

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

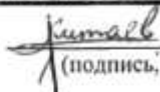
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

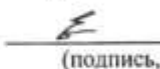
СИСТЕМА ВИДЕОАНАЛИТИКИ АВТОМАТИЧЕСКОГО КОНТРОЛЯ И УЧЁТА АВТОТРАНСПОРТА С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ КОМПЬЮТЕРНОГО ЗРЕНИЯ

Студент

 02.06.2026
(подпись, дата)

Китаев Ярослав Евгеньевич
(ФИО полностью)

Руководитель

 02.06.2026
(подпись, дата)

Гурин Е.И.
(фамилия, инициалы)

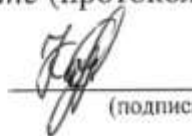
Нормоконтролёр

 02.06.2026
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой


(подпись)

Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

08 июня 2026 № 15)

Секретарь ГЭК


(подпись)

Гурин Е.И.
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Καφεδρα

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 07 » 04 2026 Г.

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Китаев Ярослав Евгеньевич гр. 22ВВП1 факультета ВТ
2. Тема работы Система видеоаналитики автоматического контроля и учета автотранспорта с использованием методов компьютерного зрения
- Тема утверждена приказом ПГУ № 347/о от "7" апреля 2026 г.
3. Руководитель бакалаврской работы Гурин Е.И.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать систему автоматического контроля и учета автотранспорта с использованием методов компьютерного зрения. Система должна уметь распознавать автотранспорт по государственному регистрационному номеру и иметь удобный графический интерфейс.

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There is no handwriting or other markings on the paper.

5. Перечень подлежащих разработке вопросов

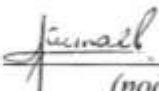
Требуется произвести анализ предметной области: изучить существующие аналоги, реализации и подходы. Выбрать язык программирования, среду разработки. Изучить нейронные сети для распознавания автотранспорта и считывания текста, библиотеки для захвата видеопотока. Выполнить настройку среды разработки и развёртывания. Спроектировать архитектуру системы. Реализовать систему. Провести тестирование получившейся системы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	16	07.04.2026 – 08.04.2026	
2	Изучение аналогов	16	09.04.2026 – 10.04.2026	
3	Выбор технологий разработки	16	11.04.2026 – 12.04.2026	
4	Изучение моделей для распознавания машин и считывания государственных регистрационных знаков	24	13.04.2026 – 15.04.2026	
5	Настройка среды разработки	8	16.04.2026	
6	Реализация серверной части программы	128	17.04.2026 – 02.05.2026	
7	Реализация веб-интерфейса	96	03.05.2026 – 14.05.2026	
8	Тестирование	16	15.05.2026 – 16.05.2026	
9	Оформление пояснительной записки	40	17.05.2026 – 21.05.2026	

Дата выдачи задания " 7 " апреля 2026 г.

Руководитель бакалаврской работы  Гурин Е.И.
(подпись, дата)

Задание к исполнению принял студент  Китаев Я.Е.
(подпись, дата)

Работу к защите допустить
Декан факультета  04.06.2020 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 152 листа, 26 рисунков, 9 таблиц, 20 источников, 3 приложения.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА, АНАЛИЗ ВИДЕОПОТОКА, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ДЕТЕКЦИЯ ТРАНСПОРТА, СЧИТЫВАНИЕ ЗНАЧЕНИЯ ГОСУДАРСТВЕННЫХ РЕГИСТРАЦИОННЫХ ЗНАКОВ, ТРЕКИНГ ОБЪЕКТОВ, PYTHON, YOLO, CRNN, POSTGRESQL, REACT, DOCKER

Цель работы – разработать систему видеоаналитики автоматического контроля и учёта автотранспорта с использованием методов компьютерного зрения.

Для достижения этой цели были выполнены анализ предметной области и существующих аналогов, спроектирована архитектура системы, а также разработаны серверная часть и веб-интерфейс.

Разрабатываемая система видеоаналитики для автоматического контроля и учёта автотранспорта должна обеспечивать детекцию транспортных средств в видеопотоке, автоматическое распознавание государственных регистрационных знаков, фиксацию направления движения, а также ведение базы данных событий с сохранением фотографий въезда и выезда.

В результате работы был создан программный модуль для автоматического контроля и учёта автотранспорта, удовлетворяющий поставленным требованиям.

					ПГУ1.090301.14.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Система видеоаналитики автоматического контроля и учета автотранспорта с использованием методов компьютерного зрения Пояснительная записка	Лит.		Лист	Листов
Разраб.		Китаев Я.Е.		02.06.20				2	68
Провер.		Гурин Е.И.		02.06.20					
Н. Контр		Бычков А.С.		02.06.20					
Утверд.		Митрохин М.А.		02.06.20					
						ФВТ, каф ВТ гр. 22ВВП1			

Содержание

Перечь сокращений и обозначений	5
Введение	7
1 Система видеоаналитики автоматического контроля и учета	
автотранспорта с использованием методов компьютерного зрения.....	8
1.1 Анализ актуальности.....	8
1.2 Описание аналогов	9
1.3 Технические требования сервера.....	13
1.4 Постановка задачи.....	14
1.5 Выбор инструментов разработки.....	17
1.5.1 Выбор языка программирования	17
1.5.2 Выбор СУБД.....	19
1.5.3 Выбор библиотеки для захвата и обработки видеопотока.....	21
1.5.4 Выбор фреймворка для обработки запросов	22
1.5.5 Выбор IDE	24
1.6 Проектирование системы видеоаналитики.....	25
1.6.1 Паттерн Service-Oriented Architecture	25
1.6.2 Архитектура приложения	27
1.6.3 Проектирование БД.....	33
1.7 Выбор и разработка моделей.....	35
1.7.1 Выбор модели для распознавания автотранспорта	35
1.7.2 Разработка модели для выделения области номерного знака	36
1.7.3 Разработка модели для считывания значения номерного знака	40
1.8 Разработка программы.....	44
1.8.1 Описание программы.....	44
1.8.2 Разработка графического интерфейса программы	53
1.9 Тестирование.....	54
1.10 Развёртывание приложения	57
2 Безопасность жизнедеятельности	64

2.1 Эргономические требования и режим труда при работе с компьютером	64
2.2 Пожарная безопасность при эксплуатации компьютерной техники	65
Заключение.....	66
Список использованных источников.....	67
Приложение А – UML диаграммы.....	69
Приложение Б – Листинг программы.....	74
Приложение В – Презентация	143

Перечь сокращений и обозначений

Данная работа содержит следующие сокращения и обозначения

БД – база данных

СУБД – система управления базами данных

API (от англ. Application Programming Interface) – программный интерфейс приложения

CNN (от англ. Convolutional Neural Network) – сверточная нейронная сеть

CPU (от англ. Central Processing Unit) – центральный процессор

CRNN (от англ. Convolutional Recurrent Neural Network) – сверточная рекуррентная нейронная сеть

CRUD (от англ. Create, Read, Update, Delete) – базовые операции с данными

CTC (от англ. Connectionist Temporal Classification) – функция потерь для последовательностей

HTTP (от англ. HyperText Transfer Protocol) – протокол передачи гипертекста

HTTPS (от англ. HyperText Transfer Protocol Secure) – защищённый протокол передачи гипертекста

IDE (от англ. Integrated Development Environment) – интегрированная среда разработки

IP (от англ. Internet Protocol) – интернет-протокол

JPEG (от англ. Joint Photographic Experts Group) – формат сжатия изображений

LSTM (от англ. Long Short-Term Memory) – тип рекуррентной нейронной сети

mAP (от англ. mean Average Precision) – средняя точность детекции

OCR (от англ. Optical Character Recognition) – оптическое распознавание символов

ORM (от англ. Object-Relational Mapping) – объектно-реляционное отображение

REST (от англ. Representational State Transfer) – архитектурный стиль взаимодействия компонентов

					ПГУ1.090301.14.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

RNN (от англ. Recurrent Neural Network) – рекуррентная нейронная сеть

RTSP (от англ. Real Time Streaming Protocol) – протокол потоковой передачи данных в реальном времени

SQL (от англ. Structured Query Language) – язык структурированных запросов

TCP (от англ. Transmission Control Protocol) – протокол управления передачей

UML (от англ. Unified Modeling Language) – унифицированный язык моделирования

URL (от англ. Uniform Resource Locator) – унифицированный указатель ресурса

USB (от англ. Universal Serial Bus) – универсальная последовательная шина

YOLO (от англ. You Only Look Once) – семейство нейросетевых моделей для детекции

					ПГУ1.090301.14.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире все больше процессов замещается системами автоматизированного контроля. Многие заводы и частные предприятия все чаще выбирают установить роботизированную систему нежели нанять человека. Система не может заболеть или совершить глупую ошибку, которая может принести к большим убыткам.

Одной из таких сфер является учёт автотранспорта на территории того или иного учреждения. Традиционные методы учёта автотранспорта, основанные на ручном контроле, имеют большое количество недостатков, включая человеческий фактор, который может привести к ошибкам и снижению точности данных. В связи с этим всё больше предприятий выбирают системы автоматизированного контроля, которые обеспечивают более высокую надёжность и точность учёта.

Разрабатываемая система представляет собой решение, которое позволяет минимизировать роль охранника в процессе проверки транспорта. Это позволит снизить ошибки и оптимизировать управления транспортом.

					ПГУ1.090301.14.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Система видеоаналитики автоматического контроля и учета автотранспорта с использованием методов компьютерного зрения

1.1 Анализ актуальности

В последние годы нейронные сети стали развиваться стремительными темпами, что позволяет внедрять технологии искусственного интеллекта во всё более разнообразные сферы производства и жизни людей. Одной из таких сфер является проверка и учёт автотранспорта при въездах на территорию охраняемого офиса и склада.

В настоящий момент проверка транспорта в компании «Феникс» осуществляется охранником, он вручную проверяет многочисленные списки, сверяет их с номером машины, после чего принимает решение о возможности доступа транспорта на охраняемую территорию. Такой подход влечёт за собой огромное количество неудобств.

Во-первых, вместо того, чтобы следить за безопасностью на территории офиса путём её патрулирования, охранник обязан всегда находится вблизи въезда.

Во-вторых, большую роль играет человеческий фактор, пока охранник проверит все списки разрешённого транспорта может пройти не мало времени, что в итоге приведёт к образованию большой очереди на въезд. Также возникает большая вероятность ошибок, связанных с невнимательностью или усталостью сотрудника.

В-третьих, это сильно ограничивает возможность ведения списков учёта транспорта посетившего территорию охраняемого предприятия.

Для решения данной проблемы было разработано большое количество разнообразных систем, включая цифровые системы ведения баз данных автотранспортных средств, которым разрешён въезд на территорию офиса.

					ПГУ1.090301.14.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

Одной из самых передовых систем для решения поставленной задачи, является комплекс с использованием видеоаналитики и методов компьютерного зрения для обнаружения и распознаванию государственных регистрационных знаков. Такая система позволяет максимально минимизировать человеческий фактор, так как не только хранит информацию о транспорте, но и автоматически распознаёт номерные знаки, сравнивает их с данными из БД, управляет штрафбаумом и ведёт учёт автотранспорта.

Актуальность разработки такой системы обусловлено необходимостью увеличения пропускной способности и уменьшению нагрузки на рабочий персонал.

1.2 Описание аналогов

Обзор и анализ существующих аналогов является одним из важнейших этапов разработки любой системы. Данный этап позволяет проанализировать существующие системы в исследуемой области, выявить их достоинства и недостатки, определить, что стоит добавлять в разрабатываемую систему, а что будет избыточным для неё. В моём случае я просмотрел существующие на рынке системы контроля и управления доступом транспортных средств.

В ходе анализа было выявлено два основных подхода к реализации подобных систем. Первый подход – это серверная реализация, где основные вычисления производятся на сервере. Второй подход – это использование автономных камер, которые содержат в себе необходимое программное обеспечение для распознавания автомобильных номеров и управления устройствами.

Рассмотрим несколько представителей описанных выше подходов к реализациям изучаемой системы.

Первый аналог – это «UniServer AUTO: Контроль проезда» [1] этот продукт представляет собой программное обеспечение, реализующее серверный подход. Система включает в себя возможность распознавания

государственных регистрационных знаков автотранспорта, ведение журнала событий посредством использования базы данных, имеет удобный графический интерфейс для пользователя, показано на рисунке 1, работает со шлагбаумами, обладает возможностью масштабирования путём подключения нескольких камер для отслеживания.

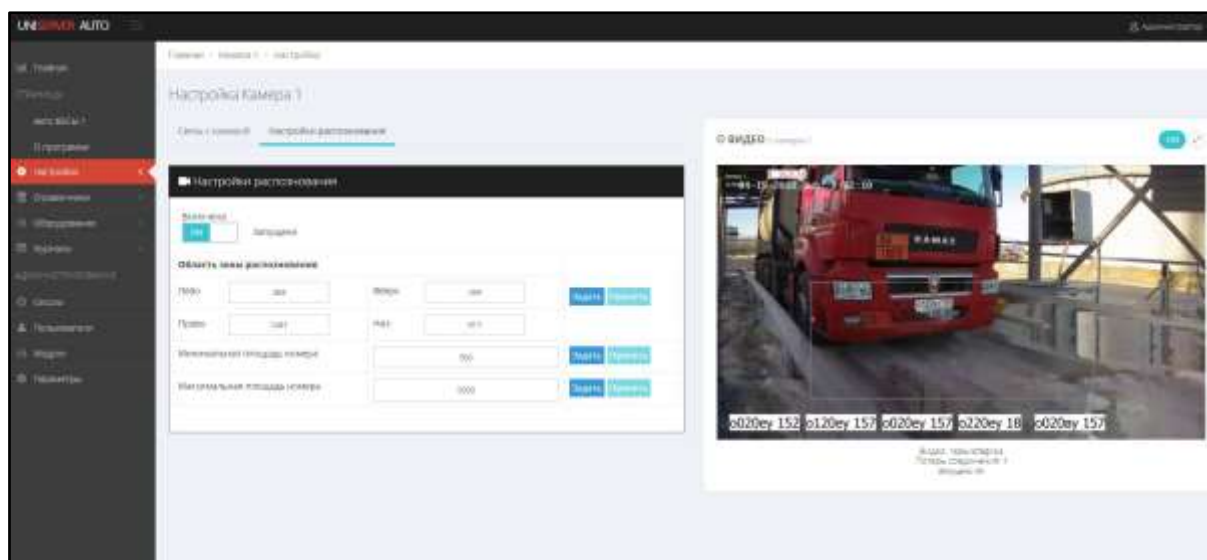


Рисунок 1 – Обзор аналога «UniServer AUTO: Контроль проезда»

Основными недостатками системы «UniServer AUTO: Контроль проезда» являются её коммерческая основа и закрытый исходный код, что не позволит просто добавить систему в общую видеосистему компании.

Второй аналог – система «Автомаршал» [2], которая также является серверным решением и представляет собой специализированное программное обеспечение, устанавливаемое на сервер. Функционал системы включает в себя распознавание автомобильных номеров с подключаемых камер, занесение событий в базу данных, формирование отчётов, а также управление шлагбаумом и другими внешними устройствами через специальные контроллеры. Пример интерфейса системы показан на рисунке 2.



Рисунок 2 – обзор аналога «Автомаршал»

Однако, как и в предыдущем примере, система является платной и имеет закрытый исходный код.

Третий аналог – продукт «DVS-102-UN» [3] от компании RusGuard. В отличие от описанных выше примеров этот продукт представляет собой аппаратный комплекс, реализующий автономный подход, показано на рисунке 3. Этот продукт представляет собой камеру, которая оснащена модулем распознавания автомобильных номеров и собственной базой данных для хранения разрешённых номеров. Главным достоинством описываемой камеры является возможность работы даже при отсутствии подключения к интернету.



Рисунок 3 – обзор аналога «DVS-102-UN»

Несмотря на высокую надёжность и отказоустойчивость, данный аналог обладает рядом существенных минусов. Во-первых, сложность централизованного управления базами разрешённых номеров в нескольких камерах. Во-вторых, проблематичность масштабирования системы для отслеживания нового въезда необходимо докупать дорогостоящее оборудование. В-третьих, система является закрытой и платной. В-четвёртых, отсутствует встроенный графический интерфейс.

Четвёртый аналог – устройство «Трал Паркинг 2» [4] также относится к классу автономных аппаратных комплексов. Оно выполнено в виде устройства, которое обрабатывает видеосигнал, от подключённой камеры. Решение об открытии шлагбаума принимается локально на основе базы номеров хранящихся на USB-накопителе, установленном в устройство, пример на 4 рисунке.

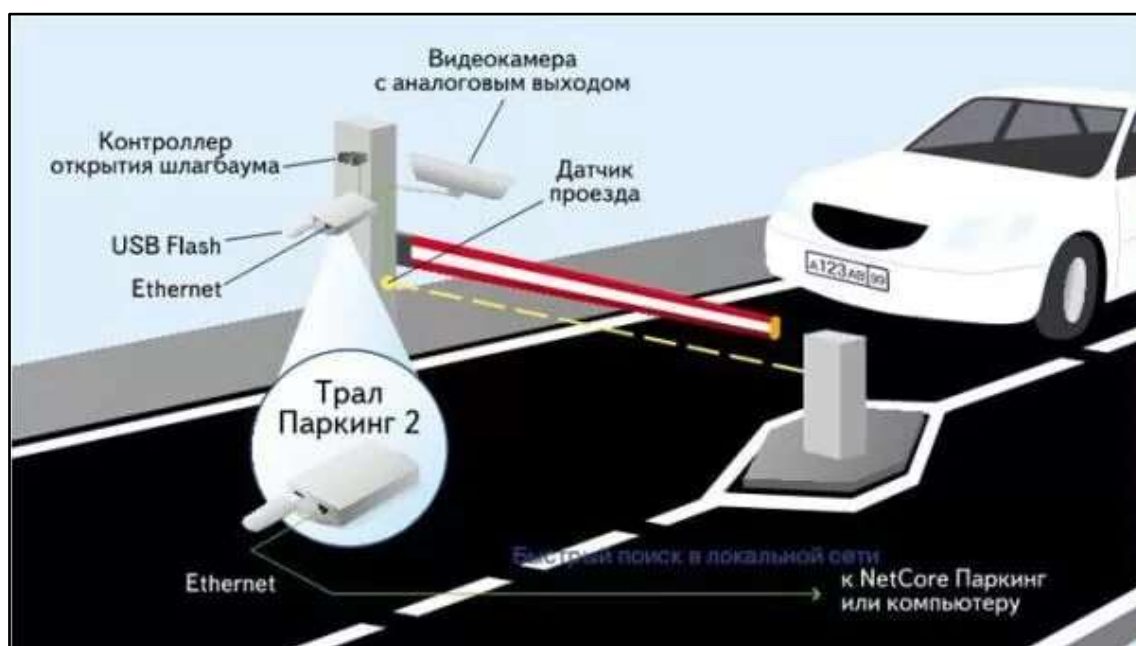


Рисунок 4 – обзор аналога «Трал Паркинг 2»

К недостаткам, как и у предыдущего автономного решения, можно отнести отсутствие графического интерфейса для управления, закрытую

архитектуру. Кроме того, хранение важной информации на съёмном USB-накопителе не безопасно.

Исходя из изученных систем, можно сделать вывод, что на рынке существуют как мощные серверные решения с возможностями интеграции и масштабирования, так и автономные устройства. Однако все рассмотренные все рассмотренные продукты объединяет одно: они являются платными и распространяются с закрытым исходным кодом. Это усложняет возможность интегрирования в систему видеоаналитики компании. Таким образом разработка собственной системы является актуальной задачей.

В таблице 1 представлено сравнение существующих аналогов.

Таблица 1 – Сравнение аналогов

Название компании	Встроенный графический интерфейс	Простое масштабирование	Платная система	Открытый исходный код
Автомаршал	+	+	+	-
Uniserver Auto: Контроль проезда	+	+	+	-
Камера распознавания автомобильных номеров DVS-102-UN От rusguard	-	-	+	-
«Трал Паркинг 2»	-	-	+	-

1.3 Технические требования сервера

Доступ к разрабатываемой системе будет осуществляться через сайт, для его работы и работы системы необходимо развернуть сервер.

Требования к серверу:

- процессор не ниже Intel Core i5;
- видеокарта не ниже Nvideo Geforce RTX 3060;
- объем оперативной памяти не менее 16 ГБ;
- объем встроенной памяти не меньше 256 ГБ;
- операционная система Ubuntu не ниже 22 версии;
- установленный Docker Engine версии не ниже 20.10 версии;
- установленный Docker compose версии не ниже 2.32.1 версии.

1.4 Постановка задачи

Необходимо разработать систему видеоаналитики автоматического контроля и учёта автотранспорта на охраняемой территории с использованием методов компьютерного зрения. Она должна обеспечивать следующий функционал:

- автоматически распознавать государственные регистрационные номера;
- фиксацию времени въезда/выезда;
- верификацию распознанных номеров с базой данных разрешённых автомобильных номеров;
- автоматическое управление шлагбаумом;
- представление удобного интерфейса для использования.

Кроме того, разрабатываемая система должна иметь возможность интегрирования в общую систему видеоаналитики компании.

Система должна строиться на основе микро сервисной архитектуры, чтобы обеспечить её масштабируемость, отказоустойчивость и встраиваемость:

- модуль компьютерного зрения: Взаимодействие с IP-камерами по RTSP, обнаружение и выделение транспортных средств в кадре, выделение

					ПГУ1.090301.14.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

области номера из кадра автомобиля и распознавание символов, подсчёт времени, принятие решения об открытии шлагбаума, занесение данных в БД;

- модуль хранения данных;
- модуль бизнес логики;
- модуль графического интерфейса: обрабатывает пользовательские запросы и помогает визуализировать данные, получаемые от других модулей.

Для удобного взаимодействия с системой необходимо разработать веб-интерфейс:

- аутентификация пользователя по логину и паролю с разграничением прав доступа;
- отображение таблица с информацией о заехавших машинах и времени их пребывания на охраняемой территории;
- панель администратора, через которую необходимо вносить номера машин, которым разрешён въезд, информацию о владельцах транспортных средств;
- при нажатии на номер, выводить информацию о владельце;
- при нажатии на время въезда или выезда транспорта выводить фотографии, сделанные в тот момент;
- инструмент для поиска записей в базе данных;
- просмотр камер в реальном времени;
- кнопка для управления шлагбаумом.

Необходимо реализовать три уровня доступа к системе:

Права администратора показаны на рисунке 5:

- выдача прав доступа;
- добавление новых камер;
- добавление линий;
- добавление водителей;
- добавление транспортных средств;

- просмотр списка камер;
- просмотр страниц камер;
- поиск водителей.

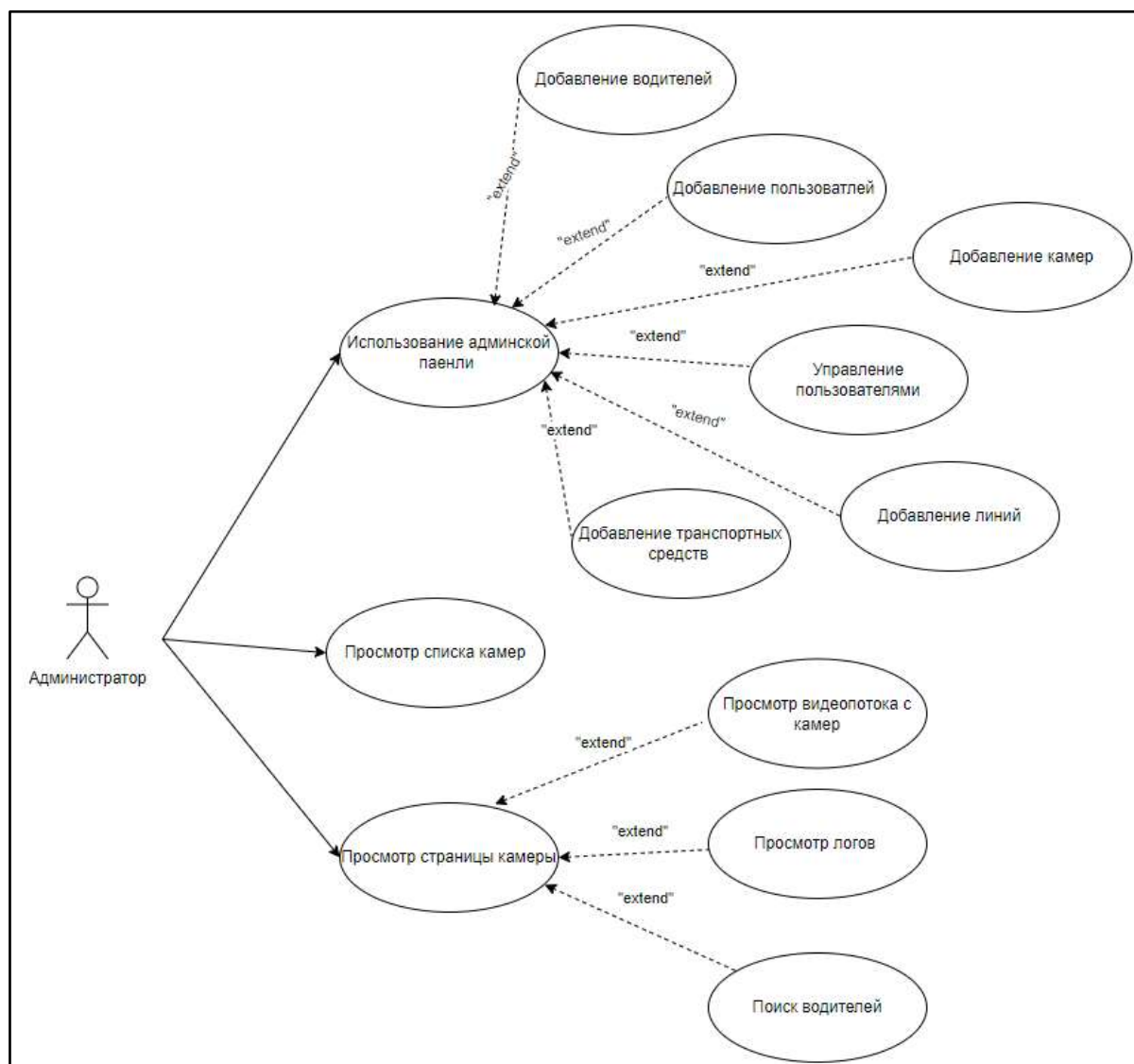


Рисунок 5 – UML – диаграмма вариантов использования для администратора

Права охранника показаны на рисунке 6:

- добавление водителей;
- добавление транспортных средств;
- просмотр списка камер;
- просмотр страниц камер;
- поиск водителей.

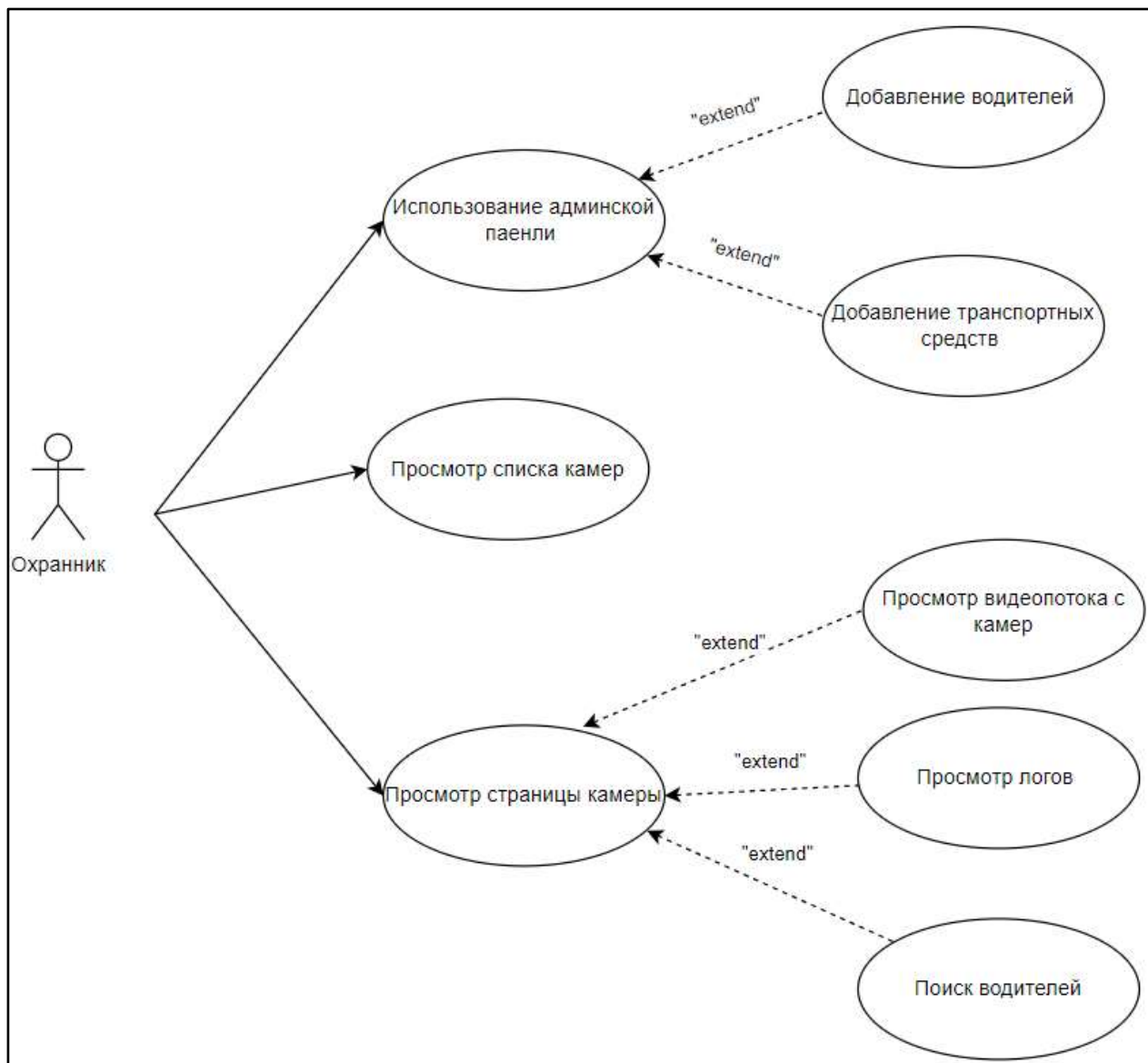


Рисунок 6 – UML – диаграмма вариантов использования для охранника

1.5 Выбор инструментов разработки

1.5.1 Выбор языка программирования

Выбор языка программирования является одним из важнейших этапов в разработке любого приложения. Все языки программирования обладают как плюсами, так и минусами в определённых областях. Необходимо выбрать такой язык программирования, который будет хорошо подходить под поставленные задачи, а именно захват видео потока, работа с нейронными сетями и обрабатывание HTTPS запросов. Учитывания эти требования было

проанализировано несколько языков программирования, которые были изучены в университете, он имеет свои плюсы и минусы.

Java предлагает большой выбор библиотек для разработки и масштабирования серверных приложений. Платформа обеспечивает высокую устойчивость к нагрузкам за счёт встроенного управления памятью и поддерживает параллельную обработку множества клиентских соединений. Также Java имеет богатый выбор фреймворков для разработки серверной части приложения, например Spring Framework позволяет быстро создавать корпоративные приложения с готовой поддержкой аутентификации, авторизации и интеграции с базами данных, а Jakarta EE предоставляет стандартизированные модели для крупномасштабных систем. Однако в области обработки видеопотока и работы с нейронными сетями Java показывает не очень хороший результат. Хотя и существуют для работы с нейронными сетями, они не очень популярны, что приводит к трудности их изучения. Также Java уступает в скорости обработки вычислений из-за накладных расходов JVM и менее прямого доступа к памяти.

C/C++ является одним из самых производительных языков программирования в области компьютерного зрения и обработки видеопотока. Для задач по захвату видеопотока в реальном времени и выполнения нейронных сетей на процессоре и видеокарте C++ обеспечивает наиболее полный контроль над аппаратными ресурсами. Также многие фреймворки для захвата и обработки видеопотока, например «OpenCV» и «GStream», были написаны на языке C/C++, что гарантирует минимальные задержки при выполнении математических операций. Но, несмотря на очень хорошие показатели в работе с компьютерным зрением и обработкой видеопотока, C/C++ имеет значительные недостатки при разработке серверной части, в частности необходимость ручной реализации базовых компонентов. Также, стоит отметить, малое количество готовых фреймворков, которые упрощают разработку серверной части.

С учётом проведённого анализа был выбран язык программирования Python. Несмотря на то, что он уступает C/C++ в скорости выполнения низкоуровневых операций, это нивелируется тем, что все операции по обработке нейронных сетей и видеопотока будут выполняться через скомпилированные модули, написанными на C/C++, библиотеки «OpenCV» и «GStream». Также Python имеет большой выбор фреймворков для разработки серверной части сайта, среди которых Django – высокоуровневый фреймворк со встроенной админ-панелью и аутентификацией, FastIP – современный асинхронный фреймворк для создания API с автоматической документацией. Сочетание этих факторов делает Python лучшим выбором для поставленной задачи.

1.5.2 Выбор СУБД

При разработке любого приложения стоит ответственно отнестись к выбору СУБД. Благодаря правильно подобранному способу хранения данных и обращению к ним, можно ускорить работу системы, упростить её масштабирование и обеспечить надёжность.

Прежде чем остановиться на выборе одной СУБД, необходимо разобраться, какие существуют БД и для каких задач они подходят.

Самым популярным видом БД являются реляционные. Данные в них хранятся в виде таблиц с явными связями, что гарантирует целостность данных и простоту их получения. Реляционные БД используют язык SQL для доступа к данным.

В противовес им существуют нереляционные БД. Они имеют гибкую структуру, что позволяет хранить разнообразные виды данных и выдерживать высокие нагрузки. В нереляционных базах данные могут храниться в виде документов, ключ-значений, графов и т.д. Это позволяет легко масштабироваться.

Сравнив эти два подхода к организации структуры БД, было решено использовать реляционные БД, потому что данные в приложении будут храниться в текстовом формате и требуют строгой структуры.

MySQL — одна из самых популярных реляционных СУБД. Она проста в настройке, высоко производительная при типовых задачах и имеет большой список документации. MySQL подходит под сценарии с преимуществом операций чтения, что делает её хорошим выбором для блогов, интернет-магазинов и других контент-ориентированных платформ. Однако в сложных сценариях, требующих дополнительных функций или кастомизации, она может оказаться менее гибкой.

PostgreSQL, в свою очередь, выделяется расширенным набором возможностей. Эта СУБД поддерживает сложные типы данных, включая JSON и массивы, предлагает обширный выбор инструментов для работы. PostgreSQL активно развивается, регулярно получает обновления и отличается высокой надёжностью. Она хорошо масштабируется, подходит для проектов с интенсивными записями и сложными запросами, а также позволяет расширять функционал за счёт пользовательских функций и плагинов.

SQLite — это встраиваемая СУБД, которая не требует отдельного сервера и работает как библиотека внутри приложения. Все данные хранятся в одном файле, что делает SQLite очень удобной для локальных решений: мобильных приложений, десктоп-программ и т.д. Её главные преимущества — минимальная настройка и компактность. SQLite отлично подходит для кэширования данных, хранения настроек, работы в офлайн-режиме или тестирования. Однако из-за архитектуры она не рассчитана на высоконагруженные проекты и сценарии с одновременным доступом множества пользователей.

Учитывая все факторы, для работы проекта было решено использовать PostgreSQL. Она обеспечивает необходимый уровень гибкости и надёжности, подходит для масштабирования и даёт запас возможностей на случай роста

нагрузки. В отличие от SQLite, PostgreSQL поддерживает многопользовательский доступ и сложные транзакции, а по сравнению с MySQL предлагает более широкие возможности для работы с разнообразными типами данных и сложными запросами.

1.5.3 Выбор библиотеки для захвата и обработки видеопотока

Выбор правильной библиотеки для захвата и обработки видеопотока позволит терять как можно меньше кадров, что очень важно для видеоаналитики, а также эффективно выполнять предобработку кадров, для более точного распознавания автомобильных номеров.

Для выбора такой библиотеки было проанализировано несколько вариантов совместимых с Python.

Первый – OpenCV [5], она одна из наиболее часто используемых библиотек для решений задач компьютерного зрения и обработки видео. OpenCV поддерживает захват видео с различных источников: веб-камер, локальных видеофайлов, сетевых потоков. Она предоставляет удобные решения для работы с кадрами, их предварительной обработки, а также хорошо интегрируется с библиотеками машинного обучения и фреймворками для построения серверных приложений.

Второй – GStreamer [6], это мультимедийный фреймворк с поддержкой Python-привязок. В отличие от узкоспециализированных библиотек, GStreamer представляет собой модульную систему конвейеров, где каждый элемент выполняет определённую функцию: захват, декодирование, фильтрацию, кодирование, вывод и многое другое. Это позволяет гибко настраивать обработку видеопотока. GStreamer поддерживает обширный спектра форматов и протоколов передачи видеопотока, а также работу с несколькими источниками одновременно. Фреймворк отличается высокой производительностью и масштабируемостью, что делает его подходящим для обработки видео в реальном времени.

Третьим была рассмотрена `imageio` [7] – библиотека Python для чтения и записи изображений и видео. Она абстрагирует низкоуровневые детали работы с форматами, предоставляя простой API для захвата и сохранения видеопотока. `imageio` поддерживает множество форматов файлов и плагинов, что упрощает интеграцию с существующими инструментами. Однако по функциональности и гибкости настройки она уступает `OpenCV` и `GStreamer`, особенно в сценариях, требующих сложной обработки или потоковой передачи данных в реальном времени.

После сравнения трёх вариантов стало понятно, что `OpenCV` и `GStreamer` дополняют друг друга: `GStreamer` надёжно забирает поток с камеры по RTSP и декодирует его, а `OpenCV` работает уже с отдельными кадрами.

`GStreamer` используется для надёжного и производительного захвата и маршрутизации видеопотока, особенно при работе с сетевыми источниками и необходимости аппаратного ускорения. Гибкость конвейеров `GStreamer` даёт возможность внедрять фильтрацию, транскодирование и буферизацию на ранних этапах обработки.

`OpenCV` применяется для непосредственной работы с видеокадрами: их предобработки, анализа с помощью алгоритмов компьютерного зрения и интеграции с моделями нейронных сетей.

Комбинация этих библиотек позволяет достичь баланса между производительностью захвата, гибкостью настройки конвейера и удобством дальнейшей обработки видеоданных.

1.5.4 Выбор фреймворка для обработки запросов

Для реализации серверной части приложения с обработкой HTTPS-запросов был проведён анализ популярных фреймворков на Python. Основными критериями для выбора стали: производительность, масштабируемость, наличие встроенных механизмов безопасности, поддержка REST API, простота интеграции с другими компонентами системы.

Django [8] — высокоуровневый веб-фреймворк, ориентированный на быструю разработку полноценных веб-приложений. Он даёт возможность использовать обширный набор встроенных функций: ORM для работы с базами данных, систему аутентификации и авторизации, админ-панель, поддержку миграции схем БД и встроенную документацию API. Django позволяет существенно сократить время разработки за счёт готовых решений. Фреймворк хорошо масштабируется для корпоративных приложений и имеет обширную документацию и активное сообщество, что упрощает поиск решений типовых задач.

Flask [9] — простой микрофреймворк с минималистичным ядром. Flask предоставляет базовый функционал для обработки HTTP-запросов и позволяет подключать расширения по мере необходимости. Его главным достоинством является простота освоения и низкая нагрузка на систему. Однако для реализации сложной функциональности, например панель администратора требуется подключение большого количества дополнительных библиотек, что увеличивает время разработки и риск ошибок.

FastAPI [10] — современный асинхронный фреймворк для создания API. Он использует аннотации типов Python и автоматически генерирует интерактивную документацию. FastAPI очень производительный благодаря асинхронной обработке запросов и поддержке WebSockets. Фреймворк оптимизирован для создания REST и GraphQL API, хорошо интегрируется с асинхронными БД. Однако встроенные механизмы аутентификации и админ-панели требуют дополнительной настройки. FastAPI оптимален для высоконагруженных API-сервисов, но может быть избыточным для приложений с традиционной веб-интерфейсной частью.

После сравнение этих трёх фреймворков был выбран Django, потому что он включает в себя все необходимые компоненты для серверной части: обработку HTTPS-запросов, аутентификацию, админ-панель, ORM, защиту от

атак. Это позволит сократить время разработки и снизить зависимость от сторонних библиотек.

1.5.5 Выбор IDE

При разработке любого приложения важно подобрать удобную среду разработки — она влияет на скорость написания кода, отладку, визуализацию данных и интеграцию с библиотеками. Рассмотрим несколько популярных вариантов.

PyCharm [11] — IDE от компании JetBrains, заточенная под Python. Она обеспечивает интеллектуальное автодополнение кода и возможности для рефакторинга, поддерживает популярные фреймворки и предлагает удобные инструменты для отладки. Кроме того, PyCharm позволяет работать с виртуальными окружениями и управлять зависимостями, а также предоставляет расширенные возможности для взаимодействия с базами данных и веб-технологиями. Однако PyCharm требует значительных ресурсов компьютера и может быть избыточным решением для небольших задач.

Visual Studio [12] и Visual Studio Code [13] — продукты Microsoft, но с разными подходами. Visual Studio представляет собой полнофункциональную IDE для множества языков, включая Python, который поддерживается через расширение Python Tools for Visual Studio. Эта среда хорошо подходит для крупных мультязычных проектов благодаря мощным инструментам отладки и тестирования. В то же время она достаточно тяжеловесна, потребляет много ресурсов системы и менее распространена среди Python-разработчиков, особенно тех, кто занимается машинным обучением.

VS Code — лёгкий, быстрый и расширяемый редактор кода. Он отличается высокой производительностью и гибкостью за счёт богатой экосистемы расширений: можно настроить поддержку Python, установить плагины для работы с множеством библиотек, интегрировать инструменты для анализа данных и машинного обучения. VS Code поддерживает виртуальные

окружения, предлагает удобные средства отладки и подсветку синтаксиса, а также позволяет запускать и тестировать код прямо в редакторе.

Anaconda [14] вместе со Spyder [15] —представляет собой дистрибутив Python с предустановленными библиотеками для научных вычислений и менеджером пакетов Conda, который упрощает управление зависимостями и создание изолированных окружений. Spyder — это IDE, входящая в состав Anaconda и спроектированная специально для научных расчётов и работы с данными. Она предлагает встроенную консоль IPython, обозреватель переменных, инструменты для визуализации и удобную интеграцию с библиотеками машинного обучения. Такое сочетание хорошо подходит для обучения моделей и анализа данных, где важна быстрая проверка гипотез и визуальная оценка результатов.

С учётом всех факторов для реализации проекта оптимальным выбором будет являться комбинация VS Code, Anaconda и Spyder. VS Code обеспечит гибкий и производительный редактор кода с широкими возможностями настройки и поддержкой современных инструментов разработки. Anaconda даст доступ к проверенному набору библиотек для машинного обучения и надёжному менеджеру зависимостей, а Spyder предоставит удобную среду для визуализации и быстрой отладки моделей. нейронные сети без излишней нагрузки на систему.

1.6 Проектирование системы видеоаналитики

1.6.1 Паттерн Service-Oriented Architecture

Сервис ориентированная архитектура – это паттерн, при котором система состоит из независимых или слабо зависимых модулей, каждый из которых реализует определённую функцию. Сервисы взаимодействуют между собой через протоколы и интерфейсы, это позволяет разрабатывать, масштабировать и развёртывать их независимо друг от друга.

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		25

В разрабатываемой системе сервисно-ориентированная архитектура представлена двумя независимыми сервисами, каждый из которых решает разные задачи.

Сервис видеоаналитики отвечает за захват видеопотока с камеры, детекцию, трекинг транспортных средств, распознавание государственных регистрационных знаков с помощью нейронной сети, а также за потоковую передачу обработанных кадров клиентскому приложению в режиме реального времени.

Сервис API представляет REST-интерфейс для обработки запросов от сайта части, работу с панелью администратора и загрузку файлов.

Оба сервиса работают с единой базой данных PostgreSQL. Сервис видеоаналитики отвечает за сохранение результатов распознавания и регистрацию событий въезда/выезда. Сервис API обеспечивает доступ к этим данным со стороны клиентского приложения, а также управляет конфигурацией камер и учётными записями пользователей.

Взаимодействие между сервисами и клиентским приложением происходит через прокси, который перенаправляет запросы в необходимые сервисы.

Выбор сервис-ориентированной архитектуры для системы видеоаналитики сделан на основе следующих фактов. Во-первых, разделение функциональности видеоаналитики и REST API на независимые сервисы позволяет использовать разный стек технологий, подходящий под конкретную задачу. Во-вторых, такое разделение даёт возможность независимого масштабирования: при увеличении количества камер будет достаточно масштабировать только сервис видеоаналитики, не затрагивая API-сервис. В-третьих, слабая связанность сервисов упрощает изменение в отдельные сервисы.

1.6.2 Архитектура приложения

Архитектура системы видеоаналитики строится по клиент-серверному принципу с разделением на фронтенд и бэкенд части, а также с отдельным модулем для распознавания с помощью нейронных сетей и интеграцией с сетевыми видеоисточниками.

Внешнее взаимодействие с системой производится через модуль прокси-сервера Nginx, который принимает все входящие HTTP и WebSocket-соединения, маршрутизируя статический контент и запросы к API на нижележащие сервисы, что обеспечивает безопасность, балансировку нагрузки.

Модуль бизнес-логики, представлен в виде фреймворка Django. Он отвечает за управление пользователями, аутентификацию на основе JWT-токенов, CRUD-операции с разными сущностями: камеры, сотрудники и транспортные средства, а также за предоставление структурированного REST API для фронтенд-приложения.

Наиболее сложной и ресурсоёмкой частью системы является модуль видеоаналитики, который является асинхронным приложением. Данный сервер выполняет всю логику захвата, декодирования и обработки видеопотоков, полностью изолируя эти процессы от веб-части. Захват видеоданных реализован с помощью фреймворка GStreamer, который обеспечивает надежное подключение к RTSP-камерам, аппаратно-ускоренное декодирование видеокодеков и эффективную предобработку кадров с минимальными задержками, преобразуя сетевой поток в последовательность изображений, пригодных для дальнейшей обработки. Полученные кадры передаются в модуль компьютерного зрения, реализованный через библиотеку OpenCV, которая выполняет операции масштабирования, аннотирования и отрисовки вспомогательных графических элементов.

Ключевым звеном аналитического конвейера является двухэтапный процесс детекции и распознавания, реализованный на базе моделей семейства

YOLO. На первом этапе предобученная модель YOLOv8m выполняет детекцию и трекинг транспортных средств в видеопотоке, присваивая каждому объекту уникальный идентификатор и отслеживая его перемещение для последующего определения направления движения путём математического анализа пересечения вектора движения с заданными в конфигурации линиями. После того как направление движения транспортного средства распознано, на втором происходит считывания из области с транспортным средством номерного знака при помощи модели YOLOv8n, которая была переобучена на датасете для решения задачи детекции государственных регистрационных знаков. Выделенные изображения номерных знаков подаются на вход сверточной рекуррентной нейронной сети, выступающей в роли распознавателя символов. Модель CRNN обеспечивает извлечение текстовой строки номерного знака, при этом для повышения точности применяется алгоритм стабилизации результатов на основе голосования по истории последовательных распознаваний для одного и того же трека объекта.

Обмен данными между веб-сервером Django и модулем видеоаналитики организован через общую базу данных PostgreSQL, куда модуль с распознаванием номеров, сохраняет зафиксированные события въезда и выезда, включая распознанный номер и путь к сохранённому полноразмерному кадру. Также этот модуль в реальном времени транслирует аннотированные JPEG-кадры и метаданные о текущих распознаваниях напрямую подключенным веб-клиентам по протоколу WebSocket через библиотеку Socket.IO, что позволяет отображать видео и события в интерфейсе без лишних задержек и избыточной нагрузки на HTTP-сервер.

Таким образом, предложенная архитектура обеспечивает эффективное разделение ответственности: Django отвечает за управление данными и REST API, в то время как изолированный и высокопроизводительный Eventlet-сервер с GStreamer, OpenCV и трио нейросетевых моделей решает задачу потокового распознавания в реальном времени.

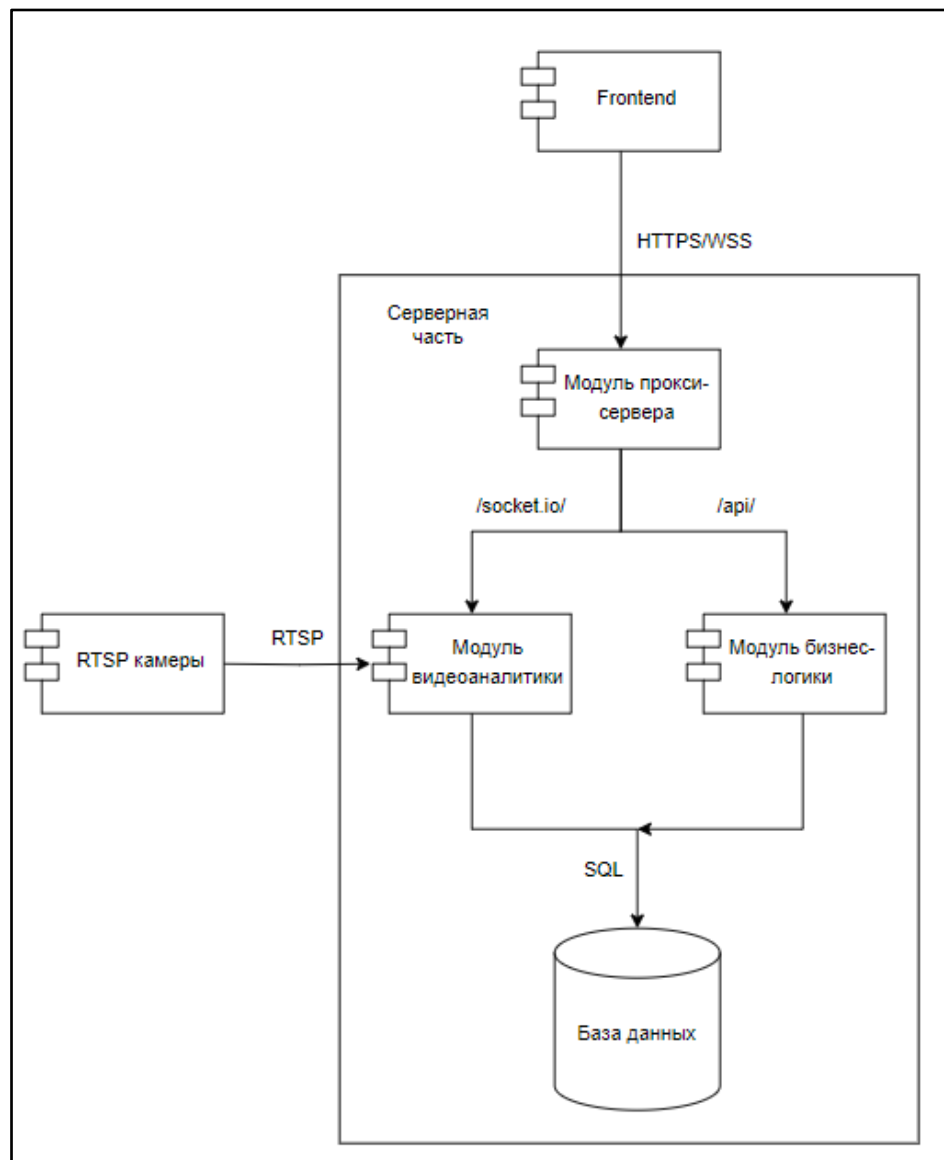
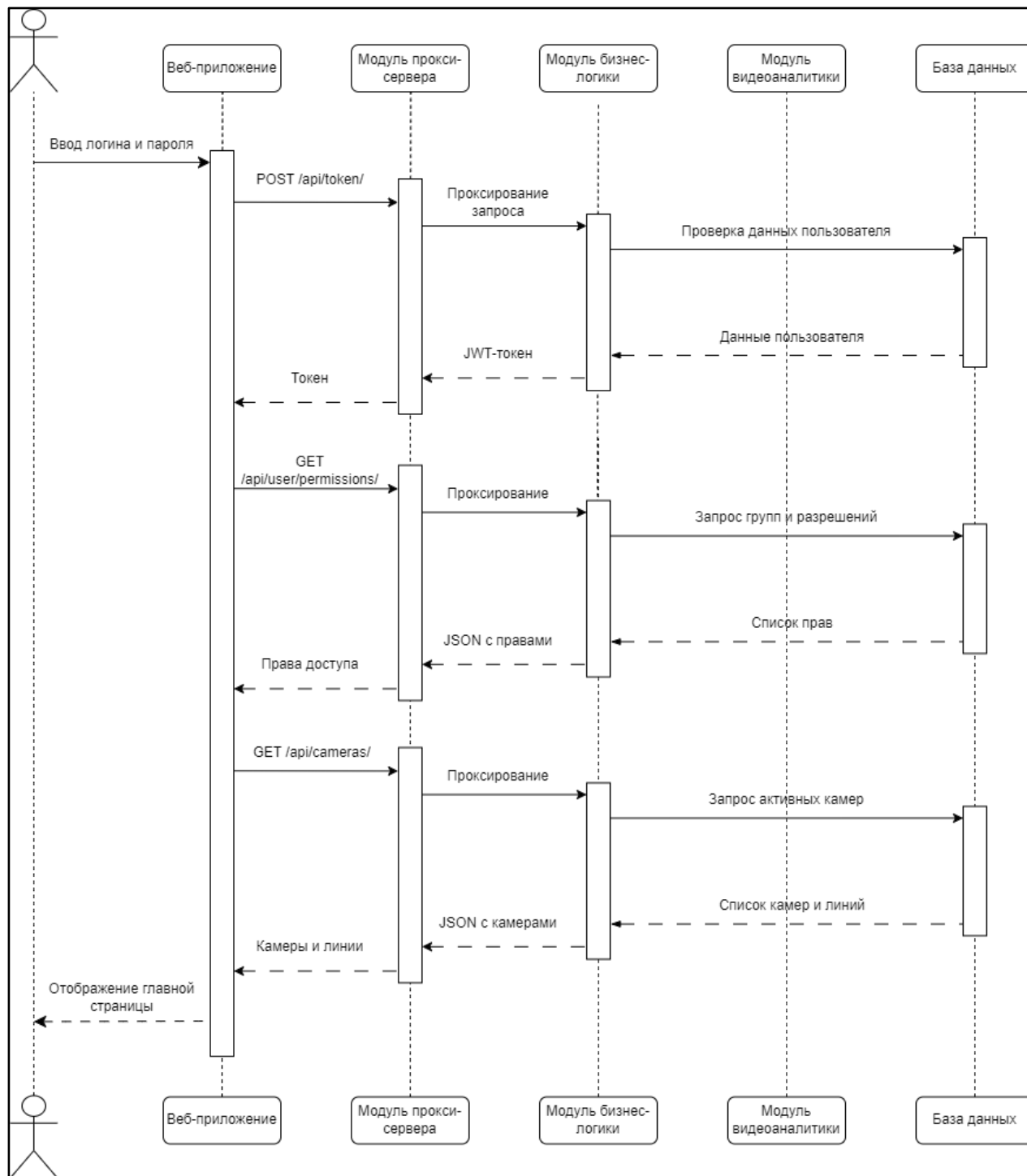
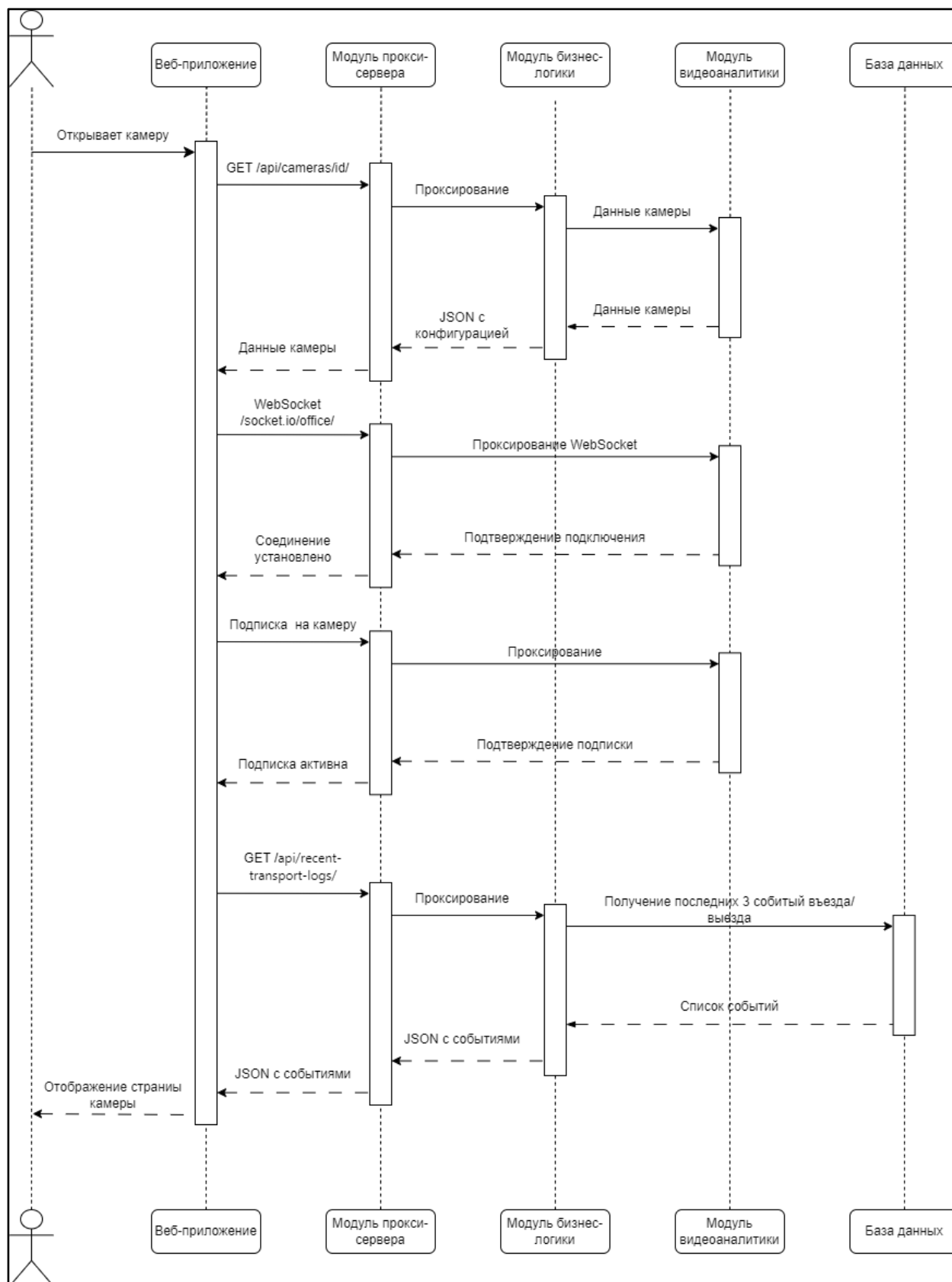


Рисунок 7 – Архитектура модулей

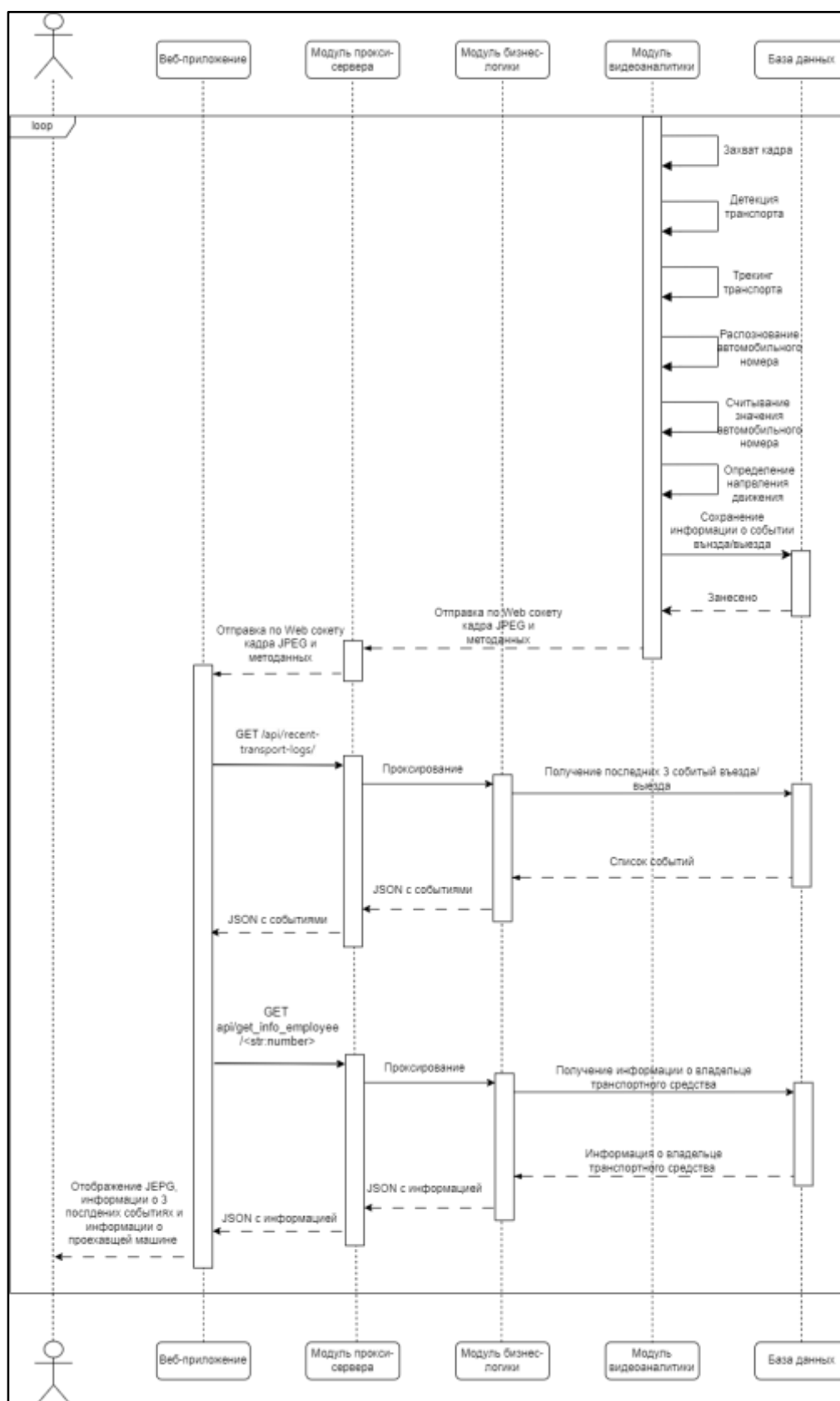
Последовательность вызова и работы модулей при различных вариантах использования представлены на рисунках 8 -10.



Рисунки 8 – Диаграмма последовательности авторизации



Рисунки 9 – Диаграмма последовательности отображение страницы камеры



Рисунки 10 – Диаграмма последовательности обработки видеопотока

1.6.3 Проектирование БД

Для хранения информации о транспорте, сотрудниках и логах необходимо реализовать базу данных со следующими таблицами.

Таблица Camera, содержащая информацию о линиях для определения направления движения транспорта, она содержит в себе поля, представленные в таблице 2.1.

Таблица 2.1 – Поля таблицы Camera

Поле	Тип данных	Описание
id	PK	Уникальный идентификатор поля
url	varchar(500)	Ссылка на видеопоток камеры
detection_type	varchar(20)	Тип детекции
name	varchar(100)	Название камеры
coordinates	varchar(500)	Координаты расположения камеры
is_active	boolean	Работает камера или нет

Таблица CamersLine, содержащая список доступных камер, она содержит в себе поля, представленные в таблице 2.2.

Таблица 2.2 – Поля таблицы CamersLine

Поле	Тип данных	Описание
id	PK	Уникальный идентификатор линии
name	varchar(150)	Название линии
camera_id	integer (FK)	Ссылка на камеру
points	JSON	Координаты точек линии
direction	varchar(10)	Направление контроля
is_active	boolean	Отображать линию или нет
color	varchar(20)	Цвет линии
description	text	Описание линии
created_at	datetime	Дата и время создания
updated_at	datetime	Дата и время последнего изменения

Таблица Employee, содержащая информацию о сотрудниках, она содержит в себе поля, представленные в таблице 2.3.

Таблица 2.3 – Поля таблицы Employee

Поле	Тип данных	Описание
id	PK	Уникальный идентификатор сотрудника
name	varchar(100)	Имя
surname	varchar(100)	Фамилия
company	varchar(200)	Название компании

Таблица Vehicle, содержащая информацию о транспорте, она содержит в себе поля, представленные в таблице 2.4.

Таблица 2.4 – Поля таблицы Vehicle

Поле	Тип данных	Описание
id	PK	Уникальный идентификатор транспортного средства
number	varchar(20)	Государственный регистрационный знак
brand	varchar(100)	Марка автомобиля
employee_id	integer (FK)	Ссылка на элемент таблицы <i>employee</i>

Таблица VehicleEvent, содержащая информацию событиях въезда и выезда, она содержит в себе поля, представленные в таблице 2.5.

Таблица 2.5 – Поля таблицы VehicleEvent

Поле	Тип данных	Описание
id	PK	Уникальный идентификатор события
number	varchar(20)	Распознанный номер автомобиля
vehicle_id	integer (FK)	Ссылка на элемент таблицы <i>vehicle</i>

Поле	Тип данных	Описание
employee_id	integer (FK)	Ссылка на элемент таблицы employee
arrival_time	datetime	Время въезда
departure_time	datetime	Время выезда
stay_duration	DurationField	Продолжительность пребывания
arrival_photo	varchar(500)	Ссылка на фотографию въезда
departure_photo	varchar(500)	Ссылка на фотографию выезда

Для наглядного понимания архитектуры БД, на рисунке 11 представлена схема модели базы данных.

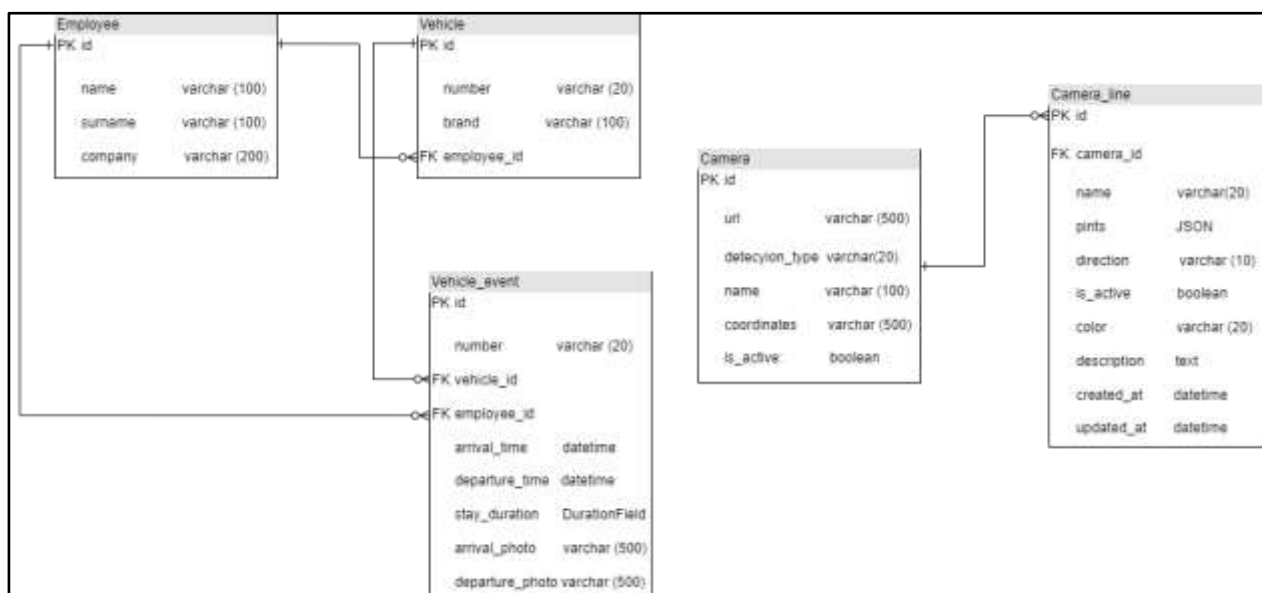


Рисунок 11 – Схема модели базы данных

1.7 Выбор и разработка моделей

1.7.1 Выбор модели для распознавания автотранспорта

Для распознавания автотранспорта в видеопотоке было решено использовать пред обученную модель YOLO, изучив документацию [16] и посмотрев сравнение моделей разных серий, было решено использовать YOLO 8 серии, потому что она имеет широкую поддержку сообщества и экосистемы.

Среди выбранной серии было произведено сравнение разных моделей
таблица 3.

Таблица 3 – Сравнение моделей

Модель	Параметры (М)	mAP50-95	Скорость CPU ONNX (мс)	FLOPs (Б)
YOLOv8n	3.2	37.3	80.4	0.99
YOLOv8s	11.2	44.9	128.4	1.20
YOLOv8m	25.9	50.2	234.7	1.83
YOLOv8l	43.7	52.9	375.2	2.39
YOLOv8x	68.2	53.9	479.1	3.53

Для детекции и трекинга транспортных средств была выбрана модель YOLOv8m. Варианты YOLOv8n и YOLOv8s при меньших требованиях к ресурсам демонстрируют существенно более низкую точность. Модели YOLOv8l и YOLOv8x незначительно превосходят YOLOv8m по точности, однако требуют значительно больше вычислительных ресурсов. Таким образом, YOLOv8m обеспечивает оптимальный баланс между точностью и скоростью инференса, что критично при обработке видеопотока в реальном времени.

1.7.2 Разработка модели для выделения области номерного знака

В качестве базы для переобучения была выбрана модель YOLOv8n. Этот вариант обоснован по следующим причинам: входное изображение уже предварительно обрезано и содержит единственный объект детекции, задача проще по сравнению с детекцией транспорта на общем плане, а YOLOv8n обеспечивает высокую скорость, достаточную точность для специализированного одноклассового датасета.

Для переобучения за основу был взят готовый датасет [17], который содержит изображения автомобилей с российскими государственными

регистрационными знаками, потом он был дополнен фотографиями с камеры на территории офиса.

Для обучения было создано два файла. Train_plate.py, содержит основные настройки для обучения модели. Ниже приведен пример кода.

```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')
results = model.train(
    data='yolo_plate_dataset/dataset_config.yaml',
    epochs=100,
    imgsz=640,
    batch=8,
    name='yolov8n_plate_detector',
    workers=4,
    device=0,
    patience=50,
    save=True,
    exist_ok=True
)
print(" Обучение завершено!")
print(f" Модель: best.pt")
```

Dataset_config.yaml, задаёт один класс — license_plate — и определяет пути к тренировочной, валидационной и тестовой выборкам. Обучение проводилось в течение 100 эпох. Ниже приведен пример кода.

```
path: yolo_plate_dataset
train: train/images
val: val/images
test: test/images
nc: 1
names: ['license_plate']
```

На рисунке 12 представлены сводные графики обучения модели YOLOv8n на протяжении 100 эпох

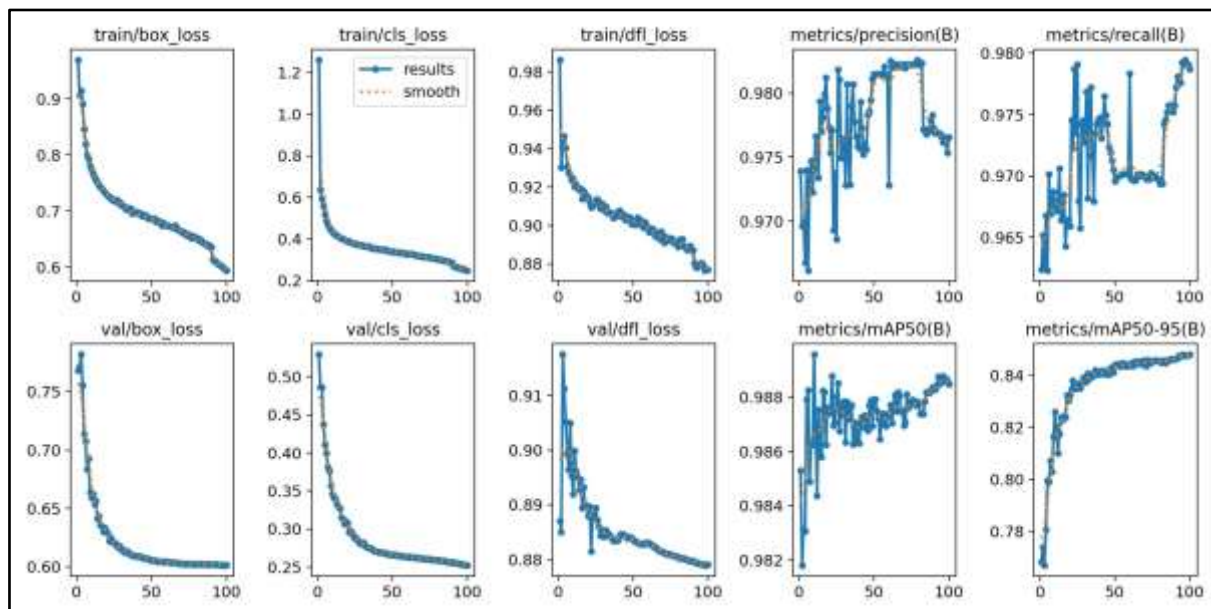


Рисунок 12 - Графики потерь и метрик

На рисунке 13 показана кривая Precision-Recall, которая характеризует соотношение точности и полноты при различных порогах уверенности модели. Площадь под кривой составляет 0.988. Практически прямоугольная форма кривой означает, что модель сохраняет точность, близкую к единице, при полноте вплоть до 0.97, что подтверждает высокое качество детекции номерных знаков.

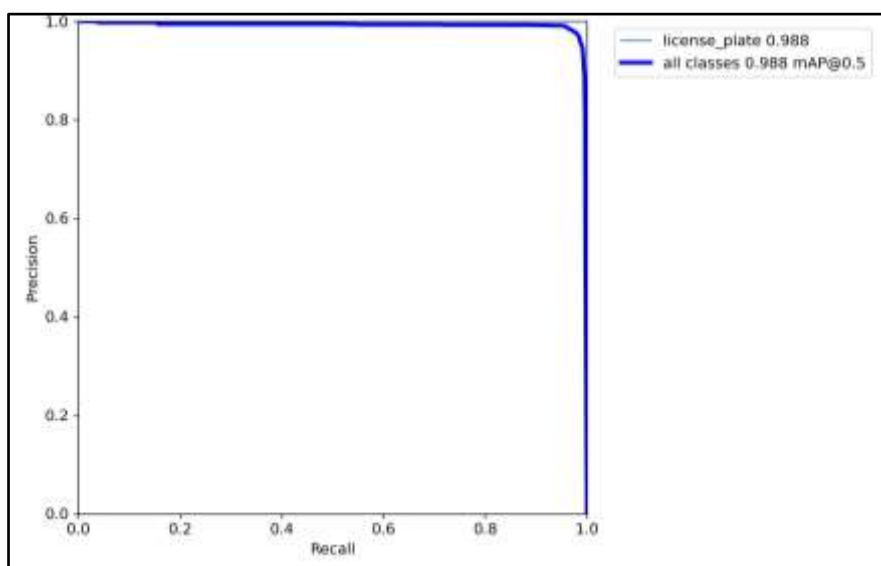


Рисунок 13 – Кривая Precision-Recall

На рисунке 14 приведена F1-Confidence кривая, отображающая зависимость F1-меры от порога уверенности. Максимальное значение $F1 = 0.98$ достигается при $\text{confidence} = 0.482$, что определяет оптимальный порог срабатывания модели в продуктивном режиме.

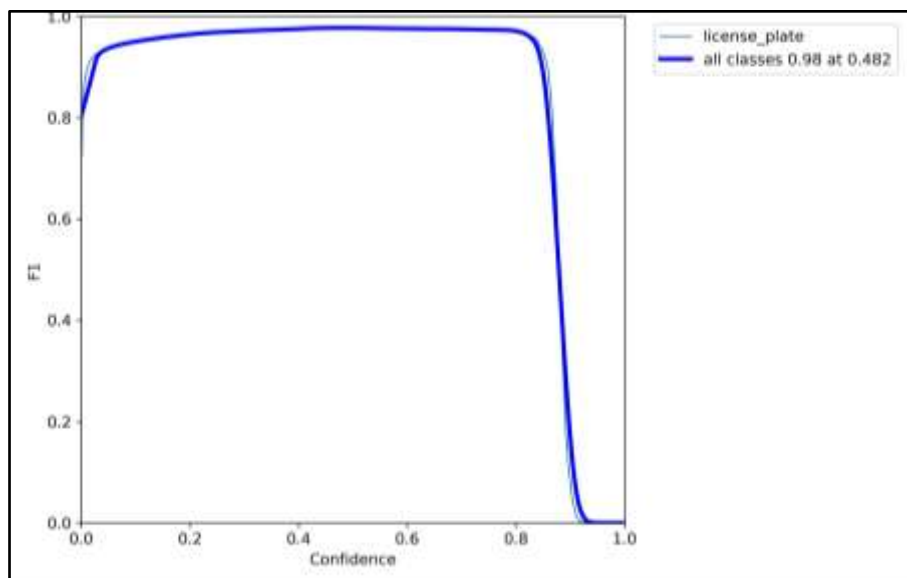


Рисунок 14 – F1-Confidence кривая

На рисунке 15 представлена нормализованная матрица ошибок. Модель правильно классифицирует 99% объектов класса `license_plate`, допуская лишь 1% пропущенных срабатываний. Ложные срабатывания на фоне отсутствуют, что подтверждает высокую специализацию модели к задаче обнаружения номерных знаков.

Достигнутые показатели подтверждают, что переобученная модель YOLOv8n обеспечивает высокую точность детекции государственных регистрационных знаков.

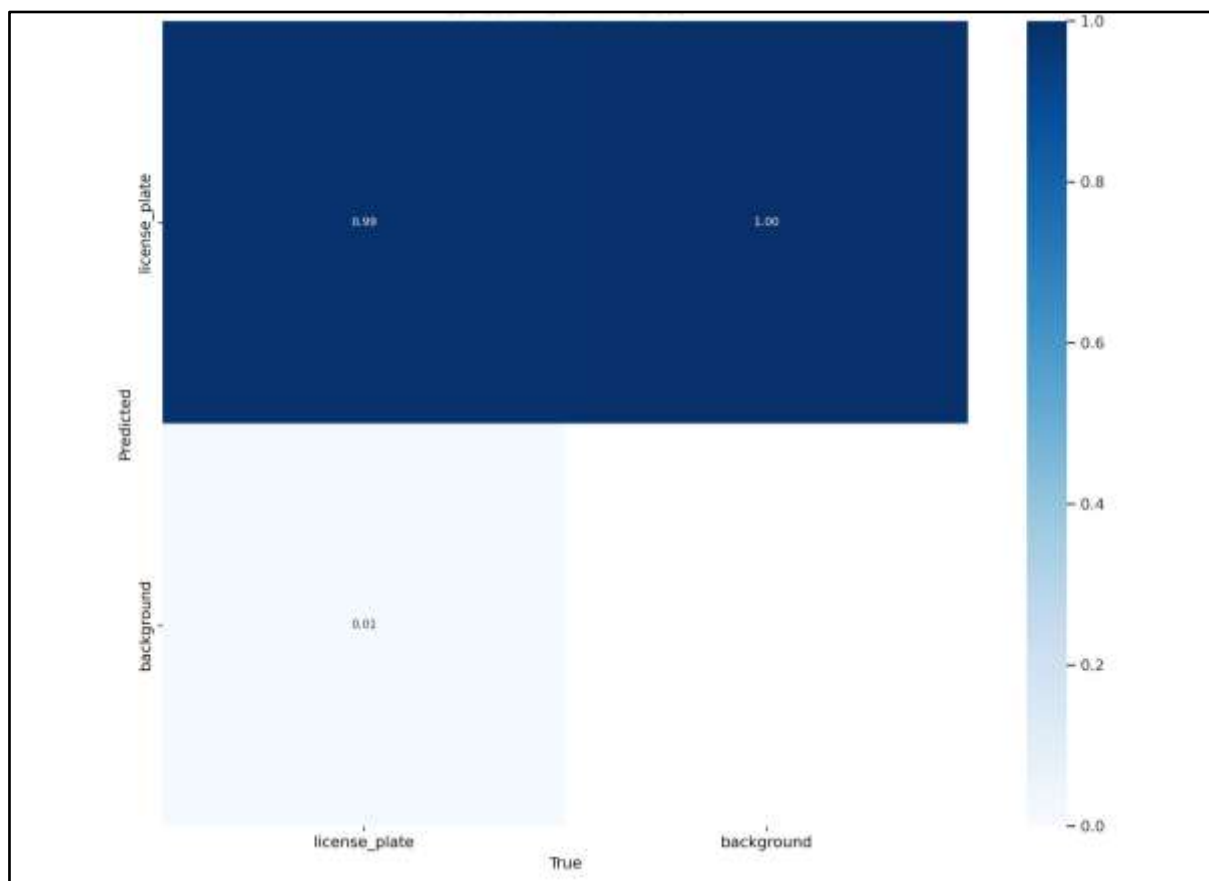


Рисунок 15 – Нормализованная матрица ошибок

1.7.3 Разработка модели для считывания значения номерного знака

Для считывания значения номерного знака было решено обучить кастомную CRNN модель. CNN-часть извлекает из изображения низкоуровневые признаки. RNN-часть анализирует эти признаки как последовательность, понимая контекст символов.

Чтобы обучить модель был взят готовый датасет [18], который содержит изображения государственных регистрационных знаков России, потом он был дополнен фотографиями с камеры на территории офиса.

Далее был написан код для описания модели и её обучения. Ниже приведен пример кода.

```
class CRNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.cnn = nn.Sequential(
```

```

        nn.Conv2d(1, 64, 3, padding=1), nn.ReLU(True),
nn.MaxPool2d(2, 2),
        nn.Conv2d(64, 128, 3, padding=1), nn.ReLU(True),
nn.MaxPool2d(2, 2),
        nn.Conv2d(128, 256, 3, padding=1),
nn.BatchNorm2d(256), nn.ReLU(True),
        nn.Conv2d(256, 256, 3, padding=1),
nn.ReLU(True), nn.MaxPool2d((2, 1), (2, 1)),
        nn.Conv2d(256, 512, 3, padding=1),
nn.BatchNorm2d(512), nn.ReLU(True),
        nn.Conv2d(512, 512, 3, padding=1),
nn.ReLU(True), nn.MaxPool2d((2, 1), (2, 1))
    )
    self.rnn = nn.LSTM(1024, 256, bidirectional=True,
num_layers=2, batch_first=True)
    self.classifier = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.cnn(x)
        b, c, h, w = x.shape
        x = x.reshape(b, c * h, w).permute(0, 2, 1)
        x, _ = self.rnn(x)
        return x.permute(1, 0, 2).log_softmax(2)

class OCRDataset(Dataset):
    def __init__(self, root, alphabet):
        self.samples = []
        img_dir = os.path.join(root, "img")

        for fname in os.listdir(img_dir):
            if fname.endswith('.png'):
                name = fname.rsplit('.', 1)[0]
                if '_' in name:
                    name = name.rsplit('_', 1)[0]
                self.samples.append((os.path.join(img_dir,
fname), name))

        self.alphabet = '-' + alphabet
        self.c2i = {c: i for i, c in
enumerate(self.alphabet)}
        print(f"Loaded {len(self.samples)} samples from
{root}")

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        path, text = self.samples[idx]
        img = Image.open(path).convert('L').resize((128,
32))

```

```

        img = torch.tensor(np.array(img) / 255.,
dtype=torch.float32).unsqueeze(0)
        # Оставляем только символы из алфавита
        text = ''.join([c for c in text if c in self.c2i])
        if len(text) == 0:
            text = '0'
        label = torch.tensor([self.c2i[c] for c in text])
        return img, label

def collate(batch):
    imgs, labels = zip(*batch)
    return torch.stack(imgs), torch.cat(labels),
torch.tensor([len(l) for l in labels])

alphabet = "0123456789АВЕКМНОРСТУХ"
train_ds = OCRDataset("train", alphabet)
val_ds = OCRDataset("val", alphabet)
train_dl = DataLoader(train_ds, batch_size=32, shuffle=True,
collate_fn=collate)
val_dl = DataLoader(val_ds, batch_size=32, shuffle=False,
collate_fn=collate)

device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
model = CRNN(len(train_ds.alphabet)).to(device)
opt = torch.optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.CTCLoss(blank=0, zero_infinity=True)

# Обучение
for epoch in range(50):
    model.train()
    total_loss = 0
    for imgs, labels, lens in train_dl:
        imgs, labels, lens = imgs.to(device),
labels.to(device), lens.to(device)
        out = model(imgs)
        loss = loss_fn(out, labels,
torch.full((imgs.size(0),), out.size(0),
dtype=torch.long).to(device), lens)
        opt.zero_grad()
        loss.backward()
        opt.step()
        total_loss += loss.item()

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for imgs, labels, lens in val_dl:
            imgs, labels, lens = imgs.to(device),
labels.to(device), lens.to(device)
            out = model(imgs)

```

```

        val_loss += loss_fn(out, labels,
torch.full((imgs.size(0),), out.size(0),
dtype=torch.long).to(device), lens).item()

```

После 7 эпох модель показала очень хорошие результаты. Точность модели составило 99% рисунок 16.

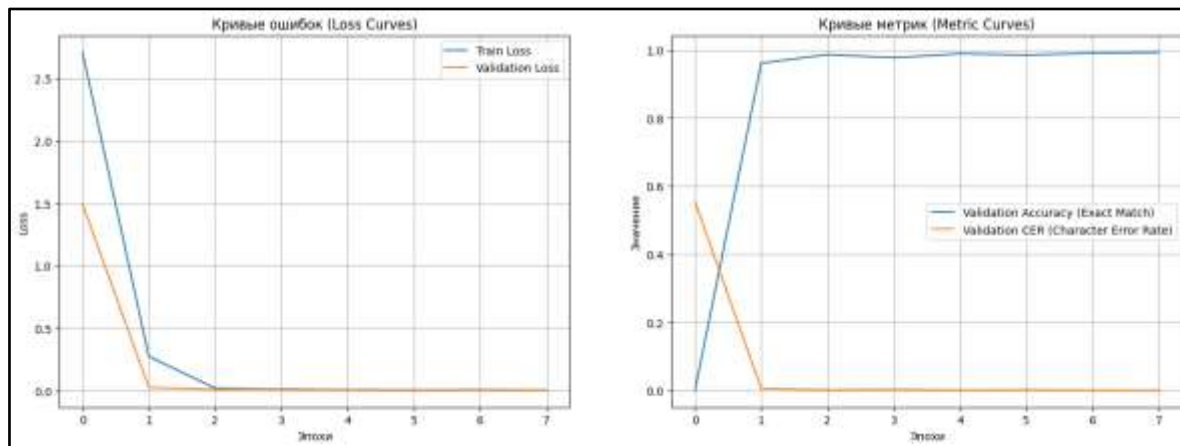


Рисунок 16 – Кривые обучения модели

Матрица ошибок показывает не частые неточности при распознавании похожих символов рисунок 17.

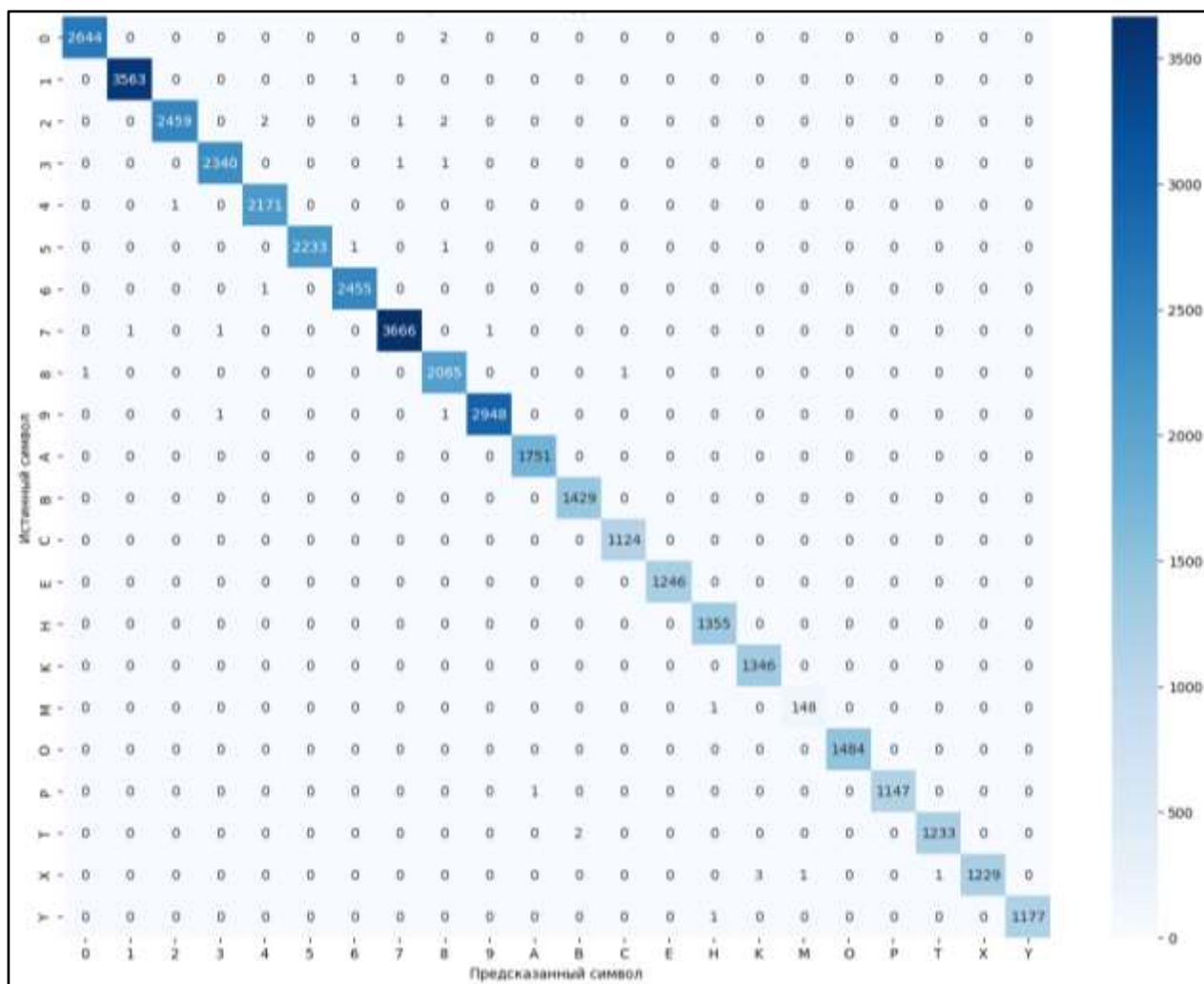


Рисунок 17 – Матрица ошибок

1.8 Разработка программы

1.8.1 Описание программы

Программная часть системы разделена на два независимых компонента: веб-приложение на Django и сервер видеоаналитики на Python. Такое разделение позволяет каждому компоненту работать отдельно, не зависимо друг от друга.

1.8.1.1 Модуль веб-приложение на Django

Django отвечает за связь с веб-интерфейсом и API. Она построена по модульной схеме, где каждый файл выполняет свою задачу.

Файл models.py содержит описание структуры базы данных в виде Python-классов. Каждый класс — это таблица в PostgreSQL, а его атрибуты — поля этой таблицы. Django сам создаёт миграции и поддерживает схему базы в актуальном состоянии.

Класс Vehicle описывает таблицу, в которой хранятся данные о транспортных средствах. Также этот класс реализует функцию save, которая конвертирует русские буквы в их латинские аналоги. Ниже приведен пример кода.

```
class Vehicle(models.Model):
    number = models.CharField(max_length=20,
verbose_name="Номер", unique=True)
    brand = models.CharField(max_length=100,
verbose_name="Марка")
    employee = models.ForeignKey(
        Employee,
        on_delete=models.CASCADE,
        related_name='vehicles',
        verbose_name="Владелец",
        db_column='employee_id'
    )
    class Meta:
        db_table = 'vehicle'
        verbose_name = 'Транспортное средство'
        verbose_name_plural = 'Транспортные средства'
        ordering = ['number']
        managed = True
    def save(self, *args, **kwargs):
        ru_to_lat = {
            'А': 'A', 'В': 'B', 'Е': 'E', 'К': 'K', 'М': 'M',
            'Н': 'H', 'О': 'O', 'Р': 'P', 'С': 'C', 'Т': 'T',
            'У': 'Y', 'Х': 'X',
        }
        if self.number:
            self.number = ''.join(ru_to_lat.get(c, c) for c
in self.number).upper()
            super().save(*args, **kwargs)
    def str(self):
        return f"{self.number} - {self.brand} (Владелец:
{self.employee})"
```

Класс Employee описывает таблицу, которая хранит данные о сотрудниках. Ниже приведен пример кода.

```
class Employee(models.Model):
```

```

        name = models.CharField(max_length=100,
verbose_name="Имя")
        surname = models.CharField (max_length=100, verbose_name
="Фамилия")
        company = models.CharField(max_length=200,  verbose_name
="Компания")
        class Meta:
            db_table = 'employee'
            verbose_name = 'Сотрудник'
            verbose_name_plural = 'Сотрудники'
            ordering = ['surname', 'name']
            managed = True
        def __str__(self):
            return f"{self.surname} {self.name}
({self.company}) "

```

Класс VehicleEvent описывает таблицу, которая хранит данные о событиях транспорта.

Класс CameraLine описывает таблицу, которая хранит данные о линиях для определения направления движения транспорта.

Класс Camera описывает таблицу, которая хранит информацию о камерах.

Также, каждый из вышеперечисленных классов содержит подкласс Meta, который описывает как будет называться таблица в БД и какие названия у этих таблиц будет в панели администратора.

Файл serializers.py преобразует объекты в JSON и обратно. Этот файл описывает те же таблицы, что указаны в файле models.py. Ниже приведен пример кода.

```

class VehicleSerializer(serializers.ModelSerializer):
    class Meta:
        model = Vehicle
        fields = ['id', 'number', 'brand', 'employee']

```

Файл views.py описывает контроллеры, которые обрабатывают HTTP-запросы. Каждый класс и функция соответствуют определённому эндпоинту API.

Класс TransportLogs возвращает несколько записей из таблицы VehicleEvent. Ниже приведен пример кода.

```

class TransportLogs(viewsets.ReadOnlyModelViewSet):
    serializer_class = VehicleEventSerializer
    def get_queryset(self):
        queryset = VehicleEvent.objects.all().order_by('-
id')
        camera_id =
self.request.query_params.get('camera_id')
        if camera_id:
            queryset = queryset.filter(camera_id=camera_id)
        return queryset
    def list(self, request, *args, **kwargs):
        queryset = self.get_queryset()
        page = int(request.query_params.get('page', 1))
        per_page = int(request.query_params.get('per_page',
10))

        total_count = queryset.count()
        total_pages = max(1, (total_count + per_page - 1) //
per_page)

        start = (page - 1) * per_page
        end = start + per_page
        serializer =
self.get_serializer(queryset[start:end], many=True)
        return Response({
            'count': total_count,
            'page': page,
            'per_page': per_page,
            'total_pages': total_pages,
            'results': serializer.data
        })

```

Функция InfoEmployeeView ищет транспортное средство по гос. номеру и возвращает информацию о нём и его владельце. Ниже приведен пример кода.

```

def InfoEmployeeView(request, number):
    try:
        vehicle =
Vehicle.objects.select_related('employee').get(number=number)
        vehicle_serializer = VehicleSerializer(vehicle)
        employee_serializer =
EmployeeSerializer(vehicle.employee)
        return Response({
            'vehicle': vehicle_serializer.data,
            'employee': employee_serializer.data
        })
    except Vehicle.DoesNotExist:
        return Response(
            {'error': f'Транспортное средство с номером
{number} не найдено'},
            status=404
        )

```

Файл `urls.py` связывает URL-адреса с обработчиками из `views.py`. Это карта маршрутов: когда приходит запрос на определённый адрес, Django по этой карте понимает, какой обработчик вызывать.

Файл `admin.py` настраивает встроенную админ-панель Django. Это готовый интерфейс для управления данными, который не нужно писать с нуля — достаточно описать, какие поля показывать в списке, по каким разрешить поиск и фильтрацию.

Класс `CameraAdmin`, настраивает админ-панель для загрузки информации о камерах. Ниже приведен пример кода.

```
@admin.register(Camera)
class CameraAdmin(admin.ModelAdmin):
    """Админка камер"""
    list_display = ('id', 'name', 'url', 'detection_type',
'is_active', 'coordinates')
    list_filter = ('detection_type', 'is_active')
    search_fields = ('name', 'url', 'coordinates')
    list_editable = ('is_active',)
    fields = ('name', 'url', 'detection_type',
'coordinates', 'is_active')
    inlines = [CameraZoneInline, CameraLineInline]
    def get_queryset(self, request):
        qs = super().get_queryset(request)
        user = request.user
        return qs
```

1.8.1.2 Модуль для обработки видеопотока

Программный комплекс построен на основе асинхронного веб-сервера с использованием библиотек `eventlet`, `socketio` и `OpenCV`. Ядром системы является класс `VideoStreamManager`, который управляет пулом подключений к базе данных, загружает нейросетевые модели и инициализирует отдельные потоки-обработчики для каждой IP-камеры. Для обеспечения высокой производительности некоторые операции выносятся в пул потоков `ThreadPoolExecutor`.

Инициализация менеджера начинается с создания пула соединений с СУБД PostgreSQL для хранения логов распознавания и событий проезда.

Параметры подключения берутся из файла конфигурации окружения с помощью библиотеки `dotenv`. Ниже приведен пример кода.

```
def init_db_pool(self):
    try:
        import psycopg2.pool
        self.db_pool = psycopg2.pool.SimpleConnectionPool(
            1, 3,
            host=os.getenv('DB_HOST', 'db'),
            database=os.getenv('DB_NAME', 'vadb'),
            # ...
        )
    except Exception as e:
        sys.exit(1)
```

Загрузка моделей YOLO реализована в методе `load_model`. Для детекции автомобилей используется предобученная модель `yolov8m.pt`, а для обнаружения номерных пластин — кастомная модель `best.pt`. Обе модели переносятся на графический ускоритель для инференса в реальном времени.

За распознавание текста номерного знака отвечает класс `CRNN`, архитектура которого объединяет сверточные слои (CNN) и рекуррентную сеть (LSTM). CNN-часть экстрагирует пространственные признаки из изображения размером 128x32 пикселя, последовательно уменьшая высоту карты признаков до двух пикселей. Это позволяет трансформировать графическое представление номера в последовательность векторов, пригодную для обработки рекуррентным слоем. Ниже приведен пример кода.

```
class CRNN(nn.Module):
    def __init__(self, num_classes):
        super(CRNN, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            # ...
            nn.MaxPool2d((2, 1), (2, 1)) # Итоговая высота: 2
        )
        self.rnn = nn.LSTM(512 * 2, 256, bidirectional=True,
num_layers=2)
        self.classifier = nn.Linear(512, num_classes)
```

Менеджер распознавания LicensePlateOCR загружает веса модели crnn_ocr_model_best.pth, производит адаптацию ключей state_dict под текущую структуру, и переводит модель в режим оценки. Предобработка входного изображения включает в себя преобразование в оттенки серого, масштабирование и нормализацию. Для корректной работы с российскими форматами номеров используется алфавит '0123456789АВСЕНКМОРТХУ', а декодирование выхода сети выполняется с учетом СТС-логики: метод decode_prediction убирает дублирующиеся символы и blank-токены, разделяющие повторяющиеся знаки.

Ключевым модулем контроля доступа является встроенная в класс CameraStreamer система работы с виртуальными линиями. Данные о линиях извлекаются из базы данных при инициализации камеры. Для каждого обнаруженного автомобиля ведется история перемещения центраида его Bounding Box. Метод update_vehicle_position сохраняет последние координаты и временные метки в буфер для сглаживания траектории. Ниже приведен пример кода.

```
def update_vehicle_position(self, track_id, centroid):
    if track_id not in self.vehicle_positions:
        self.vehicle_positions[track_id] = []
    self.vehicle_positions[track_id].append((centroid[0],
centroid[1], time.time()))
```

Функция check_line_crossing реализует математический метод для определения факта пересечения и стороны, с которой произошло движение. Для этого используется векторное произведение point_side_of_line и проверка пересечения segments_intersect. Если автомобиль пересек линию и направление соответствует заданной конфигурации, событие фиксируется, а треку присваивается статус enter или exit с защитой от повторных срабатываний (cooldown).

Модуль recognize_license_plate решает проблему нестабильности единичных кадров внедрением механизма накопления и голосования. Для каждого уникального идентификатора трека создается накопитель

raw_recognitions. В него добавляются все первичные результаты OCR, полученные с разной степенью уверенности, пока автомобиль находится в кадре. Процесс голосования запускается, когда накоплено не менее трех кадров. Используя класс Counter, система определяет наиболее часто встречающийся текст номера и вычисляет среднюю уверенность среди идентичных гипотез. Если частота доминирующей гипотезы превышает порог в 40%, а итоговая уверенность выше 0.3, результат считается финальным, кэшируется и отправляется на сохранение в базу данных, после чего накопитель очищается. Ниже приведен пример кода.

```
if acc['frames'] >= 3:
    text_counts = Counter(acc['texts'])
    most_common_text, count = text_counts.most_common(1)[0]
    # ...
    if count >= max(2, len(acc['texts']) * 0.4) and
(avg_confidence > 0.3):
        self.plate_cache[final_cache_key] =
(most_common_text, avg_confidence, current_time)
```

Сохранение распознанного номера и направления движения в СУБД реализовано в методе save_plate_to_db. Логика метода разделена по типу события. При въезде создается новая запись в таблице vehicle_event, связывающая транспортное средство и время прибытия. При выезде выполняется поиск открытой записи о въезде, время которой не превышает заданный лимит (24 часа). Помимо точного совпадения строк, реализован нечеткий поиск на основе расстояния Левенштейна. Если точного совпадения не найдено, система фильтрует все активные события, подходящие по длине строки, и сопоставляет распознанный номер с теми, у кого редакционное расстояние меньше двух. Это нивелирует мелкие ошибки OCR. Ниже приведен пример кода.

```
def _levenshtein_distance(self, s1: str, s2: str) -> int:
    if len(s1) < len(s2): return
self._levenshtein_distance(s2, s1)
    # ... динамическое программирование вычисления
редакционного расстояния
    for i, c1 in enumerate(s1):
        curr_row = [i + 1]
```

```

        for j, c2 in enumerate(s2):
            insertions = prev_row[j + 1] + 1
            deletions = curr_row[j] + 1
            substitutions = prev_row[j] + (c1 != c2)
            curr_row.append(min(insertions, deletions,
substitutions))
        prev_row = curr_row
    return prev_row[-1]

```

При успешном закрытии события вычисляется длительность пребывания транспортного средства на территории, а к записи прикрепляется путь к сохраненному полноразмерному кадру, фиксирующему лучший результат распознавания с наибольшей уверенностью.

Для захвата и декодирования RTSP потока используется GStreamer, интегрированный в OpenCV. В основном цикле обработки камеры, реализованном в методе run класса CameraStreamer, конфигурируется специализированный конвейер, обеспечивающий низкую латентность и стабильность приема H.265 потоков. Ниже приведен пример кода.

```

GST_LATENCY = int(os.getenv('GST_LATENCY', '100'))
gst_pipeline = (
    f'rtspsrc location={self.rtsp_url} latency={GST_LATENCY}
,
    f'drop-on-latency=true config-interval=1 protocols=tcp '
    f'! rtph265depay ! h265parse config-interval=-1 '
    f'! avdec_h265 ! videoconvert '
    f'! video/x-raw,format=BGR '
    f'! appsink sync=false max-buffers=1 drop=true'
)
cap = cv2.VideoCapture(gst_pipeline, cv2.CAP_GSTREAMER)

```

Формируемый конвейер GStreamer начинается с элемента rtspsrc, который устанавливает TCP-соединение с камерой, запрашивает поток и реализует механизм контроля задержек. Латентность ограничена параметром latency, а директива drop-on-latency=true гарантирует отбрасывание устаревших пакетов, позволяя избежать накопления очереди. Далее звено rtph265depay извлекает сжатые данные из RTP-пакетов, а h265parse преобразует их в байтовый поток. Установка config-interval=-1 требует вставки SPS/PPS заголовков перед каждым ключевым кадром, что обеспечивает мгновенный

старт декодирования без ожидания следующего IDR-кадра. Аппаратное или программное декодирование производится элементом `avdec_h265`, преобразующим сжатый фрейм в несжатые пиксели. Последний этап конвейера — фильтр `videoconvert`, приводящий цветовое пространство к формату BGR, и `appsink`, который с настройками `sync=false` и `max-buffers=1` немедленно передает готовый кадр в цикл обработки без блокировок, игнорируя временные метки потока.

Отображение обработанного видеопотока с аннотациями осуществляется через протокол WebSocket. Асинхронный сервер Socket.IO генерирует события для каждой камеры. Клиенты могут динамически подписываться на конкретную камеру, отправляя событие `subscribe` с идентификатором. Сервер кодирует каждый обработанный в `CameraStreamer.process_frame` кадр в JPEG, сериализует его в Base64 и рассылает по комнатам подписчиков вместе с метаданными. Ниже приведен пример кода.

```
_, buffer = cv2.imencode('.jpg', original_frame,
encode_params)
frame_base64 = base64.b64encode(buffer).decode('utf-8')
sio.emit(f'camera_{self.camera_id}_frame', {
    'frame': frame_base64,
    'recognized_plates': recognized_plates,
    # ...
}, room=f'camera_{self.camera_id}')
```

Таким образом, серверная часть представляет собой модульное решение, объединяющее двухэтапный детектор (YOLO), рекуррентный оптический распознаватель (CRNN), геометрический анализатор траекторий и отказоустойчивый слой взаимодействия с реляционной базой данных, что обеспечивает полный цикл автоматической фиксации проездов без участия оператора.

1.8.2 Разработка графического интерфейса программы

Frontend приложение реализовано на языке JavaScript с использованием библиотеки React и работает в браузере пользователя. Оно дает возможность просматривать данные, поступающие от сервера, и предоставляет интерфейс

для взаимодействия с системой. Модуль отображает видеопотоки с камер, логи и информацию о пользователях.

1.9 Тестирование

Тестирование является неотъемлемой частью разработки программного обеспечения. Оно позволяет выявить, в каких сценариях система работает правильно, а где присутствуют ошибки, требующие доработки.

Существует несколько видов тестирования, различавшиеся по уровню доступа к системе. При тестировании «чёрного ящика» тестировщик оценивает работу системы, не зная её внутренней структуры. При таком подходе проверяется соответствие функциональности и задания. Противоположный подход — тестирование «белого ящика». Этот подход подразумевает знание внутренней архитектуры и кода. Проверяются отдельные модули и алгоритмы, выявляются потенциальные уязвимости и неэффективные участки кода.

Также тестирование разделяется на ручное и автоматизированное. В рамках ручного тестирования специалист самостоятельно выполняет тестовые сценарии, имитируя действия конечного пользователя и фиксируя результаты. Автоматизированное тестирование, напротив, опирается на заранее написанные скрипты, которые автоматически прогоняют набор проверок.

Так как основной задачей является проверка удобства и корректности работы пользовательского интерфейса, представленного сайтом, был выбран ручной вид тестирования. Он позволяет наиболее полноценно оценить визуальные и интерактивные аспекты системы.

Система видеоаналитики реализует следующие функции для удобного взаимодействия с данными:

- авторизация;
- просмотр журнала событий;
- поиск пользователей;
- поиск транспорта по государственному регистрационному номеру;

- просмотр камер;
- отображение информации о последнем проехавшем автомобиле;
- добавление пользователей;
- добавление новых автомобилей.

Для проверки корректности работы заявленного функционала было проведено ручное тестирование по заранее подготовленному набору сценариев таблица 4.

Таблица 4 – Тестирование функционала приложения.

ID	Назначение теста	Входные данные/действия	Ожидаемый результат	Фактический результат
1	Проверка успешного входа по логину и паролю	Открыт сайт [19], ввести логин и пароль	Успешный вход, открылась главная страница	Открылась главная страница
2	Проверка доступа к странице с камерой	Выполнен вход, находитесь на главной странице, нажатие на камеру	Открытие станицы с камерой	Открылась страница с камерой
3	Просмотр фотографии въезда/выезда	Выполнен вход, нахождение на страницы камеры, нажатие на время въезда/выезда	Открытие фотографии въезда/выезда	Открылась фотография въезда
4	Проверка отображения информации о последней проехавшей машине	Выполнен вход, нахождение на страницы камеры, дожидаться проезда машины	Появление информации о проехавшей машине	Появилась информация о проехавшей машине

ID	Назначение теста	Входные данные/действия	Ожидаемый результат	Фактический результат
5	Проверка поисковой системы по номеру	Выполнен вход, нахождение на страницы камеры, имеется номер для поиска	Вывод информация о владельце машины	Вывелась информация о владельце машины
6	Проверка поисковой системы по владельцу	Выполнен вход, нахождение на страницы камеры, имеется ФИО владельца	Вывод информация о машинах владельца	Вывелась информация о машинах владельца
7	Проверка таблицы просмотра событий транспорта	Выполнен вход, нахождение на страницы камеры, нажата кнопка «Открыть все логи»	Появление таблицы «Логи автомобилей»	Вывелась таблица «Логи автомобилей»
8	Проверка поисковой системы по таблице «Логи автомобилей»	Выполнен вход, открыта таблица «Логи автомобилей», установка параметров для поиска	Вывод логов, удовлетворяющих условия поиска	Вывелись логи удовлетворяющие условия поиска
9	Проверка добавления новых пользователей	Выполнен вход, открыта панель администратора, открыта страница для добавления пользователей, введены данные фамилия, имя и компания, в которой работает пользователь	Добавление нового пользователя	Добавился новый пользователь

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

56

ID	Назначение теста	Входные данные/действия	Ожидаемый результат	Фактический результат
10	Проверка добавления нового транспорта	Выполнен вход, открыта панель администратора, открыта страница для добавления транспорта, введены номер, марка транспорта и владелец	Добавление нового транспорта	Добавился новый транспорт

1.10 Развёртывание приложения

Для развёртывания приложения было решено использовать Docker container рисунок 18. Он позволит удобно переносить и скачивать все необходимые библиотеки для работы.

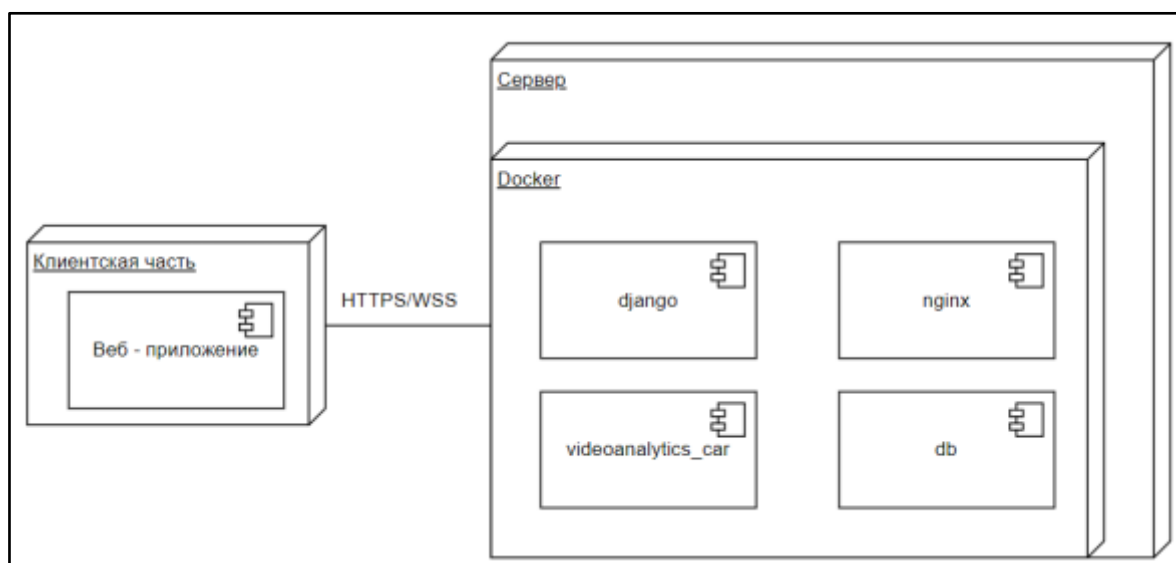


Рисунок 18 – UML-диаграмма развёртывания

Серверная часть приложения будет развёрнута на Ubuntu. Для установки на неё Docker container необходимо прописать последовательность команд в консоли:

- sudo apt update;
- sudo apt install apt-transport-https ca-certificates curl software-properties-common;
- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.asc;
- sudo chmod a+r /etc/apt/keyrings/docker.asc;
- echo "deb [arch=\$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \$(. /etc/os-release && echo "\${UBUNTU_CODENAME:-\$VERSION_CODENAME}") stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null;
- sudo apt update;
- sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin.

Далее для установки необходимых зависимостей, необходимо создать и прописать requirements.txt, в котором описываются библиотеки и их версии рисунки 19, 20.

```
ultralytics==8.3.75
python-dotenv==1.0.1
numpy==1.26.4

pandas==2.2.0
tqdm==4.66.0
loguru==0.7.2

psycpg2-binary==2.9.9

python-socketio==5.9.0
eventlet==0.33.3
python-engineio>=4.7.0
python-jose==3.3.0
```

Рисунок 19 – Пример requirements.txt для контейнера с видеоаналитикой

```
Django==5.2.7
djangorestframework==3.14.0
gunicorn==21.2.0
psycpg2==2.9.7
whitenoise==6.6.0
django-cors-headers==4.3.1
matplotlib==3.7.2
dj-database-url==2.1.0
djangorestframework-simplejwt>=5.3.0
```

Рисунок 20 – Пример requirements.txt для контейнера с серверной частью

После прописания версий библиотек необходимо прописать Dockerfile, который описывает последовательность команд для установки библиотек и настройки окружения рисунки 21 - 23.

```
FROM python:3.11
WORKDIR /app
RUN apt-get update && apt-get install -y \
    gcc \
    postgresql-client \
    && rm -rf /var/lib/apt/lists/*
RUN mkdir -p /root/.pip
RUN echo "[global]\nindex-url = https://pypi.tuna.tsinghua.edu.cn/simple/\ntrusted-host = pypi.tuna.tsinghua.edu.cn" > /root/.pip/pip.conf
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
RUN mkdir -p /app/staticfiles /app/media
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Рисунок 21 – Пример Dockerfile для контейнера с серверной частью

```

FROM pytorch/pytorch:2.9.0-cuda12.8-cudnn9-runtime

ENV DEBIAN_FRONTEND=noninteractive
ENV PYTHONUNBUFFERED=1

RUN apt-get update && apt-get install -y \
    g++ \
    build-essential \
    cmake \
    git \
    pkg-config \
    ffmpeg \
    libgl1 \
    libglu1-mesa-dev \
    libsm6 \
    libxext6 \
    libxrender1 \
    libgomp1 \
    libjpeg-dev \
    libpng-dev \
    libtiff-dev \
    libavcodec-dev \
    libavformat-dev \
    libswscale-dev \
    libv4l-dev \
    libxvidcore-dev \
    libx264-dev \
    libgtk-3-dev \
    libatlas-base-dev \
    gfortran \
    python3-dev \
    gstreamer1.0-tools \
    gstreamer1.0-libav \
    gstreamer1.0-plugins-base \
    gstreamer1.0-plugins-good \
    gstreamer1.0-plugins-bad \
    gstreamer1.0-plugins-ugly \
    libgstreamer1.0-dev \
    libgstreamer-plugins-base1.0-dev \
    && rm -rf /var/lib/apt/lists/*

RUN useradd -m -u 1000 -s /bin/bash admin
WORKDIR /home/admin

WORKDIR /opt

RUN git clone --depth 1 https://github.com/opencv/opencv.git
RUN git clone --depth 1 https://github.com/opencv/opencv_contrib.git

WORKDIR /opt/opencv/build

```

Рисунок 22 – Пример Dockerfile для контейнера с видеоаналитикой

```

RUN cmake \
  -D CMAKE_BUILD_TYPE=Release \
  -D CMAKE_INSTALL_PREFIX=/opt/conda \
  -D OPENCV_EXTRA_MODULES_PATH=/opt/opencv_contrib/modules \
  -D WITH_GSTREAMER=ON \
  -D WITH_FFMPEG=ON \
  -D WITH_CUDA=OFF \
  -D BUILD_opencv_python3=ON \
  -D BUILD_TESTS=OFF \
  -D BUILD_EXAMPLES=OFF \
  -D BUILD_DOCS=OFF \
  -D BUILD_PERF_TESTS=OFF \
  -D WITH_GAPI=OFF \
  -D BUILD_opencv_gapi=OFF \
  ..

RUN make -j$(nproc)

RUN make install

RUN ldconfig

RUN python -c "import cv2; print(cv2.getBuildInformation())"

RUN pip install --no-cache-dir \
  ultralytics==8.3.75 \
  python-dotenv==1.0.1 \
  numpy==1.26.4 \
  pandas==2.2.0 \
  tqdm==4.66.0 \
  loguru==0.7.2 \
  psycopg2-binary==2.9.9 \
  python-socketio==5.9.0 \
  eventlet==0.33.3 \
  python-engineio>=4.7.0 \
  python-jose==3.3.0

RUN pip install --no-cache-dir --default-timeout=300 lapx

RUN rm -rf /opt/conda/lib/python3.11/site-packages/cv2/gapi
COPY requirements.txt /home/admin/requirements.txt
COPY best.pt /home/admin/best.pt
COPY yolov8m.pt /home/admin/yolov8m.pt
COPY video_stream_server.py /home/admin/video_stream_server.py
COPY crnn_ocr_model_best.pth /home/admin/crnn_ocr_model_best.pth

RUN usermod -a -G 33 admin

RUN chown -R admin:admin /home/admin

USER admin
WORKDIR /home/admin

CMD ["python", "video_stream_server.py"]

```

Рисунок 23 – Пример Dockerfile для контейнера с видеоаналитикой

После описания Dockerfile необходимо прописать конфигурационный файл docker-compose.yml, который позволяет описать логику запуска контейнеров рисунки 24 - 26.

```

db:
  image: postgres:15-alpine
  volumes:
    - postgres_data:/var/lib/postgresql/data
  ports:
    - "5432:5432"
  environment:
    - POSTGRES_DB=${DB_NAME}
    - POSTGRES_USER=${DB_USER}
    - POSTGRES_PASSWORD=${DB_PASSWORD}
  restart: unless-stopped
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U ${DB_USER} -d ${DB_NAME}"]
    interval: 30s
    timeout: 10s
    retries: 3
  networks:
    - videoanalytics_network

```

Рисунок 24 – Пример кода для запуска контейнера с PostgreSQL

```

yolo:
  build: ./yolo
  depends_on:
    db:
      condition: service_healthy
  volumes:
    - media_volume:/app/media
    - ./logs:/app/logs
    - /home/xenusad/VA/detections:/home/admin/detections:rw
  env_file:
    - ./yolo/.env
  ports:
    - "5000:5000"
    - "8554:8554"
  deploy:
    resources:
      reservations:
        devices:
          - driver: nvidia
            count: all
            capabilities: [gpu]
  networks:
    - videoanalytics_network
  restart: unless-stopped

```

Рисунок 25 – Пример кода для запуска контейнера с видеоналитикой

```

django:
  build: ./django
  depends_on:
    db:
      condition: service_started
  environment:
    - DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@db:5432/${DB_NAME}
    - DEBUG=${DEBUG:-False}
    - SECRET_KEY=${SECRET_KEY}
    - ALLOWED_HOSTS=${ALLOWED_HOSTS:-localhost,127.0.0.1,videoanalytics.xenus.dev}
    - DJANGO_SETTINGS_MODULE=config.settings
  volumes:
    - static_volume:/app/staticfiles
    - media_volume:/app/media
  restart: unless-stopped
  networks:
    - videoanalytics_network
  command: >
    sh -c "
      python manage.py migrate --noinput &&
      python manage.py collectstatic --noinput &&
      gunicorn config.wsgi:application --bind 0.0.0.0:8000 --workers 3
    "

```

Рисунок 26 – Пример кода для запуска контейнера с серверной частью

Также необходимо добавить env файлы, в которых буду храниться пароли.

2 Безопасность жизнедеятельности

Безопасность жизнедеятельности человека, который большую часть времени проводит за компьютером является важной темой в век компьютерных технологий.

При непрерывной работе с компьютером на человека воздействует большое количество неблагоприятных факторов. Монитор является источником мерцания, из-за чего развивается зрительное утомление, появляются сухость и покраснение глаз. Особенно ярко эти симптомы проявляются, если в помещении недостаточная освещённость либо яркость и контрастность экрана настроены неправильно.

Кроме того, при длительной сидячей работе неизбежно страдает опорно-двигательный аппарат, что в итоге приводит к проблемам со спиной.

В связи с этим в данном разделе описываются меры предосторожности и рекомендации по организации рабочего места, позволяющие минимизировать негативное воздействие длительной работы за компьютером [20].

2.1 Эргономические требования и режим труда при работе с компьютером

Для начало стоит уделить внимание эргономике: стол и кресло необходимо подобрать таким образом, чтобы в положении сидя локти образовывали прямой угол, а взгляд был направлен слегка вниз. Расстояние от глаз до монитора должно составлять не менее 50–70 сантиметров.

Не менее важно организовать правильное освещение. Необходимо, чтобы естественный свет поступал в помещение сбоку от монитора, а искусственный свет был равномерным и не создавал бликов на экране. Яркость самого монитора следует настраивать с учётом освещённости в комнате.

Обязательным условием сохранения работоспособности и здоровья является соблюдение режима труда и отдыха. Рекомендуются делать короткие

перерывы каждые 45 – 60 минут. Во время отдыха стоит сделать простую гимнастику для глаз и мышц.

2.2 Пожарная безопасность при эксплуатации компьютерной техники

При использовании компьютерной техники важно соблюдать нормы пожарной безопасности, поскольку оборудование является электроустановкой, подверженной риску перегрева и короткого замыкания. Прежде всего необходимо правильно организовать прокладку кабелей: провода не должны быть перекручены, пережаты мебелью или проходить в местах с высокой механической нагрузкой. Кабели следует размещать в специальных кабель-каналах либо закреплять на стене с помощью держателей, чтобы исключить их повреждение и спутывание.

Розетки и удлинители должны быть исправны, рассчитаны на необходимую нагрузку и не перегружены множеством подключённых устройств.

В помещении, где эксплуатируется компьютерная техника, необходимо иметь исправный огнетушитель, который следует разместить в легкодоступном месте.

При возникновении возгорания в первую очередь необходимо отключить питание оборудования. Если возгорание небольшое его можно потушить самостоятельно при помощи огнетушителя. Нельзя тушить горящую электронику водой — это может привести к поражению электрическим током и усилению пожара.

Это базовые рекомендации по организации работы, которые могут помочь сохранить здоровье при работе с компьютером.

Заключение

В ходе выполнения выпускной квалификационной работы была спроектирована и разработана система видеоаналитики для автоматического контроля и учёта автотранспорта с использованием методов компьютерного зрения.

Было проведено исследование выбранной области: выявлены достоинства и недостатки имеющихся на рынке систем. На основе полученной информации были поставлены цели по разработке, которых удалось достичь: создать рабочую систему, которая способна распознавать государственные регистрационные знаки и вести учёт транспортных средств. Также был разработан графический интерфейс в виде сайта, для простого доступа к информации о транспорте.

Практическая значимость работы заключается в снижении трудозатрат на ручной контроль транспорта и повышении точности учёта. Внедрение подобной системы способствует автоматизации и может быть полезно для организаций, управляющих парковками и контрольно-пропускными пунктами.

Перспективы дальнейшего развития системы включают расширение функционала за счёт распознавания транспортных средств по типам, а также повышение устойчивости модели к сложным условиям съёмки.

Таким образом, разработанная система видеоаналитики демонстрирует потенциал для практического использования и закладывает основу для создания более сложных систем.

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		66

Список использованных источников

1. UniServer Auto: сайт компании // UniServer. [Электронный ресурс]. – URL: <https://uniserver-auto.com/> (дата обращения: 22.04.2025).
2. Mallenom: сайт компании // Mallenom. [Электронный ресурс]. – URL: <https://www.mallenom.ru> (дата обращения: 22.04.2025).
3. RusGuard: поиск товара DVS-102-UN // RusGuard. [Электронный ресурс]. – URL: https://rusguard.shop/search/search_do/?search_string=DVS-102-UN (дата обращения: 22.04.2025).
4. Tral: система «Tral Parking 2» // Tral. [Электронный ресурс]. – URL: https://www.tral.ru/production/tral_parking/tral_parking_2/ (дата обращения: 22.04.2025).
5. OpenCV: официальная документация библиотеки компьютерного зрения [Электронный ресурс]. – URL: <https://docs.opencv.org/> (дата обращения: 22.04.2025).
6. GStreamer: официальная документация фреймворка [Электронный ресурс]. – URL: <https://gstreamer.freedesktop.org/documentation/> (дата обращения: 22.04.2025).
7. imageio: документация библиотеки Python [Электронный ресурс]. – URL: <https://imageio.readthedocs.io/> (дата обращения: 22.04.2025).
8. Django: официальная документация веб-фреймворка [Электронный ресурс]. – URL: <https://www.djangoproject.com/> (дата обращения: 22.04.2025).
9. Flask: официальная документация микрофреймворка [Электронный ресурс]. – URL: <https://flask.palletsprojects.com/> (дата обращения: 22.04.2025).
10. FastAPI: официальная документация фреймворка [Электронный ресурс]. – URL: <https://fastapi.tiangolo.com/> (дата обращения: 22.04.2025).
11. PyCharm: официальный сайт IDE // JetBrains [Электронный ресурс]. – URL: <https://www.jetbrains.com/pycharm/> (дата обращения: 22.04.2025).

12. Visual Studio: официальный сайт IDE // Microsoft [Электронный ресурс]. – URL: <https://visualstudio.microsoft.com/> (дата обращения: 22.04.2025).
13. Visual Studio Code: официальный сайт редактора кода // Microsoft [Электронный ресурс]. – URL: <https://code.visualstudio.com/> (дата обращения: 22.04.2025).
14. Anaconda Documentation [Электронный ресурс] // Anaconda. – URL: <https://docs.anaconda.com/> (дата обращения: 22.04.2025).
15. Spyder [Электронный ресурс]. – URL: <https://www.spyder-ide.org/> (дата обращения: 22.04.2025).
16. YOLOv8 Models: документация и сравнение моделей [Электронный ресурс] // Ultralytics. – URL: <https://docs.ultralytics.com/ru/compare/yolov8-vs-yolo11> (дата обращения: 22.04.2025).
17. Car plate detecting dataset [Электронный ресурс] // Hugging Face. – URL: https://huggingface.co/datasets/AY000554/Car_plate_detecting_dataset (дата обращения: 22.04.2025).
18. Nomeroff Russian license plates [Электронный ресурс] // Kaggle. – URL: <https://www.kaggle.com/datasets/evgrafovmaxim/nomeroff-russian-license-plates> (дата обращения: 22.04.2025).
19. Video Analytics Xenus: сайт системы видеоаналитики [Электронный ресурс] // Xenus. – URL: <https://videoanalytics.xenus.dev> (дата обращения: 22.04.2025).
20. Безопасность жизнедеятельности: организация рабочего места пользователя ПК [Электронный ресурс]. – URL: <https://ele74197079.narod.ru/pk.pdf> (дата обращения: 22.04.2025).

Приложение А

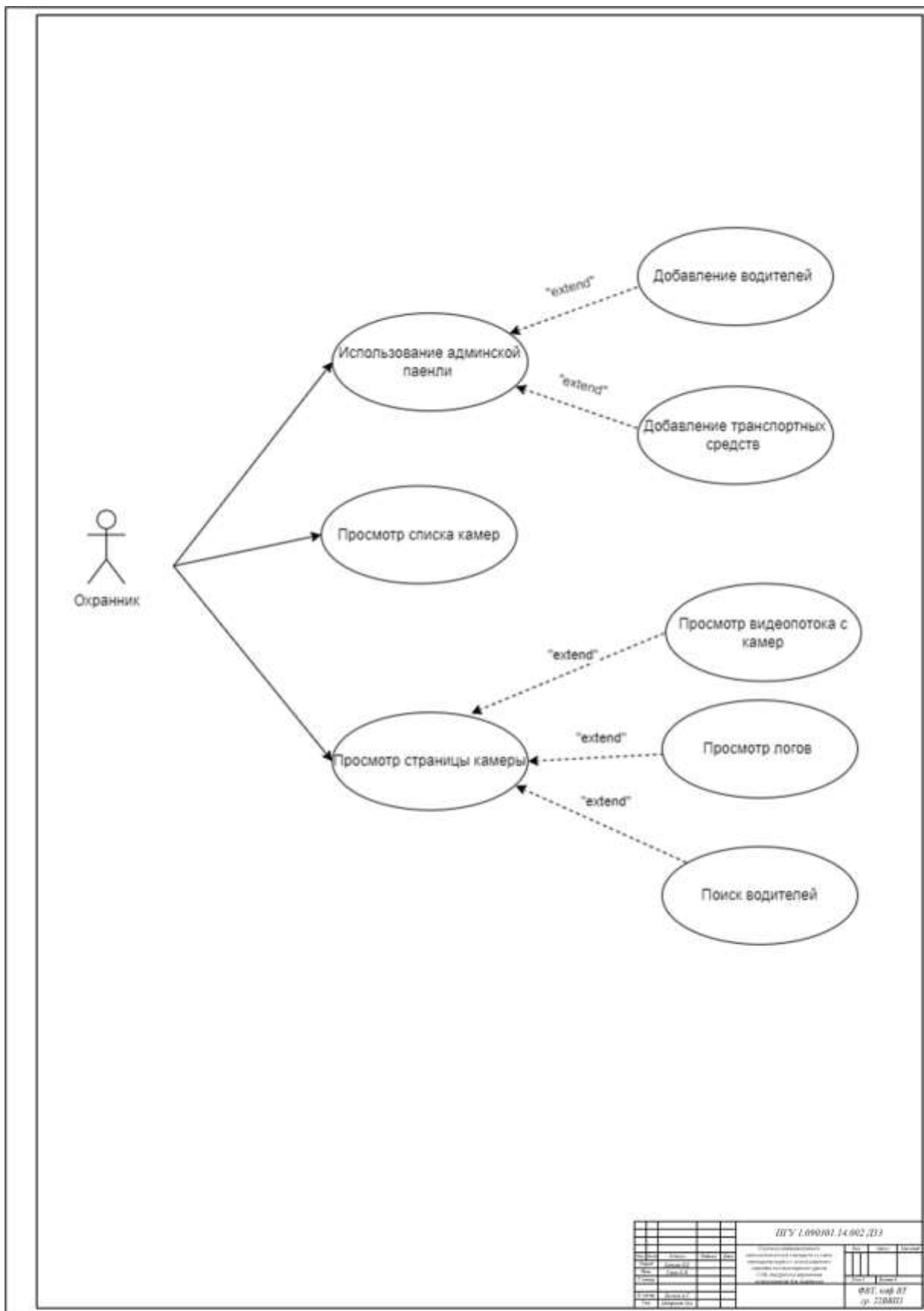
(обязательное)

Система видеоаналитики автоматического контроля и учета автотранспорта с
использованием методов компьютерного зрения

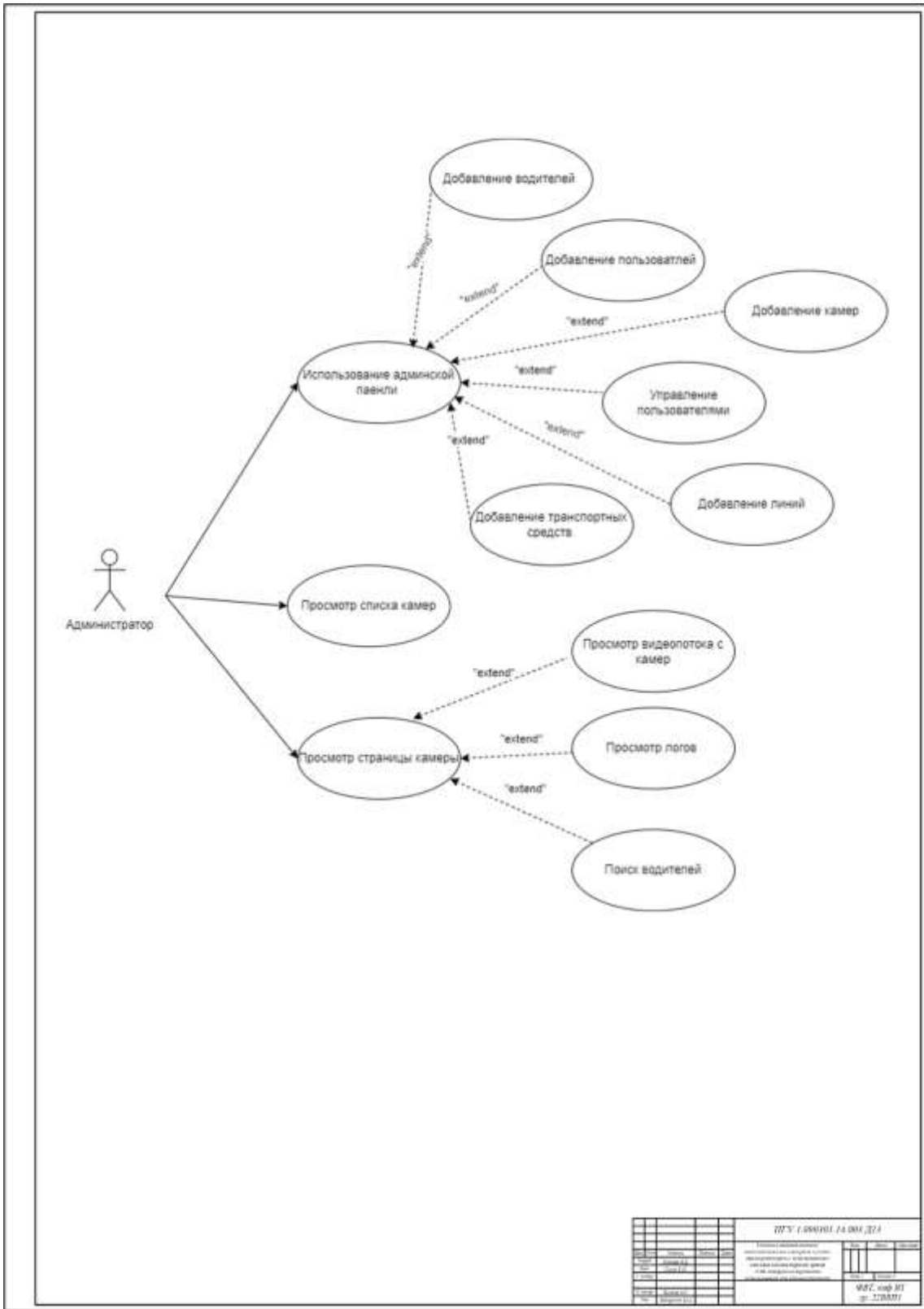
Графическая часть

					ПГУ1.090301.14.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

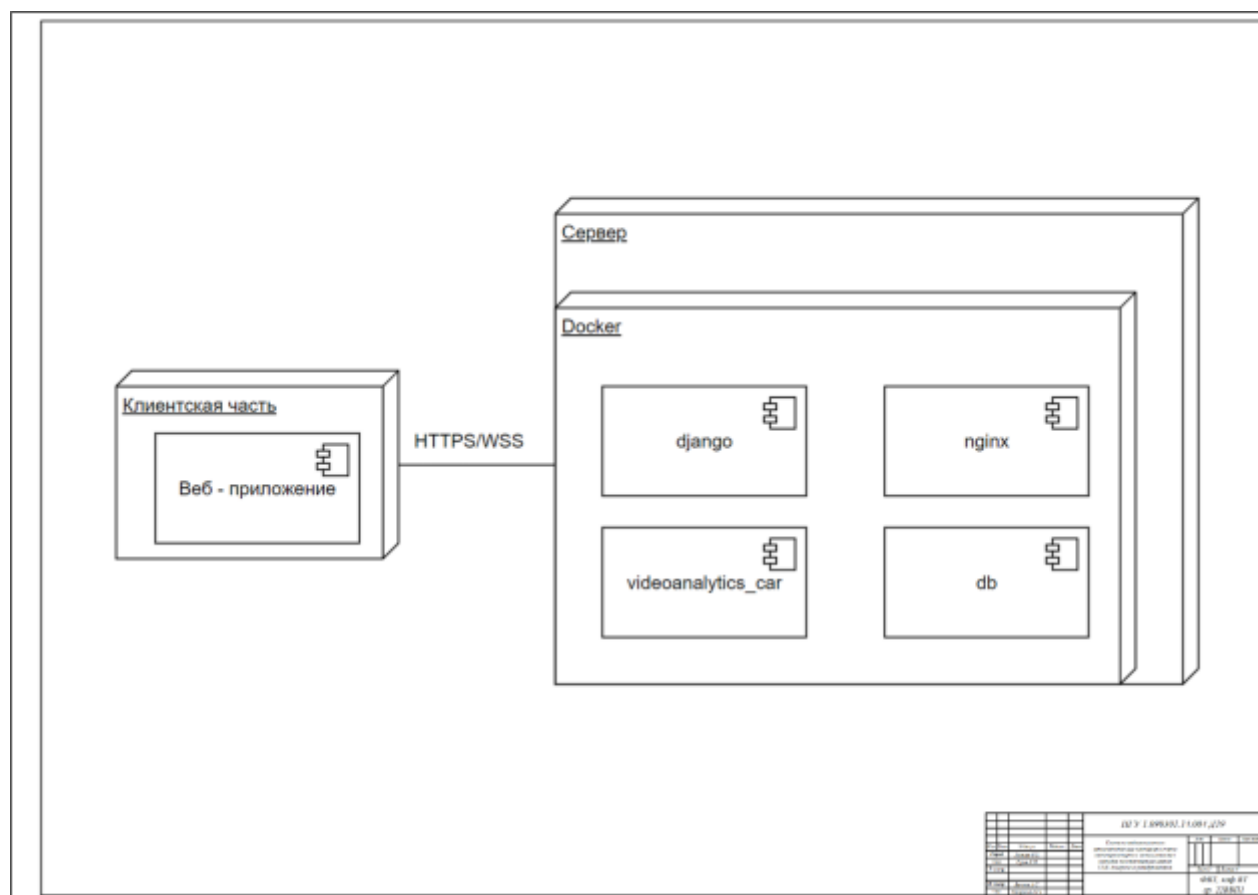
UML – диаграмма вариантов использования для охранника



UML – диаграмма вариантов использования для администратора



UML – диаграмма развёртывания



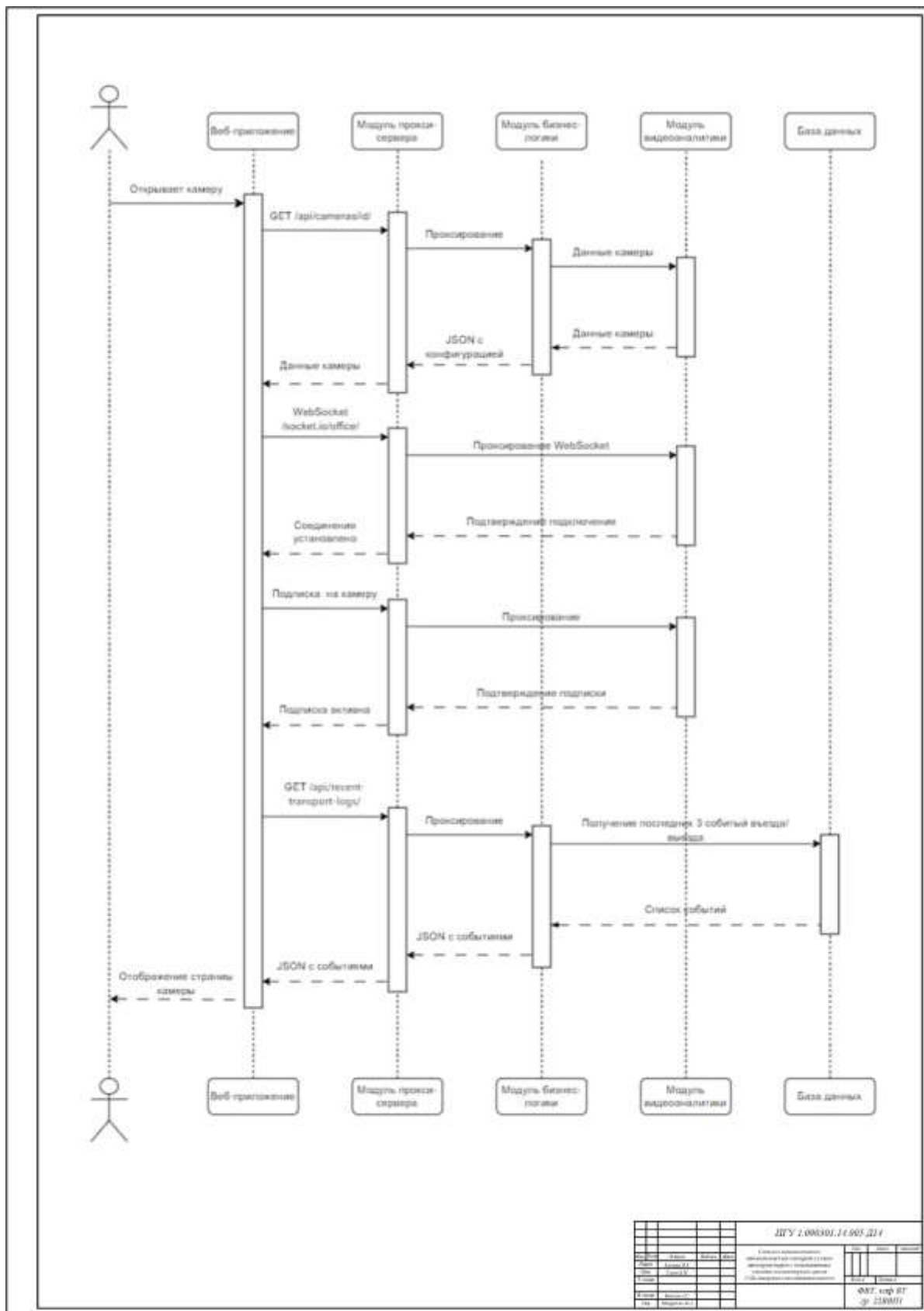
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

72

UML – диаграмма последовательности



Приложение Б

(обязательное)

Система видеоаналитики автоматического контроля и учета автотранспорта с
использованием методов компьютерного зрения

Листинг программы

					ПГУ1.090301.14.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

Файл docker-compose.yml

```
services:
  redis:
    image: redis:alpine
    volumes:
      - redis_data:/data
    networks:
      - videoanalytics_network
    restart: unless-stopped

  db:
    image: postgres:15-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=${DB_NAME}
      - POSTGRES_USER=${DB_USER}
      - POSTGRES_PASSWORD=${DB_PASSWORD}
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER} -d
${DB_NAME}"]
      interval: 30s
      timeout: 10s
      retries: 3
    networks:
      - videoanalytics_network

  Videoanalytics_car:
    profiles: ["no start"]
    build: ./yolo
    depends_on:
      db:
        condition: service_healthy
    #environment:
    # -
    DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@db:5432/${DB_
NAME}
    # - PYTHONUNBUFFERED=1
    volumes:
      #- ./models:/app/models
      - media_volume:/app/media
      - ./logs:/app/logs
      -
/home/xenusad/VA/detections:/home/admin/detections:rw
    env_file:
      - ./yolo/.env
    ports:
```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

```

    - "5000:5000"
    - "8554:8554"
  deploy:
    resources:
      reservations:
        devices:
          - driver: nvidia
            count: all
            capabilities: [gpu] # Если есть GPU
  networks:
  - videoanalytics_network
  #network_mode: host
  restart: unless-stopped

django:
  #build: ./django
  image: registry.xenus.dev/video-analytics-
system/is_backend/django:develop
  depends_on:
    db:
      condition: service_started
  environment:
    -
DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@db:5432/${DB_
NAME}
    - DEBUG=${DEBUG:-False}
    - SECRET_KEY=${SECRET_KEY}
    - ALLOWED_HOSTS=${ALLOWED_HOSTS:-
localhost,127.0.0.1,videoanalytics.xenus.dev}
    - DJANGO_SETTINGS_MODULE=config.settings
  volumes:
    - static_volume:/app/staticfiles
    - media_volume:/app/media
  restart: unless-stopped
  networks:
  - videoanalytics_network
  command: >
    sh -c "
      python manage.py migrate --noinput &&
      python manage.py collectstatic --noinput &&
      gunicorn config.wsgi:application --bind 0.0.0.0:8000
--workers 3
    "

nginx:
  build: ./nginx
  volumes:
    - /etc/letsencrypt:/etc/letsencrypt:ro
    - static_volume:/app/staticfiles
    - media_volume:/app/media
    - /var/www/dist:/var/www/dist:ro

```

```

    ports:
      - "80:80"
      - "443:443"

volumes:
  postgres_data:
  redis_data:
  static_volume:
  media_volume:
  grafana_data:

networks:
  videoanalytics_network:
    driver: bridge
    name: videoanalytics_network

video-recorder:
  profiles: ["no start"]
  build: ./video-recorder
  container_name: video-recorder
  depends_on:
    db:
      condition: service_healthy
  volumes:
    - /home/xenusad/VA/recordings:/recordings:rw # Папка
на хосте
    - ../.env:/home/admin/.env:ro
  environment:
    - RECORDINGS_DIR=/recordings
    - SEGMENT_DURATION=300
    - MAX_RECORDINGS_GB=100
    - AUTO_RECORD=true
    - RECORD_DETECTION_TYPE=3
    - DB_HOST=db
    - DB_PORT=5432
    - DB_NAME=${DB_NAME}
    - DB_USER=${DB_USER}
    - DB_PASSWORD=${DB_PASSWORD}
  networks:
    - videoanalytics_network
  restart: unless-stopped

```

Файл nginx.conf

```

server {
    listen 443 ssl;
    http2 on;
    server_name videoanalytics.xenus.dev;

    ssl_certificate
/etc/letsencrypt/live/videoanalytics.xenus.dev/fullchain.pem;

```

					ПГУ1.090301.14.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

```

        ssl_certificate_key
/etc/letsencrypt/live/videoanalytics.xenus.dev/privkey.pem;
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    client_max_body_size 150M;
    set_real_ip_from 0.0.0.0/0;
    real_ip_header X-Forwarded-For;

    root /var/www/dist/;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /socket.io/ {
        proxy_pass http://videoanalytics_car:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_buffering off;
        proxy_cache off;
        proxy_read_timeout 86400;
    }

    location ~
^/(admin|api|static_backend|media_backend|upload_file|delete_file
)/ {
        proxy_pass http://django:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

}

server {
    listen 80;
    server_name videoanalytics.xenus.dev;

    return 301 https://$server_name$request_uri;
}

```

```
}
```

Файл urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.views.generic import RedirectView
from .views import UserPermissionsView
from django.conf import settings
from django.conf.urls.static import static
from . import views
from rest_framework.routers import DefaultRouter
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
    TokenVerifyView,
)

router = DefaultRouter()
router.register(r'getModel3d', views.Models3dViewSet,
basename='getModel3d')
router.register(r'getTransportLog', views.TransportLogs,
basename='getTransportLog')
router.register(r'rusguard-events',
views.RusGuardEventViewSet, basename='rusguard-events')

urlpatterns = [
    path('api/token/', TokenObtainPairView.as_view(),
name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(),
name='token_refresh'),
    path('api/token/verify/', TokenVerifyView.as_view(),
name='token_verify'),
    path('api/cameras/', views.CameraListView.as_view(),
name='camera-list'),
    path('api/cameras/<int:pk>',
views.CameraDetailView.as_view(), name='camera-detail'),
    path('upload_file/', views.upload_file,
name='upload_file'),
    path('delete_file/', views.delete_file,
name='delete_file'),
    path('api/getModel3d/<str:type_name>',
views.Models3dByTypeView, name='models3d-by-type'),
    path('api/getModel3d/<int:type_number>',
views.Models3dByNumberView, name='models3d-by-number'),
    path('api/camera/<int:camera_id>/work_time/<str:date>',
views.EmployeeWorkTimeView.as_view(), name='employee_work_time'),
    path('api/gate_open/', views.GateOpenView,
name='Gate_open'),
    path('api/get_info_employee/<str:number>',
views.InfoEmployeeView, name='info_employee'),
```

```

        path('api/cameras/<int:camera_id>/transport-
logs/search/', views.TransportLogsSearchView.as_view(),
name='transport-logs-search'),
        path('api/', include(router.urls)),
        path('api/user/permissions/',
UserPermissionsView.as_view(), name='user_permissions'),
        path('api/rusguard/event/', views.rusguard_event,
name='rusguard_event'),
        path('api/rusguard/events/',
views.RusGuardEventsView.as_view(), name='rusguard_events'),
        path('api/rusguard/events/search/',
views.RusGuardEventsSearchView.as_view(),
name='rusguard_events_search'),
        path('api/rusguard/events/paginated/',
views.RusGuardEventsPaginatedView.as_view(),
name='rusguard_events_paginated'),
        path('api/cameras/<int:camera_id>/recent-transport-
logs/', views.RecentTransportLogsView.as_view(), name='recent-
transport-logs'),
        path('api/universal-search/', views.UniversalSearchView,
name='universal-search'),
        path('api/3d-models/', views.get_all_3d_models,
name='all-3d-models'),
    ]

```

```

# Для медиа-файлов в режиме разработки
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)

```

Файл serializers.py

```

# website/serializers.py
from rest_framework import serializers
from .models import Camera, CameraZone, CameraLine,
Models3d, Employee, Vehicle, VehicleEvent, PersonEvent,
PersonZoneSession, FaceEmbedding, FaceRecognitionLog,
RusGuardEvent

class CameraZoneSerializer(serializers.ModelSerializer):
    class Meta:
        model = CameraZone
        fields = [
            'id',
            'name',
            'camera',
            'polygon',
            'is_active',
            'color',
            'description',
            'created_at',
            'updated_at'

```

```

    ]

class CameraLineSerializer(serializers.ModelSerializer):
    class Meta:
        model = CameraLine
        fields = [
            'id',
            'name',
            'camera',
            'points',
            'direction',
            'is_active',
            'color',
            'description',
            'created_at',
            'updated_at'
        ]

class CameraSerializer(serializers.ModelSerializer):
    zones = CameraZoneSerializer(many=True, read_only=True)
    lines = CameraLineSerializer(many=True, read_only=True)

    class Meta:
        model = Camera
        fields = [
            'id',
            'name',
            'detection_type',
            'coordinates',
            'is_active',
            'zones',
            'lines'
        ]

class Models3dSerializer(serializers.ModelSerializer):
    class Meta:
        model = Models3d
        fields = ['id', 'path', 'type', 'number']

class EmployeeSerializer(serializers.ModelSerializer):
    class Meta:
        model = Employee
        fields = ['id', 'name', 'surname', 'company']

class VehicleSerializer(serializers.ModelSerializer):
    class Meta:
        model = Vehicle
        fields = ['id', 'number', 'brand', 'employee']

class VehicleEventSerializer(serializers.ModelSerializer):
    employee = EmployeeSerializer(read_only=True)

```

```

vehicle = VehicleSerializer(read_only=True)
class Meta:
    model = VehicleEvent
    fields = [
        'id', 'number', 'arrival_time', 'departure_time',
        'stay_duration', 'vehicle', 'employee',
        'arrival_photo', 'departure_photo'
    ]
    read_only_fields = ['stay_duration']

class PersonEventSerializer(serializers.ModelSerializer):
    camera_name =
serializers.CharField(source='camera.name', read_only=True)
    zone_name = serializers.CharField(source='zone.name',
read_only=True)
    line_name = serializers.CharField(source='line.name',
read_only=True)
    employee_name = serializers.SerializerMethodField()

class Meta:
    model = PersonEvent
    fields = [
        'id',
        'camera',
        'camera_name',
        'zone',
        'zone_name',
        'line',
        'line_name',
        'employee',
        'employee_name',
        'track_id',
        'event_type',
        'event_time',
        'duration_seconds',
        'point_x',
        'point_y',
        'metadata',
        'created_at',
    ]

    def get_employee_name(self, obj):
        if obj.employee:
            return f"{obj.employee.surname}
{obj.employee.name}"
        return None

class
PersonZoneSessionSerializer(serializers.ModelSerializer):
    camera_name =
serializers.CharField(source='camera.name', read_only=True)

```

```

        zone_name = serializers.CharField(source='zone.name',
read_only=True)
        employee_name = serializers.SerializerMethodField()

        is_active = serializers.SerializerMethodField()
        duration_formatted = serializers.SerializerMethodField()

class Meta:
    model = PersonZoneSession
    fields = [
        'id',
        'camera',
        'camera_name',
        'zone',
        'zone_name',
        'employee',
        'employee_name',
        'track_id',
        'entered_at',
        'exited_at',
        'duration_seconds',
        'duration_formatted',
        'status',
        'is_active',
        'metadata',
        'created_at',
        'updated_at',
    ]

    def get_employee_name(self, obj):
        if obj.employee:
            return f"{obj.employee.surname}
{obj.employee.name}"
        return None

    def get_is_active(self, obj):
        return obj.status == 'open'

    def get_duration_formatted(self, obj):
        if not obj.duration_seconds:
            return None

        seconds = int(obj.duration_seconds)
        minutes = seconds // 60
        secs = seconds % 60

        return f"{minutes} мин {secs} сек"

class
WorkplaceDailySummarySerializer(serializers.Serializer):
    zone_id = serializers.IntegerField()

```

```

zone_name = serializers.CharField()
total_duration_seconds = serializers.FloatField()
total_duration_minutes = serializers.FloatField()
total_duration_formatted = serializers.CharField()

class FaceEmbeddingSerializer(serializers.ModelSerializer):
    class Meta:
        model = FaceEmbedding
        fields = ['id', 'person_id', 'person_name',
'embedding', 'created_at', 'updated_at']

class
FaceRecognitionLogSerializer(serializers.ModelSerializer):
    class Meta:
        model = FaceRecognitionLog
        fields = ['id', 'camera_id', 'frame_id',
'person_id', 'person_name', 'confidence', 'embedding',
'face_image', 'created_at']

class RusGuardEventSerializer(serializers.ModelSerializer):
    class Meta:
        model = RusGuardEvent
        fields = '__all__'

```

Файл models.py

```

from django.db import models
from django.contrib.postgres.fields import ArrayField
from django.db.models import JSONField
from rest_framework import viewsets
import uuid

class Camera(models.Model):
    """
    Таблица camera
    """
    url = models.CharField(max_length=500, null=True,
blank=True)
    detection_type = models.CharField(max_length=20,
null=True, blank=True)
    name = models.CharField(max_length=100)
    coordinates = models.CharField(max_length=500,
null=True, blank=True)
    is_active = models.BooleanField(default=True)

    class Meta:
        db_table = 'camera'
        verbose_name = 'Камера'
        verbose_name_plural = 'Камеры'
        managed = True

    def __str__(self):

```

```

        return self.name

class Group(models.Model):
    """
    Таблица для групп с дополнительными полями
    """
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=150, unique=True)
    description = models.TextField(null=True, blank=True)
    permissions_id = ArrayField(
        models.IntegerField(),
        null=True,
        blank=True,
        verbose_name="ID разрешений"
    )

    class Meta:
        db_table = 'website_group' # ← ИЗМЕНЕНО: уникальное
имя таблицы
        managed = True

    def __str__(self):
        return self.name

class AuthGroupPermissions(models.Model):
    """
    Таблица для разрешений групп
    """
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=100, null=True,
blank=True)
    description = models.TextField(null=True, blank=True)
    cameras = models.ForeignKey(
        Camera,
        on_delete=models.CASCADE,
        db_column='cameras_id',
        null=True,
        blank=True
    )

    class Meta:
        db_table = 'website_group_permissions' # ←
ИЗМЕНЕНО: уникальное имя таблицы
        managed = True

    def __str__(self):
        return self.name or f"Permission {self.id}"

class Models3d(models.Model):
    path = models.CharField('Путь к 3д модели',
max_length=500, blank=True)

```

```

TYPE_CHOICES = [
    ('car', 'Автомобиль'),
    ('people', 'Человек'),
    ('street label', 'Метка улица'),
    ('room label', 'Метка помещение'),
    ('territory', 'Территория'),
    ('other', 'Другое'),
]
type = models.CharField('Тип
модели', max_length=500, choices=TYPE_CHOICES, default='other',
blank=True)
number = models.IntegerField('Номер модели',
null=True, blank=True)
class Meta:
    verbose_name = 'Загрузка моделей'
    verbose_name_plural = 'Загрузка моделей'

def __str__(self):
    return self.path or f"модели #{self.id}"

class Employee(models.Model):
    name = models.CharField(max_length=100,
verbose_name="Имя")
    surname = models.CharField(max_length=100,
verbose_name="Фамилия")
    company = models.CharField(max_length=200,
verbose_name="Компания")

    class Meta:
        db_table = 'employee'
        verbose_name = 'Сотрудник'
        verbose_name_plural = 'Сотрудники'
        ordering = ['surname', 'name']
        managed = True

    def __str__(self):
        return f"{self.surname} {self.name}
({self.company})"

class Vehicle(models.Model):
    number = models.CharField(max_length=20,
verbose_name="Номер", unique=True)
    brand = models.CharField(max_length=100,
verbose_name="Марка")
    employee = models.ForeignKey(
        Employee,
        on_delete=models.CASCADE,
        related_name='vehicles',
        verbose_name="Владелец",
        db_column='employee_id'

```

```

)

class Meta:
    db_table = 'vehicle'
    verbose_name = 'Транспортное средство'
    verbose_name_plural = 'Транспортные средства'
    ordering = ['number']
    managed = True

def save(self, *args, **kwargs):
    # Конвертация русских букв
    ru_to_lat = {
        'А': 'A', 'В': 'B', 'Е': 'E', 'К': 'K', 'М':
'M',
        'Н': 'H', 'О': 'O', 'Р': 'P', 'С': 'C', 'Т':
'T',
        'У': 'Y', 'Х': 'X',
    }
    if self.number:
        self.number = ''.join(ru_to_lat.get(c, c) for c
in self.number).upper()
    super().save(*args, **kwargs)

def str(self):
    return f"{self.number} - {self.brand} (Владелец:
{self.employee})"

class VehicleEvent(models.Model):
    number = models.CharField(max_length=500,
verbose_name="Номер", null=True, blank=True)
    vehicle = models.ForeignKey(
        Vehicle,
        on_delete=models.CASCADE,
        related_name='events',
        verbose_name="Транспортное средство",
        db_column='vehicle_id',
        null=True,
        blank=True
    )
    employee = models.ForeignKey(
        Employee,
        on_delete=models.CASCADE,
        related_name='vehicle_events',
        verbose_name="Сотрудник",
        db_column='employee_id',
        null=True,
        blank=True
    )
    arrival_time = models.DateTimeField(verbose_name="Время
приезда", null=True, blank=True)

```

```

        departure_time =
models.DateTimeField(verbose_name="Время уезда", null=True,
blank=True)
        stay_duration = models.DurationField(verbose_name="Время
пребывания", null=True, blank=True)
        arrival_photo = models.CharField(max_length=500,
verbose_name="Ссылка на фото приезда", null=True, blank=True)
        departure_photo = models.CharField(max_length=500,
verbose_name="Ссылка на фото уезда", null=True, blank=True)

class Meta:
    db_table = 'vehicle_event'
    verbose_name = 'Событие транспортного средства'
    verbose_name_plural = 'События транспортных средств'
    ordering = ['-arrival_time']
    managed = True

    def __str__(self):
        vehicle_info = f"{self.vehicle.number} -
{self.vehicle.brand}"
        employee_info = f"{self.employee.surname}
{self.employee.name}"
        arrival = self.arrival_time.strftime("%d.%m.%Y
%H:%M") if self.arrival_time else "Не указано"
        return f"{vehicle_info} | {employee_info} |
{arrival}"

    def save(self, *args, **kwargs):
        """Автоматически вычисляем время пребывания при
сохранении"""
        if self.arrival_time and self.departure_time:
            self.stay_duration = self.departure_time -
self.arrival_time
        super().save(*args, **kwargs)

class CameraZone(models.Model):
    name = models.CharField(max_length=150,
verbose_name='Название зоны')
    camera = models.ForeignKey(
        Camera,
        on_delete=models.CASCADE,
        related_name='zones',
        db_column='camera_id',
        verbose_name='Камера'
    )
    polygon = models.JSONField(
        verbose_name='Полигон зоны',
        help_text='Список точек полигона в формате [[x1,
y1], [x2, y2], ...]'
    )

```

```

        is_active = models.BooleanField(default=True,
verbose_name='Активна')
        color = models.CharField(max_length=20, null=True,
blank=True, verbose_name='Цвет зоны')
        description = models.TextField(null=True, blank=True,
verbose_name='Описание')
        created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Создано')
        updated_at = models.DateTimeField(auto_now=True,
verbose_name='Обновлено')

class Meta:
    db_table = 'camera_zone'
    verbose_name = 'Зона камеры'
    verbose_name_plural = 'Зоны камер'
    ordering = ['camera_id', 'id']
    managed = True

def __str__(self):
    return f"{self.camera.name} - {self.name}"

class CameraLine(models.Model):
    DIRECTION_CHOICES = [
        ('both', 'В обе стороны'),
        ('in', 'Только вход'),
        ('out', 'Только выход'),
    ]

    name = models.CharField(max_length=150,
verbose_name='Название линии')
    camera = models.ForeignKey(
        Camera,
        on_delete=models.CASCADE,
        related_name='lines',
        db_column='camera_id',
        verbose_name='Камера'
    )
    points = models.JSONField(
        verbose_name='Точки линии',
        help_text='Две точки линии в формате [[x1, y1], [x2,
y2]]'
    )
    direction = models.CharField(
        max_length=10,
        choices=DIRECTION_CHOICES,
        default='both',
        verbose_name='Направление контроля'
    )
    is_active = models.BooleanField(default=True,
verbose_name='Активна')

```

```

        color = models.CharField(max_length=20, null=True,
blank=True, verbose_name='Цвет линии')
        description = models.TextField(null=True, blank=True,
verbose_name='Описание')
        created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Создано')
        updated_at = models.DateTimeField(auto_now=True,
verbose_name='Обновлено')

class Meta:
    db_table = 'camera_line'
    verbose_name = 'Линия камеры'
    verbose_name_plural = 'Линии камер'
    ordering = ['camera_id', 'id']
    managed = True

def __str__(self):
    return f"{self.camera.name} - {self.name}"

class PersonEvent(models.Model):
    """
    Журнал всех событий (вход/выход/пересечение линии)
    """

    EVENT_TYPE_CHOICES = [
        ('zone_enter', 'Вход в зону'),
        ('zone_exit', 'Выход из зоны'),
        ('line_cross_in', 'Пересечение линии: вход'),
        ('line_cross_out', 'Пересечение линии: выход'),
        ('line_cross', 'Пересечение линии'),
    ]

    camera = models.ForeignKey(
        Camera,
        on_delete=models.CASCADE,
        related_name='person_events',
        db_column='camera_id',
        verbose_name='Камера'
    )

    zone = models.ForeignKey(
        'CameraZone',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='person_events',
        db_column='zone_id',
        verbose_name='Зона'
    )

    line = models.ForeignKey(

```

```

        'CameraLine',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='person_events',
        db_column='line_id',
        verbose_name='Линия'
    )

    employee = models.ForeignKey(
        Employee,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='person_events',
        db_column='employee_id',
        verbose_name='Сотрудник'
    )

    track_id = models.IntegerField(verbose_name='ID трека')

    event_type = models.CharField(
        max_length=30,
        choices=EVENT_TYPE_CHOICES,
        verbose_name='Тип события'
    )

    event_time = models.DateTimeField(verbose_name='Время
события')

    duration_seconds = models.FloatField(
        null=True,
        blank=True,
        verbose_name='Длительность, сек'
    )

    point_x = models.IntegerField(null=True, blank=True,
verbose_name='X точки')
    point_y = models.IntegerField(null=True, blank=True,
verbose_name='Y точки')

    metadata = models.JSONField(
        null=True,
        blank=True,
        verbose_name='Доп. данные'
    )

    created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Создано')

    class Meta:

```

```

        db_table = 'person_event'
        verbose_name = 'Событие человека'
        verbose_name_plural = 'События людей'
        ordering = ['-event_time']
        managed = True

    def __str__(self):
        target = self.zone.name if self.zone else
self.line.name if self.line else 'Без объекта'
        return f"{self.get_event_type_display()} |
track={self.track_id} | {target}"

class PersonZoneSession(models.Model):
    """
    Сессия пребывания человека в зоне (одна запись = один
период)
    """

    STATUS_CHOICES = [
        ('open', 'Открыта'),
        ('closed', 'Закрыта'),
    ]

    camera = models.ForeignKey(
        Camera,
        on_delete=models.CASCADE,
        related_name='person_zone_sessions',
        db_column='camera_id',
        verbose_name='Камера'
    )

    zone = models.ForeignKey(
        'CameraZone',
        on_delete=models.CASCADE,
        related_name='person_zone_sessions',
        db_column='zone_id',
        verbose_name='Зона'
    )

    employee = models.ForeignKey(
        Employee,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='person_zone_sessions',
        db_column='employee_id',
        verbose_name='Сотрудник'
    )

    track_id = models.IntegerField(verbose_name='ID трека')

```

```

        entered_at = models.DateTimeField(verbose_name='Время
входа')

        exited_at = models.DateTimeField(
            null=True,
            blank=True,
            verbose_name='Время выхода'
        )

        duration_seconds = models.FloatField(
            null=True,
            blank=True,
            verbose_name='Длительность, сек'
        )

        status = models.CharField(
            max_length=10,
            choices=STATUS_CHOICES,
            default='open',
            verbose_name='Статус'
        )

        metadata = models.JSONField(
            null=True,
            blank=True,
            verbose_name='Доп. данные'
        )

        created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Создано')
        updated_at = models.DateTimeField(auto_now=True,
verbose_name='Обновлено')

        class Meta:
            db_table = 'person_zone_session'
            verbose_name = 'Сессия пребывания в зоне'
            verbose_name_plural = 'Сессии пребывания в зоне'
            ordering = ['-entered_at']
            managed = True

        def __str__(self):
            return f"track={self.track_id} |
зона={self.zone.name} | {self.entered_at}"

        class FaceEmbedding(models.Model):
            person_id = models.CharField(max_length=100,
unique=True)
            person_name = models.CharField(max_length=200)
            photo = models.ImageField(upload_to='faces/', null=True,
blank=True)

```

```

embedding = models.JSONField(null=True, blank=True)
embedding_status = models.CharField(
    max_length=20,
    choices=[
        ('pending', 'Ожидает обработки'),
        ('processing', 'Обрабатывается'),
        ('completed', 'Готово'),
        ('failed', 'Ошибка'),
    ],
    default='pending'
)
error_message = models.TextField(blank=True,
default='') # ДОБАВИТЬ default=''
created_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)

class Meta:
    db_table = 'website_faceembedding' # ДОБАВИТЬ

def __str__(self):
    return self.person_name

class FaceRecognitionLog(models.Model):
    camera_id = models.IntegerField() # ID камеры
    frame_id = models.CharField(max_length=100) # ID кадра
    person_id = models.CharField(max_length=100,
null=True) # ID человека, если распознано
    person_name = models.CharField(max_length=200,
null=True) # Имя человека
    confidence = models.FloatField() # Уверенность
распознавания
    embedding = JSONField(null=True) # Эмбединг лица
    face_image = models.TextField() # Изображение лица
(можно хранить base64 строку или путь)
    created_at = models.DateTimeField(auto_now_add=True) #
Время распознавания

def __str__(self):
    return f"Camera {self.camera_id} - {self.person_name}
if self.person_name else 'Unknown'"

class RusGuardEvent(models.Model):
    """
    События от СКУД RusGuard - только важное
    """
    EVENT_TYPE_CHOICES = [
        ('enter', 'Вход'),
        ('exit', 'Выход'),
    ]

    employee = models.ForeignKey(

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        Employee,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        verbose_name='Сотрудник'
    )
    employee_name = models.CharField(max_length=200,
verbose_name='Имя сотрудника')
    event_type = models.CharField(max_length=10,
choices=EVENT_TYPE_CHOICES, verbose_name='Тип события')
    event_time = models.DateTimeField(verbose_name='Время
события')
    details = models.CharField(max_length=200, null=True,
blank=True, verbose_name='Детали')

    class Meta:
        db_table = 'rusguard_event'
        verbose_name = 'Событие СКУД'
        verbose_name_plural = 'События СКУД'
        ordering = ['-event_time']
        managed = True

    def __str__(self):
        return f"{self.get_event_type_display()} |
{self.employee_name} | {self.event_time}"

```

Файл admin.py

```

from django.contrib import admin
from django.contrib.auth.models import User, Group
from .models import Camera, Models3d, Employee, Vehicle,
CameraZone, CameraLine, FaceEmbedding
from django import forms
from django.utils.html import format_html
import json

class FileManagerMixin:
    change_form_template = "admin/form.html"
    add_form_template = "admin/form.html"
    def change_view(self, request, object_id, form_url='',
extra_context=None):
        extra_context = extra_context or {}
        extra_context['folder_name'] =
self.file_manager_folder
        return super().change_view(request, object_id,
form_url, extra_context)

    def add_view(self, request, form_url='',
extra_context=None):
        extra_context = extra_context or {}
        extra_context['folder_name'] =
self.file_manager_folder

```

```

        return super().add_view(request, form_url,
extra_context)

class CameraZoneInline(admin.TabularInline):
    model = CameraZone
    extra = 1
    fields = ('name', 'polygon', 'is_active', 'color',
'description')
    verbose_name = "Зона"
    verbose_name_plural = "Зоны"

class CameraLineInline(admin.TabularInline):
    model = CameraLine
    extra = 1
    fields = ('name', 'points', 'direction', 'is_active',
'color', 'description')
    verbose_name = "Линия"
    verbose_name_plural = "Линии"

@admin.register(Camera)
class CameraAdmin(admin.ModelAdmin):
    """Админка камер"""
    list_display = ('id', 'name', 'url', 'detection_type',
'is_active', 'coordinates')
    list_filter = ('detection_type', 'is_active')
    search_fields = ('name', 'url', 'coordinates')
    list_editable = ('is_active',) # Можно редактировать
прямо в списке

    # Поля для редактирования
    fields = ('name', 'url', 'detection_type',
'coordinates', 'is_active')
    inlines = [CameraZoneInline, CameraLineInline]

    def get_queryset(self, request):
        qs = super().get_queryset(request)
        user = request.user

        # Ограничение видимости для обычных пользователей
        # Убираем фильтрацию по allowed_users, так как этого
поля нет

        if not user.is_superuser and not user.is_staff:
            # Если нужно ограничить доступ, можно
использовать другие критерии
            # Например, показать только определенные камеры
            pass

        return qs

@admin.register(Models3d)
class Models3dAdmin(FileManagerMixin, admin.ModelAdmin):

```

```

list_display = ['id', 'path', 'type', 'number']
list_editable = ['type', 'number']
list_filter = ['type']
search_fields = ['path', 'type']
fields = ['path', 'type', 'number']
file_manager_folder = 'Models3d'

class VehicleInline(admin.TabularInline):
    """Инлайн для отображения машин в карточке сотрудника"""
    model = Vehicle
    extra = 1 # Количество пустых форм для добавления
    fields = ('number', 'brand')
    verbose_name = "Транспортное средство"
    verbose_name_plural = "Транспортные средства"

@admin.register(Employee)
class EmployeeAdmin(admin.ModelAdmin):
    """Админка сотрудников"""
    list_display = ('id', 'name', 'surname', 'company',
'vehicle_count')
    list_filter = ('company',)
    search_fields = ('name', 'surname', 'company')
    fields = ('name', 'surname', 'company')
    inlines = [VehicleInline] # Показываем машины
сотрудника прямо в его карточке

    def vehicle_count(self, obj):
        """Количество машин у сотрудника"""
        return obj.vehicles.count()
    vehicle_count.short_description = 'Количество машин'

@admin.register(Vehicle)
class VehicleAdmin(admin.ModelAdmin):
    """Админка транспортных средств"""
    list_display = ('id', 'number', 'brand',
'employee_info')
    list_filter = ('brand', 'employee__company')
    search_fields = ('number', 'brand', 'employee__name',
'employee__surname')
    fields = ('number', 'brand', 'employee')
    autocomplete_fields = ['employee'] # Удобный поиск
сотрудника при выборе

    def employee_info(self, obj):
        """Информация о владельце"""
        if obj.employee:
            return f"{obj.employee.name}
{obj.employee.surname} ({obj.employee.company})"
        return "Нет владельца"
    employee_info.short_description = 'Владелец'

```

```

def save_model(self, request, obj, form, change):
    # Конвертируем русские буквы в латиницу
    ru_to_lat = {
        'А': 'A', 'В': 'B', 'Е': 'E', 'К': 'K', 'М':
'M',
        'Н': 'H', 'О': 'O', 'Р': 'P', 'С': 'C', 'Т':
'T',
        'У': 'Y', 'Х': 'X',
        'а': 'A', 'в': 'B', 'е': 'E', 'к': 'K', 'м':
'M',
        'н': 'H', 'о': 'O', 'р': 'P', 'с': 'C', 'т':
'T',
        'у': 'Y', 'х': 'X',
    }

    # Конвертируем номер
    if obj.number:
        obj.number = ''.join(ru_to_lat.get(char, char)
for char in obj.number).upper()

    # Конвертируем brand (на случай если тоже русскими)
    if obj.brand:
        obj.brand = ''.join(ru_to_lat.get(char, char)
for char in obj.brand)

    super().save_model(request, obj, form, change)

@admin.register(CameraZone)
class CameraZoneAdmin(admin.ModelAdmin):
    """Админка зон камер"""
    list_display = ('id', 'name', 'camera', 'is_active',
'color')
    list_filter = ('camera', 'is_active')
    search_fields = ('name', 'camera__name')
    fields = ('name', 'camera', 'polygon', 'is_active',
'color', 'description')

@admin.register(CameraLine)
class CameraLineAdmin(admin.ModelAdmin):
    """Админка линей камер"""
    list_display = ('id', 'name', 'camera', 'direction',
'is_active', 'color')
    list_filter = ('camera', 'direction', 'is_active')
    search_fields = ('name', 'camera__name')
    fields = ('name', 'camera', 'points', 'direction',
'is_active', 'color', 'description')

class FaceEmbeddingForm(forms.ModelForm):
    class Meta:
        model = FaceEmbedding

```

```

        fields = ['person_id', 'person_name', 'photo',
'embedding', 'embedding_status']
        widgets = {
            'embedding': forms.Textarea(attrs={
                'rows': 3,
                'cols': 80,
                'placeholder': 'Будет заполнено
автоматически или вставьте JSON вручную'
            }),
        }

    def clean_embedding(self):
        embedding = self.cleaned_data.get('embedding')
        if embedding and isinstance(embedding, str):
            try:
                parsed = json.loads(embedding)
                if not isinstance(parsed, list):
                    raise forms.ValidationError("Эмбединг
должен быть массивом чисел")
            except json.JSONDecodeError:
                raise forms.ValidationError("Неверный формат
JSON")

        return embedding

@admin.register(FaceEmbedding)
class FaceEmbeddingAdmin(admin.ModelAdmin):
    form = FaceEmbeddingForm

    list_display = ('person_id', 'person_name',
'photo_preview', 'embedding_status_badge', 'embedding_info',
'created_at')
    list_filter = ('embedding_status', 'created_at')
    search_fields = ('person_id', 'person_name')
    readonly_fields = ('embedding_status', 'error_message',
'created_at', 'updated_at', 'embedding_info')

    fieldsets = (
        ('Основная информация', {
            'fields': ('person_id', 'person_name')
        }),
        ('Фото лица', {
            'fields': ('photo',),
            'description': 'Загрузите фото лица. Эмбединг
будет создан автоматически'
        }),
        ('Эмбединг', {
            'fields': ('embedding', 'embedding_info',
'embedding_status', 'error_message'),
            'description': 'Эмбединг создается
автоматически. Статус обновляется при запуске DeepFace'
        }),
    )

```

```

        ('Временные метки', {
            'fields': ('created_at', 'updated_at'),
            'classes': ('collapse',)
        }),
    )

    def photo_preview(self, obj):
        if obj.photo:
            return format_html('<img src("{}" width="50"
height="50" style="object-fit: cover;" />', obj.photo.url)
            return "-"
        photo_preview.short_description = "Фото"

    def embedding_status_badge(self, obj):
        colors = {
            'pending': '#ffc107',
            'processing': '#17a2b8',
            'completed': '#28a745',
            'failed': '#dc3545',
        }
        return format_html(
            '<span style="background-color: {}; color:
white; padding: 4px 8px; border-radius: 4px;">{}</span>',
            colors.get(obj.embedding_status, '#6c757d'),
            obj.get_embedding_status_display()
        )
        embedding_status_badge.short_description = "Статус"

    def embedding_info(self, obj):
        if obj.embedding:
            try:
                if isinstance(obj.embedding, str):
                    data = json.loads(obj.embedding)
                else:
                    data = obj.embedding
                length = len(data)
                return f"✓ {length} чисел"
            except:
                return "X Ошибка формата"
        return "-"
        embedding_info.short_description = "Эмбеddинг"

    def save_model(self, request, obj, form, change):
        if obj.photo and not obj.embedding:
            obj.embedding_status = 'pending'
        elif obj.embedding:
            obj.embedding_status = 'completed'

        if isinstance(obj.embedding, str) and obj.embedding:
            try:

```

```

        obj.embedding = json.loads(obj.embedding)
    except:
        pass

```

```

    super().save_model(request, obj, form, change)

```

Файл video_stream_server.py

```

#!/usr/bin/env python3
import os
import sys

import cv2
import eventlet
from eventlet import wsgi
from datetime import datetime, timezone, timedelta
import base64
import time
import signal
import json
import socketio
from ultralytics import YOLO
from dotenv import load_dotenv
import logging
from concurrent.futures import ThreadPoolExecutor
import numpy as np
import torch
import torch.nn as nn
from collections import OrderedDict
from torchvision import transforms
from collections import Counter
from torch.ao.quantization import get_default_qconfig,
QConfigMapping
    from torch.ao.quantization.quantize_fx import prepare_fx,
convert_fx
    import torch.ao.quantization.quantize_fx as quantize_fx

eventlet.monkey_patch(all=True, thread=False)

# Настройка логирования
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s',
    datefmt='%Y-%m-%d %H:%M:%S'
)
logger = logging.getLogger(__name__)

# Загружаем .env файл
load_dotenv('/home/admin/.env')

# Словарь соответствия типов детекции

```

```

DETECTION_TYPES = {
    '1': [0],          # people
    '2': [2, 3, 5, 7], # vehicles
    '3': [2],          # cars
    '4': [0] # all
}

# Создаем Socket.IO сервер
sio = socketio.Server(
    cors_allowed_origins='*',
    async_mode='eventlet',
    logger=False,
    engineio_logger=False,
    ping_timeout=60,
    ping_interval=25,
    max_http_buffer_size=10 * 1024 * 1024
)
app = socketio.WSGIApp(sio)

class CRNN(nn.Module):
    def __init__(self, num_classes):
        super(CRNN, self).__init__()

        # --- CNN часть (Глаза) ---
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2), # -> height: 16

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, 2), # -> height: 8

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(True),

            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d((2, 1), (2, 1)), # -> height: 4

            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(True),

            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d((2, 1), (2, 1)) # -> height: 2
        )

        # --- RNN часть (Мозг) ---

```

```

        self.rnn = nn.LSTM(512 * 2, 256, bidirectional=True,
num_layers=2, batch_first=True)

        # --- Classifier (Пот) ---
        self.classifier = nn.Linear(512, num_classes)

    def forward(self, x):
        # Прогоняем через CNN
        x = self.cnn(x)

        # "Распрямляем" выход CNN для подачи в RNN
        batch, channels, height, width = x.size()
        x = x.reshape(batch, channels * height, width)

        # Меняем оси местами для RNN
        x = x.permute(0, 2, 1)

        # Прогоняем через RNN
        x, _ = self.rnn(x)

        # Прогоняем через классификатор
        x = self.classifier(x)

        # Для CTCLoss нам нужен формат (sequence_length,
batch, num_classes)
        x = x.permute(1, 0, 2)
        x = nn.functional.log_softmax(x, dim=2)

        return x

class LicensePlateOCR:
    """Класс для распознавания номерных знаков с помощью
CRNN (адаптированный под вашу модель)"""

    def __init__(self,
model_path='/home/admin/crnn_ocr_model_best.pth '):
        self.device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')

        # АЛФАВИТ ИЗ ВАШЕГО КОДА
        self.alphabet = '0123456789АВСЕНКМОРТХУ'
        self.num_classes = len(self.alphabet) + 1 # +1 для
blank токена CTC

        self.int_to_char = {i + 1: char for i, char in
enumerate(self.alphabet)}
        self.int_to_char[0] = '' # CTC Blank token

        self.model = None
        self.model_path = model_path
        self.load_model()

```

```

        self.transform = transforms.Compose([
            transforms.ToPILImage(),
            transforms.Grayscale(),
            transforms.Resize((32, 128)),
            transforms.ToTensor(),
            #transforms.Normalize(mean=[0.5], std=[0.5])
        ])
        self.save_debug = os.getenv('SAVE_DEBUG_IMAGES',
'true').lower() == 'true'
        self.debug_folder = os.getenv('DEBUG_FOLDER',
'/home/admin/detections/debug')
        self.camera_id_for_debug = None

        self.full_frames_folder =
os.getenv('FULL_FRAMES_FOLDER',
'/home/admin/detections/full_frames')
        self.full_frames_folder_1 =
os.getenv('FULL_FRAMES_FOLDER', '/app/media/detections')
        self.best_frames = {}

    def save_debug_image(self, image, stage, track_id,
confidence=0):
        """Сохраняет изображение на разных этапах
обработки"""
        try:
            if not self.save_debug:
                logger.info(f"Проблемы с сохранением в
папку")
                return

            timestamp =
datetime.now().strftime("%Y%m%d_%H%M%S_%f")[:-3]
            cam_id = self.camera_id_for_debug if
self.camera_id_for_debug else 0

            # Имя файла: camera_trackid_stage_timestamp.jpg
            filename =
f"cam{cam_id}_track{track_id}_{stage}_{timestamp}.jpg"
            filepath = os.path.join(self.debug_folder,
filename)

            cv2.imwrite(filepath, image)
            logger.debug(f"Отладка сохранена: {filename}")

        except Exception as e:
            logger.error(f"Ошибка сохранения отладки: {e}")

    def load_model(self):
        """Загрузка обычной (неквантованной) CRNN модели"""
        try:

```

```

        # Убираем пробел в конце пути
        model_path_clean = self.model_path.strip()

        if not os.path.exists(model_path_clean):
            logger.warning(f"Модель OCR не найдена:
{model_path_clean}")
            self.model = None
            return

        logger.info(" Загрузка CRNN модели для
распознавания номеров...")
        logger.info(f"    Алфавит: {self.alphabet}")
        logger.info(f"    Количество классов:
{self.num_classes}")

        # Загружаем checkpoint
        checkpoint = torch.load(model_path_clean,
map_location='cpu')

        # Создаем обычную модель
        self.model = CRNN(self.num_classes)

        # Извлекаем state_dict
        if isinstance(checkpoint, dict):
            if 'state_dict' in checkpoint:
                state_dict = checkpoint['state_dict']
            elif 'model' in checkpoint:
                state_dict = checkpoint['model']
            else:
                state_dict = checkpoint
        else:
            state_dict = checkpoint

        # Очищаем ключи (убираем префикс 'module.')
        new_state_dict = OrderedDict()
        for k, v in state_dict.items():
            name = k.replace('module.', '')
            new_state_dict[name] = v

        # Загружаем веса
        self.model.load_state_dict(new_state_dict,
strict=False)

        # Переносим на устройство
        self.model = self.model.to(self.device)
        self.model.eval()

        logger.info(f" CRNN модель загружена на
{self.device}")

        # Тест модели

```

```

        try:
            test_img = np.zeros((32, 128, 3),
dtype=np.uint8)
            test_text, _ =
self.recognize_plate(test_img, track_id=-1)
        except Exception as test_e:
            logger.warning(f" Тест модели не пройден:
{test_e}")

    except Exception as e:
        logger.error(f" Ошибка загрузки CRNN модели:
{e}")

        import traceback
        traceback.print_exc()
        self.model = None

    def _order_points(self, pts: np.ndarray) -> np.ndarray:
        """Упорядочивает углы прямоугольника"""
        rect = np.zeros((4, 2), dtype="float32")
        s = pts.sum(axis=1)
        rect[0] = pts[np.argmin(s)]
        rect[2] = pts[np.argmax(s)]
        diff = np.diff(pts, axis=1)
        rect[1] = pts[np.argmin(diff)]
        rect[3] = pts[np.argmax(diff)]
        return rect

    def _four_point_transform(self, image: np.ndarray, pts:
np.ndarray) -> np.ndarray:
        """Применяет перспективное преобразование для
выравнивания"""
        rect = self._order_points(pts)
        (tl, tr, br, bl) = rect

        # Вычисляем ширину
        widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] -
bl[1]) ** 2))
        widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] -
tl[1]) ** 2))
        maxWidth = max(int(widthA), int(widthB))

        # Вычисляем высоту
        heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] -
br[1]) ** 2))
        heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] -
bl[1]) ** 2))
        maxHeight = max(int(heightA), int(heightB))

        if maxWidth <= 0 or maxHeight <= 0:
            logger.warning(f"Невалидные размеры:
{maxWidth}x{maxHeight}")

```

```

        return image

    maxWidth = min(maxWidth, 800)
    maxHeight = min(maxHeight, 300)

    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]
    ], dtype="float32")

    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, M, (maxWidth,
maxHeight))

    # Поворачиваем если нужно (номер должен быть
горизонтальным)
    if warped.shape[0] > warped.shape[1]:
        warped = cv2.rotate(warped,
cv2.ROTATE_90_CLOCKWISE)

    return warped

def preprocess_image(self, plate_img, track_id=-1):
    """Предобработка изображения номерного знака для OCR
(улучшенная версия)"""
    try:
        # 2. Затем используем transforms для приведения
к нужному формату
        # Конвертируем BGR в RGB для PIL
        if len(plate_img.shape) == 3:
            rgb = cv2.cvtColor(plate_img,
cv2.COLOR_BGR2RGB)
        else:
            rgb = cv2.cvtColor(plate_img,
cv2.COLOR_GRAY2RGB)

        self.save_debug_image(rgb, f"02_rgb", track_id)
        lab = cv2.cvtColor(rgb, cv2.COLOR_RGB2LAB)

        # 3. Применяем transforms
        tensor = self.transform(rgb)

        # 4. Добавляем batch dimension
        tensor = tensor.unsqueeze(0)

        return tensor.to(self.device)

    except Exception as e:

```

```

        logger.error(f"Ошибка предобработки изображения:
{e}")

        return None

    def decode_prediction(self, preds):
        """Декодирование выхода модели в текст с корректной
СТС-логикой"""
        preds = preds.permute(1, 0, 2).argmax(dim=2)[0]

        decoded_seq = []
        prev_char_idx = -1

        for char_idx in preds:
            char_idx = char_idx.item()

            if char_idx == 0:
                prev_char_idx = -1
            else:
                if char_idx != prev_char_idx:
                    char = self.int_to_char.get(char_idx,
'?')

                    decoded_seq.append(char)
                    prev_char_idx = char_idx

        result = "".join(decoded_seq)
        return result

    @torch.no_grad()
    def recognize_plate(self, plate_img, track_id=-1):
        """Распознавание текста на номерном знаке"""
        if self.model is None:
            return None, 0.0

        try:
            # Предобработка
            tensor = self.preprocess_image(plate_img,
track_id)

            if tensor is None:
                return None, 0.0

            '''
            debug_tensor = tensor[0, 0].cpu() # [32, 128]
            debug_img = ((debug_tensor * 0.5 + 0.5) *
255).numpy().astype(np.uint8)
            self.save_debug_image(debug_img,
f"05_tensor_input", track_id)
            '''

            # Инференс
            output = self.model(tensor)

```

```

        # Декодируем
        text = self.decode_prediction(output)

        # Вычисляем уверенность (приблизительно)
        confidence = self.calculate_confidence(output)

        # Постобработка - проверяем формат номера
        if text and len(text) >= 6:
            return text, confidence

        return None, confidence

    except Exception as e:
        return None, 0.0

    def calculate_confidence(self, output):
        """Расчет уверенности распознавания"""
        try:
            probs = torch.exp(output)
            max_probs, _ = probs.max(dim=2)
            confidence = max_probs.mean().item()
            return confidence
        except:
            return 0.0

class CameraStreamer:
    """Класс для обработки одной камеры - МАКСИМАЛЬНАЯ
    ПРОИЗВОДИТЕЛЬНОСТЬ"""

    def __init__(self, camera_data, plate_model,
                 vehicle_model, db_pool):

        self.camera_id = camera_data['id']
        self.camera_name = camera_data['name']
        self.rtsp_url = camera_data['url']
        self.detection_type =
str(camera_data['detection_type']) if
camera_data['detection_type'] else '1'

        self.lines = camera_data.get('lines', [])

        self.vehicle_positions = {}
        self.vehicle_directions = {}
        self.line_cross_history = {}
        # Определяем классы для детекции
        self.detect_classes =
DETECTION_TYPES.get(self.detection_type, [0])

```

```

self.direction_min_frames = 5
self.line_cross_cooldown = 3.0

self.line = {}

self.plate_model = plate_model
self.vehicle_model = vehicle_model

self.vehicle_classes = [2, 3, 5, 7]
self.vehicle_conf_threshold = 0.35
self.plate_conf_threshold = 0.25

self.vehicle_tracker = {}
self.next_vehicle_id = 1

self.db_pool = db_pool
self.running = True
self.frame_count = 0
self.total_detections = 0

self.has_new_event = False
self.number_event = "unknown"

# Параметры из .env
self.process_every = 1
self.resize_frame = os.getenv('RESIZE_FRAME',
'false').lower() == 'true'
self.frame_width = int(os.getenv('FRAME_WIDTH',
'640'))
self.frame_height = int(os.getenv('FRAME_HEIGHT',
'480'))
self.jpeg_quality = int(os.getenv('JPEG_QUALITY',
'75'))

self.enable_ocr = os.getenv('ENABLE_OCR',
'true').lower() == 'true'
self.recognize_only_with_direction =
os.getenv('RECOGNIZE_ONLY_WITH_DIRECTION', 'true').lower() ==
'true'

self.ocr = None # Будет установлен из менеджера
self.plate_cache = {} # Кэш для распознанных
номеров

self.cache_timeout = 2.0 # секунд

self.track_buffer_size = 15

self.enable_tracking = os.getenv('ENABLE_TRACKING',
'true').lower() == 'true'

```

```

        self.save_debug_images =
os.getenv('SAVE_DEBUG_IMAGES', 'true').lower() == 'true'
        self.debug_folder = os.getenv('DEBUG_FOLDER',
'/home/admin/detections/debug')
        self.save_full_frames =
os.getenv('SAVE_FULL_FRAMES', 'true').lower() == 'true'
        self.full_frames_folder =
os.getenv('FULL_FRAMES_FOLDER',
'/home/admin/detections/full_frames')
        self.full_frames_folder_1 =
os.getenv('FULL_FRAMES_FOLDER', '/app/media/detections')

        self.raw_recognitions = {}
        if self.save_debug_images:
            os.makedirs(self.debug_folder, exist_ok=True)
            logger.info(f" Папка для отладки:
{self.debug_folder}")

        if self.save_full_frames:
            os.makedirs(self.full_frames_folder,
exist_ok=True)
            os.makedirs(self.full_frames_folder_1,
exist_ok=True)
            logger.info(f"Папка для полных кадров:
{self.full_frames_folder}")
            self.track_history = {}
            self.min_confidence_count = 3

        logger.info(f" Камера {self.camera_id}:
{self.camera_name} - тип: {self.detection_type}")
        logger.info(f"    Режим: Максимальная
производительность (без ограничения FPS)")
        logger.info(f"    Масштабирование: {'Включено' if
self.resize_frame else 'Выключено'}")

    def set_ocr(self, ocr):
        """Устанавливает OCR и передает camera_id для
отладки"""
        self.ocr = ocr
        if self.ocr and self.save_debug_images:
            self.ocr.camera_id_for_debug = self.camera_id
            logger.info(f" OCR для камеры {self.camera_id}
настроен для отладки")

    def _stabilize_text(self, track_id: int, new_text: str)
-> str:
        """Стабилизация текста на основе истории
распознаваний"""
        if track_id == -1:

```

```

        return new_text # Для объектов без трекинга не
стабилизируем

        # Инициализируем историю для нового track_id
        if track_id not in self.track_history:
            self.track_history[track_id] = []

        # Добавляем новый результат
        self.track_history[track_id].append(new_text)

        # Ограничиваем размер буфера
        if len(self.track_history[track_id]) >
self.track_buffer_size:
            self.track_history[track_id].pop(0)

        # Находим наиболее частый результат
        counts = Counter(self.track_history[track_id])
        if not counts:
            return ""

        best_text, best_count = counts.most_common(1)[0]

        # Возвращаем только если есть достаточное количество
совпадений
        return best_text if best_count >=
self.min_confidence_count else ""

    def recognize_license_plate(self, plate_img, track_id,
vehicle_track_id=None):
        """Распознавание номерного знака с накоплением
результатов"""
        if not self.enable_ocr or not self.ocr:
            return None, 0.0

        current_time = time.time()
        final_cache_key =
f"{self.camera_id}_{track_id}_final" if track_id != -1 else None

        # Проверяем финальный кэш (уже стабилизированный
результат)
        if final_cache_key and final_cache_key in
self.plate_cache:
            cached_text, cached_conf, cached_time =
self.plate_cache[final_cache_key]
            if current_time - cached_time <
self.cache_timeout * 5:
                return cached_text, cached_conf

        plate_text, confidence =
self.ocr.recognize_plate(plate_img, track_id)

```

```

if plate_text and confidence > 0.2:
    # Инициализируем накопитель для этого track_id
    if not hasattr(self, 'raw_recognitions'):
        self.raw_recognitions = {}

    if track_id not in self.raw_recognitions:
        self.raw_recognitions[track_id] = {
            'texts': [],
            'confidences': [],
            'frames': 0,
            'last_update': current_time
        }

    acc = self.raw_recognitions[track_id]
    acc['texts'].append(plate_text)
    acc['confidences'].append(confidence)
    acc['frames'] += 1
    acc['last_update'] = current_time

    # Ограничиваем историю
    if len(acc['texts']) > 20:
        acc['texts'] = acc['texts'][-20:]
        acc['confidences'] = acc['confidences'][-
20:]

    # === ГОЛОСОВАНИЕ: выбираем лучший результат ===
    if acc['frames'] >= 3:
        from collections import Counter
        text_counts = Counter(acc['texts'])

        most_common_text, count =
text_counts.most_common(1)[0]

        text_confidences = [
            conf for text, conf in zip(acc['texts'],
acc['confidences'])
            if text == most_common_text
        ]
        avg_confidence = sum(text_confidences) /
len(text_confidences) if text_confidences else 0

        direction = 'unknown'
        if vehicle_track_id and vehicle_track_id in
self.vehicle_directions:
            direction =
self.vehicle_directions[vehicle_track_id]

        if count >= max(2, len(acc['texts'])) * 0.4)
and (avg_confidence > 0.3) and direction != 'unknown' :
            # Сохраняем финальный результат

```

```

        self.plate_cache[final_cache_key] =
(most_common_text, avg_confidence, current_time)

        # === ДОБАВЛЕНО: Сохраняем кадр в
best_frames с направлением ===
        if self.save_full_frames and
hasattr(self, 'current_frame') and self.current_frame is not
None:
            if not hasattr(self, 'best_frames'):
                self.best_frames = {}
            self.best_frames[track_id] =
(self.current_frame.copy(), avg_confidence, most_common_text,
direction)

        # Логируем статистику
        logger.info(f" Номер {most_common_text}
"
                    f"(голосов:
{count}/{len(acc['texts'])}, "
                    f"уверенность:
{avg_confidence:.2f}, "
                    f"направление: {direction})")

        # Очищаем накопитель
        del self.raw_recognitions[track_id]

        # Сохраняем в БД с направлением
        self.save_plate_to_db(most_common_text,
avg_confidence, track_id, direction)

        return most_common_text, avg_confidence

        best_text = text_counts.most_common(1)[0][0]
        best_conf = max([
            conf for text, conf in zip(acc['texts'],
acc['confidences'])
            if text == best_text
        ]) if acc['confidences'] else 0
        return best_text, best_conf

    return plate_text, confidence

    return None, confidence

def point_side_of_line(self, point, line_p1, line_p2):
    """Определяет, с какой стороны линии находится
точка"""
    # Векторное произведение
    return ((line_p2[0] - line_p1[0]) * (point[1] -
line_p1[1]) -

```

```

        (line_p2[1] - line_p1[1]) * (point[0] -
line_p1[0]))

    def segments_intersect(self, p1, p2, p3, p4):
        """Проверяет, пересекаются ли два отрезка"""
        def ccw(A, B, C):
            return (C[1] - A[1]) * (B[0] - A[0]) > (B[1] -
A[1]) * (C[0] - A[0])

        return ccw(p1, p3, p4) != ccw(p2, p3, p4) and
ccw(p1, p2, p3) != ccw(p1, p2, p4)

    def get_vehicle_centroid(self, box):
        """Получение центра бокса автомобиля"""
        x1, y1, x2, y2 = box
        return ((x1 + x2) // 2, (y1 + y2) // 2)

    def update_vehicle_position(self, track_id, centroid):
        """Обновление позиции автомобиля"""
        if track_id not in self.vehicle_positions:
            self.vehicle_positions[track_id] = []

        current_time = time.time()

self.vehicle_positions[track_id].append((centroid[0],
centroid[1], current_time))

        # Ограничиваем историю
        if len(self.vehicle_positions[track_id]) > 30:
            self.vehicle_positions[track_id].pop(0)

    def check_line_crossing(self, track_id,
current_centroid):
        """Проверка пересечения линий и определение
направления"""
        if track_id not in self.vehicle_positions:
            return None

        history = self.vehicle_positions[track_id]
        if len(history) < 2:
            return None

        # Предыдущая и текущая позиции
        prev = (history[-2][0], history[-2][1])
        curr = current_centroid
        current_time = time.time()

        # Инициализируем историю пересечений
        if track_id not in self.line_cross_history:
            self.line_cross_history[track_id] = {}

```

```

detected_direction = None

for line in self.lines:
    line_id = line['id']
    p1 = tuple(line['points'][0])
    p2 = tuple(line['points'][1])
    direction_config = line['direction'] # 'in',
'out', 'both'

    # Проверяем кулдаун
    last_cross =
self.line_cross_history[track_id].get(line_id, 0)
    if current_time - last_cross <
self.line_cross_cooldown:
        continue

    # Проверяем пересечение
    if not self.segments_intersect(prev, curr, p1,
p2):
        continue

    # Определяем сторону до и после пересечения
    prev_side = self.point_side_of_line(prev, p1,
p2)
    curr_side = self.point_side_of_line(curr, p1,
p2)

    # Определяем направление пересечения
    if prev_side < 0 and curr_side > 0:
        crossing_type = 'in' # Пересечение
"внутри"
    elif prev_side > 0 and curr_side < 0:
        crossing_type = 'out' # Пересечение
"наружу"
    else:
        continue # Не пересекает линию

    # Проверяем соответствие настройкам линии
    if direction_config != 'both' and crossing_type
!= direction_config:
        continue

    # Запоминаем пересечение
    self.line_cross_history[track_id][line_id] =
current_time

    # Определяем итоговое направление
    if crossing_type == 'in':
        detected_direction = 'enter'
        direction_text = 'ВЪЕЗД'
    else:

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        detected_direction = 'exit'
        direction_text = 'ВЫЕЗД'

        logger.info(f" Камера {self.camera_id}: Авто
#{track_id} - {direction_text} "
                    f"(линия: {line['name']}), тип:
{crossing_type})")

        # Сохраняем направление
        self.vehicle_directions[track_id] =
detected_direction

        break # Берем только первое пересечение

    return detected_direction

    def get_vehicle_direction(self, track_id,
current_centroid):
        """Получение направления движения автомобиля"""
        # Проверяем пересечение линий
        crossing_direction =
self.check_line_crossing(track_id, current_centroid)

        if crossing_direction:
            return crossing_direction

        # Если пересечения не было, возвращаем последнее
известное направление
        return self.vehicle_directions.get(track_id,
'unknown')

    def cleanup_old_recognitions(self):
        """Очистка старых нестабилизированных распознаваний
и кадров"""
        current_time = time.time()

        # Очистка накопителей распознаваний
        if hasattr(self, 'raw_recognitions') and
self.raw_recognitions:
            to_delete = []
            for track_id, acc in
self.raw_recognitions.items():
                if current_time - acc['last_update'] > 10.0:
                    to_delete.append(track_id)

            for track_id in to_delete:
                del self.raw_recognitions[track_id]
                logger.debug(f" Очищен накопитель для
track_id={track_id}")

            to_delete_vehicles = []

```

```

        for track_id, positions in
self.vehicle_positions.items():
            if positions and current_time - positions[-1][2]
> 10.0:
                to_delete_vehicles.append(track_id)

        for track_id in to_delete_vehicles:
            del self.vehicle_positions[track_id]
            if track_id in self.vehicle_directions:
                del self.vehicle_directions[track_id]
            if track_id in self.line_cross_history:
                del self.line_cross_history[track_id]
            logger.debug(f"Очищены позиции автомобиля
track_id={track_id}")

        # Очистка старых кадров
        if hasattr(self, 'best_frames') and
self.best_frames:
            to_delete_frames = []
            for track_id in list(self.best_frames.keys()):
                # Если трек больше не активен (нет в
raw_recognitions)
                    if not hasattr(self, 'raw_recognitions') or
track_id not in self.raw_recognitions:
                        # Проверяем, не в кэше ли финальный
результат
                            final_cache_key =
f"{self.camera_id}_{track_id}_final"
                            if final_cache_key not in
self.plate_cache:
                                to_delete_frames.append(track_id)

            for track_id in to_delete_frames:
                del self.best_frames[track_id]
                logger.debug(f" Очищен кадр для
track_id={track_id}")

def ensure_db_connection(self):
    """Проверяет БД и переподключает если нужно"""
    try:
        conn = self.db_pool.getconn()
        cursor = conn.cursor()
        cursor.execute("SELECT 1")
        cursor.close()
        self.db_pool.putconn(conn)
        return True
    except Exception as e:
        logger.warning(f" БД недоступна: {e},
переподключаю...")

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        # Закрываем старый пул
        try:
            self.db_pool.closeall()
        except:
            pass

        # 5 попыток переподключиться
        for attempt in range(5):
            try:
                import psycopg2.pool
                self.db_pool =
psycopg2.pool.SimpleConnectionPool(
                    1, 3, # УМЕНЬШИТЬ с 10 до 3
                    host=os.getenv('DB_HOST', 'db'),
                    port=os.getenv('DB_PORT', '5432'),
                    database=os.getenv('DB_NAME',
'vadb'),
                    user=os.getenv('DB_USER', 'vauser'),
                    password=os.getenv('DB_PASSWORD',
'vapassword'),
                    connect_timeout=5
                )
                logger.info(f" Переподключено! (попытка
{attempt+1}))")
                return True
            except Exception as e:
                logger.error(f" Попытка {attempt+1}/5:
{e}")

                if attempt < 4:
                    eventlet.sleep(2 * (attempt + 1))

                logger.error(" БД недоступна после 5 попыток")
                return False

        def open_gate(self, plate_text, vehicle_id):
            """Открытие шлагбаума для известного номера"""
            logger.info(f" Проезд разрешён: {plate_text},
vehicle_id={vehicle_id}")

        def save_plate_to_db(self, plate_text, confidence,
track_id=-1, direction='unknown'):
            """Сохранение распознанного номера в БД с учётом
направления и нечёткого поиска."""
            if not self.ensure_db_connection():
                logger.error(f" БД недоступна: {plate_text}")
                return

            conn = None
            cursor = None
            try:
                if not self.db_pool or not plate_text:
                    return

```

```

conn = self.db_pool.getconn()
cursor = conn.cursor()
current_time = datetime.now(timezone.utc)

# Проверяем, есть ли vehicle_id для этого номера
cursor.execute("SELECT id FROM vehicle WHERE
number = %s", (plate_text,))
vehicle_result = cursor.fetchone()
vehicle_id = vehicle_result[0] if vehicle_result
else None

if vehicle_id:
    logger.info(f"Номер {plate_text} найден в БД
(vehicle_id={vehicle_id}) - ОТКРЫВАЮ ШЛАГБАУМ!")
    self.open_gate(plate_text, vehicle_id)
else:
    logger.info(f" Номер {plate_text} не найден
в БД - шлагбаум не открыт")

# Получаем путь к фото, если есть
photo_path = None
if hasattr(self, 'best_frames') and track_id in
self.best_frames:
    photo_path = self.save_best_frame(track_id)
if direction == 'enter':
    employee_id = None
    if vehicle_id:
        cursor.execute("SELECT employee_id FROM
vehicle WHERE id = %s", (vehicle_id,))
        emp_result = cursor.fetchone()
        employee_id = emp_result[0] if
emp_result else None

        cursor.execute("""
        SELECT id FROM vehicle_event
        WHERE number = %s AND departure_time IS
NULL
        """, (plate_text,))
        if cursor.fetchone():
            logger.info(f" Пропуск повторного
въезда: {plate_text}")
        else:
            cursor.execute("""
            INSERT INTO vehicle_event
(vehicle_id, employee_id, number, arrival_time, arrival_photo)
VALUES (%s, %s, %s, %s, %s)
RETURNING id
            """, (vehicle_id, employee_id,
plate_text, current_time, photo_path))
            new_id = cursor.fetchone()[0]

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

120

```

        self.has_new_event = True
        self.number_event = plate_text
        logger.info(f" ВЪЕЗД: {plate_text}
(event_id={new_id}))")
        conn.commit()
        cursor.close()
        self.db_pool.putconn(conn)
        return
    if direction != 'exit':
        # Если направление неизвестно - ничего не
делаем (можно добавить логирование)
        conn.commit()
        cursor.close()
        self.db_pool.putconn(conn)
        return
    # --- 1. Точное совпадение ---
    cursor.execute("""
        SELECT id, arrival_time
        FROM vehicle_event
        WHERE number = %s AND departure_time IS NULL
        ORDER BY arrival_time DESC
        LIMIT 1
        """, (plate_text,))
    exact = cursor.fetchone()

    if exact:
        event_id, arrival_time = exact
        stay_duration = current_time - arrival_time
        cursor.execute("""
            UPDATE vehicle_event
            SET departure_time = %s, stay_duration =
%s, departure_photo = %s
            WHERE id = %s
            """, (current_time, stay_duration,
photo_path, event_id))
        self.has_new_event = True
        self.number_event = plate_text
        logger.info(f"ВЪЕЗД (точное совпадение):
{plate_text} (event_id={event_id})")
        conn.commit()
        cursor.close()
        self.db_pool.putconn(conn)
        return
    MAX_DISTANCE = 1
    MAX_AGE_HOURS =
int(os.getenv('OPEN_EVENT_MAX_AGE_HOURS', '24'))
    time_limit = current_time -
timedelta(hours=MAX_AGE_HOURS)
    length = len(plate_text)

```

```

# Фильтруем кандидатов: длина ±1 и (опционально)
совпадение первого символа
# Первый символ редко ошибается, это ускоряет
поиск

cursor.execute("""
    SELECT id, number, arrival_time
    FROM vehicle_event
    WHERE departure_time IS NULL
    AND arrival_time >= %s
    AND LENGTH(number) BETWEEN %s AND %s
    AND LEFT(number, 1) = %s
    ORDER BY arrival_time DESC
""", (time_limit, length - 1, length + 1,
plate_text[0] if plate_text else ''))

candidates = cursor.fetchall()
best_match = None
best_distance = MAX_DISTANCE + 1
best_arrival = None

for ev_id, ev_number, ev_arrival in candidates:
    if ev_number == plate_text: # на случай,
если точное не сработало
        best_match = (ev_id, ev_number,
ev_arrival)
        best_distance = 0
        break
    dist =
self._levenshtein_distance(plate_text, ev_number)
    if dist <= MAX_DISTANCE:
        if dist < best_distance or (dist ==
best_distance and (best_arrival is None or ev_arrival >
best_arrival)):
            best_match = (ev_id, ev_number,
ev_arrival)
            best_distance = dist
            best_arrival = ev_arrival

if best_match:
    event_id, matched_number, arrival_time =
best_match
    stay_duration = current_time - arrival_time
    cursor.execute("""
        UPDATE vehicle_event
        SET departure_time = %s, stay_duration =
%s, departure_photo = %s
        WHERE id = %s
""", (current_time, stay_duration,
photo_path, event_id))
    self.has_new_event = True
    self.number_event = plate_text

```

```

        logger.info(f" ВЫЕЗД (нечёткое, расстояние
{best_distance}): "
                    f"{plate_text} сопоставлен с
{matched_number} (event_id={event_id})")
        conn.commit()
        cursor.close()
        self.db_pool.putconn(conn)
        return
    cursor.execute("""
        SELECT id, departure_time
        FROM vehicle_event
        WHERE number = %s AND departure_time IS NOT
NULL

        ORDER BY departure_time DESC
        LIMIT 1
    """, (plate_text,))
    last_dep = cursor.fetchone()

    if last_dep:
        time_since = (current_time -
last_dep[1]).total_seconds()
        if time_since < 30:
            logger.info(f"Пропуск выезда (предыдущий
был {time_since:.0f} сек назад): {plate_text}")
            conn.commit()
            cursor.close()
            self.db_pool.putconn(conn)
            return

        # Создаём выезд без въезда
        employee_id = None
        if vehicle_id:
            cursor.execute("SELECT employee_id FROM
vehicle WHERE id = %s", (vehicle_id,))
            emp_result = cursor.fetchone()
            employee_id = emp_result[0] if emp_result
        else None

        cursor.execute("""
            INSERT INTO vehicle_event (vehicle_id,
employee_id, number, departure_time, departure_photo)
            VALUES (%s, %s, %s, %s, %s)
            RETURNING id
        """, (vehicle_id, employee_id, plate_text,
current_time, photo_path))
        event_id = cursor.fetchone()[0]
        self.has_new_event = True
        self.number_event = plate_text
        logger.warning(f"ВЫЕЗД без въезда: {plate_text}
(event_id={event_id})")

```

```

        conn.commit()
        cursor.close()
        self.db_pool.putconn(conn)

except Exception as e:
    logger.error(f"Ошибка сохранения номера в БД:
{e}")

    try:
        if conn:
            conn.rollback()
            cursor.close()
            self.db_pool.putconn(conn)
    except:
        pass

def _levenshtein_distance(self, s1: str, s2: str) ->
int:
    """Вычисляет расстояние Левенштейна между двумя
строками."""
    if len(s1) < len(s2):
        return self._levenshtein_distance(s2, s1)
    if len(s2) == 0:
        return len(s1)
    prev_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        curr_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = prev_row[j + 1] + 1
            deletions = curr_row[j] + 1
            substitutions = prev_row[j] + (c1 != c2)
            curr_row.append(min(insertions, deletions,
substitutions))
        prev_row = curr_row
    return prev_row[-1]

def save_best_frame(self, track_id):
    """Сохранение лучшего полного кадра для номера"""
    if track_id not in self.best_frames:
        return None

    if len(self.best_frames[track_id]) == 4:
        frame, confidence, plate_text, direction =
self.best_frames[track_id]
    else:
        frame, confidence, plate_text =
self.best_frames[track_id]
        direction = 'unknown'

    try:
        # Очищаем номер от недопустимых символов для
имени файла

```

```

        safe_plate = plate_text.replace('/',
'_').replace('\\', '_').replace(':', '_').replace(' ', '_')

        timestamp =
datetime.now().strftime("%Y%m%d_%H%M%S")
        #filename =
f"cam{self.camera_id}_track{track_id}_plate_{safe_plate}_conf{con
fidence:.2f}_{timestamp}.jpg"
        filename =
f"cam{self.camera_id}_{timestamp}_plate_{safe_plate}.jpg"
        filepath = os.path.join(self.full_frames_folder,
filename)
        filepath1 =
os.path.join(self.full_frames_folder_1, filename)

        # Рисуем информацию на кадре
annotated_frame = frame.copy()

        # Добавляем текст с информацией
info_texts = [
            f"Camera: {self.camera_id}",
            f"Plate: {plate_text}",
            f"Confidence: {confidence:.2f}"
        ]

        y_offset = 100
        for text in info_texts:
            cv2.putText(annotated_frame, text, (10,
y_offset), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
            y_offset += 25

        cv2.imwrite(filepath, annotated_frame)
        cv2.imwrite(filepath1, annotated_frame)
        logger.info(f" Сохранен полный кадр:
{filename}")

        # Очищаем после сохранения
del self.best_frames[track_id]

        return f"/media_backend/detections/{filename}"

    except Exception as e:
        logger.error(f"Ошибка сохранения полного кадра:
{e}")

        return None

    def process_frame(self, frame):
        """Максимально быстрая обработка кадра"""
        try:
            recognized_plates = []
            original_frame = frame.copy()

```

```

self.current_frame = frame.copy()

for line in self.lines:
    p1 = tuple(line['points'][0])
    p2 = tuple(line['points'][1])

    # Цвет линии зависит от направления
    if line['direction'] == 'in':
        color = (0, 255, 0) # Зеленый - въезд
    elif line['direction'] == 'out':
        color = (0, 0, 255) # Красный - выезд
    else:
        color = (255, 0, 0) # Синий - оба
направления

    cv2.line(original_frame, p1, p2, color, 2)

    # Подписываем линию
    mx, my = (p1[0] + p2[0]) // 2, (p1[1] +
p2[1]) // 2

    label = f"({line['direction']})"
    cv2.putText(original_frame, label, (mx, my -
10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# === ЭТАП 1: Поиск автомобилей с трекингом ===
vehicle_results = self.vehicle_model.track(
    frame,
    classes=self.vehicle_classes,
    conf=self.vehicle_conf_threshold,
    verbose=False,
    persist=True
)

if vehicle_results[0].boxes is not None:
    vehicle_boxes =
vehicle_results[0].boxes.xyxy.cpu().numpy()

    # Получаем track_id для автомобилей
    if vehicle_results[0].boxes.id is not None:
        vehicle_track_ids =
vehicle_results[0].boxes.id.int().cpu().tolist()
    else:
        vehicle_track_ids = [-1] *
len(vehicle_boxes)

    for v_idx, v_box in
enumerate(vehicle_boxes):
        vx1, vy1, vx2, vy2 = map(int, v_box)

```

```

100:         if (vx2 - vx1) < 100 or (vy2 - vy1) <
                continue

        vehicle_track_id =
vehicle_track_ids[v_idx]

        centroid =
self.get_vehicle_centroid((vx1, vy1, vx2, vy2))

self.update_vehicle_position(vehicle_track_id, centroid)
        direction =
self.get_vehicle_direction(vehicle_track_id, centroid)

        # Цвет в зависимости от направления
        if direction == 'enter':
            color = (0, 255, 0) # Зеленый -
въезд

            dir_text = "ENTER"
        elif direction == 'exit':
            color = (0, 0, 255) # Красный -
выезд

            dir_text = "EXIT"
        else:
            color = (255, 255, 0) # Желтый -
неизвестно

            dir_text = "?"

        # Отрисовка с ID и направлением
        cv2.rectangle(original_frame, (vx1,
vy1), (vx2, vy2), color, 2)

        label = f"Car #{vehicle_track_id}
[{dir_text}]"

        label_size = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 2)[0]
        cv2.rectangle(original_frame,
                        (vx1, vy1 - label_size[1] -
10),
                        (vx1 + label_size[0], vy1),
                        color, -1)
        cv2.putText(original_frame, label,
                    (vx1, vy1 - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)

        '''
        direction =
self.vehicle_directions.get(vehicle_track_id, 'unknown')

        # OCR только если направление определено

```

```

        should_recognize = (not
self.recognize_only_with_direction) or (direction in ['enter',
'exit'])

        # === ЭТАП 2: Поиск номеров (только если
нужно) ===

        if not should_recognize:
            continue
        '''

        vehicle_roi = frame[vy1:vy2, vx1:vx2]

        if vehicle_roi.size > 0:
            # Убрали .track() - теперь обычный
predict

            plate_results = self.plate_model(
                vehicle_roi,
                classes=self.detect_classes,
                verbose=False,
                conf=self.plate_conf_threshold
            )

            if plate_results[0].boxes is not
None:

                plate_boxes =
plate_results[0].boxes.xyxy.cpu().numpy()
                plate_confs =
plate_results[0].boxes.conf.cpu().numpy()
                for p_idx, (p_box, p_conf) in
enumerate(zip(plate_boxes, plate_confs)):
                    px1, py1, px2, py2 =
map(int, p_box)

                    plate_width = px2 - px1
                    plate_height = py2 - py1
                    if (plate_width < 35 or
plate_height < 13) or plate_width <= plate_height + 5:
                        continue
                    global_px1 = vx1 + px1
                    global_py1 = vy1 + py1
                    global_px2 = vx1 + px2
                    global_py2 = vy1 + py2
                    plate_roi =
frame[global_py1:global_py2, global_px1:global_px2]

                    if plate_roi.size > 0 and
self.enable_ocr and self.ocr:

                        # Используем ID
автомобиля + индекс номера как track_id
                        track_id =
f"{vehicle_track_id}_{p_idx}" if vehicle_track_id != -1 else
p_idx

```

Изм.	Лист	№ докум.	Подпись	Дата

```

# Преобразуем строку в
число для совместимости
hash(track_id) % 1000000

track_id_hash =

plate_text, ocr_conf =
self.recognize_license_plate(plate_roi, track_id_hash,
vehicle_track_id)

if plate_text:

recognized_plates.append({
    'text':
plate_text,
    'confidence':
ocr_conf,
    'box':
[global_px1, global_py1, global_px2, global_py2],
    'detection_confidence': float(p_conf),
    'track_id':
track_id_hash,
    'vehicle_id':
vehicle_track_id,
    'direction':
direction
})

cv2.rectangle(original_frame, (global_px1, global_py1),
(global_px2, global_py2), (0, 255, 0), 2)
label =
f"[Car{vehicle_track_id}] {plate_text} ({ocr_conf:.2f})"
label_size =
cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)[0]
cv2.rectangle(original_frame,
(global_px1, global_py1 - label_size[1] - 10),
(global_px1 + label_size[0], global_py1),
(0, 255,
0), -1)
cv2.putText(original_frame, label,
(global_px1,
global_py1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)

# Кодируем и отправляем

```

```

        encode_params = [cv2.IMWRITE_JPEG_QUALITY,
self.jpeg_quality]
        _, buffer = cv2.imencode('.jpg', original_frame,
encode_params)
        frame_base64 =
base64.b64encode(buffer).decode('utf-8')

        sio.emit(f'camera_{self.camera_id}_frame', {
            'frame': frame_base64,
            'total_detections': self.total_detections,
            'recognized_plates': recognized_plates,
            'timestamp': time.time(),
            'event': self.has_new_event,
            'number_event': self.number_event,
            'frame_number': self.frame_count,
            'resolution':
f"{original_frame.shape[1]}x{original_frame.shape[0]}"
        }, room=f'camera_{self.camera_id}')

        self.has_new_event = False
        self.number_event = "unknown"

    except Exception as e:
        logger.error(f"Ошибка обработки кадра: {e}")

def run(self):
    """Основной цикл обработки камеры с GStreamer и
детальными логами"""
    cap = None
    reconnect_attempts = 0
    max_reconnect_attempts = 5

    # Настройки GStreamer
    GST_LATENCY = int(os.getenv('GST_LATENCY', '100'))

    gst_pipeline = (
        f'rtspsrc location={self.rtsp_url} '
        f'latency={GST_LATENCY} '
        f'drop-on-latency=true '
        f'config-interval=1 '
        f'protocols=tcp '
        f'! rtph265depay '
        f'! h265parse config-interval=-1 '
        f'! avdec_h265 '
        f'! videoconvert '
        f'! video/x-raw,format=BGR '
        f'! appsink sync=false max-buffers=1 drop=true'
    )

    # Убираем лишние пробелы в пайплайне
    gst_pipeline = ' '.join(gst_pipeline.split())

```

```

        #logger.info(f" Пайплайн: {gst_pipeline}")
        cleanup_counter = 0
        while self.running:
            try:
                # Подключение к RTSP потоку
                if cap is None:
                    cap = cv2.VideoCapture(gst_pipeline,
cv2.CAP_GSTREAMER)

                    #cap = cv2.VideoCapture(self.rtsp_url)
                    #cap.set(cv2.CAP_PROP_BUFFERSIZE, 10)
                    # Проверяем подключение
                    if not cap.isOpened():
                        if cap:
                            logger.error(f"Подключние к
потоку ужасно")

                            cap.release()
                            cap = None
                            eventlet.sleep(5) # Пауза перед
повторным подключением

                                continue
                                # Получаем информацию о потоке
                                width =
int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                                height =
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                                fps = cap.get(cv2.CAP_PROP_FPS)
                                reconnect_attempts = 0
                                ret, frame = cap.read()

                                if not ret:
                                    cap.release()
                                    cap = None
                                    eventlet.sleep(1)
                                    continue
                                    self.frame_count += 1
                                    cleanup_counter += 1
                                    self.process_frame(frame.copy())
                                    if cleanup_counter >= 100:
                                        self.cleanup_old_recognitions()
                                        cleanup_counter = 0
                                        eventlet.sleep(0)
                                except Exception as e:
                                    if cap:
                                        cap.release()
                                        cap = None
                                        eventlet.sleep(5)

                                if cap:
                                    cap.release()

def stop(self):
    """Остановка камеры"""

```

					ПГУ1.090301.14.001 ПЗ	Лист
						131
Изм.	Лист	№ докум.	Подпись	Дата		

```

        self.running = False
class VideoStreamManager:
    """Менеджер видеопотоков"""

    def __init__(self):
        self.cameras = {}
        self.greenlets = {}
        self.model = None
        self.vehicle_model = None
        self.ocr = None
        self.db_pool = None
        self.running = True
        self.executor = ThreadPoolExecutor(max_workers=4) #
Для тяжелых операций

        self.init_db_pool()
        self.load_model()
        self.load_ocr_model()
        self.load_cameras()

    def init_db_pool(self):
        """Создание пула соединений PostgreSQL"""
        try:
            import psycopg2.pool

            self.db_pool =
psycopg2.pool.SimpleConnectionPool(
                1, # минимум соединений
                3, # максимум соединений
                host=os.getenv('DB_HOST', 'db'),
                port=os.getenv('DB_PORT', '5432'),
                database=os.getenv('DB_NAME', 'vadb'),
                user=os.getenv('DB_USER', 'vauser'),
                password=os.getenv('DB_PASSWORD',
'vapassword'),
                connect_timeout=5
            )
            logger.info(" Пул подключений PostgreSQL
создан")

            # Проверяем подключение
            conn = self.db_pool.getconn()
            cursor = conn.cursor()
            cursor.execute("SELECT 1")
            cursor.close()
            self.db_pool.putconn(conn)
            logger.info(" Проверка подключения к PostgreSQL
успешна")

        except Exception as e:

```

```

        logger.error(f" Ошибка создания пула PostgreSQL:
{e}")

        sys.exit(1)

    def load_model(self):
        """Загрузка YOLO модели"""
        logger.info(" Загрузка YOLO моделей")

        model_path = '/home/admin/best.pt'
        if not os.path.exists(model_path):
            logger.warning(f"Модель не найдена по пути
{model_path}, загружаем из интернета")
            model_path = 'best.pt'

        try:
            self.model = YOLO(model_path)
            self.model.to('cuda')
            if hasattr(self.model, 'names'):
                logger.info(f" Классы в модели:
{self.model.names}")
                logger.info(f"   Всего классов:
{len(self.model.names)}")

                # Если есть класс 'license_plate' или
'number_plate'
                for idx, name in self.model.names.items():
                    if 'plate' in name.lower() or 'number'
in name.lower():
                        logger.info(f"   Найден класс
номерного знака: {idx} - {name}")

                logger.info(" Модель для номеров загружена")

            vehicle_model_path = '/home/admin/yolov8m.pt'
            if not os.path.exists(vehicle_model_path):
                logger.info(" Скачивание YOLOv8m модели...")
                self.vehicle_model = YOLO('yolov8m.pt')
            else:
                self.vehicle_model =
YOLO(vehicle_model_path)

            self.vehicle_model.to('cuda')
            logger.info(" Модель для автомобилей (YOLOv8m)
загружена")

            # Выводим информацию о классах модели
автомобилей
            if hasattr(self.vehicle_model, 'names'):
                # Классы автомобилей в COCO: 2-car, 3-
motorcycle, 5-bus, 7-truck

```

```

        vehicle_classes = {k: v for k, v in
self.vehicle_model.names.items()
                                if k in [2, 3, 5, 7]}
        logger.info(f" Классы транспорта в модели:
{vehicle_classes}")
    except Exception as e:
        logger.error(f" Ошибка загрузки модели: {e}")
        sys.exit(1)

    def load_ocr_model(self):
        """Загрузка CRNN модели для распознавания номеров"""
        ocr_model_path = os.getenv('OCR_MODEL_PATH',
'/home/admin/crnn_ocr_model_best.pth ')
        self.ocr = LicensePlateOCR(ocr_model_path)

    def load_cameras(self):
        """Загрузка камер из БД"""
        try:
            conn = self.db_pool.getconn()
            cursor = conn.cursor()
            cursor.execute("""
                SELECT id, name, url, detection_type,
coordinates, is_active
                FROM camera
                WHERE is_active = true
                AND url IS NOT NULL
                AND url != ''
                AND detection_type = '0'
                ORDER BY id
            """)

            cameras_data = cursor.fetchall()
            cursor.close()
            self.db_pool.putconn(conn)

            if not cameras_data:
                logger.warning("⚠️ Активные камеры не
найденны")
                return

            logger.info(f" Найдено активных камер:
{len(cameras_data)}")

            for row in cameras_data:
                camera_id = row[0]

                lines = self.load_camera_lines(camera_id)

                camera_data = {
                    'id': camera_id,

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        'name': row[1],
        'url': row[2],
        'detection_type': str(row[3]) if row[3]
is not None else '1',
        'coordinates': row[4],
        'is_active': row[5],
        'lines': lines # Добавляем линии
    }

    streamer = CameraStreamer(camera_data,
self.model, self.vehicle_model, self.db_pool)
    streamer.ocr = self.ocr
    self.cameras[camera_id] = streamer
    logger.info(f" - Камера {camera_id}:
{row[1]} (линий: {len(lines)})")

    except Exception as e:
        logger.error(f" Ошибка загрузки камер: {e}")
        import traceback
        traceback.print_exc()

def load_camera_lines(self, camera_id):
    """Загрузка линий для камеры из БД"""
    lines = []
    try:
        conn = self.db_pool.getconn()
        cursor = conn.cursor()

        cursor.execute("""
            SELECT id, name, points, direction, color,
is_active
            FROM camera_line
            WHERE camera_id = %s AND is_active = true
            ORDER BY id
            """, (camera_id,))

        rows = cursor.fetchall()
        cursor.close()
        self.db_pool.putconn(conn)

        for row in rows:
            line_id, name, points_json, direction,
color, is_active = row

            # Парсим JSON с точками
            if isinstance(points_json, str):
                points = json.loads(points_json)
            else:
                points = points_json

```

```

        # Проверяем, что у нас 2 точки
        if len(points) != 2:
            logger.warning(f"Линия {line_id} имеет
неверное количество точек: {len(points)}")
            continue

        lines.append({
            'id': line_id,
            'name': name,
            'points': points,  # [[x1, y1], [x2,
y2]]
            'direction': direction or 'both',  #
'in', 'out', 'both'
            'color': color,
            'is_active': is_active
        })

        logger.info(f"    Линия {line_id}: {name}
(direction: {direction})")

    except Exception as e:
        logger.error(f"Ошибка загрузки линий для камеры
{camera_id}: {e}")
        if 'conn' in locals():
            self.db_pool.putconn(conn)

    return lines

def start_all_cameras(self):
    """Запуск обработки всех камер"""
    if not self.cameras:
        logger.warning("Нет камер для запуска")
        return

    for camera_id, streamer in self.cameras.items():
        # Используем eventlet.spawn вместо threading
        greenlet = eventlet.spawn(streamer.run)
        self.greenlets[camera_id] = greenlet
        logger.info(f" Запущена камера {camera_id}")

def stop_all_cameras(self):
    """Остановка всех камер"""
    logger.info(" Остановка всех камер...")

    for camera_id, streamer in self.cameras.items():
        streamer.running = False

    # Ждем завершения всех greenlet'ов
    for camera_id, greenlet in self.greenlets.items():
        try:
            greenlet.wait(timeout=5)

```

```

        except:
            pass

    if self.db_pool:
        self.db_pool.closeall()

    if self.executor:
        self.executor.shutdown(wait=False)

    self.running = False
    logger.info(" Все камеры остановлены")

# Socket.IO события
@sio.event
def connect(sid, environ, auth=None):
    """Подключение клиента"""
    #logger.info(f" Клиент подключился: {sid}")
    return True

@sio.event
def disconnect(sid):
    """Отключение клиента"""
    #logger.info(f" Клиент отключился: {sid}")

    # Очищаем комнаты клиента
    rooms = sio.rooms(sid)
    for room in rooms:
        if room != sid and room.startswith('camera_'):
            sio.leave_room(sid, room)

@sio.event
def subscribe(sid, data):
    """Подписка на камеру"""
    camera_id = data.get('cameraId')
    if not camera_id:
        return {'error': 'cameraId required'}

    # Проверяем, существует ли камера
    if not stream_manager or camera_id not in
stream_manager.cameras:
        return {'error': f'Camera {camera_id} not found'}

    # Добавляем клиента в комнату камеры
    room_name = f'camera_{camera_id}'
    sio.enter_room(sid, room_name)

    #logger.info(f" Клиент {sid} подписался на камеру
{camera_id}")

```

```

# Отправляем приветственное сообщение
sio.emit('subscribed', {
    'cameraId': camera_id,
    'message': f'Subscribed to camera {camera_id}'
}, room=sid)

return {
    'success': True,
    'cameraId': camera_id,
    'message': f'Subscribed to camera {camera_id}'
}

@sio.event
def unsubscribe(sid, data):
    """Отписка от камеры"""
    camera_id = data.get('cameraId')
    if not camera_id:
        return {'error': 'cameraId required'}

    room_name = f'camera_{camera_id}'
    sio.leave_room(sid, room_name)

    logger.info(f" Клиент {sid} отписался от камеры {camera_id}")

    return {'success': True, 'cameraId': camera_id}

@sio.event
def get_cameras_list(sid, data):
    """Получение списка активных камер"""
    if not stream_manager:
        return {'error': 'Stream manager not initialized'}

    cameras_list = []
    for camera_id, streamer in
stream_manager.cameras.items():
        cameras_list.append({
            'id': camera_id,
            'name': streamer.camera_name,
            'detection_type': streamer.detection_type,
            'status': 'online' if streamer.running else
'offline',
            'frame_count': streamer.frame_count,
            'total_detections': streamer.total_detections
        })

    return {'cameras': cameras_list}

```

```

@sio.event
def get_camera_status(sid, data):
    """Получение статуса конкретной камеры"""
    camera_id = data.get('cameraId')
    if not camera_id or not stream_manager or camera_id not
in stream_manager.cameras:
        return {'error': 'Camera not found'}

    streamer = stream_manager.cameras[camera_id]
    return {
        'cameraId': camera_id,
        'name': streamer.camera_name,
        'status': 'online' if streamer.running else
'offline',
        'frame_count': streamer.frame_count,
        'total_detections': streamer.total_detections,
        'detection_type': streamer.detection_type
    }

# Глобальный менеджер
stream_manager = None

def signal_handler(sig, frame):
    """Обработчик сигналов для graceful shutdown"""
    logger.info(f"\n Получен сигнал {sig}")
    if stream_manager:
        stream_manager.stop_all_cameras()
    sys.exit(0)

def main():
    global stream_manager

    logger.info("=" * 50)
    logger.info(" Видеостриминг сервер запускается...")
    logger.info("=" * 50)

    # Регистрируем обработчики сигналов
    signal.signal(signal.SIGINT, signal_handler)
    signal.signal(signal.SIGTERM, signal_handler)

    try:
        # Инициализируем менеджер
        stream_manager = VideoStreamManager()

        # Запускаем все камеры
        stream_manager.start_all_cameras()

```

```

5000")

# Запускаем Socket.IO сервер
logger.info(" Запуск Socket.IO сервера на порту

# Создаем сокет и запускаем сервер
sock = eventlet.listen(('0.0.0.0', 5000))

# Запускаем сервер с логгированием
wsgi.server(
    sock,
    app,
    log_output=False,
    debug=False,
    keepalive=65
)

except KeyboardInterrupt:
    logger.info("\n Получен сигнал остановки")
except Exception as e:
    logger.error(f"Критическая ошибка: {e}")
    import traceback
    traceback.print_exc()
finally:
    if stream_manager:
        stream_manager.stop_all_cameras()
    logger.info(" Сервер остановлен")

if __name__ == "__main__":
    main()

```

Файл Dockerfile

```

FROM pytorch/pytorch:2.9.0-cuda12.8-cudnn9-runtime

ENV DEBIAN_FRONTEND=noninteractive
ENV PYTHONUNBUFFERED=1

RUN apt-get update && apt-get install -y \
    g++ \
    build-essential \
    cmake \
    git \
    pkg-config \
    ffmpeg \
    libgl1 \
    libglib2.0-0 \
    libsm6 \
    libxext6 \
    libxrender1 \
    libgomp1 \

```

```

libjpeg-dev \
libpng-dev \
libtiff-dev \
libavcodec-dev \
libavformat-dev \
libswscale-dev \
libv4l-dev \
libxvidcore-dev \
libx264-dev \
libgtk-3-dev \
libatlas-base-dev \
gfortran \
python3-dev \
gstreamer1.0-tools \
gstreamer1.0-libav \
gstreamer1.0-plugins-base \
gstreamer1.0-plugins-good \
gstreamer1.0-plugins-bad \
gstreamer1.0-plugins-ugly \
libgstreamer1.0-dev \
libgstreamer-plugins-base1.0-dev \
&& rm -rf /var/lib/apt/lists/*

```

```

RUN useradd -m -u 1000 -s /bin/bash admin
WORKDIR /home/admin

```

```

WORKDIR /opt

```

```

RUN git clone --depth 1 https://github.com/opencv/opencv.git
RUN git clone --depth 1
https://github.com/opencv/opencv_contrib.git

```

```

WORKDIR /opt/opencv/build

```

```

RUN cmake \
  -D CMAKE_BUILD_TYPE=Release \
  -D CMAKE_INSTALL_PREFIX=/opt/conda \
  -D OPENCV_EXTRA_MODULES_PATH=/opt/opencv_contrib/modules
\
  -D WITH_GSTREAMER=ON \
  -D WITH_FFMPEG=ON \
  -D WITH_CUDA=OFF \
  -D BUILD_opencv_python3=ON \
  -D BUILD_TESTS=OFF \
  -D BUILD_EXAMPLES=OFF \
  -D BUILD_DOCS=OFF \
  -D BUILD_PERF_TESTS=OFF \
  -D WITH_GAPI=OFF \
  -D BUILD_opencv_gapi=OFF \
  ..

```

```

RUN make -j$(nproc)

RUN make install

RUN ldconfig

RUN python -c "import cv2; print(cv2.getBuildInformation())"

RUN pip install --no-cache-dir \
    ultralytics==8.3.75 \
    python-dotenv==1.0.1 \
    numpy==1.26.4 \
    pandas==2.2.0 \
    tqdm==4.66.0 \
    loguru==0.7.2 \
    psycpg2-binary==2.9.9 \
    python-socketio==5.9.0 \
    eventlet==0.33.3 \
    python-engineio>=4.7.0 \
    python-jose==3.3.0

RUN pip install --no-cache-dir --default-timeout=300 lapx

RUN rm -rf /opt/conda/lib/python3.11/site-packages/cv2/gapi
COPY requirements.txt /home/admin/requirements.txt
COPY best.pt /home/admin/best.pt
COPY yolov8m.pt /home/admin/yolov8m.pt
COPY video_stream_server.py
/home/admin/video_stream_server.py
COPY
crnn_ocr_model_best.pth /home/admin/crnn_ocr_model_best.pth

RUN usermod -a -G 33 admin

# Делаем admin владельцем файлов
RUN chown -R admin:admin /home/admin

# Переключаемся на пользователя admin
USER admin
WORKDIR /home/admin

# Запуск по умолчанию
CMD ["python", "video_stream_server.py"]

```

Приложение В

(обязательное)

Система видеоаналитики автоматического контроля и учета автотранспорта с
использованием методов компьютерного зрения

Презентация

					ПГУ1.090301.14.001 ПЗ	Лист
						143
Изм.	Лист	№ докум.	Подпись	Дата		

Система видеоаналитики автоматического контроля и учёта автотранспорта с использованием методов компьютерного зрения

Автор: студент группы 22ВВП1 Китаев Ярослав Евгеньевич

Руководитель: д.т.н., профессор кафедры «Вычислительная техника» Гурин Евгений Иванович

Актуальность выбранной темы

Исключение человеческого фактора

Охранник вручную сверяет списки, что приводит к ошибкам и задержкам

Очереди на въезд

При большом потоке машин ручная проверка создаёт пробки на КПП

Слабый учёт

Невозможно автоматически вести точный журнал всех въездов и выездов

Разработанная система решает эти проблемы

2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

144

Обзор аналогов

UniServer AUTO



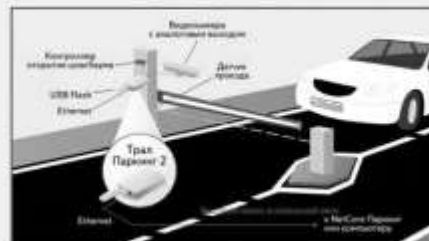
DVS-102-UN (RusGuard)



Автомаршал



Трал Паркинг 2



Сравнение аналогов

Продукт	Подход	GUI	Масштабируемость	Открытый код	Стоимость
UniServer AUTO	Серверный	✓	✓	✗	Платная
Автомаршал	Серверный	✓	✓	✗	Платная
DVS-102-UN	Автономный	✗	✗	✗	Платная
Трал Паркинг 2	Автономный	✗	✗	✗	Платная
Разрабатываемая система	Серверный	✓	✓	✓	Открытая

Постановка задачи

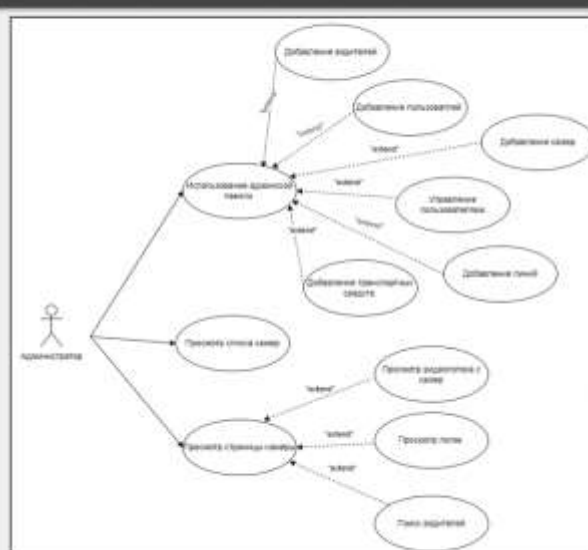
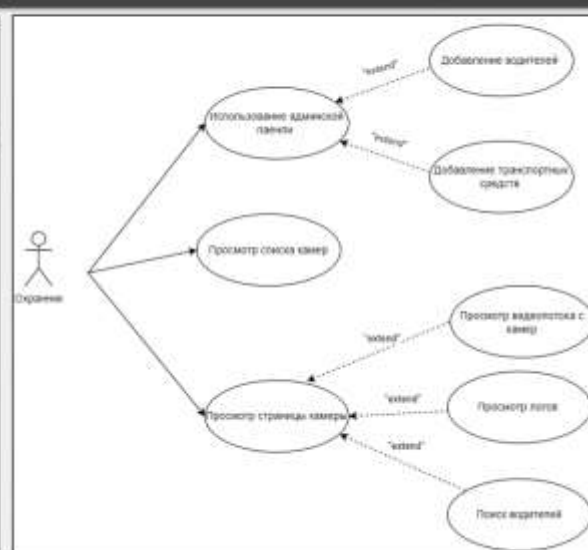
Необходимо разработать систему видеоаналитики автоматического контроля и учёта автотранспорта на охраняемой территории с использованием методов компьютерного зрения.

Она должна обеспечивать следующий функционал:

- автоматически распознавать государственные регистрационные номера
- фиксацию времени въезда/выезда
- верификацию распознанных номеров с базой данных разрешённых автомобильных номеров;
- представление удобного интерфейса для использования.

5

Диаграммы вариантов использования



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

146

Используемые технологии



django



docker.



7

Преимущества разрабатываемой системы

Нейросетевое ядро

Двухэтапная детекция: YOLOv8m → YOLOv8n → CRNN.
mAP@0.5 = 0.988

Безопасность

JWT-авторизация, 2 уровня доступа: охранник,
администратор

Веб-интерфейс

React SPA с просмотром видео в реальном времени
через WebSocket

Контейнеризация

Развёртывание одной командой: `docker compose up -d`

8

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

147

Выбор модели для распознавания автотранспорта

Сравнение моделей YOLOv8

Модель	Параметры	mAP50-95
YOLOv8n	3.2M	37.3
YOLOv8s	11.2M	44.9
YOLOv8m ✓	25.9M	50.2
YOLOv8l	43.7M	52.9
YOLOv8x	68.2M	53.9

В результате сравнения моделей была выбрана YOLOv8m, потому что она показывает оптимальный баланс между точностью и скоростью детекции

9

Результаты обучения модели для выделения области номерного знака



0.988	mAP@0.5
0.854	mAP 0.5-0.95
0.980	F1
0.976	Recall

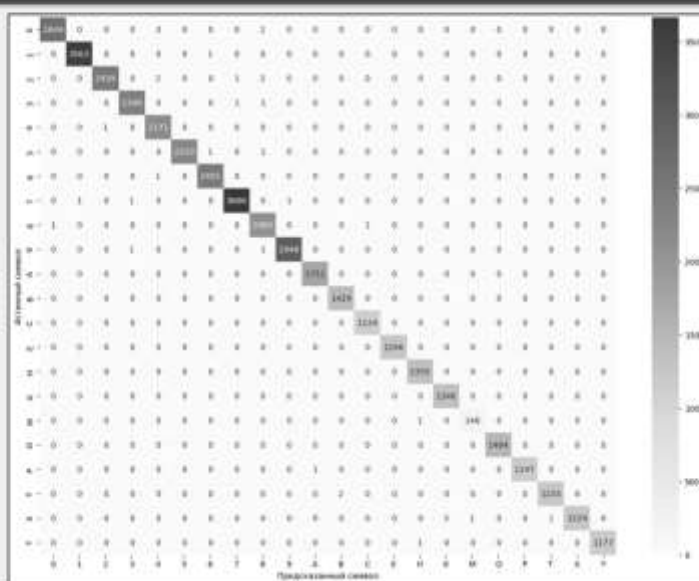
10

Пример работы модели для выделения области номерного знака



11

Результаты обучения CRNN



12

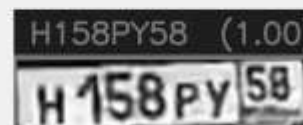
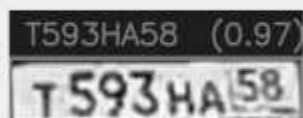
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

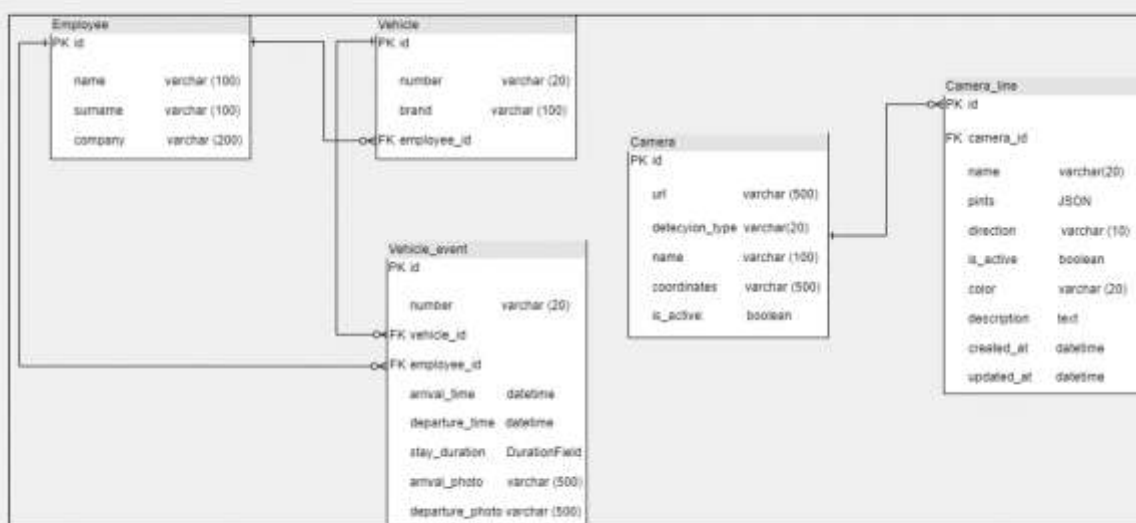
149

Пример считывания символов



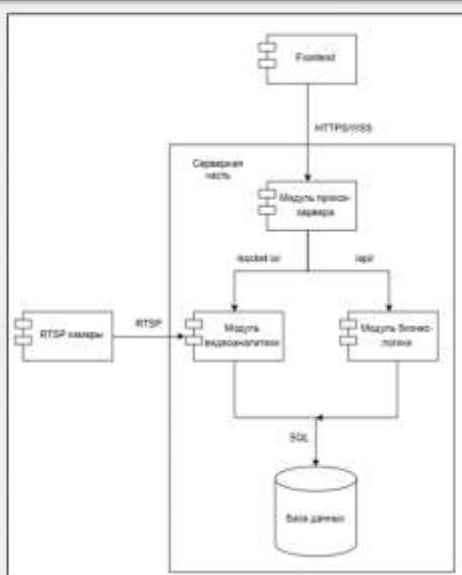
13

Схема базы данных



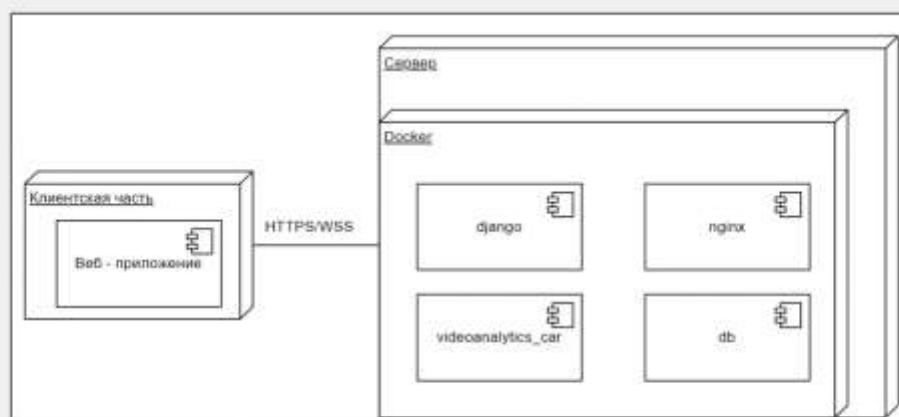
14

Архитектура системы



15

Развертывание системы



16

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.14.001 ПЗ

Лист

151

Спасибо за внимание!

					ПГУ1.090301.14.001 ПЗ	Лист
						152
Изм.	Лист	№ докум.	Подпись	Дата		