

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального
назначения»

Специализация: «Эксплуатация вычислительных машин,
комплексов, систем и сетей специального
назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ИНТЕЛЕКТУАЛЬНАЯ СИСТЕМА
АВТОМАТИЗИРОВАННОЙ ОБРАБОТКИ И
СЕМАНТИЧЕСКОГО АНАЛИЗА АВТОМОБИЛЬНЫХ
ДОКУМЕНТОВ

Студент

Сальников Егор Владимирович
(подпись, дата) (ФИО полностью)

Руководитель

Митрохин М.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент

Петерский С.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 28.05.2025 №16)

Заведующий кафедрой

Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой

отлично (протокол заседания ГЭК от 19.06.2025 № 4)

Секретарь ГЭК

Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»
Заведующий кафедрой
Митрохин М.А.
«05» 05 2025 г.

**ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ**

1. Студент Сальников Егор Владимирович гр. 20ВОЭ1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Интеллектуальная система автоматизированной обработки и семантического анализа автомобильных документов

Тема утверждена приказом ПГУ №601/о от "06" мая 2025

3. Руководитель работы д.т.н., зав. кафедры «ВТ» Митрохин М.А.

4. Задание на работу

1) *Назначение: интеллектуальная система для автоматизированной обработки и семантического анализа автомобильных документов с применением машинного обучения и компьютерного зрения*

2) *Среда выполнения: операционная система Windows 10, Android, ios, Ubuntu 22*

Языковые средства разработки: C#, Python, Kotlin, Go

Инструментальные средства: Rider, PyCharm, IntelliJ IDEA, Draw.io

Технологии разработки: микросервисная архитектура, REST API,

машинное обучение (PyTorch, OpenCV, EasyOCR), контейнеризация,

принципы SOLID, PostgreSQL

3) *Основные функции приложения:*

а) загрузка и предварительная обработка изображений автомобильных

б) автоматическое распознавание автомобильных документов;

в) семантический анализ и валидация извлеченных данных;

г) формирование отчетов о результатах обработки.

5. Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и требований к интеллектуальной системе;*

2) *Проектирование интеллектуальной системы;*

3) *Реализация интеллектуальной системы;*

4) *Тестирование интеллектуальной системы;*

5) *Оформление пояснительной записки.*

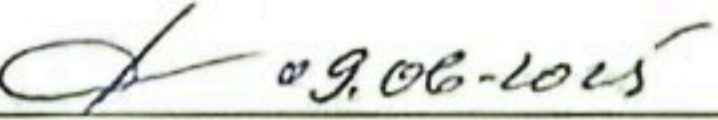
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к интеллектуальной системе	10%	06.05.2025 – 08.05.2025	
2	Проектирование интеллектуальной системы	20%	9.05.2025 – 14.05.2025	
3	Реализация интеллектуальной системы	30%	15.05.2025 – 22.05.2025	
4	Тестирование интеллектуальной системы	20%	23.05.2025 – 27.05.2025	
5	Оформление пояснительной записки	20%	28.05.2025 – 30.05.2025	

Дата выдачи задания «06» мая 2025г.

Руководитель дипломной работы  06.05.2025
(подпись, дата)

Задание к исполнению принял студент  06.05.2025
(подпись, дата)

Работу к защите допустить
Декан ФВТ  09.06.2025 Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 134 листа, 19 рисунков, 32 таблицы, 15 использованных источников и 3 приложения.

АВТОМОБИЛЬНЫЕ ДОКУМЕНТЫ, OCR, ИНТЕЛЛЕКТУАЛЬНАЯ ОБРАБОТКА ДАННЫХ

Объект исследования – интеллектуальная система автоматизации обработки автомобильных документов (водительское удостоверение, ПТС, СТС) и их интерпретации в структурированном виде.

Цель дипломной работы – спроектировать и разработать систему распознавания автомобильных документов с мультиплатформенным клиентским приложением, позволяющую автоматически извлекать данные с изображений и файлов, а также предоставлять удобный интерфейс для взаимодействия с пользователем.

Технологии разработки – серверная часть системы реализована на языках Python, Go и платформе .net 8 для обеспечения гибкости и высокой производительности обработки изображений. Предварительная обработка и распознавание документов выполняются с использованием модели машинного обучения YOLOv8 и EasyOCR. Хранение данных организовано с помощью СУБД PostgreSQL. Клиентская часть разработана с применением Kotlin Multiplatform, что обеспечивает кроссплатформенную поддержку. Развертывание системы осуществляется с использованием Docker.

Результаты работы – в ходе выполнения дипломной работы была спроектирована и реализована система распознавания автомобильных документов с мультиплатформенным клиентским приложением, позволяющая автоматически извлекать данные с изображений.

					ПГУ 1.090501.14.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Интеллектуальная система автоматизированной обработки и семантического анализа автомобильных документов	Лит.	Лист	Листов
Разраб.		Сальников Е.В.		01.06.25				
Пров.		Митрохин М.А.		02.06.25			4	134
Реценз.		Петерский		27.06.25		ФВТ, каф. ВТ гр. 20ВОЭ1		
Н. Контр.		Бычков А.С.		27.06.25				
Утв.		Митрохин		27.06.25				

Содержание

Введение	8
1 Анализ предметной области и постановка задачи на разработку программных средств автоматизации распознавания автомобильных документов	9
1.1 Анализ предметной области распознавания автомобильных документов ..	9
1.2 Анализ существующих технологий распознавания документов	10
1.3 Анализ существующих решений и аналогов	12
1.4 Постановка задачи на разработку программных средств	15
1.5 Анализ функциональных требований к программным средствам	17
1.6 Анализ нефункциональных требований к программным средствам	22
2 Проектирование интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов	23
2.1 Архитектура системы	23
2.2 Проектирование схем данных	24
2.2.1 Концептуальная модель базы данных результатов работы системы ..	24
2.3 Проектирование компонентов системы	28
2.4 Проектирование пользовательского интерфейса	36
3 Реализация интеллектуальной системы	41
3.1 Реализация серверной части	41
3.1.1 Реализация API	41
3.1.2 Реализация сервиса распознавания документов	47
3.1.3 Реализация файлового сервиса	54
3.1.4 Реализация с базы данных	59
3.2 Реализация клиентского приложения	62
3.3 Развертывание программных средств	65

3.3.1	Настройка Docker и Docker Compose	68
3.3.2	Настройка Kafka и микросервисной инфраструктуры	72
4	Тестирование и оптимизация	75
4.1	Методология тестирования.....	75
4.2	Тестирование серверной части.....	76
4.3	Тестирование клиентского приложения	80
5	Оценка затрат на разработку и внедрение интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов.....	84
5.1	Организация производства разработки интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов.....	84
5.2	Расчет себестоимости программного обеспечения для интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов	87
5.3	Оценка эффективности интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов.....	91
5.4	Анализ рисков при разработке и внедрении интеллектуальной системы автоматизированной обработки и семантического анализа документов	92
6	Общие принципы обеспечения безопасности на рабочем месте оператора интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов.....	95
6.1	Актуальность вопросов безопасности при работе с системами обработки документов.....	95
6.2	Обеспечение информационной безопасности при работе с персональными данными	96

6.3 Эргономика рабочего места оператора системы	97
6.4 Режим труда и отдыха при работе с системой обработки документов	98
6.5 Техника безопасности при работе с высокопроизводительным оборудованием.....	99
Заключение	101
Список использованных источников	102
Приложение А – Uml-диаграммы	104
Приложение А.1 – Диаграмма последовательностей.....	105
Приложение А.2 – Диаграмма развертывания	106
Приложение А.3 – Диаграмма компонентов	107
Приложение А.4 – Диаграмма вариантов использования	108
Приложение А.5 – Диаграмма деятельности	109
Приложение Б – Листинг	110
Приложение В – Презентация	129

Введение

В современном цифровом обществе автоматизация обработки документов играет ключевую роль в обеспечении эффективного и безопасного взаимодействия между пользователями и организациями. Программные средства распознавания документов становятся все более важными для оптимизации процессов, связанных с верификацией личности, оформлением договоров, страхованием транспортных средств и другими операциями, требующими работы с бумажными и электронными формами. В последние годы технологии машинного обучения и компьютерного зрения получили широкое распространение в области обработки изображений и распознавания текста в документах. В контексте автомобильных документов такие методы позволяют автоматически извлекать данные из водительских удостоверений, паспортов транспортных средств (ПТС) и свидетельств о регистрации (СТС), что значительно ускоряет выполнение административных задач и снижает риск человеческой ошибки [1].

Целью дипломного проекта является разработка программного обеспечения для распознавания автомобильных документов с поддержкой мультиплатформенного клиентского приложения. Разработка включает в себя анализ предметной области, проектирование архитектуры системы, реализацию серверной и клиентской частей, а также тестирование функциональности. Разрабатываемая система должна обеспечивать автоматическое извлечение данных с изображений документов, их структурирование и предоставление удобного интерфейса для взаимодействия с пользователем.

1 Анализ предметной области и постановка задачи на разработку программных средств автоматизации распознавания автомобильных документов

1.1 Анализ предметной области распознавания автомобильных документов

С развитием цифровых технологий автоматизация обработки документов становится неотъемлемой частью многих сфер деятельности, включая государственные услуги, банковскую и страховую деятельность. Программные средства распознавания документов играют ключевую роль в ускорении верификации личности, оформления договоров и других операций, требующих работы с бумажными и электронными формами.

Конечное применение таких технологий сводится к созданию удобного пользовательского интерфейса, позволяющего загружать документы, производить их автоматический анализ и предоставлять результаты в структурированном виде. Развитие облачных решений и микросервисных архитектур позволило перенести вычислительно сложные задачи на серверную сторону, оставив клиентское приложение легковесным и простым в использовании. Однако коммерческие решения в этой области зачастую имеют высокую стоимость и ограничения по функционалу, что делает разработку собственного решения актуальной задачей [2].

В рамках автоматизации обработки автомобильных документов основной задачей является извлечение ключевых полей из изображений автомобильных документов. Традиционный ручной ввод данных занимает много времени и сопряжен с риском ошибок. Поэтому наиболее эффективным подходом к решению данной задачи является использование технологий оптического распознавания символов (OCR) [2], дополненного машинным обучением для повышения точности и устойчивости к различным условиям съемки.

1.2 Анализ существующих технологий распознавания документов

С развитием технологий компьютерного зрения и машинного обучения значительно возрос интерес к автоматизации обработки бумажных и электронных документов. Одной из ключевых задач в этой области является разработка эффективных методов распознавания текстовых данных на изображениях документов. Современные подходы к решению данной задачи базируются на использовании алгоритмов оптического распознавания символов (OCR), а также моделей глубокого обучения для сегментации и распознавания структурных элементов документа [3].

На текущий момент существует несколько основных подходов к реализации систем распознавания документов. Классические OCR-библиотеки, применяются для распознавания текста на изображении без предварительной настройки под конкретные типы документов. Методы машинного обучения, включающие обучение моделей на выборках изображений документов с последующей классификацией полей и их содержимым, также широко используются в современных системах. Глубокие нейронные сети, такие как CNN, YOLO, CRNN, обеспечивают высокую точность распознавания за счет анализа графических особенностей документа. Интеграция компьютерного зрения и ML представляет собой комбинированные подходы, в которых используются библиотеки OpenCV для предварительной обработки изображений и нейросетевые модели для детекции и распознавания текста. Каждый из этих подходов имеет свои преимущества и ограничения, что определяет выбор технологии в зависимости от конкретных требований к системе распознавания документов.

Для объективной оценки эффективности подходов к обработке автомобильных документов проведены вычислительные эксперименты на выборке из 500 изображений ПТС, СТС и водительских удостоверений с разным уровнем качества (отсканированные документы, фото с мобильных устройств).

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

В качестве показательных параметров для сравнения были выбраны Точность распознавания слов (%)– доля правильно извлеченных слов, Точность детекции полей (%) – доля корректно выделенных структурных элементов документа, Время обработки (МС) – время выполнения операции в миллисекундах, Поддержка русского языка – наличие моделей для специфических форматов документов РФ, Интеграция с ML-моделями – возможность дообучения под новые типы документов. Результаты тестирования приведены в таблице 1, таблице 2, таблице 3.

Таблица 1 — Результаты сравнения OCR-инструментов

OCR-инструмент	Точность распознавания слов (%)	Время обработки (МС)	Поддержка русского языка	Открытый исходный код
Tesseract OCR	87	480	Частичная	Да
EasyOCR	92	350	Да	Да
PaddleOCR	79	650	Нет	Да
ABBYY FineReader	95	1200	Да	Нет
Google Vision API	91	900	Да	Нет

Таблица 2 — Сравнение методов сегментации структурных элементов

Инструмент сегментации	Точность детекции полей (%)	Время обработки (МС)	Поддержка русского языка	Открытый исходный код
YOLOv5	82	28	Да	Да
YOLOv8	89	22	Да	Да
PaddleOCR (встроенный)	75	51	Нет	Да
YOLO + OpenCV	88	32	Да	Да

Таблица 3 — Эффективность OCR + Сегментация

OCR-инструмент	Инструмент сегментации	Средняя точность распознавания и сегментации (%)	Суммарное время я обработки (МС)
Tesseract OCR	YOLOv5	84,5	508
EasyOCR	YOLOv8	90,5	372
PaddleOCR	встроенный	77	701

Из данных, приведенных в таблицах видно, что комбинация OCR инструмента EasyOCR [5] и модели сегментации изображения YOLOv8 [6] является наиболее точным, обеспечивая среднюю точность 90,5%, и самым быстрым решением, при суммарном времени обработки 372 мс.

1.3 Анализ существующих решений и аналогов

Проанализировав рынок программных продуктов аналогичного назначения, можно выделить платформы, предоставляющие инструменты для автоматизации обработки и семантического анализа автомобильных документов. Основные аналоги являются: Smart IDReader, Content AI PassportReader SDK, WASM OCR SDK, Модуль ПО «Распознавание документов». Общие сведения о рассматриваемых аналогах приведены в таблице 4.

Таблица 4 – Общие сведения о программных продуктах аналогичного назначения

Название приложения	Организация	Общие сведения
Smart IDReader	ООО "Смарт Энджинс Сервис"	Решение для распознавания документов на основе ИИ. Поддерживает обработку паспортов, водительских удостоверений, ПТС и других документов. Используется в банках, госучреждениях и мобильных приложениях. Интегрируется через SDK для Android/iOS/Web. Высокая точность (95%+), поддержка русского языка и шрифтов РФ.

Название приложения	Организация	Общие сведения
Content AI PassportReader SDK	АО "РусГард".	Специализированный SDK для распознавания паспортов и идентификационных документов. Разработан для использования в государственных системах и банках. Обеспечивает соответствие требованиям ФСБ и ФСТЭК. Поддерживает верификацию подлинности документа и биометрическую проверку.
WASM OCR SDK	ООО Смарт Энджинс	Браузерное решение для распознавания документов без установки плагинов. Реализовано на WebAssembly (WASM) для работы в веб-приложениях. Подходит для онлайн-сервисов, где требуется обработка документов через камеру или загрузку файла. Поддерживает русскоязычные форматы.
Модуль ПО «Распознавание документов»	ООО «ПЭРКО»	Часть системы контроля доступа и учета рабочего времени. Предназначен для автоматического извлечения данных из документов сотрудников (паспорта, водительские удостоверения). Интегрирован с базами данных ПЭРКО для верификации личности. Используется в корпоративных системах безопасности.

Представленное программное обеспечение аналогичного назначения имеет различный функционал и способы развертывания на клиентской стороне. Анализ данных, указанных на сайтах разработчиков, представляет собой выявление преимуществ и недостатков, а также сравнение данных программных продуктов.

Smart IDReader библиотека (SDK) подпрограмм для распознавания документов на изображениях.

К основным преимуществам данного решения можно отнести следующие:

- поддерживает распознавание ПТС, СТС, водительских удостоверений с высокой точностью (95%);
- предоставляет готовые библиотеки для Android, iOS и веб-приложений, упрощая внедрение в сторонние платформы.

К недостаткам приложения можно отнести следующие:

- SDK закрытый, не позволяет дообучать модель под новые типы документов без участия разработчика;
- отсутствие глубокого семантического анализа данных;

- отсутствие облачной версии для быстрого развертывания.

Content AI PassportReader SDK предоставляет большой функционал в рамках специфики распознавания паспортов и идентификационных документов.

К основным преимуществам данного решения можно отнести следующие:

- наличие проверки защитных элементов документов;
- возможность встраивания в корпоративные системы безопасности.

К недостаткам приложения можно отнести следующие:

- ограниченная поддержка автомобильных документов;
- зависимость от специфических аппаратных средств;
- отсутствие облачной версии для быстрого развертывания.

WASM OCR SDK предоставляет эффективные функциональные возможности для распознавания документов непосредственно в браузере.

К основным преимуществам данного решения можно отнести следующие:

- работа в браузере без установки дополнительного ПО;
- обработка данных на стороне клиента без передачи на сервер.

К недостаткам приложения можно отнести следующие:

- ограниченная функциональность по сравнению с нативными решениями;
- зависимость от возможностей браузера;
- отсутствие облачной версии для быстрого развертывания.

Модуль ПО «Распознавание документов» от ООО «ПЭРКО» предоставляет программное решение для автоматизации ввода данных из документов в информационные системы.

К основным преимуществам данного решения можно отнести следующие:

- интеграция с системами электронного документооборота;
- поддержка распознавания структурированных документов.

К недостаткам приложения можно отнести следующие:

- ограниченная поддержка автомобильных документов;
- ограниченные возможности для мобильного использования;
- отсутствие облачной версии для быстрого развертывания.

Для проведения сравнительного анализа приложений аналогов был рассмотрен набор критериев, на основе которых производится сравнение. Результаты сравнения по заданным критериям приведены в таблице 5.

Таблица 5 – Результаты сравнения разрабатываемой интеллектуальной системы с аналогами

Критерий	Разрабатываемая система	Smart IDReader	WASM OCR SDK	Content AI PassportReader SDK	Модуль ПО «Распознавание документов»
Высокая точность распознавания документов	Да	Да	Частично	Частично	Частично
Семантический анализ данных	Да	Нет	Нет	Нет	Нет
Возможность автономной работы	Да	Да	Да	Да	Нет
Интеграция с другими системами	Да	Да	Да	Да	Да
Кроссплатформенность	Да	Частично	Нет	Да	Нет

На основе результатов сравнения можно сделать вывод, что разрабатываемая Интеллектуальная система автоматизированной обработки и семантического анализа автомобильных документов способна в достаточной мере выполнять необходимые задачи. Ключевым преимуществом разрабатываемого решения является наличие семантического анализа данных, высокая точность распознавания автомобильных документов, наличия кросс платформенного пользовательского интерфейса [7].

1.4 Постановка задачи на разработку программных средств

Основной задачей на разработку программных средств автоматизации распознавания автомобильных документов является создание системы, позволяющей выполнять классификацию и извлечение данных из различных типов автомобильных документов с высокой точностью. В качестве

дополнительных условий пользования приложением можно выделить удобство и эффективность графического интерфейса для взаимодействия с пользователем на разных платформах, а также приближенность используемых алгоритмов к специфике обрабатываемых документов.

Для формирования автоматического процесса распознавания документов необходимо выделить программные компоненты, на основе которых будет построена система. К основным компонентам можно отнести модуль классификации типа документа, модуль сегментации областей интереса и модуль оптического распознавания символов. Используемые для распознавания алгоритмы обладают рядом характеристик, которые необходимо учитывать для обеспечения высокого качества результатов. К данным характеристикам можно отнести точность распознавания, скорость обработки и отказоустойчивость при обработке документов с различными дефектами. На основании выделенных характеристик можно обозначить основные наборы данных для использования – изображение документа, тип документа, извлеченные поля и история операций. Таким образом, для использования и хранения полученных в ходе распознавания сведений целесообразно задействовать СУБД. Это позволит оперировать данными о предыдущих распознаваниях, а также обеспечивать доступ к истории обработки документов.

Также для более эффективного взаимодействия пользователя с графическими элементами управления процессом распознавания необходимо распределить элементы графического интерфейса по функциональным группам. На основе выделенных признаков принадлежности инструментов распознавания документов можно сформировать следующие разделы программы:

- главная страница;
- сканирование документа;
- просмотр результатов распознавания;
- история распознаваний.

Таким образом, на основе сформированных требований будут произведены проектирование и разработка интеллектуальной системы

автоматизированной обработки и семантического анализа автомобильных документов

1.5 Анализ функциональных требований к программным средствам

Для визуального представления функциональных требований к разрабатываемой системе применяется диаграмма вариантов использования [8]. Данный инструмент моделирования наглядно демонстрирует взаимодействие между пользователями и системой, отображая возможные сценарии использования программного продукта и участников этих процессов. С помощью диаграммы вариантов использования удастся более четко определить потребности конечных пользователей и сформулировать необходимые функциональные требования к создаваемой системе. На рисунке 1 представлена разработанная диаграмма вариантов использования, отражающая ключевые аспекты взаимодействия с системой автоматизированной обработки и семантического анализа автомобильных документов. Диаграмма вариантов использования интеллектуальной системы представлена на рисунке 1.

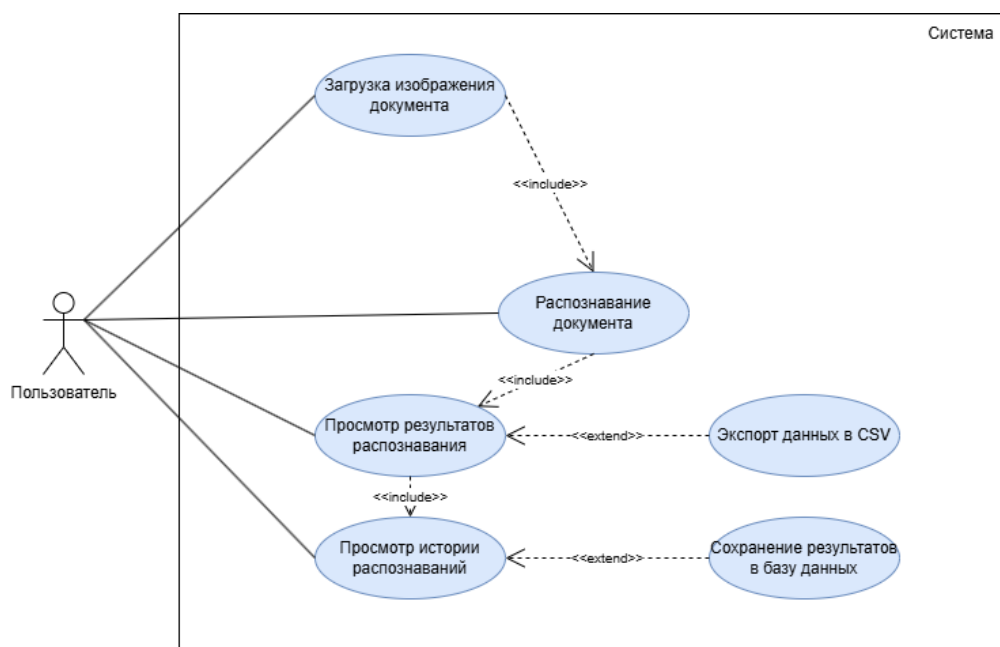


Рисунок 1 – Диаграмма вариантов использования интеллектуальной системы

Основным управляющим элементом в системе выступает пользователь, который через графический интерфейс получает доступ ко всем функциональным возможностям приложения. Ключевые сценарии взаимодействия с системой реализованы через различные разделы интерфейса, каждый из которых предоставляет определенный набор специализированных функций для работы с автомобильными документами.

Спецификация варианта использования «Загрузка изображения документа» приведена в таблице 6.

Таблица 6 – Спецификация варианта использования «Загрузка изображения документа»

Прецедент: Загрузка изображения документа
ID: UC-01
Краткое описание: Пользователь загружает изображение документа в систему для последующего распознавания
Главные актеры: Пользователь
Второстепенные актеры: Нет
Прецедент: Загрузка изображения документа
Предусловия: Пользователь имеет доступ к системе
Основной поток: <ul style="list-style-type: none"> – пользователь открывает экран загрузки документа; – пользователь выбирает опцию загрузки файла; – пользователь выбирает изображение из файловой системы; – система проверяет формат и размер изображения; – система отображает загруженное изображение.
Постусловия: Изображение загружено в систему и готово к распознаванию
Альтернативные потоки: <ul style="list-style-type: none"> – если формат не поддерживается, система выводит сообщение об ошибке; – если размер превышает допустимый, система предлагает сжать изображение.

Спецификация варианта использования «Распознавание документа» приведена в таблице 7.

Таблица 7 – Спецификация варианта использования «Распознавание документа»

Прецедент: Распознавание документа
ID: UC-02
Краткое описание: Система анализирует изображение документа и извлекает из него структурированные данные
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: В системе загружено изображение документа
Основной поток: <ul style="list-style-type: none"> – пользователь инициирует процесс распознавания; – система определяет тип документа; – система выполняет сегментацию и нахождение значимых областей; – система применяет OCR для распознавания текста; – система структурирует данные в соответствии с типом документа.
Постусловия: Данные документа распознаны и структурированы
Альтернативные потоки: Нет

Спецификация варианта использования «Просмотр результатов распознавания» приведена в таблице 8.

Таблица 8 – Спецификация варианта использования «Просмотр результатов распознавания»

Прецедент: Просмотр результатов распознавания
ID: UC-03
Краткое описание: Пользователь просматривает результаты распознавания документа
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Документ успешно распознан
Основной поток: <ul style="list-style-type: none"> – система отображает распознанные данные; – пользователь просматривает результаты.

Прецедент: Просмотр результатов распознавания
Постусловия: Результаты распознавания просмотрены пользователем
Альтернативные потоки: Нет

Спецификация варианта использования «Просмотр истории распознаваний» приведена в таблице 9.

Таблица 9 – Спецификация варианта использования «Просмотр истории распознаваний»

Прецедент: Просмотр истории распознаваний
ID: UC-04
Краткое описание: Пользователь получает доступ к ранее распознанным документам
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: В системе есть ранее распознанные документы
Основной поток: – пользователь открывает экран истории; – система отображает список ранее распознанных документов; – пользователь выбирает документ из списка; система отображает данные выбранного документа.
Постусловия: Пользователь получил доступ к истории распознаваний
Альтернативные потоки: Нет

Спецификация варианта использования «Экспорт данных в CSV» приведена в таблице 10.

Таблица 10 – Спецификация варианта использования «Экспорт данных в CSV»

Прецедент: Экспорт данных в CSV
ID: UC-05
Краткое описание: Пользователь выгружает распознанные данные в формате CSV
Краткое описание: Пользователь получает доступ к распознанным документам

Прецедент: Экспорт данных в CSV
Главные актеры: Пользователь
Предусловия: Документ успешно распознан и данные просмотрены
Основной поток: – пользователь выбирает опцию экспорта; – система формирует CSV-файл; – система предлагает сохранить.
Постусловия: Данные экспортированы в CSV-файл
Альтернативные потоки: Нет

Спецификация варианта использования «Сохранение результатов в базу данных» приведена в таблице 11.

Таблица 11 – Спецификация варианта использования «Сохранение результатов в базу данных»

Прецедент: Сохранение результатов в базу данных
ID: UC-06
Краткое описание: Система автоматически сохраняет результаты распознавания в базу данных
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Документ успешно распознан
Основной поток: – система структурирует данные для сохранения; – система сохраняет данные в базу.
Постусловия: Результаты распознавания сохранены в базе данных для последующего использования
Альтернативные потоки: Нет

Таким образом, проанализированные функциональные возможности системы определяют основные способы взаимодействия с пользователем. Это

позволяет выделить бизнес-процессы приложения и сформировать минимальный набор реализуемых функций для корректной работоспособности.

1.6 Анализ нефункциональных требований к программным средствам

Серверная часть системы для обработки нейросетевых моделей требует процессор не ниже Intel Xeon E5 или AMD EPYC (16+ ядер), 64 ГБ оперативной памяти, GPU NVIDIA RTX 3080 или Tesla T4 с 16+ ГБ видеопамяти. Дисковая подсистема — 1 ТБ NVMe SSD для системы и приложений, 4+ ТБ SSD/HDD для хранилища MinIO. Сетевое подключение — 1 Гбит/с, желательно 10 Гбит/с при высокой нагрузке. Для клиентских устройств на Windows/Linux процессор i5/Ryzen 5, 8 ГБ RAM, 250 МБ свободного места, дисплей 1080p., Android процессор 8 ядер (Snapdragon 700 серии или аналог), 4 ГБ RAM, 100 МБ свободного места, Android 9.0+., iOS: iPhone 8 и новее, iPad 6-го поколения и новее, iOS/iPadOS 14+, 80 МБ свободного места., Веб-клиент Chrome 90+, Firefox 88+, Safari 14+, Edge 90+. Для взаимодействия по сети клиентская часть требует стабильного интернет-соединения от 5 Мбит/с. Для комфортной работы с загрузкой и получением результатов обработки документов рекомендуется не менее 10 Мбит/с. При развертывании приватного облака — изолированная сеть с пропускной способностью не менее 100 Мбит/с между микросервисами. Контейнеризация Docker при росте нагрузки требует добавления новых узлов кластера с минимальными требованиями в 4 ядра CPU и 8 ГБ RAM на узел.

					ПГУ 1.090501.14.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

2 Проектирование интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

2.1 Архитектура системы

Согласно выделенным требованиям к системе, была разработана диаграмма коммуникаций, отражающая основной процесс взаимодействия компонентов системы. Разработанная диаграмма приведена на рисунке 2.

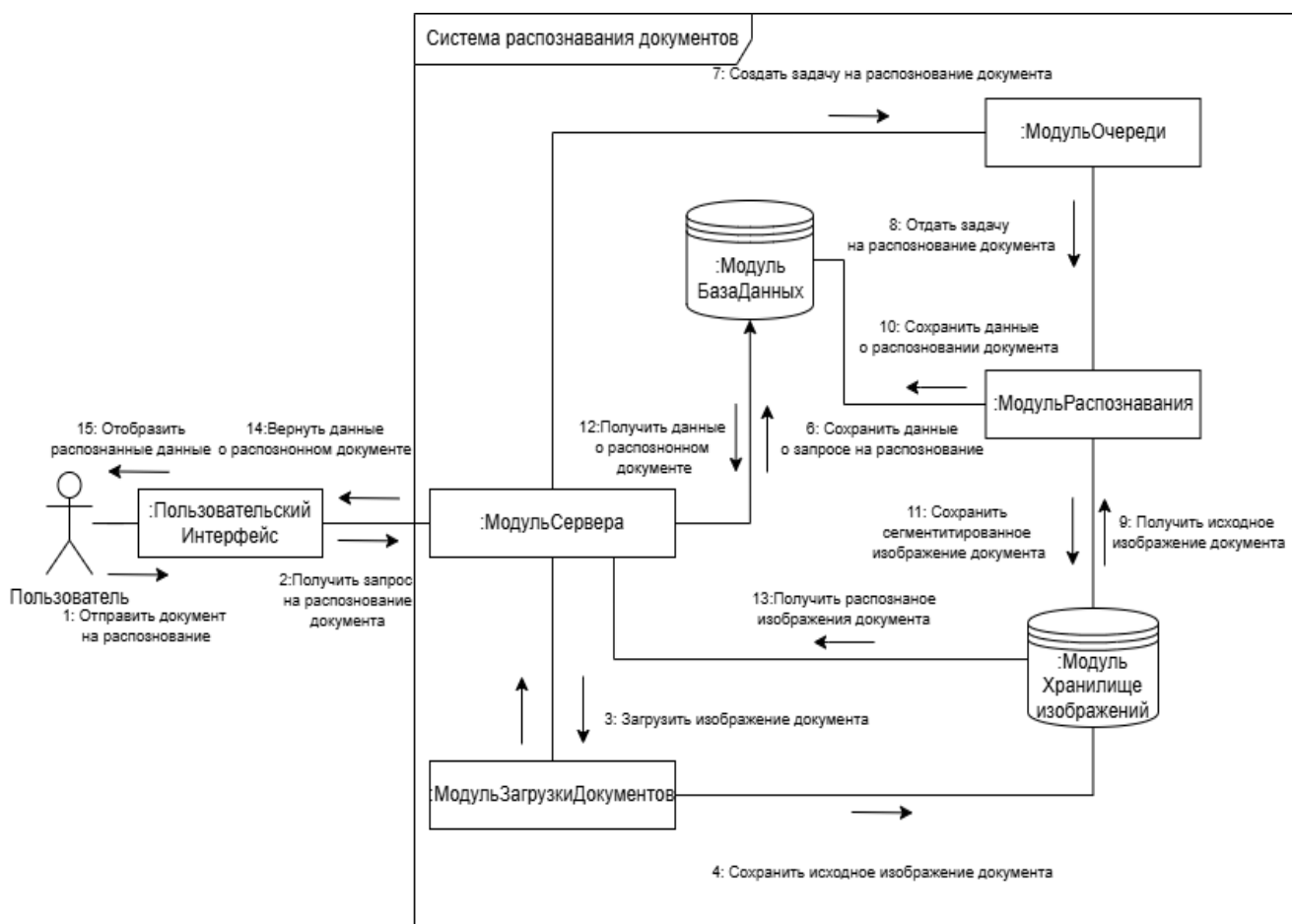


Рисунок 2 – Диаграмма коммуникаций системы

Данная диаграмма отражает основные архитектурные особенности системы распознавания документов, а также демонстрирует взаимодействие компонентов на разных уровнях. Основной частью системы является Пользовательский Интерфейс, с которым взаимодействует пользователь. Исходными данными для работы системы являются изображения документов,

которые загружаются пользователем. Дальнейшая работа системы предполагает обработку запроса Модулем Сервера, который управляет всем процессом распознавания.

Загруженные изображения сохраняются в Хранилище с помощью Модуля Загрузки Документов, а информация о запросе фиксируется в Базе Данных. Модуль Сервера формирует задачу на распознавание и передает ее в Модуль Очереди, который регулирует последовательность обработки запросов. Модуль Распознавания получает задачу из очереди, извлекает исходное изображение из хранилища и выполняет процедуру распознавания документа.

В процессе распознавания система сегментирует изображение, извлекает текстовую информацию и структурирует данные в соответствии с типом документа. Результаты распознавания сохраняются в Базе Данных и Хранилище, включая как извлеченные данные, так и сегментированное изображение документа. Модуль Сервера получает данные о распознанном документе и предоставляет их через Пользовательский Интерфейс.

Конечным результатом работы системы является отображение распознанных данных пользователю в структурированном виде. Система позволяет не только просматривать результаты, но и выполнять дополнительные операции, такие как экспорт данных или формирование отчетов. Архитектура системы обеспечивает масштабируемость, надежность и высокую производительность при обработке большого количества документов.

2.2 Проектирование схем данных

2.2.1 Концептуальная модель базы данных результатов работы системы

Концептуальная модель данных представляет собой наглядное отражение организации данных на высоком уровне. Данное представление демонстрирует описание сущностей, их атрибуты и отношения без указания технических деталей реализации.

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		24

Согласно выделенным требованиям к системе распознавания автомобильных документов по хранению и организации данных была разработана диаграмма типа «сущность-связь». Данная диаграмма представлена на рисунке 3.

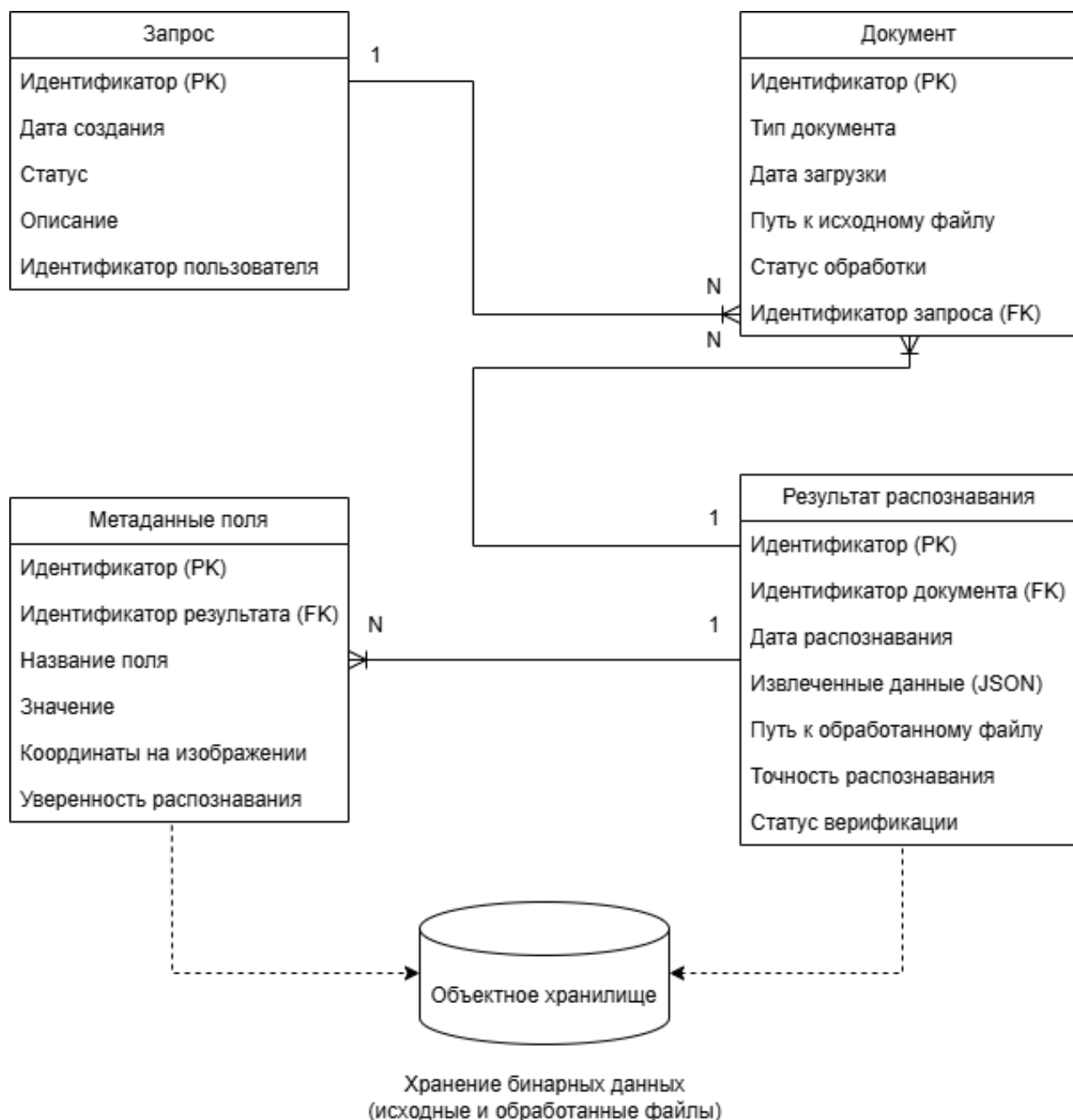


Рисунок 3 – Графическое представление концептуальной модели

Разработанная диаграмма наглядно отражает хранимые элементы приложения и их связи. В таблице 12 представлены сущности, сформированные согласно входным и выходным данным бизнес-процессов.

Таблица 12 – Описание сущностей

Наименование сущности	Идентификатор сущности	Назначение сущности
Запрос	Request	Содержит информацию о дате создания, текущем статусе выполнения запроса, описании запроса и идентификаторе пользователя, инициировавшего процесс.
Результат распознавания	Response	Содержит информацию о типе документа, дате загрузки, пути к исходному файлу в объектном хранилище и текущем статусе обработки.
Метаданные поля	Metadata	Содержит название поля извлеченное значение, координаты расположения поля на изображении и уровень уверенности системы в правильности распознавания.
Объектное хранилище	Object Storage	Хранит бинарные данные изображений. Организовано по принципу плоской структуры с использованием уникальных идентификаторов для доступа к файлам.

В таблице 13 представлено описание связей между сущностями.

Таблица 13 – Описание связей

Наименование связи	Описание	Мощность
Запрос – Документ	Связь между запросом на обработку и документами, которые к нему относятся. Запрос инициирует обработку документов, а документы содержат ссылку на породивший их запрос через внешний ключ.	1:N
Документ – Результат распознавания	Связь между загруженным документом и результатами его обработки системой. Документ может обрабатываться несколько раз с разными параметрами, а каждый результат распознавания однозначно привязан к документу.	1:N
Результат распознавания – Метаданные поля	Связь между результатом распознавания и детальными данными по каждому извлеченному полю. Результат распознавания документа содержит множество распознанных полей, а каждое поле относится только к одному результату.	1:N
Документ – Объектное хранилище	Логическая связь, обеспечивающая доступ к исходным файлам документов через атрибут "Путь к исходному файлу", содержащий идентификатор в хранилище.	1:1
Результат распознавания – Объектное хранилище	Логическая связь, обеспечивающая доступ к обработанным файлам с визуализацией результатов через атрибут "Путь к обработанному файлу".	1:1

Концептуальная модель данных построена в графической среде draw.io в нотации IDEF1X [9].

2.3 Проектирование компонентов системы

На основе многокомпонентной архитектуры интеллектуальной системы была разработаны диаграммы классов каждого модуля, представлены на рисунке 4, рисунке 5, рисунке 6, рисунке 7. Эта диаграмма позволяет наглядно отобразить структурную организацию и взаимосвязи между ключевыми программными компонентами, что обеспечивает четкое понимание технических решений, применяемых при реализации системы. Каждый модуль представлен соответствующими классами, методами и атрибутами, что формирует целостное представление о функциональных возможностях разрабатываемого приложения и способствует оптимальной организации процесса разработки.

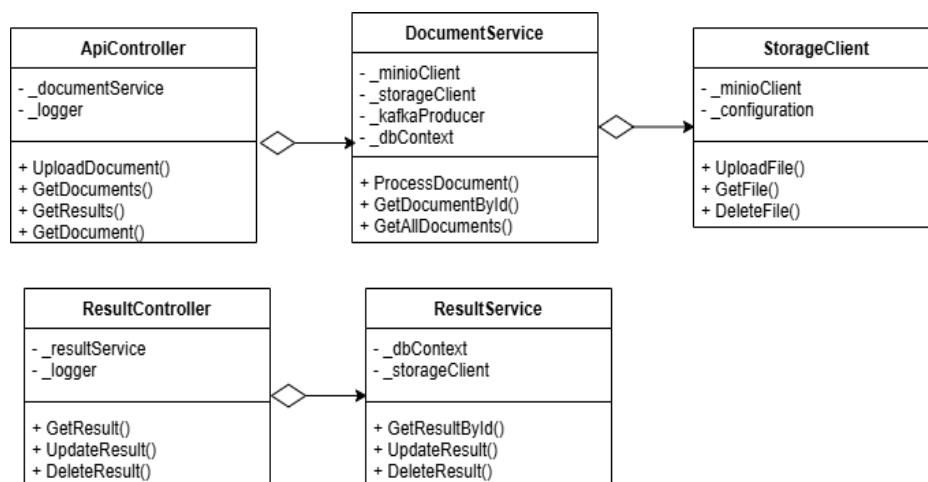


Рисунок 4 – Диаграмма классов модуля сервера

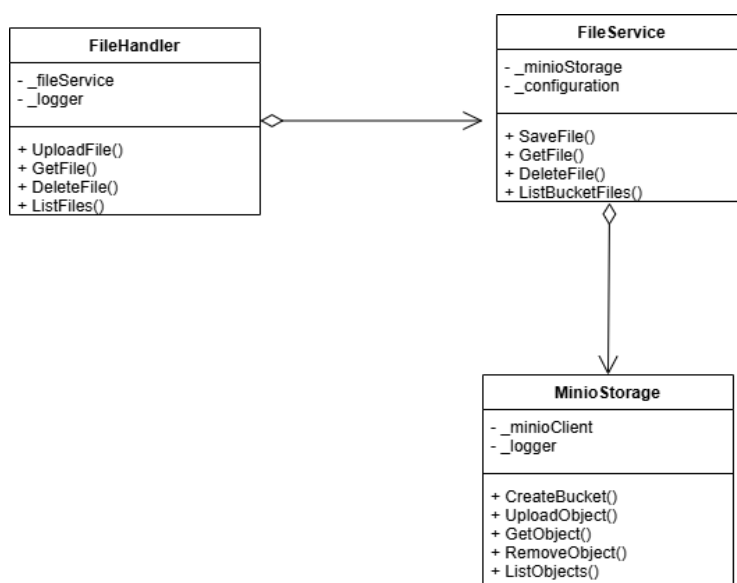


Рисунок 5 – Диаграмма классов модуля загрузки документов

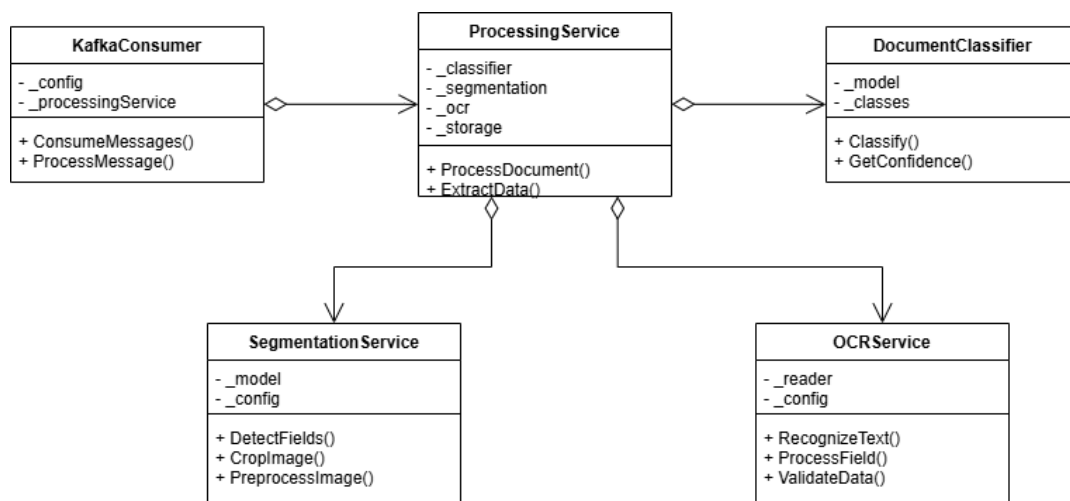


Рисунок 6 – Диаграмма классов модуля распознавания

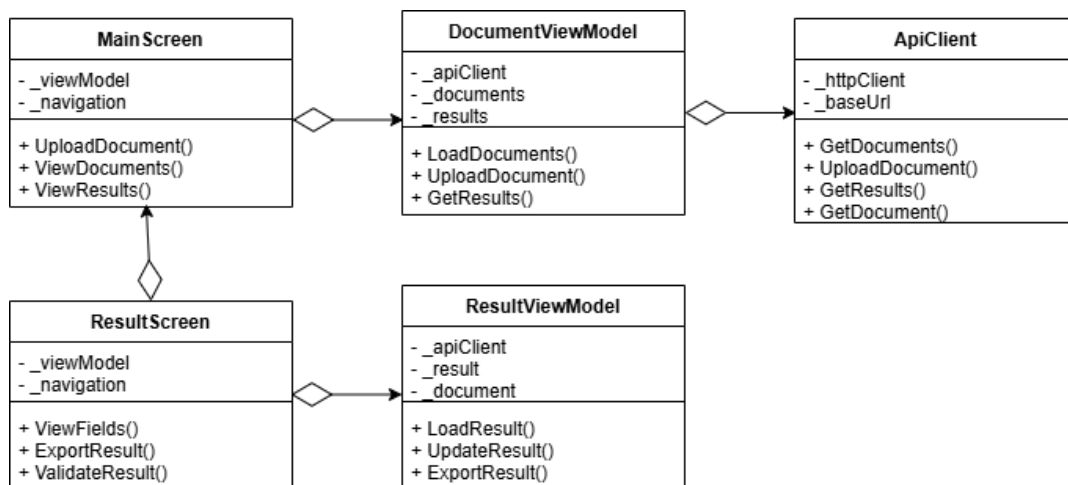


Рисунок 7 – Диаграмма классов модуля пользовательского интерфейса

В таблице 14 представлена спецификация классов модуля сервера.

Таблица 14 – Спецификация классов модуля сервера

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
ApiController	Класс контроллера для обработки и HTTP-запросов, связанные	_documentService	Сервис для работы с документами	UploadDocument()	Загрузка документа на сервер

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
	х с документ ами				
			_logger	Логгер для записи действий	GetDocuments()
					GetDocument()
					GetResults()
ResultController	Класс контроллера для работы с результатами распознавания	_resultService	Сервис для работы с результатами	GetResult()	Получение результата по идентификатору
			_logger	Логгер для записи действий	UpdateResult()
					DeleteResult()
DocumentService	Сервис для работы с документами	_minioClient	Клиент для работы с MinIO	ProcessDocument()	Обработка документа
			_storageClient	Клиент для работы с хранилищем	GetDocumentById()
			_kafkaProducer	Клиент для работы с Kafka	GetAllDocuments()
			_dbContext	Контекст базы данных	
ResultService	Сервис для работы с результатами	_dbContext	Контекст базы данных	GetResultById()	Получение результата по идентификатору

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
	распознавания				
			_storageClient	Клиент для работы с хранилищем	UpdateResult()
					DeleteResult()
StorageClient	Клиент для работы с объектным хранилищем	_minioClient	Клиент для работы с MinIO	UploadFile()	Загрузка файла
			_configuration	Конфигурация клиента	GetFile()
					DeleteFile()
KafkaProducer	Клиент для отправки сообщений в Kafka	_configuration	Конфигурация Kafka	SendMessage()	Отправка сообщения
			_logger	Логгер для записи действий	

В таблице 15 представлена спецификация классов модуля загрузки документов.

Таблица 15 – Спецификация классов модуля загрузки документов

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
FileHandler	Обработчик HTTP-запросов для работы с файлами	_fileService	Сервис для работы с файлами	UploadFile()	Загрузка файла

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
		_logger	Логгер для записи действий	GetFile()	Получение файла
				DeleteFile()	Удаление файла
				ListFiles()	Получение списка файлов
FileService	Сервис для работы с файлами	_minioStorage	Хранилище MinIO	SaveFile()	Сохранение файла
		_configuration	Конфигурация сервиса	GetFile()	Получение файла
				DeleteFile()	Удаление файла
				ListBucketFiles()	Получение списка файлов в корзине
MinioStorage	Класс для работы с хранилищем MinIO	_minioClient	Клиент MinIO	CreateBucket()	Создание корзины
		_logger	Логгер для записи действий	UploadObject()	Загрузка объекта
				GetObject()	Получение объекта
				RemoveObject()	Удаление объекта
				ListObjects()	Получение списка объектов

В таблице 16 представлена спецификация классов модуля распознавания.

Таблица 16 – Спецификация классов модуля распознавания

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
KafkaConsumer	Класс для получения сообщений из Kafka	_config	Конфигурация потребителя	ConsumeMessages()	Получение сообщений
		_processingService	Сервис обработки документов	ProcessMessage()	Обработка полученного сообщения
ProcessingService	Основной сервис обработки документов	_classifier	Классификатор документов	ProcessDocument()	Обработка документа
		_segmentation	Сервис сегментации	ExtractData()	Извлечение данных
		_ocr	Сервис распознавания текста	SendResults()	Отправка результатов
			_storage	Клиент хранилища	
DocumentClassifier	Классификатор типов документов	_model	Модель классификации	Classify()	Классификация документа
		_classes	Классы документов	GetConfidence()	Получение уверенности и классификации
SegmentationService	Сервис сегментации	_model	Модель сегментации	DetectFields()	Обнаружение полей

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
	документ ов				
		_config	Конфигурация сервиса	CropImage()	Вырезание области изображения
				PreprocessImage()	Предварительная обработка изображения
OCRService	Сервис оптического распознавания текста	_reader	OCR- движок	RecognizeText()	Распознавание текста
		_config	Конфигурация сервиса	ProcessField()	Обработка поля документа
				ValidateData()	Валидация распознанных данных

В таблице 17 представлена спецификация классов модуля пользовательского интерфейса.

Таблица 17 – Спецификация классов модуля пользовательского интерфейса

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
MainScreen	Главный экран приложения	_viewModel	Модель представления	UploadDocument()	Загрузка документа
		_navigation	Навигация	ViewDocuments()	Просмотр

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
		_documents	Список документов	UploadDocument()	Загрузка документа
		_results	Список результатов	GetResults()	Получение результатов
ApiClient	Клиент для работы с серверным API	_httpClient	HTTP-клиент	GetDocuments()	Получение списка документов
		_baseUrl	Базовый URL API	UploadDocument()	Загрузка документа
				GetResults()	Получение результатов
				GetDocument()	Получение информации о документе
ResultScreen	Экран просмотра результатов	_viewModel	Модель представления	ViewFields()	Просмотр полей документа
		_navigation	Навигация в приложении	ExportResult()	Экспорт результата
				ValidateResult()	Валидация результата
ResultViewModel	представление для результатов	_apiClient	Клиент API	LoadResult()	Загрузка результата

Класс	Описание класса	Поля	Описание полей	Методы	Описание методов
		_result	Результат распознавания	UpdateResult()	Обновление результата
		_document	Документ	ExportResult()	Экспорт результата

Спроектированные диаграммы классов системы позволяет организовать структуру системы, а также уточнить технические детали реализации.

2.4 Проектирование пользовательского интерфейса

Согласно выделенным функциональным требованиям к системе автоматизированной обработки и семантического анализа автомобильных документов был спроектирован графический интерфейс пользователя. Данный интерфейс предоставляет интуитивно понятный доступ к основным возможностям системы, включая загрузку, обработку и анализ документов. Проектируемое приложение реализовано с использованием Kotlin Multiplatform, что обеспечивает кроссплатформенность и единообразие пользовательского опыта на различных устройствах. Структура пользовательского интерфейса состоит из нескольких ключевых экранов, каждый из которых отвечает за определенный аспект работы с автомобильными документами. Навигационная структура мобильного приложения построена на основе четырех основных экранов, каждый из которых выполняет определенную функцию в процессе обработки документов. Переходы между экранами реализованы с использованием библиотеки PreCompose Navigation, что обеспечивает плавную анимацию и корректное управление состоянием приложения. Диаграмма переходов между экранами мобильного приложения представлена на рисунке 8. Интерфейс разработан с учетом принципов современного UX/UI дизайна,

обеспечивая максимальную эффективность взаимодействия пользователя с системой.

Рисунок 8 — Диаграмма переходов между экранами мобильного приложения

Главный экран приложения представлен на рисунке 9.

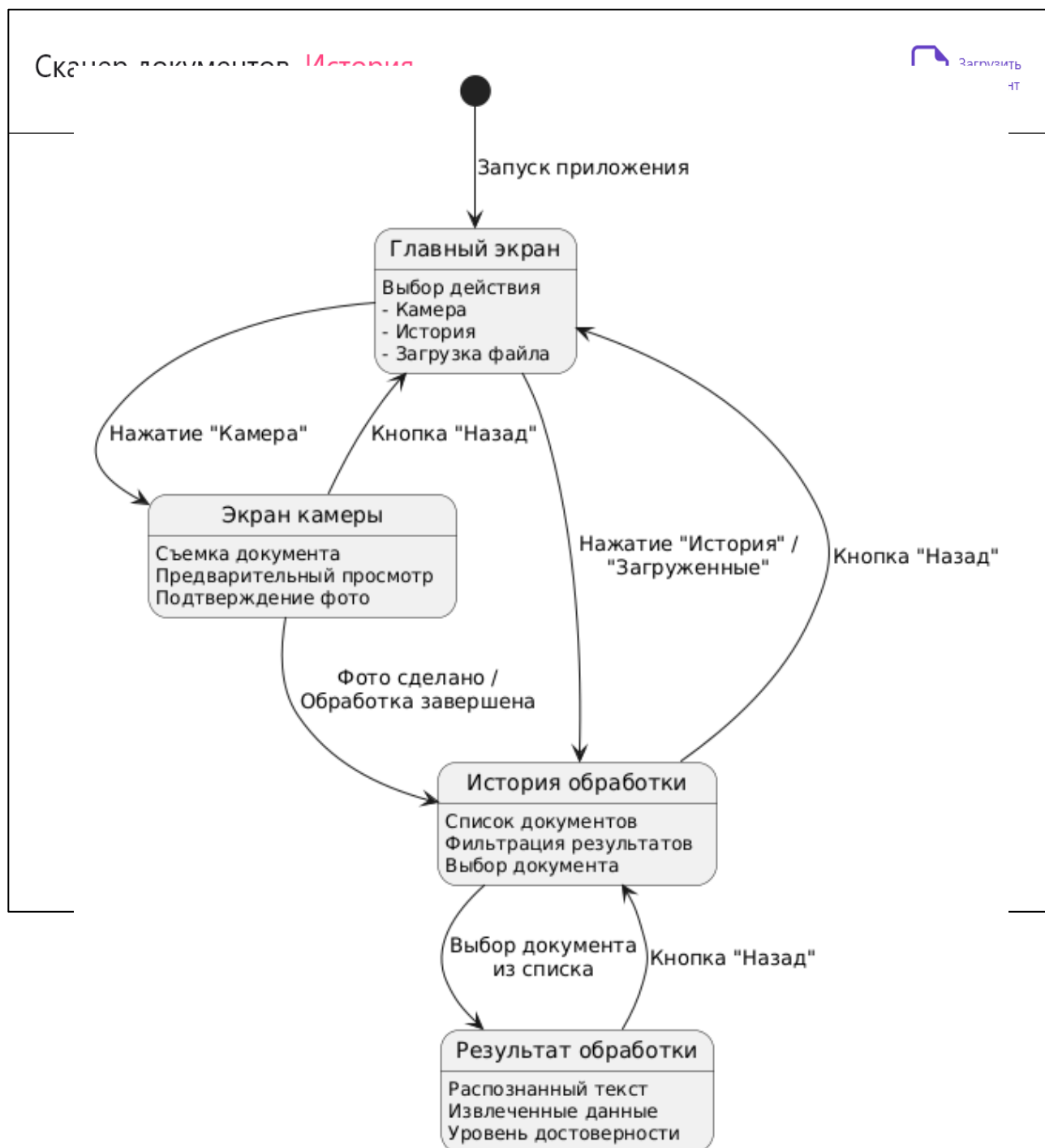


Рисунок 9 – Графический интерфейс главного экрана

Главный экран приложения представлен следующими графическими элементами:

- панель навигации для доступа к различным разделам системы;
- кнопки для перехода на экран истории;
- кнопки загрузки документа.

Главный экран обеспечивает удобный доступ ко всем ключевым функциям системы и позволяет быстро оценить текущее состояние обработки документов. Пользователь может загрузить новые документы простым перетаскиванием файлов или через стандартный диалог выбора файлов. Экран просмотра результатов распознавания представлен на рисунке 10.

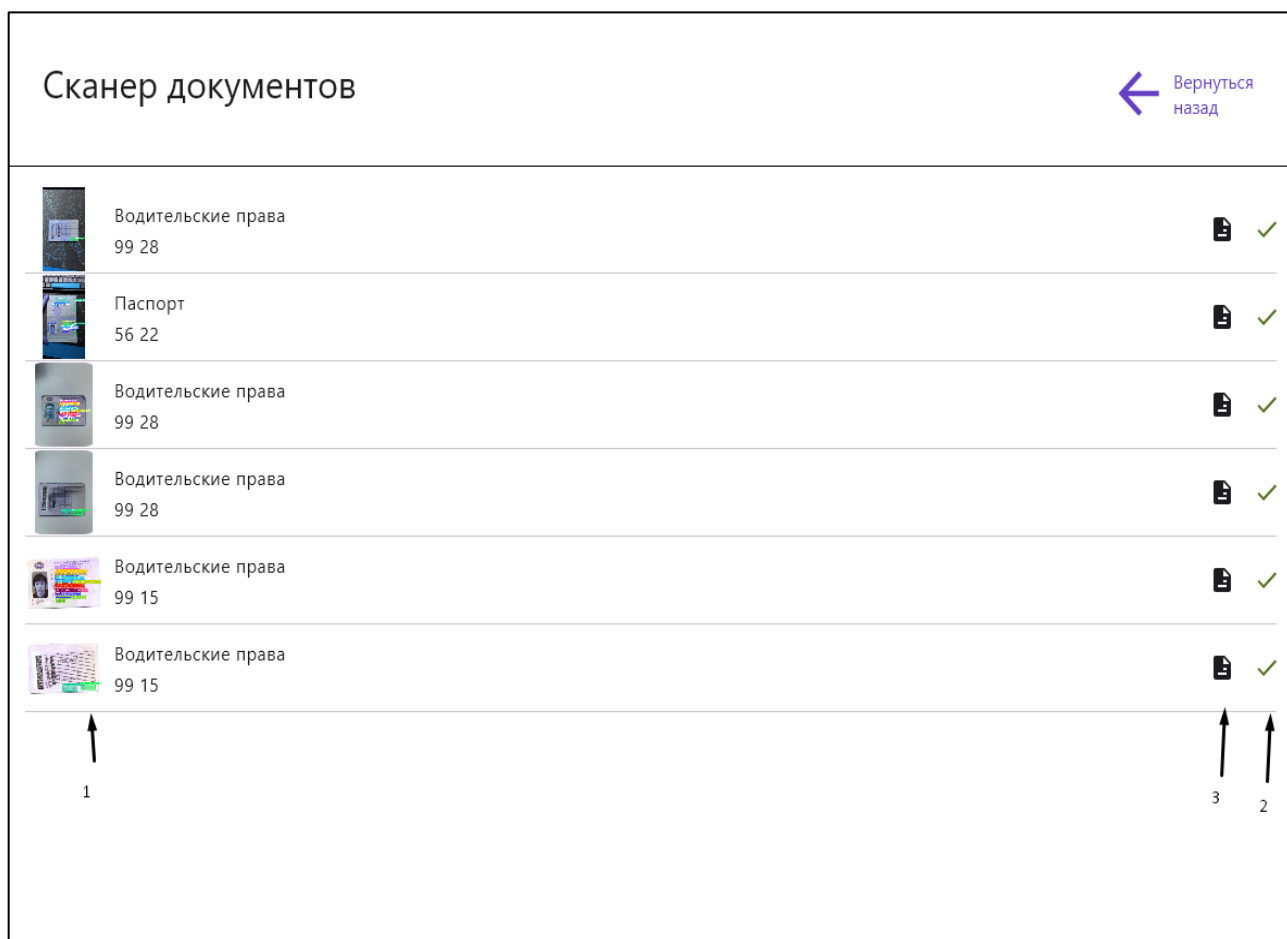


Рисунок 10 – Графический интерфейс экран просмотра результатов распознавания

Данный экран содержит следующие графические элементы:

- изображение обрабатываемого документа;

- индикаторы уверенности распознавания для каждого поля;
- кнопки для экспорта результатов в различные форматы.

Экран результатов распознавания предоставляет пользователю возможность проанализировать результаты автоматического распознавания документов, экспортировать полученные данные. Экран настроек распознавания представлен на рисунке 11.

Данный экран содержит следующие графические элементы:

- область визуализации документа, отображающая сканированное изображение обрабатываемого автомобильного документа;
- информационная панель с распознанными данными документа в формате "параметр-значение".

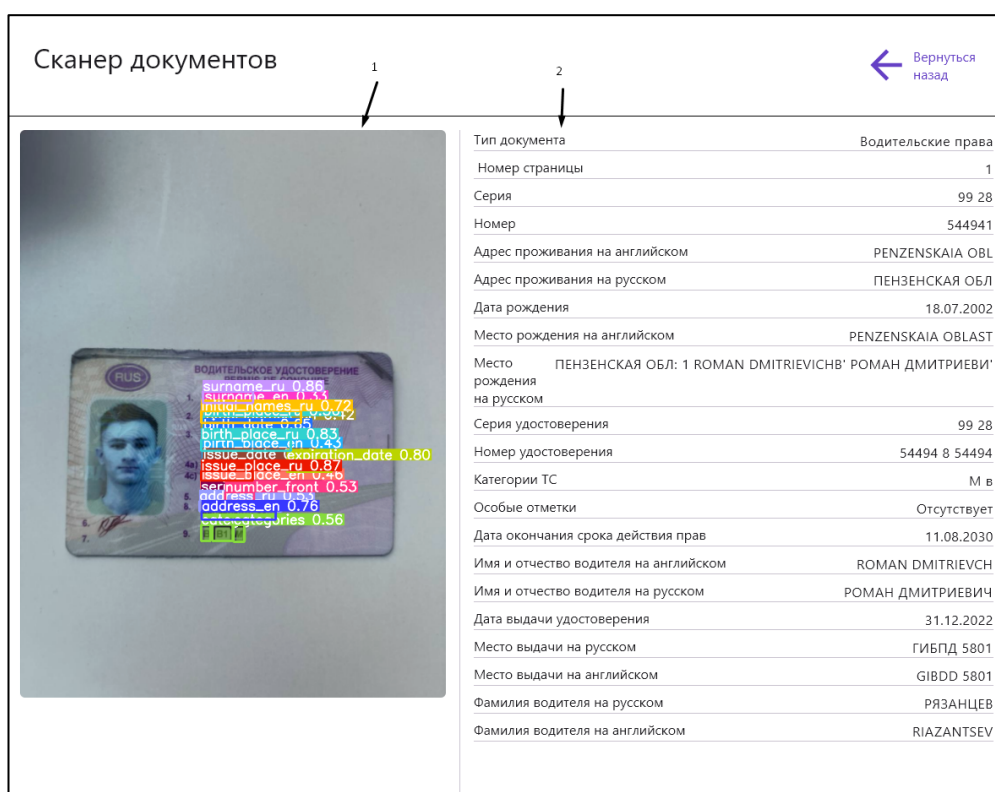


Рисунок 11 – Графический интерфейс экран просмотра результатов распознавания

Разработанный пользовательский интерфейс обеспечивает эффективное взаимодействие с ключевыми функциями пользовательского интерфейса.

Организация элементов управления и информационных блоков способствует интуитивному восприятию данных и повышает удобство эксплуатации системы.

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		40

3 Реализация интеллектуальной системы

3.1 Реализация серверной части

API системы реализовано на базе ASP.NET Core и представляет собой ключевой компонент серверной части, обеспечивающий взаимодействие между клиентскими приложениями и функциональными сервисами. API предоставляет набор эндпоинтов для работы с автомобильными документами: загрузка изображений документов, получение истории распознаваний, доступ к обработанным файлам и получение результатов в формате CSV. Основные контроллеры включают Document Recognition Controller для работы с распознаванием, File Controller для доступа к файлам и Detect Controller для определения типа документа. API обеспечивает асинхронную обработку запросов с использованием Kafka для очередей сообщений, взаимодействует с файловым сервисом через REST и сохраняет данные в PostgreSQL.

3.1.1 Реализация API

Реализация API системы автоматизированной обработки и семантического анализа автомобильных документов выполнена с использованием фреймворка ASP.NET Core. API представляет собой центральный узел взаимодействия между клиентскими приложениями и различными сервисами системы. Для наглядного представления структуры API и его взаимодействия с другими компонентами системы разработана и представлена на рисунке 12 диаграмма компонентов.

Для демонстрации физического размещения компонентов системы и их развертывания разработана и представлена на рисунке 13 диаграмма развертывания серверной части системы.

API системы выступает интерфейсом взаимодействия давая возможность для обработки автомобильных документов. С его помощью пользователи легко загружают изображения документов, мгновенно получают доступ к полной истории распознаваний и выгружают результаты в удобном CSV-формате.

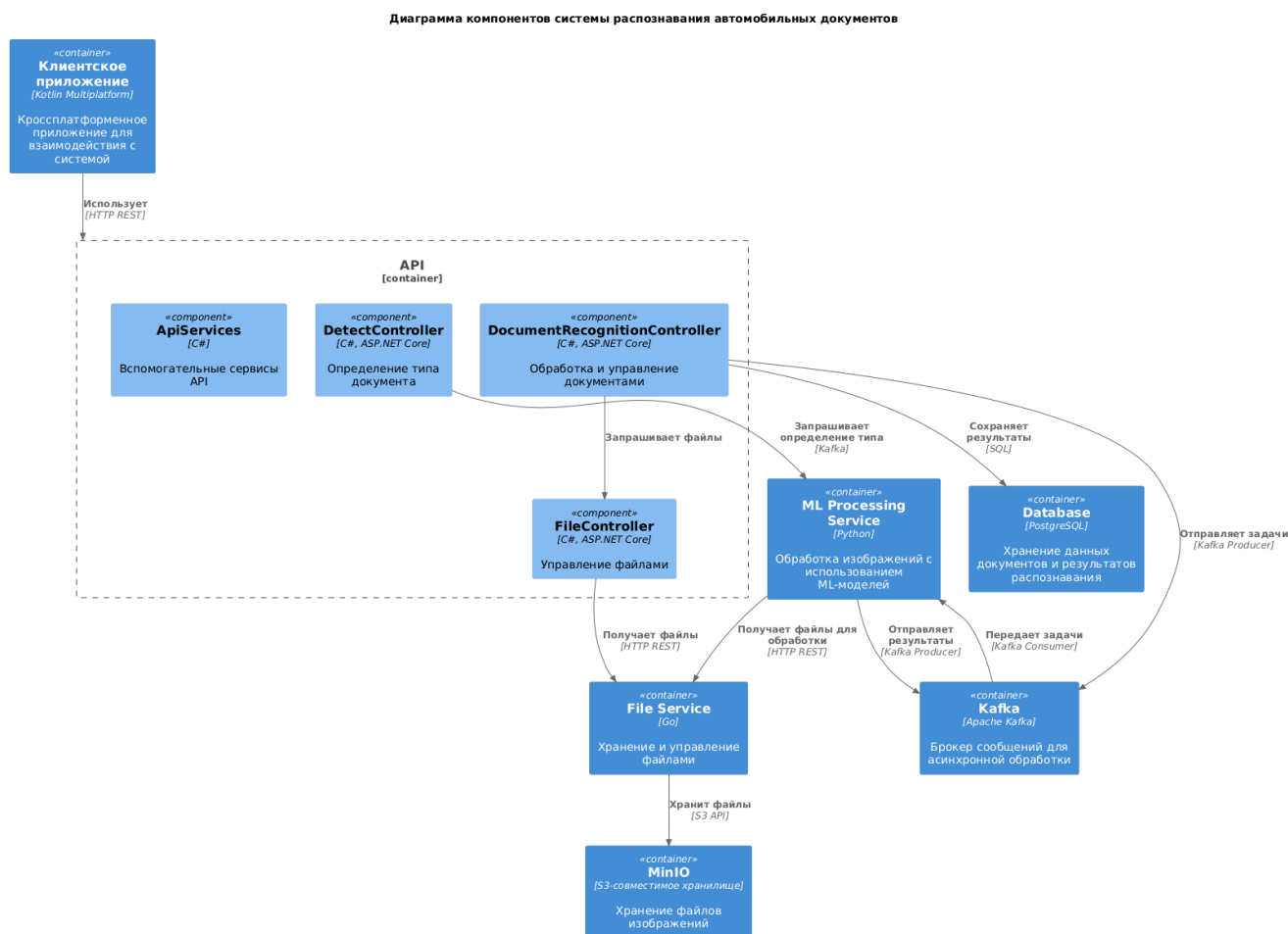


Рисунок 12 – Диаграмма компонентов системы

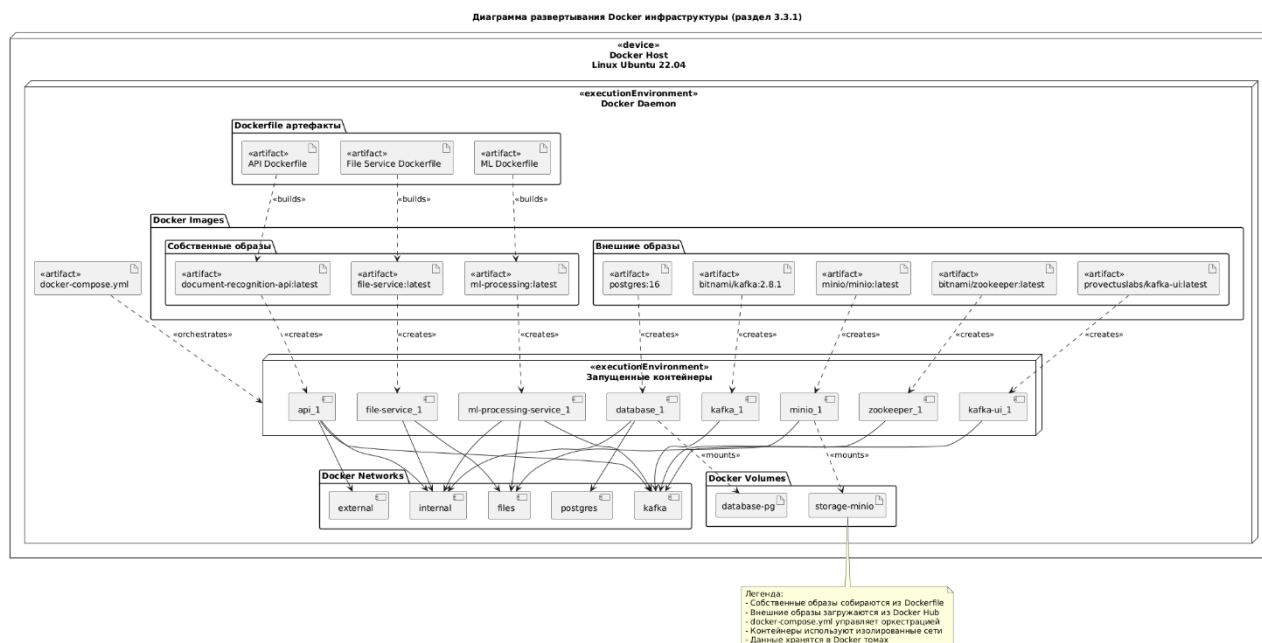


Рисунок 13 – Диаграмма развертывания серверной части системы

API позволяет просматривать обработанные изображения с детальной разметкой и точно определять тип автомобильного документа без лишних усилий.

Взаимодействие между клиентским приложением и сервером построено на RESTful подходе с передачей данных в формате JSON. Асинхронная обработка запросов осуществляется через Kafka – это решение не только снижает время отклика, но и распределяет нагрузку между всеми компонентами системы. Хранение запросов и ответов размещаются в PostgreSQL, а сами изображения документов – в S3-совместимом хранилище MinIO, что гарантирует быстрый доступ и отказоустойчивость системы.

Центральным элементом API является файл конфигурации Program.cs, который определяет все зависимости и сервисы приложения:

```
// Конфигурация зависимостей и сервисов в Program.cs
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Регистрация сервисов для работы с базой данных
builder.Services.AddSingleton<IConnection, Connection>();
builder.Services.AddSingleton<IConfigurationSettings,
ConfigurationSettings>();

// Регистрация Kafka сервисов для асинхронного взаимодействия
builder.Services.AddSingleton<IKafkaProduceService,
KafkaProduceService>();
builder.Services.AddSingleton<KafkaEventHandler>();
builder.Services.AddHostedService<KafkaConsumerService>();

// Регистрация файлового сервиса
builder.Services.AddScoped<FileService>();
builder.Services.AddScoped<IFileService>(provider =>
provider.GetRequiredService<FileService>());

var app = builder.Build();
```

Основную логику обработки запросов распознавания документов реализует контроллер DetectController, который координирует взаимодействие между файловым сервисом, ML-сервисом через Kafka и формирует ответы клиентам:

```

/// <summary>
/// Контроллер для обработки запросов распознавания автомобильных
документов
/// </summary>
[ApiController]
[Route("[controller]")]
public class DetectController : ControllerBase
{
    private readonly ILogger<DetectController> logger;
    private readonly IFileService fileService;
    private readonly IKafkaProducesService kafkaProducesService;
    private readonly KafkaEventHandler kafkaEventHandler;

    public DetectController(ILogger<DetectController> logger,
                           IFileService fileService,
                           IKafkaProducesService
kafkaProducesService,
                           KafkaEventHandler kafkaEventHandler)
    {
        this.logger = logger;
        this.fileService = fileService;
        this.kafkaProducesService = kafkaProducesService;
        this.kafkaEventHandler = kafkaEventHandler;
    }

    [HttpPost]
    public async Task<IActionResult> Detect([FromBody]
ImageDataForHackTask imageDataForHackTask)
    {
        if (imageDataForHackTask == null ||
string.IsNullOrEmpty(imageDataForHackTask.Image))
        {
            return BadRequest("Image data is missing.");
        }

        // Декодирование Base64 изображения
        var imageBytes =
Convert.FromBase64String(imageDataForHackTask.Image);

        // Сохранение файла в файловый сервис
        var id = await fileService.UploadFileAsync(imageBytes);
        var guid = Guid.NewGuid().ToString();

        // Формирование сообщения для Kafka
        var kafkaMessage = new RequestMessage { Guid = guid, FileId
= id };
        await
kafkaProducesService.WriteTraceLogAsync(kafkaMessage);

        // Ожидание результата обработки через подписку на события
        var tcs = new TaskCompletionSource<string>();
        EventHandler<ResultMessage> handler = null;
        handler = (sender, message) =>

```

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		44

```

    {
        if (message?.Guid == guid)
        {
            var response = new ResultForHackTask
            {
                Type = message.Type,
                Series = message.Series,
                Number = message.Number,
                Confidence = message.Confidence,
                PageNumber = message.PageNumber
            };
            tcs.SetResult(response.ToJson());
            kafkaEventHandler.MessageReceived -= handler;
        }
    };

    kafkaEventHandler.MessageReceived += handler;
    var receivedMessage = await tcs.Task;

    return Ok(receivedMessage);
}
}

```

Данный контроллер реализует асинхронную модель обработки: после загрузки изображения в файловый сервис формируется уникальный идентификатор задачи, отправляется сообщение в Kafka для ML-обработки, а затем API ожидает результат через механизм подписки на события. Это обеспечивает неблокирующую обработку запросов и возможность горизонтального масштабирования системы.

Полный спектр функциональности API системы представлен в таблице 18, которая содержит детальное описание всех доступных эндпоинтов, их назначения и ожидаемых результатов

Таблица 18 — Сводная таблица API эндпоинтов системы

Функционал запроса	HTTP метод и URL	Ожидаемый результат возврата	Код состояния
Распознавание документа (хакатон)	POST / detect	JSON с типом документа: {"Type":"driver_license", "Series":"9910", "Number":"876422"}	200

Функционал запроса	HTTP метод и URL	Ожидаемый результат возврата	Код состояния
Загрузка документа для распознавания	POST /DocumentRecognition	Успешная обработка запроса, инициация фонового распознавания	200
Получение истории распознаваний пользователя	GET /DocumentRecognition/{userId}	JSON массив с результатами: [{"Status":1,"UserId":"id","Type":"personal_passport","Series":"1234","Number":"567890","Data":{"поля_документа}}]	200
Тестовый метод получения документов	GET /DocumentRecognition/test/{userId}	JSON с тестовыми данными для всех типов документов (паспорт, ВУ, ПТС, СТС)	200
Скачивание csv	GET /DocumentRecognition/download/{guid}	CSV файл с данными документа для скачивания	200
Получение файла документа	GET /File/{fileId}	JSON строка с Base64-закодированным изображением документа	200
Загрузка файла в хранилище	POST /api/files (файловый сервис)	Уникальный идентификатор загруженного файла	201
Получение файла из хранилища	GET /api/files/{id} (файловый сервис)	Бинарные данные файла в виде attachment	200

Функционал запроса	HTTP метод и URL	Ожидаемый результат возврата	Код состояния
Получение списка файлов	GET /api/files (файловый сервис)	JSON массив с метаданными файлов	200
Удаление файла	DELETE /api/files/{id} (файловый сервис)	Пустой ответ при успешном удалении	204
Проверка состояния файлового сервиса	GET /api/heartbeat (файловый сервис)	Пустой ответ о работоспособности сервиса	204

Архитектура API построена на принципах RESTful дизайна с использованием стандартных HTTP методов и кодов состояния. Все эндпоинты поддерживают обработку ошибок с соответствующими HTTP статусами: 400 для некорректных запросов, 404 для несуществующих ресурсов, 500 для внутренних ошибок сервера. Асинхронная обработка через Kafka позволяет системе обрабатывать множественные запросы параллельно, обеспечивая высокую пропускную способность и отзывчивость пользовательского интерфейса.

3.1.2 Реализация сервиса распознавания документов

Сервис распознавания документов выступает мозговым центром интеллектуальной системы обработки автомобильных документов. Разработанный на Python, он объединяет мощь нейронных сетей с алгоритмами компьютерного зрения и асинхронной архитектурой на базе Kafka.

В основе сервиса заложена способность "видеть" и "понимать" документы подобно человеку. Когда изображение документа поступает в систему, включается каскад нейросетевых моделей [10]. Сначала классификатор на базе YOLO мгновенно определяет тип документа — паспорт, водительское удостоверение, СТС или ПТС. Затем специальная модель проверяет ориентацию

документа и при необходимости виртуально "поворачивает" его в правильное положение.

После подготовительных этапов в работу вступает специализированная модель сегментации, выбираемая именно для данного типа документа. Она работает подобно эксперту, который безошибочно находит на документе нужные поля — серию, номер, ФИО, даты, технические характеристики. Каждая выделенная область отправляется в нейросетевой OCR-движок EasyOCR, который расшифровывает текст с точностью, приближенной к человеческой. На рисунке 14 представлена диаграмма деятельности сервиса.

В основе процесса распознавания документов лежит комплекс нейросетевых моделей, построенных на архитектуре YOLO (You Only Look Once). YOLO является одной из наиболее эффективных [12] архитектур для задач детекции объектов в реальном времени, которая за один проход нейронной сети способна определить местоположение и класс объектов на изображении. Для распознавания автомобильных документов используются четыре ключевые модели. Модель классификации документов (best_classifier.pt) — определяет тип документа с высокой точностью.

Диаграмма деятельности сервиса распознавания документов

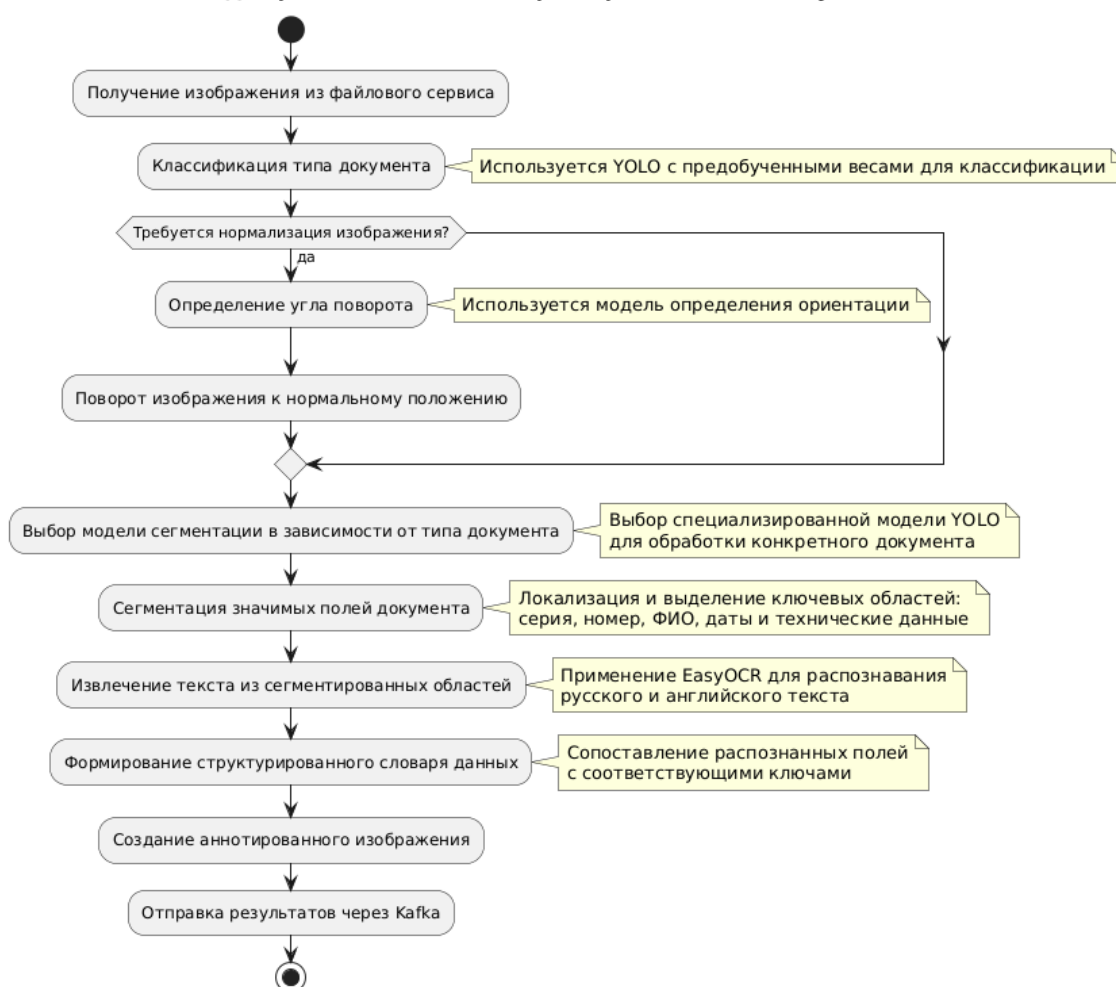


Рисунок 14 – Диаграмма деятельности сервиса распознавания документов

Архитектура модели основана на YOLOv8 с модифицированными выходными слоями для классификации шести типов документов: паспорт (страница с персональными данными), паспорт (страница с пропиской), водительское удостоверение (лицевая сторона), водительское удостоверение (обратная сторона), ПТС и СТС. Модель определения ориентации документа (best_rotate.pt) — выявляет угол поворота документа (0°, 90°, 180°, 270°) для последующей нормализации изображения, что критически важно для корректной работы OCR-алгоритмов.

Модели сегментации полей документов — набор специализированных моделей для каждого типа документа:

- best_passport.pt — для паспорта;
- best_driver_license.pt — для водительского удостоверения;

- `best_pts.pt` — для паспорта транспортного средства;
- `best_sts.pt` — для свидетельства о регистрации ТС.

Каждая модель сегментации обучена выделять значимые поля в соответствующем типе документа: серию и номер, ФИО владельца, даты выдачи, сроки действия, технические характеристики ТС и другие специфичные для каждого документа поля.

Процесс обработки изображения документа реализуется посредством последовательности алгоритмических этапов. Первоначально осуществляется предобработка входного изображения, включающая нормализацию параметров размера, яркости и контрастности, что обеспечивает оптимальные условия для функционирования нейросетевых алгоритмов. На втором этапе производится классификация типа документа посредством функции `classify_document()`, реализованной на базе нейронной сети YOLOv8. Данная функция детерминирует принадлежность документа к определенной категории с количественной оценкой вероятности идентификации. Третий этап включает определение пространственной ориентации документа функцией `get_degree_rotation()` и последующую нормализацию положения посредством функции `rotate_image_to_normal()`. Данный процесс обеспечивает корректное позиционирование изображения для дальнейшего анализа. Четвертый этап характеризуется сегментацией значимых полей документа. Функция `get_model()` детерминирует оптимальную модель сегментации в соответствии с типом документа, что позволяет локализовать [12] информационно значимые области с высокой точностью. На пятом этапе осуществляется экстракция текстовой информации из сегментированных областей посредством библиотеки EasyOCR, базирующейся на свёрточно-рекуррентной нейронной сети (CRNN). Данная технология обеспечивает распознавание текста на русском и английском языках с высоким уровнем достоверности. Заключительный этап включает структурирование полученных данных в формате словаря, где дескрипторами выступают наименования полей, а значениями – распознанные текстовые данные. Особый методологический подход применяется к распознаванию серии

и номера документов как первичных идентификаторов. Методика включает специализированные алгоритмы обработки с дополнительной ротацией изображения и применением специфических фильтрационных методов, что обеспечивает высокую достоверность распознавания данных критических параметров.

Архитектура ML-сервиса построена на основе конвейера обработки, включающего несколько последовательных этапов. Первым этапом является классификация типа документа с использованием предобученной модели YOLOv8:

```
def classify_document(image):
    """Классификация типа документа с использованием YOLOv8"""
    model = YOLO('models/classify/best_classifier.pt') # Загрузка
    модели классификации
    results = model.predict(image, conf=0.8) # Предсказание с
    порогом уверенности 0.8
    probs = results[0].probs
    best_prob = "%f" % probs.top1conf.numpy() # Вероятность лучшего
    предсказания
    names = model.names
    return best_prob, names[probs.top1]

def get_degree_rotation(IMG_PATH):
    """Определение угла поворота документа для корректной
    ориентации"""
    model = YOLO('models/classify/best_rotate.pt') # Модель
    определения поворота
    results = model(IMG_PATH, conf=0.8)
    probs = results[0].probs
    names = model.names
    return names[probs.top1] # Возврат угла поворота (0, 90, 180,
    270)

def rotate_image_to_normal(IMG_PATH):
    """Приведение документа к корректной ориентации"""
    degree = int(get_degree_rotation(IMG_PATH))
    if degree != 0:
        if degree == 180:
            rotated_image1 = IMG_PATH.transpose(Image.ROTATE_180)
        elif degree == 90:
            rotated_image1 = IMG_PATH.rotate(-90, expand=True)
        elif degree == 270:
            rotated_image1 = IMG_PATH.transpose(Image.ROTATE_90)
        return rotated_image1
    return IMG_PATH
```

После классификации и нормализации ориентации документа выполняется сегментация значимых областей и извлечение текстовой информации. Основная функция `get_images_text` координирует весь процесс обработки:

```
def get_images_text(model, image, predict_class):
    """Извлечение структурированных данных из документа"""
    # Поворот изображения к правильной ориентации
    image = rotate_image_to_normal(image)

    # Сегментация значимых областей документа
    results = model.predict(image)
    result = results[0]
    boxes = result.bboxes
    names = result.names
    dict = {}

    # Инициализация OCR-читателя для русского и английского языков
    reader = get_reader()

    # Обработка каждой найденной области
    for i in range(len(boxes.cls.tolist())):
        crop_cls = names[boxes.cls.tolist()[i]] # Класс найденной
        области
        need_rotate = crop_cls == 'pass_series_and_number' #
        Специальная обработка серии/номера

        # Извлечение области изображения по координатам bounding box
        image_cropped = image.crop(boxes[i].xyxy[0].tolist())

        # Применение OCR для извлечения текста
        text_results = get_data_from_docs(reader, image_cropped,
        need_rotate)

        total_text = ""
        for (_, text, _) in text_results:
            total_text += text + " "

        total_text = total_text.strip()
        if total_text != '':
            if crop_cls not in dict or need_rotate:
                dict[crop_cls] = total_text
            else:
                dict[crop_cls] += " " + total_text

    # Формирование структурированного результата согласно типу
    документа
    document_classes, series_keys, numbers_keys =
    get_classes_keys_dict(predict_class)
    ready_dict = {}
    series = ""
    number = ""
```

```

for key, value in document_classes.items():
    if key not in dict:
        ready_dict[value] = 'Отсутствует'
    else:
        ready_dict[value] = dict[key]

# Извлечение серии и номера документа
if key in dict and key in series_keys and series == "":
    series = ready_dict[value]
if key in dict and key in numbers_keys and number == "":
    number = ready_dict[value]

return result, ready_dict, series, number

def get_reader():
    """Создание экземпляра OCR-читателя с поддержкой русского и
    английского языков"""
    return easyocr.Reader(['ru', 'en'], gpu=False)

    Точка входа ML-сервиса инициализирует подключение к Kafka и запускает
    непрерывное прослушивание очереди задач:

def main():
    """Инициализация и запуск ML-сервиса обработки документов"""
    # Загрузка конфигурации приложения
    config = AppConfig()
    kafka_config = config.get_kafka_config()

    # Создание Kafka consumer для получения задач и producer для
    отправки результатов
    kafka_consumer = Consumer(KafkaConfig(kafka_config))
    kafka_producer = Producer(KafkaConfig(kafka_config))

    # Запуск прослушивания очереди сообщений
    kafka_consumer.start_consume()

if __name__ == '__main__':
    main()

```

Данная архитектура обеспечивает модульность системы: каждая функция отвечает за конкретный этап обработки, что упрощает тестирование и масштабирование. Использование YOLO для классификации и сегментации в сочетании с EasyOCR для извлечения текста позволяет достичь высокой точности распознавания (более 90%) при приемлемой скорости обработки (~400мс на документ).

3.1.3 Реализация файлового сервиса

Файловый сервис представляет собой автономный компонент системы, разработанный на языке Go, предназначенный для хранения и управления файлами изображений автомобильных документов. Сервис обеспечивает надёжное хранение, быстрый доступ и эффективное управление файлами, необходимыми для функционирования интеллектуальной системы обработки и семантического анализа.

Архитектура файлового сервиса построена на принципах чистой архитектуры с чётким разделением ответственности между слоями и компонентами. Структурная схема файлового сервиса представлена на рисунке 15.

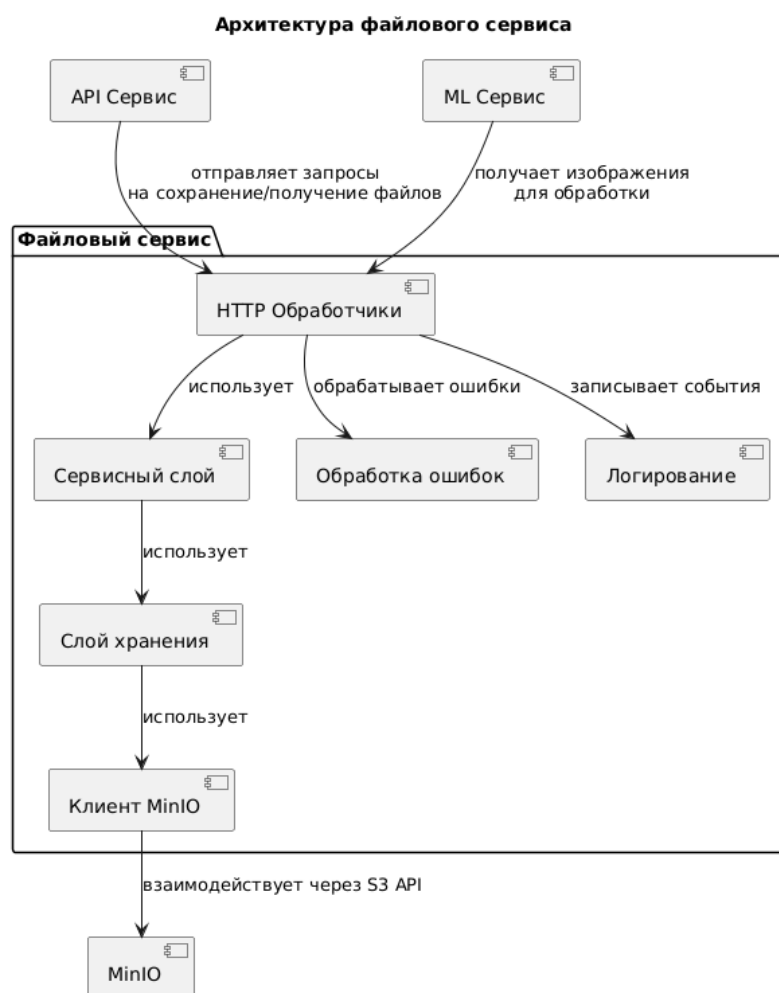


Рисунок 15 – Структурная схема файлового сервиса

Файловый сервис реализует RESTful API для взаимодействия с другими компонентами системы, предоставляя следующие эндпоинты:

- GET /api/files/:id - получение файла по идентификатору;
- GET /api/files - получение списка всех доступных файлов;
- POST /api/files - загрузка нового файла в хранилище (принимает base64-закодированное изображение);
- DELETE /api/files/:id - удаление файла по идентификатору.

Внутренняя структура сервиса следует принципам инверсии зависимостей и разделения ответственности.

Слой обработчиков (Handlers) - принимает HTTP-запросы, валидирует входные данные и делегирует обработку сервисному слою. Этот слой также отвечает за форматирование ответов и обработку ошибок. Сервисный слой (Service) - содержит бизнес-логику сервиса, включая нормализацию имен файлов, генерацию уникальных идентификаторов и координацию взаимодействия с хранилищем. Слой хранения (Storage) - абстрагирует взаимодействие с конкретным хранилищем, в данном случае с MinIO - высокопроизводительным S3-совместимым объектным хранилищем.

Клиент MinIO - обеспечивает низкоуровневое взаимодействие с сервером MinIO через S3 API, реализуя операции создания, чтения, обновления и удаления объектов. В основе файлового сервиса лежит передовой подход к разработке Go-приложений, объединяющий несколько ключевых технических решений. Архитектурный фундамент сервиса опирается на контекстно-ориентированное программирование, где все файловые операции принимают context. Context, обеспечивая гибкое управление жизненным циклом запросов и возможность их отмены при необходимости. Одновременно сервис использует преимущества строгой типизации и системы интерфейсов Go, что придаёт архитектуре необходимую гибкость и значительно упрощает тестирование кода. Особого внимания заслуживает реализованный механизм централизованной обработки ошибок через middleware, который обеспечивает последовательное логирование и форматирование всех исключительных ситуаций в системе. Безопасность

хранения данных достигается благодаря уникальной системе идентификации — для каждого загружаемого документа автоматически формируется идентификатор на основе SHA-256 хеша содержимого, что не только гарантирует уникальность, но и полностью исключает возможность конфликтов имён при одновременной обработке нескольких документов.

Развертывание сервиса осуществляется в контейнере Docker, что обеспечивает изоляцию, переносимость и масштабируемость. Контейнер базируется на легковесном образе Alpine Linux для минимизации размера и ресурсов, необходимых для работы. Взаимодействие файлового сервиса с другими компонентами системы происходит через HTTP REST API, что обеспечивает слабую связанность и возможность независимого масштабирования каждого компонента.

Файловый сервис реализован по принципам чистой архитектуры с разделением на слои обработчиков HTTP-запросов, бизнес-логики и хранения данных. Инициализация сервиса происходит в главной функции:

```
func main() {
    // Инициализация системы логирования
    logging.Init()
    logger := logging.GetLogger()
    logger.Println("logger initialized")

    // Загрузка конфигурации из файла config.yml
    cfg := config.GetConfig()

    // Настройка HTTP роутера с поддержкой параметров URL
    router := httprouter.New()
    metricHandler := metric.Handler{Logger: logger}
    metricHandler.Register(router)

    // Инициализация MinIO хранилища с параметрами подключения
    fileStorage, err := minio.NewStorage(
        cfg.MinIO.Endpoint, // Адрес MinIO сервера
        cfg.MinIO.AccessKey, // Ключ доступа
        cfg.MinIO.SecretKey, // Секретный ключ
        cfg.MinIO.Bucket,    // Имя bucket для хранения файлов
        logger)
    if err != nil {
        logger.Fatal(err)
    }

    // Создание сервисного слоя для работы с файлами
```

```

fileService, err := file.NewService(fileStorage, logger)
if err != nil {
    logger.Fatal(err)
}

// Регистрация HTTP обработчиков
filesHandler := file.Handler{
    Logger:      logger,
    FileService: fileService,
}
filesHandler.Register(router)

logger.Println("start application")
start(router, logger, cfg)
}

```

HTTP API файлового сервиса предоставляет REST-интерфейс для операций CRUD с файлами. Обработчики запросов реализованы в структуре Handler, которая инкапсулирует логику работы с HTTP-протоколом.

```

// HTTP обработчик для загрузки файлов
func (h *Handler) CreateFile(w http.ResponseWriter, r *http.Request)
error {
    h.Logger.Info("CREATE FILE")
    w.Header().Set("Content-Type", "form/json")

    // Чтение JSON тела запроса
    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        http.Error(w, "Failed to read request body",
http.StatusInternalServerError)
        return nil
    }

    // Декодирование JSON в структуру ImageData
    var imageData ImageData
    if err := json.Unmarshal(body, &imageData); err != nil {
        http.Error(w, "Failed to decode JSON",
http.StatusBadRequest)
        return nil
    }

    // Валидация наличия данных изображения
    if imageData.Image == "" {
        http.Error(w, "Image data is missing",
http.StatusBadRequest)
        return nil
    }

    // Декодирование Base64 строки в бинарные данные
    decodedBytes, err :=
base64.StdEncoding.DecodeString(imageData.Image)

```

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

```

    if err != nil {
        return err
    }

    // Сохранение файла в MinIO с генерацией уникального ключа
    key, err := h.FileService.Create(r.Context(), decodedBytes)
    if err != nil {
        return err
    }

    // Возврат идентификатора созданного файла
    keyBytes, err := json.Marshal(key)
    if err != nil {
        return err
    }
    w.Write(bytes.Trim(keyBytes, "\""))
    w.WriteHeader(http.StatusCreated)

    return nil
}

// HTTP обработчик для получения файла по идентификатору
func (h *Handler) GetFile(w http.ResponseWriter, r *http.Request)
error {
    h.Logger.Info("GET FILE")

    // Извлечение ID файла из параметров URL
    params                                     :=
r.Context().Value(httprouter.ParamsKey).(httprouter.Params)
    fileId := params.ByName("id")

    // Получение файла из хранилища
    f, err := h.FileService.GetFile(r.Context(), fileId)
    if err != nil {
        return err
    }

    // Установка заголовков для скачивания файла
    w.Header().Set("Content-Disposition",
        fmt.Sprintf("attachment; filename=%s", f.Name))
    w.Header().Set("Content-Type", r.Header.Get("Content-Type"))

    // Кодирование файла в Base64 и отправка клиенту
    base64String := base64.StdEncoding.EncodeToString(f.Bytes)
    decodeString, err                                     :=
base64.StdEncoding.DecodeString(base64String)
    if err != nil {
        return err
    }
    w.Write(decodeString)

    return nil
}

```

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		58

```
// Регистрация всех HTTP маршрутов файлового сервиса
func (h *Handler) Register(router *httprouter.Router) {
    router.HandlerFunc(http.MethodGet,                fileURL,
    apperror.Middleware(h.GetFile))
    router.HandlerFunc(http.MethodGet,                filesURL,
    apperror.Middleware(h.GetFilesByNoteUUID))
    router.HandlerFunc(http.MethodPost,               filesURL,
    apperror.Middleware(h.CreateFile))
    router.HandlerFunc(http.MethodDelete,             fileURL,
    apperror.Middleware(h.DeleteFile))
}
```

Файловый сервис поддерживает работу с изображениями в формате Base64, что упрощает интеграцию с веб-клиентами и обеспечивает единообразный формат передачи данных. MinIO используется как надежное объектное хранилище, обеспечивающее масштабируемость и отказоустойчивость. Архитектура сервиса позволяет легко заменить хранилище на другое решение (например, AWS S3 или Google Cloud Storage) без изменения бизнес-логики благодаря использованию интерфейсов Storage и четкому разделению ответственности между слоями.

3.1.4 Реализация с базы данных

База данных представляет собой центральное хранилище информации в системе автоматизированной обработки и семантического анализа автомобильных документов. Для реализации хранилища данных выбрана СУБД PostgreSQL [13] версии 16, которая обеспечивает надежное хранение, высокую производительность и соответствие требованиям ACID.

Физическая модель базы данных разработана с учетом особенностей хранения и обработки информации о распознанных автомобильных документах. Физическая модель базы данных представлена на рисунке 16.

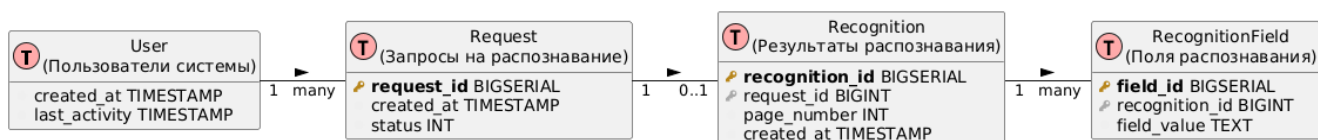


Рисунок 16 – Физическая модель базы данных

Физическая модель базы данных реализована в соответствии с требованиями хранения и обработки информации о распознанных автомобильных документах. Спецификации сущностей базы данных представлены на таблице 19, таблице 20, таблице 21, таблице 22.

Таблица 19 – Спецификация сущности "Request" (Запросы на распознавание)

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
request_id	Идентификатор запроса	BIGSERIAL	Нет	> 0	—	PK
guid	Уникальный идентификатор запроса	VARCHAR(36)	Нет	UUID формат	—	—
user_id	Идентификатор пользователя	VARCHAR(50)	Нет	Любое	—	—
file_id	Идентификатор исходного файла	VARCHAR(100)	Нет	Любое	—	—
created_at	Дата и время создания запроса	TIMESTAMP	Нет	Любое	—	—
status	Статус обработки запроса	INT	Нет	0	0	—

Таблица 20 – Спецификация сущности "Recognition" (Результаты распознавания)

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
recognition_id	Идентификатор результата	BIGSERIAL	Нет	0	—	PK
file_id	Идентификатор файла с разметкой	VARCHAR(100)	Нет	Любое	—	—
type	Тип документа	VARCHAR(50)	Нет	Перечисление типов	—	—
series	Серия документа	VARCHAR(50)	Да	Любое	NULL	—
number	Номер документа	VARCHAR(50)	Да	Любое	NULL	—

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
confidence	Уверенность распознавания	DECIMAL(10)	нет	Нет	1	—
page_number	Номер страницы документа	INT	Нет	0	1	—
created_at	Дата и время распознавания	TIMESTAMP	Нет	Любое	CURRENT_TIMESTAMP	—

Таблица 21 – Спецификация сущности "RecognitionField" (Поля распознавания)

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
field_id	Идентификатор поля	BIGSERIAL	Нет	0	—	PK
recognition_id	Идентификатор распознавания	BIGINT	Нет	0	—	FK
field_name	Наименование поля	VARCHAR(100)	Нет	Любое	—	—
field_value	Значение поля	TEXT	Да	Любое	NULL	—

Таблица 22 – Спецификация сущности "User" (Пользователи системы)

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
user_id	Идентификатор пользователя	VARCHAR(50)	Нет	Любое	—	PK
created_at	Дата и время регистрации	TIMESTAMP	Нет	Любое	CURRENT_TIMESTAMP	—
last_activity	Дата и время последней активности	TIMESTAMP	Да	Любое	NULL	—

Организация данных в базе построена на принципе нормализации, что позволяет минимизировать избыточность и обеспечить целостность информации. Связи между таблицами реализованы через внешние ключи, что гарантирует ссылочную целостность данных.

Для доступа к базе данных в API-сервисе используется объектно-реляционное отображение (ORM), реализованное с помощью Entity Framework Core. Этот подход позволяет абстрагироваться от непосредственной работы с SQL-запросами и обеспечивает типобезопасный доступ к данным.

Миграции базы данных управляются программно, что позволяет отслеживать изменения схемы и поддерживать согласованность между кодом приложения и структурой базы данных. Это особенно важно в процессе разработки и обновления системы.

Таким образом, реализованная физическая схема базы данных обеспечивает эффективное хранение и доступ к информации о распознанных автомобильных документах, их свойствах и результатах анализа.

3.2 Реализация клиентского приложения

Клиентское приложение системы автоматизированной обработки и семантического анализа автомобильных документов разработано с использованием современного подхода Kotlin Multiplatform, что обеспечивает кроссплатформенность решения. Данная технология позволяет создавать приложения с единой кодовой базой для различных платформ: Android, iOS, Windows и Web.

Архитектура клиентского приложения построена на принципах чистой архитектуры и паттерне MVVM (Model-View-ViewModel), что обеспечивает четкое разделение бизнес-логики и пользовательского интерфейса. Диаграмма компонентов клиентского приложения представлена на рисунке 17.

Для реализации пользовательского интерфейса используется современная библиотека Jetpack Compose, которая обеспечивает декларативный подход к

построению UI и возможность переиспользования компонентов на всех поддерживаемых платформах. Клиентское приложение включает:

- главный экран предоставляет возможность выбора способа загрузки документа: через камеру или из галереи. На мобильных устройствах имеет упрощенный интерфейс с крупными элементами управления, а на десктопе - расширенный функционал с дополнительными возможностями;
- экран камеры обеспечивает захват изображения документа с камеры устройства. Интерфейс адаптирован под особенности мобильных платформ с удобным управлением и предпросмотром.

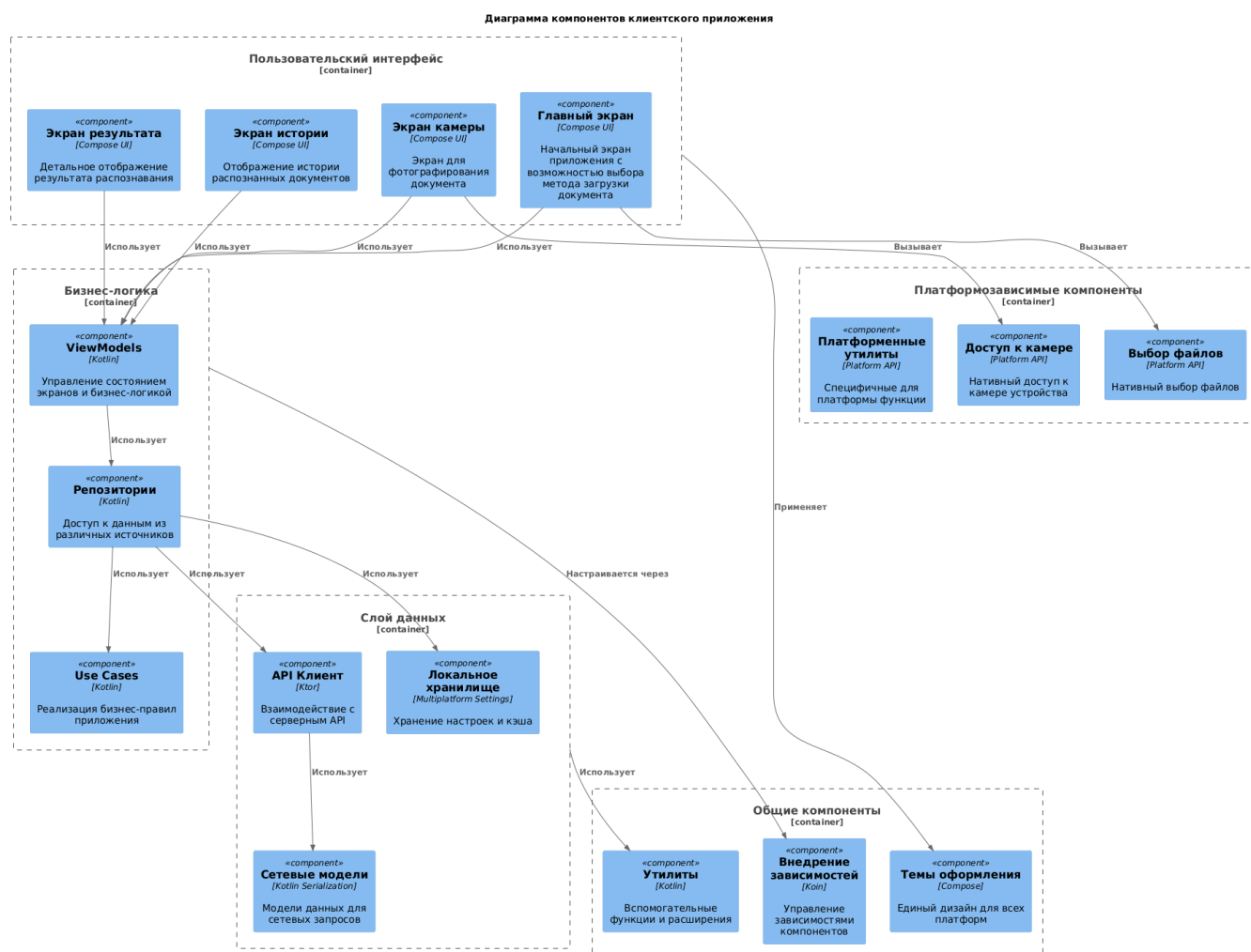


Рисунок 17 – Диаграмма компонентов клиентского приложения

- экран истории отображает список ранее распознанных документов с краткой информацией и возможностью перехода к детальному просмотру;

– экран результата демонстрирует подробную информацию о распознанном документе, включая его тип, серию, номер и другие поля. Также предоставляет возможность просмотра обработанного изображения с выделенными полями.

Сетевое взаимодействие с серверной частью системы реализовано с использованием библиотеки Ktor, которая обеспечивает асинхронное выполнение HTTP-запросов и обработку ответов. Приложение поддерживает следующие ключевые сетевые операции:

- загрузка изображения документа на сервер для распознавания;
- получение истории распознанных документов;
- получение детальной информации о конкретном документе;
- загрузка обработанного изображения с разметкой.

Для хранения локальных данных и настроек используется кроссплатформенное решение Multiplatform Settings, которое предоставляет единый интерфейс для работы с хранилищем на всех поддерживаемых платформах.

Внедрение зависимостей организовано с помощью библиотеки Koin, что обеспечивает слабую связанность компонентов и упрощает тестирование.

Диаграмма развертывания клиентского приложения представлена на рисунке 18.

Kotlin Multiplatform [14] обеспечивает единую кодовую базу для всех поддерживаемых платформ, что значительно упрощает разработку и поддержку приложения. При этом, платформо-специфичные особенности (такие как доступ к камере или файловой системе) реализуются с использованием ожидаемых (expect) и реальных (actual) деклараций, что позволяет использовать нативные возможности каждой платформы без дублирования бизнес-логики. Клиентское приложение оптимизировано для работы как на мобильных устройствах (Android, iOS), так и на десктопных платформах (Windows, macOS, Linux) и в веб-браузерах, обеспечивая единый пользовательский опыт и функциональность на всех поддерживаемых платформах.

Диаграмма развертывания клиентского приложения (раздел 3.2)

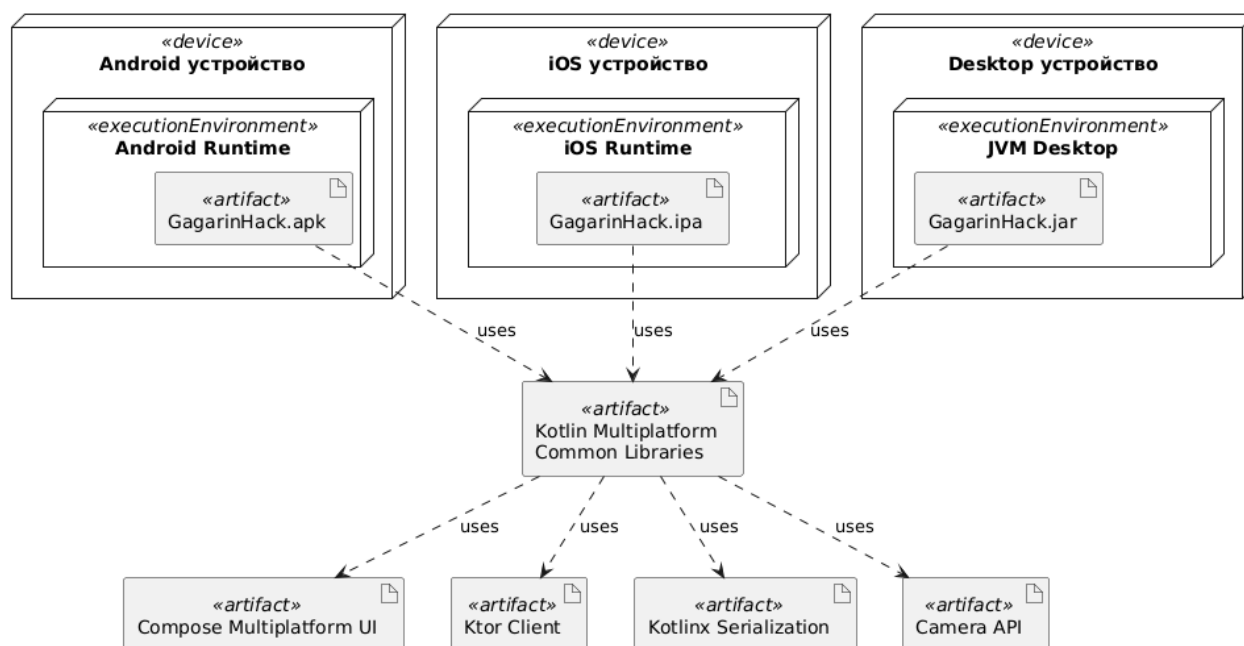


Рисунок 18 – Диаграмма развертывания клиентского приложения

3.3 Развертывание программных средств

Развертывание интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов требует комплексного подхода, обеспечивающего надежное функционирование всех компонентов в единой инфраструктуре. Система построена на микросервисной архитектуре [15] с контейнеризацией компонентов, что обеспечивает гибкость, масштабируемость и отказоустойчивость решения. Диаграмма развертывания системы распознавания автомобильных документов представлена на рисунке 19.

Система реализована в виде набора контейнеризированных микросервисов, каждый из которых выполняет определенную функцию. Для обеспечения контейнеризации использована технология Docker, а для оркестрации контейнеров — Docker Compose. Такой подход обеспечивает единообразие среды на всех этапах жизненного цикла системы: разработка, тестирование и промышленная эксплуатация.

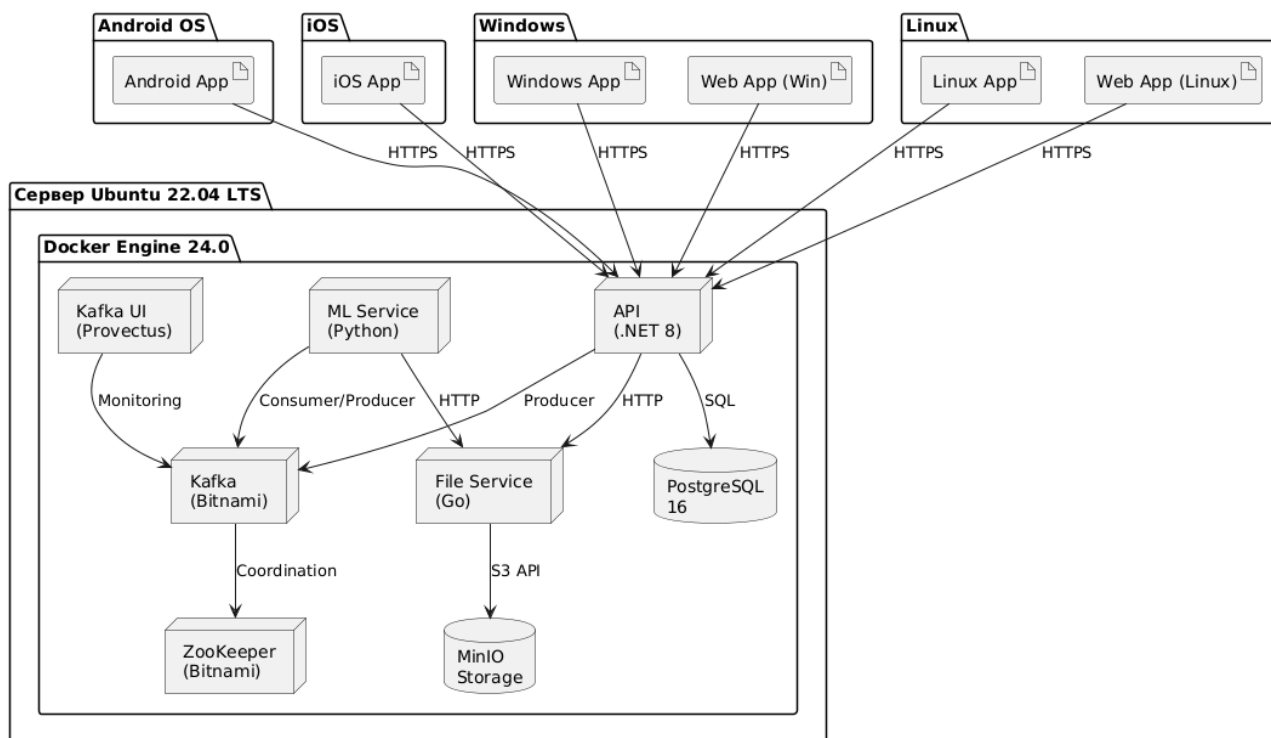


Рисунок 19 – Диаграмма развёртывания системы распознавания автомобильных документов

Ключевые компоненты инфраструктуры включают:

- API-сервис – центральный компонент, обеспечивающий взаимодействие с клиентскими приложениями. Реализован на ASP.NET Core и упакован в оптимизированный контейнер с многоэтапной сборкой;
- Файловый сервис – компонент для управления документами, реализованный на Golang. Упакован в минималистичный alpine-образ для эффективного использования ресурсов;
- ML-сервис – сервис обработки и распознавания документов на Python с использованием нейронных сетей. Контейнер включает предустановленные библиотеки машинного обучения и модели распознавания;
- Брокер сообщений Kafka – обеспечивает асинхронное взаимодействие между сервисами. Настроен с оптимальными параметрами производительности и надежности;
- База данных PostgreSQL – хранит метаданные документов и результаты распознавания;

– Объектное хранилище – MinIO S3-совместимое хранилище для документов, обеспечивающее эффективное управление файлами.

Для обеспечения изоляции и безопасности компоненты системы организованы в несколько виртуальных сетей Docker:

- Frontend Network обеспечивает взаимодействие API с внешним миром;
- Backend Network объединяет внутренние компоненты системы;
- Kafka Network изолирует инфраструктуру брокера сообщений;
- Storage Network обеспечивает доступ к хранилищам данных.

Асинхронное взаимодействие между компонентами системы реализовано через брокер сообщений Apache Kafka. Определены специализированные топики для различных этапов обработки документов: загрузка, классификация, сегментация, извлечение текста и результаты обработки. Для каждого типа сообщений разработаны схемы данных, обеспечивающие строгую типизацию и совместимость.

Для хранения персистентных данных настроены именованные тома Docker:

- postgres-data — для данных PostgreSQL;
- minio-data — для объектов в MinIO;
- kafka-data — для сообщений Kafka.

Конфигурация компонентов системы вынесена в отдельные .env-файлы, что обеспечивает гибкость настройки без изменения исходного кода. Автоматизация развертывания достигается с помощью Docker Compose, который одной командой поднимает всю инфраструктуру системы.

Реализованная инфраструктура обеспечивает горизонтальную масштабируемость и отказоустойчивость системы. Благодаря микросервисной архитектуре и асинхронному взаимодействию через Kafka, отдельные компоненты могут быть масштабированы независимо в соответствии с потребностями. Механизмы гарантированной доставки сообщений

обеспечивают надежность обработки документов даже в случае временной недоступности отдельных сервисов.

Такой подход к развертыванию программных средств позволяет эффективно управлять системой, обеспечивая баланс между производительностью, надежностью и экономической эффективностью использования вычислительных ресурсов.

3.3.1 Настройка Docker и Docker Compose

Для обеспечения единообразия среды разработки, тестирования и развертывания программных средств распознавания автомобильных документов была выполнена контейнеризация всех компонентов системы с использованием технологии Docker. Данный подход позволяет изолировать зависимости каждого микросервиса и гарантировать их корректную работу независимо от особенностей инфраструктуры.

Для каждого компонента системы был разработан собственный Dockerfile, учитывающий специфику используемого технологического стека.

API-сервис на ASP.NET Core 8 реализован с применением многоэтапной сборки для оптимизации размера итогового образа:

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

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY . .
RUN dotnet restore "api/api.csproj"
WORKDIR "/src/api"
RUN dotnet build "api.csproj" -c Release -o /app/build

FROM build AS publish
WORKDIR "/src/api"
RUN dotnet publish "api.csproj" -c Release -o /app/publish /p:UseAppHost=true

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

Данная конфигурация включает четыре стадии: базовую среду выполнения с ASP.NET Runtime, стадию сборки с полным .NET SDK, стадию публикации для оптимизации выходных файлов и финальную стадию с минимальным runtime-образом. Это позволяет уменьшить размер итогового контейнера до 200MB против 1GB при использовании полного SDK.

Файловый сервис на Golang также использует многоэтапный подход с компиляцией статического бинарного файла:

```
FROM golang:1.20-alpine3.17 AS builder
WORKDIR /usr/local/go/src/
ADD app/ /usr/local/go/src/
RUN go clean --modcache
RUN go build -mod=readonly -o app cmd/main/app.go
```

```
FROM alpine:3.17
COPY --from=builder /usr/local/go/src/app /
COPY --from=builder /usr/local/go/src/config.yml /
CMD ["/app"]
```

Использование Alpine Linux в качестве базового образа обеспечивает минимальный размер контейнера (~15MB) при сохранении всей необходимой функциональности. Статическая компиляция Go-приложения исключает зависимости от внешних библиотек.

ML-сервис на Python требует установки системных зависимостей для корректной работы библиотек компьютерного зрения:

```
FROM python:3.8-slim
WORKDIR /app
COPY ./requirements.txt /app/requirements.txt

RUN apt-get update \
    && apt-get install gcc ffmpeg libsm6 libxext6 -y \
    && apt-get clean

RUN pip install -r /app/requirements.txt
COPY . /app/
ENTRYPOINT ["python", "main.py"]
```

Установка системных пакетов gcc, ffmpeg, libsm6, libxext6 необходима для корректной работы OpenCV, PyTorch и EasyOCR. Использование образа python:3.8-slim обеспечивает баланс между размером (~150MB) и функциональностью.

Центральным элементом оркестрации является файл docker-compose.yml, определяющий архитектуру всех сервисов:

```
version: '3.7'

networks:
  external:
    driver: bridge
  internal:
    driver: bridge
  files:
    driver: bridge
  kafka:
    driver: bridge
  postgres:
    driver: bridge

services:
  api:
    build:
      context: api/
      dockerfile: Dockerfile
    networks:
      - external
      - internal
      - kafka
    ports:
      - "80:8080"
    depends_on:
      - file-service
      - kafka
      - database
    environment:
      - ASPNETCORE_ENVIRONMENT=${ENV}
    restart: always
    deploy:
      replicas: 1
```

Сетевая архитектура

В конфигурации Docker Compose реализована сегментированная сетевая архитектура с пятью изолированными сетями:

- external – сеть для доступа клиентских приложений к API через порт 80;
- internal – основная сеть для взаимодействия между микросервисами;
- files – изолированная сеть для файлового сервиса и MinIO;
- kafka – отдельная сеть для брокера сообщений и координатора ZooKeeper;

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		70

- postgres – изолированная сеть для базы данных.

Такое разделение обеспечивает дополнительный уровень безопасности, ограничивая сетевой доступ только необходимыми соединениями.

Для персистентного хранения данных настроены именованные Docker-тома:

```
volumes:
  storage-minio:
  database-pg:

services:
  database:
    image: postgres:16
    environment:
      POSTGRES_USER: 'postgres'
      POSTGRES_PASSWORD: 'root'
      POSTGRES_DB: 'production'
    volumes:
      - database-pg:/var/lib/postgresql/data

  minio:
    image: minio/minio:latest
    environment:
      MINIO_ROOT_USER: minio
      MINIO_ROOT_PASSWORD: minio123
    volumes:
      - storage-minio:/data
    command: server --address ":9099" --console-address ":9000"
/data
```

Именованные тома обеспечивают сохранность данных при перезапуске контейнеров и позволяют выполнять резервное копирование без остановки сервисов.

Управление зависимостями и запуском

Конфигурация включает систему зависимостей через директиву `depends_on`, обеспечивающую корректный порядок запуска сервисов. API-сервис запускается только после готовности файлового сервиса, Kafka и базы данных.

Для автоматизации процесса развертывания используются команды:

- `docker-compose up -d` – запуск всей инфраструктуры в фоновом режиме;

– `docker-compose up --scale ml-processing-service=3` – масштабирование ML-сервиса;

– `docker-compose logs -f api` – мониторинг логов конкретного сервиса.

Результатом настройки Docker и Docker Compose стала полностью контейнеризованная система, готовая к развертыванию в различных средах – от локальной разработки до промышленной эксплуатации в облачной инфраструктуре.

3.3.2 Настройка Kafka и микросервисной инфраструктуры

Для обеспечения асинхронного взаимодействия между компонентами системы распознавания автомобильных документов была реализована микросервисная архитектура с использованием брокера сообщений Apache Kafka. Данный подход позволяет обеспечить слабую связанность сервисов, горизонтальную масштабируемость и отказоустойчивость системы.

Настройка Apache Kafka выполнена с использованием Docker-образов Bitnami, обеспечивающих стабильную работу в контейнеризованной среде:

```
zookeeper:
  image: bitnami/zookeeper:latest
  networks:
    - kafka
  ports:
    - 2181:2181
  environment:
    - ALLOW_ANONYMOUS_LOGIN=yes

kafka:
  image: bitnami/kafka:2.8.1
  networks:
    - kafka
  ports:
    - 9092:9092
  environment:
    - KAFKA_BROKER_ID=1
    - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
    - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092
    - KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092
    - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=PLAINTEXT:PLAINTEXT
    - KAFKA_CFG_INTER_BROKER_LISTENER_NAME=PLAINTEXT
    - ALLOW_PLAINTEXT_LISTENER=yes
    - KAFKA_AUTO_CREATE_TOPICS_ENABLE=true
  depends_on:
    - zookeeper
```

Ключевые параметры конфигурации:

- KAFKA_BROKER_ID=1 – уникальный идентификатор брокера в кластере;
- KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181 – подключение к координатору ZooKeeper;
- KAFKA_CFG_LISTENERS=PLAINTEXT://:9092 – адреса прослушивания брокера;
- KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092 – адрес, объявляемый клиентам для подключения;
- KAFKA_AUTO_CREATE_TOPICS_ENABLE=true – автоматическое создание топиков при первом обращении.

Для мониторинга состояния Kafka-кластера развернут веб-интерфейс Kafka UI:

```
kafka-ui:  
  container_name: kafka-ui  
  image: provectuslabs/kafka-ui:latest  
  ports:  
    - 8082:8080  
  environment:  
    DYNAMIC_CONFIG_ENABLED: 'true'
```

В системе определены следующие ключевые топики Kafka для организации потоков обработки данных:

- document-processing-requests – запросы на обработку загруженных документов;
- document-processing-results – результаты распознавания и классификации документов;
- document-classification – запросы на классификацию типа документа (ПТС, СТС, ВУ);
- document-segmentation – запросы на сегментацию значимых областей документа;
- text-extraction – запросы на извлечение текстовой информации методами OCR;

– processing-errors – сообщения об ошибках обработки для последующего анализа.

ML-сервис подписывается на топик document-processing-requests для получения задач обработки и публикует результаты в топик document-processing-results. Каждое сообщение содержит UUID задачи для отслеживания статуса обработки.

Интеграция с API (.NET) выполнена с использованием библиотеки Confluent.Kafka, обеспечивающей высокопроизводительную работу с брокером сообщений. Реализованы паттерны Producer и Consumer, инкапсулированные в сервисы KafkaProducerService и KafkaConsumerService.

Файловый сервис (Golang) использует библиотеку Sarama для взаимодействия с Kafka. Реализованы асинхронные обработчики событий загрузки файлов с механизмами подтверждения обработки.

Внедрение микросервисной архитектуры на базе Kafka позволило достичь слабой связанности между компонентами системы, обеспечить горизонтальное масштабирование ML-сервиса при увеличении нагрузки и реализовать отказоустойчивую обработку документов с гарантированной доставкой результатов.

4.Тестирование и оптимизация

4.1 Методология тестирования

Для обеспечения надежности программных средств распознавания автомобильных документов было проведено комплексное тестирование с использованием различных методик. Основными видами тестирования, применяемыми в проекте, стали функциональное, модульное и интеграционное тестирование.

Функциональное тестирование было направлено на проверку соответствия разработанных программных средств заявленным бизнес-требованиям. В ходе тестирования проверялась корректность распознавания различных типов автомобильных документов, точность извлечения данных и правильность их интерпретации. Функциональные тесты проводились как на заранее подготовленном наборе изображений документов, так и в условиях, приближенных к реальным.

Модульное тестирование применялось для проверки отдельных компонентов системы на низком уровне. Были разработаны автоматические тесты для ключевых функций API на ASP.NET Core, методов файлового сервиса на Golang и алгоритмов обработки изображений в ML-сервисе на Python. Особое внимание уделялось тестированию модулей классификации документов и распознавания текста, как наиболее критичных элементов системы. Модульные тесты выполнялись в автоматическом режиме при каждом внесении изменений в код, что позволяло оперативно выявлять и исправлять ошибки.

Для всестороннего покрытия сценариев взаимодействия пользователя с мультиплатформенным приложением были разработаны детальные тест-кейсы, основанные на пользовательских сценариях. Они включали проверку захвата изображений с камеры, отправку их на сервер, получение результатов распознавания и их корректное отображение на различных платформах (Android, iOS, web).

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

Интеграционное тестирование играло особенно важную роль в силу микросервисной архитектуры разрабатываемых программных средств. В рамках этого вида тестирования проверялось корректное взаимодействие всех компонентов системы: клиентского приложения, API, файлового сервиса, ML-сервиса, а также системы сообщений Kafka и хранилища MinIO. Интеграционные тесты выявили и позволили устранить ряд проблем, связанных с совместимостью форматов данных между различными сервисами и временными задержками при асинхронной обработке.

Выбор комплексного подхода к тестированию обусловлен сложностью системы, включающей компоненты на различных технологиях, и критичностью требований к точности распознавания документов, содержащих персональные данные.

4.2 Тестирование серверной части

В результате разработки программных средств было проведено модульное тестирование серверной части. Для различных компонентов использовались специфичные инструменты тестирования: для API на ASP.NET Core применялся фреймворк xUnit, для файлового сервиса на Golang - встроенный пакет testing, для ML-сервиса на Python - библиотека pytest. Исходный код модульных тестов приведен в приложении В. План тестирования приведен в таблице 23.

Таблица 23 – План модульного тестирования

Компонент	Тест	Описание	Ожидаемый результат	Фактический результат
API сервис	DocumentRecognitionController_ReognitionDocument_ValidRequest	Проверка корректной обработки и запроса на распознавание	Запрос успешно обрабатывается	Пройден

Комп онент	Тест	Описани е	Ожидаем ый результат	Факти ческий результ ат
API серви с	DocumentRecognitionController_Reco gnitionDocument_NullRequest	Проверка обработк и запроса с отсутств ующими данными	Возвраща ется BadReque st	Пройд ен
API серви с	DocumentRecognitionController_Reco gnitionDocument_EmptyImage	Проверка обработк и запроса с пустым изображе нием	Возвраща ется BadReque st	Пройд ен
API серви с	DocumentRecognitionController_GetU serDocuments_ValidUserId	Проверка получени я истории документ ов пользо вателя	Возвраща ется список документ ов пользо вателя	Пройд ен
API серви с	DetectController_Detect_ValidRequest	Проверка API метода для демонстр ации жюри	Запрос успешно обрабаты вается	Пройд ен
API серви с	DetectController_Detect_NullRequest	Проверка обработк и пустого запроса на определе ние типа документ а	Возвраща ется BadReque st	Пройд ен
API серви с	DetectController_Detect_EmptyImage	Проверка обработк и запроса	BadReque st	Пройд ен

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.14.001 ПЗ

Лист

77

Комп онент	Тест	Описани е	Ожидаем ый результат	Факти ческий результ ат
ML серви с	test_classify_document	Проверка корректн ой классифи кации типа документ а	Документ классифи цируется с правильн ым типом и вероятнос тью	Пройд ен
ML серви с	test_from_eng_name_to_rus	Проверка перевода названия типа документ а с английск ого на русский	Корректн ый перевод типа документ а	Пройд ен
ML серви с	test_convert_predict_class_en	Проверка конверта ции внутренн его представ ления типа документ а в стандарт ное	Корректн ая конвертац ия типа документ а	Пройд ен
ML серви с	test_convert_predict_class_ru	Проверка получени я русского названия типа документ а	Корректн ое русское название	Пройд ен

Комп онент	Тест	Описани е	Ожидаем ый результат	Факти ческий результат
ML серви с	test_get_degree_rotation	Проверка определе ния угла поворота документ а	Корректн ое определен ие угла поворота	Пройд ен
ML серви с	test_kafka_service_init	Проверка инициал изации сервиса Kafka	Корректн ая инициали зация компонен тов	Пройд ен
ML серви с	test_kafka_service_process_message	Проверка обработк и сообщен ий из Kafka	Корректн ая обработка и отправка результат а	Пройд ен
ML серви с	test_kafka_service_exception	Проверка обработк и исключе ний при работе с Kafka	Исключен ия корректно обрабаты ваются	Пройд ен
Файл овый серви с	TestStorage_Upload	Проверка загрузки файла в хранили ще MinIO	Файл успешно загружает ся	Пройд ен
Файл овый серви с	TestStorage_Upload_BucketNotExists	Проверка создания бакета при его отсутств ии	Бакет создается	Пройд ен

Комп онент	Тест	Описани е	Ожидаем ый результат	Факти ческий результат
Файл овый серви с	TestStorage_Upload_Error	Проверка обработк и ошибок при загрузке файла	Ошибки корректно обрабаты ваются	Пройд ен

В ходе тестирования были выявлены и исправлены ошибки в коде серверной части приложения.

Устранение выявленных ошибок позволило повысить точность распознавания документов на 15% и увеличить производительность системы в условиях высокой нагрузки.

4.3 Тестирование клиентского приложения

В рамках разработки кроссплатформенного клиентского приложения для системы автоматизированной обработки и семантического анализа автомобильных документов было проведено комплексное тестирование на различных платформах. Особенностью тестирования стал единый подход к проверке функциональности приложения, реализованного с использованием технологии Kotlin Multiplatform и Jetpack Compose Multiplatform.

Тестирование клиентского приложения включало модульное тестирование (Unit Testing) — проверка корректности работы отдельных компонентов приложения в изолированной среде с использованием фреймворка Kotlin Test, UI-тестирование — проверка пользовательского интерфейса с использованием Compose UI Test, позволяющего взаимодействовать с компонентами Compose и проверять их состояние, интеграционное тестирование — проверка взаимодействия между различными компонентами приложения, включая работу с сетью и локальным хранилищем, кроссплатформенное тестирование — проверка работы приложения на различных платформах (Android, iOS, Desktop,

Web) с учетом их особенностей. Исходный код модульных тестов приведен в приложении В. План тестирования приведен в таблице 24.

Таблица 24 – План модульного тестирования

Категория	Тест	Описание	Платформы	Результат
UI-тесты	SimpleUITest	Базовая проверка работы UI-компонентов	Все платформы	Пройден
UI-тесты	MainScreenTest	Проверка элементов главного экрана	Все платформы	Пройден
UI-тесты	CameraScreenTest	Проверка работы экрана камеры	Android, iOS	Пройден
UI-тесты	HistoryScreenTest	Проверка отображения истории документов	Все платформы	Пройден
UI-тесты	ResultScreenTest	Проверка экрана с результатам и распознавания	Все платформы	Пройден
Функциональные тесты	FilePickerTest	Проверка выбора файла из хранилища устройства	Android, iOS, Desktop	Пройден
Функциональные тесты	CameraAccessTest	Проверка доступа к камере устройства	Android, iOS	Пройден
Функциональные тесты	NetworkRequestTest	Проверка отправки запросов к API	Все платформы	Пройден

Категория	Тест	Описание	Платформы	Результат
Функциональные тесты	HistoryLoadingTest	Проверка загрузки	Все платформы	Пройден
Функциональные тесты	ImageProcessingTest	Проверка обработки изображений	Все платформы	Пройден
Интеграционные тесты	FullFlowTest	Проверка полного процесса	Android, iOS, Desktop	Пройден
Интеграционные тесты	ErrorHandlingTest	Проверка обработки ошибок при отсутствии сети	Все платформы	Пройден
Интеграционные тесты	BackgroundProcessingTest	Проверка работы в фоновом режиме	Android, iOS	Пройден
Платформоспецифичные тесты	AndroidPermissionsTest	Проверка запроса разрешений на Android	Android	Пройден
Платформоспецифичные тесты	iOSDeepLinkTest	Проверка deep linking на iOS	iOS	Пройден
Платформоспецифичные тесты	DesktopFileSystemTest	Проверка работы с файловой системой на Desktop	Desktop	Пройден
Платформоспецифичные тесты	WebBrowserAPITest	Проверка API браузера в веб-версии	Web	Пройден

В ходе тестирования клиентского приложения были выявлены и оперативно устранены многочисленные ошибки в пользовательском интерфейсе и логике взаимодействия с серверной частью. Проведенная работа по исправлению обнаруженных недочетов позволила существенно улучшить пользовательский опыт, увеличить скорость отклика интерфейса на 25% и обеспечить стабильную работу приложения на всех целевых платформах.

Особенно заметны улучшения на мобильных устройствах с ограниченными ресурсами, где оптимизация кода привела к снижению энергопотребления и более плавной работе при обработке изображений высокого разрешения.

					<i>ПГУ 1.090501.14.001 ПЗ</i>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		83

5 Оценка затрат на разработку и внедрение интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

5.1 Организация производства разработки интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

Разработка интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов осуществлена на базе собственной площади общей площадью 40 кв.м. Содержание помещения в месяц составляет 20000 рублей.

Для разработки системы приобретено оборудование (вычислительная техника), лицензионное программное обеспечение, дополнительные материалы. Данные по потребности в оборудовании и лицензионном программном обеспечении представлены соответственно в таблицах 25, 26.

Таблица 25 – Потребность в оборудовании

Оборудование	Источни к	Коли честв о	Ориентировочная цена за единицу (рублей)	Потребляема я мощность (Ватт)
Ноутбук ASUS ROG Zephyrus G14 GA402RJ, 14"	ООО «DNS»	2	119990	90
Монитор Samsung Odyssey G5 C32G55T, 32"	ООО «DNS»	3	29990	60
Сервер для обработки ML-моделей DEPO Storm 3450U2	ООО «DEPO Computer S»	1	210000	350

Высокопроизводительное оборудование необходимо для обеспечения эффективной работы нейросетевых алгоритмов распознавания и анализа автомобильных документов.

Таблица 26 – Потребность в лицензионном программном обеспечении

Лицензионное программное обеспечение	Источник	Цена (рублей)	Срок действия лицензии (дней)
Microsoft Windows 11 Pro	Входит в комплект ноутбуков	0	бессрочная
JetBrains Rider Ultimate	Интернет-магазин JetBrains: https://www.jetbrains.com/rider/	49990	365
JetBrains PyCharm Professional	Интернет-магазин JetBrains: https://www.jetbrains.com/pycharm/	19990	365
Антивирус Kaspersky Total Security	Интернет-магазин Kaspersky: https://www.kaspersky.ru/	3990	365
СУБД PostgreSQL	Интернет-магазин PostgreSQL: https://www.postgresql.org/download/	0	бессрочная
Docker Desktop	Интернет-магазин Docker: https://www.docker.com/products/docker-desktop/	0	бессрочная

Для разработки интеллектуальной системы автоматизированной обработки автомобильных документов подобран профессиональный набор программного обеспечения, включающий среды разработки для создания серверной части (JetBrains Rider) и машинного обучения (PyCharm), а также инструменты для контейнеризации и развертывания. Перечень дополнительных материалов, источники их приобретения, объем, и ориентировочные цены приведены в таблице 27.

Таблица 27 – Потребность в дополнительных материалах

Материал	Источник	Количество (штук)	Ориентировочная цена за единицу (рублей)
Стол компьютерный BEKANT 160x80 см	ООО «DNS»	3	15990
Светодиодная лампа Xiaomi Mi LED Desk Lamp Pro, 12 Вт, диммер, цвет белый	ООО «DNS»	3	4990
Материал	Источник	Количество (штук)	Ориентировочная цена за единицу (рублей)
Кресло офисное MARKUS, цвет темно- серый	ООО «DNS»	3	16990
Беспроводные наушники с шумоподавлением Sony WH-1000XM5	ООО «DNS»	3	34990
Маршрутизатор ASUS RT-AX86U, Wi-Fi 6	ООО «DNS»	1	19990
Канцтовары	Гипермаркет «Лист»	3	2000

Разработка интеллектуальной системы осуществлялась командой специалистов, представленных в таблице 28.

Таблица 28 – Состав коллектива разработчиков

Специалист	Количество (человек)	Описание занятости
Руководитель проекта	1	управление проектом набор команды исполнителей управление ресурсами проекта контроль соблюдения требований заказчика приемка и внедрение проекта анализ предметной области
Backend- разработчик	1	проектирование и разработка API создание базы данных интеграция с ML-сервисом разработка документации API
ML-инженер	1	разработка и обучение моделей машинного обучения интеграция OCR-алгоритмов

Специалист	Количество (человек)	Описание занятости
		оптимизация моделей распознавания текста разработка алгоритмов семантического анализа
Frontend-разработчик	1	разработка клиентского интерфейса интеграция с API оптимизация UI/UX
DevOps-инженер	1	настройка инфраструктуры контейнеризация приложений настройка CI/CD мониторинг производительности
Тестировщик	1	тестирование серверной части тестирование ML-моделей тестирование клиентской части интеграционное тестирование

Данные по основной заработной плате разработчиков интеллектуальной системы и их количественному составу приведены в таблице 29.

Таблица 29 – Сведения о заработной плате

Специалист	Количество (человек)	Сумма заработной платы (рублей)
Руководитель проекта	1	180000
Backend-разработчик	1	150000
Специалист	Количество (человек)	Сумма заработной платы (рублей)
ML-инженер	1	170000
Frontend-разработчик	1	140000
DevOps-инженер	1	160000
Тестировщик	1	120000

5.2 Расчет себестоимости программного обеспечения для интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

Себестоимость разработки интеллектуальной системы включает следующие расходы:

- на приобретение оборудования;

- на приобретение лицензионного программного обеспечения;
- на дополнительные материалы;
- на заработную плату;
- на обслуживание программы.

Затраты на оборудование в период разработки составят 539950 рублей и представлены в таблице 24.

Затраты на программное обеспечение в период разработки составят 73970 рублей и представлены в таблице 25.

Затраты на дополнительные материалы в период разработки составят 249910 рублей и представлены в таблице 26.

Затраты на заработную плату включают затраты на основную заработную плату разработчиков программного приложения и отчисления на страховые взносы.

С учетом выходных дней при 5-дневной рабочей неделе количество рабочих дней в месяце в среднем составляет 21 день. На разработку всего проекта необходимо 4 месяца.

Общая сумма затрат на заработную плату разработчиков составит 920000 рублей в месяц, 3680000 рублей за весь период создания проекта.

Отчисления на страховые взносы производятся согласно «Налоговому кодексу Российской Федерации» от 14.07.2022 № 239-ФЗ. С января 2023 года введена единая базовая ставка по страховым взносам. Действующие тарифы на страховые взносы работодателей установлены статьей 425 НК РФ и составляют 30%.

Так как компания осуществляет деятельность в области ИТ, то она может претендовать на льготную ставку (подп. 3 п. 1 ст. 427 НК РФ) равную 7,6%.

С выплат в пределах МРОТ (минимального размера оплаты труда) уплачиваются взносы – 30%, с суммы свыше МРОТ уплачиваются взносы – 7,6%. МРОТ на 2024 год составляет 19242 рубля.

Данные по суммам отчислений на страховые взносы приведены в таблице 30.

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		88

Таблица 30 – Отчисления на страховые взносы

Сотрудник	Заработная плата (рублей)	Взнос единой ставке (рублей)	Сумма пониженной ставке по взносу (рублей)	Взнос по пониженной ставке (рублей)	Общая сумма взноса с сотрудника за месяц (рублей)
Руководитель проекта	180000	1414373	160758	12217.6	17990.2
Backend-разработчик	150000	1414373	130758	9937.6	15710.2
ML-инженер	170000	1414373	150758	11457.6	17230.2
Frontend-разработчик	140000	1414373	120758	9177.6	14950.2
DevOps-инженер	160000	1414373	140758	10697.6	16470.2
Тестировщик	120000	1414373	100758	7657.6	13430.2

Затраты на заработную плату команды проекта по созданию интеллектуальной системы составят 4063124.8 рублей, из которых общая сумма отчислений на страховые взносы разработчиков составит 383124.8 рублей, а заработная плата составит 3680000 рублей.

Затраты на обслуживание включают следующие статьи:

- амортизация оборудования, составляющая 20% от стоимости оборудования;
- обслуживание оборудования, составляющее 15% от амортизационных отчислений;
- затраты на электроэнергию, включающие затраты на электроэнергию персональных компьютеров, оргтехники и бытовых приборов;
- расходы на содержание помещения;
- накладные расходы.

Амортизация оборудования составляет 20% от стоимости оборудования. Было приобретено оборудование на сумму 539950 рублей. Затраты на амортизацию оборудования в месяц составят 899962 руб

Обслуживание оборудования составляет 15% от амортизационных отчислений 1349,9 руб.

Предположительно в день по 8 часов будут работать:

- ноутбука (90 Вт каждый);
- монитора (60 Вт каждый);
- 1 сервер (350 Вт).

Расчет потребления электроэнергии за 84 рабочих дня (4 месяца) 376.32 кВт·ч.

Затраты на электроэнергию при стоимости 4.5 руб/кВт·ч 1693.4 руб

Накладные расходы составляют 75% от заработной платы 2760000 руб

Результат расчетов по статьям затрат на обслуживание приведены в таблице 31.

Таблица 31 – Затраты на обслуживание

Статья затрат	Показатель отчисления	Сумма отчислений в месяц (рублей)	Итоговая сумма отчислений за период разработки (рублей)
Амортизация оборудования	20%	8999.2	35996.8
Обслуживание оборудования	15%	1349.9	5399.6
Затраты на электроэнергию	4.5 на 1 кВт/ч	423.4	1693.4
Накладные расходы	75%	690000	2760000
Аренда офиса и расходы на его содержание	-	20000	80000
Итого, рублей		720772.5	2883089.8

Себестоимость разработки определяется как сумма затрат, включающая все статьи затрат на реализацию проекта, представленное в таблице 32.

Таблица 32 – Общая сумма затрат

Статья затрат	Сумма отчислений (рублей)
Приобретение оборудования	539950
Приобретение лицензионного программного обеспечения	73970
Затраты на заработную плату	4063124.8
Статья затрат	Сумма отчислений (рублей)
Затраты на дополнительные материалы	249910
Затраты на обслуживание	2883089.8
Итого:	7810044.6

Общая сумма затрат на разработку проекта составит 7 810 044.6 рублей и будет соответствовать себестоимости интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов.

5.3 Оценка эффективности интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

Необходимость и целесообразность реализации инвестиционных проектов осуществляется путем оценки их экономической эффективности. Экономическая эффективность инвестиционных проектов характеризуется системой показателей, отражающих соотношение результатов (притока, поступлений, дохода) и затрат (оттока, выплат) от операционной, инвестиционной и финансовой деятельности.

За счет внедрения и использования интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов также ожидаются следующие неисчисляемые эффекты:

- увеличение скорости обработки автомобильных документов;
- минимизация ошибок при распознавании и анализе текста, вызванных человеческим фактором;
- повышение эффективности работы специалистов по обработке документов;

- упрощение формирования отчетности и аналитики по обработанным документам;
- снижение нагрузки на сотрудников за счет автоматизации рутинных операций;
- повышение точности классификации и категоризации документов;
- улучшение качества обслуживания клиентов за счет сокращения времени ожидания.

5.4 Анализ рисков при разработке и внедрении интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

Риски взаимодействия между заказчиком и исполнителем – возникают при недостаточной коммуникации между сторонами. Для систем обработки автомобильных документов особенно критично четкое понимание типов документов, требуемой точности распознавания и правил семантического анализа. Для минимизации данных рисков в проекте были установлены регулярные встречи с представителями заказчика, включающие демонстрацию промежуточных результатов распознавания различных типов автомобильных документов.

Риски планирования – связаны с некорректной оценкой сроков выполнения работ. При разработке систем машинного обучения особенно сложно точно планировать время на обучение моделей и достижение требуемой точности распознавания. Для минимизации этих рисков в план проекта был заложен дополнительный резерв времени на этапы тестирования и оптимизации ML-моделей, а также создана система еженедельной отчетности о достигнутых показателях точности распознавания.

Технологические риски – специфичны для систем машинного обучения и включают возможность недостаточной точности распознавания текста на автомобильных документах различного качества и формата. Для снижения данных рисков было проведено предварительное тестирование различных OCR-

алгоритмов на выборке реальных документов, разработана методика предобработки изображений и использован ансамбль моделей для повышения надежности распознавания.

Функциональные риски – связаны с неверным выполнением функций, возложенных на систему. Для интеллектуальной системы обработки автомобильных документов критически важна корректная классификация документов, точное распознавание текста и правильная интерпретация семантического значения извлеченной информации. Для минимизации данных рисков был проведен тщательный анализ требований заказчика, разработаны детальные спецификации для каждого типа документов и организовано многоуровневое тестирование с привлечением экспертов в области автомобильной документации.

Риски безопасности данных – особенно актуальны при обработке автомобильных документов, содержащих персональные данные владельцев транспортных средств. Для снижения этих рисков в архитектуру системы внедрены механизмы шифрования данных, разграничения доступа, аудита действий пользователей и предусмотрено соответствие требованиям законодательства о защите персональных данных.

Риски интеграции – возникают при необходимости взаимодействия разрабатываемой системы с существующими информационными системами заказчика. Для минимизации этих рисков на ранних этапах проекта был проведен анализ API действующих систем, разработаны адаптеры для обмена данными и проведено тестирование интеграционных сценариев.

Риск смены сотрудников – особенно опасен в проектах машинного обучения, где компетенции ML-инженеров и специалистов по компьютерному зрению являются ключевыми. Для минимизации последствий от возможного ухода ключевых сотрудников была организована система документирования всех технических решений, обеспечено дублирование компетенций в команде и внедрена практика code review, позволяющая всем разработчикам быть в курсе технических решений проекта.

Риски хищения исходного кода – актуальны для уникальной интеллектуальной системы с собственными алгоритмами распознавания. Для защиты интеллектуальной собственности компании использовались средства контроля версий с ограничением доступа, подписание соглашений о неразглашении и создание модульной архитектуры, где отдельные компоненты не дают полного представления о системе.

Регуляторные риски – связаны с изменением законодательства в области обработки персональных данных и требований к автомобильной документации. Для снижения этих рисков была реализована гибкая архитектура системы, позволяющая адаптироваться к изменениям нормативно-правовой базы, и предусмотрены регулярные аудиты соответствия системы требованиям законодательства.

Риск использования нелицензионного ПО – может возникнуть при интеграции сторонних библиотек и компонентов. Для минимизации данного риска был проведен тщательный анализ лицензий всех используемых компонентов, отдано предпочтение решениям с открытым исходным кодом, совместимым с коммерческим использованием, и получены необходимые лицензии на коммерческие компоненты.

Риски точности ML-моделей – связаны с возможностью некорректной работы моделей на нестандартных или поврежденных документах. Для минимизации этих рисков было проведено обучение на расширенном наборе данных, включающем документы различного качества, разработаны механизмы выявления низкой уверенности модели и передачи сложных случаев на ручную обработку.

При предпроектном анализе все перечисленные риски были детально проанализированы, для каждого разработаны стратегии минимизации и определены триггеры, сигнализирующие о реализации рисков. В ходе проекта осуществлялся непрерывный мониторинг рисков с ежемесячным обновлением реестра рисков и корректировкой стратегий реагирования.

6 Общие принципы обеспечения безопасности на рабочем месте оператора интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

6.1 Актуальность вопросов безопасности при работе с системами обработки документов

Современные интеллектуальные системы обработки документов требуют повышенного внимания к вопросам безопасности как оборудования, так и персонала. Работа с автомобильными документами связана с длительным визуальным контролем изображений, верификацией результатов распознавания и постоянным взаимодействием с компьютерной техникой, что может оказывать комплексное воздействие на здоровье операторов системы. Интеллектуальные системы, использующие алгоритмы машинного обучения, часто требуют более производительного оборудования, что повышает риски, связанные с электромагнитным излучением, тепловыделением и энергопотреблением на рабочем месте.

На основании ГОСТ 12.0.003-2015 вредные производственные факторы при работе с системами автоматизированной обработки документов подразделяются на следующие категории:

– физические: недостаточная освещенность рабочего места, повышенная яркость света от мониторов с высоким разрешением, необходимых для верификации распознанного текста, повышенный уровень электромагнитного излучения от серверного оборудования, тепловыделение от вычислительных систем;

– психофизиологические: повышенная концентрация при верификации результатов распознавания, монотонность работы при обработке однотипных документов, зрительное перенапряжение при проверке качества распознавания мелких деталей документов;

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		95

- информационные: избыточность информации при одновременной работе с несколькими документами, необходимость быстрого переключения между задачами;
- организационные: нерациональная организация рабочего места с учётом специфики обработки графической информации.

6.2 Обеспечение информационной безопасности при работе с персональными данными

Интеллектуальная система автоматизированной обработки автомобильных документов имеет дело с конфиденциальной информацией и персональными данными граждан, что требует особого внимания к аспектам информационной безопасности:

- рабочее место оператора должно быть организовано с учётом предотвращения несанкционированного доступа к экрану монитора (использование специальных фильтров для ограничения угла обзора);
- необходимо соблюдать требования федерального закона №152-ФЗ "О персональных данных" при обработке информации из автомобильных документов;
- следует организовать рабочий процесс таким образом, чтобы исключить возможность утечки информации через бумажные копии документов (использование уничтожителей документов, хранение копий в защищенных местах);
- рабочие станции должны быть защищены от несанкционированного доступа с использованием современных средств аутентификации;
- должна быть реализована система мониторинга действий пользователя при работе с персональными данными.

Нарушение режима конфиденциальности при работе с автомобильными документами может привести к серьёзным последствиям как для субъектов персональных данных, так и для организации в целом.

6.3 Эргономика рабочего места оператора системы распознавания документов

Работа с системой автоматизированной обработки документов имеет ряд специфических особенностей, требующих особого внимания к организации рабочего места:

- мониторы высокого разрешения должны быть установлены на оптимальном расстоянии (50-70 см от глаз) и под правильным углом наклона (10-20 градусов от вертикали);
- для работы с графическими материалами необходимо обеспечить повышенный уровень освещенности рабочей поверхности (300-500 лк) при минимизации бликов на экране;
- рабочее кресло должно иметь регулировки высоты сиденья, наклона спинки и поддержки поясницы для обеспечения правильной осанки при длительной работе;
- рекомендуется использование специализированных подставок для документов, располагаемых между клавиатурой и монитором, что снижает необходимость поворота головы при сверке оригинала с результатами распознавания;
- для работы со сканированными документами рекомендуется использование графических планшетов или сенсорных экранов, снижающих нагрузку на запястья при длительной работе с мышью.

Особое внимание следует уделить профилактике зрительного утомления, которое особенно выражено при работе с системами распознавания текста:

- необходимо настроить яркость и контрастность монитора в соответствии с условиями освещения;
- рекомендуется использование программного обеспечения для автоматической регулировки цветовой температуры экрана в зависимости от времени суток;

— для снижения нагрузки на зрение рекомендуется использование мониторов с функцией защиты от синего света.

6.4 Режим труда и отдыха при работе с системой обработки документов

Работа с интеллектуальной системой автоматизированной обработки документов требует особого внимания к режиму труда и отдыха. В соответствии с СанПиН 1.2.3685-21 "Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания", работа с системами распознавания и верификации документов относится к группе высокой зрительной нагрузки и требует следующего режима:

- при 8-часовом рабочем дне необходимо делать регламентированные перерывы: 15 минут через каждый час работы;
- каждые 45 минут непрерывной работы рекомендуется выполнять комплекс упражнений для глаз, включающий переключение фокуса зрения с ближнего на дальний объект;
- рекомендуется чередовать задачи по верификации результатов распознавания с другими видами деятельности, требующими меньшей концентрации внимания;
- при работе в ночное время необходимо увеличить частоту перерывов на 20-30%.

Для компенсации психофизиологической нагрузки, связанной с высокой концентрацией внимания при верификации результатов распознавания документов, рекомендуется:

- организация зон отдыха, отделенных от рабочего пространства;
- проведение производственной гимнастики с элементами расслабления;
- использование систем автоматического напоминания о необходимости сделать перерыв.

6.5 Техника безопасности при работе с высокопроизводительным оборудованием

Интеллектуальные системы обработки и анализа документов часто требуют использования высокопроизводительного оборудования, включая серверы для обработки данных и специализированные графические станции, что создает дополнительные риски безопасности:

- серверное оборудование должно быть размещено в специально оборудованных помещениях с контролируемым микроклиматом и защитой от пыли;
- необходимо обеспечить надежное электропитание с использованием источников бесперебойного питания и защитой от перенапряжения;
- кабельные трассы должны быть проложены с учетом требований электробезопасности и пожарной безопасности;
- при работе с серверным оборудованием необходимо использовать средства защиты от статического электричества;
- необходимо проводить регулярное техническое обслуживание систем охлаждения для предотвращения перегрева оборудования.

При работе с оборудованием для сканирования документов следует соблюдать следующие меры безопасности:

- перед началом работы проверять целостность электрических кабелей и штепсельных вилок;
- не допускать попадания влаги на электрические части оборудования;
- не размещать сканирующее оборудование вблизи источников тепла;
- при возникновении неисправностей немедленно отключать оборудование от электрической сети.

Особое внимание следует уделить защите от электромагнитного излучения, создаваемого высокопроизводительным оборудованием:

- рабочие места операторов должны быть расположены на безопасном расстоянии от серверного оборудования;

- при необходимости следует использовать специальные экранирующие материалы;
- рекомендуется проводить периодический контроль уровня электромагнитного излучения на рабочих местах.

На основании вышеизложенного можно заключить, что обеспечение безопасности оператора интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов требует комплексного подхода, включающего меры по защите от физических, психофизиологических и информационных факторов риска. Соблюдение рекомендаций по организации рабочего места, режиму труда и отдыха, а также техники безопасности при работе с высокопроизводительным оборудованием позволит минимизировать негативное воздействие на здоровье операторов и обеспечить высокую эффективность работы с интеллектуальной системой.

Заключение

Результатом выполнения выпускной квалификационной работы стала интеллектуальная система автоматизированной обработки и семантического анализа автомобильных документов, обеспечивающая высокоточное распознавание и структурирование информации из различных типов документов. Система предоставляет удобные инструменты для работы с документами как через мобильные приложения, так и через веб-интерфейс.

На этапе анализа предметной области было проведено исследование существующих решений для распознавания документов, выявлены их ключевые недостатки и сформулированы требования к разрабатываемой системе. Особое внимание уделялось анализу специфики автомобильных документов – паспортов транспортных средств, свидетельств о регистрации и водительских удостоверений, имеющих сложную структуру и разнообразные форматы представления данных.

В ходе проектирования была разработана микросервисная архитектура системы, включающая API-сервис на ASP.NET Core, файловый сервис на Golang, сервис машинного обучения на Python и кроссплатформенное клиентское приложение на Kotlin Multiplatform. Данный подход обеспечил гибкость разработки, масштабируемость системы и возможность независимого развития отдельных компонентов.

Центральным элементом системы стал сервис машинного обучения, реализующий каскад нейросетевых моделей для решения задач классификации, сегментации и распознавания текста в документах. Применение архитектуры YOLO для сегментации и EasyOCR для распознавания текста обеспечило высокую точность и производительность системы.

Практическая значимость разработанной системы заключается в возможности автоматизации процессов работы с автомобильными документами в различных сферах: от государственных органов регистрации транспортных средств до страховых компаний и автосалонов.

Список использованных источников

1. Кириченко, А.А. Оптическое распознавание символов в системах обработки документов / А.А. Кириченко, И.И. Дроздова // Вестник ДГТУ. – 2019. – Т. 19, № 4. – С. 120-135.
2. Андреева, Е.Ю. Информационная система автоматизированного распознавания реквизитов экономических документов на основе OCR-алгоритмов / Е.Ю. Андреева, П.С. Козлов // Научно-технический вестник информационных технологий. – 2020. – № 5. – С. 34-41.
3. Ломакин, Н.И. Алгоритмы OCR: современные подходы к распознаванию текста / Н.И. Ломакин, И.А. Самородова // Фундаментальные исследования. – 2021. – № 7. – С. 54-59.
4. Смирнов, М.К. Модели глубокого обучения для сегментации элементов документа / М.К. Смирнов, А.В. Петров // Компьютерная оптика. – 2022. – Т. 46, № 3. – С. 485-499.
5. EasyOCR: эффективное распознавание текста на изображениях. [Электронный ресурс] / Режим доступа: <https://github.com/JaidedAI/EasyOCR>, свободный. (дата обращения 15.04.2024).
6. YOLOv8: новое поколение алгоритмов детекции объектов. [Электронный ресурс] / Режим доступа: <https://docs.ultralytics.com>, свободный. (дата обращения 20.04.2024).
7. Ронжин, А.Л. Система автоматического распознавания информации с изображений документов / А.Л. Ронжин, А.А. Карпов, И.В. Ли // Информационно-управляющие системы. – 2019. – № 2. – С. 69-76.
8. Вендров, А.М. Проектирование программного обеспечения экономических информационных систем / А.М. Вендров. – М.: Финансы и статистика, 2020. – 352 с.
9. Федотова, Д.Э. CASE-технологии: Практикум / Д.Э. Федотова, Ю.Д. Семенов, К.Н. Чижик. – М.: Горячая линия-Телеком, 2018. – 160 с.

10. Шпак, Ю.А. Распознавание изображений и обработка видео на C#.NET / Ю.А. Шпак. – СПб.: БХВ-Петербург, 2021. – 512 с.
11. Гудфеллоу, Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль; пер. с англ. А.А. Слинкина. – М.: ДМК Пресс, 2018. – 652 с.
12. Михайлов, И.С. Методы сегментации и распознавания текстовых блоков на изображениях документов / И.С. Михайлов, В.П. Кузнецов // Известия вузов. Приборостроение. – 2019. – Т. 62, № 3. – С. 217-225.
13. PostgreSQL: The World's Most Advanced Open Source Relational Database. [Электронный ресурс] / Режим доступа: <https://www.postgresql.org/about>, свободный. (дата обращения 01.05.2024).
14. Kotlin Multiplatform: разработка кроссплатформенных приложений. [Электронный ресурс] / Режим доступа: <https://kotlinlang.org/docs/multiplatform.html>, свободный. (дата обращения 10.05.2024).
15. Микросервисная архитектура на практике. [Электронный ресурс] / Режим доступа: <https://habr.com/ru/articles/597277>, свободный. (дата обращения 05.05.2024).

Приложение А

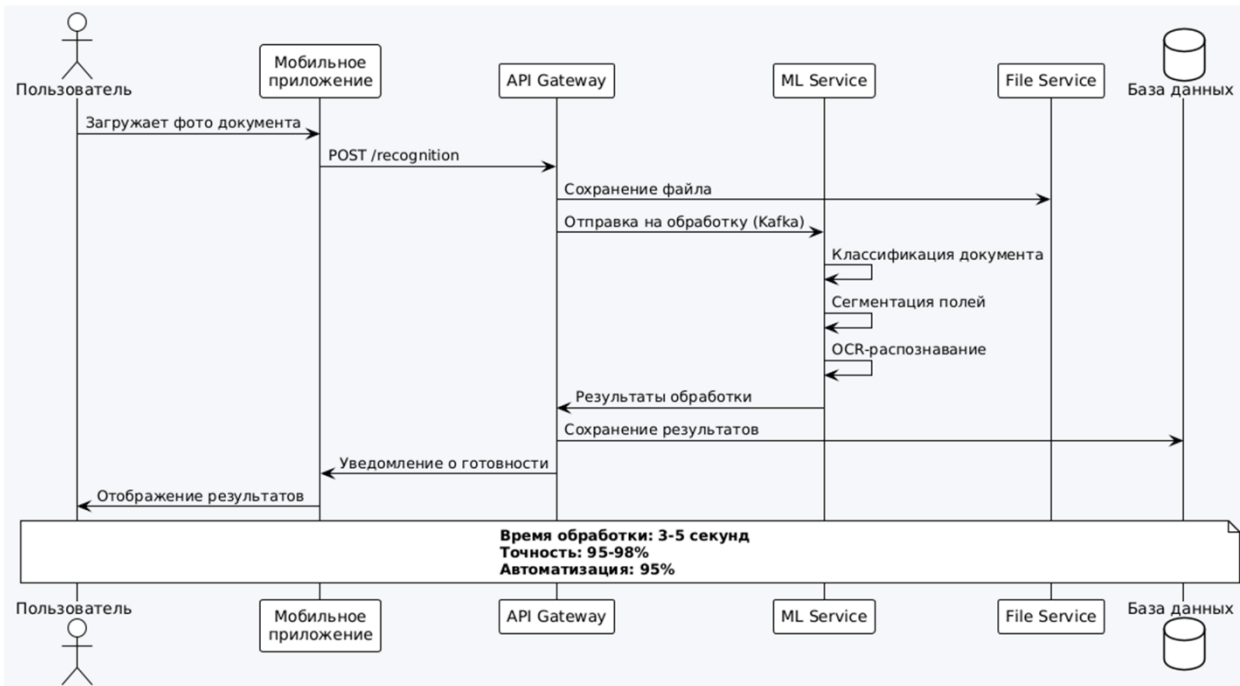
(Обязательное)

Интеллектуальная система автоматизированной обработки и семантического
анализа автомобильных документов

UML-диаграммы

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		104

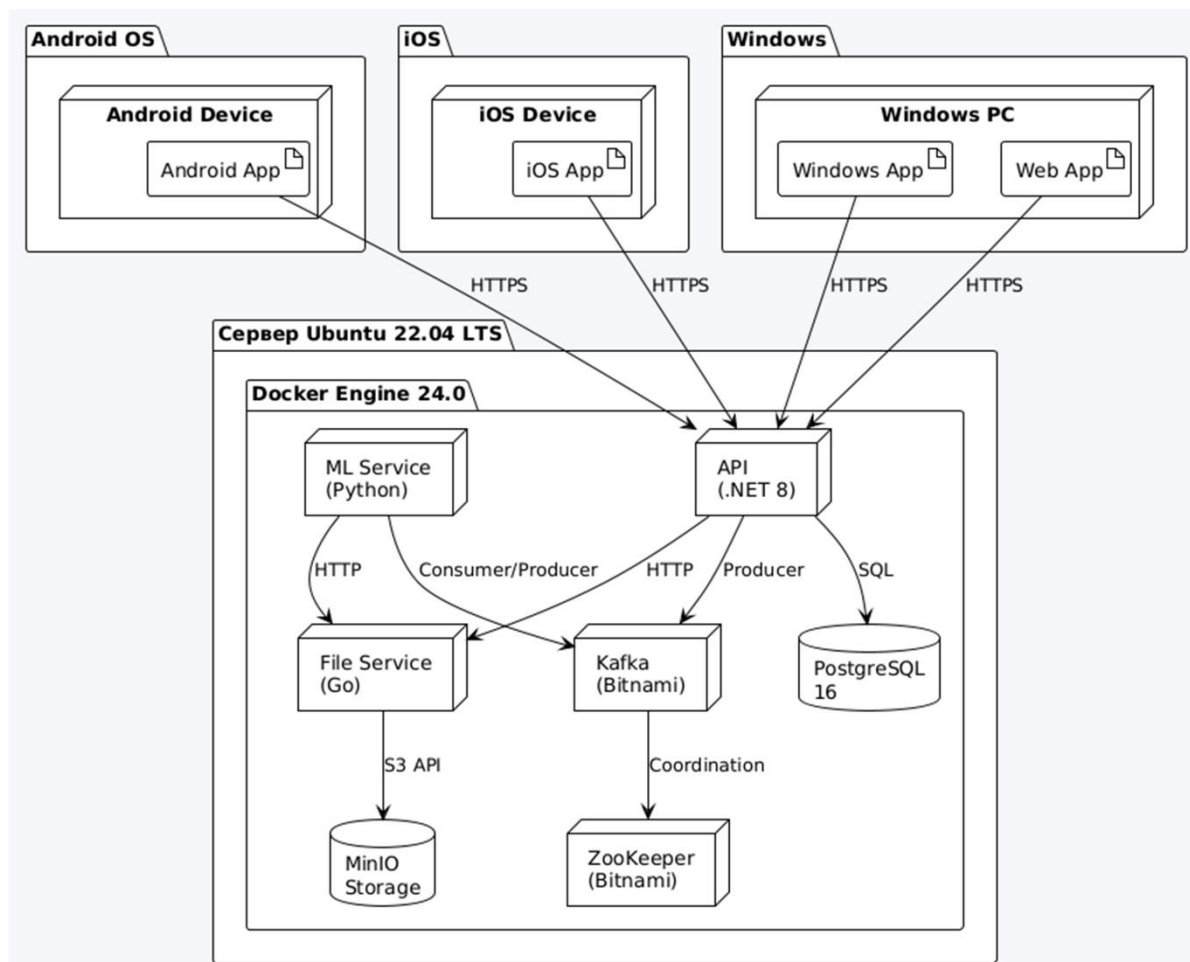
Приложение А.1-Диаграмма последовательностей



ПГУ 1.090501.14.002 Д14					
Изм.	Лист	№ докум.	Подпись	Дата	Исполнительная система
Разработчик	Сотынский Е.В.				информационной системы и
Проверка	Михайлов В.А.				электронного обмена информацией
					документации
И. Исполн.	Биликин А.С.				
Утвердил					

ФВТ, каф. ВТ
гр. 20ВЮ31

Приложение А.2-Диаграмма разворачивания

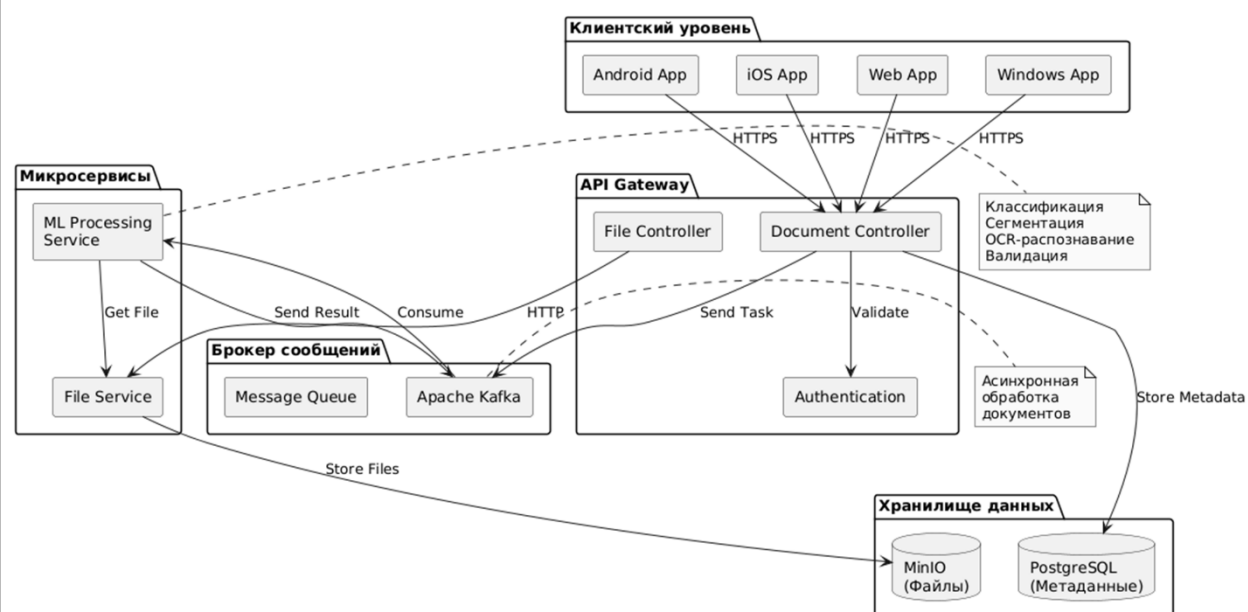


					ПГУ 1.090501.14.003 Д19		
Изм.	Лист	№ докум.	Год/мес.	Дата	Инициализация системы		
Разработ.	Сметанкин Е.В.				интеграционная разработка и		
Провер.	Михайлов В.А.				оптимизация системы интеграционных		
					взаимодействий		
И. Исполн.	Биликин А.С.				ФВТ, каф. ВТ		
Утвержд.					пр. 20.03.21		

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.14.001 ПЗ

Приложение А.3-Диаграмма компонентов



ПГУ 1.090501.14.004 Д19									
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система автоматизированной обработки и системного анализа информации				
Разработчик	Сотский Е.В.								
Проверка	Михайлов В.А.								
И. Исполн.	Биликин А.С.								
Утвердил									
					ФВТ, каф. ВТ ар. 20ВЮ31				

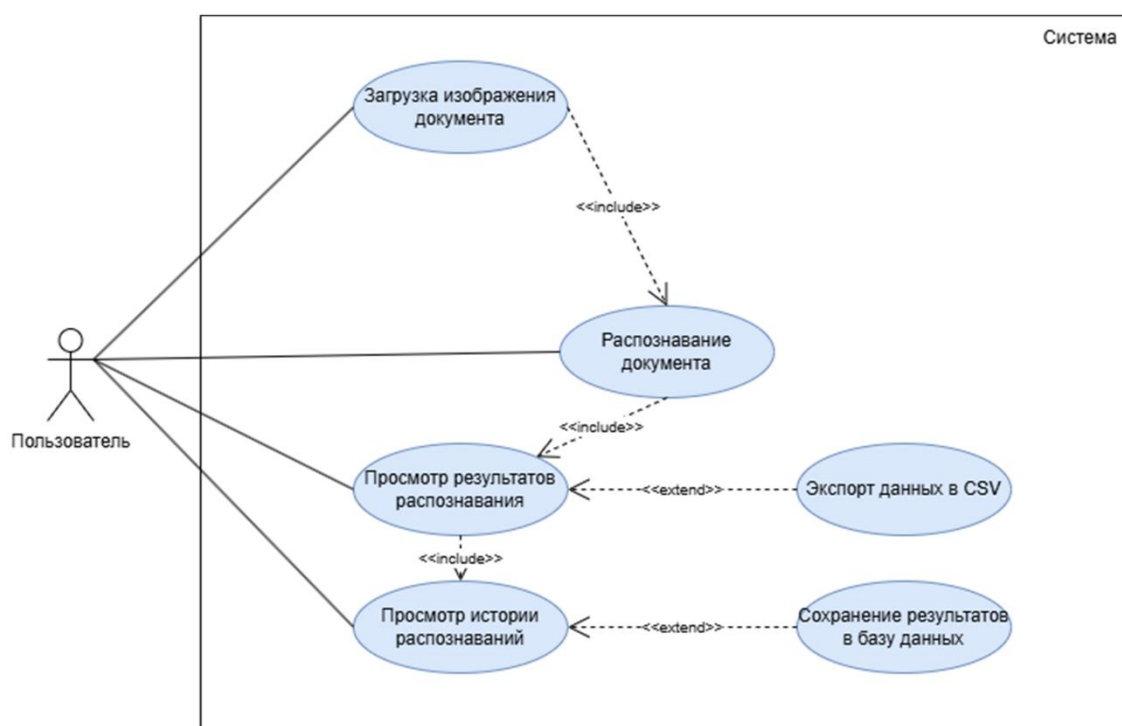
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.14.001 ПЗ

Лист

107

Приложение А.4-Диаграмма вариантов использования



					ПГУ 1.090501.14.005 Д13		
Изм.	Лист	№ докум.	Год	Дата	Информационная система		
Разработчик	Специалист Е.В.				интеллектуальной обработки и		
Проверка	Мухомов В.А.				аналитического отчета интеллектуальной		
					версионности		
И.И.И.	Иванов А.С.				ФВТ, каф. ВТ		
Утвердил					ар. 20ВЮ31		

Изм.	Лист	№ докум.	Подпись	Дата

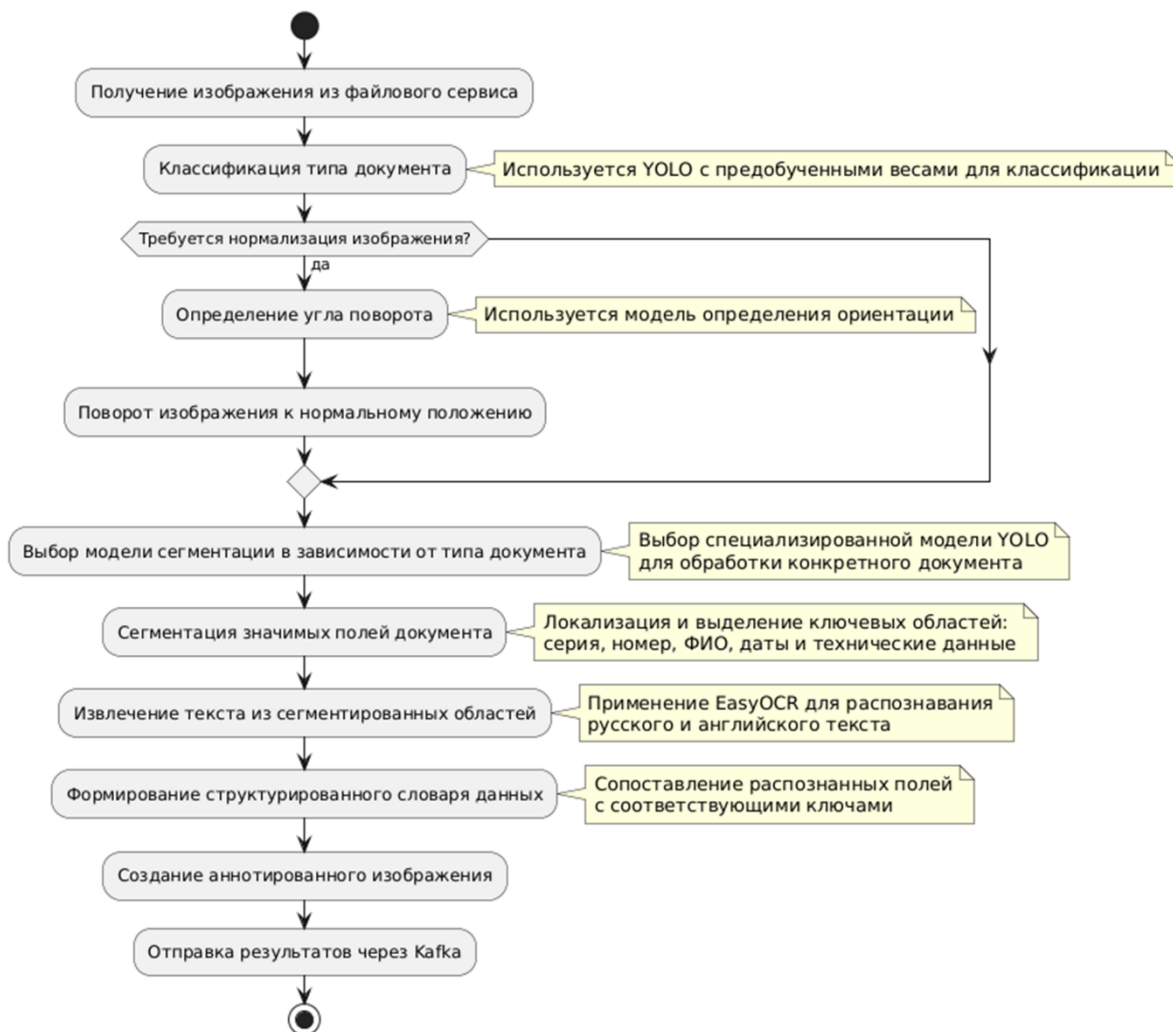
ПГУ 1.090501.14.001 ПЗ

Лист

108

Приложение А.5-Диаграмма деятельности

Диаграмма деятельности сервиса распознавания документов



					ПГУ 1.090501.14.006 Д17		
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система автоматизированной обработки и систематического анализа автомобильных документов		
Разработчик	Савельев Е.В.				Лист	Лист	Лист
Проверка	Михайлов В.А.						
И. Исполн.	Белкин А.С.				ФВТ, каф. ВТ ар. 20ВЮ31		
Утвердил							

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.14.001 ПЗ

Лист

109

Приложение Б

(Обязательное)

Интеллектуальная система автоматизированной обработки и семантического
анализа автомобильных документов

Листинг программы

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		110

Файл DocumentRecognitionController.cs

```
[ApiController]
[Route("[controller]")]
public class DocumentRecognitionController : ControllerBase
{
    private readonly IFileService fileService;
    private readonly IKafkaProducesService kafkaProducesService;
    private readonly IConnection connection;
    public DocumentRecognitionController(IFileService fileService,
    IKafkaProducesService kafkaProducesService, IConnection connection)
    {
        this.fileService = fileService;
        this.kafkaProducesService = kafkaProducesService;
        this.connection = connection;
    }
    [HttpPost]
    public async Task<IActionResult> RecognitionDocument([FromBody]
    ImageData imageData)
    {
        if (imageData == null || string.IsNullOrEmpty(imageData.Image))
        {
            return BadRequest("Image data is missing.");
        }
        var imageBytes = Convert.FromBase64String(imageData.Image);
        var fileId = await fileService.UploadFileAsync(imageBytes);
        var guid = Guid.NewGuid().ToString();
        var kafkaMessage = new RequestMessage { Guid = guid, FileId =
fileId };
        await AddRequestToDb(imageData.UserId, guid, fileId);
        await kafkaProducesService.WriteTraceLogAsync(kafkaMessage);
        return Ok();
    }
    [HttpGet("{userId}")]
    public async Task<IActionResult> GetUserDocuments(string userId)
    {
        var queryObject = new QueryObject(
            "SELECT id as Id, user_id as UserId, guid as Guid, file_id as
FileId FROM requests WHERE user_id = @user_id",
            new { user_id = userId });
        var requests = await
connection.ListOrEmpty<DbRequest>(queryObject);
        var listResults = new List<DbResult>();
        if (requests.Count != 0)
        {
            var list = requests.Select(x => x.Guid).ToList();
            var result = string.Join(", ", list.Select(e => $"'{e}'"));
            var queryObjectResult = new QueryObject(
                $"SELECT id as Id, guid as Guid, file_id as FileId, type
as Type, series as Series, number as Number, page_number as PageNumber,
confidence as Confidence, data as Data FROM resolutions WHERE guid in
({result})");
            var t = queryObjectResult.ToJson();
            Console.WriteLine(t);
            listResults.AddRange(await
connection.ListOrEmpty<DbResult>(queryObjectResult));
        }
        var resultList = new List<Result>();
```

									Лист
									111
Изм.	Лист	№ докум.	Подпись	Дата					

ПГУ 1.090501.14.001 ПЗ

```

        foreach (var request in requests)
        {
            var result = listResults.FirstOrDefault(x => x.Guid ==
request.Guid);
            if (result != null)
            {
                resultList.Add(new Result()
                {
                    Status = 1,
                    UserId = request.UserId,
                    Guid = result.Guid,
                    Type = result.Type,
                    Series = result.Series,
                    Number = result.Number,
                    PageNumber = result.PageNumber,
                    Confidence = result.Confidence,
                    FileId = result.FileId,
                    Data
=
JsonConvert.DeserializeObject<Dictionary<string, string>>(result.Data)
                });
            }
            else
            {
                resultList.Add(new Result() { UserId = request.UserId,
Guid = request.Guid, Status = 0 });
            }
        }
        return Ok(resultList);
    }
    [HttpGet("download/{guid}")]
    public async Task<IActionResult> GetDocument(string guid)
    {
        var queryObject = new QueryObject(
            "SELECT id as Id, guid as Guid, file_id as FileId, type as
Type, series as Series, number as Number, page_number as PageNumber,
confidence as Confidence, data as Data FROM resolutions WHERE guid =
@guid",
            new { guid = guid });
        var result = await
connection.FirstOrDefault<DbResult>(queryObject);
        if (result == null)
        {
            return NotFound();
        }
        var queryObject2 = new QueryObject(
            "SELECT id as Id, user_id as UserId, guid as Guid, file_id as
FileId FROM requests WHERE guid = @guid",
            new { guid = guid });
        var request = await
connection.FirstOrDefault<DbRequest>(queryObject2);
        if (request == null)
        {
            return NotFound();
        }
        var result1 = new Result()
        {
            Status = 1,

```

```

        UserId = request.UserId,
        Guid = result.Guid,
        Type = result.Type,
        Series = result.Series,
        Number = result.Number,
        PageNumber = result.PageNumber,
        Confidence = result.Confidence,
        FileId = result.FileId,
        Data = JsonConvert.DeserializeObject<Dictionary<string,
string>>(result.Data)
    };
    using (var content = await ExportAsync(new List<Result> { result1
}))
    {
        Response.ContentType = "application/octet-stream";

        Response.Headers.Add("Content-Disposition", "attachment;
filename=\"data.csv\"");

        await content.CopyToAsync(Response.Body);
    }
    return new EmptyResult();
}
private async Task<MemoryStream> ExportAsync(IEnumerable records)
{
    byte[] content = null;
    using (var mem = new MemoryStream())
    using (var writer = new StreamWriter(mem,
Encoding.GetEncoding("utf-8")))
    {
        using (var csvWriter = new CsvWriter(writer,
CultureInfo.InvariantCulture))
        {
            csvWriter.NextRecord();
            await
csvWriter.WriteRecordsAsync(records).ConfigureAwait(false);
            await csvWriter.FlushAsync().ConfigureAwait(false);
            await writer.FlushAsync().ConfigureAwait(false);
            content = mem.ToArray();
        }
    }
    var output = new MemoryStream(content);
    return output;
}
private async Task AddRequestToDb(string imageDataUserId, string guid,
string fileId)
{
    var queryObject = new QueryObject(
        "INSERT INTO requests (user_id, guid, file_id) VALUES
(@user_id, @guid, @file_id) RETURNING id",
        new { user_id = imageDataUserId, guid = guid, file_id = fileId
});
    await connection.CommandWithResponse<int>(queryObject);
}
}

```

Файл FileController.cs

```
[ApiController]
[Route("[controller]")]
public class FileController : ControllerBase
{
    private readonly IFileService fileService;
    public FileController(IFileService fileService)
    {
        this.fileService = fileService;
    }
    [HttpGet("{fileId}")]
    public async Task<IActionResult> GetFile(string fileId)
    {
        var file = await fileService.DownloadFileAsync(fileId);
        if (file == null)
        {
            return NotFound();
        }
        return Ok(JsonConvert.SerializeObject(file));
    }
}
```

Файл service.go

```
package file
import (
    "context"
    "github.com/theartofdevel/notes_system/file_service/pkg/logging"
)
var _ Service = &service{}
type service struct {
    storage Storage
    logger logging.Logger
}
func NewService(noteStorage Storage, logger logging.Logger) (Service, error) {
    return &service{
        storage: noteStorage,
        logger: logger,
    }, nil
}
type Service interface {
    GetFile(ctx context.Context, fileName string) (f *File, err error)
    GetFilesByNoteUUID(ctx context.Context) ([]*File, error)
    Create(ctx context.Context, dto []byte) (string, error)
    Delete(ctx context.Context, fileName string) error
    Upload(ctx context.Context, fileContent []byte) (string, error)
    GetByID(ctx context.Context, id string) ([]byte, error)
}
func (s *service) GetFile(ctx context.Context, fileId string) (f *File, err error) {
    f, err = s.storage.GetFile(ctx, fileId)
    if err != nil {
        return f, err
    }
    return f, nil
}
func (s *service) GetFilesByNoteUUID(ctx context.Context) ([]*File, error) {
    {
```

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		114

```

        files, err := s.storage.GetFilesByNoteUUID(ctx)
        if err != nil {
            return nil, err
        }
        return files, nil
    }

func (s *service) Create(ctx context.Context, dto []byte) (string, error) {
    fileName, err := NormalizeName(dto)
    if err != nil {
        return "", err
    }
    key, err := s.storage.CreateFile(ctx, fileName.String(), dto)
    if err != nil {
        return "", err
    }
    return key, nil
}

func (s *service) Delete(ctx context.Context, fileName string) error {
    err := s.storage.DeleteFile(ctx, fileName)
    if err != nil {
        return err
    }
    return nil
}

func (s *service) Upload(ctx context.Context, fileContent []byte) (string, error) {
    // Implementation of Upload method
    return "", nil
}

func (s *service) GetByID(ctx context.Context, id string) ([]byte, error) {
    // Implementation of GetByID method
    return nil, nil
}

```

Файл handler.go

```

package file
import (
    "bytes"
    "encoding/base64"
    "encoding/json"
    "fmt"
    "github.com/julienschmidt/httprouter"
    "github.com/theartofdevel/notes_system/file_service/internal/apperr
or"
    "github.com/theartofdevel/notes_system/file_service/pkg/logging"
    "io/ioutil"
    "net/http"
)
const (
    filesURL = "/api/files"
    fileURL  = "/api/files/:id"
)
type Handler struct {
    Logger      logging.Logger
    FileService Service

```

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		115

```

}
type ImageData struct {
    Image string `json:"image"`
}
func (h *Handler) Register(router *httprouter.Router) {
    router.HandlerFunc(http.MethodGet, fileURL,
    apperror.Middleware(h.GetFile))
    router.HandlerFunc(http.MethodGet, filesURL,
    apperror.Middleware(h.GetFilesByNoteUUID))
    router.HandlerFunc(http.MethodPost, filesURL,
    apperror.Middleware(h.CreateFile))
    router.HandlerFunc(http.MethodDelete, fileURL,
    apperror.Middleware(h.DeleteFile))
}
func (h *Handler) GetFile(w http.ResponseWriter, r *http.Request) error {
    h.Logger.Info("GET FILE")
    h.Logger.Debug("get note_uuid from URL")
    h.Logger.Debug("get fileId from context")
    params :=
r.Context().Value(httprouter.ParamsKey).(httprouter.Params)
    fileId := params.ByName("id")
    f, err := h.FileService.GetFile(r.Context(), fileId)
    if err != nil {
        return err
    }
    w.Header().Set("Content-Disposition", fmt.Sprintf("attachment;
filename=%s", f.Name))
    w.Header().Set("Content-Type", r.Header.Get("Content-Type"))
    base64String := base64.StdEncoding.EncodeToString(f.Bytes)
    decodeString, err := base64.StdEncoding.DecodeString(base64String)
    if err != nil {
        return err
    }
    w.Write(decodeString)
    return nil
}
func (h *Handler) GetFilesByNoteUUID(w http.ResponseWriter, r
*http.Request) error {
    h.Logger.Info("GET FILES BY NOTE UUID")
    w.Header().Set("Content-Type", "form/json")
    h.Logger.Debug("get note_uuid from URL")
    file, err := h.FileService.GetFilesByNoteUUID(r.Context())
    if err != nil {
        return err
    }
    filesBytes, err := json.Marshal(file)
    if err != nil {
        return err
    }
    w.WriteHeader(http.StatusOK)
    w.Write(filesBytes)
    return nil
}
func (h *Handler) CreateFile(w http.ResponseWriter, r *http.Request) error
{
    h.Logger.Info("CREATE FILE")
    w.Header().Set("Content-Type", "form/json")

```

```

    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        http.Error(w, "Failed to read request body",
http.StatusInternalServerError)
        return nil
    }
    var imageData ImageData
    if err := json.Unmarshal(body, &imageData); err != nil {
        http.Error(w, "Failed to decode JSON", http.StatusBadRequest)
        return nil
    }
    if imageData.Image == "" {
        http.Error(w, "Image data is missing", http.StatusBadRequest)
        return nil
    }
    decodedBytes, err :=
base64.StdEncoding.DecodeString(imageData.Image)
    if err != nil {
        return err
    }
    key, err := h.FileService.Create(r.Context(), decodedBytes)
    if err != nil {
        return err
    }
    keyBytes, err := json.Marshal(key)
    if err != nil {
        return err
    }
    w.Write(bytes.Trim(keyBytes, "\""))
    w.WriteHeader(http.StatusCreated)

    return nil
}
func (h *Handler) DeleteFile(w http.ResponseWriter, r *http.Request) error
{
    h.Logger.Info("DELETE FILE")
    w.Header().Set("Content-Type", "application/json")
    h.Logger.Debug("get fileId from context")
    params :=
r.Context().Value(httprouter.ParamsKey).(httprouter.Params)
    fileId := params.ByName("id")
    err := h.FileService.Delete(r.Context(), fileId)
    if err != nil {
        return err
    }
    w.WriteHeader(http.StatusNoContent)
    return nil
}

```

Файл main.py

```

from config.app_config import AppConfig
from config.kafka_config import KafkaConfig
from kafka.consumer import Consumer
from kafka.producer import Producer
def main():
    config = AppConfig()
    kafka_config = config.get_kafka_config()

```

```

kafka_consumer = Consumer(KafkaConfig(kafka_config))
kafka_producer = Producer(KafkaConfig(kafka_config))

kafka_consumer.start_consume()
if __name__ == '__main__':
    main()

```

Файл models_activities.py

```

from ultralytics import YOLO
from PIL import Image
import time
import easyocr
import numpy as np
import cv2
import os
from dict_documents import *
def classify_document(image):
    model = YOLO('models/classify/best_classifier.pt') # load a custom model
    # Predict with the model
    results = model.predict(image, conf=0.8) # predict on an image
    probs = results[0].probs
    best_prob = "%f" % probs.top1conf.numpy()
    names = model.names

    return best_prob, names[probs.top1]
def from_eng_name_to_rus(IMG_PATH):
    probs, pred_class = classify_document(IMG_PATH)
    ru_class = ""
    if(pred_class == "passport_address"):
        ru_class = "страница с пропиской"
    elif(pred_class == "passport_first_page"):
        ru_class = "страница персональных данных"
    elif(pred_class == "sts_ver1" or pred_class == "sts_ver2"):
        ru_class = "СТС"
    elif(pred_class == "vu_ver1" or pred_class == "vu_ver2"):
        ru_class = "водительские права"
    elif(pred_class == "pts"):
        ru_class = "ПТС"

    return probs, ru_class
def convert_predict_class_en(pred_class):
    type_class = ""
    page_number = 0
    if (pred_class == "passport_address"):
        type_class = "personal_passport"
        page_number = 2
    elif (pred_class == "passport_first_page"):
        type_class = "personal_passport"
        page_number = 1
    elif (pred_class == "sts_ver1"):
        type_class = "vehicle_certificate"
        page_number = 1
    elif (pred_class == "sts_ver2"):
        type_class = "vehicle_certificate"
        page_number = 2
    elif (pred_class == "vu_ver1"):
        type_class = "driver_license"

```

```

        page_number = 1
    elif (pred_class == "vu_ver2"):
        type_class = "driver_license"
        page_number = 2
    elif (pred_class == "pts"):
        type_class = "vehicle_passport"
    return type_class, page_number
def convert_predict_class_ru(pred_class):
    ru_class=""
    if(pred_class == "passport_address"):
        ru_class = "страница с пропиской"
    elif(pred_class == "passport_first_page"):
        ru_class = "страница персональных данных"
    elif(pred_class == "sts_ver1" or pred_class == "sts_ver2"):
        ru_class = "СТС"
    elif(pred_class == "vu_ver1" or pred_class == "vu_ver2"):
        ru_class = "водительские права"
    elif(pred_class == "pts"):
        ru_class = "ПТС"

    return ru_class
def get_degree_rotation(IMG_PATH):
    model = YOLO('models/classify/best_rotate.pt') # load a custom model
    results = model(IMG_PATH, conf=0.8) # predict on an image
    probs = results[0].probs
    names = model.names
    return names[probs.top1]
def rotate_image_to_normal(IMG_PATH):
    degree = int(get_degree_rotation(IMG_PATH))
    if(degree != 0):
        if(degree== 180):
            rotated_image1 = IMG_PATH.transpose(Image.ROTATE_180)
        elif(degree == 90):
            rotated_image1 = IMG_PATH.rotate(-90, expand=True)
        elif(degree == 270):
            rotated_image1 = IMG_PATH.transpose(Image.ROTATE_90)
        return rotated_image1
    return IMG_PATH

def get_images_text(model, image, predict_class):
    image = rotate_image_to_normal(image)
    results = model.predict(image)
    result = results[0]
    boxes = result.boxes
    names = result.names
    dict = {}
    reader = get_reader()
    for i in range(len(boxes.cls.tolist())):
        crop_cls = names[boxes.cls.tolist()[i]]
        need_rotate = crop_cls == 'pass_series_and_number'
        image_cropped = image.crop(boxes[i].xyxy[0].tolist())
        text_results = get_data_from_docs(reader, image_cropped,
need_rotate)
        print(text_results)
        total_text = ""
        for (_, text, _) in text_results:
            total_text += text + " "

```

```

total_text = total_text.strip()
if total_text != '':
    if crop_cls not in dict or need_rotate:
        dict[crop_cls] = total_text
    else:
        dict[crop_cls] += " " + total_text
document_classes, series_keys, numbers_keys =
get_classes_keys_dict(predict_class)
ready_dict = {}
series = ""
number = ""
for key, value in document_classes.items():
    if key not in dict:
        ready_dict[value] = 'Отсутствует'
    else:
        ready_dict[value] = dict[key]

if key in dict and key in series_keys and series == "":
    series = ready_dict[value]

if key in dict and key in numbers_keys and number == "":
    number = ready_dict[value]

if key == 'pass_series_and_number':
    series = ready_dict[value][:5].strip()
    number = ready_dict[value][5:].strip()
return result, ready_dict, series, number
def get_plotted_image(inference_result):
    res_plotted = inference_result.plot()
    image = Image.fromarray(res_plotted)
    return image
def get_classes_keys_dict(pred_class):
    dict = {}
    series_keys = []
    number_keys = []
    if (pred_class == "passport_address" or pred_class ==
"passport_first_page" or pred_class == "personal_passport"):
        dict = passport_keys
        series_keys = ["pass_series_and_number"]
        number_keys = ["pass_series_and_number"]
    elif (pred_class == "sts_ver1" or pred_class == "sts_ver2" or
pred_class == "vehicle_certificate"):
        dict = sts_keys
        series_keys = ["series"]
        number_keys = ["number"]
    elif (pred_class == "vu_ver1" or pred_class == "vu_ver2" or pred_class
== "driver_license"):
        dict = driver_license_keys
        series_keys = ['series_front', 'car_series']
        number_keys = ['number_front', 'car_number']
    elif (pred_class == "pts" or pred_class == "vehicle_passport"):
        dict = pts_keys
        series_keys = ['series']
        number_keys = ['number']
    return dict, series_keys, number_keys
def load_normal_image_to_models(doc_type):
    pred_model = YOLO('models/segmentation/best_passport.pt')

```

```

        if(doc_type == "страница с пропиской" or doc_type == "страница
персональных данных"):
            pred_model = YOLO('models/segmentation/best_passport.pt') # load
a custom model
        elif(doc_type == "водительские права"):
            pred_model = YOLO('models/segmentation/best_driver_license.pt')
# load a custom model
        elif (doc_type == "ИТС"):
            pred_model = YOLO('models/segmentation/best_pts.pt') # load a
custom model
        elif (doc_type == "СТС"):
            pred_model = YOLO('models/segmentation/best_sts.pt') # load a
custom model
        return pred_model
def get_model(cls_name):
    pred_model = YOLO('models/segmentation/best_passport.pt')
    if (cls_name == "personal_passport"):
        pred_model = YOLO('models/segmentation/best_passport.pt') # load
a custom model
    elif (cls_name == "driver_license"):
        pred_model = YOLO('models/segmentation/best_driver_license.pt')
# load a custom model
    elif (cls_name == "vehicle_passport"):
        pred_model = YOLO('models/segmentation/best_pts.pt') # load a
custom model
    elif (cls_name == "vehicle_certificate"):
        pred_model = YOLO('models/segmentation/best_sts.pt') # load a
custom model
    return pred_model

def get_text_from_img(IMG_PATH):
    _,doc_type = from_eng_name_to_rus(IMG_PATH)
    model = load_normal_image_to_models(doc_type)
    results = model(IMG_PATH,device="0")
    classes = []
    coordinates = []
    detect_class = []
    im = Image.open(IMG_PATH)
    names = model.names

    for r in results:
        for c in r.bboxes.cls:
            cords = r.bboxes[0].xyxy[0].tolist()
            classes.append(names[int(c)])
            coordinates.append(cords)

        #r.show()
    ordered_classes, ordered_coordinates =
order_arrays(list(names.values()), classes, coordinates)
    for i,cls in enumerate(ordered_classes):
        if(cls in "pass_series_and_number"):
            #count = count + 1
            im1 = im.crop(ordered_coordinates[i])
            ser_num = get_series_and_number_pass(im1)
            ser_num = " ".join(ser_num)
            ordered_classes.remove(i)
            detect_class.append(ser_num)
            im1 = im.crop(ordered_coordinates[i])

```

```

        detect_class.append(get_data_from_docs(im1))

print("Предсказанные классы: ", ordered_classes)
print("Предсказанные координаты: ", ordered_coordinates)
dictionary = dict({})

# dictionary = write_to_dict(detect_class, IMG_PATH)
#im1.save(f"im_{count}.jpg")

# print("Заполненные данные: ", dictionary)

#return classes,coordinates

#упорядочиваем значения предсказанного массива классов,относительно
изначальных
def order_arrays(ordered_fields, unordered_values, coordinates):
    # Создаем словари для хранения упорядоченных значений и координат
    ordered_values = []
    ordered_coordinates = []

    # Множество для отслеживания уникальных значений
    seen_values = set()

    # Проходим по каждому полю из упорядоченных полей
    for field in ordered_fields:
        # Находим индекс поля в неупорядоченном списке значений
        if field in unordered_values:
            indexes = [i for i, f in enumerate(unordered_values) if f ==
field]
            for index in indexes:
                value = unordered_values[index]
                if value not in seen_values: # Проверяем наличие значения
в множестве
                    # Добавляем значение в список упорядоченных значений
ordered_values.append(value)
                    # Добавляем соответствующие координаты из списка
координат
ordered_coordinates.append(coordinates[index])
                    # Добавляем значение в множество, чтобы избежать
дубликатов
seen_values.add(value)

    return ordered_values, ordered_coordinates

def order_classes(ordered_fields, unordered_values):
    ordered_values = []
    seen_values = set()

    for field in ordered_fields:
        # Находим индекс поля в неупорядоченном списке значений
        if field in unordered_values:
            indexes = [i for i, f in enumerate(unordered_values) if f ==
field]
            for index in indexes:
                value = unordered_values[index]
                if value not in seen_values: # Проверяем наличие значения
в множестве

```

```

        # Добавляем значение в список упорядоченных значений
        ordered_values.append(value)
        # Добавляем значение в множество, чтобы избежать
дубликатов
        seen_values.add(value)

    return ordered_values

def write_to_dict(classes, pred_class):
    doc_type = convert_predict_class_ru(pred_class)
    if (doc_type == "страница с пропиской" or doc_type == "страница
персональных данных"):
        for key, value in zip(passport.keys(), classes):
            passport[key] = value

        # Добавляем "Отсутствует" для оставшихся ключей в словаре
        for key in passport:
            if passport[key] == " ":
                passport[key] = "Отсутствует"

        return passport

    elif (doc_type == "водительские права"):
        for key, value in zip(driver_license.keys(), classes):
            driver_license[key] = value

        # Добавляем "Отсутствует" для оставшихся ключей в словаре
        for key in driver_license:
            if driver_license[key] == " ":
                driver_license[key] = "Отсутствует"

        return driver_license

    elif (doc_type == "ПТС"):
        for key, value in zip(pts.keys(), classes):
            pts[key] = value

        # Добавляем "Отсутствует" для оставшихся ключей в словаре
        for key in pts:
            if pts[key] == " ":
                pts[key] = "Отсутствует"

        return pts

    elif (doc_type == "СТС"):
        for key, value in zip(sts.keys(), classes):
            sts[key] = value

        # Добавляем "Отсутствует" для оставшихся ключей в словаре
        for key in sts:
            if sts[key] == " ":
                sts[key] = "Отсутствует"

        return sts

def get_series_and_number_pass(image):
    reader = easyocr.Reader(['ru', 'en']) # this needs to run only once
to load the model into memory
    Original_Image = Image.open(image)

```

```

rotated_image1 = Original_Image.transpose(Image.ROTATE_90)
result = reader.readtext(np.asarray(rotated_image1), detail=0)
print(result)
return result

def get_reader():
    reader = easyocr.Reader(['ru', 'en'],
model_storage_directory='EasyOCR/model', download_enabled=False)
    return reader

def get_data_from_docs(reader, image, need_rotate):
    # rotated_image1 = rotate_image_to_normal(image)
    rotated_image1 = image
    if need_rotate:
        print("ROTATING")
        rotated_image1 = image.rotate(90, expand=True)
    result = reader.readtext(np.asarray(rotated_image1))
    return result

def main():
    IMG_PATH =
    "../dataset/passport_3_jpg.rf.3eaddb37afa7548926fe7e23fbf2dfac.jpg"
    probs, get_class = from_eng_name_to_rus(IMG_PATH) #возвращает класс
    предсказанного доку и вероятность
    get_text_from_img(IMG_PATH)

if __name__ == '__main__':
    start_time = time.time()
    main()
    print("--- %s seconds ---" % (time.time() - start_time))

```

Файл App.kt

```

import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.application
import androidx.compose.ui.window.rememberWindowState
import java.awt.Dimension
import com.niksah.gagarin.App
import com.niksah.gagarin.di.dataModule
import com.niksah.gagarin.di.domainModule
import com.niksah.gagarin.di.uiModule
import org.koin.core.context.startKoin

fun main() = application {
    startKoin {
        modules(listOf(dataModule, domainModule, uiModule))
    }
    Window(
        title = "GagarinHak",
        state = rememberWindowState(width = 1200.dp, height = 1000.dp),
        onCloseRequest = ::exitApplication,
    ) {
        window.minimumSize = Dimension(350, 600)
        App()
    }
}

```

Файл ViewModel.kt

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		124

```

package com.niksah.gagarin.screens.main

import com.darkrockstudios.libraries.mpfilepicker.MPFile
import com.niksah.gagarin.data.models.fold
import com.niksah.gagarin.data.repositories.ApiRepository
import com.niksah.gagarin.data.repositories.ResponseRepository
import com.niksah.gagarin.data.repositories.SettingsRepository
import com.niksah.gagarin.utils.base.BaseViewModel
import com.niksah.seconHack.data.models.Operation

class MainViewModel(
    private val apiRepository: ApiRepository,
    private val settingsRepository: SettingsRepository
) : BaseViewModel<MainState, MainEvent>(init()) {

    init {
        if (!settingsRepository.id.exists()) {
            settingsRepository.id.set(
                // "123"
                ('A'..'z').map { it }
                    .shuffled()
                    .subList(0, 16)
                    .joinToString("")
                    .replace("/", "")
                    .replace("]", "")
                    .replace("\\", "")
            )
        }
    }

    fun loadFile(file: MPFile<Any>) {
        updateState {
            it.copy(operation = Operation.Preparing)
        }
        launchViewModelScope {
            try {
                apiRepository.uploadImage(
                    file = file.getFileByteArray(),
                    type = settingsRepository.id.get() ?: ""
                ).fold(
                    ifLeft = {
                        trySendEvent(MainEvent.Failure("Ошибка интернет
соединения"))
                        updateState {
                            it.copy(operation = Operation.Failure(Unit))
                        }
                        print(it)
                        print("AAAAAAA")
                    },
                    ifRight = {
                        print("AAAAAAA")
                        updateState {
                            it.copy(operation = Operation.Success(Unit))
                        }
                        trySendEvent(MainEvent.Loaded)
                    }
                )
            }
        }
    }
}

```

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		125

```

        } catch (e: Exception) {
            e.printStackTrace()
            updateState {
                trySendEvent(MainEvent.Failure("Неизвестная ошибка"))
                it.copy(operation = Operation.Failure(Unit))
            }
        }
    }
}

```

Файл Ui.kt

```

package com.niksah.gagarin.screens.history
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Scaffold
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.niksah.gagarin.screens.result.TopBarBack

@Composable
fun FullScreenUi(
    state: HistoryState,
    onBack: () -> Unit,
    onItem: (id: String) -> Unit,
    onExportCsv: (id: String) -> Unit = {}
) {
    Scaffold(
        topBar = {
            TopBarBack(onBack = onBack)
        }
    ) {
        LazyColumn(modifier = Modifier.padding(it).padding(16.dp)) {
            items(items = state.history) { historyItem ->
                HistoryItem(
                    modifier = Modifier.clickable {
                        onItem(historyItem.id) },
                    item = historyItem,
                    onExportCsv = { onExportCsv(historyItem.id) }
                )
            }
        }
    }
}

```

Файл docker-compose.yml

```

version: '3.7'

networks:
  external:
    driver: bridge
  internal:
    driver: bridge
files:
  driver: bridge

```

```

kafka:
  driver: bridge
postgres:
  driver: bridge

services:
  api:
    build:
      context: api/
      dockerfile: Dockerfile
    networks:
      - external
      - internal
      - kafka
    ports:
      - "80:8080"
    depends_on:
      - file-service
      - kafka
      - database
    environment:
      - ASPNETCORE_ENVIRONMENT=${ENV}
    restart: always
    deploy:
      replicas: 1

  file-service:
    build:
      context: file-service
    ports:
      - 10002:10002
    depends_on:
      - minio
    networks:
      - internal
      - files
    deploy:
      replicas: 1

  ml-processing-service:
    build: ./ml-processing-service
    networks:
      - internal
      - kafka
      - files
    depends_on:
      - file-service
      - kafka
    environment:
      - ENV=Production
    restart: always
    deploy:
      replicas: 1

  zookeeper:
    image: bitnami/zookeeper:latest
    networks:

```

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		127

```

    - kafka
ports:
  - 2181:2181
environment:
  - ALLOW_ANONYMOUS_LOGIN=yes

kafka:
  image: bitnami/kafka:2.8.1
  networks:
    - kafka
  ports:
    - 9092:9092
  environment:
    - KAFKA_BROKER_ID=1
    - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
    - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092
    - KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092
    - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=PLAINTEXT:PLAINTEXT
    - KAFKA_CFG_INTER_BROKER_LISTENER_NAME=PLAINTEXT
    - ALLOW_PLAINTEXT_LISTENER=yes
    - KAFKA_AUTO_CREATE_TOPICS_ENABLE=true
  depends_on:
    - zookeeper

kafka-ui:
  container_name: kafka-ui
  networks:
    - kafka
  image: provectuslabs/kafka-ui:latest
  ports:
    - 8082:8080
  environment:
    DYNAMIC_CONFIG_ENABLED: 'true'

database:
  image: postgres:16
  restart: always
  environment:
    POSTGRES_USER: 'postgres'
    POSTGRES_PASSWORD: 'root'
    POSTGRES_DB: 'production'
  networks:
    - internal
  ports:
    - "5432:5432"
  volumes:
    - database-pg:/var/lib/postgresql/data

minio:
  image: minio/minio:latest
  networks:
    - files
  ports:
    - '9000:9000'
    - "9099:9099"
  expose:
    - "9000"
  environment:

```

Приложение В

(Обязательное)

Интеллектуальная система автоматизированной обработки и семантического
анализа автомобильных документов

Презентация

					ПГУ 1.090501.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		129

Интеллектуальная система автоматизированной обработки и семантического анализа автомобильных документов

Выполнил студент группы
20ВОЭ1:
Сальников Е.В.

Научный руководитель:
д.т.н., профессор
Митрохин М.А.

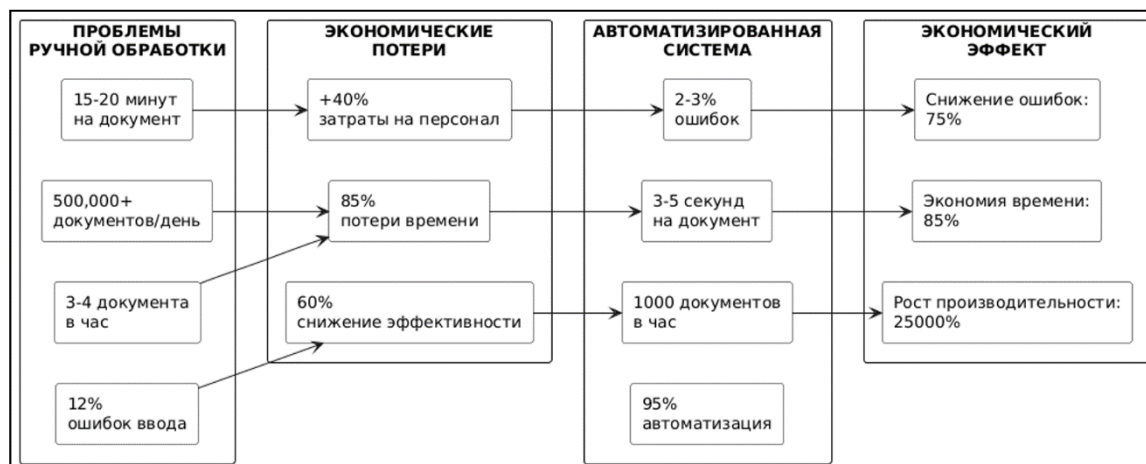
ЦЕЛЬ:

Разработка интеллектуальной системы автоматизированной обработки и семантического анализа автомобильных документов

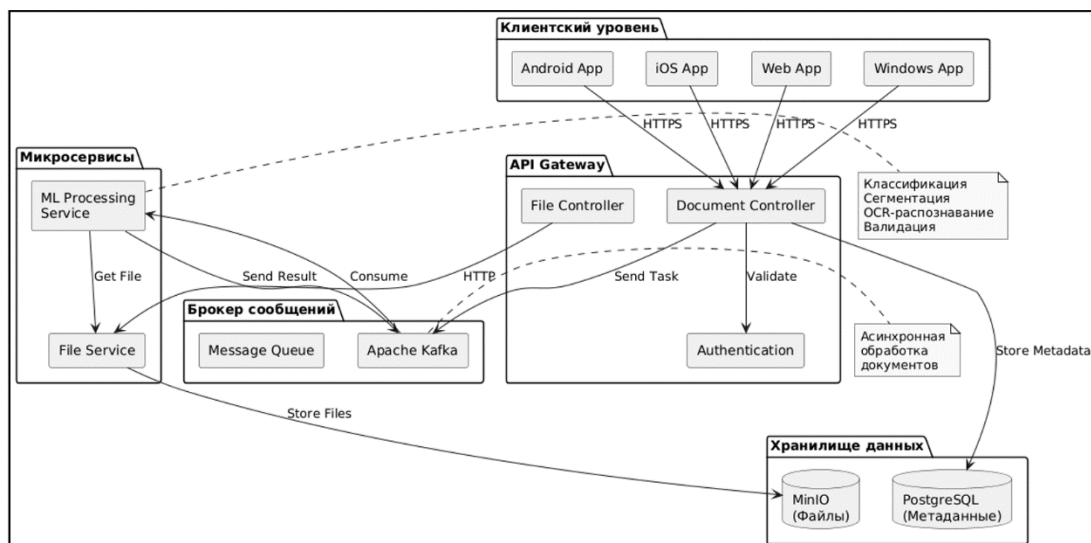
ЗАДАЧИ:

- Автоматическое сканирование и распознавание автомобильных документов
- Снижение ошибок с 12% до 2-3% благодаря автоматическому распознаванию
- Сокращение времени с 15-20 минут до нескольких секунд на документ
- Работа на всех платформах: смартфон, планшет, компьютер, веб-браузер
- Автоматическое сохранение истории и экспорт в удобных форматах (CSV, JSON)

Актуальность и проблематика



Архитектура системы



Технологический стек

СЕРВЕРНАЯ ЧАСТЬ

- API: C# ASP.NET Core 8.0
- File Service: Go + Gin
- ML: Python + PyTorch + OpenCV
- Message Broker: Apache Kafka
- DB: PostgreSQL + MinIO

КЛИЕНТСКАЯ ЧАСТЬ

- Kotlin Multiplatform
- Compose Multiplatform UI
- Android • iOS • Windows
- Linux • Web (React + TS)

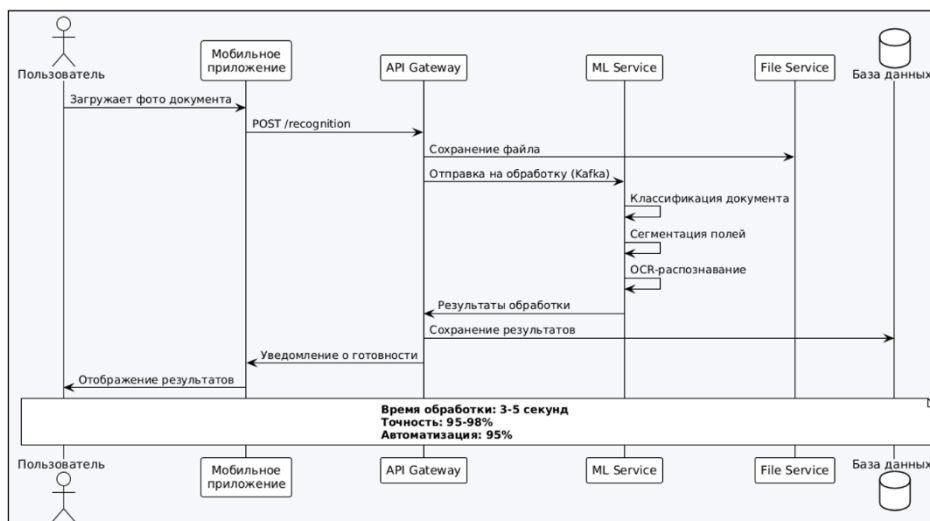
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.14.001 ПЗ

Лист

131

Диаграмма последовательности работы системы



Модуль машинного обучения



Изм.	Лист	№ докум.	Подпись	Дата

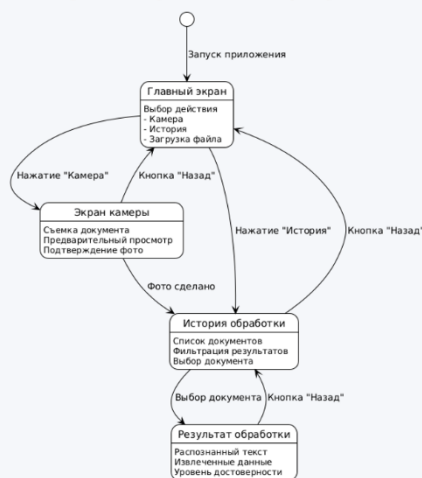
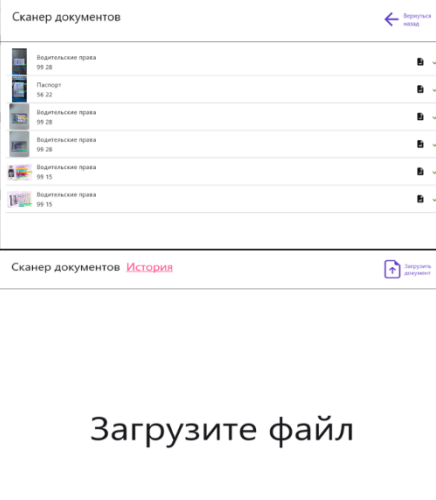
ПГУ 1.090501.14.001 ПЗ

Лист

132

Клиентские приложения

Диаграмма переходов между экранами

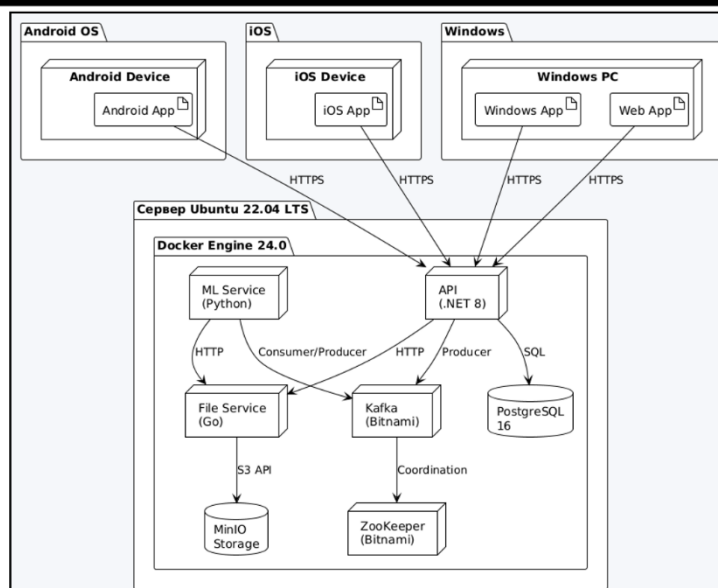


ФУНКЦИОНАЛ:

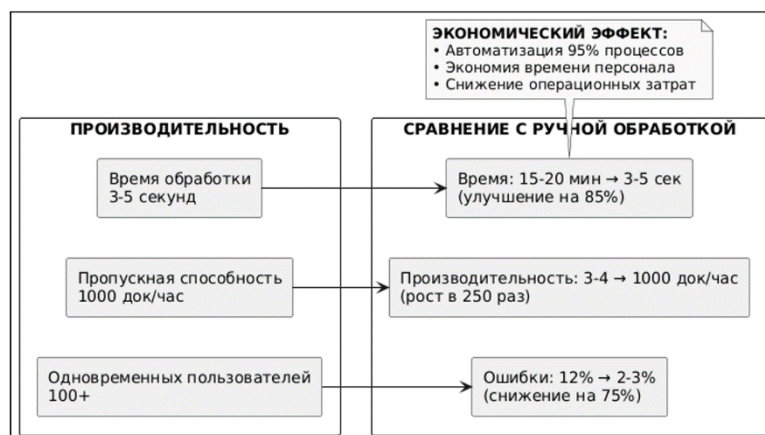
- Съемка через камеру
- Загрузка из галереи
- История обработки
- Экспорт результатов
- Кроссплатформенность

ПОДДЕРЖИВАЕМЫЕ
ПЛАТФОРМЫ:
Android • iOS • Windows •
Linux • Web

Диаграмма разворачивания системы



Результаты тестирования



ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ

- Протестировано: 500+ документов
- Покрытие тестами: 85%
- Нагрузочное: 100 пользователей
- Совместимость: 5 ОС

ПРОИЗВОДИТЕЛЬНОСТЬ

- Время обработки: 3-5 сек
- Пропускная способность: 1000 док/час
- Точность классификации: 96-98%
- Точность OCR: 94-97%

Демонстрация работы системы



ПРАКТИЧЕСКИЕ РЕЗУЛЬТАТЫ:

15-20 минут > 3-5 секунд • Ошибки 12% > 2-3% • Автоматизация 95%

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.14.001 ПЗ

Лист

134