Министерство просвещения РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»

Институт математики, физики, информатики и технологий

Кафедра информатики, информационных технологий
и методики обучения информатике

ИНФОРМАЦИОННАЯ СИСТЕМА ДЛЯ ПОИСКА ИДЕНТИЧНЫХ ИЗОБРАЖЕНИЙ НА ОБЩЕДОСТУПНЫХ РЕСУРСАХ В СЕТИ ИНТЕРНЕТ

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.02 – Информационные системы и технологии

Исполнитель: студент группы
ИСиТ-1801
Института математики, физики,
информатики и технологий
Савельев С. Д.

Руководитель: кандидат
педагогических наук,
доцент кафедры ИИТ и МОИ
Газейкина А. И.

Работа допущена к защите
«____»_____ 2022 г.
Зав. кафедрой _____

Екатеринбург – 2022

Оглавление

Введение 4

Глава 1. Теоретические основы разработки информационных систем защиты авторского права 6

1.1 Анализ способов защиты авторского права на результаты интеллектуальной деятельности, размещённые в сети Интернет 6

1.2 Анализ средств разработки информационной системы поиска идентичных изображений 7

1.3 Техническое задание на разработку информационной системы поиска идентичных изображений 12

Глава 2. Реализация информационной системы поиска идентичных изображений 12

2.1 Структура информационной системы поиска идентичных изображений 16

2.2 Алгоритм определения подписи изображения 23

2.3 Алгоритм нахождения ближайших соседей изображения по определённой подписи 31

2.4 Демонстрация работы информационной системы поиска идентичных изображений 34

2.4.1 Демонстрация работы клиентской части информационной системы 34

2.4.2 Демонстрация работы серверной части информационной системы 35

2.4.3 Демонстрация работы внутреннего хранилища подписей изображений 36

2.5 Экспертная оценка разработанной информационной системы поиска идентичных изображений 37

Заключение 39

Список используемой литературы 40

Приложения 42

Введение

До появления интернета связь (передача голоса и данных) ограничивалась территориями между станциями, покрытыми сетью малого радиуса действия. Это было в 1950-х и начале 1960-х годов. Термин "интернет" используется для обозначения глобальной связи через протокол TCP/IP (Transmission Control Protocol/Internet Protocol). Первые открытия Интернета были сделаны в Великобритании; это было в 1858 году, когда был проложен первый кабель, но это была техническая неудача, он проработал всего несколько дней. Восемь лет спустя был проложен более совершенный кабель, который ознаменовал собой величайший успех в этой области. Кабель оставался в эксплуатации более века. Интернет с момента своего появления помог миру в процессе глобализации и остаётся жизненно важным инструментом, связывающим международные границы. Те, кто придумал технологию интернета, видели необходимость в том, чтобы позволить компьютерам обмениваться информацией для исследовательской работы и разработок в военной сфере и в области развития. Первые компьютеры, которые были привлечены к этой технологии, были сделаны в MIT (Массачусетский технологический институт) и DARPA (Агентство перспективных оборонных исследовательских проектов). Однако основой для создания Интернета стало изобретение ARPANET. Главная цель создания интернет-технологии заключалась в том, чтобы обеспечить платформу для коммуникации, даже если самый прямой путь был недоступен. Это было сделано для того, чтобы создать альтернативные пути коммуникации для использования такими профессионалами, как инженеры, врачи, ученые, библиотекари и компьютерные эксперты. Дальнейшее развитие получила электронная почта, открытая Рэем Томлинсоном в 1972 году. Он ввел символ @ для обозначения адреса и имени пользователя. Затем появился протокол FTP, позволяющий заинтересованным сторонам обмениваться данными между интернет-сайтами. В 1989 году эксперт Тим Бернерс придумал идеальный

интерфейс для распространения информации по всему миру. Это было то, что позже стало известно как www (World Wide Web – «Всемирная паутина») [1].

Важным фактором популяризации «Всемирной паутины» стала возможность каждому человеку внести свой вклад в процесс развития цифровой экосистемы или проявить себя в ней. Программисты, например, писали более удобные инструменты для осуществления взаимодействий внутри этой экосистемы, дизайнеры, художники – рисовали и выкладывали или продавали свои работы на обозрение анонимной публике. Однако, по-прежнему остаётся открытым вопрос сохранения авторства своей экспозиции: ведь любой может выдать код программиста, картину художника, макет дизайна или что-либо иное, в чём сохранение авторства может быть ценным, за своё произведение.

Актуальность темы связана с востребованностью системы для подтверждения авторства изображения в локализированном пространстве интернета.

Продукт разработки: информационная система поиска идентичных изображений.

Цель работы: проектирование и реализация информационной системы поиска идентичных изображений.

Задачи:

1. Изучить существующие способы защиты авторского права на результаты интеллектуальной деятельности, размещённые в сети Интернет.
2. Провести анализ инструментов, пригодных для реализации серверной и клиентской частей информационной системы поиска идентичных изображений.
3. Разработать соответствующую техническому заданию информационную систему: серверную и клиентскую части.
4. Провести апробацию разработанной информационной системы.

Глава 1. Теоретические основы разработки информационных систем защиты авторского права

1.1 Анализ способов защиты авторского права на результаты интеллектуальной деятельности, размещённые в сети Интернет

Авторское право – юридический термин, означающий право создателя на произведение, являющееся результатом его интеллектуального труда. Существует две категории таких прав: имущественные (исключительные) – когда автор имеет право использовать материал и разрешать или запрещать его использование другими лицами, а также неимущественные – автор имеет право требовать признания и указания его авторства без привязанности к месту использования. Автор имеет право передать имущественное право другому лицу, неимущественное право передавать нельзя.

Под защиту авторского права в интернете попадают различные виды контента – тексты, изображения, видеоролики и другие материалы, а также производные, то есть созданные в результате переработки или изменения оригинала. В том числе защитить можно лишь часть контента, если её можно признать результатом интеллектуального труда, например, блок в тексте статьи или персонаж в видеоролике.

В различных странах защита авторского права на законодательном уровне осуществляется по-разному. В судебной системе США претензия на нарушение авторского права рассматривается только в случае регистрации материалов в соответствующем департаменте. В России защита авторского права регулируется главой 70 Гражданского Кодекса Российской Федерации и позволяет отстоять приоритетное владение произведением в суде [2][13]. В Англии, где ранее остальных Елизаветой II была зачата система отстаивания прав на единоличное получение прибыли со своего творчества, предварительная регистрация авторского права также не осуществляется и может быть доказана в суде при возникновении спорных ситуаций.

Широко распространённый способ обозначить авторское право в интернете – явное определение авторства прямо в материале. Графические произведения помечают «водяным знаком», то есть графической отметкой, принадлежащей определённому физическому или юридическому лицу [2]. Тексты или код защищают перечислением автором, подтверждённым датой создания материала.

Английское название системы защиты авторских прав – DRM (Digital Right Management). Управление цифровыми правами в настоящее время представляет собой набор различных стратегий и устройств для предотвращения копирования и цифрового распространения медиаконтента, такого как книги, фильмы, музыка, журналы, программное обеспечение, теле- и радиопрограммы. DRM относится к типам технологий, которые встраиваются в информационные системы, то есть становятся частью их комплекса. Одной из актуальных проблем применения DRM в случаях нарушения авторских прав в Интернете является уровень анонимности, предоставляемый Интернетом: он не только затрудняет методы преследования нарушителей, но и поощряет такое преступное поведение, что было доказано в исследовании Hinduja & Ingram (2009).

Без достаточной защиты авторских прав художники, писатели и кинематографисты не захотели бы создавать новые произведения, поскольку они не смогли бы получать прибыль от них в долгосрочной перспективе из-за альтернативных каналов продаж и распространения, которые другие люди или группы людей стали бы использовать для бесплатного распространения произведений. Тысячи сайтов используют "заимствованный" контент с других сайтов в виде картинок, баннеров и даже текстовой информации.

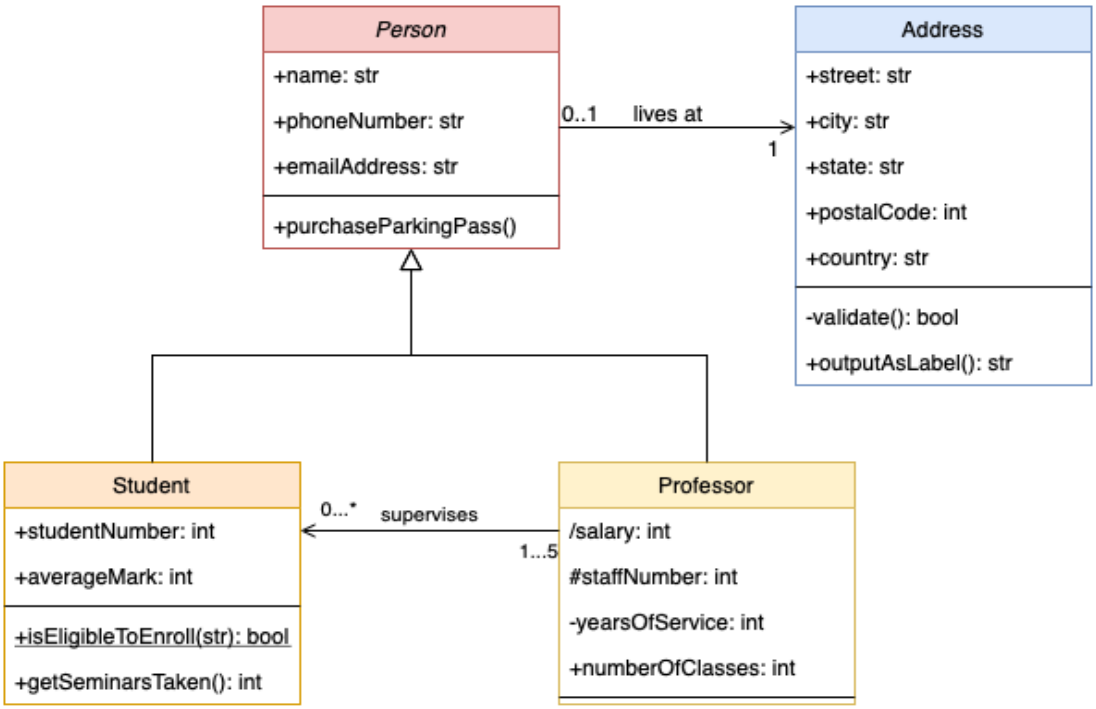
1.2 Анализ средств разработки информационной системы поиска идентичных изображений

Термин «информационная система» подразумевает прикладную программную подсистему, ориентированную на сбор, хранение, поиск и

обработку информации для принятия решений и реализации функции управления.

Техническим проектом определяется план, состоящий из нескольких этапов: разработка технических решений, документации и заданий по системе и её частям.

Для описания модели системы, то есть способа организации взаимодействий элементов друг с другом, можно использовать различные общедоступные инструменты моделирования, такие как UML (унифицированный язык моделирования), программные системы или онлайн-сервисы построения диаграмм и блок-схем, такие как, например, draw.io [3][7].



Этап разработки любой информационной системы содержит в себе совокупность действий:

- Кодирование (написание программного кода).
- Первоначальное тестирование.
- Разработка справочной системы.

Написание программного кода осуществляется с помощью языка программирования, то есть набора правил, преобразующего строки или графические элементы программы (в случае визуальных языков программирования) в различные виды машинного кода. В момент написания работы существует около 8 тысячи языков программирования, и каждый год их становится всё больше. Многие из них написаны в императивной форме, то есть как последовательность операций для выполнения, другие – в декларативной, в которой описывается желаемый результат, а не способ его достижения. Описание языка программирования обычно разделяется на два компонента - синтаксис (форма) и семантику (смысл), которые обычно определяются формальным языком. Некоторые языки определяются документом спецификации (например, язык программирования C определен стандартом ISO), в то время как другие языки (например, Perl) имеют доминирующую реализацию, которая рассматривается как эталон. Некоторые языки имеют оба варианта, при этом базовый язык определяется стандартом, а расширения, взятые из доминирующей реализации, являются общими. Различают языки программирования высокого уровня и низкого уровня. Отличительным свойством языка программирования высокого уровня является сильная абстракция от деталей компьютера: он может использовать элементы естественного языка, быть проще в использовании или автоматизировать (или даже полностью скрывать) значительные области вычислительных систем (например, управление памятью), делая процесс разработки программы более простым и понятным, чем при использовании языка более низкого уровня. Объем абстракции определяет, насколько "высокоуровневым" является язык программирования. Примерами высокоуровневых языков программирования считаются:

- JavaScript – легковесный, интерпретируемый или just-in-time компилируемый язык программирования, являющийся одной из основных технологий «World Wide Web» наряду с CSS и HTML [4].
- Python – интерпретируемый язык программирования общего назначения [5].
- Java – компилируемый, объектно-ориентированный язык программирования общего назначения, основанный на классах [6].

В качестве языка программирования описанной в техническом задании информационной системы наиболее подходящими стали Python и JavaScript из-за своей большой встроенной функциональности и полноты документации по их использованию.

Для написания программного кода чаще используются инструменты под названием «интегрированные среды разработки» (IDE – Integrated Development Environment). В случае работы над системой с помощью языка программирования Python лучше остальных подходит «PyCharm» (рис. 1), а для разработки на JavaScript – «WebStorm» (рис. 2) от JetBrains [8][9]. Существующие альтернативы интегрированной среде разработки – текстовые редакторы, такие как Notepad, Notepad++, Sublime Text; редакторы кода, такие как Visual Studio Code [10][11].

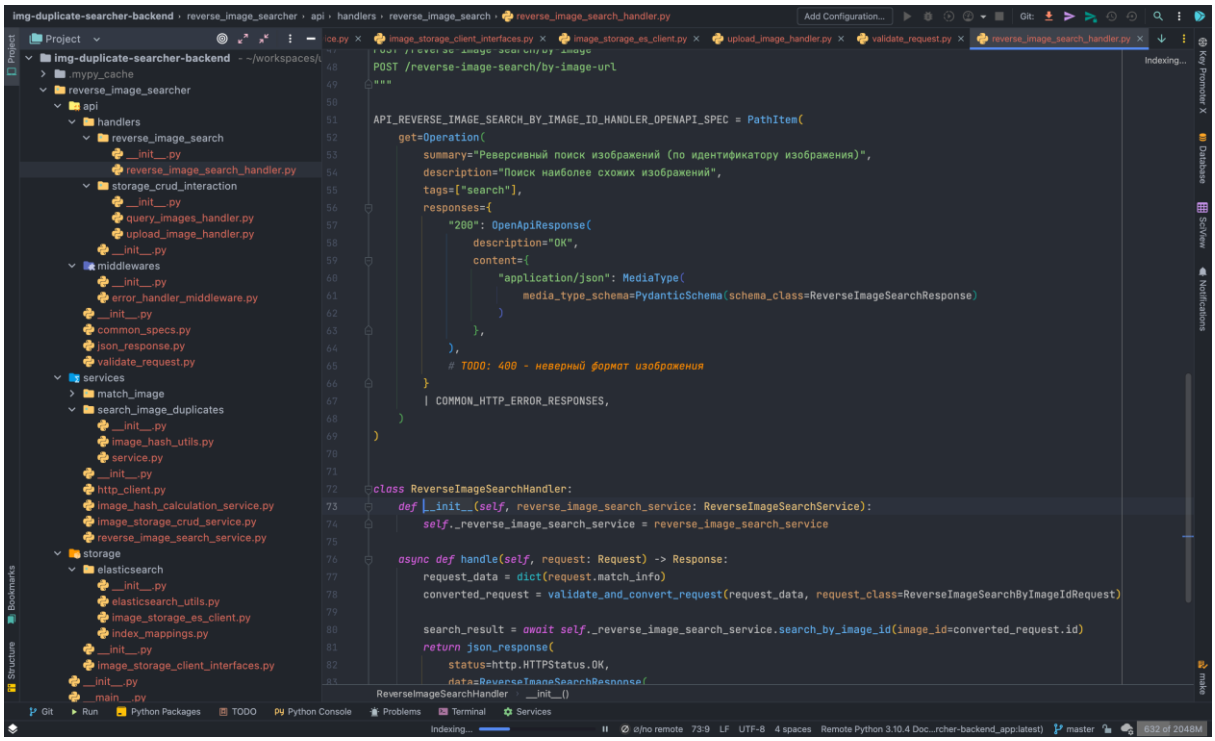


рис. 1 IDE «PyCharm»

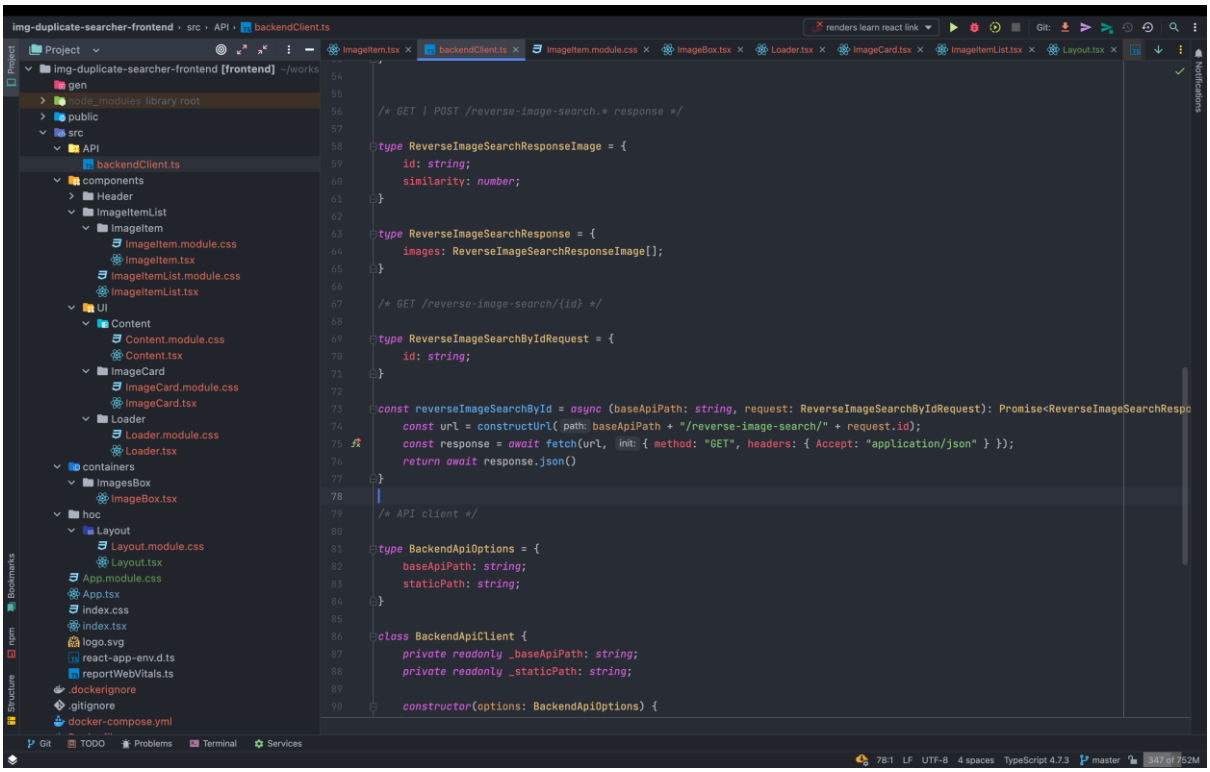


рис. 2 IDE «WebStorm»

Поисковая система — это информационно-поисковая система, предназначенная для поиска информации, хранящейся в компьютерной системе. Результаты поиска обычно представляются в виде списка и обычно называются «попаданиями». Поисковые системы помогают минимизировать время, необходимое для поиска информации, и количество информации, с которой необходимо ознакомиться, подобно другим методам управления информационной перегрузкой.

Apache Lucene — это бесплатная библиотека с открытым исходным кодом для поисковых систем, изначально написанная на языке Java Дугом Каттингом. Она поддерживается Apache Software Foundation и выпускается под лицензией Apache Software License. Lucene широко используется в качестве стандартной основы для приложений поиска, не связанных с исследованиями.

Elasticsearch — это поисковая система, основанная на библиотеке Lucene. Она представляет собой распределенную, многопользовательскую полнотекстовую поисковую систему с веб-интерфейсом HTTP и документами JSON без схем. Elasticsearch разработан на языке Java. Официальные клиенты доступны на Java, .NET (C#), PHP, Python, Apache Groovy, Ruby и многих других языках. Согласно рейтингу DB-Engines, Elasticsearch является самой популярной корпоративной поисковой системой [12].

1.3 Техническое задание на разработку информационной системы поиска идентичных изображений

Техническое задание составлено на основе ГОСТ 34.602-89.

1. Общие сведения.

1.1. Название организации-заказчика.

ФГБОУ ВО «УрГПУ».

1.2. Название продукта разработки (проектирования).

«Информационная система поиска идентичных изображений»

1.3. Назначение продукта.

Для авторов и дистрибьютеров графического контента в интернете.

1.4. Плановые сроки начала и окончания работ.

В соответствии с планом выполнения ВКР (01.09.2021 – 21.05.2022).

2. Характеристика области применения продукта.

2.1. Процессы и структуры, в которых предполагается использование продукта разработки.

Использование продукта как часть информационной системы дистрибьютера графического контента в интернете.

2.2. Характеристика персонала (количество, квалификация, степень готовности)

Персонал отсутствует.

3. Требования к продукту разработки.

3.1. Требования к продукту в целом.

Продукт должен уметь определять оригинальность изображения с учётом возможного изменения свойств изображения, таких как размер, наличие фильтров, качество.

3.2. Аппаратные требования.

- Процессор: Intel Core i5 10400;
- Объём физической памяти: 10+ ГБ;
- Объём оперативной памяти: 2 ГБ 2133 МГц.

3.3. Указание системного программного обеспечения (операционные системы, браузеры, программные платформы и т.п.).

- *Операционная система:* Ubuntu 16.04+ x64, Debian 9+ x64.

3.4. Указание программного обеспечения, используемого для реализации.

- PyCharm – интегрированная среда разработки.
- Python – верхне-уровневый язык программирования.
- Elasticsearch – программная поисковая система.
- Nginx – веб-сервер, обратный прокси-сервер.

- WebStorm – интегрированная среда разработки.
- Typescript – верхне-уровневый язык программирования.

3.5. Особенности реализации серверной и клиентской частей.

Серверная часть предоставляет REST API и имеет графический интерфейс Swagger;

Клиентская часть реализована в качестве демонстрации базовых возможностей системе и может не отображать всего функционала.

3.6. Форматы входных и выходных данных

Входные данные – управляющая информация и поисковый запрос, содержащий изображение.

Выходные данные – список изображений, которые считаются идентичными изображению, переданному во входных данных.

3.7. Порядок взаимодействия с другими системами, возможности обмена информацией.

Не предусмотрено.

3.8. Меры защиты информации.

Не предусмотрено.

4. Требования к пользовательскому интерфейсу.

4.1. Общая характеристика пользовательского интерфейса.

Пользовательский интерфейс реализуется в виде сайта с двумя страницами: страница, отображающая галерею изображений в хранилище и страница поиска идентичных изображений.

4.2. Размещение информации на экране, дизайн экрана.

Не предусмотрено

5. Требования к документированию.

5.1. Перечень сопроводительной документации.

Не предусмотрено.

5.2. Требования к содержанию отдельных документов.

Не предусмотрено.

6. Порядок сдачи-приемки продукта.

В соответствии с планом выполнения ВКР.

Глава 2. Реализация информационной системы поиска идентичных изображений

2.1 Структура информационной системы поиска идентичных изображений

Для разработки информационной системы поиска идентичных изображений была реализована следующая инфраструктура информационной системы (рис. 3):

- HTTP сервер – это сервер, принимающий и направляющий интернет-трафик по заданным в конфигурации настройкам.
- Клиентская часть – это сервер, определяющий отображение пользовательского интерфейса и взаимодействие пользователей с серверной частью, а именно, отвечающий за: отрисовку элементов интерфейса для предоставления возможности пользования человеком информационной системы поиска идентичных изображений, предварительную обработку и отправку пользовательских запросов в серверную часть информационной системы, обработку и синхронизацию ответов на отправленные запросы к серверной части и их представление в комфортном для чтения человеком виде, отдачу обратной связи пользователю на совершённые им действия, обработку ошибок сервера.
- Серверная часть – это сервер, определяющий функционал информационной системы поиска идентичных изображений, а именно, отвечающий за: взаимодействие пользователя с внешним хранилищем, взаимодействие пользователя с внутренним хранилищем, составление и отправку на хранение подписи полученного изображения, обработку и отправку на хранение полученного изображения, осуществление поиска идентичных переданному пользователем изображения среди сохранённых

изображений во внутреннем хранилище, обработку ошибок хранилища.

- Внешнее хранилище – неконтролируемый информационной системой поиска идентичных изображений сервер, отвечающий за предоставление изображений по запросу серверной части.
- Внутреннее хранилище – сервер, определяющий функционал хранения необходимых для работы информационной системы данных, а именно, отвечающий за: хранение и предоставление по запросу серверной частью системы изображений, хранение и предоставление по запросу серверной частью системы подписей изображений.

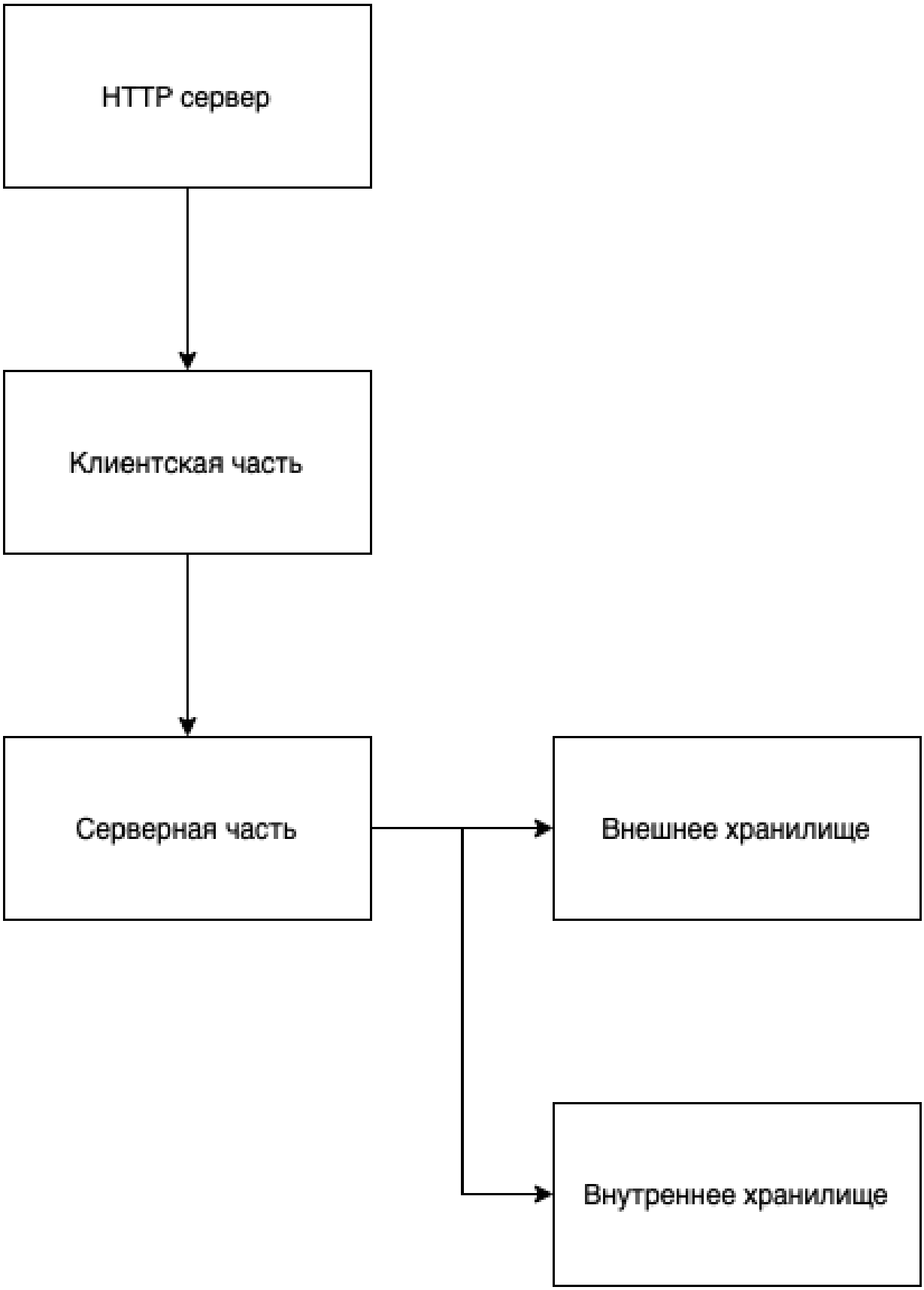


рис. 3 Инфраструктура ИС

Для реализации клиентской части информационной системы, основной функцией которого является обеспечение взаимодействия пользователя с серверной частью информационной системы, была реализована следующая архитектура кода:

- Страница с имеющимися в информационной системе изображениями (рис. 4).
    - Компонент «Шапка», содержащий компонент «Логотип».
    - Компонент «Галерея», содержащий множество компонентов «Изображение», отрисованных в виде блока с изображением и идентификатором в подписи, при нажатии на который вызывается функционал поиска идентичных этому изображений.
- Страница обратной связи поиска идентичных изображений (рис. 5).
    - Компонент «Шапка», содержащий компонент «Логотип».
    - Компонент «Выбранное для поиска изображение», содержащий компонент «Изображение» и кнопку «Сбросить», возвращающую при нажатии на страницу с имеющимися в информационной системе изображениями.
    - Компонент «Найденные изображения», содержащий множество компонентов «Изображение», отрисованных в стандартном виде и дополненных информацией о проценте идентичности изображения к запрошенному.

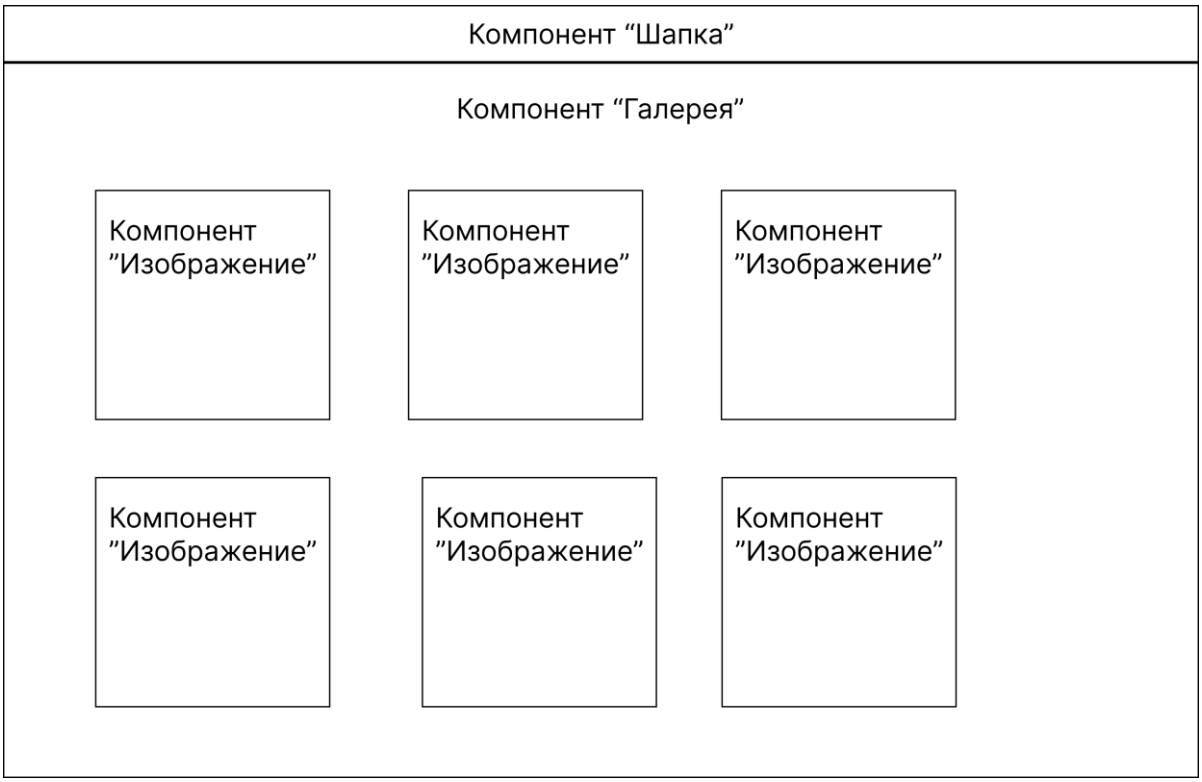


рис. 4 Страница с имеющимися в ИС изображениями

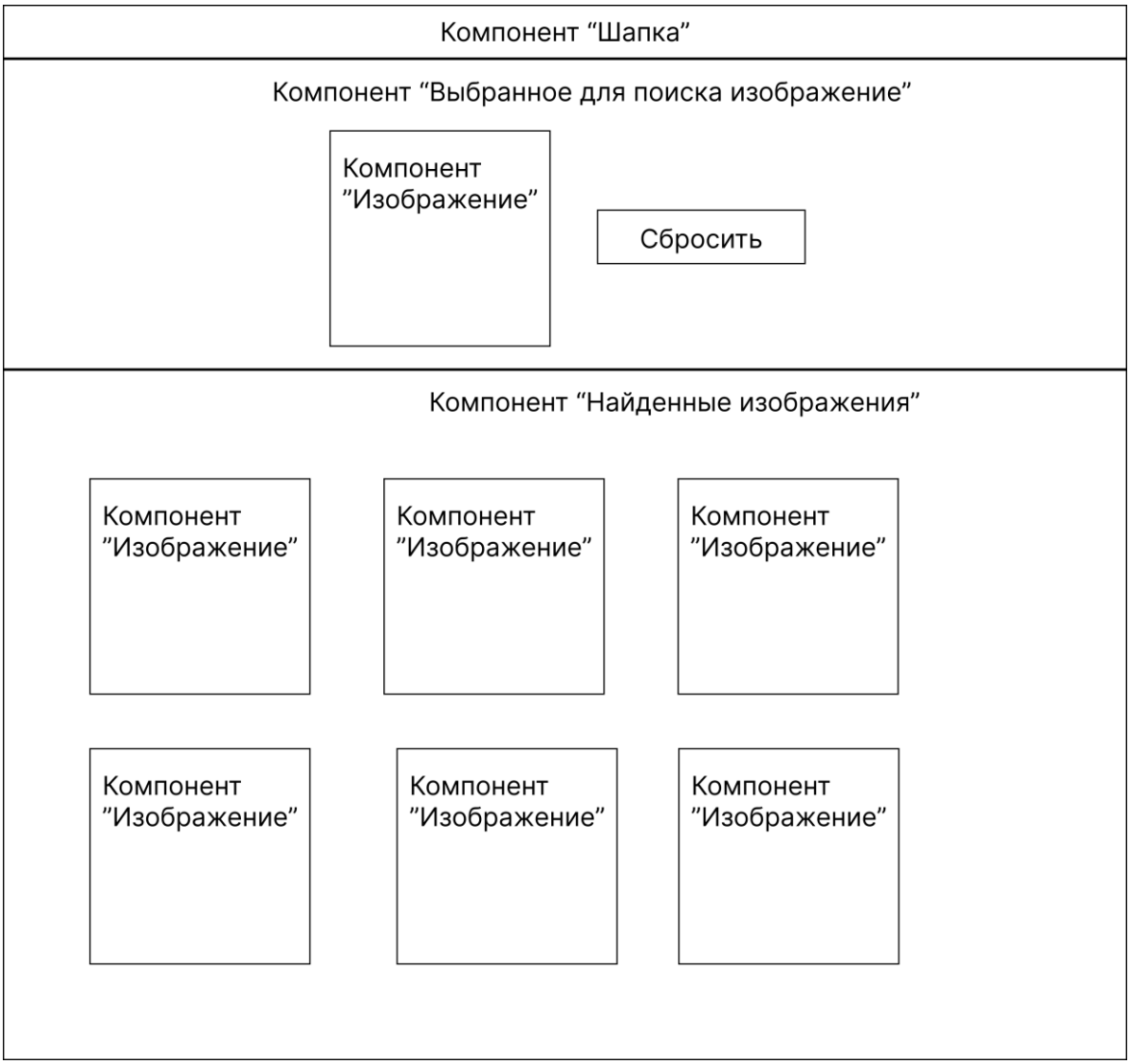


рис. 5 Страница обратной связи поиска идентичных изображений

Для реализации функционала технической части информационной системы была реализована следующая архитектура кода (рис. 6):

- Клиент взаимодействия с внешним хранилищем – класс, отвечающий за получение изображений по HTTP ссылкам во внешнее хранилище.
- Клиент взаимодействия с внутренним хранилищем – класс, отвечающий за хранение и получение изображений и их подписей по

соответствующим им идентификаторам или иным свойствам из внутреннего хранилища.

- Клиент обработки изображений – класс, отвечающий за предварительную обработку изображения (проверку корректности, сжатие, изменение размерности и конвертацию в необходимый тип) для отправки во внутреннее хранилище.
- Клиент создание подписи изображения – класс, отвечающий за создание подписи необработанного изображения с использованием алгоритма создания подписи изображения, более подробно описанного в п. 2.2.
- Маршрутизация – множество классов, отвечающих за проверку корректности полученного запроса на выполнение любого действия и соответствующих ему входных параметров, обеспечение маршрутизации запроса к необходимому ему клиенту.

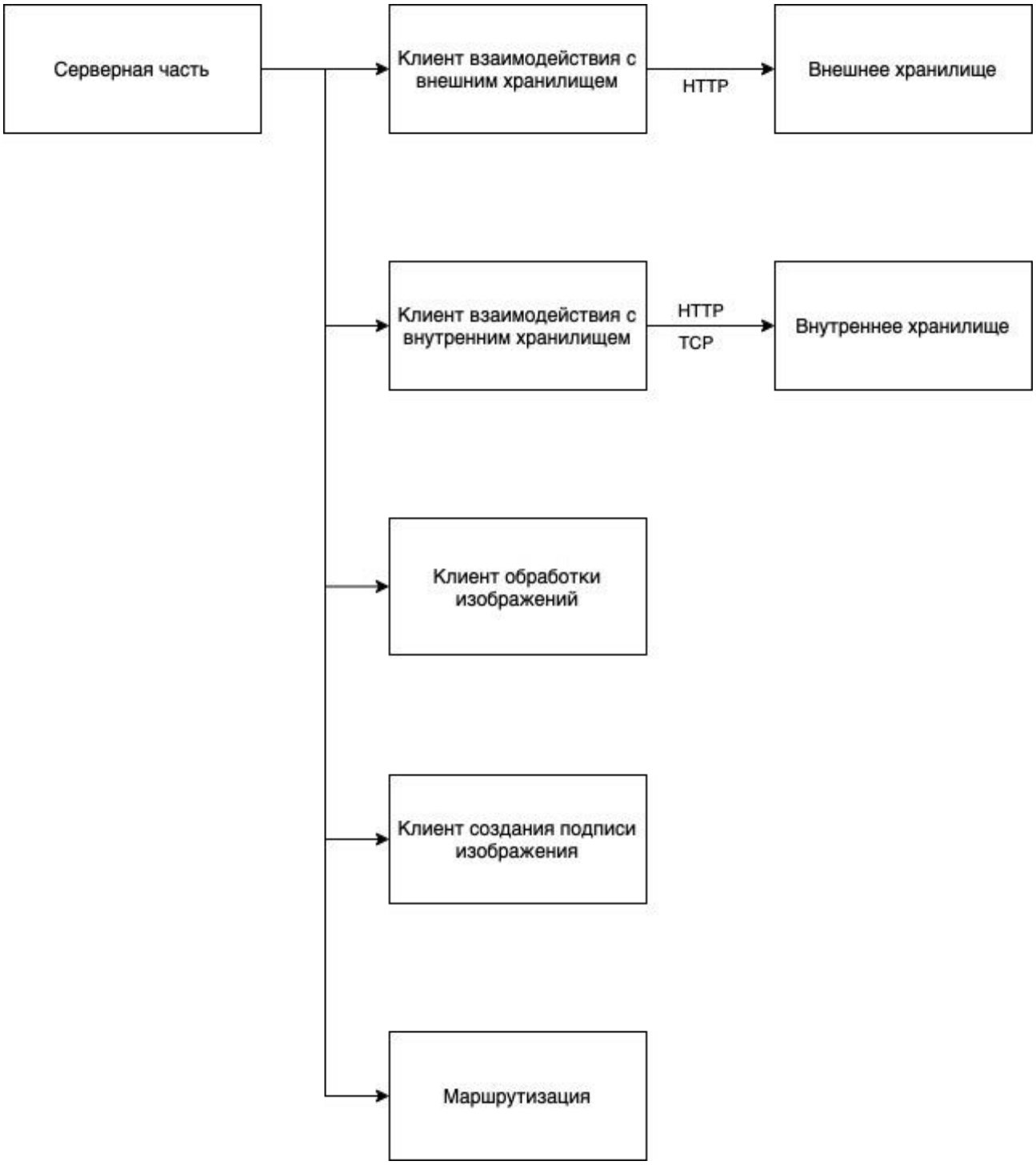


рис. 6 Архитектура кода

2.2 Алгоритм определения подписи изображения

Обнаружение дубликатов наиболее эффективно решается в два отдельных этапа. На первом этапе изображения сводятся к небольшим подписям, причем

подписи двух разных версий одного и того же изображения имеют небольшое векторное расстояние по сравнению с типичным расстоянием между подписями разных изображений. Затем, на более медленном втором этапе, производится детальное сравнение дубликатов-кандидатов, выявленных на первом этапе. Детальное сравнение изображений документов может выявить изменения столь же незначительные, как перемещение десятичной точки.

Для реализации первого этапа используется следующий алгоритм. Сигнатура изображения кодирует относительную яркость различных областей изображения; она может быть применена в общем случае к изображениям текстовых документов, линейным (например, мультфильмам) и непрерывным тональным изображениям. Хотя в литературе уже существует ряд схем подписи изображений, нет ни одной подписи, которая была бы применима к такому широкому классу изображений. Основное ограничение такой подписи заключается в том, что она не рассчитана на обработку большого количества обрезки или поворота изображения. Такой выбор дизайна подходит для баз данных изображений документов и веб-поиска.

В работе адаптирован метод О'Гормана и Рабиновича для вычисления подписи на фотографии удостоверения личности: метод размещает сетку точек на фотографии удостоверения личности и вычисляет вектор из 8 бит для каждой точки в сетке. Каждый бит записывает, является ли окрестность этой точки темнее или светлее, чем окрестности ее 8 диагональных или ортогональных соседей. В сценарии использования подпись изображения, вычисленная по фотографии, сравнивается с эталонной подписью, записанной на идентификационной карте, и подписывается с помощью криптографии с открытым ключом [14].

Еще одна технология, которую стоит рассмотреть, заключается в том, что содержание изображения диктует окрестности, используемые в подписи. Шмид и Мор используют детектор угловых точек для определения "интересных точек" на изображении. Подпись включает в себя статистический обзор окрестностей

каждой интересной точки. По сравнению с методами на основе сетки, такими как описанный в текущей работе, этот подход имеет преимущество: возможность обрабатывать большие объемы обрезки изображений, а если статистика в каждой интересной точке инвариантна к вращению, то ещё и возможность обрабатывать произвольные объемы вращения изображения. С другой стороны, для получения надежной подписи требуется несколько сотен «интересных» точек, и этот подход не подходит при работе с текстовыми документами или линейными изображениями, которые могут содержать тысячи «интересных» точек с очень похожей статистикой [15].

В литературе также имеются специализированные методы подписи изображений документов, использующие оптическое распознавание символов или специфические для документа характеристики, такие как расположение абзацев, длина строк и форма букв. Результаты этих специализированных методы будут более точными, чем описанный в этой работе, однако, менее эффективными для первого этапа фильтрации, поскольку, уменьшая количество возможных пар дубликатов в сотни или тысячи раз, для второго этапа можно использовать более точные и детальные алгоритмы полного сопоставления изображений.

В идеальной системе сигнатура должна быть достаточно мала для эффективного поиска ближайших соседей, достаточно чувствительна для эффективной фильтрации базы данных на предмет возможных дубликатов и при этом достаточно надежна для поиска дубликатов, которые были изменены в размере, отсканированы или сжаты с потерями.

Алгоритм, используемый в реализуемой системы (рис. 7), был вдохновлён алгоритмом подписи О'Гормана, но отличается почти на каждом шаге. В него добавлен функционал для обработки обрезания изображения, а также расширены двухуровневые значения относительной темноты, вместо просто «темнее» или «светлее» – до пяти уровней для обеспечения большей чувствительности и надежности.

```python
def generate_signature(self, image: bytes) -> np.ndarray:
    # Шаг 1: Загрузка изображения как массив уровней серого
    im_array = self.preprocess_image(image)

    # Шаг 2a: Определение границ для обрезки
    image_limits: Optional[list[tuple[int, int]]]
    if self.crop_percentiles is not None:
        image_limits = self.crop_image(
            im_array,
            lower_percentile=self.lower_percentile,
            upper_percentile=self.upper_percentile,
            fix_ratio=self.fix_ratio,
        )
    else:
        image_limits = None

    # Шаг 2b: Расчёт центральных точек сетки
    x_coords, y_coords = self.compute_grid_points(im_array, n=self.n, window=image_limits)

    # Шаг 3: Расчёт уровней серого для каждой P x P квадрата с центром в каждой точке сетки
    avg_grey = self.compute_mean_level(im_array, x_coords, y_coords, P=self.P)

    # Шаг 4a: Расчёт массива различий для каждой точки сетки с каждой соседней точкой сетки
    diff_mat = self.compute_differentials(avg_grey, diagonal_neighbors=self.diagonal_neighbors)

    # Шаг 4b: Сужаем различия до 2n+1 значений
    self.normalize_and_threshold(diff_mat, identical_tolerance=self.identical_tolerance, n_levels=self.n_levels)

    # Шаг 5: Преобразуем массив значений в подпись
    return np.ravel(diff_mat).astype("int8")
```

рис. 7 Алгоритм создания подписи изображения

Шаг 1. Если изображение цветное, сначала проводится его преобразование в изображение, имеющее 8-битную шкалу серого цвета, с помощью стандартных алгоритмов преобразования цвета, где чистый белый цвет обозначается 255, а чистый черный – 0.

```python
    @staticmethod
    def preprocess_image(image: bytes) -> np.ndarray:
        try:
            img = Image.open(BytesIO(image))
        except IOError:
            raise CorruptImageError

        return rgb2gray(np.asarray(img.convert("RGB"), dtype=np.uint8))
```

рис. 8 Алгоритм создания подписи изображения (шаг 1)

Шаг 2. На изображение накладывается сетка точек размерностью 9 x 9 (для больших баз данных может использоваться более крупная сетка, например, 11 x 11, что позволит улучшить фильтрацию на первом этапе). Сетка

определяется таким образом, чтобы она была устойчива к умеренному обрезанию изображения, исходя из предположения, что при таком обрезании обычно удаляются относительно лишенные особенностей части изображения, например поля изображения документа или темный низ картины. Для каждого столбца изображения вычисляется сумма абсолютных значений различий между соседними пикселями в этом столбце. Затем вычисляется общая сумма всех столбцов и изображение обрезается по столбцам на 5% и 95%, то есть по таким столбцам, чтобы 5% от общей суммы различий находилось по обе стороны обрезанного изображения. Аналогичным образом обрезаются строки изображения (используя суммы исходных необрезанных строк).

Концептуально, обрезанное изображение делится на сетку блоков размерностью 10 x 10. Каждая внутренняя точка сетки округляется до ближайшего пиксел, тем самым создавая сетку точек на изображении размерностью 9 x 9.

```
68          # Шаг 2a: Определение границ для обрезки
69          image_limits: Optional[list[tuple[int, int]]]
70          if self.crop_percentiles is not None:
71              image_limits = self.crop_image(
72                  im_array,
73                  lower_percentile=self.lower_percentile,
74                  upper_percentile=self.upper_percentile,
75                  fix_ratio=self.fix_ratio,
76              )
77          else:
78              image_limits = None
79
80          # Шаг 2b: Расчёт центральных точек сетки
81          x_coords, y_coords = self.compute_grid_points(im_array, n=self.n, window=image_limits)
```

```python
    @staticmethod
    def compute_grid_points(
        image: np.ndarray, n: int = 9, window: Optional[list[tuple[int, int]]] = None
    ) -> tuple[np.ndarray, np.ndarray]:

        # Если ограничивающие значения не переданы, будем использовать всё исходное изображение
        if window is None:
            window = [(0, image.shape[0]), (0, image.shape[1])]

        x_coords = np.linspace(window[0][0], window[0][1], n + 2, dtype=int)[1:-1]
        y_coords = np.linspace(window[1][0], window[1][1], n + 2, dtype=int)[1:-1]

        return x_coords, y_coords
```

рис. 9 Алгоритм создания подписи изображения (шаг 2)

Шаг 3. В каждой точке сетки вычисляется средний уровень серого для квадрата размерностью P x P с центром в точке сетки. Формула для вычисления размерности квадрата: $P = max\{2, \lfloor 0.5 + min\{n, m\} \div 20 \rfloor\}$, где m и n – размеры изображения в пикселях. Квадраты имеют мягкие цветовые края, что означает, что вместо того, чтобы использовать уровни серого цвета пикселя, можно использовать среднее значение блока с центром в этом пикселе.

```python
    @staticmethod
    def compute_mean_level(
        image: np.ndarray, x_coords: np.ndarray, y_coords: np.ndarray, P: Optional[int] = None
    ) -> np.ndarray:

        if P is None:
            P = max([2, int(0.5 + min(image.shape) / 20.0)])  # на каждую страницу

        avg_grey = np.zeros((x_coords.shape[0], y_coords.shape[0]))

        for i, x in enumerate(x_coords):  # дорогая по ресурсам реализация
            lower_x_lim = int(max([x - P / 2, 0]))
            upper_x_lim = int(min([lower_x_lim + P, image.shape[0]]))
            for j, y in enumerate(y_coords):
                lower_y_lim = int(max([y - P / 2, 0]))
                upper_y_lim = int(min([lower_y_lim + P, image.shape[1]]))

                avg_grey[i, j] = np.mean(
                    image[lower_x_lim:upper_x_lim, lower_y_lim:upper_y_lim]
                )

        return avg_grey
```

рис. 10 Алгоритм создания подписи изображения (шаг 3)

Шаг 4. Для каждой точки сетки вычисляется массив, состоящий из 8-ми элементов, каждый из которых даёт сравнение среднего уровня серого цвета

квадрата точки сетки и восьми ее соседей. Результат сравнения может быть «намного темнее», «темнее», «такой же», «светлее» или «намного светлее», представленные в числовом выражении как -2, -1, 0, 1 и 2, соответственно. «Одинаковыми» считаются те средние значения, которые отличаются не более чем на 2 по шкале от 0 до 255. Устанавливается граница между «намного темнее» и «темнее» так, чтобы эти два значения были одинаково распределены; то же самое делается для «светлее» и «намного светлее». Этот шаг обоснован тем, что «одинаковый» может быть очень распространен в изображениях с плоским фоном (например, в текстовых документах), и поэтому его не следует включать в выравнивание гистограммы, применяемое к другим значениям. Точки сетки в первой или последней строке или столбце имеют менее 8 соседей (всего таких точек – 104); для упрощения результаты сравнения таких точек заполняются нулями.

```python
    @staticmethod
    def compute_differentials(grey_level_matrix: np.ndarray, diagonal_neighbors: bool = True) -> np.ndarray:
        right_neighbors = -np.concatenate(
            (np.diff(grey_level_matrix), np.zeros(grey_level_matrix.shape[0]).reshape((grey_level_matrix.shape[0], 1))),
            axis=1,
        )
        left_neighbors = -np.concatenate((right_neighbors[:, -1:], right_neighbors[:, :-1]), axis=1)
        down_neighbors = -np.concatenate(
            (
                np.diff(grey_level_matrix, axis=0),
                np.zeros(grey_level_matrix.shape[1]).reshape((1, grey_level_matrix.shape[1])),
            )
        )
        up_neighbors = -np.concatenate((down_neighbors[-1:], down_neighbors[:-1]))

        if diagonal_neighbors:
            # реализация будет работать только для m x m квадрата сетки
            diagonals = np.arange(-grey_level_matrix.shape[0] + 1, grey_level_matrix.shape[0])
            upper_left_neighbors = sum(
                [np.diagflat(np.insert(np.diff(np.diag(grey_level_matrix, i)), 0, 0), i) for i in diagonals]
            )
            lower_right_neighbors = -np.pad(upper_left_neighbors[1:, 1:], (0, 1), mode="constant")  # type: ignore
            # разворачиваем анти-диагональные различия
            flipped = np.fliplr(grey_level_matrix)
            upper_right_neighbors = sum(
                [np.diagflat(np.insert(np.diff(np.diag(flipped, i)), 0, 0), i) for i in diagonals]
            )
            lower_left_neighbors = -np.pad(upper_right_neighbors[1:, 1:], (0, 1), mode="constant")  # type: ignore

            return np.dstack(
                (
                    upper_left_neighbors,
                    up_neighbors,
                    np.fliplr(upper_right_neighbors),
                    left_neighbors,
                    right_neighbors,
                    np.fliplr(lower_left_neighbors),
                    down_neighbors,
                    lower_right_neighbors,
                )
            )

        return np.dstack((up_neighbors, left_neighbors, right_neighbors, down_neighbors))
```

рис. 11 Алгоритм создания подписи изображения (шаг 4.1)

```python
    @staticmethod
    def normalize_and_threshold(
        difference_array: np.ndarray, identical_tolerance: float = 2 / 255.0, n_levels: int = 2
    ):
        # очень близкие значения считаем эквивалентными
        mask = np.abs(difference_array) < identical_tolerance
        difference_array[mask] = 0.0

        if np.all(mask):
            # если изображение не имеет существенных признаков
            return None

        positive_cutoffs = np.percentile(difference_array[difference_array > 0.0], np.linspace(0, 100, n_levels + 1))
        negative_cutoffs = np.percentile(difference_array[difference_array < 0.0], np.linspace(100, 0, n_levels + 1))

        for level, interval in enumerate([positive_cutoffs[i : i + 2] for i in range(positive_cutoffs.shape[0] - 1)]):
            difference_array[(difference_array >= interval[0]) & (difference_array <= interval[1])] = level + 1

        for level, interval in enumerate([negative_cutoffs[i : i + 2] for i in range(negative_cutoffs.shape[0] - 1)]):
            difference_array[(difference_array <= interval[0]) & (difference_array >= interval[1])] = -(level + 1)

        return None
```

рис. 12 Алгоритм создания подписи изображения (шаг 4.2)

Шаг 5. Подпись изображения — это конкатенация массивов, состоящих из 8-ми элементов, соответствующих точкам сетки, упорядоченных слева направо и сверху вниз. Таким образом, подпись представляет собой вектор длины 648. Хранение таких векторов осуществляется в 648-байтовых массивах, но поскольку некоторые записи в первых и последних строках и столбцах известны как нули, а также каждый байт используется для хранения только 5 значений, подпись может быть представлена как $544 \times \log_2 5 = 1264$ бита.

```
92      # Шаг 5: Преобразуем массив значений в подпись
93      return np.ravel(diff_mat).astype("int8")
```

рис. 13 Алгоритм создания подписи изображения (шаг 5)

2.3 Алгоритм нахождения ближайших соседей изображения по определённой подписи

Нормализованное расстояние между двумя сигнатурами изображений (u и v) $\Delta(u, v) = \frac{\|u-v\|}{\|u\|+\|v\|}$, где $\|x\|$ представляет собой L_2 норму вектора, то есть его Евклидово расстояние. Относительно нормы L_1 (или расстояния Хэмминга), L_2 придаёт большее значение более существенным различиям; это целесообразно, поскольку разница в 1 в координатах может быть вызвана округлением, но разница в 2 должна считаться значимой.

Чтобы сравнить расстояние с фиксированным порогом, недостаточно просто использовать формулу $\|u - v\|$, необходимо его нормализовать. Пороговое значение, равное 0.6, кажется хорошим выбором, поскольку дает обоснованную достоверность преобразования изображений, не допуская большого количества ложных совпадений. Для текстовых документов лучше использовать немного более низкий порог, поскольку их подписи обычно имеют гораздо большее количество нулей, чем подписи непрерывных тоновых изображений. Решение, которое позволяет использовать порог 0.6 для любого типа изображений, заключается в изменении базовой арифметики: считать разницу между 0 и 2 или -2 в формуле $\|u - v\|$ как 3.

По запрошенной подписи u теперь необходим способ найти все подписи v в базе данных на нормализованном расстоянии 0.6 или меньше. Обычный способ решения этой задачи заключается в разбиении подписей на (возможно, несмежные и пересекающиеся) «слова» из k байтов и поиске всех v, которые точно совпадают с u хотя бы в одном «слове». Для каждого N «слова» индексируемая таблица указывает на все подписи с каждым данным «словом». Параметры k и N выбираются так, чтобы любая v в пределах нормализованного расстояния от 0.6 обязательно совпадала хотя бы с одним «словом», а также не требовалось много места для хранения индексов таблиц. Этот метод решает проблему поиска ближайшего соседа в высоких размерностях путем нахождения точных совпадений в ряде более низкоразмерных проекций. Существуют также более сложные алгоритмы и анализ поиска ближайшего соседа в высоких измерениях.

В рассматриваемом случае нормализованное расстояние 0.6 позволяет u и v различаться на каждом байте, поэтому нет наиболее удачного выбора длины слова k. Рекомендуем решением является объединить в индексных таблицах –2 и –1 так, чтобы представлять их через –1, и аналогично объединить 2 и 1. Записи в таблице, хотя и индексируются «словами» из 3-х возможных букв, все равно указывают на более точные 5-буквенные сигнатуры. В таком виде разумным выбором будет $k = 10$ и $N = 100$. Подпись v с порогом в 0.6 с большой вероятностью совпадет с каким-то «словом» u. Если каждая буква в объединенной версии «слова» v имеет вероятность совпадения с соответствующей буквой в объединенной версии «слова» u 0.75, то каждое слово имеет вероятность совпадения $0.75^{10} \approx 0.05$, и, следовательно, вероятность совпадения хотя бы одного из 100 «слов» составляет около $1 - 0.05^{100}$, что больше 0.993. Однако случайная подпись v вряд ли совпадет с каким-то «словом». Если каждая буква в объединённой v имеет вероятность совпадения с соответствующей буквой в объединённой u 0.5, то каждое слово имеет вероятность совпадения $0.5^{100} \approx 0.001$, и, следовательно, вероятность того, что

хотя бы одно из 81 слова совпадет, составляет около $1 - 0.999^{100} \approx 0.1$. Таким образом, только около 10% подписей в базе данных нуждаются в вычислении их фактических нормализованных расстояний к u. Поскольку вычисление нормализованных расстояний происходит очень быстро, это вполне допустимо для базы данных с миллионами изображений. Большие значения дают большую дискриминационную способность, но занимают больше места в памяти. При увеличении значения N, пороговые к 0.6 подписи будут иметь гораздо больше совпадений слов, чем случайные.

Следует обратить внимание, что при использовании метода поиска ближайших соседей общий процесс обнаружения дубликатов изображений становится трехуровневой схемой. Индексация по словам позволяет найти пары подписей-кандидатов, которые, в свою очередь, приводят к парам изображений-кандидатов.

2.4 Демонстрация работы информационной системы поиска идентичных изображений

2.4.1 Демонстрация работы клиентской части информационной системы

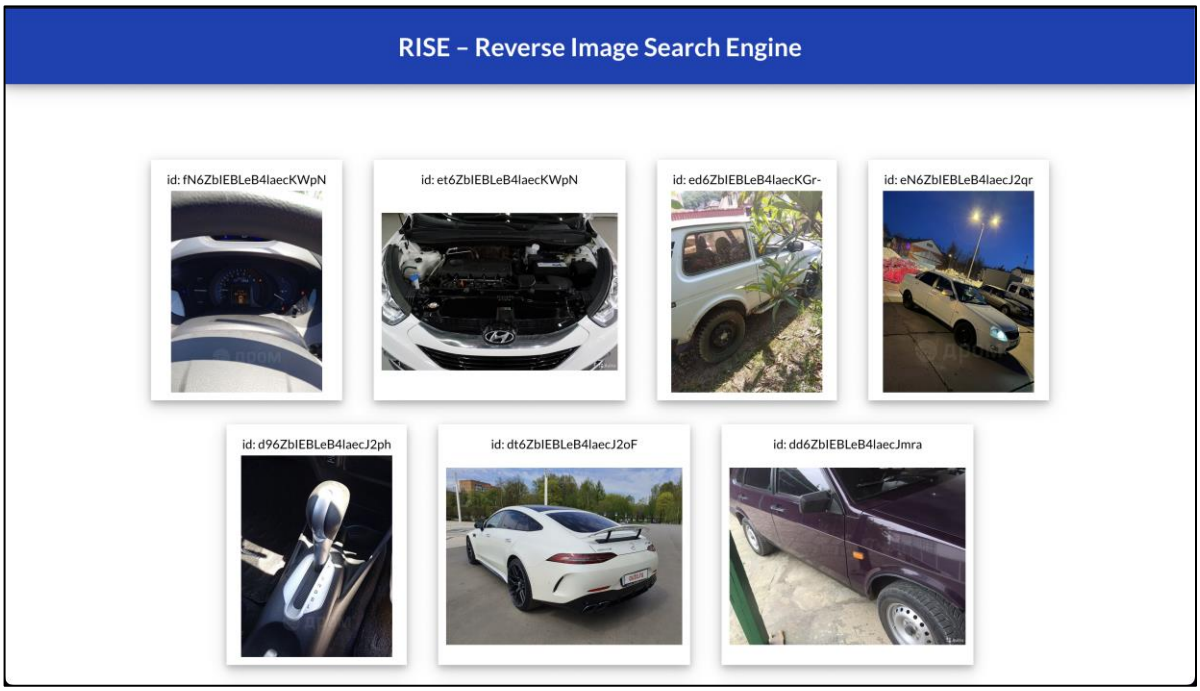


рис. 14 Демонстрация галереи

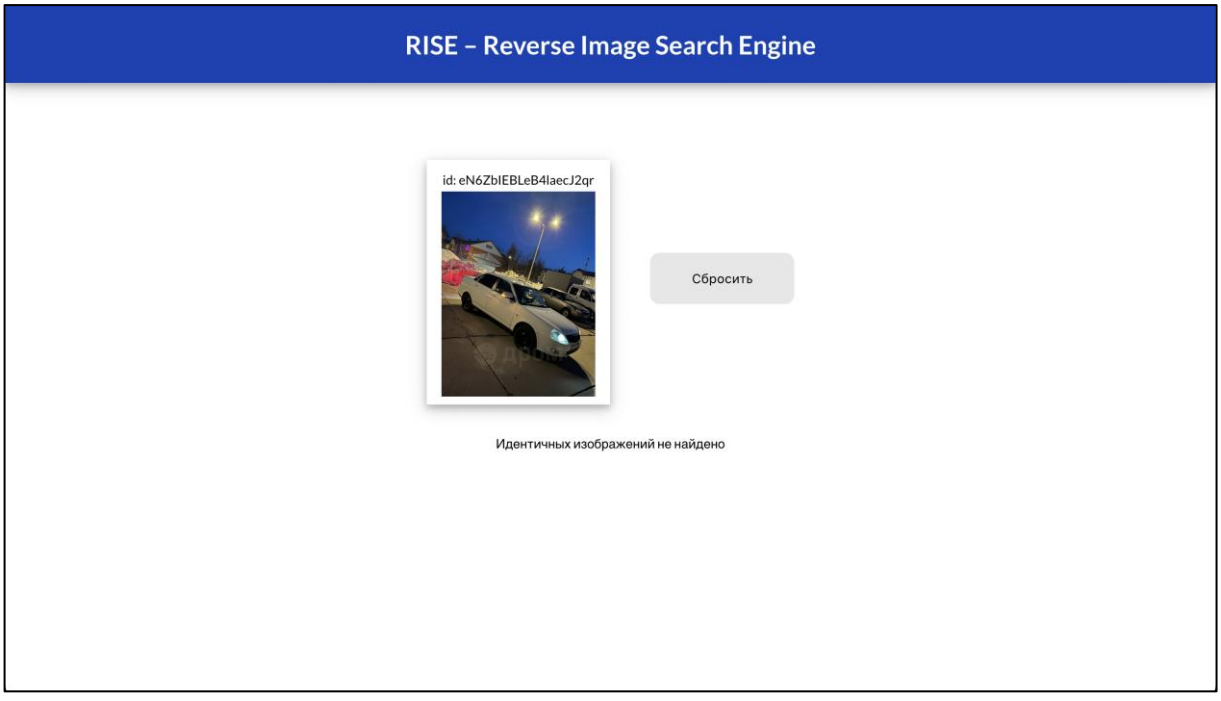


рис. 15 Демонстрация поиска идентичных изображений с результатом «не найдено»

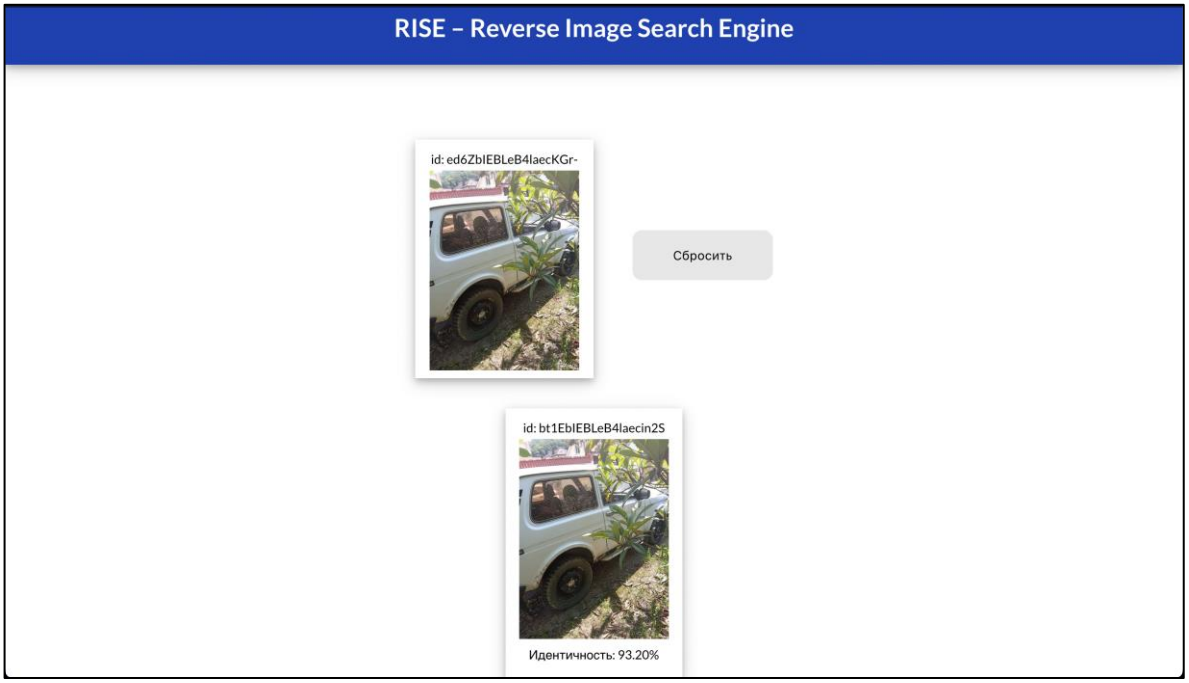


рис. 16 Демонстрация поиска идентичных изображений с результатом «найдено»

2.4.2 Демонстрация работы серверной части информационной системы

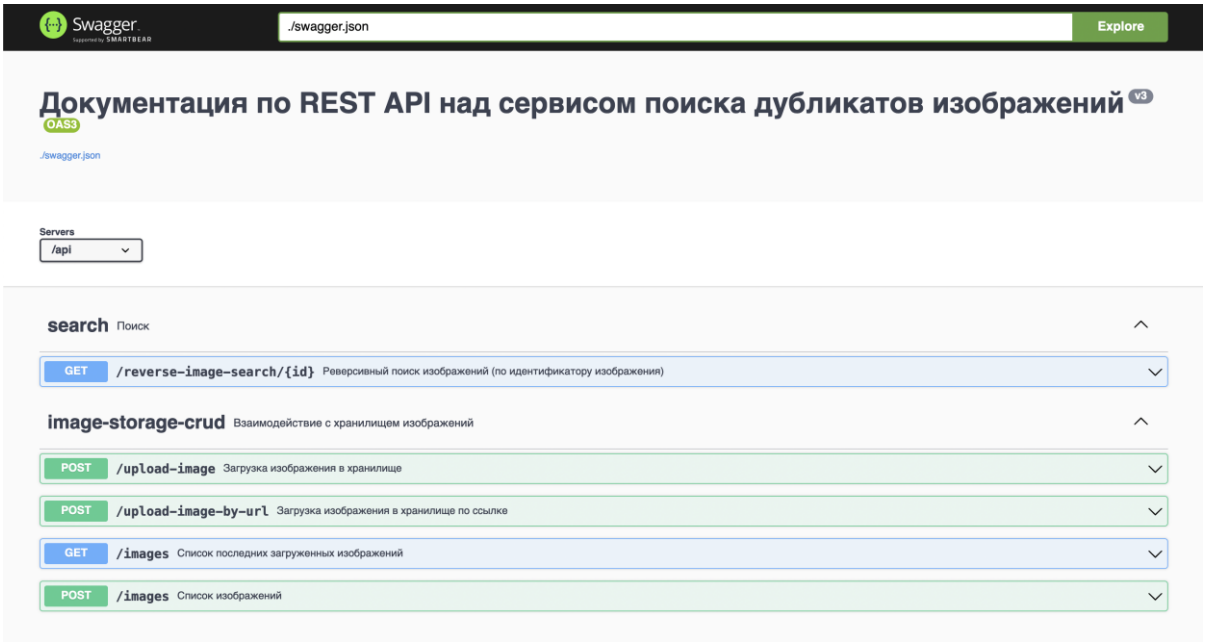


рис. 17 Демонстрация интерфейса серверной части ИС

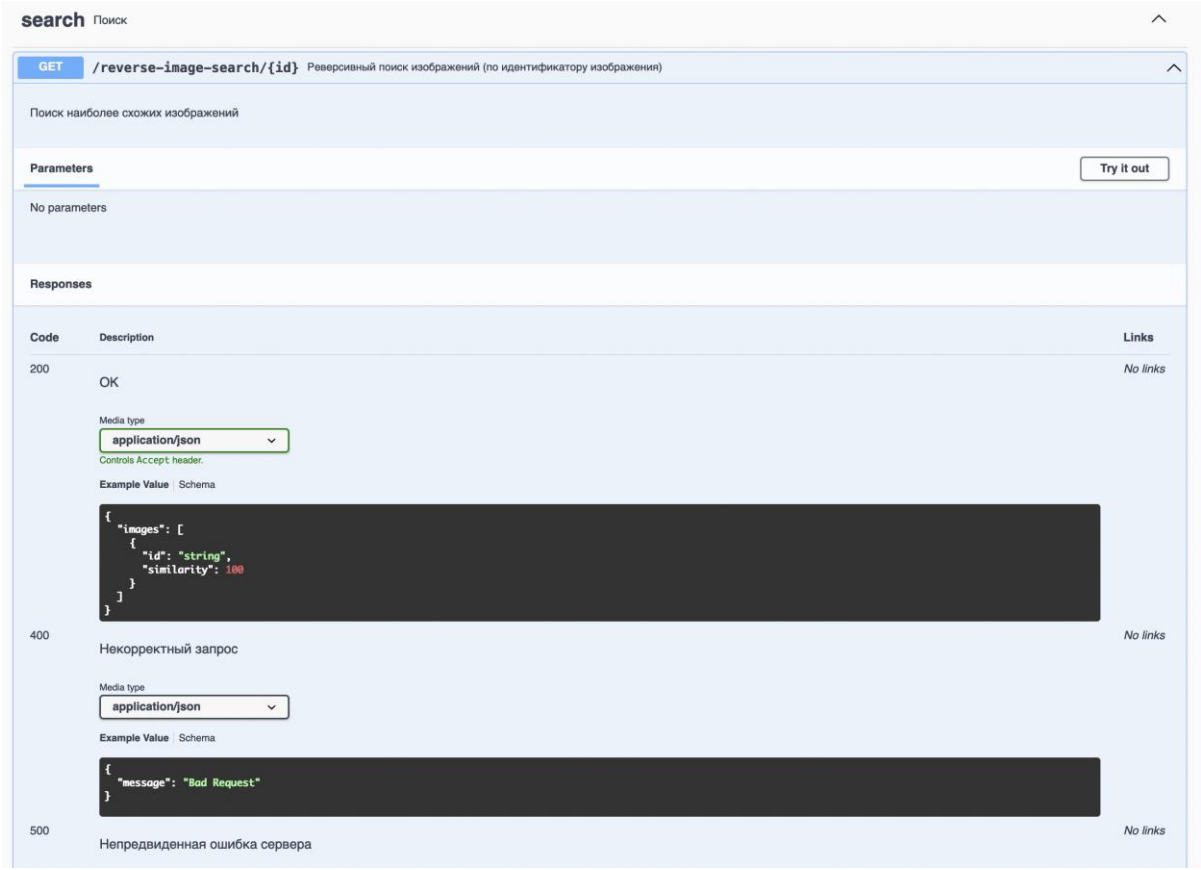


рис. 18 Демонстрация спецификации метода поиска идентичных изображений в интерфейсе серверной части ИС

2.4.3 Демонстрация работы внутреннего хранилища подписей изображений

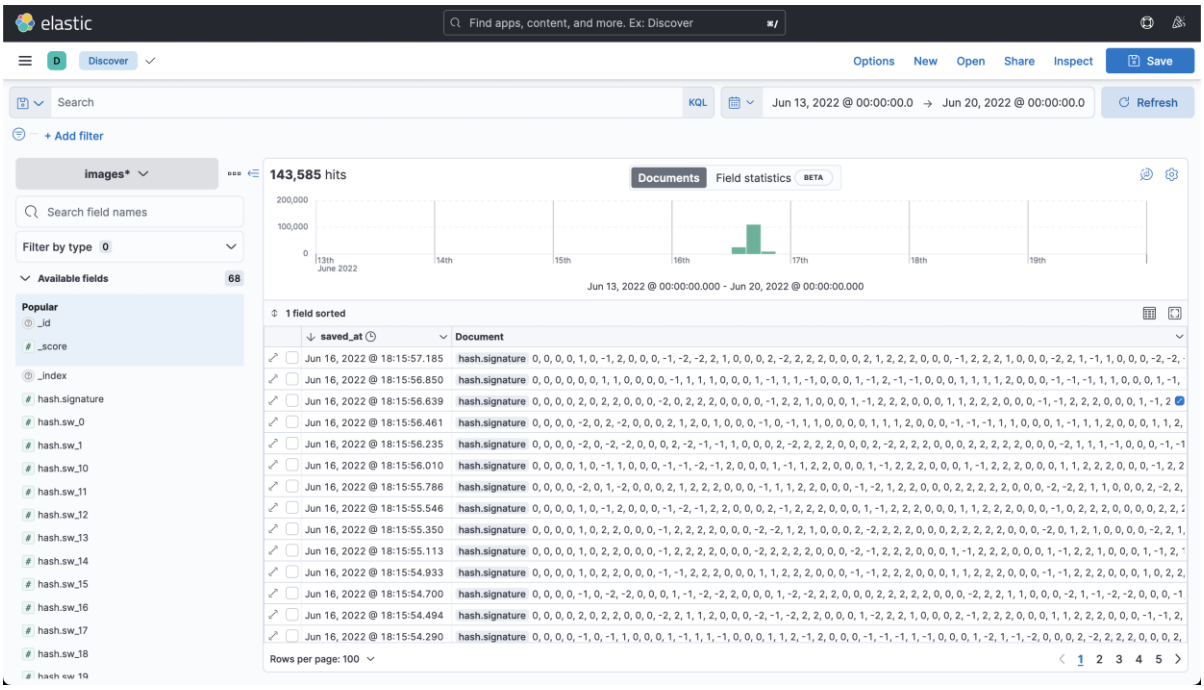


рис. 19 Демонстрация интерфейса хранилища подписей изображений

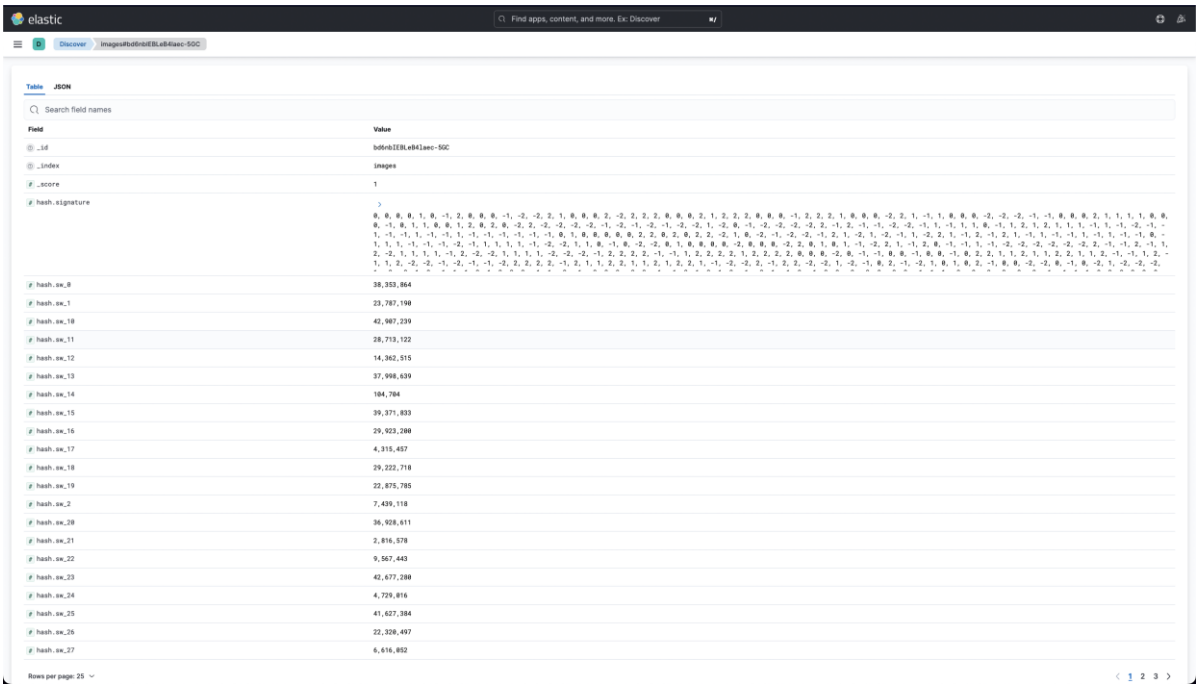


рис. 20 Демонстрация структуры хранения подписи изображения

2.5 Экспертная оценка разработанной информационной системы поиска идентичных изображений

Разработанная информационная система была представлена для анализа и оценки экспертам: трём квалифицированным разработчикам в компании SpectrumData с опытом работы более 5 лет в сфере разработки и управления информационных систем. Экспертами было проведено трёхдневное исследование информационной системы по следующим критериям по 5-тибальной шкале (где 0 – отсутствует, 5 – отлично):

К1. Простота освоения и интеграции информационной системы.

К2. Стабильность работы информационной системы.

К3. Соответствие реализованного функционала описанному в техническом задании.

Эксперт	К1	К2	К3	Рекомендации по улучшению
Белобоков Михаил	4	5	5	Для упрощения интеграции стоит добавить возможность массового заполнения подписей изображений по списку ссылок на них,

				поскольку во многих компаниях изображения хранятся на собственном отдельном ресурсе (файловой системе с выходом в интернет или облаке).
Лямин Леонид	5	5	5	Для уменьшения порога понимания системы тестировщиками стоит добавить возможность загружать изображение из пользовательского интерфейса сайта, реализованного в клиентской части, а не только HTTP запросом на сервер.
Устинов Данил	5	5	5	Для возможности использования системы в других компаниях следует провести нагрузочное тестирование и выяснить, сколько запросов в секунду сможет обрабатывать такой алгоритм и выбранная поисковая база данных.

Заключение

При выполнении данной выпускной работы была спроектирована и реализована автоматизированная информационная система поиска идентичных изображений в соответствии с техническим заданием:

1. Изучил существующие способы защиты авторского права на результаты интеллектуальной деятельности, размещённые в сети Интернет.
2. Провёл анализ инструментов, пригодных для реализации серверной и клиентской частей информационной системы поиска идентичных изображений: рассмотрел интегрированные среды разработки, хранилища изображений, поисковые хранилища, языки программирования, средства создания сервера.
3. Разработал соответствующую техническому заданию информационную систему: реализована серверная и клиентская части, создана база данных изображений, имитирующая общее хранилище изображений, построена инфраструктура, обеспечивающая работоспособность комплекса информационной системы.
4. Провёл апробацию разработанной информационной системы: опросил 3-х квалифицированных в разработке информационной системы человек и получил положительные отзывы.

Таким образом, следует считать, что результаты разработки соответствуют всем требованиям технического задания, работа носит законченный характер, цель работы достигнута и задачи выполнены.

Список используемой литературы

1. World Wide Web. – URL: https://www.britannica.com/topic/World-Wide-Web (дата обращения: 08.04.2022).

2. Цифровой водяной знак. – URL: https://ru.wikipedia.org/wiki/Цифровой_водяной_знак (дата обращения: 08.04.2022)

3. Unified Modeling Language. – URL: https://www.uml.org/ (дата обращения: 08.04.2022).

4. JavaScript. – URL: https://www.javascript.com/ (дата обращения: 08.04.2022).

5. Python. – URL: https://www.python.org/ (дата обращения: 08.04.2022).

6. Java. – URL: https://www.java.com/ru/ (дата обращения: 08.04.2022).

7. Draw.io – Diagrams for Confluence and Jira. – URL: https://drawio-app.com/ (дата обращения: 08.04.2022).

8. PyCharm: The Python IDE for Professional Developers by JetBrains. – URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения: 08.04.2022).

9. WebStorm: The smartest JavaScript IDE, by JetBrains. – URL: https://www.jetbrains.com/ru-ru/webstorm/ (дата обращения: 08.04.2022).

10. Sublime Text. – URL: https://www.sublimetext.com/ (дата обращения: 08.04.2022).

11. Visual Studio Code - Code Editing. Redefined. – URL: https://code.visualstudio.com/ (дата обращения: 08.04.2022).

12. Free and Open Search: The Creators of Elasticsearch, ELK & Kibana | Elastic. – URL: https://www.elastic.co/ (дата обращения: 08.04.2022).

13. Авторское право: гл. 70 Гражданского Кодекса Российской Федерации (часть четвёртая) от 18.12.2006 №230-ФЗ (ред. от 11.06.2022, с изм. от 16.06.2022) [Электронный ресурс]. – URL:

https://www.consultant.ru/document/cons_doc_LAW_64629/0b318126c43879a845405f1fb1f4342f473a1eda/ (дата обращения 20.06.2022).

14. L. O’Gorman Secure identification documents via pattern recognition and public-key cryptography / L. O'Gorman and I. Rabinovich // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 1998. – p. 1097-1102. doi: 10.1109/34.722623.

15. C. Schmid Image retrieval using local characterization / C. Schmid and R. Mohr // Proceedings of 3rd IEEE International Conference on Image Processing. – 1996. – p. 781-784 vol. 2. doi: 10.1109/ICIP.1996.561020.

Приложения

Приложение 1.

```
from io import BytesIO
from typing import Optional, Any

import numpy as np
from PIL import Image
from skimage.color import rgb2gray


class CorruptImageError(RuntimeError):
    pass


class ImageSignatureGenerator(object):
    def __init__(
        self,
        n: int = 9,
        crop_percentiles: Optional[tuple[int, int]] = (5,
95),
        P: Optional[int] = None,
        diagonal_neighbors: bool = True,
        identical_tolerance: float = 2 / 255.0,
        n_levels: int = 2,
        fix_ratio: bool = False,
    ):
        self.crop_percentiles: Optional[tuple[int, int]] =
crop_percentiles
        if crop_percentiles is not None:
            if not (0 <= (lower_limit := crop_percentiles[0])
< 100):
                raise ValueError(f"Нижняя граница должна
входить в интервал [0, 100), а получена: {lower_limit}")
            if not (0 < (upper_limit := crop_percentiles[1])
<= 100):
                raise ValueError(f"Верхняя граница должна
входить в интервал (0, 100], а получена: {upper_limit}")
            if not (lower_limit < upper_limit):
                raise ValueError(f"Верхняя граница должна
быть строго больше нижней: {lower_limit} < {upper_limit}")

            self.lower_percentile, self.upper_percentile =
crop_percentiles
        else:
            self.lower_percentile = 0
            self.upper_percentile = 100

        if not (n > 1):
```

```
                raise ValueError(f"Размер сетки на изображении
должен быть больше 1, а получен: {n}")
            self.n = n

            if P is not None and not (P >= 1):
                raise ValueError(f"Размер области для выборки
должен быть не менее 1, а получен: {P}")
            self.P = P

            self.diagonal_neighbors = diagonal_neighbors
            self.sig_length = self.n**2 * (4 +
self.diagonal_neighbors * 4)
            self.fix_ratio = fix_ratio

            if not (0.0 <= identical_tolerance <= 1.0):
                raise ValueError(
                    "Разница отсечения для объявления соседних
точек идентичными должна быть во множестве [0.0, 1.0], "
                    f"а получена: {identical_tolerance}"
                )
            self.identical_tolerance = identical_tolerance

            if not (n_levels > 0):
                raise ValueError(
                    "Количество положительных и негативных групп
для стратификации смежных различий должно быть больше 0, "
                    f"а получено: {n_levels}"
                )
            self.n_levels = n_levels

        def generate_signature(self, image: bytes) -> np.ndarray:
            # Шаг 1: Загрузка изображения как массив уровней
серого
            im_array = self.preprocess_image(image)

            # Шаг 2a: Определение границ для обрезки
            image_limits: Optional[list[tuple[int, int]]]
            if self.crop_percentiles is not None:
                image_limits = self.crop_image(
                    im_array,
                    lower_percentile=self.lower_percentile,
                    upper_percentile=self.upper_percentile,
                    fix_ratio=self.fix_ratio,
                )
            else:
                image_limits = None

            # Шаг 2b: Расчёт центральных точек сетки
            x_coords, y_coords =
self.compute_grid_points(im_array, n=self.n, window=image_limits)
```

```
            # Шаг 3: Расчёт уровней серого для каждой P x P
квадрата с центром в каждой точке сетки
            avg_grey = self.compute_mean_level(im_array,
x_coords, y_coords, P=self.P)

            # Шаг 4a: Расчёт массива различий для каждой точки
сетки с каждой соседней точкой сетки
            diff_mat = self.compute_differentials(avg_grey,
diagonal_neighbors=self.diagonal_neighbors)

            # Шаг 4b: Сужаем различия до 2n+1 значений
            self.normalize_and_threshold(diff_mat,
identical_tolerance=self.identical_tolerance,
n_levels=self.n_levels)

            # Шаг 5: Преобразуем массив значений в подпись
            return np.ravel(diff_mat).astype("int8")

        @staticmethod
        def preprocess_image(image: bytes) -> np.ndarray:
            try:
                img = Image.open(BytesIO(image))
            except IOError:
                raise CorruptImageError

            return rgb2gray(np.asarray(img.convert("RGB"),
dtype=np.uint8))  # noqa

        @staticmethod
        def crop_image(
            image: np.ndarray,
            lower_percentile: int = 5,
            upper_percentile: int = 95,
            fix_ratio: bool = False,
        ) -> list[tuple[int, int]]:
            # row-wise differences
            rw = np.cumsum(np.sum(np.abs(np.diff(image, axis=1)),
axis=1))
            # column-wise differences
            cw = np.cumsum(np.sum(np.abs(np.diff(image, axis=0)),
axis=0))

            # compute percentiles
            upper_column_limit = np.searchsorted(cw,
np.percentile(cw, upper_percentile), side="left")
            lower_column_limit = np.searchsorted(cw,
np.percentile(cw, lower_percentile), side="right")
            upper_row_limit = np.searchsorted(rw,
np.percentile(rw, upper_percentile), side="left")
            lower_row_limit = np.searchsorted(rw,
np.percentile(rw, lower_percentile), side="right")
```

```
            # if image is nearly featureless, use default region
            if lower_row_limit > upper_row_limit:
                lower_row_limit = int(lower_percentile / 100.0 *
image.shape[0])
                upper_row_limit = int(upper_percentile / 100.0 *
image.shape[0])
            if lower_column_limit > upper_column_limit:
                lower_column_limit = int(lower_percentile / 100.0
* image.shape[1])
                upper_column_limit = int(upper_percentile / 100.0
* image.shape[1])

            # if fix_ratio, return both limits as the larger
range
            if fix_ratio:
                if (upper_row_limit - lower_row_limit) >
(upper_column_limit - lower_column_limit):
                    return [(lower_row_limit, upper_row_limit),
(lower_row_limit, upper_row_limit)]
                else:
                    return [(lower_column_limit,
upper_column_limit), (lower_column_limit, upper_column_limit)]

            # otherwise, proceed as normal
            return [(lower_row_limit, upper_row_limit),
(lower_column_limit, upper_column_limit)]

        @staticmethod
        def compute_grid_points(
            image: np.ndarray, n: int = 9, window:
Optional[list[tuple[int, int]]] = None
        ) -> tuple[np.ndarray, np.ndarray]:

            # Если ограничивающие значения не переданы, будем
использовать всё исходное изображение
            if window is None:
                window = [(0, image.shape[0]), (0,
image.shape[1])]

            x_coords = np.linspace(window[0][0], window[0][1], n
+ 2, dtype=int)[1:-1]
            y_coords = np.linspace(window[1][0], window[1][1], n
+ 2, dtype=int)[1:-1]

            return x_coords, y_coords

        @staticmethod
        def compute_mean_level(
            image: np.ndarray, x_coords: np.ndarray, y_coords:
np.ndarray, P: Optional[int] = None
```

```
        ) -> np.ndarray:

            if P is None:
                P = max([2, int(0.5 + min(image.shape) / 20.0)])
# на каждую страницу

            avg_grey = np.zeros((x_coords.shape[0],
y_coords.shape[0]))

            for i, x in enumerate(x_coords):  # дорогая по
ресурсам реализация
                lower_x_lim = int(max([x - P / 2, 0]))
                upper_x_lim = int(min([lower_x_lim + P,
image.shape[0]]))
                for j, y in enumerate(y_coords):
                    lower_y_lim = int(max([y - P / 2, 0]))
                    upper_y_lim = int(min([lower_y_lim + P,
image.shape[1]]))

                    avg_grey[i, j] =
np.mean(image[lower_x_lim:upper_x_lim, lower_y_lim:upper_y_lim])

            return avg_grey

        @staticmethod
        def compute_differentials(grey_level_matrix: np.ndarray,
diagonal_neighbors: bool = True) -> np.ndarray:
            right_neighbors = -np.concatenate(
                (np.diff(grey_level_matrix),
np.zeros(grey_level_matrix.shape[0]).reshape((grey_level_matrix.sh
ape[0], 1))),
                axis=1,
            )
            left_neighbors = -np.concatenate((right_neighbors[:,
-1:], right_neighbors[:, :-1]), axis=1)
            down_neighbors = -np.concatenate(
                (
                    np.diff(grey_level_matrix, axis=0),

np.zeros(grey_level_matrix.shape[1]).reshape((1,
grey_level_matrix.shape[1])),
                )
            )
            up_neighbors = -np.concatenate((down_neighbors[-1:],
down_neighbors[:-1]))

            if diagonal_neighbors:
                # реализация будет работать только для m x m
квадрата сетки
                diagonals = np.arange(-grey_level_matrix.shape[0]
+ 1, grey_level_matrix.shape[0])
```

```
                upper_left_neighbors = sum(

[np.diagflat(np.insert(np.diff(np.diag(grey_level_matrix, i)), 0,
0), i) for i in diagonals]
                )
                lower_right_neighbors = -
np.pad(upper_left_neighbors[1:, 1:], (0, 1), mode="constant")  #
type: ignore
                # разворачиваем анти-диагональные различия
                flipped = np.fliplr(grey_level_matrix)
                upper_right_neighbors = sum(

[np.diagflat(np.insert(np.diff(np.diag(flipped, i)), 0, 0), i) for
i in diagonals]
                )
                lower_left_neighbors = -
np.pad(upper_right_neighbors[1:, 1:], (0, 1), mode="constant")  #
type: ignore

                return np.dstack(
                    (
                        upper_left_neighbors,
                        up_neighbors,
                        np.fliplr(upper_right_neighbors),
                        left_neighbors,
                        right_neighbors,
                        np.fliplr(lower_left_neighbors),
                        down_neighbors,
                        lower_right_neighbors,
                    )
                )

            return np.dstack((up_neighbors, left_neighbors,
right_neighbors, down_neighbors))

        @staticmethod
        def normalize_and_threshold(
            difference_array: np.ndarray, identical_tolerance:
float = 2 / 255.0, n_levels: int = 2
        ):
            # очень близкие значения считаем эквивалентными
            mask = np.abs(difference_array) < identical_tolerance
            difference_array[mask] = 0.0

            if np.all(mask):
                # если изображение не имеет существенных
признаков
                return None
```

```
            positive_cutoffs =
np.percentile(difference_array[difference_array > 0.0],
np.linspace(0, 100, n_levels + 1))
            negative_cutoffs =
np.percentile(difference_array[difference_array < 0.0],
np.linspace(100, 0, n_levels + 1))

            for level, interval in enumerate([positive_cutoffs[i
: i + 2] for i in range(positive_cutoffs.shape[0] - 1)]):
                difference_array[(difference_array >=
interval[0]) & (difference_array <= interval[1])] = level + 1

            for level, interval in enumerate([negative_cutoffs[i
: i + 2] for i in range(negative_cutoffs.shape[0] - 1)]):
                difference_array[(difference_array <=
interval[0]) & (difference_array >= interval[1])] = -(level + 1)

            return None

        @staticmethod
        def normalized_distance(_a: np.ndarray, _b: np.ndarray) -
> np.floating:
            b = _b.astype(int)
            a = _a.astype(int)
            norm_diff = np.linalg.norm(b - a)
            norm1 = np.linalg.norm(b)
            norm2 = np.linalg.norm(a)
            return norm_diff / (norm1 + norm2)


    def make_image_sign_record(
        image_signature_generator: ImageSignatureGenerator, k:
int, N: int, image_data: bytes, metadata=None
    ) -> dict[str, Any]:
        record = dict()
        signature =
image_signature_generator.generate_signature(image_data)

        record["signature"] = signature.tolist()

        if metadata:
            record["metadata"] = metadata

        words = get_words(signature, k, N)
        max_contrast(words)

        words = words_to_int(words)

        for i in range(N):
            record["".join(["sw_", str(i)])] = words[i].tolist()
```

```
        return record


    def get_words(array, k, N):
        # generate starting positions of each word
        word_positions = np.linspace(0, array.shape[0], N,
endpoint=False).astype("int")

        # check that inputs make sense
        if k > array.shape[0]:
            raise ValueError("Word length cannot be longer than
array length")
        if word_positions.shape[0] > array.shape[0]:
            raise ValueError("Number of words cannot be more than
array length")

        # create empty words array
        words = np.zeros((N, k)).astype("int8")

        for i, pos in enumerate(word_positions):
            if pos + k <= array.shape[0]:
                words[i] = array[pos : pos + k]
            else:
                temp = array[pos:].copy()
                temp.resize(k, refcheck=False)
                words[i] = temp

        return words


    def words_to_int(word_array):
        width = word_array.shape[1]

        # Three states (-1, 0, 1)
        coding_vector = 3 ** np.arange(width)
        # The 'plus one' here makes all digits positive, so that
the
        # integer represntation is strictly non-negative and
unique
        return np.dot(word_array + 1, coding_vector)


    def max_contrast(array):
        array[array > 0] = 1
        array[array < 0] = -1

        return None


    def normalized_distances(_target_array, _vec, nan_value=1.0):
        target_array = _target_array.astype(int)
```

```
vec = _vec.astype(int)
topvec = np.linalg.norm(vec - target_array, axis=1)
norm1 = np.linalg.norm(vec, axis=0)
norm2 = np.linalg.norm(target_array, axis=1)
finvec = topvec / (norm1 + norm2)
finvec[np.isnan(finvec)] = nan_value

return finvec
```