предоставляет коэффициент уверенности, который варьируется от 0 до 1. По этому коэффициенту можно установить фильтрацию и оставлять среди распознанных объектов только те, в которых детектор достаточно уверен. Примеры работы детекторов разных архитектур и их степени уверенности в результате можно увидеть на рисунке 4.

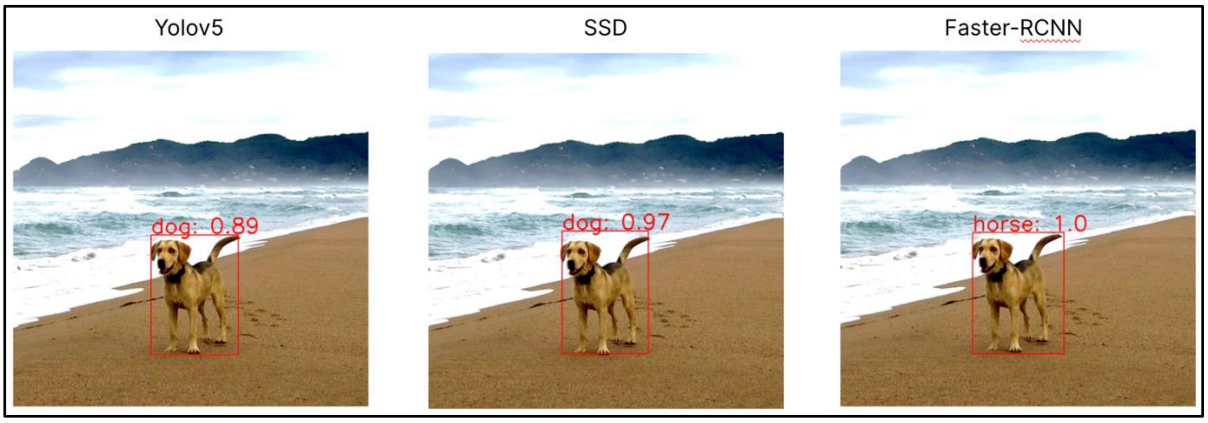

Рисунок 4 – Пример работы детекторов разных архитектур

К минусам использования детекторов в данной задаче можно отнести невозможность точного определения окружающих объекты элементов. К окружающим элементам можно отнести, например, траву и песок, небо и облака. Это все, что обычно окружает одушевленные предметы.

Обычно детекторы обучают, чтобы находить одушевленные предметы. Например, на рисунке 4, все детекторы распознали только одушевленный предмет, при этом два детектора из трех указали на то, что найденный объект – это собака, один детектор, что найденный объект – лошадь. Но в задаче поиска окружения одушевленных объектов может помочь сегментация.

Сегментация изображения – задача поиска групп пикселей, каждая из которых характеризует один смысловой объект [11].

При сегментации каждый пиксель изображения относится к определенной группе. Таким образом, каждый пиксель изображения будет проклассифицирован, что приводит к возможности определения всех одушевленных объектов изображения и их окружения. Среди минусов сегментации более длительное время работы в сравнении с детекторами. Среди плюсов использования сегментаторов: потенциально полезная информация о количестве занимаемого пространства каждым из найденных объектов. При этом к минусам сегментаторов можно отнести то, что найденные объекты никак нельзя отфильтровать по уверенности сегментатора в предсказании, поскольку каждый пиксель точно должен иметь класс. Пример работы сегментаторов разных архитектур можно увидеть на рисунке 5.

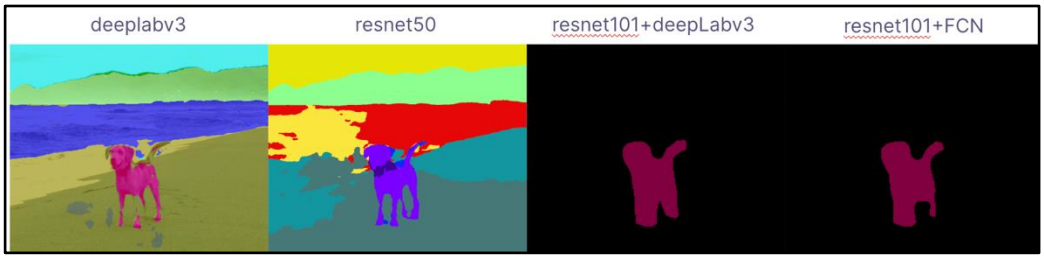

Рисунок 5 – Пример работы сегментаторов разных архитектур

В итоговой версии сервиса было принято решение использовать сегментатор SegFormer [12]. Время работы этого сегментатора на основе трансформеров составляет 0,3 секунды и по визуальной оценке, этот сегментатор

качественно справляется с оценкой объектов и их окружения на изображении.

Реализация работы модели определения объектов и их окружений в сервисе определения уникальности изображения представлена в листинге 2 приложения.

3.3. Определение эмоции изображения

В основе всех изображений лежит эмоция. Эмоция может прятаться в лице человека или животного, может передаваться с помощью цветов, задействованных в изображении, а может просто подразумеваться при использовании геометрических фигур или образов.

Распознавание эмоции также одна из ключевых задач в поиске составляющих изображения. Во время проведения экспериментов по определению основной эмоции изображения был проведен анализ самых ярких эмоций, которые изображения могут передавать. Все выделенные эмоции и критерии, по которым их можно определять, отображены в таблице 2.

Таблица 2 – Критерии разметки набора данных для классификации эмоций

Эмоция	**Критерии, по которым изображение относится к эмоции**
Счастье	1. Фотография имеет яркую и насыщенную цветовую гамма. 2. В случае людей и животных радостное выражение лица и улыбка.
Нежность	1. В цветовой гамме присутствуют розовые и оранжевые теплые оттенки. 2. На фото изображены объятия. 3. У фото романтический подтекст.
Спокойствие	1. Пастельная, приглушенная, светлая цветовая гамма. 2. Фотография не имеет ярко выраженных эмоций.
Меланхоличность	1. Фотография имеет серую или темную цветовую гамму. 2. Фотография вызывает эмоции плача, грусти или печали.
Испуг	1. Фотография несет в себе агрессивный подтекст. 2. Фото содержит распространенные фобии людей.

По выделенным эмоциям был произведен сбор датасета для проведения эксперимента по автоматическому определению эмоции изображения с использованием нейросети.

Среди категорий изображений, которые могут передавать эмоции и должны присутствовать в наборе для каждого класса, были выделены следующие:

1) люди;
2) животные;
3) предметы;
4) архитектура;
5) природа;
6) интерьер.

В итоговом собранном датасете было распределение изображений, отображенное в таблице 3, при этом для каждого датасета распределение категорий было приблизительно равным.

Таблица 3 – Распределение фотографий в наборе данных для распознавания эмоций

Тренировочный набор данных	**Валидационный набор данных**	**Тестовый набор данных**
2275	286	294

Первичные эксперименты по обучению нейронных сетей оказались безуспешными, но доказали, что технически автоматическое определение эмоции изображения возможно.

В одном из экспериментов использовалась архитектура Inception-Resnet v2, которая претренирована на датасете Imagenet и дообучена на собранном датасете. Обучение проходило на 300 эпохах. Валидационная метрика accuracy равна 0,61, валидационная ошибка (категориальная кроссэнтропия) равна 0,9742. График изменения ошибка представлен на рисунке 6, график изменения метрики accuracy представлен на рисунке 7.

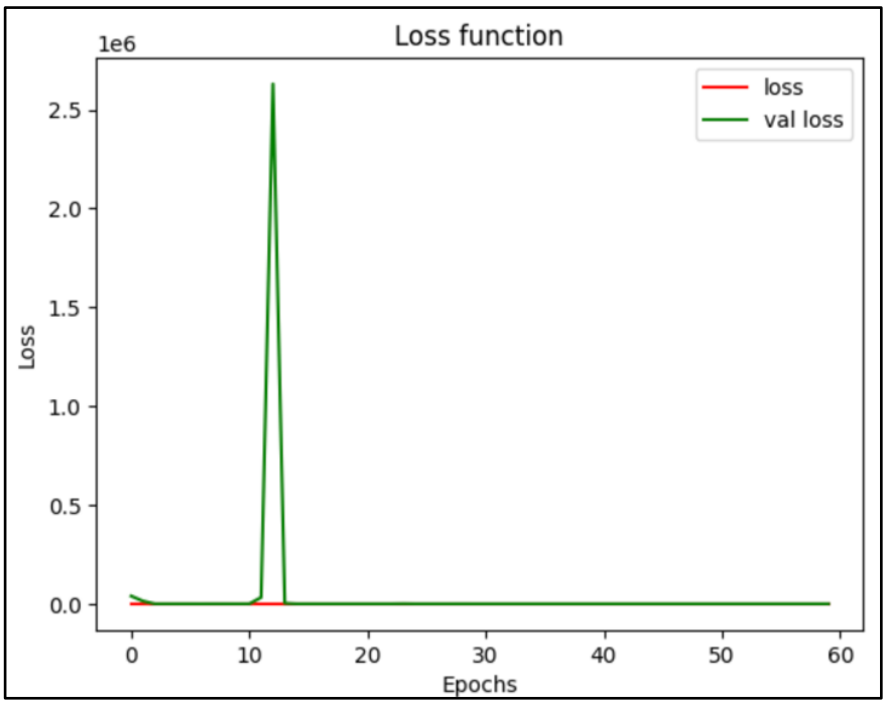

Рисунок 6 – График изменения ошибки во время проведения эксперимента

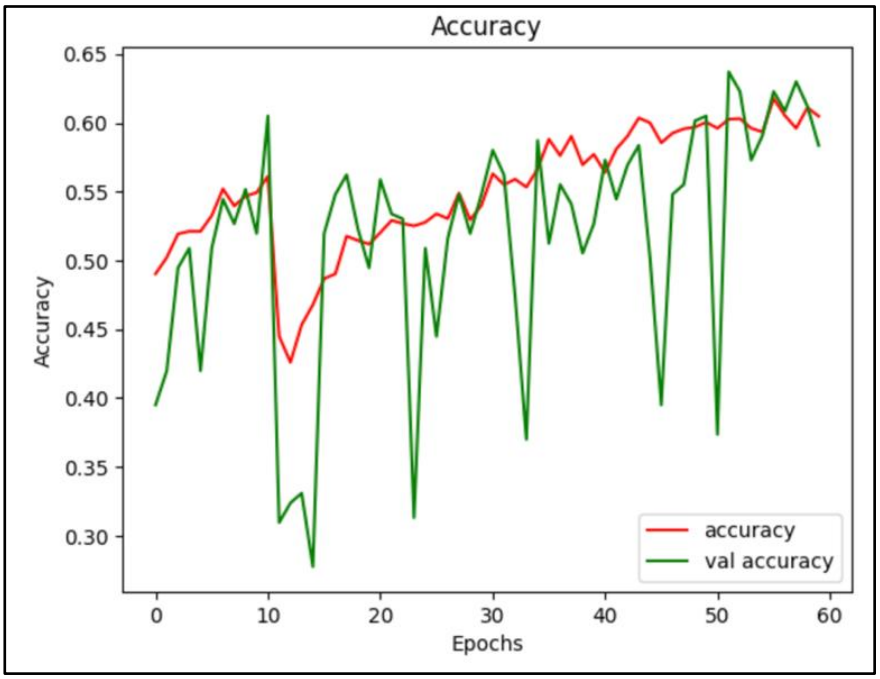

Рисунок 7 – График изменения метрики accuracy во время проведения эксперимента

Нейронная сеть с такими низкими метриками качества не могла быть добавлена в итоговый сервис, поэтому эксперименты продолжились.

Следующей опробованной нейронной сетью стала модель CLIP [13]. Модель CLIP была разработана исследователями OpenAI для проверки способности моделей обобщать произвольные задачи классификации изображений с нулевым результатом. В данном случае перед моделью стоит задача классификации – присвоении одной метки класса эмоции всему изображению.

Модель CLIP переносится на большинство задач и часто является конкурентоспособной благодаря полностью контролируемому базовому состоянию без необходимости какого-либо специального обучения набору данных. При этом скорость работы модели на изображениях составляет примерно 0,25 секунды.

После этого эксперимента было принято решение использовать модель CLIP в качестве классификатора эмоций в сервисе по определению уникальности изображений.

Реализация работы модели определения эмоций в сервисе определения уникальности изображений представлена в листинге 3.

Листинг 3 – Реализация работы модели определения эмоций

```
import io
from abc import ABC

import PIL.Image as Image
import torch
import torchvision
from transformers import CLIPProcessor, CLIPModel
from datetime import datetime

print(torch.__version__)
print(torchvision.__version__)

from model.processing_model_base import ProcessingModelBase

EMOTION_LABELS = ['happy', 'scary', 'calm', 'tender', 'melancholy']
MODEL_NAME = "openai/clip-vit-base-patch32"

model = CLIPModel.from_pretrained(MODEL_NAME)
processor = CLIPProcessor.from_pretrained(MODEL_NAME)
def get_emotion_name(predict):
    return EMOTION_LABELS[(predict.index(max(predict)))]
class EmotionModel(ProcessingModelBase, ABC):
    def __init__(self):
        super().__init__()

    def process_data(self, bytes_data):
        started_time = datetime.utcnow()
```

```
        image = Image.open(io.BytesIO(bytes_data)).convert('RGB')
        inputs = processor(text=EMOTION_LABELS, images=image,
return_tensors="pt", padding=True)
        outputs = model(**inputs)
        logits_per_image = outputs.logits_per_image
        predict = list(logits_per_image.softmax(dim=1)[0])
        tags = [get_emotion_name(predict)]

        ended_time = datetime.utcnow()
        print(f'TIME:{ended_time - started_time}')

        return [{'name': tag} for tag in tags]
```

3.4. Определение ассоциаций к найденным на изображении тегам

Следующим этапом выявления концептуально важных тегов фотографии стал поиск ассоциаций к тегам, которые уже были определены.

К этому моменту на изображении автоматически уже должны быть найдены предметы, окружение предметов, эмоция и основные цвета.

Идея поиска ассоциаций к найденным тегом появилась из желания добавить большего контекста фотографии, не останавливаться только на том, что физически присутствует на изображении. Например, набор слов «море», «песок», «солнце», «лето» может логически дополниться словом «отдых», что может заметно расширить описание изображения.

На данный момент не понятно, как можно в задаче поиска ассоциаций использовать найденные цвета, так как они представлены вектором из чисел. Использование цветов в задаче поиска ассоциаций – одна из запланированных доработок системы. Поэтому сервис поиска ассоциаций по тегам использует найденные предметы, окружение и эмоцию изображения.

Среди рассмотренных алгоритмов поиска ассоциаций к набору слов были такие алгоритмы, как ассоциативные правила, решающие деревья, RNN-архитектуры нейронных сетей и трансформерные архитектуры нейронных сетей.

Ассоциативные правила были отброшены в связи с тем, что открытых датасетов, подходящих для этой задачи, не было найдено, а раметка собственного датасета подразумевала бы продумываение тех правил, которые должны были бы быть выведены с помощью алгоритмов. Решающие

деревья в данном случае имели длительное время обработки запроса. RNN-архитектуры показали себя как очень требовательные к ресурсам машины, на которой они запущены, и их время обработки в среднем выше, чем время обработки данных с помощью трансформерных архитектур.

Выбранная предварительно обученная модель имеет архитектуру NLI.

Эта архитектура является трансформером, что значит, что она основана на механизме внимания без использования реккурентных нейронных сетей. Главное преимущество трансформеров заключается в их способности обрабатывать длительные зависимости в последовательностях. Кроме того, они очень производительны, могут обрабатывать последовательности параллельно. Это особенно полезно в задачах вроде машинного перевода, анализа настроений и синтеза текста [14].

Данная модель NLI может по предложенным ей возможным меткам класса выбрать те, которые подойдут по контексту существующему набору слов. При таком варианте все теги с предыдущих шагов объединяются в одну строку и подаются на вход модели как предложение.

Среди возможных меток ассоциаций к тегам были выбраны 400 наиболее встречающихся слов английского языка, среди которых были существительные и прилагательные.

Английский язык был использован в связи с тем, что модель была предварительно обучена на предложениях и словосочетаниях английского языка.

Примеры слов из словаря ассоциаций:

1) wedding – свадьба;
2) access – доступ;
3) car – машина;
4) experience – опыт;
5) nervous – нервозный;
6) kind – добрый.

При этом модель дает не только набор слов, которые наиболее сильно подходят контексту предложения, но и определяет, насколько сильно это слово подходит. Таким образом можно отсеять ассоциации, которые не имеют достаточной схожести с изначальным текстом.

Реализация работы модели, которая отвечает за поиск ассоциаций в системе определения уникальности фотографии представлен в листинге 4.

Листинг 4 – Реализация работы модели определения ассоциаций

```
from abc import ABC

import torch
import torchvision
from transformers import pipeline
from datetime import datetime

from model.processing_model_base import ProcessingModelBase
print(torch.__version__)
print(torchvision.__version__)

with open('./model/association_dict.txt', 'r') as dict_file:
    CANDIDATE_LABELS = dict_file.read().split("\n")


MODEL_NAME = 'cross-encoder/nli-distilroberta-base'

MODEL = pipeline("zero-shot-classification", model=MODEL_NAME)


class AssociationModel(ProcessingModelBase, ABC):
    def __init__(self):
        super().__init__()

    def process_data(self, sequence_to_classify):
        predict = MODEL(sequence_to_classify, CANDIDATE_LABELS,
multi_label=True)
        result = [predict['labels'][i] for i in
range(len(predict['labels']))
                  if predict['scores'][i] > 0.9]

        return [{'name': tag} for tag in result]
```

4. РАСЧЕТ УНИКАЛЬНОСТИ

Все найденные составляющие можно разделить на две группы:

1) категориальные составляющие, к которым относятся предметы, окружение, ассоциации, поскольку они представлены вектором из слов;

2) численные составляющие, к которым относится цветовая составляющая, поскольку она представлена вектором из чисел.

При выведении формулы уникальности основной упор был именно на то, чтобы данную формулу можно было логически понять пользователю системы. Любые не интерпретируемые расчеты, алгоритмы хотелось исключить. Для данных групп было принято решение разделить формулы уникальности и показывать пользователю в интерфейсе два показателя уникальности (отдельно по цветовой составляющей и отдельно по всем категориальным составляющим совместно), поскольку на данном этапе не получилось вывести обобщающую формулу для интерпретируемого расчета. Вектор для всех категориальных составляющим можно привести к числовому виду. Сделать это можно, поскольку у каждой возможной составляющей есть два состояния: она присутствует на изображении или не присутствует. Состояния можно закодировать в векторе с помощью 0, если признак отсутствует, или 1, если признак присутствует. В таком случае, если словарь всех возможных категориальных признаков будет содержать слова [«машина», «кровать», «тетрадь»], то вектор [«машина»] можно закодировать как [1, 0, 0], а вектор [«кровать», «тетрадь»] как [0, 1, 1]. При этом важно учитывать, что словарь всех возможных значений должен быть известен заранее, чтобы можно было закодировать все найденные признаки изображения.

В этой работе выдвигается предположение, что уникальность может варьироваться в некоторых пределах, например, от 0% до 100%.

Можно сделать предположение, что если вектор полностью совпадает с каким-либо другим, уже записанным в системе вектором, то его уникальность равняется 0%. При этом если вектор полностью не совпадает

ни с каким из записанных в системе векторов, то его уникальность равняется 100%.

Так как количество слов в словаре всех возможных категориальных признаков ограничено, уникальность не может расти неконтролируемо. С каждым новым появившемся, отличающимся от всех остальных признаком, уникальность растет на определенную величину, определенный шаг. Этот шаг может зависеть от количества всех возможных слов в словаре и от количества векторов, с которыми сравнивается поступивший новый вектор.

В таблице 4 показано, какие возможные варианты уникальности могут быть в разных ситуациях.

Таблица 4 – Возможные варианты уникальности

Количество всех возможных слов в словаре	**Количество векторов, с которыми сравнивается входящий вектор**	**Все возможные значения уникальности**
2	1	0, ½, 1
2	2	0, ⅓, ⅔, 1
2	3	0, ¼, ½, ¾, 1
2	4	0, ⅕, ⅖, ⅗, ⅘, 1

Например, входящий вектор – [0, 0]. Количество всех возможных слов в словаре в этом случае равняется 2. Предположим, что в сервисе уже есть один вектор, который описывает другую фотографию. В этом примере для того, чтобы использовать дроби, будем использовать не процентную шкалу, а шкалу от 0 до 1. Если существующий вектор имеет значение [0, 0], то уникальность нового входящего вектора – 0. Если существующий вектор имеет значение [0,1] или [1,0], то поступающий вектор совпадает с существующим вектором на одной позиции. В таком случае можно считать, что уникальность нового вектора равняется 1/2. Если существующий вектор имеет значение [1,1], то новый входящий вектор имеет значение уникальности 1. При этом, если бы в сервисе было уже два вектора, то входящий вектор мог бы:

1) совпадать полностью с каким-то из уже имеющихся – значение уникальности равно 0;

2) не совпадать ни с каким из векторов – значение уникальности равно 1;

3) совпадать хотя бы с одним значением из всех векторов (например, если вектора [0, 1] и [1,1], то входящий вектор [0,0] совпадает на одно значение с одним из векторов) – 2/3 уникальности;

4) совпадать с двумя значениями из двух векторов (например, если вектора [0, 1] и [1,0], то входящий вектор [0,0] совпадает с двумя значениями из двух векторов) – 1/3 уникальности.

Размер роста уникальности можно вычислить путем деления максимального значения уникальности на количество вариантов уникальности. Вычисление уникальности с применением роста уникальности отражено в формуле (3):

$$U = \begin{cases} 0, uniqCount = 0 \\ 1, uniqCount = num \\ \left(uniqCount - (num - 1)\right) * \dfrac{1}{(len - 1) * num + 1} \end{cases}, \quad (3)$$

где U – это уникальность;

$uniqCount$ – количество различающихся позиций во всех векторах суммарно;

num – количество векторов, с которыми сравнивается входящий вектор;

len – количество всех возможных слов в словаре.

Как было указано ранее, цветовая составляющая сильно отличается от категориального представления. Для того, чтобы уникальность цветовой составляющей на первом этапе можно было считать по той же формуле, что и категориальную, было принято решение применять квантование цветовой составляющей. В таком случае вектор цветовой составляющей может также быть представлен в виде [0, 0, …, 1].

После представления вектора цветов в такой же форме, как и для категориальной составляющей, к ней можно применить ту же формулу расчета уникальности.

5. ПРОЕКТИРОВАНИЕ

В данной главе изложены требования к разрабатываемой системе, описана архитектурная диаграмма сервиса, диаграмма взаимодействия пользователя с сервисом.

5.1. Функциональные и нефункциональные требования к системе

Функциональные требования описывают поведение системы, т.е. ее действия (вычисления, преобразования, проверки, обработку и т.д.) [15].

Разрабатываемая система должна удовлетворять следующим функциональным требованиям:

1) пользователь должен иметь возможность аутентифицироваться в системе;

2) пользователь должен иметь возможность создать галерею, в рамках которой будет производиться сравнение загруженных фотографий;

3) пользователь должен иметь возможность загрузки новой фотографии в существующую галерею;

4) система должна предоставлять для пользователя полученные теги и итоговую метрику уникальности загруженной фотографии в рамках галереи.

Нефункциональные требования описывают свойства системы (удобство использования, безопасность, надежность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения [15].

Разрабатываемая система должна удовлетворять следующим нефункциональным требованиям:

1) система должна быть веб-сервисом;

2) серверная часть системы должна быть разработана с помощью языка Python;

3) клиентская часть системы должна быть разработана на JavaScript, с помощью библиотеки React;

4) развертывание веб-приложения должно быть реализовано с помочью платформы виртуализации Docker.

5.2. Архитектура сервиса

Основываясь на выдвинутых требованиях и обзоре архитектурных паттернов, была спроектирована архитектура сервиса, отображенная на рисунке 8.

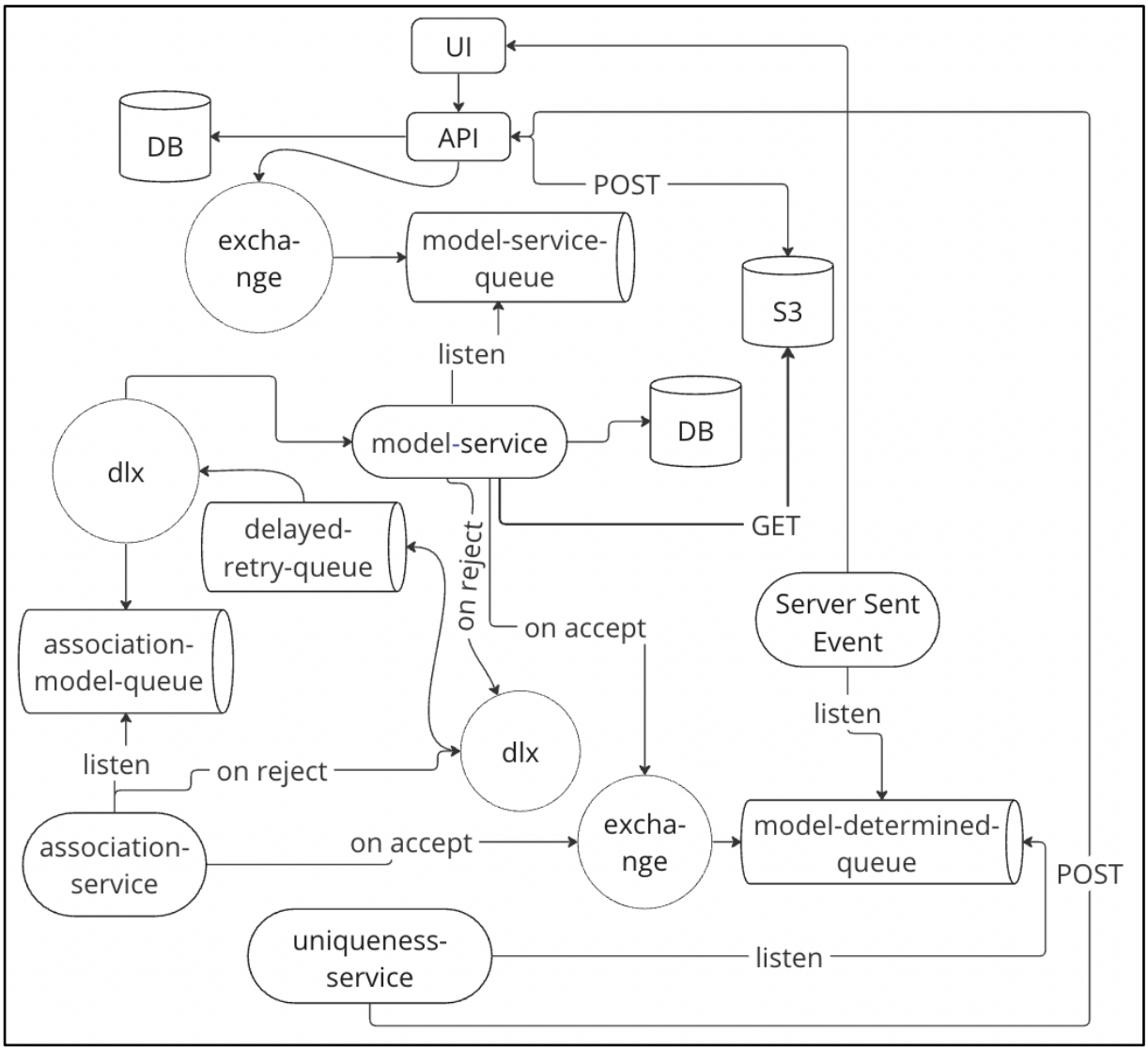

Рисунок 8 – Архитектурная диаграмма сервиса

Данная архитектура использует паттерн событийно-ориентированной архитектуры, что значит, что все сервисы общаются с друг с другом на

основании произошедших событий. В этой диаграмме model-service – это общее представление микросервисов с алгоритмами компьютерного зрения.

Под этим сервисом подразумеваются как сервис с распознаванием эмоций, так и сервис с распознаванием цветов и предметов.

Пример события в системе: событие PhotoSaved сигнализирует о том, что фотография, отправленная пользователем в сервис, была успешно сохранена и может быть обработана сервисами для автоматизированного нахождения признаков.

Архитектура сервиса включает в себя следующие микросервисы.

1. UI – пользовательский интерфейс. Функции сервиса: предоставлять интерфейс для пользователя, коммуницировать с сервисом API.

2. API – сервис, ответсвенный за предоставление необходимых для UI данных. Функции сервиса: получать фотографии от пользователя, записывать фото в S3-хранилище, отправлять фото на последующую обработку, записывать информацию о фото в базу данных, запрашивать результаты обработки фото у остальных сервисов.

3. Сервис color-service отвечает за распознавание цветов на фотографии. Функции сервиса: принимать запрос на обработку фото из очереди, получать фото из S3-хранилища, распознавать доминантные цвета фотографии, передавать результаты распознавания в очередь с результатами.

4. Сервис emotion-service отвечает за распознавание основной эмоции фотографии. Функции сервиса: принимать запрос на обработку фото из очереди, получать фото из S3-хранилища, распознавать основную эмоцию фотографии, передавать результаты распознавания в очередь с результатами.

5. Сервис object-service отвечает за распознавание объектов и их окружения на фотографии. Функции сервиса: принимать запрос на обработку фото из очереди, получать фото из S3-хранилища, распознавать объекты и их окружение на фотографии, передавать результаты распознавания в очередь с результатами.

6. Сервис association-service отвечает за определение ассоциаций к уже найденным тегам. Функции сервиса: принимать запрос на обработку фотографии из очереди, определять ассоциации по тегам, которые были распознаны другими моделями, отправлять результат обработки в очередь с результатами.

7. Server Send Event – сервис, отправляющий на фронтенд сигнал о том, что результаты обработки фото готовы. Функции сервиса: отправлять события о том, что что-либо можно отобразить на UI.

8. Сервис uniqueness-service – сервис расчета уникальности. Функции сервиса: запрашивать теги для рассчитываемой фотографии, рассчитывать уникальность фотографии.

В архитектурной схеме также содержатся атрибуты RabbitMQ.

RabbitMQ – это программное обеспечение с открытым исходным кодом для брокера сообщений (иногда называемое промежуточным ПО, ориентированным на сообщения), которое реализует протокол Advanced Message Queuing Protocol (AMQP).

Сервер RabbitMQ написан на языке программирования Erlang и построен на базе фреймворка Open Telecom Platform для кластеризации и отказоустойчивости. Клиентские библиотеки для взаимодействия с брокером доступны для всех основных языков программирования [16].

Exchange – обменник или точка обмена. В него отправляются сообщения. Exchange распределяет сообщение в одну или несколько очередей. Он маршрутизирует сообщения в очередь на основе созданных связей (bindings) между ним и очередью [17].

Queue (очередь) – структура данных на диске или в оперативной памяти, которая хранит ссылки на сообщения и отдает их копии consumers (потребителям) [18].

Работа с получением и отправкой сообщений в очередь реализована в классе `MessagesTrafficController` (листинг 3 приложения).

6. РЕАЛИЗАЦИЯ

При реализации сервиса были использованы технологии, описанные в таблице 5.

Таблица 5 – Использованные при разработке технологии

Часть системы	Технологии
Frontend	Javascript, React, MobX
Backend	Python, Flask, Pydantic, RabbitMQ, PostgreSQL, poetry
Машинное обучение	Python, PyTorch, scikit-learn, transformers, OpenCV
Тестирование	Cypress, Pytest, Postman
Развертывание	Docker, docker-compose, GitLab CI
Дизайн	Figma

6.1. Интерфейс приложения

Интерфейс приложения было решено сделать ненагруженным, в светлых тонах, чтобы внимание пользователя было сосредоточено на загружаемых фотографиях.

На рисунке 9 отображен интерфейс входа в веб-приложение. На данном этапе в сервисе можно только аутентифицироваться, регистрация новых пользователей недоступна.

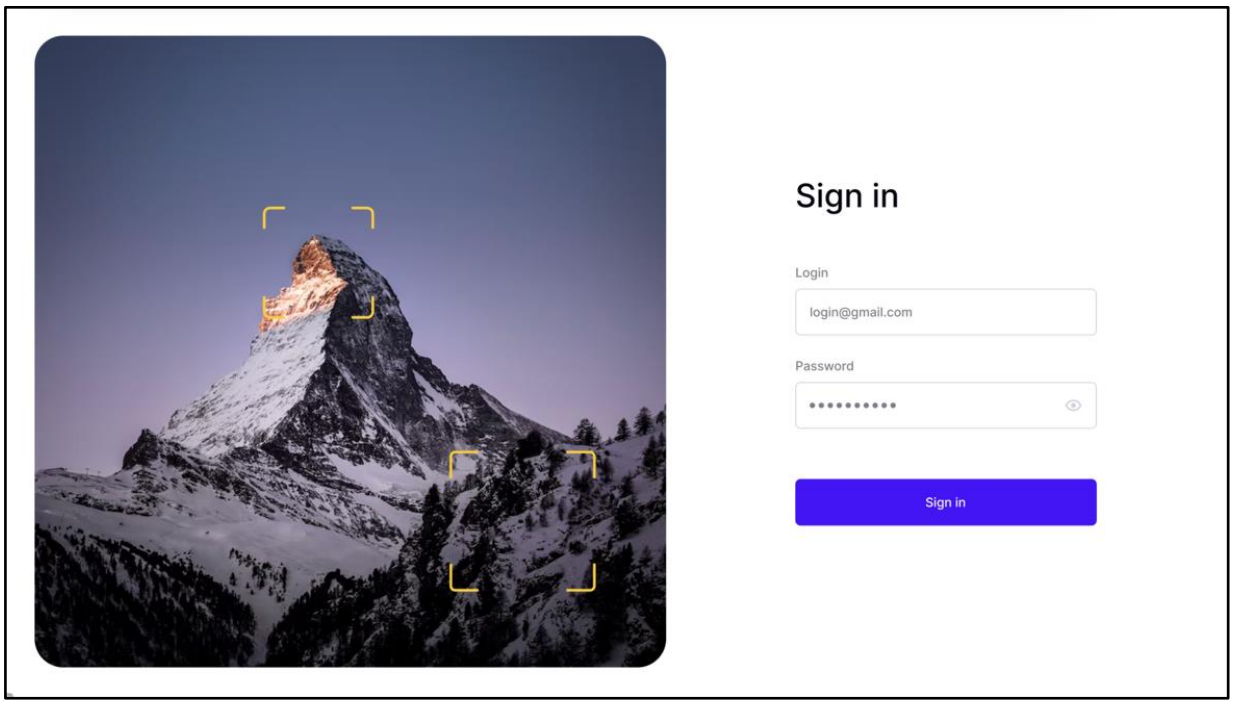

Рисунок 9 – Интерфейс входа в систему

На рисунке 10 отображен интерфейс после входа пользователя в систему. На этом экране приложения пользователь может создать галерею, в рамках которой будет оцениваться уникальность фотографии.

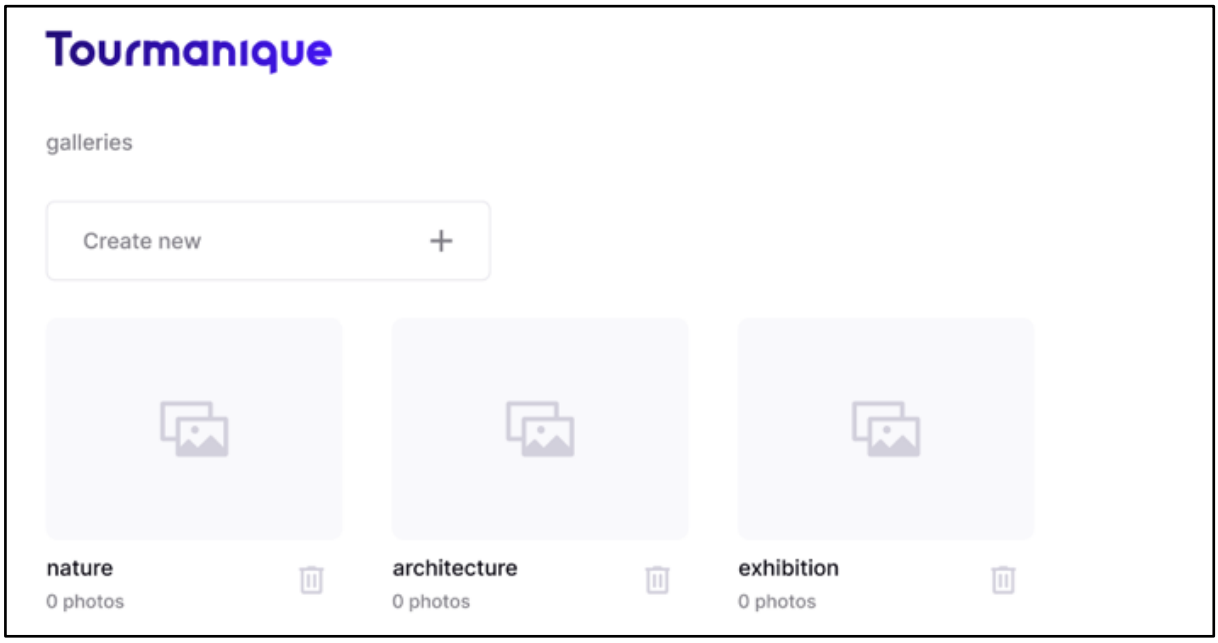

Рисунок 10 – Интерфейс для создания галерей

На рисунке 11 отображена страница внутри галереи, на которой пользователь может загрузить новую фотографию для автоматической оценки уникальности.

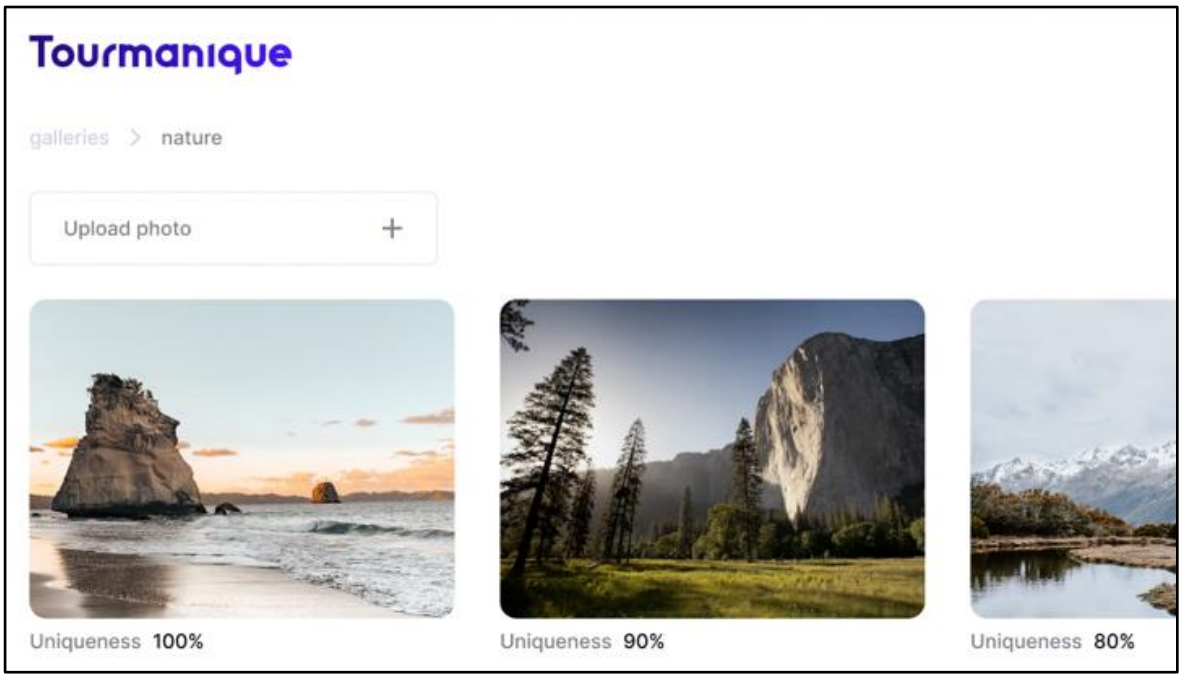

Рисунок 11 – Интерфейс добавления фотографий в сервис

Последний из возможных на текущий момент интерфейсов – интерфейс, который отображает определенные системой автоматически составляющие и показывает итоговую метрику уникальности. Этот интерфейс отображен на рисунке 12.

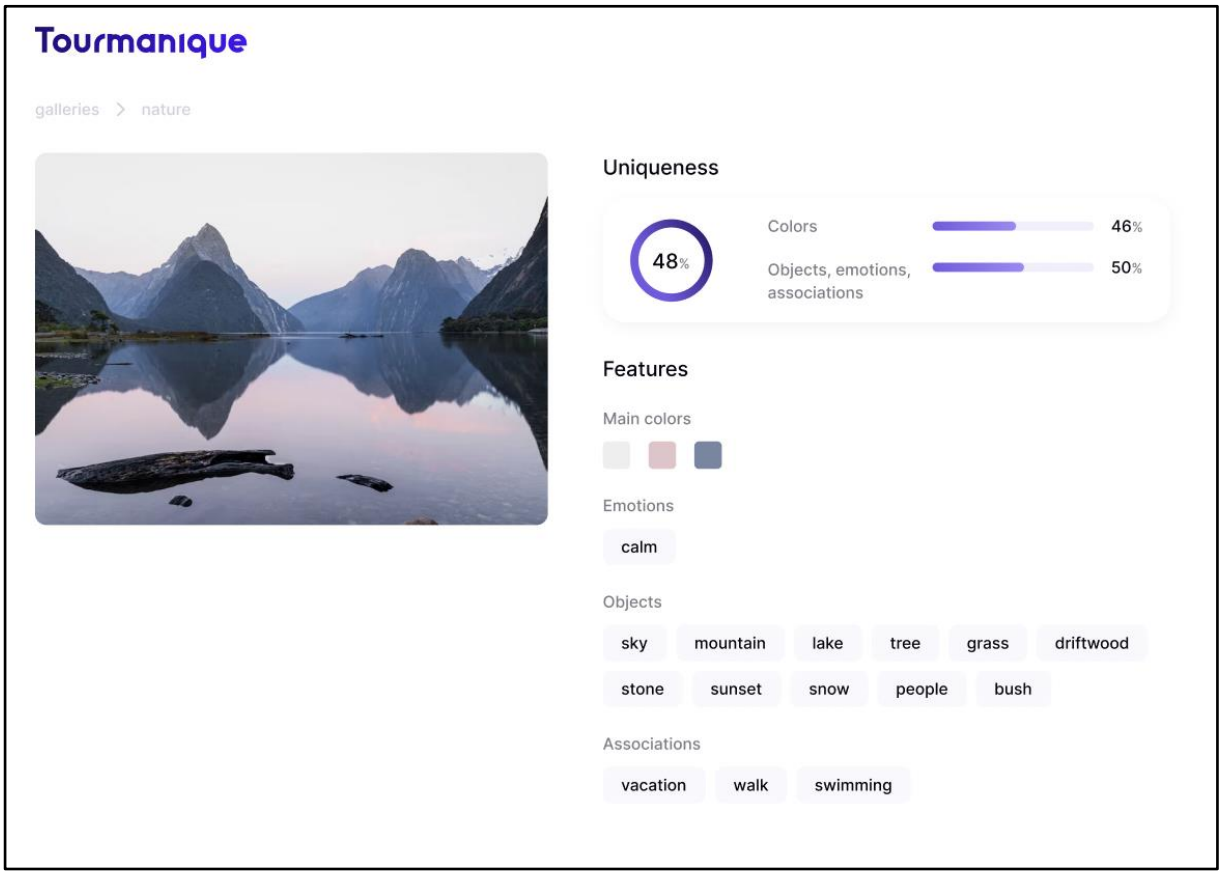

Рисунок 12 – Интерфейс обзора фотографии с определенными с помощью машинного обучения признаками фото

6.2. Реализация аутентификации и хранения

S3-хранилище – сервис, где хранятся цифровые данные большого объема. Работает по одноименному протоколу.

В качестве S3-хранилища в этом проекте используется MinioS3. Это опенсорсное объектное хранилище, совместимое с Amazon S3 API [19].

Для работы с хранилищем, в которое отправлялись фотографии, был реализован специальный класс `S3Helper`. Его реализация описана в листинге 4 приложения.

На данный момент системой может пользоваться один пользователь. Так как эта работа необходима для доказательства гипотезы, было принято решение не усложнять аутентификацию пользователя использованием базы данных и использованием хэширования. На данный момент в системе сравнивается логин и пароль, и, если они полностью совпали, пользователь может пользоваться системой. Реализация описана в листинге 7.

Листинг 7 – Реализация эндпоинта аутентификации пользователя

```
@auth_blueprint.route('/login', methods=['POST'])
def log_in():
    user_data = request.json.get('login')
    password_data = request.json.get('password')

    if user_data != auth_username or password_data != auth_password:
        return jsonify({'msg': 'Bad username or password'}),
HTTPStatus.UNAUTHORIZED

    access_token = create_access_token(identity=USER_ID,

expires_delta=datetime.timedelta(days=30))

    return jsonify({
        "accessToken": {
            "value": access_token
        }
    }), HTTPStatus.ACCEPTED
```

6.3. Реализация получения фото на сервере

После того, как пользователь в интерфейсе выбирает фотографию, для которой необходимо провести автоматическое получение составляющих и метрики уникальности, серверная часть веб-приложения получает запрос. Эндпоинт получения фотографии был реализован согласно листингу 8.

Листинг 8 – Реализация эндпоинта получения фотографии

```
@photos_blueprint.route('/<int:gallery_id>/upload-photo', methods=['POST'])
@jwt_required()
def add_photo(gallery_id):
    photo_bytes = request.get_data()
    current_user_id = get_jwt_identity()
    if not GetGalleryQuery().by_id(gallery_id):
        return jsonify({'msg': 'Not Found'}), HTTPStatus.NOT_FOUND
    if not IsUserHasAccess().to_gallery(current_user_id, gallery_id):
        return jsonify({'msg': 'Forbidden'}), HTTPStatus.FORBIDDEN

    photo_s3_path = create_path_for_photo()

    S3Helper().s3_upload_file(
        file_path_in_bucket=photo_s3_path,
        file_bytes=photo_bytes,
```

```
        public=True,
    )

    photo_hash =
str(imagehash.average_hash(Image.open(io.BytesIO(photo_bytes))))
    photo_entity = Photo(photo_file_path_s3=photo_s3_path,
                         hash=photo_hash,
                         gallery_id=gallery_id,
                         date_of_upload=datetime.utcnow(),
                         color_uniqueness=None,
                         tag_uniqueness=None,
                         overall_uniqueness=None,
                         )
    photo_id = NewPhotoCommand().create(photo_entity)

    message_with_photo_parameters = {
        'photo_id': photo_id,
        'path_to_photo_in_s3': photo_s3_path,
    }


RabbitMqMessagePublisher().publish_message_to_exchange(exchange_name=rabbit
mq_photo_for_models_exchange_name,

message=message_with_photo_parameters)

    return jsonify({'msg': 'OK'}), HTTPStatus.OK
```

6.4. Развертывание приложения

Для того, чтобы приложение можно было развернуть на любом сервере, была использована технология Docker.

Docker – это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением [20].

Для того, чтобы приложение можно было использовать как docker-контейнер, были написаны Dockerfile и docker-compose.yaml для каждого сервиса. Сервисы бэкенда имели во многом схожую структуру.

Docker Compose – это средство для определения и запуска приложений Docker с несколькими контейнерами [21].

Dockerfile для сервиса API описан в листинге 9.

Листинг 9 – Dockerfile для API-сервиса

```
FROM python:3.10.9-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 . .

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

CMD poetry run python application.py
```

Docker-compose файл для сервиса API описан в листинге 5 приложения. В этом файле описаны контейнеры, которые содержат сервис API, его базу данных, сервис RabbitMQ.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан веб-сервис для определения уникальности изображения на основе алгоритмов машинного обучения. Среди рассмотренных алгоритмов были добавлены в качестве основных:

1) определение перцептивного хэша для вычисления полного плагиата фотографии;

2) определение цветовой составляющей изображения на основе метода KNN;

3) определение предметов на фотографии и их окружения с помощью методов семантической сегментации;

4) определение общего настроения фотографии на основе нейронной сети архитектуры Inception-Resnet;

5) определение ассоциаций к уже найденным составляющим фотографии.

При этом были решены следующие задачи.

1. Рассмотрена уникальность в пределах сервиса с точки зрения составляющих фотографии.

2. Рассмотрены архитектурные паттерны, подходящие для создания веб-приложения с несколькими алгоритмами машинного обучения.

3. Разработан веб-сервис с несколькими алгоритмами машинного обучения, нацеленными на вычисление уникальности фотографии в рамках сервиса.

4. Разработана формула для определения уникальности фотографии по найденным автоматически составляющим.

5. Рассмотрены возможные варианты сравнения определенных составляющих фотографии.

В настоящий момент производится уточнение методов распознавания объектов и окружения на изображении.

Также в числе дальнейших экспериментов, продолжающих работу по поиску составляющих фотографии:

1) использование эмбеддингов фотографии для не интерпретируемой оценки уникальности;

2) анализ изображения с использованием моделей, умеющих вычленять основную идею фотографии в текст, автоматически собирать описание фотографии;

3) распознавание замкнутых контуров фотографии для дальнейшего анализа частей изображений, находящихся в замкнутых контурах;

4) определение ключевых точек, дескрипторов фотографии для дальнейшего сравнения и более устойчивого поиска плагиата фотографий;

5) поиск возможности определения композиции фотографии по ключевым предметам на ней и сравнение композиций;

6) оптимизация работы алгоритмов, использующихся в данной работе, поскольку время обработки фотографии на данный момент составляет порядка 5 минут.

ЛИТЕРАТУРА

1. Zigflow. [Электронный ресурс] URL: https://www.zi-flow.com/blog/digital-marketing-statistics (дата обращения: 01.04.2024 г.).

2. Многоуровневая архитектура. [Электронный ресурс] URL: https://appmaster.io/ru/glossary/mnogourovnevaia-arkhitektura (дата обращения: 01.04.2024 г.).

3. Model View Controller (MVC) опыт использования, выводы. [Электронный ресурс] URL: https://habr.com/ru/articles/249263/ (дата обращения: 01.04.2024 г.).

4. Как событийно-ориентированная архитектура решает проблемы современных веб-приложений. [Электронный ресурс] URL: https://habr.com/ru/companies/piter/articles/530514/ (дата обращения: 01.04.2024 г.).

5. Что такое событийная (Event Driven) архитектура. [Электронный ресурс] URL: https://apptractor.ru/develop/event-driven-architecture.html (дата обращения: 01.04.2024 г.).

6. Ожегов С.И., Шведова Н.Ю. Толковый словарь русского языка. – 4-е изд. – М., 1997. – 944 с.

7. Ушаков Д.Н. Толковый словарь современного русского языка. – Москва: Аделант, 2014. – 800 с.

8. Westland S., Cheung V. RGB Systems // Handbook of Visual Display Technology. – 2015. – P. 1–6.

9. Tamilselvi P., Nagasankar P., Geetha S. Color based K-Means Clustering For Image Segmentation to Identify the Infected Leaves. // 2019 Fifth International Conference on Science Technology Engineering and Mathematics (ICONSTEM). – Chennai, India, March 14–15, 2019. – 86 с.

10. Конушин А.С. Конспект лекций по компьютерному зрению ВМК МГУ. [Электронный ресурс] URL: https://teach-in.ru/file/synopsis/pdf/computer-vision-M.pdf (дата обращения: 03.01.2024 г.).

11. Сегментация изображений. [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php? ?title=Сегментация_изображений (дата обращения: 10.02.2024 г.).

12. Xie E., Wenhai W., Zhiding Y., Anandkumar A., Alvarez J. M., Luo P. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers [Электронный ресурс] // arXiv.org 2021 г. Дата обновления: 31.05.2021 г. URL: https://arxiv.org/abs/2105.15203 (дата обращения: 10.02.2024 г.).

13. Radford A., Kim J.W., Hallacy C., Ramesh A., Goh G., Agarwal S., Girish S., Askell A., Mishkin P., Clark J., Krueger G., Sutskever I. Learning Transferable Visual Models From Natural Language Supervision [Электронный ресурс] // arXiv.org 2021 г. Дата обновления: 26.02.2021 г. URL: https://arxiv.org/abs/2103.00020 (дата обращения: 10.02.2024 г.).

14. Объясняем простым языком, что такое трансформеры. [Электронный ресурс] URL: https://habr.com/ru/companies/cloud_mts/articles/770202/ (дата обращения: 19.02.2024 г.).

15. Куликов С.С. Тестирование программного обеспечения. Базовый курс. 2 издание. – Издательство Четыре Четверти, 2017. – 312 с.

16. DockerHub: RabbitMQ. [Электронный ресурс] URL: https://hub.docker.com/_/rabbitmq (дата обращения: 19.02.2024 г.).

17. RabbitMQ. Часть 2. Разбираемся с Exchanges. [Электронный ресурс] URL: https://habr.com/ru/articles/489086/ (дата обращения: 19.02.2024 г.).

18. RabbitMQ. Часть 3. Разбираемся с Queues и Bindings. [Электронный ресурс] URL: https://habr.com/ru/articles/490960/ (дата обращения: 19.02.2024 г.).

19. MinIO для самых маленьких. [Электронный ресурс] URL: https://habr.com/ru/companies/veeam/articles/517392/ (дата обращения: 19.02.2024 г.).

20. Понимая Docker. [Электронный ресурс] URL: https://habr.com/ru/articles/253877/ (дата обращения: 19.02.2024 г.).

21. Развертывание нескольких контейнеров с помощью Docker Compose. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/azure/ai-services/containers/docker-compose-recipe (дата обращения: 19.02.2024 г.).

ПРИЛОЖЕНИЕ. Листинги реализации.

Листинг 1 – Реализация метода K-средних для поиска цветовой составляющей

```
import logging
from abc import ABC

import cv2
import io
from datetime import datetime
from sklearn.cluster import KMeans
from collections import Counter
import imageio

from model.processing_model_base import ProcessingModelBase


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


class ColorModel(ProcessingModelBase, ABC):
    def __init__(self):
        super().__init__()

    def process_data(self, bytes_data):
        # ToDo use only utc now
        started_time = datetime.now()
        image = imageio.imread(io.BytesIO(bytes_data), as_gray=False,
pilmode="RGB")
        colors, values = self.get_base_colors(image, 4)
        ended_time = datetime.now()

        logging.warning(f'TIME:{ended_time - started_time}')

        return [{'red': str(color[0]), 'green': str(color[1]), 'blue':
str(color[2])} for color in colors]

    @staticmethod
    def get_base_colors(image, number_of_colours):
        image = modify_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, list(counts.values())
```

Листинг 2 – Реализация работы модели определения объектов и их окружения

```
from abc import ABC

import torch
import torchvision

print(torch.__version__)
print(torchvision.__version__)

from torch import nn
import pandas as pd
import io
import PIL.Image as Image
from transformers import SegformerFeatureExtractor,
SegformerForSemanticSegmentation
from datetime import datetime

from model.processing_model_base import ProcessingModelBase


OBJECT_LABELS = pd.read_csv('./model/OBJECT_LABELS.csv')
MODEL_NAME = "nvidia/segformer-b5-finetuned-ade-640-640"

feature_extractor = SegformerFeatureExtractor.from_pretrained(MODEL_NAME)
model = SegformerForSemanticSegmentation.from_pretrained(MODEL_NAME)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)


class ObjectModel(ProcessingModelBase, ABC):
    def __init__(self):
        super().__init__()

    def process_data(self, bytes_data):
        started_time = datetime.now()
        image = Image.open(io.BytesIO(bytes_data)).convert('RGB')

        pixel_values = feature_extractor(image,
return_tensors="pt").pixel_values.to(device)
        predict = model(pixel_values)
        logits = nn.functional.interpolate(predict.logits.detach().cpu(),
                                           size=image.size[::-1],
                                           mode='bilinear',
                                           align_corners=False)

        seg = logits.argmax(dim=1)[0]
        label_name = OBJECT_LABELS['name']
        labels = seg.unique()

        tags = []
        for label in labels:
            tags.append(label_name[int(label)])

        ended_time = datetime.now()
        print(f'TIME:{ended_time - started_time}')

        return [{'name': tag} for tag in tags]
```

Листинг 3 – Класс `MessagesTrafficController` для работы с RabbiMQ-queues

```
class MessagesTrafficController:
    def __init__(self, model_type):
        self.model_type = model_type
        self.model_request_queue_name = f'{self.model_type}-queue'
        self.message_packer = MessagePacker(model_type)
        self.model = ColorModel
        self.connection = BlockingConnection(connection_parameters)

    def start_listening_to_the_queue(self):
        while True:
            try:

channel = self.connection.channel()

channel.exchange_declare(exchange=rabbitmq_requests_exchange_name,
exchange_type='fanout')

channel.exchange_declare(exchange=rabbitmq_models_queues_dlx_name,
exchange_type='direct')

channel.exchange_declare(exchange=rabbitmq_models_retry_queue_dlx_name,
exchange_type='direct')
                channel.basic_qos(prefetch_count=1)
                channel.queue_declare(
                    queue=self.model_request_queue_name,
                    arguments={
                        "x-dead-letter-exchange":
rabbitmq_models_queues_dlx_name,
                        'x-dead-letter-routing-key':
self.model_request_queue_name
                    },
                    durable=True,
                    exclusive=False,
                    auto_delete=False,
                )
                channel.queue_declare(
                    queue=rabbitmq_models_retry_queue_name,
                    arguments={
                        'x-message-ttl': rabbitmq_models_retry_delay_ms,
                        "x-dead-letter-exchange":
rabbitmq_models_retry_queue_dlx_name,
                    },
                    durable=True,
                    exclusive=False,
                    auto_delete=False,
                )


channel.queue_bind(exchange=rabbitmq_requests_exchange_name,
queue=self.model_request_queue_name)

channel.queue_bind(exchange=rabbitmq_models_queues_dlx_name,
queue=rabbitmq_models_retry_queue_name,

routing_key=self.model_request_queue_name)
```

```
channel.queue_bind(exchange=rabbitmq_models_retry_queue_dlx_name,
queue=self.model_request_queue_name,

routing_key=self.model_request_queue_name)

                channel.basic_consume(self.model_request_queue_name,
self.request_message_processing)
                channel.start_consuming()
                logging.warning('Started consumption from the queue:
{0}'.format(self.model_request_queue_name))

            except (ConnectionClosedByBroker, AMQPConnectionError):
                logging.warning('Connection was closed, retrying...')
                continue

            except AMQPChannelError as e:
                logging.error('Caught a channel error: {0},
stopping...'.format(e))

            except Exception as e:
                logging.error('Unexpected error occurred: {0}'.format(e))

    def request_message_processing(self, channel, method_frame,
header_frame, body):
        photo_id, photo_bytes =
self.message_packer.unpack_the_message_body(body)

        try:
            result = self.submit_for_processing(photo_bytes)
            message_with_result =
self.message_packer.pack_the_message_body(photo_id, result)
            self.put_message_to_result_queue(message_with_result)

        except Exception as e:
            logging.error('Unexpected error occurred: {0}'.format(e))
            retry_count = self.find_retry_count(header_frame)

            if retry_count >= rabbitmq_models_max_retry_number:
                channel.basic_ack(delivery_tag=method_frame.delivery_tag)
                logging.info('Cannot be processed.')
            else:

channel.basic_reject(delivery_tag=method_frame.delivery_tag, requeue=False)
                logging.info('Message rejected.')
            return

        channel.basic_ack(delivery_tag=method_frame.delivery_tag)

    @staticmethod
    def find_retry_count(header_frame):
        retry_count = 0
        if header_frame.headers is not None and 'x-death' in
header_frame.headers:
            retry_count = header_frame.headers['x-death'][0]['count']
        return retry_count

    def put_message_to_result_queue(self, message_with_result):
        channel = self.connection.channel()
        channel.queue_declare(queue=rabbitmq_results_queue_name,
durable=True)
```

```
        channel.basic_publish(
            exchange='',
            routing_key=rabbitmq_results_queue_name,
            body=message_with_result,
            properties=BasicProperties(
                delivery_mode=2,
            )
        )

    def submit_for_processing(self, photo_bytes):
        processing_result = self.model().process_data(photo_bytes)
        return processing_result
```

Листинг 4 – Класс для работы с S3-хранилищем

```
class S3Helper(metaclass=Singleton):
    def __init__(self):
        self.session = boto3.session.Session(
            aws_access_key_id=s3_config.s3_access_key_id,
            aws_secret_access_key=s3_config.s3_secret_access_key,
        )

    def get_client(self) -> S3Client:
        client: S3Client = self.session.client(
            's3',
            endpoint_url=s3_config.s3_endpoint,
            use_ssl=s3_config.s3_use_ssl
        )
        return client

    def get_resource(self) -> S3ServiceResource:
        resource: S3ServiceResource = self.session.resource(
            's3',
            endpoint_url=s3_config.s3_endpoint,
            use_ssl=s3_config.s3_use_ssl,
        )
        return resource

    @staticmethod
    def s3_get_full_file_url(
            file_path_in_bucket: str,
            s3_bucket_name: Optional[str] = s3_config.s3_bucket_name,
            s3_prefix: Optional[str] = s3_config.s3_prefix,
    ):
        result_path_in_bucket = append_prefix(path=file_path_in_bucket,
prefix=s3_prefix)

        return build_object_url(
            endpoint_url=s3_config.s3_endpoint,
            bucket=s3_bucket_name,
            path=result_path_in_bucket,
        )

    def list_buckets(self):
        client = self.get_client()
        buckets_response = client.list_buckets()

        # check buckets list returned successfully
```

```
        if buckets_response['ResponseMetadata']['HTTPStatusCode'] == 200:
            for s3_buckets in buckets_response['Buckets']:
                logging.debug(f' *** Bucket Name: {s3_buckets["Name"]} -
Created on {s3_buckets["CreationDate"]} \n')
        else:
            logging.debug(' *** Failed while trying to get buckets list
from your account')

        return buckets_response['Buckets']

    def s3_upload_file(
            self,
            file_path_in_bucket: str,
            file_bytes: bytes,
            s3_bucket_name: str = s3_config.s3_bucket_name,
            s3_prefix: Optional[str] = s3_config.s3_prefix,
            public: Optional[bool] = False,
            create_subdirs: Optional[bool] = False,
    ):
        bucket: Bucket = self.get_resource().Bucket(s3_bucket_name)
        result_path_in_bucket = append_prefix(path=file_path_in_bucket,
prefix=s3_prefix)

        if create_subdirs:
            print(f'Create subdir:
{get_parent_path(result_path_in_bucket)}')
            self._ensure_dir(get_parent_path(result_path_in_bucket))

        logging.warning(f'Started upload to bucket={s3_bucket_name} with
key={result_path_in_bucket}')
        with io.BytesIO(file_bytes) as buffer:
            extra_args = {'ACL': ACL_PRIVATE if not public else
ACL_PUBLIC_READ}
            bucket.upload_fileobj(Key=result_path_in_bucket,
                                  Fileobj=buffer,
                                  ExtraArgs=extra_args)

        logging.warning(f'File uploaded to bucket={s3_bucket_name} with
key={result_path_in_bucket}')

        return result_path_in_bucket

    def s3_set_existed_object_acl(
            self,
            file_path_in_bucket: str,
            s3_bucket_name: str = s3_config.s3_bucket_name,
            s3_prefix: Optional[str] = s3_config.s3_prefix,
            public: Optional[bool] = False,
    ):
        resource: S3ServiceResource = self.get_resource()
        result_path_in_bucket = append_prefix(path=file_path_in_bucket,
prefix=s3_prefix)

        object_acl = resource.ObjectAcl(s3_bucket_name,
result_path_in_bucket)

        logging.warning(f'Started setting ACL to {ACL_PUBLIC_READ if public
else ACL_PRIVATE} to object with'
                        f' key={result_path_in_bucket} in the
bucket={s3_bucket_name}')
```

```
        if public:
            _ = object_acl.put(ACL=ACL_PUBLIC_READ)
        else:
            _ = object_acl.put(ACL=ACL_PRIVATE)

    def s3_create_folder(
            self,
            file_path_in_bucket: str,
            s3_bucket_name: str = s3_config.s3_bucket_name,
            s3_prefix: Optional[str] = s3_config.s3_prefix,
            is_path_absolute: bool = False,
            create_subdirs: bool = False,
    ):
        s3_resource = self.get_resource()
        bucket: Bucket = s3_resource.Bucket(s3_bucket_name)

        if is_path_absolute:
            file_path_in_bucket_with_prefix = file_path_in_bucket
        else:
            file_path_in_bucket_with_prefix =
append_prefix(path=file_path_in_bucket, prefix=s3_prefix)

        result_folder_path_in_bucket =
create_folder_path(path=file_path_in_bucket_with_prefix)

        if create_subdirs:
            self._ensure_dir(get_parent_path(result_folder_path_in_bucket))

        with io.BytesIO() as buffer:
            bucket.upload_fileobj(Key=result_folder_path_in_bucket,
Fileobj=buffer, ExtraArgs={'ACL': 'private'})

        logging.warning(f'Folder created into bucket={s3_bucket_name} with
key={result_folder_path_in_bucket}')

    def s3_download_file(
            self,
            file_path_in_bucket,
            s3_bucket_name: str = s3_config.s3_bucket_name,
            s3_prefix: Optional[str] = s3_config.s3_prefix,
    ) -> bytes:
        bucket: Bucket = self.get_resource().Bucket(s3_bucket_name)
        result_path_in_bucket = append_prefix(path=file_path_in_bucket,
prefix=s3_prefix)
        print(result_path_in_bucket)
        with io.BytesIO() as buffer:
            bucket.download_fileobj(Key=result_path_in_bucket,
Fileobj=buffer)
            result = buffer.getvalue()
        logging.warning(
            f'File downloaded from bucket={s3_bucket_name} with
key={result_path_in_bucket} data[:10]={result[:min(10, len(result))]}')
        return result

    def s3_delete_file(
            self,
            file_path_in_bucket,
            s3_bucket_name: str = s3_config.s3_bucket_name,
            s3_prefix: Optional[str] = s3_config.s3_prefix,
    ):
```

```
        bucket: Bucket = self.get_resource().Bucket(s3_bucket_name)
        result_path_in_bucket = append_prefix(path=file_path_in_bucket,
prefix=s3_prefix)

        bucket.delete_objects(
            Delete={
                'Objects': [
                    {
                        'Key': result_path_in_bucket,
                    },
                ],
            },
        )
        logging.warning(
            f'File deleted from bucket={s3_bucket_name} with
key={result_path_in_bucket}')

    def _ensure_dir(self, path: str, s3_bucket_name: str =
s3_config.s3_bucket_name):
        bucket: Bucket = self.get_resource().Bucket(s3_bucket_name)

        if path in ('', '/', '.'):
            return
        print(f'Prefix: {path}')
        print(len(list(bucket.objects.filter(Prefix=path))))

        if len(list(bucket.objects.filter(Prefix=path))) == 0:
            logging.debug(f'Want to create dir: {path}')
            self.s3_create_folder(file_path_in_bucket=path,
is_path_absolute=True, create_subdirs=True)
```

Листинг 5 – Файл docker-compose.yml для API-сервиса

```
version: '3.8'

services:
  run-api-locally:
    container_name: tourmanique-api-local-starter
    restart: "no"
    image: tianon/true
    depends_on:
      - tourmanique-api
      - tourmanique-api-postgres
      - tourmanique-api-rabbitmq

  tourmanique-api:
    container_name: tourmanique-api
    image: tourmanique-api:latest
    restart: unless-stopped
    networks:
      - tourmanique-net

  tourmanique-api-postgres:
    container_name: tourmanique-api-postgres
    image: postgres:13.3-alpine
    restart: always
    volumes:
      - tourmanique-api-postgres-data:/var/lib/postgresql/data/
```

Окончание листинга 5 приложения

```
    healthcheck:
      test: "pg_isready -U $$POSTGRES_USER"
      timeout: 10s
      interval: 10s
      retries: 3
    networks:
      - tourmanique-net


  tourmanique-api-rabbitmq:
    container_name: tourmanique-api-rabbitmq
    image: rabbitmq:3.9.13-management-alpine
    restart: always
    healthcheck:
      test: rabbitmq-diagnostics -q ping
      interval: 10s
      timeout: 10s
      retries: 3
    networks:
      - tourmanique-net


volumes:
  tourmanique-api-postgres-data:

networks:
  tourmanique-net:
    driver: bridge
```