

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«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

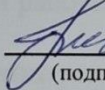
СИСТЕМА ВИДЕОАНАЛИТИКИ
АВТОМАТИЧЕСКОГО КОНТРОЛЯ И УЧЕТА
РАБОЧЕГО ВРЕМЕНИ СОТРУДНИКОВ С
ИСПОЛЬЗОВАНИЕМ КОМПЬЮТЕРНОГО ЗРЕНИЯ

Студент

 27.05.26
(подпись, дата)

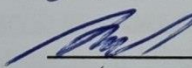
Рябов Святослав Игоревич
(ФИО полностью)

Руководитель

 28.05.26
(подпись, дата)

Митрохина Н.Ю.
(фамилия, инициалы)

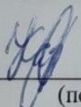
Нормоконтролёр

 28.05.26
(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 г. № 16)

Секретарь ГЭК


(подпись)

Гурин Е.И.
(фамилия, инициалы)

Пенза 2026

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»



«Утверждаю»
Заведующий кафедрой

М.А. Митрохин

« 07 » 04 20 26 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Рябов Святослав Игоревич гр. 22ВВП1 факультета ВТ

2. Тема работы Система видеоаналитики автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения

Тема утверждена приказом ПГУ № 347/о от " 07 " апреля 2026 г.

3. Руководитель бакалаврской работы Митрохина Н.Ю.

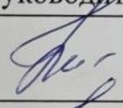
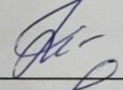
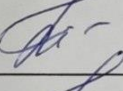
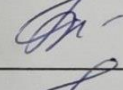
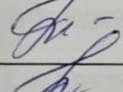
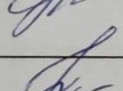
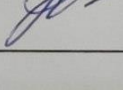
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать информационную систему автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения. Система должна быть спроектирована с применением модульной архитектуры, развернутой в Docker-контейнерах. В состав системы должны входить: модуль видеоаналитики для захвата и обработки видеопотоков с IP-камер, модуль бизнес-логики, клиентское веб-приложение и база данных. Клиентская часть должна обеспечивать просмотр видеопотоков в реальном времени и отображение аналитической информации.

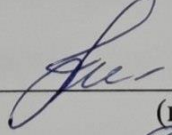
5. Перечень подлежащих разработке вопросов

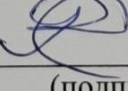
Требуется произвести анализ предметной области: изучить существующие аналоги систем видеонаблюдения, рассмотреть применяемые технологии компьютерного зрения и подходы к трекингу людей. Выбрать язык программирования, среду разработки, нейросетевую модель для детекции и алгоритм трекинга. Выполнить настройку среды разработки и контейнеризации. Спроектировать архитектуру системы, включая модули видеоаналитики, бизнес-логики, клиентского приложения и базы данных. Реализовать захват видеопотоков с IP-камер, детекцию людей, трекинг объектов, анализ контрольных зон и линий. Настроить интеграцию с системой контроля и управления доступом. Разработать REST API и клиентское веб-приложение. Выполнить контейнеризацию всех компонентов системы. Провести тестирование системы на реальных видеопотоках.

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	Анализ требований	16	09.04.2026 – 10.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	24	11.04.2026 – 13.04.2026	
4	Настройка среды разработки	24	14.04.2026 – 16.04.2026	
5	Реализация серверной части программы	144	17.04.2026 – 04.05.2026	
6	Реализация веб-интерфейса	48	05.05.2026 – 15.05.2026	
7	Оформление пояснительной записки	40	16.05.2026 – 21.05.2026	

Дата выдачи задания " 7 " апреля 2026 г.

Руководитель бакалаврской работы  Митрохина Н.Ю.
(подпись, дата)

Задание к исполнению принял студент  Рябов С.И.
(подпись, дата)

Работу к защите допустить
Декан факультета  04.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 148 листов, 15 рисунков, 16 источников, 3 приложения.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА, КОНТРОЛЬ РАБОЧЕГО ВРЕМЕНИ, ВИДЕОАНАЛИТИКА, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ДЕТЕКЦИЯ ЛЮДЕЙ, ТРЕКИНГ ОБЪЕКТОВ, YOLO, СИСТЕМА КОНТРОЛЯ ДОСТУПА, DJANGO, REACT, POSTGRESQL, DOCKER, PYTHON

Цель работы – разработать информационную систему автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения.

В рамках работы выполнены анализ предметной области и существующих аналогов, проектирование архитектуры распределенной системы, разработка серверного модуля видеоаналитики, создание клиентского веб-приложения, а также настройка интеграции с СКУД.

Разрабатываемая система должна обеспечивать автоматическое обнаружение людей в видеопотоке, отслеживание их перемещения, определение времени нахождения в контрольных зонах, сверку с данными СКУД, а также отображение видеопотоков и аналитической информации через веб-интерфейс.

В результате работы была создана информационная система, удовлетворяющая поставленным требованиям. Серверная часть реализована на языке Python, клиентская – на JavaScript с использованием React, все компоненты упакованы в Docker-контейнеры. Система протестирована в рабочих условиях и подтвердила корректность отслеживания людей, анализа активностей и интеграции с СКУД

					ПГУ1.090301.22.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Система видеоаналитики автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Рябов С.И.		27.05.24					
Провер.		Митрохина Н.Ю.		28.05.24			2	148	
Н. Контр		Бычков А.С.		28.05.24		ФВТ, каф ВТ гр. 22ВВП1			
Утверд.		Митрохин М.А.		28.05.24					

Содержание

Введение	7
1 Система видеоаналитики для автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения.	8
1.1 Анализ предметной области.....	8
1.1.1 Актуальность разработки	8
1.1.2 Анализ существующих аналогов.....	9
1.1.3 Постановка задачи.....	16
1.2 Выбор технологий и средств разработки приложений	17
1.2.1 Выбор среды разработки	17
1.2.2 Требования к аппаратному обеспечению	19
1.2.3 Выбор языка программирования	20
1.2.4 Выбор СУБД	22
1.2.5 Выбор библиотеки для захвата и обработки видеопотока.....	23
1.2.6 Выбор фреймворка для обработки запросов	24
1.3 Проектирование информационной системы	25
1.3.1 Архитектура приложения	25
1.3.2 Архитектура базы данных	30
1.4 Реализация приложения.....	32
1.4.1 Реализация БД.....	32
1.4.2 Реализация модуля Nginx	37
1.4.3 Реализация модуля Django	38
1.4.4 Реализация модуля yolo	39
1.4.5 Реализация веб-приложения.....	44
1.4.6 Тестирование.....	46
1.4.7 Контейнеризация и развертывание системы	54
2 Охрана труда	57
2.1 Классификация и категорирование помещения	57

2.2 Санитарно-гигиеническая характеристика производства.....	58
2.3 Электробезопасность	60
2.4 Пожарная безопасность	60
2.5 Искусственное освещение помещения.....	61
Заключение.....	63
Список использованных источников.....	64
Приложение А – UML диаграммы.....	66
Приложение Б – Листинг кода	71
Приложение В – Презентация	141

Введение

Современные компании часто сталкиваются с проблемой контроля трудовой дисциплины и учета рабочего времени сотрудников. Существующие методы контроля могут быть неточными и допускать ошибки во время работы. С ростом численности сотрудников предприятия потребность в автоматизации этих процессов становится особенно актуальной.

Одним из перспективных направлений является использование технологий компьютерного зрения для анализа видеопотоков с камер наблюдения. Такие системы способны обнаруживать людей в реальном времени и отслеживать их перемещение по территории.

Актуальность работы обусловлена широким распространением систем видеонаблюдения на предприятиях и потребностью в автоматизации учета рабочего времени. Существующие коммерческие решения требуют значительных затрат на приобретение оборудования и лицензий, а также не всегда соответствуют потребностям конкретного предприятия, вместо этого предлагая избыточный функционал. Разработка собственной системы на основе открытых технологий позволит снизить стоимость разработки и подстроить функционал под конкретные задачи предприятия.

Целью работы является разработка информационной системы автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения и интеграцией с системой контроля и управления доступом.

1 Система видеоаналитики для автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения.

1.1 Анализ предметной области

1.1.1 Актуальность разработки

По мере того, как компания развивается и выходит на новые уровни, неизбежно возникает необходимость в масштабировании. Это касается не только увеличения штата сотрудников, но и изменения внутренних управленческих процессов. В тот момент, когда количество персонала начинает расти линейно, классические методы ручного управления перестают работать. Руководителю становится сложнее следить за тем, соблюдается ли трудовой распорядок, вовремя ли люди приходят на рабочие места и покидают их, а также обеспечивать должный уровень безопасности.

В качестве инструмента для уменьшения управленческой нагрузки уже достаточно давно и успешно применяются специализированные системы, предназначенные для автоматического контроля рабочего времени. Подобные программно-аппаратные комплексы получили широкое распространение на крупных предприятиях и производствах. Они имеют широкий функционал, позволяющий решать множество управленческих задач. В состав таких решений часто входят модули, направленные на анализ загруженности зон, мониторинг активности и контроль складских территорий.

Суть работы такой системы заключается в видеоаналитике. Камеры наблюдения, размещенные на объекте, транслируют видеопоток, который в режиме реального времени обрабатывается алгоритмами компьютерного зрения. Система берет на себя задачу идентификации человека и фиксации событий, связанных с его перемещением и деятельностью, что позволяет вести учет без участия человека. Такая система помогает руководителю, она полностью снимает с него необходимость лично контролировать присутствие подчиненных на месте

					ПГУ1.090301.22.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

либо привлекать для этого отдельных ответственных лиц. Владелец бизнеса или директор получает доступ к достоверной информации, не затрачивая на это много времени: можно за пару кликов увидеть сводку об отработанных часах для каждого сотрудника, проверить данные опозданий и прогулов или даже проверить, в каком месте сейчас находится тот или иной специалист. Такой подход позволяет высвободить значительный временной ресурс менеджмента, который гораздо разумнее направить на решение стратегических вопросов и развитие бизнеса.

Стоит отметить, что в современных компаниях, занимающих большие территории, системы безопасности и контроля, как правило, уже включают в себя целый ряд смежных подсистем: классическое видеонаблюдение, средства контроля и управления доступом, системы мониторинга деятельности. Система контроля рабочего времени сотрудников в данном контексте выступает как часть этой экосистемы. Одним из ключевых достоинств такого решения является его узкая специализация и, как следствие, высокая адаптивность. Поскольку система ориентирована на выполнение конкретной прикладной задачи – отслеживание персонала и учет времени – она не вступает в программные или аппаратные конфликты с другим установленным оборудованием компании. Напротив, она легко интегрируется в существующую IT-архитектуру предприятия и может эффективно функционировать совместно с прочими системами безопасности.

1.1.2 Анализ существующих аналогов

Для обоснования актуальности разработки собственного программного обеспечения в рамках данной выпускной квалификационной работы необходимо провести анализ представленных на рынке продуктов, решающих схожие задачи в области видеонаблюдения и контроля рабочего времени персонала.

Среди аналогов автоматизированных систем, позволяющих осуществлять видеомониторинг территории предприятия и учет трудовой активности сотрудников, можно выделить:

– услуга «МегаФон Видеонаблюдение» [1] для корпоративных клиентов;

					ПГУ1.090301.22.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

- программное [2] обеспечение для потоковой видеоаналитики VisionLabsFaceStream;
- экосистема TRASSIR [3] для видеонаблюдения и видеоаналитики.

Продукт «Видеонаблюдение» от «МегаФон» представляет собой комплексное решение, предназначенное для обработки изображения с неограниченного количества IP-видеокамер, записи и хранения видеоданных, а также автоматизации различных бизнес-процессов, в том числе управление персоналом и системами безопасности. Продукт может быть развернут как в облачной инфраструктуре оператора, так и на локальном сервере заказчика.

Согласно официальной документации, представленной в открытом доступе, данное решение предоставляется в двух вариантах:

- облачная версия, при которой запись, хранение и обработка видеопотока осуществляются на удаленных серверах оператора;
- инсталляционная версия, предполагающая размещение всего необходимого программного обеспечения и всех данных на оборудовании заказчика.

Оплата услуги в облачном варианте основана на ежемесячной абонентской плате за каждую используемую IP-камеру. Также стоимость может меняться в зависимости от срока хранения видеоданных. Например, базовый тариф с хранением данных в течение 7 дней обойдется в 550 рублей за одну камеру в месяц, другой тариф, который обеспечивает хранение архива данных в течение 180 дней и стоит 2400 рублей за каждую камеру ежемесячно. Для инсталляционной версии, предусмотрена продажа бессрочных лицензий на каждый видеоканал. Стоимость лицензии на одну IP-камеру от 4000 рублей и до 11000 рублей.

Стоит отметить перечень дополнительных аналитических модулей, которые необходимо приобрести дополнительно к базовому тарифу или лицензии. В контексте выпускной квалификационной работы данные модули

					ПГУ1.090301.22.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

представляют наибольший интерес, потому что они выполняют поставленные или смежные задачи:

- «Контроль активности персонала» стоимостью 6000 рублей в месяц за одну камеру;

- «Распознавание лиц с подсчетом уникальных посетителей» стоимостью от 11140 до 94900 рублей в месяц в зависимости от общего количества камер в системе;

- «Тепловая карта интенсивности движения» стоимостью 4000 рублей в месяц за канал.

В заявлении на подключение услуги предусмотрена возможность заказа расчета стоимости поставки технического оборудования и установки. Минимальный срок оказания услуги составляет 12 месяцев, а в случае досрочного расторжения клиент обязан компенсировать абонентскую плату за оставшийся до истечения года период.

Исходя из изученной документации, можно сформулировать ряд положительных сторон данного аналога:

- комплексный подход к организации видеонаблюдения, включающий поставку оборудования, монтаж, настройку и дальнейшую техническую поддержку;

- наличие готовых модулей видеоаналитики, не требующих от заказчика компетенций в области видеоаналитики и машинного обучения;

- возможность выбора между облачным и локальным развертыванием системы в зависимости от потребностей.

Однако, несмотря на перечисленные достоинства, данное решение обладает рядом существенных ограничений и недостатков, критичных в рамках поставленной в выпускной квалификационной работе задачи:

- высокая совокупная стоимость использования системы, особенно для крупных предприятий с большим числом видеокамер. Ежемесячная плата за

модуль контроля активности персонала при наличии, к примеру, 30 камер составит 200 000 рублей, что значительно увеличивает расходы компании;

- закрытый код системы, не позволяющий модифицировать алгоритмы трекинга и видеоаналитики под потребности и особенности компании;

- отсутствие возможности интеграции с существующей на предприятии системой контроля и управления доступом без привлечения разработчиков со стороны продавца;

- привязка к конкретным требованиям к IP-камерам и каналам связи, перечисленным в технической документации, что может потребовать дополнительных затрат на модернизацию уже существующей инфраструктуры видеонаблюдения.

Проведенный анализ показал, что коммерческое решение от МегаФона при всех его возможностях требует значительных затрат и не предусматривает настройки под требования компании. В связи с этим разработка собственного программного продукта, заточенного под задачи учета рабочего времени и интеграцию с существующей СКУД предприятия, является более рациональным и экономически выгодным решением.

VisionLabsFaceStream представляет собой специализированное программное обеспечение российской разработки, предназначенное для детекции и трекинга лиц в видеопотоке в режиме реального времени. Продукт ориентирован на интеграцию с системами безопасности и контроля доступа, обеспечивая захват видео с различных источников, выделение лиц на кадрах и передачу полученных данных во внешние сервисы для последующей идентификации.

Согласно официальной документации, представленной разработчиком, FaceStream имеет следующие основные функции:

- чтение видеопотоков с веб-камер, USB и IP-камер (по протоколу RTSP), а также из видеофайлов;

- обработка видеопотоков с поиском и отслеживанием лиц;

					ПГУ1.090301.22.001 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

– сортировка лучших снимков и отправка посредством HTTP-запросов во внешний сервис.

При работе с FaceStream также доступны такие возможности, как динамическое управление источниками видеопотоков через интерфейс приложения, просмотр RTSP-потоков в браузере в реальном времени и получение данных по каждому потоку (кадры в секунду, количество ошибок, количество необработанных кадров). Система может быть настроена на работу с лицами, с людьми или оба варианта одновременно.

Отдельно стоит отметить хорошую оптимизацию производительности, предусмотренные в FaceStream. К ним относятся:

– задание зоны интереса для ограничения области поиска лиц в кадре, что значительно уменьшает нагрузку на вычислительные ресурсы;

– масштабирование кадра перед обработкой с возможностью выбора разрешения, при котором снижается потребление ресурсов, но детекция происходит верно;

– режим учета перемещений в кадре, при котором детекция выполняется только в областях с движением.

Для ускорения вычислений FaceStream поддерживает использование ресурсов видеокарты. В требованиях к операционной системе представлена возможность использования в российских операционных системах РЕД ОС версии 7.3 и выше, а также Astra Linux версии 1.8 и выше.

На основе изученной документации можно выделить следующие достоинства данного решения:

– высокое качество алгоритмов детекции и трекинга лиц;

– поддержка российских операционных систем, в соответствии с требованиями государственных учреждений и компаний с требованиями к импортозамещению;

– гибкие настройки производительности, позволяющие адаптировать систему под доступные аппаратные ресурсы.

Однако, несмотря на преимущества продукта, в рамках решаемой в ВКР задачи, можно выделить ряд существенных недостатков:

- FaceStream не является полноценной системой учета рабочего времени. Продукт выполняет только функцию поставщика данных (детекция и трекинг), в то время как для полноценной работы требуется развертывание другой платформы, что влечет за собой дополнительные затраты на лицензии и оборудование;

- отсутствие возможности формирования отчетности по отработанным часам и средств сопоставления треков движения с конкретными сотрудниками, прошедшими идентификацию через СКУД;

- закрытый исходный код и лицензионная политика, которые ограничивают самостоятельную доработку алгоритмов под специфические потребности конкретного предприятия;

- высокая стоимость внедрения при масштабировании системы на большое количество камер, что делает данное решение экономически невыгодным для решения поставленной задачи контроля трудовой деятельности в компании.

Использование VisionLabsFaceStream в связке с LUNAPLATFORM целесообразно в крупных компаниях с выделенным бюджетом, однако для целей, поставленных в рамках данной ВКР, это решение является избыточным как по функционалу, так и по стоимости.

TRASSIR представляет собой программно-аппаратную платформу, объединяющую в себе видеонаблюдение, видеоаналитику и контроль доступа. Он имеет модульную архитектуру: видеонаблюдение реализуется ядром TRASSIR VMS, а дополнительные возможности можно приобрести с отдельными компонентами.

У системы есть два модуля, функционал которых соответствует решаемой задаче. Во-первых, это модуль видеоаналитики, включающий детектор человека на заданном рабочем месте, он позволяет отслеживать время прихода и ухода сотрудников, фиксировать опоздания и длительные отсутствия, а также

формировать отчеты. Во-вторых, TRASSIR предлагает модуль интеграции со сторонними СКУД и системами учета сотрудников, однако данная возможность доступна только в продвинутых версиях, в то время как базовая версия ограничена только видео-функционалом.

Для крупных территориально расположенных объектов предусмотрен компонент TRASSIR CMS, обеспечивающий единое управление серверами, пользователями и обновлениями, а также веб-интерфейс для удаленного доступа. Платформа включена в реестр российского программного обеспечения и совместима с операционной системой Astra Linux.

На основе изученных материалов можно сформулировать следующие достоинства данного аналога:

- наличие готового к применению модуля детекции присутствия на рабочем месте;
- модульная архитектура, позволяющая заказчику подбирать конфигурацию под свои задачи;
- официальный статус российского ПО и совместимость с отечественными операционными системами;
- управление крупными системами через модуль CMS, предоставляющий веб-интерфейс.

Однако, несмотря на это, TRASSIR имеет некоторые недостатки:

- высокая стоимость системы, связанная с модульным лицензированием;
- Для получения системы с функционалом, заявленным в выпускной квалификационной работе, требуется приобретение как лицензий, так и дополнительных модулей видеоаналитики и интеграции со СКУД, а также модуль CMS для централизации;
- привязка модулей видеоаналитики к специализированным видеорегистраторам TRASSIR;
- закрытый исходный код, не позволяющий модифицировать алгоритмы под нестандартные ситуации.

Таким образом, TRASSIR требует значительных финансовых вложений на этапе приобретения лицензий и аппаратного обеспечения. Разработка собственного приложения с использованием открытых технологий позволяет избежать покупки лицензий, обеспечить гибкое зонирование территории и интеграцию со СКУД.

Проведенный анализ трех представленных аналогов позволяет сделать следующие выводы. Продукт «МегаФон Видеонаблюдение» предлагает комплексный подход, но требует значительных ежемесячных затрат и не обеспечивает прямой интеграции со СКУД. VisionLabsFaceStream демонстрирует высокое качество трекинга лиц, однако не является самостоятельной системой учета рабочего времени и требует развертывания платформы LUNA PLATFORM. Платформа TRASSIR, наиболее близкая к теме ВКР по функционалу, также использует модульную архитектуру, а работа видеоаналитики привязана к специализированному оборудованию продавца.

Все аналоги имеют значительный недостаток в виде закрытого исходного кода, уменьшающего гибкость системы и увеличивающего стоимость масштабирования. Ни одно из решений не предоставляет требуемый функционал, без приобретения расширений.

Таким образом, разработка собственного приложения с использованием открытых технологий является оптимальным решением. Это позволит избежать лицензионных трат, настроить систему под требования, реализовать прямую интеграцию со СКУД и предоставить интерфейс, отображающий именно те данные, которые необходимы для управления.

1.1.3 Постановка задачи

Исходя из выявленных в ходе анализа предметной области потребностей предприятий, в рамках выполнения выпускной квалификационной работы требуется реализовать приложение. Данный продукт должен представлять собой комплексное решение, включающее модули видеоаналитики, интерфейсы, и следующий функционал:

– автоматическое обнаружение людей в видеопотоке, поступающем с камер наблюдения, и их последующий трекинг в режиме реального времени для построения траектории перемещения в пределах видимости камер;

– интеграция со смежной подсистемой контроля и управления доступом (СКУД) с целью сопоставления обезличенных данных видеоаналитики с персональными данными сотрудников, прошедших идентификацию по пропуску при входе или выходе с территории объекта;

– формирование на основе данных, поступающих с видеоаналитики, сводных таблиц по учету рабочего времени, которые предоставляют возможность оперативного получения сведений о суммарном количестве отработанных часов, продолжительности отсутствия сотрудника на своем месте, а также о прочих событиях, выходящих за рамки нормативного распорядка рабочего дня;

– предоставление удобного веб-интерфейса, доступного пользователю через стандартный браузер, который должен обеспечивать отображение требуемой аналитической информации, а также трансляцию потока с контейнера видеоаналитики для визуального контроля обстановки на объекте оператором;

– обеспечение функций администрирования разрабатываемой системы и управления ее базой данных через специализированную панель, в качестве которой планируется использовать административная панель DjangoAdmin, позволяющую осуществлять мониторинг состояния СУБД, управлять учетными данными и выполнять настройку системы.

Целевыми пользователями системы являются администраторы, охранники и руководители. Варианты использования системы для каждой из этих ролей представлены в приложении А.1.

1.2 Выбор технологий и средств разработки приложений

1.2.1 Выбор среды разработки

					ПГУ1.090301.22.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

Среда разработки Visual Studio Code. Visual Studio Code – это современная кроссплатформенная среда разработки, созданная компанией Microsoft и предназначенная для работы с различными языками программирования, такими как Java, C++, Python, JavaScript и другими. Данная среда является достаточно популярной среди разработчиков благодаря своей универсальности и удобству использования.

В программе Visual Studio Code есть множество функций, которые упрощают разработку программ:

- редактор кода имеет подсветку синтаксиса, автодополнение, форматирование кода и другие особенности, которые ускоряют процесс написания кода;
- система отладки, показывающая и исправляющая ошибки во время процесса разработки программы;
- большое количество расширений в открытом доступе, их можно установить напрямую из встроенного магазина, это позволяет адаптировать среду под себя, под определённые задачи и технологии, применяемые при разработке;
- использование системы контроля версий Git, это облегчает управление изменениями в проекте, а также позволяет вести разработку в команде;
- поддержка работы на разных операционных системах.

Одним из главных преимуществ Visual Studio Code является бесплатное распространение и открытый исходный код, что делает его более доступным для большинства пользователей.

Также есть альтернативная Visual Studio Code среда разработки разработки PyCharm [4], ее основное назначение – это разработка программ на языке Python. PyCharm имеет большое количество встроенных инструментов для работы с Python: умное автодополнение, анализа кода, система отладки и поддержку использования виртуальных окружений.

Недостаток PyCharm заключается в том, что он требует больше ресурсов компьютера и может работать хуже на слабых устройствах. Кроме того, полноценная версия среды является платной, что может быть ограничением для некоторых пользователей. Интерфейс программы также может показаться более сложным из-за большого количества встроенных функций.

Сравнивая данные среды разработки, можно отметить, что PyCharm лучше подходит для узкоспециализированной разработки на Python, особенно при работе с крупными проектами. В то же время Visual Studio Code является более лёгким и универсальным инструментом, который можно гибко настроить под различные задачи, включая разработку на Python.

Данная среда разработки была выбрана в связи с её преимуществами при решении поставленной задачи: наличием большого количества расширений, позволяющих оптимизировать процесс разработки и упростить работу с различными компонентами проекта, а также возможностью использовать её как на рабочем персональном компьютере, так и на сервере под управлением Ubuntu с графическим интерфейсом.

1.2.2 Требования к аппаратному обеспечению

Для запуска системы видеоаналитики, которая будет обрабатывать видеопоток в реальном времени, необходимо выполнить требования как к серверному оборудованию, так к рабочим станциям клиентов.

Серверная часть (модуль видеоаналитики и БД).

Системные требования могут отличаться, в зависимости от количества обрабатываемых камер. Требования, предъявляемые в данном пункте, соответствуют задаче, предъявляемой к системе, а именно стабильная обработка до 4 камер в реальном времени. Модуль детекции использует нейросетевую модель YOLOv8m, которая требует аппаратное ускорение, сервер должен соответствовать следующим требованиям:

– процессор: x86_64 (Intel/AMD) от 4 ядер и 8 потоков, тактовая частота не ниже 3.0 ГГц;

- оперативная память минимум 16 ГБ, рекомендуется 32 ГБ;
- видеокарта (GPU), обязательно требуется поддержка CUDA (NVIDIA); объем видеопамати не менее 6 ГБ. Видеокарта используется для обработки YOLO и должна обеспечивать скорость обработки 25 кадров в секунду для 4 видеопотоков одновременно;
- накопитель SSD от 256 ГБ для установки Docker-образов и временного хранения данных PostgreSQL.

Клиентская часть (Веб-интерфейс).

Клиент работает через веб-браузер и не требует высокой производительности, так как обработка выполняется на сервере. Запустить его можно на стационарном устройстве или на мобильном телефоне.

Требования для ПК:

- процессор Intel Core i3 / AMD Ryzen 3;
- оперативная память 4 ГБ;
- видеокарта любая встроенная или дискретная с поддержкой WebGL;
- экран 1366x768;
- ОС Windows 10/11, Ubuntu 20.04+.

Мобильные устройства (смартфоны и планшеты):

- iOS 14.0 и выше (Safari, Chrome);
- Android 11.0 и выше (Chrome, Яндекс Браузер);
- экран разрешение от 720x1280;
- ОЗУ от 2 ГБ.

1.2.3 Выбор языка программирования

Выбор языка программирования.

Язык программирования C++ – это высокоуровневый язык программирования, его особенность в том, что он в себе объединяет возможности и как низкоуровневых языков, так и объектно-ориентированных языков. C++ позволяет писать эффективный и оптимизированный код благодаря продвинутой работе с памятью, использованию указателей и богатому выбору библиотек. Он

поддерживает многопоточность и различные подходы к решению задач. Однако при использовании C++ для решения задач, связанных с обработкой изображений и нейросетевыми моделями, потребуется потратить много времени и ресурсов на реализацию, настройку и оптимизацию программы. Это значительно усложняет разработку на этом языке и не подходит для выпускной квалификационной работы.

Язык программирования Java – это объектно-ориентированный язык программирования, его основное направление это разработка кроссплатформенных приложений. У него простой синтаксис, автоматическое управление памятью и высокой уровень надежности. Есть возможность запускать программы на разных операционных системах, без изменения кода. Для задач видеоаналитики и Java почти не используется, а малое количество доступных библиотек усложняет разработку.

Язык программирования Python [5] – это высокоуровневый интерпретируемый язык программирования. Он имеет простотой синтаксиса, что позволяет ускорить процесс создания программы. Python – один из самых популярных в области машинного обучения, компьютерного зрения и анализа данных.

Для реализации серверной части приложения, связанной с видеоаналитикой и распознаванием людей, в Python существует большое количество библиотек. Широко распространена нейросетевая модель YOLO, предназначенная для обнаружения объектов в реальном времени. Python позволяет легко использовать данную модель, а также работать с такими библиотеками, как OpenCV, Gstreamer, которые требуются для обработки видеоданных.

Также Python хорошо подходит для реализации серверной части благодаря наличию удобных средств для работы с сетью и обработки информации. Это позволяет создать оптимизированную обработку видеопотока и передачу результатов работы программы.

После сравнения языков программирования был выбран Python, так как он имеет самые удобные и эффективные средства для реализации поставленных задач видеоаналитики с использованием нейронных сетей. Наличие готовых решений и библиотек значительно ускоряет процесс разработки и позволяет сосредоточиться на реализации основной функциональности приложения.

1.2.4 Выбор СУБД

СУБД PostgreSQL [6] – объектно-реляционная СУБД, находящаяся в открытом доступе, у нее множество средств хранения, обработки и управления данными. Данная система имеет открытый исходный код и активно совершенствуется разработчиками.

Одно из главных преимуществ PostgreSQL, она позволяет эффективно обрабатывать большие объемы данных, это позволит использовать ее в проектах, на которых требуется высокая нагрузка на базу данных. Также она является кроссплатформенной, поддерживается на разных устройствах без ограничений. Базу данных PostgreSQL можно масштабировать и детально настраивать в зависимости от потребностей. Также в PostgreSQL можно использовать сложные запросы, триггеры и процедуры, что расширяет функционал базы данных.

Использование PostgreSQL отлично подходит для реализации серверной части приложения, так как она позволяет работать одновременно нескольким пользователям. Это важно в задачах, связанных с видеоаналитикой, где происходит постоянная запись и обработка информации.

СУБД MySQL – это популярная реляционная система управления базами данных, ее широко используют в веб-разработке и десктопных программах. Ее популярность заключается в простоте использования и высокой скорости работы.

MySQL является бесплатной и доступной СУБД, это делает её более интересной для большинства разработчиков. Ее просто освоить и удобно использовать, она подходит, как для начинающих, так и для опытных пользователей. MySQL обеспечивает высокую производительность при выполнении стандартных задач.

Однако при работе с более сложными задачами и большим объёмом данных, MySQL может быть недостаточно. В некоторых моментах она уступает PostgreSQL по функциональности, особенно в работе со сложными запросами.

СУБД SQLite – это лёгкая встраиваемая система управления базами данных, ее особенность – она не требует сервера и хранит данные в файле. Она подходит для небольших приложений и проектов с малой нагрузкой.

SQLite очень просто установить и использовать, потому что не требуется настройка сервера. Также один из плюсов, это ее малый вес и высокая скорость работы при небольших объёмах данных.

Несмотря на свои преимущества, SQLite не подходит для крупных проектов. Она имеет сильно ограниченный функционал и не имеет одновременного доступа.

В результате сравнения систем управления базами данных была выбрана PostgreSQL, она соответствует требованиям к системе, у нее широкий функционал и высокая надёжность, производительность. Данная СУБД позволяет эффективно обрабатывать большой объём данных и работает стабильно при высоких нагрузках.

1.2.5 Выбор библиотеки для захвата и обработки видеопотока

Перед разработкой системы видеоаналитики нужно правильно подобрать инструменты, которые будут обрабатывать входной видеопоток, от этого зависит скорость и качество следующих этапов обработки. Это имеет существенное влияние на систему, которая одновременно должна обрабатывать множество параллельных потоков.

Было рассмотрено несколько подходящих библиотек, доступных в Python.

OpenCV [7] одна из самых распространенных библиотек в области компьютерного зрения. Она позволяет получать видеопотоки из различных источников, включая камеры, файлы и сетевые потоки. OpenCV имеет множество разных возможностей для обработки изображений и сочетается с библиотеками, предназначенными для машинного обучения.

Также был изучен фреймворк GStreamer [8], он представляет собой большую библиотеку для работы с видео. У Gstreamer есть отличие от классических библиотек, он основан на конвейерной архитектуре. Это позволяет тщательнее управлять процессами захвата, декодирования и передачи видеопотока. GStreamer поддерживает различные форматы и протоколы передачи данных, также может осуществлять параллельную обработку источников и показывать высокую производительность при работе в реальном времени.

Также рассматривался FFmpeg – это популярный инструмент для обработки аудиоданных и видеоданных. Несмотря на его широкий функционал, было принято решение не использовать его в работе, так как настройка и использование FFmpeg сложнее чем у Gstreamer.

Было выбрано сочетать 2 библиотеки: GStreamer и OpenCV. Это позволит разделить функции между разными частями системы. GStreamer отвечает за захват, передачу и предварительную обработку видеопотока, при этом использует мощности видеокарты для ускорения. OpenCV применяется для работы с кадрами: выполняет предварительную обработку, анализ изображения и взаимодействует с моделью распознавания.

1.2.6 Выбор фреймворка для обработки запросов

Было рассмотрено несколько решений для реализации серверной части, которая будет обрабатывать HTTPS-запросы. Одними из основных критериев являются: производительность при обработке запросов, возможность масштабирования, защита системы, поддержка API и возможность интеграции с другими частями системы.

Django [9] представляет собой комплексное решение, большинство возможностей встроено в этот инструмент изначально. Он предоставляет средства для работы с базами данных, систему управления пользователями, имеет собственную страницу администрирования и механизмы миграции. Django

часто используется в крупных проектах, что подтверждает эффективность этого фреймворка.

Flask отличается от Django, он является более минималистичным решением. Он выполняет только базовую обработку запросов, все остальное разработчику нужно подключать в виде дополнительных компонентов. Это делает его удобным для небольших проектов с простой архитектурой. Однако при разработке сложной системы возникает необходимость самостоятельно собирать бизнес-логику с использованием сторонних библиотек, это усложняет разработку и сопровождение проекта.

FastAPI, ориентирован на разработку современных API-сервисов и отличается своей асинхронной работой. Он обеспечивает высокую скорость работы и автоматизацию процессов в системе. Этот инструмент отлично подходит для систем с высокой нагрузкой. Но отсутствие встроенных компонентов, например административной панели или механизма аутентификации, требует дополнительных усилий для решения этих вопросов.

С учётом поставленных задач было принято решение использовать Django. Он дает возможность сразу получить готовую систему для обработки HTTPS-запросов, управления пользователями, работы с базой данных и администрирования. Это значительно упрощает разработку и делает систему оптимизированной за счёт отсутствия сторонних решений.

1.3 Проектирование информационной системы

1.3.1 Архитектура приложения

Архитектура информационной системы автоматического контроля и учета рабочего времени сотрудников состоит из двух частей: клиентской и серверной части.

Клиентская часть представляет собой веб-интерфейс, доступный пользователю через стандартный браузер. Через него пользователь системы взаимодействует с приложением: просматривает видеопотоки с камер в реальном

времени, анализирует данные, отслеживает события входа и выхода сотрудников. Чтобы получить статистическую информацию: список камер, данные о зонах, логи событий и права доступа - веб-интерфейс отправляет обычные HTTP-запросы. Для приема видеопотока и обновляющихся данных в реальном времени используется соединение по протоколу WebSocket, реализованное через библиотеку Socket.IO. Это решение позволяет мгновенно отображать обработанные кадры и обновлять данные без перезагрузки страницы. Диаграмма модулей представлена на рисунке 1.

Серверная часть системы реализована в виде набора независимых модулей, каждый из которых представляет собой отдельный контейнер, развернутый в среде Docker. Этот подход упрощает масштабирование и поддержку системы, позволяя изменять и перезапускать отдельные модули без остановки всей системы.

Модуль веб-сервера выполняет обработку всех запросов. Он принимает HTTP-запросы от клиентского приложения, раздает статические файлы веб-интерфейса и проксирует обращения к внутренним сервисам. В системе для защиты передаваемых данных используется SSL-шифрование на основе сертификатов Let'sEncrypt. Nginx также обеспечивает маршрутизацию WebSocket-соединений к контейнеру видеообработки. Все HTTP-запросы, адресованные к API (/api/), веб-сервер перенаправляет на модуль бизнес-логики по протоколу HTTP. WebSocket-соединения для получения видеопотока (/socket.io/), веб-сервер проксирует на модуль видеоаналитики.

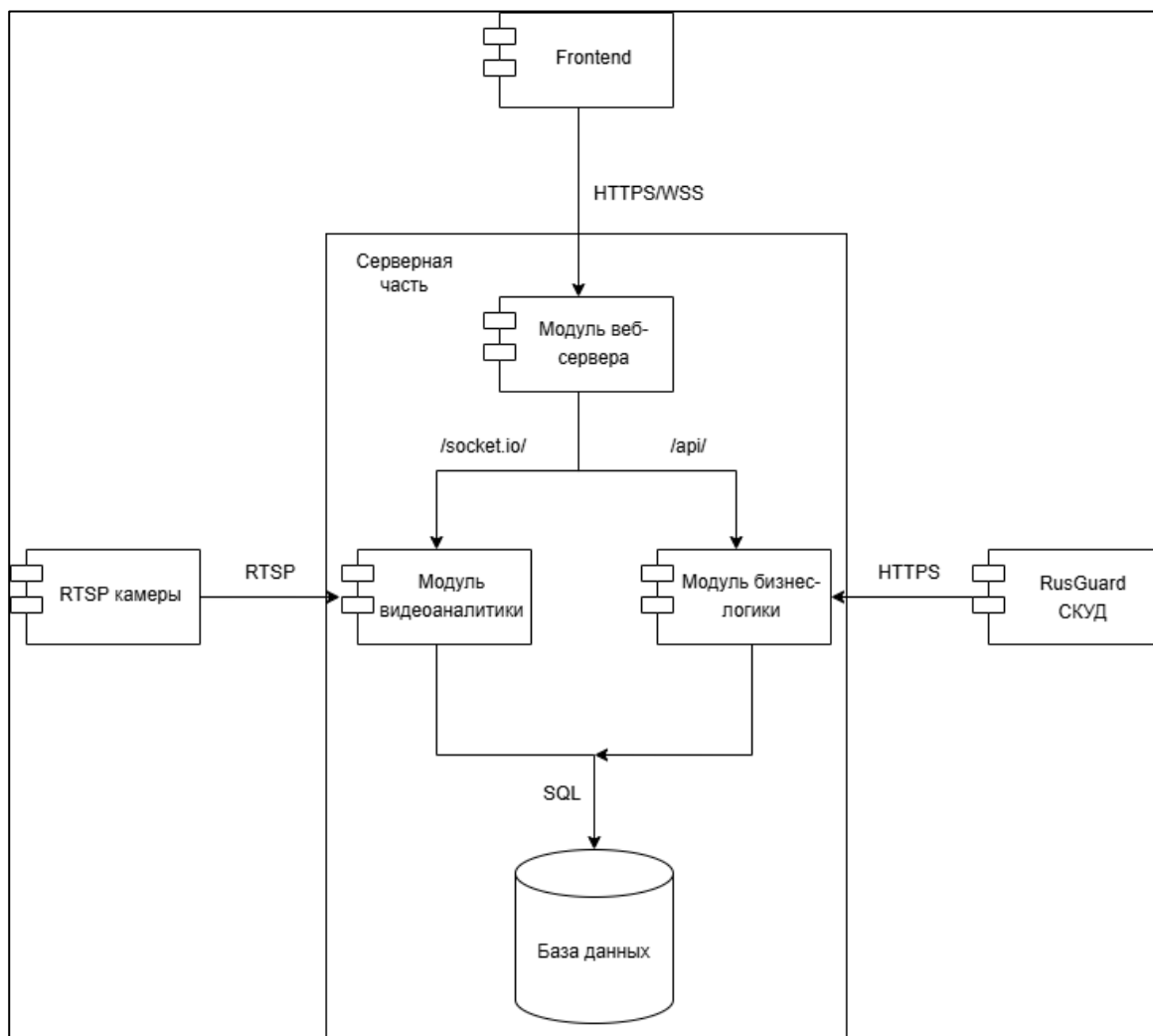


Рисунок 1 – Архитектура модулей

Серверная часть системы реализована в виде набора независимых модулей, каждый из которых представляет собой отдельный контейнер, развернутый в среде Docker. Этот подход упрощает масштабирование и поддержку системы, позволяя изменять и перезапускать отдельные модули без остановки всей системы.

Модуль бизнес-логики реализует серверную логику приложения. Он предоставляет REST API для клиентской части: аутентификация пользователей, получение списка камер и информации о них, получение логов и сводной информации по времени пребывания сотрудников в зонах. Через этот же контейнер осуществляется прием событий от внешней СКУД RusGuard. Модуль бизнес-логики работает с базой данных через встроенный ORMDjango, который

позволяет обращаться к данным как к объектам Python, не используя напрямую SQL-запросы.

Модуль видеоаналитики выполняет обработку видеопотоков в реальном времени. Основные задачи этого модуля: захват видеоряда с IP-камер по протоколу RTSP с использованием GStreamer, детекция людей с помощью нейросетевой модели YOLOv8m, отслеживание обнаруженных объектов с использованием алгоритма трекинга SORT, анализ нахождения сотрудников в зонах и пересечение линий. Обработанные кадры передаются клиентам через WebSocket-соединение. Все зафиксированные события записываются напрямую в базу данных PostgreSQL через пул соединений библиотеки psycopg2.

База данных представлена в виде СУБД PostgreSQL, в которой хранятся все данные системы: информация о камерах, зонах и линиях, журнал событий видеоаналитики, информация о пребывании в зонах, события СКУД, справочники сотрудников.

Такая архитектура позволяет разделить нагрузку и обязанности между контейнерами: каждый модуль решает только свою задачу, не затрагивая работу остальных. Это позволяет обновлять отдельные части системы и повышает отказоустойчивость за счет возможности перезапуска отдельных контейнеров без остановки всей системы.

Клиентское веб-приложение реализовано на библиотеке React и работает в браузере пользователя. Оно предоставляет выбор камер, возможность просматривать видеопотоки в реальном времени, статистику и логи событий СКУД.

На рисунке 2 представлена диаграмма развертывания приложения, отображающая размещение компонентов системы по физическим узлам.

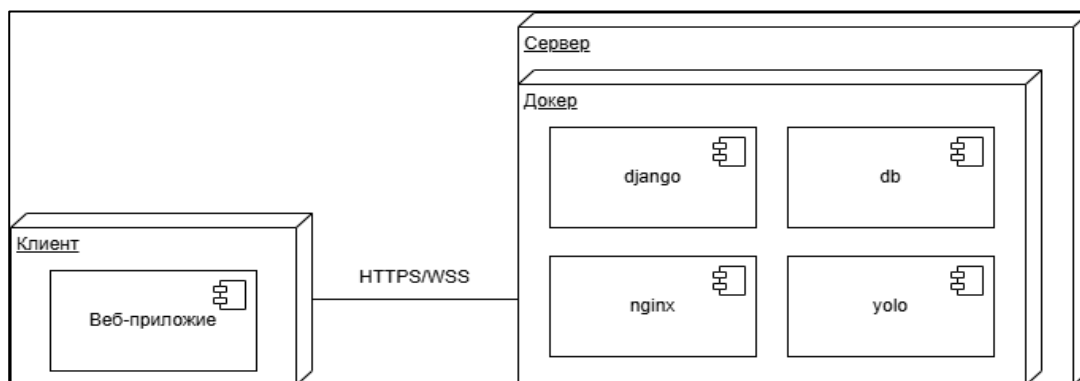


Рисунок 2 – Диаграмма развертывания

Взаимодействие модулей системы можно представить в виде диаграмм последовательности, описывающих ключевые сценарии работы приложения.

Первый сценарий – авторизация пользователя и загрузка главной страницы системы. Пользователь вводит свои учетные данные в форму авторизации клиентского приложения. Приложение отправляет POST-запрос на endpoint `/api/token/`. Веб-сервер Nginx проксирует запрос на модуль бизнес-логики Django, который проверяет учетные данные через базу данных PostgreSQL. При успешной проверке Django возвращает пару JWT-токенов, которые клиент сохраняет в локальном хранилище браузера. После авторизации клиентское приложение выполняет два запроса к серверу: получение прав доступа пользователя и список камер с их зонами и линиями. После получения данных отображается главная страница с трехмерной сценой и интерактивными метками камер. Диаграмма последовательности представлена в приложении А.2.

Второй сценарий – открытие страницы камеры и подписка на видеопоток. Пользователь выбирает камеру из списка или нажимает на метку на 3D-сцене. Клиент запрашивает информацию о камере, сводку по времени нахождения в зонах и логи СКУД. Параллельно устанавливается WebSocket-соединение с модулем видеоаналитики через веб-сервер, затем клиент подписывается на получение обработанных кадров. Диаграмма последовательности представлена в приложении А.3.

Третий сценарий – обработка видеопотока. IP-камера передает видеопоток по протоколу RTSP на модуль видеоаналитики. Модуль выполняет захват кадра через GStreamer, детекцию людей моделью YOLOv8m и отслеживанием треков алгоритмом SORT. Для каждого трека проверяется нахождение в заданных зонах и пересечение линий. При подтверждении входа в зону выполняется сверка с данными СКУД через запрос к базе данных. Если сотрудник подтвержден, в базу записывается событие zone_enter и создается сессия пребывания в зоне. При выходе из зоны сессия закрывается с записью продолжительности. Обработанный кадр с нанесенными элементами кодируется в JPEG и через Socket.IO передается всем подписанным на камеру клиентам. Диаграмма последовательности представлена в приложении А.4.

1.3.2 Архитектура базы данных

База данных информационной системы спроектирована с учетом необходимости хранения данных о камерах видеонаблюдения, контрольных зонах и линиях, событиях видеоаналитики, сессиях пребывания сотрудников в зонах, а также событиях, поступающих от системы контроля и управления доступом. Система управления базами данных (СУБД) – PostgreSQL. Для администрирования базы данных используется панель pgAdmin.

В состав базы данных входят ряд рассмотренных далее таблиц.

Таблица camera – содержит информацию об IP-камерах, подключенных к системе. Для каждой камеры хранится RTSP-адрес видеопотока (url), тип детекции (detection_type), название (name), координаты расположения на 3D-сцене (coordinates) и признак активности (is_active). Тип детекции определяет, какие объекты распознаются на видеопотоке (например, люди или автомобили).

Таблица camera_zone – описывает контрольные зоны, привязанные к конкретной камере через внешний ключ camera_id. Для каждой зоны задается название, полигон в формате JSON (массив координат точек многоугольника), цвет для отображения в веб-интерфейсе и текстовое описание. В поле description

заносятся фамилии и имена сотрудников, закрепленных за данной зоной, что используется для последующей сверки с данными СКУД.

Таблица camera_line – содержит информацию о контрольных линиях, связанных с камерой через camera_id. Каждая линия задается двумя точками в формате JSON, имеет направление контроля (both, in, out) и цветовое обозначение.

Таблица person_event – журнал всех событий, зафиксированных видеоаналитикой. Каждое событие связано с камерой (camera_id) и может относиться к зоне (zone_id), линии (line_id) или сотруднику (employee_id). Тип события определяется полем event_type, которое может принимать значения: вход в зону (zone_enter), выход из зоны (zone_exit), пересечение линии (line_cross, line_cross_in, line_cross_out). Дополнительно хранятся идентификатор трека (track_id), время события, координаты точки пересечения и метаданные в формате JSON.

Таблица person_zone_session – хранит сессии пребывания сотрудников в контрольных зонах. Каждая запись содержит ссылку на камеру, зону и сотрудника, идентификатор трека, время входа (entered_at) и выхода (exited_at), продолжительность пребывания в секундах и статус сессии (open или closed). Сессия открывается при подтверждении входа сотрудника в зону и закрывается при его выходе.

Таблица rusguard_event – журнал событий, поступающих от системы контроля и управления доступом RusGuard. Каждое событие содержит тип (enter или exit), время, фамилию и имя сотрудника (employee_name), а также ссылку на запись в справочнике сотрудников (employee_id). Эти данные используются для сверки с результатами видеоаналитики: время нахождения в зоне засчитывается только тем сотрудникам, чье последнее событие в СКУД является входом.

Таблица employee – справочник сотрудников предприятия. Содержит фамилию (surname), имя (name) и название организации (company). Используется для связывания событий видеоаналитики и СКУД с конкретным человеком.

					ПГУ1.090301.22.001 ПЗ	Лист
						31
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица `vehicle` – справочник транспортных средств. Хранит государственный номер (`number`), марку автомобиля (`brand`) и ссылку на сотрудника-владельца (`employee_id`).

Таблица `vehicle_event` – журнал событий, связанных с въездом и выездом транспортных средств. Фиксирует время прибытия (`arrival_time`) и убытия (`departure_time`), продолжительность пребывания (`stay_duration`), а также ссылки на фотографии. Длительность пребывания вычисляется автоматически при сохранении записи.

Кроме перечисленных таблиц, в базе данных присутствуют стандартные таблицы фреймворка Django (`auth_user`, `auth_group`, `auth_permission`, `django_session` и другие), обеспечивающие аутентификацию пользователей, управление правами доступа и хранение сессий.

Связи между таблицами организованы через внешние ключи:

- таблицы `camera_zone` и `camera_line` ссылаются на `camera`;
- таблицы `person_event` и `person_zone_session` связаны с `camera`, `camera_zone`, `camera_line` и `employee`;
- таблица `rusguard_event` ссылается на `employee`;
- таблица `vehicle` связана с `employee`, а `vehicle_event` – с `vehicle` и `employee`.

1.4 Реализация приложения

1.4.1 Реализация БД

При разработке использовался подход, при котором таблицы не создаются прямыми SQL-запросами, в системе реализация базы данных выполнена при помощи Django ORM. Структура базы данных описывается в виде моделей на языке Python в файле `models.py`, а физическое создание таблиц в PostgreSQL осуществляется автоматически с помощью механизма миграций.

Перед началом работы с базой данных необходимо выполнить команды создания и применения миграций:

- `python manage.py makemigrations`;

- python manage.py migrate.

Первая команда анализирует описанные модели и формирует файлы миграций, содержащие SQL-инструкции для создания таблиц и связей между ними. Вторая команда применяет эти миграции к базе данных, физически создавая таблицы.

Рассмотрим реализацию базы данных на примере основных таблиц:

Таблица camera. Модель описывает IP-камеры видеонаблюдения. Содержит поля для хранения RTSP-адреса, типа детекции, названия, координат расположения на 3D-сцене и признака активности. С таблицей связаны дочерние таблицы зон и линий через внешние ключи. Ниже приведен код модели Django Camera.

```
class Camera(models.Model):
    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'
```

Таблица camera_zone. Модель описывает контрольные зоны, привязанные к камере через внешний ключ camera_id. Полигон зоны хранится в формате JSON, что позволяет задавать границы произвольной формы. Поле description используется для хранения фамилий сотрудников, закрепленных за данной зоной. Ниже приведен код модели CameraZone.

```
class CameraZone(models.Model):
    name = models.CharField(max_length=150)
    camera = models.ForeignKey(Camera,
on_delete=models.CASCADE, db_column='camera_id')
    polygon = models.JSONField()
```

```

is_active = models.BooleanField(default=True)
    color = models.CharField(max_length=20, null=True,
blank=True)
    description = models.TextField(null=True, blank=True)
    class Meta:
db_table = 'camera_zone'

```

Таблица person_event. Журнал событий, фиксируемых видеоаналитикой. Каждое событие может быть связано с зоной, линией или сотрудником. Тип события определяется перечислением EVENT_TYPE_CHOICES, что ограничивает возможные значения и обеспечивает целостность данных. Далее приведен код модели PersonEvent.

```

class PersonEvent(models.Model):
    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, db_column='camera_id')
    zone = models.ForeignKey('CameraZone',
on_delete=models.SET_NULL, null=True, blank=True,
db_column='zone_id')
    line = models.ForeignKey('CameraLine',
on_delete=models.SET_NULL, null=True , blank=True,
db_column='line_id')
    employee = models.ForeignKey(Employee,
on_delete=models.SET_NULL, null=True, blank=True,
db_column='employee_id')
    track_id = models.IntegerField()
    event_type = models.CharField(max_length=30,
choices=EVENT_TYPE_CHOICES)
    event_time = models.DateTimeField()

```

```

duration_seconds = models.FloatField(null=True, blank=True)
metadata = models.JSONField(null=True, blank=True)
class Meta:
db_table = 'person_event'

```

Таблица person_zone_session. Хранит сессии пребывания сотрудников в зонах. При входе создается запись со статусом open, при выходе статус меняется на closed и фиксируется продолжительность. Такой подход позволяет отслеживать как завершённые, так и текущие пребывания. Ниже описана модель PersonZoneSession.

```

class PersonZoneSession(models.Model):
    STATUS_CHOICES = [
        ('open', 'Открыта'),
        ('closed', 'Закрыта'),
    ]
    camera = models.ForeignKey(Camera,
on_delete=models.CASCADE, db_column='camera_id')
    zone = models.ForeignKey('CameraZone',
on_delete=models.CASCADE, db_column='zone_id')
    employee = models.ForeignKey(Employee,
on_delete=models.SET_NULL, null=True, blank=True,
db_column='employee_id')
    track_id = models.IntegerField()
    entered_at = models.DateTimeField()
    exited_at = models.DateTimeField(null=True, blank=True)
    duration_seconds = models.FloatField(null=True, blank=True)
    status = models.CharField(max_length=10,
choices=STATUS_CHOICES, default='open')
    class Meta:
db_table = 'person_zone_session'

```

Таблица rusguard_event. Журнал событий, поступающих от СКУД RusGuard. Данные из этой таблицы используются для сверки с результатами видеоаналитики: время нахождения в зоне засчитывается только тем

сотрудникам, чье последнее событие является входом (enter). Далее приведен код модели RusGuardEvent.

```
class RusGuardEvent(models.Model):
    EVENT_TYPE_CHOICES = [
        ('enter', 'Вход'),
        ('exit', 'Выход'),
    ]
    employee = models.ForeignKey(Employee,
on_delete=models.SET_NULL, null=True, blank=True)
    employee_name = models.CharField(max_length=200)
    event_type = models.CharField(max_length=10,
choices=EVENT_TYPE_CHOICES)
    event_time = models.DateTimeField()
    details = models.CharField(max_length=200, null=True,
blank=True)
    class Meta:
        db_table = 'rusguard_event'
```

Таблица employee. Справочник сотрудников, используемый для связывания событий видеоаналитики и СКУД с конкретным человеком. Ниже приведена модель Employee.

```
class Employee(models.Model):
    name = models.CharField(max_length=100)
    surname = models.CharField(max_length=100)
    company = models.CharField(max_length=200)
    class Meta:
        db_table = 'employee'
```

Таблица vehicle_event. Журнал событий, связанных с въездом и выездом транспортных средств. При сохранении записи автоматически вычисляется продолжительность пребывания. Ниже приведена модель VehicleEvent.

```
class VehicleEvent(models.Model):
    number = models.CharField(max_length=500, null=True,
blank=True)
```

```

        vehicle = models.ForeignKey(Vehicle,
on_delete=models.CASCADE, null=True, blank=True)
        employee = models.ForeignKey(Employee,
on_delete=models.CASCADE, null=True, blank=True)
        arrival_time = models.DateTimeField(null=True, blank=True)
        departure_time = models.DateTimeField(null=True, blank=True)
        stay_duration = models.DurationField(null=True, blank=True)
        class Meta:
            db_table = 'vehicle_event'
            def save(self, *args, **kwargs):
                if self.arrival_time and self.departure_time:
                    self.stay_duration = self.departure_time - self.arrival_time
                super().save(*args, **kwargs)

```

Помимо перечисленных моделей, в системе присутствуют модели CameraLine и также стандартные модели Django (User, Group, Permission), обеспечивающие аутентификацию и управление правами доступа.

После применения миграций Django автоматически создает все описанные таблицы в базе данных PostgreSQL с соответствующими полями, индексами и внешними ключами. Для администрирования базы данных используется панель pgAdmin, подключенная к контейнеру PostgreSQL.

1.4.2 Реализация модуля Nginx

Модуль Nginx выполняет роль точки входа для всех клиентских запросов и распределяет запросы между разными модулями. Он включает два блока server: для приема соединений по HTTPS протоколу на порту 443 и для перенаправления HTTP-запросов с порта 80 на защищенное соединение.

В основном блоке настраивается SSL-шифрование с использованием сертификатов Let'sEncrypt, код настройки приведен далее.

```

listen 443 ssl;
ssl_certificate
/etc/letsencrypt/live/videoanalytics.xenus.dev/fullchain.pem;

```

```
ssl_certificate_key
/etc/letsencrypt/live/videoanalytics.xenus.dev/privkey.pem;
```

WebSocket-соединения проксируются на модуль видеообработки с отключенной буферизацией, для передачи видеопотока в реальном времени. Все API-запросы направляются в модуль django:

```
location ~ ^/(admin|api|static_backend|media_backend)/ {
    proxy_pass http://django:8000;
}
```

Второй блок server перенаправляет все HTTP-запросы на HTTPS. Таким образом, Nginx выполняет функции SSL-терминации, раздачи статических файлов, проксирования WebSocket-соединений и маршрутизации HTTP-запросов к внутренним сервисам.

1.4.3 Реализация модуля Django

Модуль Django реализует серверную логику приложения и предоставляет REST API для клиентской части. Он основан на фреймворке Django RESTFramework из нескольких компонентов компонентов.

Модели данных описаны в файле models.py. Они описывают все таблицы базы данных и связи между ними. Для работы с данными используется встроенный ORM Django, позволяющий обращаться к записям как к объектам Python без написания SQL-запросов вручную. Связи между таблицами организованы через внешние ключи: зоны и линии привязаны к камерам, события - к камерам, зонам, линиям и сотрудникам. Это связывает информацию между собой и позволяет получать связанную информацию в одном запросе, например, получить камеру вместе с ее зонами и линиями.

Сериализаторы в файле serializers.py преобразуют объекты моделей в формат JSON. Для каждой модели определен соответствующий класс. CameraSerializer может объединять связанные зоны и линии через CameraZoneSerializer и CameraLineSerializer, что позволяет получить полную конфигурацию камеры в одном запросе. PersonZoneSessionSerializer включает поля, вычисляемые в ходе выполнения кода: is_active (определяется по статусу

активности сессии) и `duration_formatted` (переводит формат времени). `VehicleEventSerializer` имеет поле `stay_duration`, доступное только для чтения, он вычисляется автоматически при сохранении данных.

Маршрутизация описана в `urls.py`. Аутентификация происходит при помощи JWT-токенов: `endpointsapi/token/`, `api/token/refresh/` и `api/token/verify/` обеспечивают получение, обновление и проверку токенов. Список камер доступен через `api/cameras/`. Данные по времени нахождения в зонах запрашивается через `api/camera/<camera_id>/work_time/`. Для логов СКУД используются `ViewSet`, он автоматически генерирует стандартный набор `endpoints`. Права пользователя можно получить через `api/user/permissions/`.

Обработчики в `views.py` реализуют бизнес-логику. Класс `CameraListView` возвращает все активные камеры, `CameraDetailView` – определенную камеру по идентификатору. Класс `EmployeeWorkTimeView` находит закрытые сессии пребывания на рабочем месте у сотрудников за указанный день, суммирует продолжительность по каждой зоне и возвращает результат. Класс `RusGuardEventsView` предоставляет доступ к списку событий СКУД с возможностью сортировать по определенным признакам. Класс `UserPermissionsView` находит права текущего пользователя из групп, в которых он состоит, и возвращает их списком для получения доступом к функциям системы.

Административная панель настраивается в файле `admin.py`. Камеры добавляются через стандартную форму с указанием названия, RTSP-адреса, типа детекции, координат. Зоны и линии добавляются на той же странице как дополнительные элементы. Для каждой зоны задается название, полигон, цвет и описание с именем закрепленного сотрудника, для каждой линии – название, две точки, направление отслеживания движения и цвет. Пользователи системы, группы и разрешения задаются стандартными средствами Django, это дает удобно настраивать доступ к функциям приложения у разных ролей.

1.4.4 Реализация модуля yolo

					ПГУ1.090301.22.001 ПЗ	Лист
						39
Изм.	Лист	№ докум.	Подпись	Дата		

Модуль YOLO является основной частью и выполняет самую ресурсоемкую задачу – обработку видеопотоков в реальном времени. Он реализован на языке Python в файле video_stream_server.py. Модуль выполняет захват кадров с IP-камер, детекцию людей нейросетевой моделью YOLOv8m, трекинг обнаруженных объектов с использованием алгоритма отслеживания SORT, анализ местоположения сотрудников, относительно заданных зон и линий, сверка с данными СКУД, а также сохранение данных в базу данных и передача клиентам обработанных кадров через WebSocket.

За асинхронную обработку камер отвечает библиотека Eventlet, которая обрабатывает нескольких камер в параллельных потоках. Библиотека Socket.IO необходима для обмена данными с клиентами в реальном времени. Сервер Socket.IO имеет параметры, разрешающие подключения с всех источников, и таймаутами для улучшения работы при долгой работе с системой. Ниже приведен код инициализации Socket.IO.

```
sio = socketio.Server(
    cors_allowed_origins='*',
    async_mode='eventlet',
    ping_timeout=60,
    ping_interval=25,
    max_http_buffer_size=10 * 1024 * 1024
)
```

Для удобного хранения данных используются классы данных. Класс Zone описывает зону и содержит идентификатор, название, полигон, цвет отображения и описание. Далее приведен код датакласса Zone.

```
@dataclass
class Zone:
    id: int
    name: str
    polygon: np.ndarray
    color: tuple = (0, 255, 255)
    description: str = None
```

Класс Line хранит информацию о контрольных линиях, которые задаются двумя точками, направлением контроля и цветом. Далее приведен код датакласса Line.

```
@dataclass
class Line:
    id: int
    name: str
    points: np.ndarray
    direction: str = 'both'
    color: tuple = (255, 0, 0)
```

Для контроля состояния зон используется класс ZoneState, хранящий все подтвержденные треки, и данные с временем входа и последнего обнаружения объекта. Ниже приведен код класса ZoneState.

```
@dataclass
class ZoneState:
    current_track_ids: set = field(default_factory=set)
    entered_at: dict = field(default_factory=dict)
    last_seen_at: dict = field(default_factory=dict)
    confirmed_track_ids: set = field(default_factory=set)
```

Модуль включает два основных класса. Класс CameraStreamer выполняет обработку видеопотока отдельной камеры. При инициализации он получает из данные камеры: идентификатор, название, RTSP-адрес, тип детекции, список зон и линий. Также настраиваются правила детекции из переменных окружения: таймаут присутствия, защитный интервал пересечения линий, время подтверждения зоны. Ниже код чтения из переменных окружения.

```
self.presence_timeout =
float(os.getenv('ZONE_PRESENCE_TIMEOUT', '2.0'))
self.line_cross_cooldown =
float(os.getenv('LINE_CROSS_COOLDOWN', '1.5'))
self.zone_confirmation_seconds =
float(os.getenv('ZONE_CONFIRMATION_SECONDS', '10.0'))
```

В трекер SORT при создании передаются параметры: максимальное время жизни трека, минимальное количество совпадения для подтверждения и порог пересечения рамок. Далее код инициализации алгоритма отслеживания.

```
self.tracker = Sort(
    max_age=sort_max_age,
    min_hits=sort_min_hits,
    iou_threshold=sort_iou_thresh
)
```

Захват видеопотока с IP-камеры реализован в методе run() с использованием GStreamer-пайплайна, который выполняет получение кадров по протоколу RTSP с декодированием H.265. Ниже приведен GStreamer пайплайн для захвата RTSP-потока.

```
gst_pipeline = (
    f'rtspsrc location={self.rtsp_url} '
    f'latency=100 drop-on-latency=true protocols=tcp '
    f'! rtph265depay ! h265parse ! avdec_h265 '
    f'! videoconvert ! video/x-raw,format=BGR '
    f'! appsink sync=false max-buffers=2 drop=true'
)
```

В бесконечном цикле кадры считываются методом cap.read() и передаются в process_frame(). Обработка кадра начинается с детекции людей моделью YOLOv8m. Модель загружается один раз при старте менеджера и выполняется на графическом процессоре. Ниже приведен код YOLO-модели.

```
results = self.model(
    frame,
    classes=self.detect_classes,
    conf=0.25,
    iou=0.45,
    verbose=False,
    device='cuda'
)
```

После трекинга в методе `process_zone_logic()` выполняется анализ зон. Проверяется местоположение каждого трека относительно каждой зоны с помощью функции `point_in_polygon()`. Далее приведен код проверки нахождения точки в полигоне.

```
def point_in_polygon(self, point, polygon):
    result = cv2.pointPolygonTest(polygon, (int(point[0]),
int(point[1])), False)
    return result >= 0
```

Параллельно ведется учет времени нахождения в зоне: если трек находится в зоне заданное в окружении время, ему присваивается имя сотрудника, привязанного к этой зоне. После подтверждения в базу данных записывается событие и создается сессия пребывания. Ниже приведен код фиксации входа в зону.

```
state.confirmed_track_ids.add(track_id)
self.manager.save_person_event(
    camera_id=self.camera_id,
    zone_id=zone.id,
    track_id=track_id,
    event_type='zone_enter',
    metadata={'confirmedAfterSec':
self.zone_confirmation_seconds}
)
self.manager.create_zone_session(
    camera_id=self.camera_id,
    zone_id=zone.id,
    track_id=track_id,
    metadata={'confirmedAfterSec':
self.zone_confirmation_seconds}
)
```

Одновременно происходит проверка пересечения линий в методе `update_line_crossings_db()`. Подтверждение входа в зону реализовано в методе `_update_zone_db_state()`.

После этого на кадр наносятся элементы: полигоны зон, линии, траектории движения треков, рамки объектов с подписями. Готовый кадр кодируется в JPEG и передается через Socket.IO всем клиентам, подписанным на эту камеру. Ниже представлен код отправки через Socket.IO.

```
sio.emit(f'camera_{self.camera_id}_frame', {
    'timestamp': time.time()
    'detections': detection_count,
    'track_ids': track_ids,
    'fps': self.real_fps,
    'frame': frame_base64,
}, room=f'camera_{self.camera_id}')
```

Класс VideoStreamManager управляет жизненным циклом всех камер. Он создает пул подключений к PostgreSQL, загружает модель YOLOv8m, получает из базы данных список активных камер с требуемым типом детекции. Создает экземпляр CameraStreamer в отдельном потоке для всех камер. Далее приведен код асинхронного запуска всех стримеров камер.

```
for camera_id, streamer in self.cameras.items():
    greenlet = eventlet.spawn(streamer.run)
    self.greenlets[camera_id] = greenlet
```

Запись в базу данных выполняются асинхронно через очередь _db_queue. Отдельный поток выполняет SQL-запросы из этой очереди, обработка запросов записи в базу данных не блокирует обработку видеопотоков на время ожидания ответа от СУБД. Ниже приведен код фонового обработчика для задач БД.

```
def _db_worker(self):
    while self.running:
        try:
            task = self._db_queue.get(timeout=1)
            task()
        except eventlet.queue.Empty: pass
```

1.4.5 Реализация веб-приложения

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, аналитическую статистику и логи событий СКУД.

Статические данные принимаются через HTTP-запросы к REST API, реализованные в файле `api.js`. Видеоряд принимается через WebSocket-соединение, управляется в `videoStream.js`. Аутентификация основано на JWT-токена. После авторизации токены сохраняются в браузере, фрагмент кода асинхронной авторизации представлден далее.

```
asynclogin(username, password) {
  const response = await fetch(`${API_BASE_URL}/api/token/`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username, password }),
  });
  const data = await response.json();
  localStorage.setItem('access_token', data.access);
  localStorage.setItem('refresh_token', data.refresh);
  return data;
}
```

При вызове метода `subscribe()` сервис проверяет наличие соединения и подписывается на камеру. После успешной подписки сервер начинает передавать обработанные кадры в формате `base64`, которые распределяются между всеми клиентами, подписанными на данную камеру. Ниже представлен код подписки на камеру.

```
subscribe(cameraId, detectionType, onFrame, onError) {
  const socket = this.getSocketByDetectionType(detectionType);
  socket.emit('subscribe', { cameraId }, (response) => {
    if (response?.error && onError) onError(response.error);
  });
  socket.on(`camera_${cameraId}_frame`, (data) => {
    const handlers = this.listeners.get(cameraId) || [];
    handlers.forEach(handler => handler(data));
  });
}
```

```
});}
```

Компонент CameraPage отображает видеопоток с выбранной камеры и аналитическую информацию. При получении каждого кадра обновляются счетчик детекций, количество обнаруженных объектов и частота кадров. Для камер с детекцией в правой части страницы отражается статистика по времени нахождения сотрудников в зонах за текущий день.

В нижней части страницы отображаются последние события СКУД. Показывается таблица с тремя последними записями: фамилия сотрудника, тип события, время и детали. При нажатии на кнопку «Открыть все логи» раскрывается полная таблица событий. Доступны поиск по фамилии сотрудника, фильтрация по типу события и дате. Параметры поиска передаются в строке запроса к REST API, фрагмент этого кода представлен ниже.

```
const params = new URLSearchParams();  
if (employee_name) params.append('employee_name',  
employee_name);  
if (event_type) params.append('event_type', event_type);  
params.append('page', page);  
params.append('per_page', perPage);
```

1.4.6 Тестирование

Тест-кейс №1. Загрузка главной страницы.

Цель проверки: зайти на веб-страницу <https://videoanalytics.xenus.dev/>, проверить что при заходе на страницу начинается загрузка.

Исходные данные: пользователь не авторизован, сервер работает исправно.

Ожидаемый результат: в центре экрана отобразится индикатор и этап загрузки страницы.

Результат тестирования: после перехода на страницу <https://videoanalytics.xenus.dev/> начинается загрузка страницы, отображается индикатор загрузки и надпись с этапом загрузки. После этого, открывается окно авторизации. Загрузка представлена на рисунке 3.

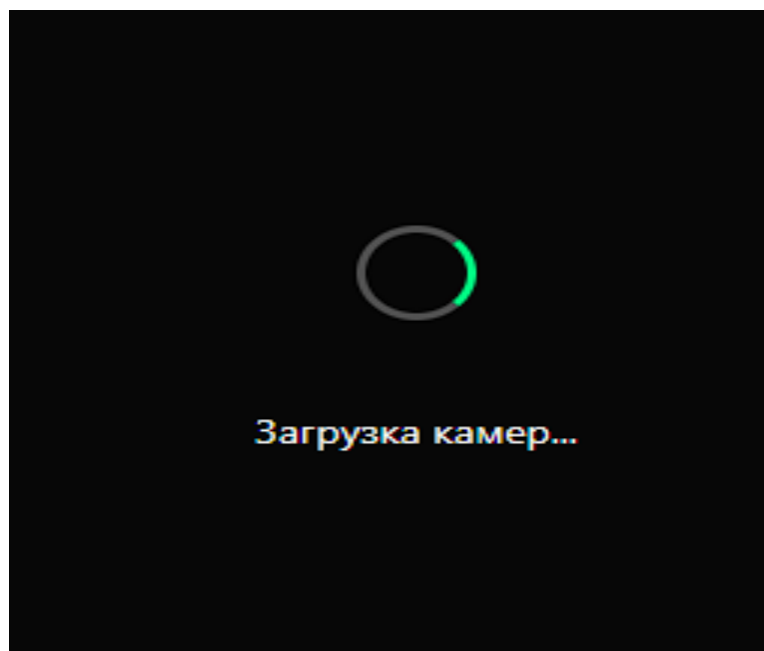


Рисунок 3– Загрузка главной страницы

Тест-кейс №2. Ошибка загрузки главной страницы.

Цель проверки: проверить, реакцию системы, если сервер недоступен, какое уведомление об этом получит пользователь.

Исходные данные: сервер недоступен.

Ожидаемый результат: вместо значка загрузки на экране отображается сообщение об ошибке и кнопку для перезагрузки страницы.

Результат тестирования: когда сервера недоступен, страница выводит сообщение “Внутренняя ошибка сервера” вместе с кнопкой “Повторить”. Результат представлен на рисунке 4.

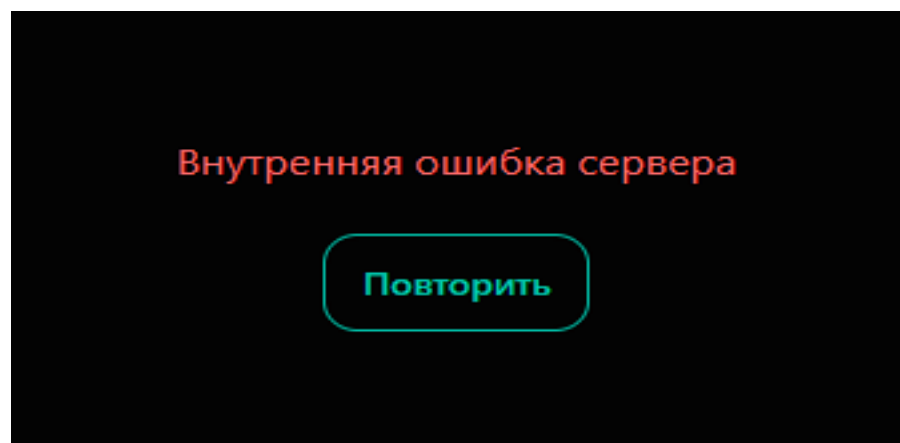


Рисунок 4 – Ошибка загрузки главной страницы

Тест-кейс №3. Успешная авторизация.

Цель проверки: убедиться, что при вводе правильных учетных данных пользователь авторизуется в системе и его направит на главную страницу.

Исходные данные: пользователь имеет учетную запись, вводит правильные логин и пароль.

Ожидаемый результат: после нажатия на кнопку «Войти» происходит перенаправление на главную страницу системы.

Результат тестирования: при вводе корректных данных система выполняет вход и открывает главную страницу. Окно авторизации представлено на рисунке 5.

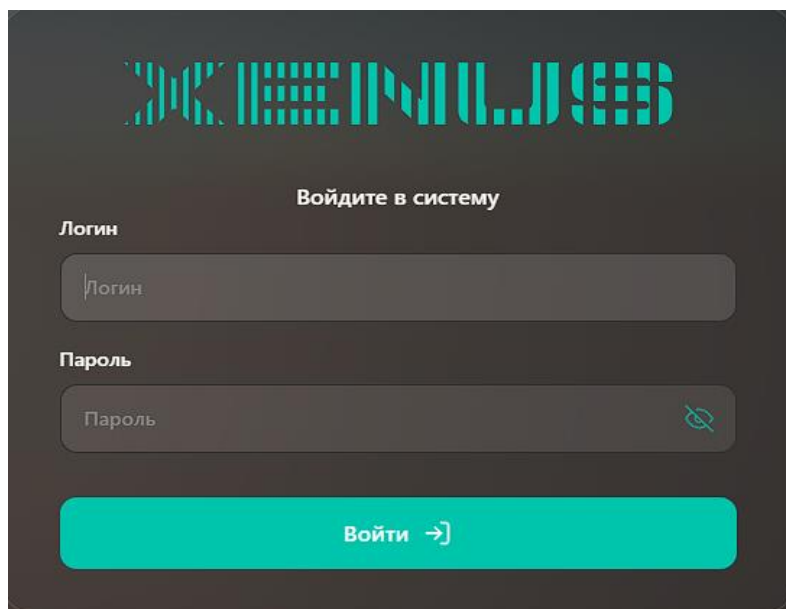


Рисунок 5 – Окно авторизации

Тест-кейс №4. Авторизация с неверными данными.

Цель проверки: убедиться, что при вводе неверных данных пользователя система отклонит попытку входа и выведет сообщение об ошибке авторизации.

Исходные данные: пользователь вводит неверный логин или пароль.

Ожидаемый результат: выводится сообщение «Не найдено активной учетной записи с указанными данными», не происходит переход на основную страницу.

Результат тестирования: при вводе некорректных данных система отклоняет попытку вход и выводит ошибку. Результат представлен на рисунке 6.

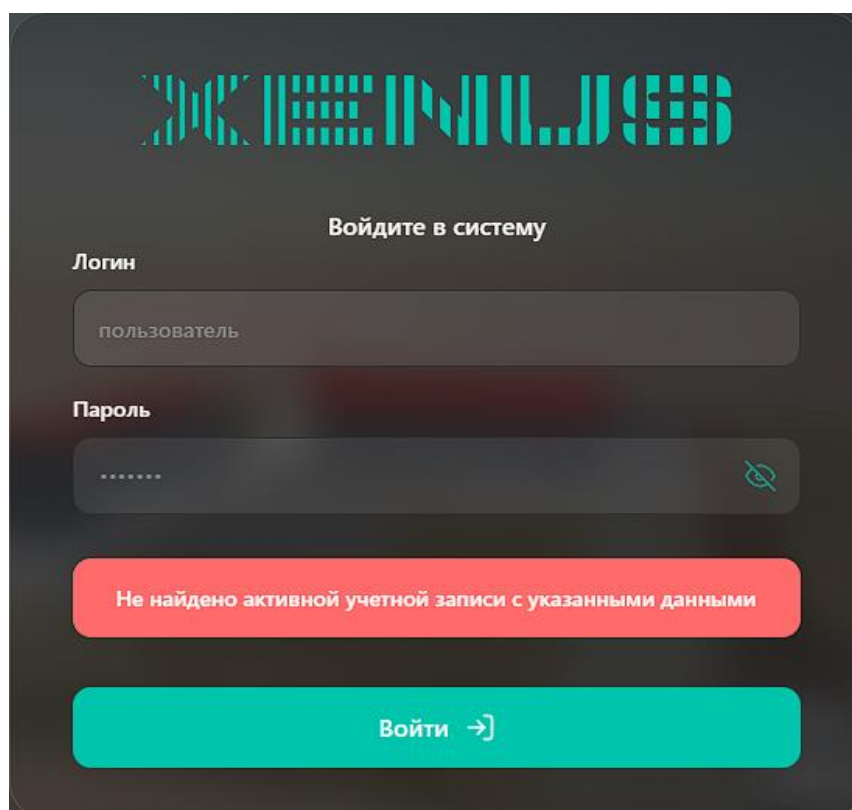


Рисунок 6 – Введены неверные данные

Тест-кейс №5. Отображение главной страницы.

Цель проверки: убедиться, что после авторизации происходит переход на основную страницу.

Исходные данные: пользователь авторизовался.

Ожидаемый результат: отображается главная страница с трехмерной моделью территории, меню выбора камер слева сверху, имя пользователя и кнопка «Выйти» справа сверху. Результат на рисунке 7.

Тест-кейс №6. Открытие камеры через список.

Цель проверки: убедиться, что при нажатии на элемент из меню камер, откроется страница данной камеры.

Исходные данные: пользователь находится на главной странице. Раскрыть меню камер, если оно закрыто, выбрать и нажать на пункт меню.

Ожидаемый результат: открывается страница просмотра видеопотока выбранной камеры.

Результат тестирования: страница камеры открылась.

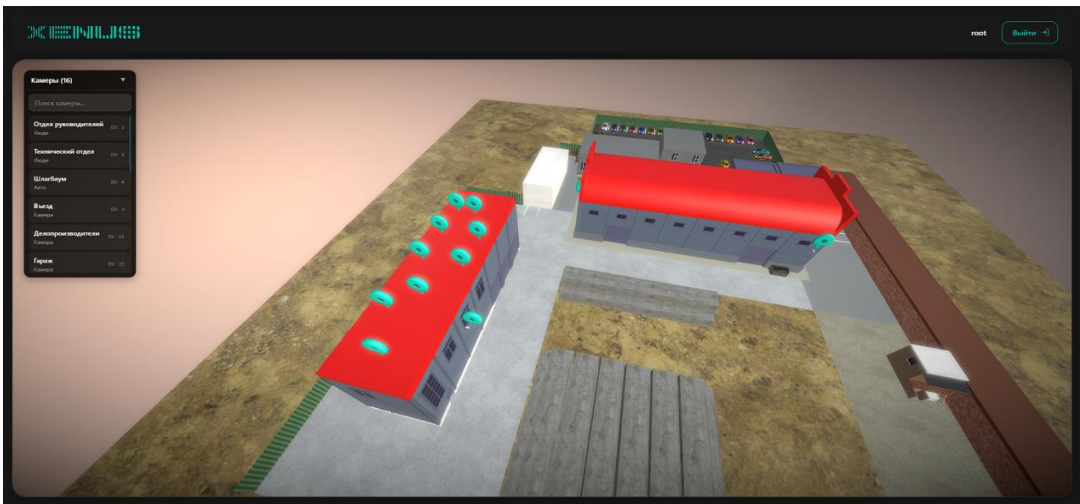


Рисунок 7 – Главная страница

Тест-кейс №7. Открытие камеры через метку на 3D-сцене.

Цель проверки: проверить, что при нажатии на метку камеры на открывается страница видеопотока.

Исходные данные: пользователь находится на главной странице, метки камер отображаются, требуется нажать на метку камеры на 3D-сцене.

Ожидаемый результат: открывается страница просмотра видеопотока выбранной камеры.

Результат тестирования: страница камеры открывается. Метки камер представлены на рисунке 8.

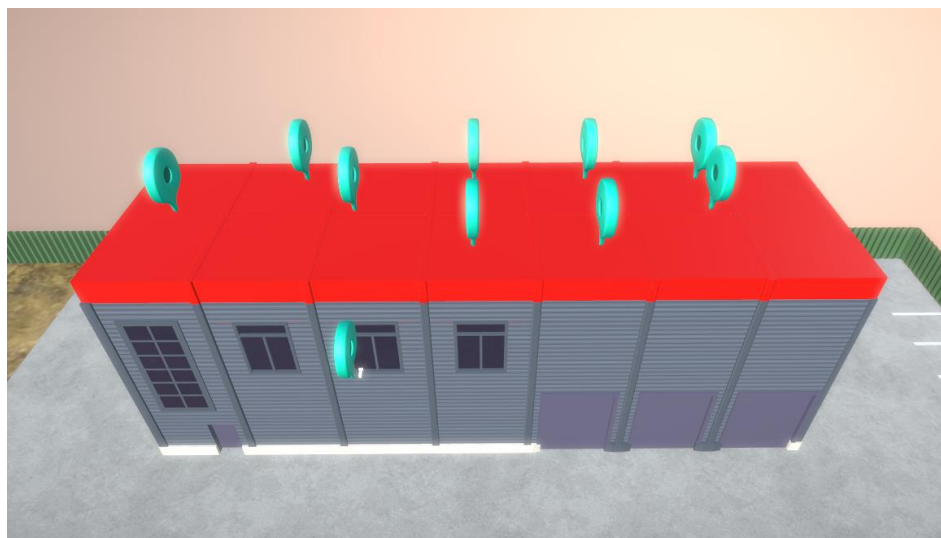


Рисунок 8 – Метки камер

Тест-кейс №8. Загрузка страницы камеры и отображение видеопотока с детекцией.

Цель проверки: проверить открытие камеры, отображение индикатора загрузки, после чего вывод видеопотока с данными.

Исходные данные: пользователь открыл камеру, которая отслеживает людей.

Ожидаемый результат: во время загрузки отображается индикатор и статус «Подключение к видеопотоку», затем появляется видеопоток с зонами, линиями и рамками вокруг людей.

Результат тестирования: индикатор загрузки отображается, после подключения транслируется видео с результатами детекции. На рисунке 9 представлена загрузка страницы камеры, на рисунке 10 – страница просмотра видеопотока с детекцией.

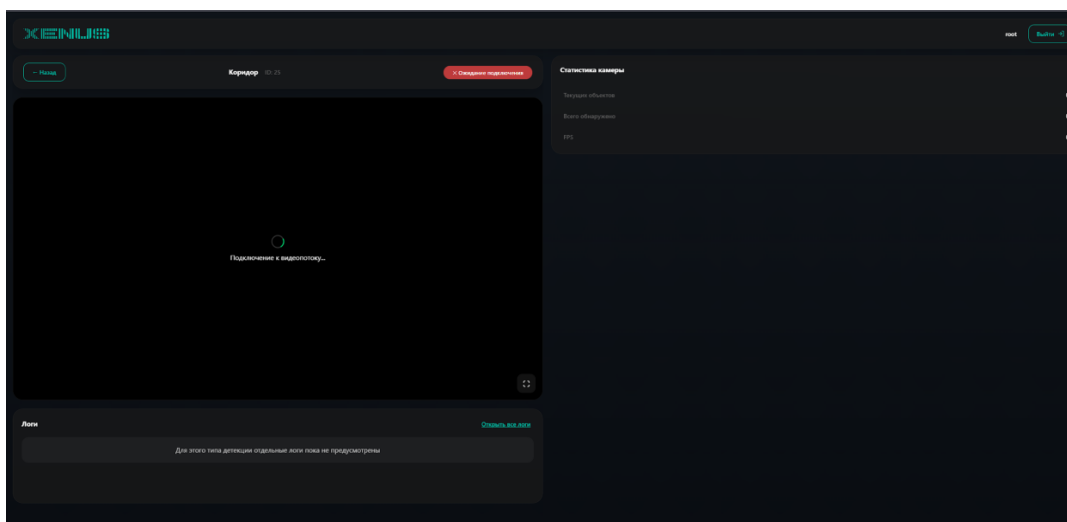


Рисунок 9— Загрузка страницы просмотра видеопотока

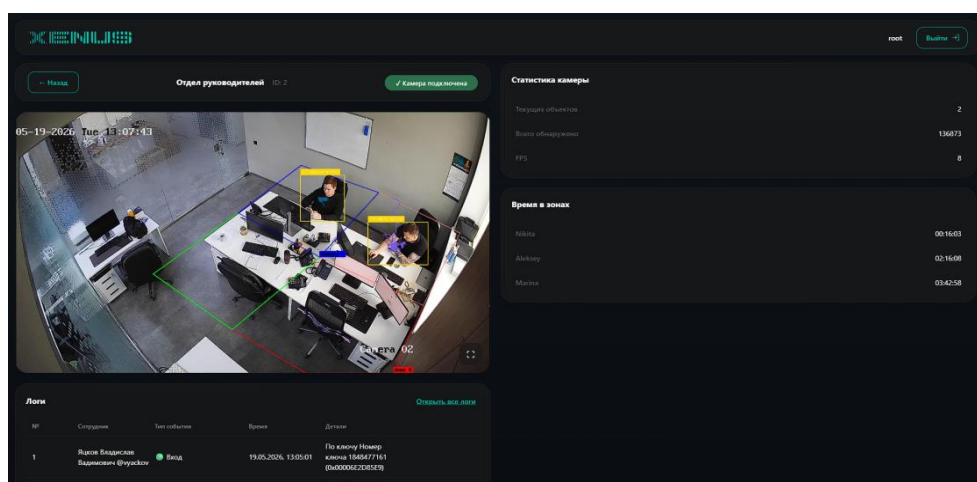


Рисунок 10 – Страница просмотра видеопотока

Тест-кейс №9. Отображение видеопотока без обработки.

Цель проверки: проверить, что при открытии камеры без детекции, открывается страница, на которую передается необработанный видеоряд.

Исходные данные: пользователь открыл камеру без детекции.

Ожидаемый результат: видеопоток не обрабатывается, нет рамок, зон и линий.

Результат тестирования: видеопоток транслируется без обработки. Результат представлен на рисунке 11.

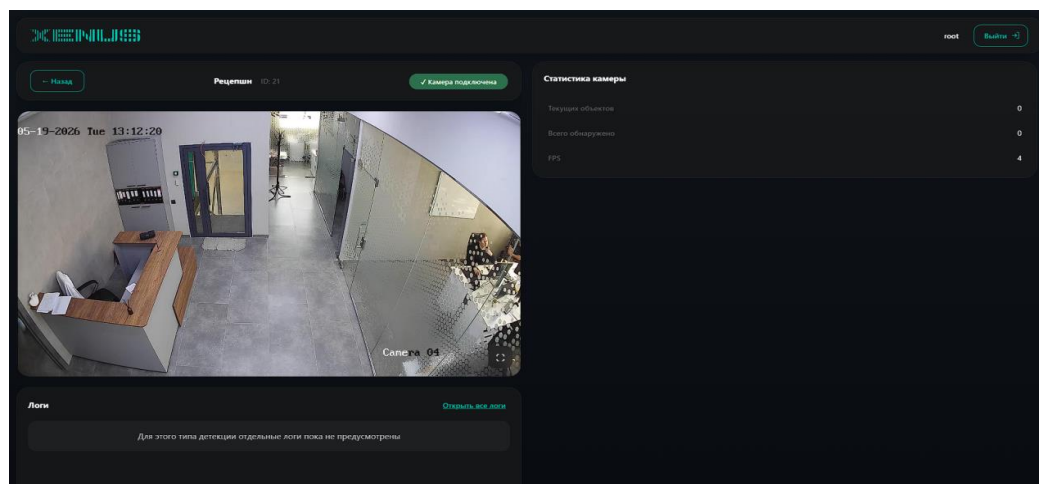


Рисунок 11 – Страница просмотра видеопотока без детекции

Тест-кейс №10. Открытие логов СКУД.

Цель проверки: убедиться, что при нажатии на кнопку «Открыть все логи» открывается окно с таблицей логов СКУД.

Исходные данные: пользователь находится на странице камеры с детекцией людей, нажимает на кнопку “Открыть все логи”.

Ожидаемый результат: открывается модальное окно с таблицей событий, на каждой странице по 10 записей, поиск и сортировка работают.

Результат тестирования: окно событий СКУД открывается, навигация и поиск работают. Окно логов представлено на рисунке 12.

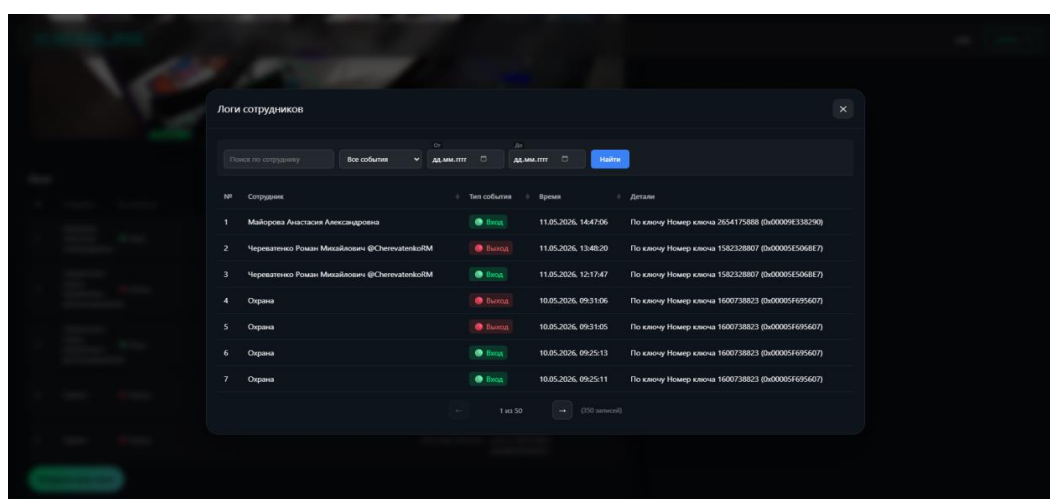


Рисунок 12 – Просмотр логов

1.4.7 Контейнеризация и развертывание системы

Модули системы упакованы в Docker-контейнеры, поэтому они могут работать независимо друг от друга. Это упрощает процесс развертывания системы и позволяет запускать бэкенд-приложение на любом сервере без необходимости вручную скачивать зависимости и настраивать окружение.

Для сервера выбрана операционная система Ubuntu. Для начала работы требуется установить Docker и DockerCompose. Установка происходит через выполнение команд в терминале:

```
sudo apt update
sudo apt install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -
o /etc/apt/keyrings/docker.asc
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
docker-compose-plugin
```

Для каждого модуля создается свой отдельный Dockerfile – в нем описывается последовательность действий для развертывания контейнера. В этом файле задается базовый образ, копируются исходные файлы модуля, устанавливаются зависимости из файла requirements.txt и подключается рабочее окружение. Например, в модуле видеоаналитики в Dockerfile используется образ Python с поддержкой CUDA для использования тензорных ядер видеокарты, а для модуля бизнес-логики используется стандартный Python с установкой Django и требуемых библиотек. Пример Dockerfile представлен на рисунках 13, requirements.txt - на рисунке 14.

```
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 \
    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 -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 13 – Код Dockerfile

```
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

redis==5.0.1
```

Рисунок 14 – Код requirements.txt

Взаимодействие между контейнерами и порядок запуска системы описаны в файле docker-compose.yml. Следующие сервисы системы описываются далее.

db – контейнер с PostgreSQL, выполняющий функцию хранения данных системы.

django – контейнер модуля бизнес-логики, после выполнения миграций базы данных запускает сервер на порту 8000.

yolo2 – контейнер модуля видеоаналитики, он использует GPU для обработки видеопотоков.

nginx – контейнер веб-сервера, принимает запросы на портах 80 и 443, отправляет статические файлы клиентам и проксирует API-запросы к Django и WebSocket-соединения к модулю видеоаналитики.

Контейнеры объединены в единую сеть, поэтому они могут взаимодействовать друг с другом, используя названия сервисов. Для модулей, требующих графический процессор, в конфигурации резервируется видеокарта NVIDIA. Фрагмент файла docker-compose.yml представлен на рисунке 15.

```
yolo2:
  profiles: ["no start"]
  build: ./yolo2
  depends_on:
    db:
      condition: service_healthy
  volumes:
    - media_volume:/app/media
    - ./logs:/app/logs
    - /home/xenusad/VA/debug_faces:/home/admin/debug_faces:rw
  env_file:
    - ./yolo2/.env
  ports:
    - "5001:5001"
    - "8555:8554"
  deploy:
    resources:
      reservations:
        devices:
          - driver: nvidia
            count: all
            capabilities: [gpu]
  networks:
    - videoanalytics_network
  restart: unless-stopped
```

Рисунок 15 – Пример кода docker-compose.yml

Использование Docker в разработке позволило упаковать каждый модуль системы в самостоятельный контейнер со всеми необходимыми зависимостями, настроить взаимодействие контейнеров через конфигурационный файл и упростить процесс развертывания всей системы на сервере.

2 Охрана труда

Разработка программного обеспечения может казаться безопасной, но на самом деле длительная работа за компьютером может негативно сказываться на здоровье человека. Есть множество скрытых угроз: электрический ток, шум персонального компьютера, электромагнитное излучение, накопление статического электричества, а также ухудшение состава воздуха из-за низкой ионизации. Это может привести к высокой утомляемости.

Одна из основных проблем – это отсутствие физической нагрузки, при малой подвижности страдает большинство групп мышц: рук, шеи и спины, со временем это может стать постоянной проблемой. Постоянное напряжение зрения перед монитором ведёт к переутомлению глаз. В этом разделе описано, как снизить воздействие вредных факторов: правильно организовать рабочее место и регулировать отдых.

2.1 Классификация и категорирование помещения

При работе с компьютерной техникой нужно оценить безопасность помещения с точки зрения пожарной и электрической безопасности. Для этого есть специальные классификации.

Все помещения делят на три класса электрической безопасности. Первый – это помещения без повышенной опасности. Обычно это помещения с низкой влажностью без пыли, с нормальной температурой и полами, не проводящими электричество. Помещения этой классификации представляют наименьший риск из-за отсутствия факторов повышенной опасности.

Второй класс – помещения с повышенной опасностью. Эта классификация присваивается, если есть один из следующих факторов: влажность воздуха выше 75%, температура поднимается выше +35°C, в воздухе присутствует токопроводящая пыль, пол из металла, бетона или кирпича.

Третий класс – особо опасные помещения. Сырость как на улице (влажность почти 100%, выделяет), либо сочетания нескольких условий из

прошлых пунктов. В таких местах лучше не использовать компьютерную технику.

Обычный офис относится к первому классу – то есть без повышенной опасности. Это значит, что соблюдены все требования безопасности и риск получить удар током крайне мал.

Также важно соблюсти все условия пожарной безопасности. Помещения делятся на категории от А до Д. Категории А и Б – это цеха с газами, бензином или горючей пылью, они представляют наибольшую опасность. Категория В – это склады, предназначенные для хранения горючих материалов, таких как: бумага, дерево или масло. Категория Г – это цеха для работы с печами или сваркой. Категория Д – самая безопасная, тут хранятся или используются холодные негорючие вещества. Например, обычный офис со столами, стульями, компьютерами относятся к категории Д, иногда можно не присваивать категорию, потому что угрозы пожарной безопасности нет. Для помещений категории Д достаточно стандартных мер предосторожности: огнетушитель и план эвакуации.

Обычное офисное помещение с влажностью не выше 60%, температурой около 20–22°C, полами из диэлектрического материала и вентиляцией отлично подходит для разработки программного обеспечения.

2.2 Санитарно-гигиеническая характеристика производства

Организация рабочих мест за компьютерами регулируется несколькими документами – Трудовым кодексом, санитарными правилами СанПиН и другими нормативами. Основная цель – предоставить возможность работать без угрозы для здоровья.

В помещениях, где работают с компьютерами, окна лучше делать на север или северо-восток, таким образом меньше ярких солнечных лучей будет попадать в помещение. На окна необходимо повесить жалюзи или приклеить светозащитную пленку. Работать с компьютером можно только при условии

					ПГУ1.090301.22.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		58

должного уровня освещения, это можно проверить при помощи специальных расчетов.

Искусственный свет должен быть равномерным, не должно быть резких перепадов. Нормальная освещённость на рабочем месте – 300-500 люкс. Лампы должны иметь нормальную цветопередачу (индекс Ra от 85%). Нельзя ставить монитор так, чтобы источник света был прямо перед глазами или за спиной – это создает блики и напрягает зрение. Яркость бликов на экране не должна превышать 40 кд/м², а потолка – 200 кд/м².

Содержание любых вредных веществ в воздухе не должно выходить за пределы установленных норм. Аналогично с уровнем шума и вибрации – они не должны превышать норм, установленных для жилых зданий и офисных помещений. Оборудование, которое создает высокий уровень шума (например, сервера или принтеры), необходимо вынести в отдельное помещение.

Параметры микроклимата в помещении зависят от затрат энергии организма. Работа за компьютером относится к лёгким категориям – Ia (до 139 Вт) и Ib (140–174 Вт). Для холодного периода года требуется температура – 21–24°C, для тёплого – 22–25°C. Влажность в помещении должна составлять 40–60%, а скорость движения воздуха не более 0,1 м/с.

Также есть правила организации рабочего пространства, площадь одного рабочего места с монитором должна составлять не менее 4,5 квадратных метров. Между столами с мониторами должно быть минимум 2 метра, а между боковыми стенками мониторов – 1,2 метра. Экран должен располагаться на расстоянии 50–70 см от глаз, в зависимости от размера символов.

Рабочее кресло должно соответствовать некоторым критериям: иметь подъёмно-поворотный механизм, регулировку высоты, угла наклона сиденья и спинки. Материал должен быть полумягкой и не накапливать статическое электричество.

Отдельно стоит выделить момент с организацией трудового процесса. При восьмичасовой работе разработчикам рекомендуется делать перерыв на 15 минут

каждый час работы. В ситуациях, где это невозможно каким-то причинам, нельзя непрерывно работать за ПЭВМ дольше 4 часов. Рекомендуется чередовать работу за компьютером с другими задачами.

2.3 Электробезопасность

По санитарным нормам, помещения с ПЭВМ обязательно должны быть оборудованы защитным заземлением в соответствии с техническими требованиями—это требование технической эксплуатации вычислительной техники. Рабочие места не должны располагаться рядом с силовыми кабелями, высоковольтными трансформаторами и другим оборудованием, которое может создавать помехи в работе ПЭВМ.

Питание для комнат с компьютерами желательно подводить от отдельного независимого источника электроэнергии. Это стабилизирует напряжение и уменьшит помехи.

При повседневной работе стоит соблюдать ряд рекомендаций. Нужно проверять, надежность соединения в трехпроводных розетках. Если сеть в помещении двухпроводная, рекомендуется подключать дисплей через специальное согласующее устройство. Системный блок нельзя ставить в влажном или пыльном месте, а также на полу у ног – его можно повредить или загрязнить. Компьютер должен стоять на надежной подставке, она должна быть прочной и не допускать тряску. Нельзя ставить системный блок или гарнитуру на поверхности, которые накапливают статическое электричество, это может навредить технике.

2.4 Пожарная безопасность

По закону о пожарной безопасности четко прописано обязанности начальства организации. Нужно проводить обязательные беседы и инструктажи с сотрудниками, рассказать, что делать в случае возгорания. Руководство обязаны контролировать, чтобы все системы пожаротушения и оповещения были в

рабочем состоянии, чтобы присутствовали огнетушители. Если случился пожар, сотрудники организации должна предоставить доступ к зданию и способствовать работе пожарных подразделений. При любом возгорании нужно немедленно звонить в пожарную службу.

При инструктаже сотрудников необходимо ознакомить с тем, где расположены эвакуационные выходы, водопровод и система оповещения пожарной охраны. Ознакомить с правилами обращения с огнем и проведения опасных работы. А также с последовательностью действий при пожаре. Инструктажи бывают разных видов: вводный для новичков, первичный, повторный, внеплановый.

При тушении возгораний в офисах с компьютерной техникой лучше всего использовать хладоновые огнетушители – они не повреждают технику. Тушение порошковыми и углекислотными огнетушителями запрещено для оборудования под напряжением выше 1 кВ и 10 кВ соответственно. Водные и воздушно-пенные огнетушители нельзя использовать при тушении техники, тушение техники под напряжением водой может представлять угрозу для жизни.

2.5 Искусственное освещение помещения

Задача расчета искусственного освещения сводится к определению необходимого количества источников света и мощности используемых ламп, которые в совокупности обеспечат нормированную освещенность рабочих мест.

Для освещения помещения выбрана система общего освещения, поскольку она даёт равномерное распределение света по всем рабочим поверхностям и уменьшает количество бликов. Общее освещение подходит для офисных помещений с компьютерами, так как разное освещение между экраном и окружением негативно влияют на зрение.

При проектировании освещения необходимо учитывать свет, отражённый от стен, потолка и остального окружения. Чем больше светлых поверхностей, тем больше света будет в помещении и тем эффективнее используются лампы.

					ПГУ1.090301.22.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Нормы освещения на рабочих местах установлены в санитарных правилах в зависимости от типа работ. Для помещений, где ведется работа с ПЭВМ, освещенность должна составлять 300–500 люкс. Также необходимо учитывать тип светильников и высоту их размещения – это влияет на равномерность распределения света по помещению.

При выборе освещения нужно учитывать коэффициент запаса, он компенсирует ухудшение освещения, связанное со временем – лампы теряют яркость и на них скапливается пыль.

В современных офисах чаще всего отдают предпочтение светодиодным светильникам. Они обладают высокой эффективностью, долгим сроком работы и хорошей цветопередачей, что удовлетворяет всем потребностям для работы оператора ПЭВМ.

					ПГУ1.090301.22.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения данной работы было изучено применение технологий компьютерного зрения, методы обработки и захвата видеопотоков с камер по протоколу RTSP, методы детекции и трекинга людей с использованием нейросетевой модели YOLOv8m.

Были получены навыки разработки программного обеспечения с использованием фреймворка Django на языке Python, опыт создания клиентского веб-приложения React с использованием протокола WebSocket для обмена данными в реальном времени. Также были освоены навыки работы с СУБД PostgreSQL. Были освоены навыки контейнеризации с использованием Docker, а также навыки настройки веб-сервера Nginx.

В рамках данной работы были разработаны: алгоритм обнаружения и отслеживания людей на основе YOLOv8m и SORT, алгоритм отслеживания людей в контрольных зонах, алгоритм сверки с событиями СКУД, серверный модуль видеообработки на Python с асинхронной передачей кадров через Socket.IO, модуль бизнес-логики на Django с REST API, веб-приложение с отображением видеопотоков и данных аналитики. Также была спроектирована база данных для хранения данных системы.

В дальнейшем систему можно усовершенствовать, добавив модуль распознавания лиц для автоматической идентификации сотрудников без привязки к контрольным зонам. Заняться масштабированием системы для одновременной обработки большего числа камер, добавление функции сохранения и просмотра записанного видеопотока, а также реализовать выгрузки отчетов в виде документов.

Список использованных источников

1. Официальный сайт Мегафона. [Электронный ресурс] – URL: https://penza.megafon.ru/corporate/services/videonabludenie_smart (дата обращения 25.05.2026).
2. Документация VisionLabs. [Электронный ресурс] –URL: <https://docs.visionlabs.ru/> (дата обращения 25.05.2026).
3. Официальный сайт видеонаблюдения TRASSIR. [Электронный ресурс] –URL: https://www.dssl.ru/l/trassir_business_solutions/ (дата обращения 25.05.2026).
4. Официальная документация PyCharm. [Электронный ресурс] – URL: <https://www.jetbrains.com/help/pycharm/getting-started.html> (дата обращения 25.05.2026).
5. Марк Лутц. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с., ил.
6. Ричард Стоунз, Нейл Мэттью. PostgreSQL. Основы. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 640 с., ил. ISBN 5-932 86-043-X.
7. Raphael Holzer. OpenCV tutorial Documentation Release 2019. [Электронный ресурс] – URL: https://opencv-tutorial.readthedocs.io/_/downloads/en/latest/pdf/ (дата обращения 25.05.2026).
8. GStreamer. [Электронный ресурс] – URL: <https://gstreamer.freedesktop.org/documentation/> (дата обращения 25.05.2026).
9. Django. [Электронный ресурс] / Режим доступа: <https://docs.djangoproject.com/en/6.0/> (дата обращения 25.05.2026).
10. Официальная документация Visual Studio Code. [Электронный ресурс] – URL:<https://code.visualstudio.com/docs/> (дата обращения 25.05.2026).
11. Yolo инструкция. [Электронный ресурс] / Режим доступа: <https://yolo-docs.readthedocs.io/en/latest/> (дата обращения 25.05.2026).
12. Мардан Азат, БарклундМортен. React быстро. 2-е межд. изд. – СПб.: Питер, 2024.

13. Официальная документация Nginx. [Электронный ресурс] / Режим доступа: <https://nginx.org/ru/docs/> (дата обращения 25.05.2026).

14. Официальная документация Docker. [Электронный ресурс] / Режим доступа: <https://docs.docker.com/> (дата обращения 25.05.2026).

15. Инструкция SORT. [Электронный ресурс] / Режим доступа: <https://blog.roboflow.com/sort-explained-real-time-object-tracking-in-python/> (дата обращения 25.05.2026).

16. Руководство пользователя RusGuard. [Электронный ресурс] / Режим доступа: <https://www.rgsec.ru/help/> (дата обращения 25.05.2026).

					ПГУ1.090301.22.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А

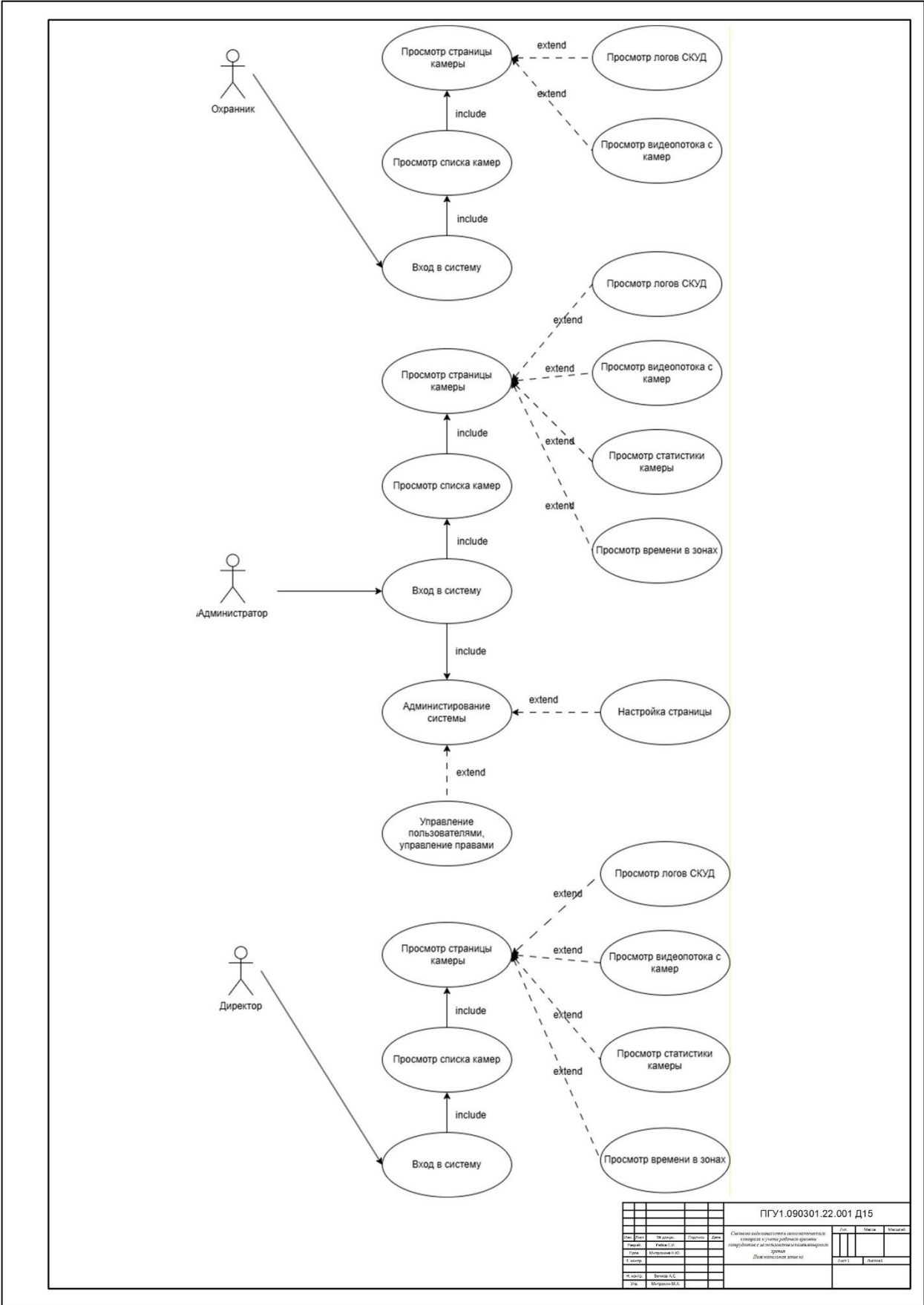
(обязательное)

Система видеоаналитики автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения

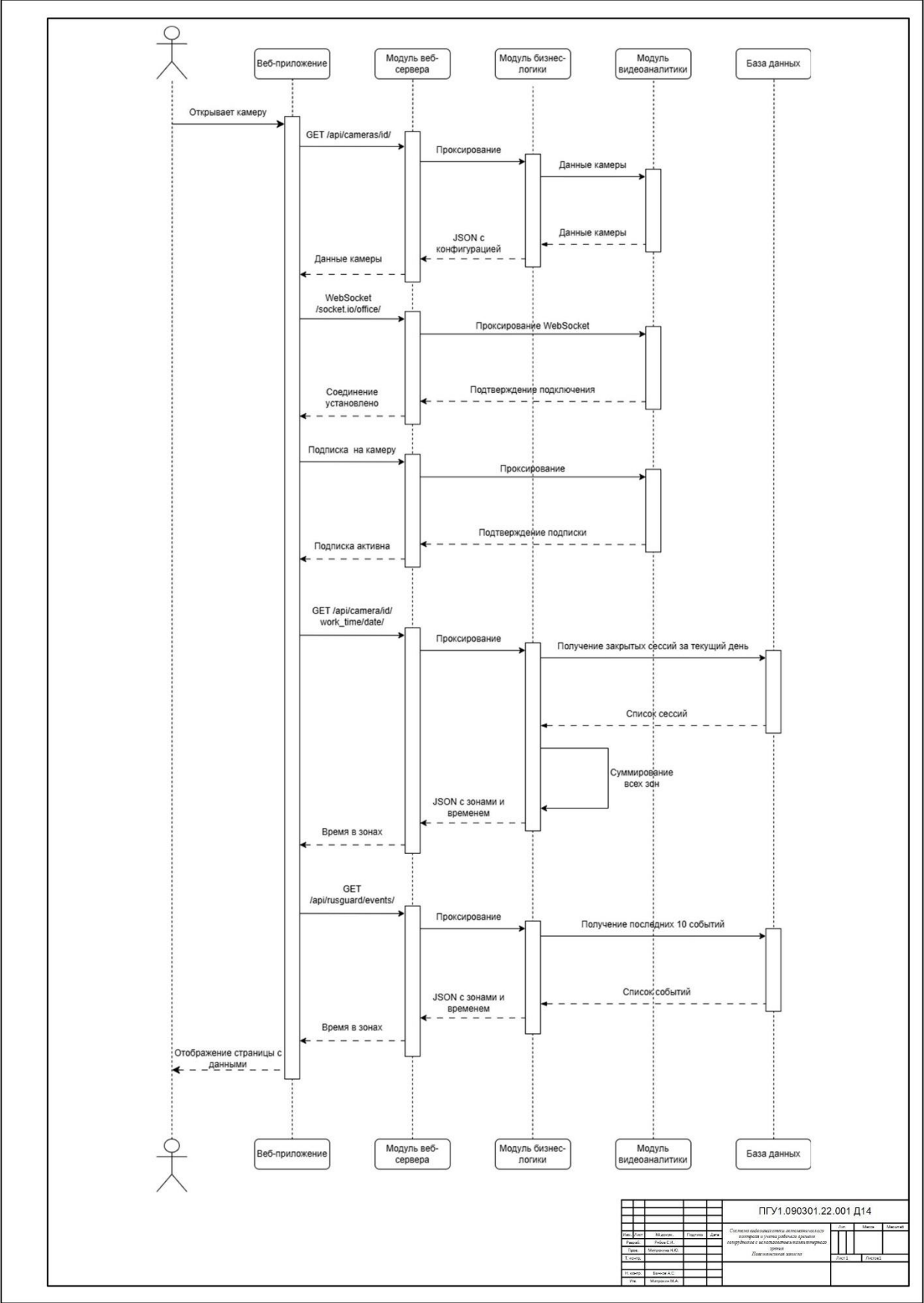
UML диаграммы

					ПГУ1.090301.22.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

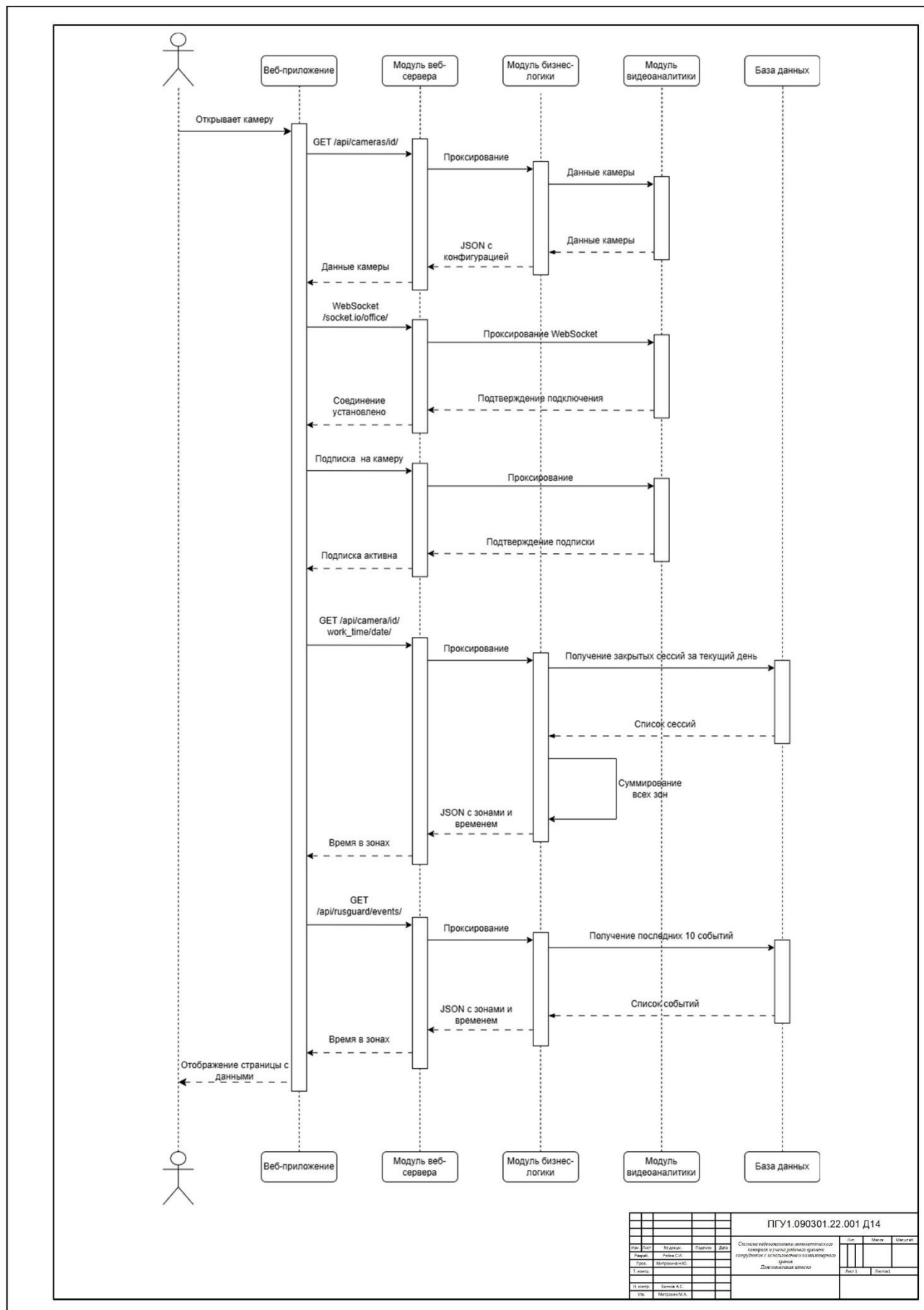
Приложение А.1 - Диаграмма вариантов использования



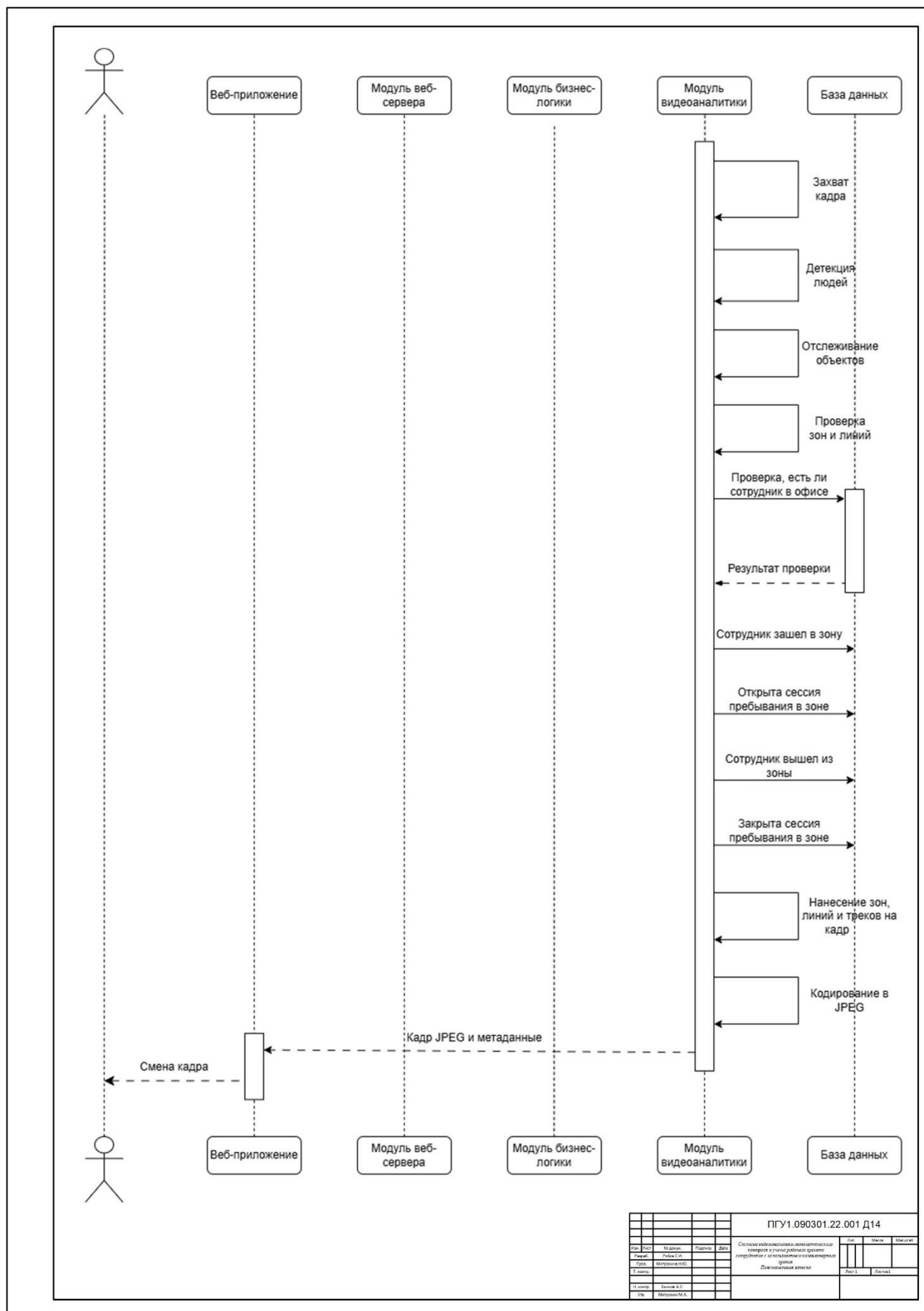
Приложение А.2 - Диаграмма последовательностей



Приложение А.3 - Диаграмма последовательностей



Приложение А.4 - Диаграмма последовательностей



Приложение Б

(Обязательное)

Система видеоаналитики автоматического контроля и учета рабочего времени
сотрудников с использованием компьютерного зрения

Листинг программы

					ПГУ1.090301.22.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

Файл docker-compose.yml

```
services:
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

backendcar:
  profiles: ["no start"]
  build: ./backendcar
depends_on:
db:
  condition: service_healthy
  environment:
    -
DATABASE_URL=postgresql://${DB_USER}:${DB_PASSWORD}@db:5432/${DB_NAME}
    - PYTHONUNBUFFERED=1
  volumes:
    - media_volume:/app/media
    - ./logs:/app/logs
env_file:
  - ./backend/.env
deploy:
  resources:
    reservations:
      devices:
        - driver: nvidia
          count: all
          capabilities: [gpu]
  networks:
    - videoanalytics_network
  restart: unless-stopped

yolo:
  profiles: ["no start"]
  build: ./yolo
```

					ПГУ1.090301.22.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

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

yolo2:
    profiles: ["no start"]
    build: ./yolo2
depends_on:
db:
    condition: service_healthy
    volumes:
        - media_volume:/app/media
- ./logs:/app/logs
    - /home/xenusad/VA/debug_faces:/home/admin/debug_faces:rw
env_file:
- ./yolo2/.env
    ports:
        - "5001:5001"
        - "8555:8554"
    deploy:
        resources:
            reservations:
                devices:
                    - driver: nvidia
                      count: all
                      capabilities: [gpu]
        networks:
            - videoanalytics_network
        restart: unless-stopped

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&&
    gunicornconfig.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"
depends_on:
    - django
    restart: unless-stopped
    networks:
        - videoanalytics_network

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

```

```

networks:
videoanalytics_network:
    driver: bridge
    name: videoanalytics_network

```

Файл yolo2\video_stream_server.py

```

import time
import os
import sys
import cv2
import base64
import signal
import eventlet
from eventlet import wsgi
import eventlet.tpool
import socketio
from ultralytics import YOLO
from dotenv import load_dotenv
import logging
from concurrent.futures import ThreadPoolExecutor
import numpy as np
from random import randint
import json
from dataclasses import dataclass, field
import uuid
from random import Random

@dataclass
class Zone:
    id: int
    name: str
    polygon: np.ndarray
    color: tuple = (0, 255, 255)

@dataclass
class Line:
    id: int
    name: str
    points: np.ndarray
    direction: str = 'both'
    color: tuple = (255, 0, 0)

@dataclass
class ZoneState:
    current_track_ids: set = field(default_factory=set)

```

					ПГУ1.090301.22.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```

entered_at: dict = field(default_factory=dict)
last_seen_at: dict = field(default_factory=dict)
confirmed_track_ids: set = field(default_factory=set)

@dataclass
class LineState:
    crossed_track_ids: dict = field(default_factory=dict)

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

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__)

load_dotenv('/home/admin/.env')

DETECTION_TYPES = {
    '3': [0],
}

# Импорт SORT
try:
    from sort import Sort
    SORT_AVAILABLE = True
    logger.info("SORT трекер загружен")
except ImportError:
    SORT_AVAILABLE = False
    logger.error("SORT не найден!")
    sys.exit(1)

# 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 CameraStreamer:
    """Асинхронная обработка камеры"""
    _rand_color_list = None
    def __init__(self, camera_data, model, manager):

```

```

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 '3'

self.manager = manager

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

self.zone_states = {zone.id: ZoneState() for zone in self.zones}
self.line_states = {line.id: LineState() for line in self.lines}

self.presence_timeout = float(os.getenv('ZONE_PRESENCE_TIMEOUT',
'2.0'))
self.line_cross_cooldown = float(os.getenv('LINE_CROSS_COOLDOWN',
'1.5'))
self.zone_confirmation_seconds =
float(os.getenv('ZONE_CONFIRMATION_SECONDS', '10.0'))

self.detect_classes = DETECTION_TYPES.get(self.detection_type, [0])

self.model = model
self.running = True
self.frame_count = 0
self.total_detections = 0
self.real_fps = 0
self.fps_samples = []
self.last_fps_time = time.time()
self.frames_for_fps = 0
self.last_fps_log_time = time.time()

self.track_zone_time = {}
self.track_zone_names = {}
self.track_assigned_name = {}
self.zone_name_threshold = 10.0
self.lost_tracks = {}
self.track_recovery_timeout = 10.0

self.track_last_seen = {}
self.track_disappear_time = {}

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'))

sort_max_age = 60
sort_min_hits = 2

```

```

sort_iou_thresh = 0.15

self.tracker = Sort(
    max_age=sort_max_age,
    min_hits=sort_min_hits,
    iou_threshold=sort_iou_thresh
)

    if CameraStreamer._rand_color_list is None:
rng = Random(42)
CameraStreamer._rand_color_list = [
    (rng.randint(0, 255), rng.randint(0, 255),
rng.randint(0, 255))
    for _ in range(5005)
]
self.rand_color_list = CameraStreamer._rand_color_list

logger.info(f"Кamera {self.camera_id}: {self.camera_name}")
logger.info(f"SORT: max_age={sort_max_age},
min_hits={sort_min_hits}, iou_thresh={sort_iou_thresh}")

    def calculate_fps(self):
self.frames_for_fps += 1
current_time = time.time()
    if current_time - self.last_fps_time>= 1.0:
self.real_fps = self.frames_for_fps
self.fps_samples.append(self.real_fps)
    if len(self.fps_samples) > 10:
self.fps_samples.pop(0)
self.frames_for_fps = 0
self.last_fps_time = current_time

    if current_time - self.last_fps_log_time>= 10:
avg_fps = sum(self.fps_samples) /len(self.fps_samples) if
self.fps_samples else 0
logger.info(f"Кamera {self.camera_id}: FPS: {self.real_fps:.1f},
Средний: {avg_fps:.1f}, Кадров: {self.frame_count}")
self.last_fps_log_time = current_time

    return self.real_fps
    return None

    def point_in_polygon(self, point, polygon):
    result = cv2.pointPolygonTest(polygon, (int(point[0]),
int(point[1])), False)
    return result >= 0

    def _ccw(self, A, B, C):
    return (C[1] - A[1]) * (B[0] - A[0]) > (B[1] - A[1]) *
(C[0] - A[0])

```

```

def segments_intersect(self, A, B, C, D):
    return self._ccw(A, C, D) != self._ccw(B, C, D) and
self._ccw(A, B, C) != self._ccw(A, B, D)

def point_side_of_line(self, point, line_p1, line_p2):
    return ((line_p2[0] - line_p1[0]) * (point[1] - line_p1[1])
-
        (line_p2[1] - line_p1[1]) * (point[0] -
line_p1[0]))

def draw_zones_and_lines(self, frame):
    for zone in self.zones:
        try:
            polygon = np.array(zone.polygon, dtype=np.int32)
            if polygon.ndim != 2 or polygon.shape[1] != 2 or
len(polygon) < 3:
                continue

            cv2.polylines(frame, [polygon], isClosed=True,
color=zone.color, thickness=2)

            x, y = map(int, polygon[0])
            zone_state = self.zone_states.get(zone.id)
            count = len(zone_state.confirmed_track_ids) if
zone_state else 0

            label = f"{zone.name}: {count}"
            (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX,
0.6, 2)
            cv2.rectangle(frame, (x, y - 25), (x + w + 10, y),
zone.color, -1)
            cv2.putText(frame, label, (x + 5, y - 7),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)
        except Exception as e:
            logger.warning(f"Ошибка отрисовки зоны {zone.id}: {e}")

    for line in self.lines:
        try:
            points = np.array(line.points, dtype=np.int32)
            if points.ndim != 2 or points.shape[1] != 2 or
len(points) != 2:
                continue

            p1 = tuple(map(int, points[0]))
            p2 = tuple(map(int, points[1]))
            cv2.line(frame, p1, p2, line.color, 2)

            mx = int((p1[0] + p2[0]) / 2)
            my = int((p1[1] + p2[1]) / 2)
            label = f"{line.name} ({line.direction})"

```

```

        (w, h), _ = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
        cv2.rectangle(frame, (mx, my - 20), (mx + w + 10,
my), line.color, -1)
        cv2.putText(frame, label, (mx + 5, my - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
        except Exception as e:
logger.warning(f"Ошибка отрисовки линии {line.id}: {e}")

    return frame

def update_line_crossings_db(self, tracks):
    now = time.time()

    for track in tracks:
        if not hasattr(track, 'centroidarr') or
len(track.centroidarr) < 2:
            continue

    track_id = int(track.id)
    prev_point = tuple(map(int, track.centroidarr[-2]))
    curr_point = tuple(map(int, track.centroidarr[-1]))

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

        if not self.segments_intersect(prev_point,
curr_point, p1, p2):
            continue

    line_state = self.line_states[line.id]
    last_cross = line_state.crossed_track_ids.get(track_id)

    if last_cross and (now -
last_cross < self.line_cross_cooldown):
        continue

    prev_side = self.point_side_of_line(prev_point, p1, p2)
    curr_side = self.point_side_of_line(curr_point, p1, p2)

    crossing_type = 'cross'
    if prev_side < 0 and curr_side > 0:
        crossing_type = 'in'
    elif prev_side > 0 and curr_side < 0:
        crossing_type = 'out'

    if line.direction != 'both' and crossing_type !=
line.direction:

```

```

        continue

    line_state.crossed_track_ids[track_id] = now

    event_type = 'line_cross'
        if crossing_type == 'in':
    event_type = 'line_cross_in'
    elif crossing_type == 'out':
    event_type = 'line_cross_out'

    self.manager.save_person_event(
        camera_id=self.camera_id,
        line_id=line.id,
        track_id=track_id,
        employee_id=None,
        event_type=event_type,
        point_x=curr_point[0],
        point_y=curr_point[1],
        metadata={'crossingType': crossing_type}
    )

    def process_zone_logic(self, tracked_dets):
        now = time.time()
    current_zone_hits = {zone.id: set() for zone in self.zones}

    active_track_ids = set()

        for det in tracked_dets:
    track_id = int(det[8])
    center_x = int((det[0] + det[2]) / 2)
    center_y = int((det[1] + det[3]) / 2)
    active_track_ids.add(track_id)

            for zone in self.zones:
    is_inside = self.point_in_polygon((center_x, center_y),
    zone.polygon)

                if is_inside:
    current_zone_hits[zone.id].add(track_id)

    self._update_track_zone_display(track_id, zone, is_inside, now)

            for zone in self.zones:
    self._update_zone_db_state(zone, current_zone_hits[zone.id], now)

        return active_track_ids

    def _update_track_zone_display(self, track_id, zone, is_inside,
    now):
        if track_id in self.lost_tracks:

```

```

recovered_name = self.lost_tracks[track_id].get('name')
    if recovered_name and track_id not in
self.track_assigned_name:
self.track_assigned_name[track_id] = recovered_name
self.track_zone_names[track_id] = recovered_name
    del self.lost_tracks[track_id]
    if track_id in self.track_disappear_time:
        del self.track_disappear_time[track_id]

if track_id in self.track_assigned_name:
    return

    if track_id not in self.track_zone_time:
self.track_zone_time[track_id] = {}

    if is_inside:
        if zone.id not in self.track_zone_time[track_id]:
self.track_zone_time[track_id][zone.id] = {
            'enter_time': now,
            'zone_name': zone.name
        }
        else:
time_in_zone = now -
self.track_zone_time[track_id][zone.id]['enter_time']
        if time_in_zone >= self.zone_name_threshold and
track_id not in self.track_assigned_name:
self.track_assigned_name[track_id] = zone.name
self.track_zone_names[track_id] = zone.name
        else:
            if zone.id in self.track_zone_time.get(track_id, {}):
                del self.track_zone_time[track_id][zone.id]

    def _update_zone_db_state(self, zone, new_track_ids, now):
        state = self.zone_states[zone.id]
old_ids= state.current_track_ids
        entered = new_track_ids - old_ids
        missing = old_ids - new_track_ids

        for track_id in entered:
state.entered_at[track_id] = now
state.last_seen_at[track_id] = now
state.current_track_ids.add(track_id)

        for track_id in new_track_ids:
state.last_seen_at[track_id] = now
state.current_track_ids.add(track_id)

        if track_id not in state.confirmed_track_ids:
entered_at = state.entered_at.get(track_id)
            if entered_at and (now - entered_at >=
self.zone_confirmation_seconds):

```

```

state.confirmed_track_ids.add(track_id)
self.manager.save_person_event(
    camera_id=self.camera_id,
    zone_id=zone.id,
    track_id=track_id,
    event_type='zone_enter',
    metadata={'confirmedAfterSec':
self.zone_confirmation_seconds}
)
self.manager.create_zone_session(
    camera_id=self.camera_id,
    zone_id=zone.id,
    track_id=track_id,
    metadata={'confirmedAfterSec':
self.zone_confirmation_seconds}
)

    for track_id in list(missing):
last_seen = state.last_seen_at.get(track_id, now)
    if now - last_seen<self.presence_timeout:
        continue

entered_at = state.entered_at.pop(track_id, None)
state.last_seen_at.pop(track_id, None)
was_confirmed = track_id in state.confirmed_track_ids
state.current_track_ids.discard(track_id)

        if was_confirmed:
state.confirmed_track_ids.discard(track_id)
duration_sec = round(now - entered_at, 2) if entered_at else None
self.manager.save_person_event(
    camera_id=self.camera_id,
    zone_id=zone.id,
    track_id=track_id,
    event_type='zone_exit',
    duration_seconds=duration_sec
)
self.manager.close_zone_session(
    camera_id=self.camera_id,
    zone_id=zone.id,
    track_id=track_id,
    duration_seconds=duration_sec
)

    def cleanup_old_tracks(self, active_track_ids):
current_time = time.time()

all_known_tracks = set(self.track_last_seen.keys())
missing_tracks = all_known_tracks - active_track_ids

        for track_id in missing_tracks:

```

```

        if track_id not in self.track_disappear_time:
self.track_disappear_time[track_id] = current_time

time_missing = current_time - self.track_disappear_time[track_id]

        if time_missing<= self.track_recovery_timeout:
            if track_id not in self.lost_tracks:
self.lost_tracks[track_id] = {
                'last_seen': current_time,
                'name':
self.track_assigned_name.get(track_id),
                'disappear_time':
self.track_disappear_time[track_id]
            }

        for track_id in list(self.track_zone_time.keys()):
            if track_id not in active_track_ids:
disappear_time = self.track_disappear_time.get(track_id, 0)
                if current_time -
disappear_time>self.track_recovery_timeout * 2:
                    del self.track_zone_time[track_id]

        for track_id in list(self.track_assigned_name.keys()):
            if track_id not in active_track_ids:
disappear_time = self.track_disappear_time.get(track_id, 0)
time_gone = current_time - disappear_time

                if time_gone> 30.0:
assigned_name = self.track_assigned_name[track_id]
                    del self.track_assigned_name[track_id]
                    if track_id in self.track_zone_names:
                        del self.track_zone_names[track_id]

        for track_id in list(self.track_last_seen.keys()):
            if track_id not in active_track_ids:
last_seen = self.track_last_seen[track_id]
                if current_time - last_seen> 120.0:
                    if track_id in self.lost_tracks:
                        del self.lost_tracks[track_id]
                    del self.track_last_seen[track_id]
                    if track_id in self.track_disappear_time:
                        del self.track_disappear_time[track_id]

        def draw_boxes(self, img, bbox, identities=None):
            for i, box in enumerate(bbox):
                x1, y1, x2, y2 = [int(i) for i in box]

track_id = int(identities[i]) if identities is not None else 0
                color = self.rand_color_list[track_id %
len(self.rand_color_list)]

```

```

track_name = self.track_zone_names.get(track_id, None)
is_assigned = track_id in self.track_assigned_name

        if track_name:
            if is_assigned:
                label = f"{track_name} #{track_id}"
            else:
                label = f"{track_name} #{track_id}"
        else:
            label = f"Person #{track_id}"

        (w, h), _ = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)

box_color = (0, 215, 255) if is_assigned else color
cv2.rectangle(img, (x1, y1), (x2, y2), box_color, 2)

        cv2.rectangle(img, (x1, y1 - 25), (x1 + w + 10, y1),
box_color, -1)
        cv2.putText(img, label, (x1 + 5, y1 - 8),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, [255, 255, 255], 1)

center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
cv2.circle(img, (center_x, center_y), 3, (0, 0, 255), -
1)

        if not is_assigned and track_id in self.track_zone_time
and self.track_zone_time[track_id]:
current_time = time.time()
        for zone_id, zone_info in
self.track_zone_time[track_id].items():
time_in_zone = current_time - zone_info['enter_time']
            if time_in_zone < self.zone_name_threshold:
                progress = min(time_in_zone /
self.zone_name_threshold, 1.0)
progress_width = int((x2 - x1) * progress)
                cv2.rectangle(img, (x1, y2), (x1 +
progress_width, y2 + 5), (0, 255, 0), -1)
            else:
                cv2.rectangle(img, (x1, y2), (x2, y2 + 5),
(0, 255, 0), -1)

        return img

    def _encode_frame(self, frame):
encode_params = [cv2.IMWRITE_JPEG_QUALITY, self.jpeg_quality]
_, buffer = cv2.imencode('.jpg', frame, encode_params)
return base64.b64encode(buffer).decode('utf-8')

    def process_frame(self, frame):
try:

```

```

        if self.resize_frame:
            frame = cv2.resize(frame, (self.frame_width,
self.frame_height))

        results = self.model(
            frame,
            classes=self.detect_classes,
            conf=0.25,
iou=0.45,
            verbose=False,
            device='cuda'
        )

detection_count = 0
track_ids = []
annotated_frame = frame.copy()
annotated_frame = self.draw_zones_and_lines(annotated_frame)

dets_to_sort = np.empty((0, 6))

        if results and len(results) > 0 and results[0].boxes is
not None:
            boxes = results[0].boxes
            if boxes.xyxy is not None and len(boxes.xyxy) > 0:
                det = boxes.data.cpu().numpy()
detection_count = len(det)

dets_list = []
            for x1, y1, x2, y2, conf, detclass in det:
dets_list.append([x1, y1, x2, y2, conf, detclass])
dets_to_sort = np.array(dets_list) if dets_list else np.empty((0,
6))

tracked_dets = self.tracker.update(dets_to_sort)
tracks = self.tracker.getTrackers()

active_track_ids = self.process_zone_logic(tracked_dets)
self.update_line_crossings_db(tracks)

        for track in tracks:
            if hasattr(track, 'centroidarr') and
len(track.centroidarr) > 1:
                color = self.rand_color_list[track.id %
len(self.rand_color_list)]
                for i in range(len(track.centroidarr) - 1):
                    cv2.line(
annotated_frame,
                        (int(track.centroidarr[i][0]),
int(track.centroidarr[i][1])),
                        (int(track.centroidarr[i+1][0]),
int(track.centroidarr[i+1][1])),

```

```

        color,
        thickness=2
    )

    if len(tracked_dets) > 0:
bbox_xyxy = tracked_dets[:, :4]
        identities = tracked_dets[:, 8].astype(int)

self.draw_boxes(annotated_frame, bbox_xyxy, identities)
track_ids = identities.tolist()
self.total_detections += detection_count

self.cleanup_old_tracks(active_track_ids)

    frame_base64 =
eventlet.tpool.execute(self._encode_frame, annotated_frame)

self.calculate_fps()

track_names = {}
    for track_id in track_ids:
        if track_id in self.track_zone_names:
track_names[str(track_id)] = self.track_zone_names[track_id]

sio.emit(f'camera_{self.camera_id}_frame', {
    'frame': frame_base64,
    'detections': detection_count,
    'total_detections': self.total_detections,
    'track_ids': track_ids,
    'track_names': track_names,
    'timestamp': time.time(),
    'fps': self.real_fps,
    'frame_number': self.frame_count,
    'resolution':
f"{annotated_frame.shape[1]}x{annotated_frame.shape[0]}"
    }, room=f'camera_{self.camera_id}')

    except Exception as e:
logger.error(f"Ошибкаобработкикадрадлякамеры {self.camera_id}:
{e}")

        import traceback
traceback.print_exc()

    def run(self):
        cap = None

gst_pipeline = (
f'rtspsrc location={self.rtsp_url} '
f'latency=100 '
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=2 drop=true'
)

while self.running:
    try:
        if cap is None:
            cap = cv2.VideoCapture(gst_pipeline,
cv2.CAP_GSTREAMER)
            if not cap.isOpened():
                if cap:
cap.release()
                                cap = None
            eventlet.sleep(5)
                                continue

            ret, frame = cap.read()

            if not ret:
cap.release()
                                cap = None
            eventlet.sleep(1)
                                continue

            self.frame_count += 1
            self.process_frame(frame)
            eventlet.sleep(0)

        except Exception as e:
            logger.error(f"Ошибка в циклекамеры {self.camera_id}: {e}")
            if cap:
cap.release()
                                cap = None
            eventlet.sleep(5)

            if cap:
cap.release()

        def stop(self):
            self.running = False

class VideoStreamManager:

    def __init__(self):
        self.cameras = {}

```

```

self.greenlets = {}
self.model = None
self.db_pool = None
self.running = True
self.executor = ThreadPoolExecutor(max_workers=4)

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

self._db_queue = eventlet.Queue()
self._db_greenlet = eventlet.spawn(self._db_worker)

    def init_db_pool(self):
        try:
            import psycopg2.pool

self.db_pool = psycopg2.pool.SimpleConnectionPool(
    1, 10,
    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")

        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 _db_worker(self):
        while self.running:
            try:
                task = self._db_queue.get(timeout=1)
            task()

            except eventlet.queue.Empty:
                pass

    def save_person_event(self, **kwargs):
self._db_queue.put(lambda: self._save_person_event_sync(**kwargs))

    def load_model(self):

```

```

logger.info("Загрузка YOLO модели...")

model_path = '/home/admin/yolov8m.pt'
if not os.path.exists(model_path):
logger.warning(f"Модель не найдена по пути {model_path}, загружаем
из интернета")
model_path = 'yolov8m.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"Всегоклассов: {len(self.model.names)}")

logger.info("Модель загружена")
except Exception as e:
logger.error(f"Ошибказагрузкимодели: {e}")
sys.exit(1)

def load_cameras(self):
conn = None
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 != ''
ANDdetection_type = '3' -- Убедитесь, что
фильтрация происходит на уровне базы данных
ORDER BY id
""")

cameras_data = cursor.fetchall()
cursor.close()

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

logger.info(f"Найденокамер: {len(cameras_data)}")

for row in cameras_data:
camera_id = row[0]
detection_type = str(row[3]) if row[3] is not None else '3'

```

```

        if detection_type != '3':
logger.info(f" Камера {camera_id}: {row[1]} - пропущена
(типдетекции: {detection_type})")
            continue

        zones = self.load_zones_for_camera(camera_id, conn)
        lines = self.load_lines_for_camera(camera_id, conn)

camera_data = {
    'id': camera_id,
    'name': row[1],
    'url': row[2],
    'detection_type': detection_type,
    'coordinates': row[4],
    'is_active': row[5],
    'zones': zones,
    'lines': lines
}

        streamer = CameraStreamer(camera_data, self.model,
self)
self.cameras[camera_id] = streamer
logger.info(f" Камера {camera_id}: {row[1]} | зон: {len(zones)} |
линий: {len(lines)}")

        except Exception as e:
logger.error(f"Ошибказагрузкикамер: {e}")
            import traceback
            traceback.print_exc()
            finally:
                if conn:
self.db_pool.putconn(conn)

        def start_all_cameras(self):
            if not self.cameras:
logger.warning("Неткамердлязапуска")
                return

            for camera_id, streamer in self.cameras.items():
greenlet = eventlet.spawn(streamer.run)
self.greenlets[camera_id] = greenlet
logger.info(f"Запущенакамера {camera_id}")

        def load_zones_for_camera(self, camera_id, conn):
            zones = []
            cursor = conn.cursor()
            cursor.execute("""
                SELECT id, name, polygon, color
                FROM camera_zone
                WHERE camera_id = %s AND is_active = true

```

```

        ORDER BY id
        """, (camera_id,))
        rows = cursor.fetchall()
    cursor.close()

    for row in rows:
        zone_id, zone_name, polygon_json, color = row

    polygon_points = polygon_json if isinstance(polygon_json, list)
    else json.loads(polygon_json)
        polygon = np.array(polygon_points, dtype=np.int32)

    color_bgr = self.parse_color(color, (0, 255, 255))

    zones.append(Zone(
        id=zone_id,
        name=zone_name,
        polygon=polygon,
        color=color_bgr
    ))

    return zones

def load_lines_for_camera(self, camera_id, conn):
    lines = []
    cursor = conn.cursor()
    cursor.execute("""
        SELECT id, name, points, direction, color
        FROM camera_line
        WHERE camera_id = %s AND is_active = true
        ORDER BY id
        """, (camera_id,))
    rows = cursor.fetchall()
    cursor.close()

    for row in rows:
        line_id, line_name, points_json, direction, color = row

    line_points = points_json if isinstance(points_json, list) else
    json.loads(points_json)
        points = np.array(line_points, dtype=np.int32)

        if len(points) != 2:
            continue

    color_bgr = self.parse_color(color, (255, 0, 0))

    lines.append(Line(
        id=line_id,
        name=line_name,
        points=points,

```

```

        direction=direction or 'both',
        color=color_bgr
    ))

    return lines

def parse_color(self, color_value, default):
    if not color_value:
        return default

    try:
        color_value = color_value.strip().lstrip('#')
        if len(color_value) != 6:
            return default

        r = int(color_value[0:2], 16)
        g = int(color_value[2:4], 16)
        b = int(color_value[4:6], 16)
        return (b, g, r)
    except Exception:
        return default

def _save_person_event_sync(self, camera_id, track_id,
event_type, zone_id=None, line_id=None, employee_id=None,
duration_seconds=None, point_x=None, point_y=None, metadata=None):
    conn = None
    try:
        conn = self.db_pool.getconn()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO person_event (
                camera_id, zone_id, line_id, employee_id, track_id,
                event_type, event_time, duration_seconds, point_x, point_y,
                metadata, created_at
            )
            VALUES (%s, %s, %s, %s, %s, %s, NOW(), %s, %s, %s,
%s, NOW())
        """, (
            camera_id, zone_id, line_id, employee_id, track_id,
            event_type, duration_seconds, point_x, point_y,
            json.dumps(metadata) if metadata is not None else None
        ))
        conn.commit()
        cursor.close()
    except Exception as e:
        logger.error(f"Ошибка сохранения события: {e}")
    if conn:
        conn.rollback()
    finally:
        if conn:
            self.db_pool.putconn(conn)

```

```

    def create_zone_session(self, **kwargs):
self._db_queue.put(lambda:
self._create_zone_session_sync(**kwargs))

    def close_zone_session(self, **kwargs):
self._db_queue.put(lambda: self._close_zone_session_sync(**kwargs))

    def _create_zone_session_sync(self, camera_id, zone_id,
track_id, employee_id=None, metadata=None):
    conn = None
    try:
        conn = self.db_pool.getconn()
        cursor = conn.cursor()
cursor.execute("""
                INSERT INTO person_zone_session (
camera_id, zone_id, employee_id, track_id,
entered_at, status, metadata, created_at, updated_at
                )
                VALUES (%s, %s, %s, %s, NOW(), 'open', %s, NOW(),
NOW())
            """, (
camera_id, zone_id, employee_id, track_id,
json.dumps(metadata) if metadata is not None else None
            ))
conn.commit()
cursor.close()
    except Exception as e:
logger.error(f"Ошибка создания сессии: {e}")
if conn:
conn.rollback()
    finally:
        if conn:
self.db_pool.putconn(conn)

    def _close_zone_session_sync(self, camera_id, zone_id,
track_id, duration_seconds):
    conn = None
    try:
        conn = self.db_pool.getconn()
        cursor = conn.cursor()
cursor.execute("""
                UPDATE person_zone_session
                SET exited_at = NOW(),
duration_seconds = %s,
                status = 'closed',
updated_at = NOW()
                WHERE camera_id = %s
                AND zone_id = %s
                AND track_id = %s
                AND status = 'open'
            """)

```

```

        """", (duration_seconds, camera_id, zone_id, track_id))
conn.commit()
cursor.close()
        except Exception as e:
logger.error(f"Ошибка закрытия сессии: {e}")
if conn:
conn.rollback()
        finally:
            if conn:
self.db_pool.putconn(conn)

    def stop_all_cameras(self):
logger.info("Остановка всех камер...")
drain_deadline = time.time() + 10
        while not self._db_queue.empty() and time.time()
<drain_deadline:
eventlet.sleep(0.1)
self.running = False
        try:
self._db_greenlet.wait(timeout=2)
        except:
            pass
        for camera_id, streamer in self.cameras.items():
streamer.running = False

        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")

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

@sio.event
def disconnect(sid):
logger.info(f"Клиент отключился: {sid}")
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):
    camera_id = data.get('cameraId')
    if not camera_id:
        return {'error': 'cameraId required'}

    if not stream_manager or camera_id not in
stream_manager.cameras:
        return {'error': f'Camera {camera_id} not found'}

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

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

    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):
    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} отписалсяоткамеры {camera_id}")

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

@sio.event
def get_cameras_list(sid, data):
    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):
    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
    }

stream_manager = None

def signal_handler(sig, frame):
    logger.info(f"\nПолучен сигнал {sig}")
    if stream_manager:
        stream_manager.stop_all_cameras()
    sys.exit(0)

def main():
    global stream_manager

    logger.info("=" * 50)
    logger.info("yolo2 - Видеостриминг сервер с SORT и логированием в
БД V2.0")
    logger.info("=" * 50)

    signal.signal(signal.SIGINT, signal_handler)
    signal.signal(signal.SIGTERM, signal_handler)

    try:
        stream_manager = VideoStreamManager()
        stream_manager.start_all_cameras()

        logger.info("Запуск Socket.IO сервера на порту 5001")

        sock = eventlet.listen(('0.0.0.0', 5001))

        wsgi.server(sock, app, log_output=True, debug=False, keepalive=65)

```

```

except KeyboardInterrupt:
logger.info("\nПолучен 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 yolo2\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 \
    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 \

```

					ПГУ1.090301.22.001 ПЗ	Лист
						98
Изм.	Лист	№ докум.	Подпись	Дата		

```

    gstreamer1.0-plugins-ugly \
    libgstreamer1.0-dev \
    libgstreamer-plugins-base1.0-dev \
&& rm -rf /var/lib/apt/lists/*

RUN useradd -m -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 \
    ..

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 \
    lap>=0.5.12 \

```

					ПГУ1.090301.22.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпись	Дата		

```
filterpy==1.4.5 \  
scipy>=1.10.0 \  
    scikit-learn>=1.3.0 \  
    scikit-image>=0.20.0 \  
redis==5.0.1
```

```
COPY requirements.txt /home/admin/requirements.txt  
COPY *.pt /home/admin/  
COPY video_stream_server.py /home/admin/video_stream_server.py  
COPY sort.py /home/admin/sort.py
```

```
RUN chown -R admin:admin /home/admin
```

```
USER admin  
WORKDIR /home/admin
```

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

Файл yolo2\videoStream.js

```
class VideoStreamClient {  
  constructor() {  
    this.socket = null;  
    this.cameras = new Map();  
    this.reconnectAttempts = 0;  
    this.maxReconnectAttempts = 5;  
    this.initSocket();  
  }  
  
  initSocket() {  
    this.socket = io({  
      transports: ['websocket'],  
      reconnection: true,  
      reconnectionAttempts: this.maxReconnectAttempts,  
      reconnectionDelay: 1000,  
      reconnectionDelayMax: 5000,  
      timeout: 20000,  
      pingTimeout: 60000,  
      pingInterval: 25000  
    });  
  
    this.setupEventHandlers();  
  }  
  
  setupEventHandlers() {  
    this.socket.on('connect', () => {  
      console.log('Socket.IO подключен');  
      this.reconnectAttempts = 0;  
      this.getCamerasList();  
    });  
  }  
}
```

					ПГУ1.090301.22.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

```

this.socket.on('disconnect', (reason) => {
  console.warn('Socket.IO отключен:', reason);
  this.handleDisconnect(reason);
});

this.socket.on('subscribed', (data) => {
  console.log(`Подписаннакамеру ${data.cameraId}: ${data.message}`);
});

this.socket.on('connect_error', (error) => {
  console.error('Ошибкаподключения:', error);
});

}

handleDisconnect(reason) {
  if (reason === 'io server disconnect') {
    setTimeout(() => {
      this.socket.connect();
    }, 1000);
  }

  this.reconnectAttempts++;
  if (this.reconnectAttempts>= this.maxReconnectAttempts) {
    console.error('Превышено максимальное количество попыток
переподключения');
    this.showError('Потеряно соединение с сервером');
  }
}

subscribeToCamera(cameraId) {
  if (!this.socket || !this.socket.connected) {
    console.warn('Socket неподключен');
    return;
  }

  console.log(`Подписываемся на камеру ${cameraId}`);

  this.socket.emit('subscribe', { cameraId }, (response) => {
    if (response && response.error) {
      console.error('Ошибкаподписки:', response.error);
      this.showError(`Ошибкаподпискинакамеру ${cameraId}:
${response.error}`);
    }
  });
}

this.socket.on(`camera_${cameraId}_frame`, (data) => {
  this.handleVideoFrame(cameraId, data);
});
}

```

```

unsubscribeFromCamera(cameraId) {
  console.log(`Отписываемся от камеры ${cameraId}`);

  this.socket.emit('unsubscribe', { cameraId });
  this.socket.off(`camera_${cameraId}_frame`);
}

handleVideoFrame(cameraId, data) {
  const imgElement =
  document.getElementById(`camera_${cameraId}`);
  if (imgElement) {
    imgElement.src = `data:image/jpeg;base64,${data.frame}`;
  }

  const statsElement =
  document.getElementById(`stats_${cameraId}`);
  if (statsElement) {
    statsElement.innerHTML = `
    <div>Детекций: ${data.detections}</div>
    <div>Всего: ${data.total_detections}</div>
    <div>Кадр: ${data.frame_number}</div>
    `;
  }
}

getCamerasList() {
  this.socket.emit('get_cameras_list', {}, (response) => {
    if (response && response.cameras) {
      console.log('Список камер:', response.cameras);
      this.updateCamerasUI(response.cameras);
    } else if (response && response.error) {
      console.error('Ошибка получения списка камер:', response.error);
    }
  });
}

updateCamerasUI(cameras) {
  const container = document.getElementById('cameras-
  container');
  if (!container) return;

  cameras.forEach(camera => {
    if (!this.cameras.has(camera.id)) {
      this.cameras.set(camera.id, camera);
      this.addCameraToUI(camera);
    }
  });
}

addCameraToUI(camera) {

```

```

        const container = document.getElementById('cameras-
container');
        const cameraDiv = document.createElement('div');
        cameraDiv.className = 'camera-item';
        cameraDiv.id = `camera_${camera.id}_container`;
        cameraDiv.innerHTML = `
<h3>${camera.name}</h3>
<img id="camera_${camera.id}" style="width: 100%; max-width: 640px;
height: auto;" />
<div id="stats_${camera.id}" class="stats"></div>
<button
onclick="videoStream.subscribeToCamera (${camera.id})">Подключиться<
/button>
`;
        container.appendChild(cameraDiv);
    }

    showError(message) {
        const errorDiv = document.createElement('div');
        errorDiv.className = 'error-message';
        errorDiv.style.cssText = `
            position: fixed;
            top: 20px;
            right: 20px;
            background: red;
            color: white;
            padding: 10px;
            border-radius: 5px;
            z-index: 1000;
        `;
        errorDiv.textContent = message;
        document.body.appendChild(errorDiv);

        setTimeout(() => {
            errorDiv.remove();
        }, 5000);
    }
}

const videoStream = new VideoStreamClient();

```

Файл yolo2\requirements.txt

```

ultralalytics==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

```

					ПГУ1.090301.22.001 ПЗ	Лист
						103
Изм.	Лист	№ докум.	Подпись	Дата		

```
python-socketio==5.9.0
eventlet==0.33.3
python-engineio>=4.7.0
python-jose==3.3.0
redis==5.0.1
```

Файл nginx\nginx.conf

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

    ssl_certificate
    /etc/letsencrypt/live/videoanalytics.xenus.dev/fullchain.pem;
    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/office/ {
        proxy_pass http://yolo2:5001;
        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 /socket.io/ {
        proxy_pass http://yolo:5000;
        proxy_http_version 1.1;
```

					ПГУ1.090301.22.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		104

```

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 nginx\Dockfile

```

FROM nginx:alpine

RUN rm /etc/nginx/conf.d/default.conf

COPY nginx.conf /etc/nginx/conf.d/app.conf

RUN mkdir -p /static /media

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```

Файл django\website\views.py

```

from django.http import HttpResponse
from rest_framework.views import APIView
from rest_framework.response import Response

```

					ПГУ1.090301.22.001 ПЗ	Лист
						105
Изм.	Лист	№ докум.	Подпись	Дата		

```

from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from .models import Camera, Models3d, PersonZoneSession,
VehicleEvent, RusGuardEvent, Vehicle, Employee
from .serializers import CameraSerializer, Models3dSerializer,
VehicleEventSerializer, RusGuardEventSerializer
from rest_framework.decorators import api_view
from django.contrib.auth.models import Permission
import os
from django.shortcuts import redirect
from django.views.decorators.csrf import csrf_exempt
from django.http import JsonResponse
from django.contrib.admin.views.decorators import
staff_member_required
from django.conf import settings
from django.utils import timezone
from datetime import datetime
import time
import json
import logging
logger = logging.getLogger(__name__)

def upload_file(request):
    print(" UPLOAD_FILE БЫЗБАНА ")
    print(f"METHOD: {request.method}")
    print(f"POST keys: {list(request.POST.keys())}")
    print(f"FILES keys: {list(request.FILES.keys())}")

    if request.method == 'POST':
        folder_name = request.POST.get('folder_name')
        uploaded_file = request.FILES.get('file')

        print(f"folder_name = {folder_name}")
        print(f"uploaded_file = {uploaded_file}")
        if uploaded_file:
            print(f"Сохраняем файл: {uploaded_file.name} в папку
{folder_name}")

        folder_path = os.path.join(settings.MEDIA_ROOT, folder_name)

        if not os.path.exists(folder_path):
            os.makedirs(folder_path, exist_ok=True)

        file_path = os.path.join(folder_path, uploaded_file.name)

        with open(file_path, 'wb+') as destination:
            for chunk in uploaded_file.chunks():
                destination.write(chunk)

        print(f" Файлсохранен: {file_path}")

```

```

        return redirect(request.META.get('HTTP_REFERER', '/'))

@csrf_exempt
def delete_file(request):
    if request.method == 'POST':
        folder_name = request.POST.get('folder_name')
        filename = request.POST.get('filename')
        file_path = os.path.join(settings.MEDIA_ROOT, folder_name,
        filename)

        try:
            if os.path.exists(file_path):
                os.remove(file_path)
                return JsonResponse({'success': True})
            return JsonResponse({'success': False, 'error': 'File
not found'})
        except Exception as e:
            return JsonResponse({'success': False, 'error':
str(e)})

        return JsonResponse({'success': False, 'error': 'Invalid
method'})

def GateOpenfun():
    print("Gate opening...")
    return

class ProfileView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        return Response({
            'username': request.user.username,
            'email': request.user.email,
            'id': request.user.id
        })

class UserPermissionsView(APIView):
    def get(self, request):
        user = request.user

        user_groups = user.groups.all()

        permissions = set()

        for group in user_groups:
            group_permissions = group.permissions.all()
            for perm in group_permissions:

```

```

permissions.add(perm.codename)
    return Response({'permissions': list(permissions)})

class CameraListView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        cameras = Camera.objects.filter(is_active=True)

        serializer = CameraSerializer(cameras, many=True)
        return Response(serializer.data)

class CameraDetailView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, pk):
        try:
            camera = Camera.objects.get(pk=pk, is_active=True)

            serializer = CameraSerializer(camera)

            return Response(serializer.data)
        except Camera.DoesNotExist:
            return Response({'error': 'Камера не найдена'},
status=404)

class Models3dViewSet(viewsets.ModelViewSet):
    queryset = Models3d.objects.all()
    serializer_class = Models3dSerializer

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()

        # Пагинация
        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
        })

class EmployeeWorkTimeView(APIView):
    def get(self, request, camera_id, date):
        try:
            date_obj = datetime.strptime(date, "%Y-%m-%d")
        except ValueError:
            return Response({'error': 'Некорректный формат даты.
Используйте формат YYYY-MM-DD'}, status=400)

        sessions = PersonZoneSession.objects.filter(
            camera_id=camera_id,
            entered_at__date=date_obj.date(),
            status="closed"
        )

        zones_summary = {}

        for session in sessions:
            zone_id = session.zone.id
            zone_name = session.zone.name
            duration = session.duration_seconds or 0

            if zone_id not in zones_summary:
                zones_summary[zone_id] = {
                    'zone_name': zone_name,
                    'total_duration_seconds': 0
                }

            zones_summary[zone_id]['total_duration_seconds'] += duration

        formatted_zones_data = []
        for zone_id, data in zones_summary.items():
            total_seconds = data['total_duration_seconds']
            total_minutes = total_seconds / 60
            formatted_zones_data.append({
                'zone_id': zone_id,
                'zone_name': data['zone_name'],

```

```

        'total_duration_seconds': total_seconds,
        'total_duration_minutes': total_minutes,
        'total_duration_formatted':
str(time.strftime("%H:%M:%S", time.gmtime(total_seconds)))
    })

    return Response({
        'camera_id': camera_id,
        'date': date,
        'zones': formatted_zones_data
    })

@api_view(['GET'])
def Models3dByTypeView(request, type_name):
    models = Models3d.objects.filter(type=type_name)
    serializer = Models3dSerializer(models, many=True)
    return Response({
        'type': type_name,
        'count': models.count(),
        'results': serializer.data
    })

@api_view(['GET'])
def Models3dByNumberView(request, type_number):
    models = Models3d.objects.filter(number=type_number)
    serializer = Models3dSerializer(models, many=True)
    return Response({
        'type': type_number,
        'count': models.count(),
        'results': serializer.data
    })

@api_view(['POST'])
def GateOpenView(request):
    GateOpenfun()
    return Response({
        'status': 'open'
    })

@api_view(['GET'])
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:
    returnResponse(
        {'error': f'Транспортное средство с номером {number} не
найдено'},
        status=404
    )

@api_view(['GET'])
def get_all_3d_models(request):
    """Получение всех 3D моделей с группировкой по типу"""
    models = Models3d.objects.all()

    result = {
        'car': [],
        'people': [],
        'street_label': [],
        'room_label': [],
        'territory': [],
        'other': []
    }

    for model in models:
        type_key = model.type.replace(' ', '_')
        if type_key in result:
            result[type_key].append({
                'id': model.id,
                'path': model.path,
                'number': model.number
            })
        else:
            result['other'].append({
                'id': model.id,
                'path': model.path,
                'number': model.number
            })

    return Response(result)

@csrf_exempt
@api_view(['POST'])
def rusguard_event(request):
    try:
        data = request.data
        raw_type = data.get('event_type')
        if raw_type == 1 or raw_type == '1':
            event_type = 'enter'
        elif raw_type == 2 or raw_type == '2':
            event_type = 'exit'
        else:
            message = data.get('Message', '')

```

```

sub_type = data.get('MessageSubType', '')
    if 'Вход' in message or 'Entry' in sub_type:
event_type = 'enter'
elif 'Выход' in message or 'Exit' in sub_type:
event_type = 'exit'
    else:
event_type = 'enter'
employee_name = data.get('EmployeeFullName', '')
    employee = None
    if employee_name:
name_parts = employee_name.split()
    if len(name_parts) >= 2:
        surname = name_parts[0]
        name = name_parts[1]
        try:
            employee = Employee.objects.filter(
                surname__iexact=surname,
                name__iexact=name
            ).first()
        except:
            pass
    event = RusGuardEvent.objects.create(
        employee=employee,
employee_name=employee_name,
event_type=event_type,
event_time=timezone.now(),
        details=data.get('Details', '')
    )

    return Response({
        'status': 'ok',
        'event_id': event.id,
        'event_type': event_type,
        'employee_found': employee is not None
    })

except Exception as e:
    return Response({'status': 'error', 'message': str(e)},
status=500)

class TransportLogsSearchView(APIView):
permission_classes = [IsAuthenticated]

    def get(self, request, camera_id):
        number = request.query_params.get('number',
        '').strip().upper()
        owner = request.query_params.get('owner', '').strip()
date_from = request.query_params.get('date_from', '')
date_to = request.query_params.get('date_to', '')
        ordering = request.query_params.get('ordering', '-
arrival_time')

```

```

        page = int(request.query_params.get('page', 1))
        per_page = int(request.query_params.get('per_page', 10))

        queryset = VehicleEvent.objects.select_related(
            'vehicle',
            'employee'
        )

        if number:
            queryset = queryset.filter(number__icontains=number)

        if owner:
            owner_parts = owner.split()

            from django.db.models import Q

            if len(owner_parts) == 1:
                queryset = queryset.filter(
                    Q(employee__surname__icontains=owner_parts[0])
                    |
                    Q(employee__name__icontains=owner_parts[0])
                )
            elif len(owner_parts) >= 2:
                queryset = queryset.filter(
                    Q(employee__surname__icontains=owner_parts[0])
                    &
                    Q(employee__name__icontains=owner_parts[1])
                )

            if date_from:
                try:
                    from_date = datetime.strptime(date_from, '%Y-%m-%d')
                except ValueError:
                    pass

            if date_to:
                try:
                    to_date = datetime.strptime(date_to, '%Y-%m-%d')
                except ValueError:
                    pass

        allowed_ordering = [
            'number', '-number',
            'employee__surname', '-employee__surname',
            'arrival_time', '-arrival_time',
            'departure_time', '-departure_time',

```

```

    ]

    if ordering in allowed_ordering:
        queryset = queryset.order_by(ordering)
    else:
        queryset = queryset.order_by('-arrival_time')

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

    if page < 1:
        page = 1
    if page > total_pages:
        page = total_pages

    start = (page - 1) * per_page
    end = start + per_page
    page_queryset = queryset[start:end]

    serializer = VehicleEventSerializer(page_queryset,
many=True)

    return Response({
        'search_query': number,
        'owner_query': owner,
        'date_from': date_from,
        'date_to': date_to,
        'ordering': ordering,
        'count': total_count,
        'page': page,
        'per_page': per_page,
        'total_pages': total_pages,
        'results': serializer.data
    })
class RusGuardEventsView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        events = RusGuardEvent.objects.all().order_by('-
event_time')[:50]
        serializer = RusGuardEventSerializer(events, many=True)
        return Response(serializer.data)

    def post(self, request):
        try:
            data = request.data

            event = RusGuardEvent.objects.create(
event_type=data.get('event_type', 'manual'),
event_time=data.get('event_time') or timezone.now(),
card_number=data.get('card_number', ''),

```

```

employee_name=data.get('employee_name', ''),
door_name=data.get('door_name', ''),
direction=data.get('direction', ''),
details=data.get('details', ''),
raw_data=data
    )

    return Response({
        'status': 'success',
        'event_id': event.id,
        'event_type': event.event_type
    }, status=201)

except Exception as e:
    return Response({'status': 'error', 'message': str(e)},
status=400)

class RecentTransportLogsView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, camera_id):
        from django.db import connection

        with connection.cursor() as cursor:
            cursor.execute("""
                SELECT id FROM vehicle_event
                ORDER BY GREATEST(
                    COALESCE(departure_time, arrival_time),
                    COALESCE(arrival_time, departure_time)
                ) DESC
                LIMIT 9
            """)
            ids = [row[0] for row in cursor.fetchall()]

        queryset = VehicleEvent.objects.filter(id__in=ids)
        id_order = {id: i for i, id in enumerate(ids)}
        logs = sorted(queryset, key=lambda x: id_order[x.id])

        serializer = VehicleEventSerializer(logs, many=True)
        return Response(serializer.data)

class RusGuardEventViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = RusGuardEvent.objects.all().order_by('-event_time')
    serializer_class = RusGuardEventSerializer

class RusGuardEventsPaginatedView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        page = int(request.query_params.get('page', 1))
        per_page = int(request.query_params.get('per_page', 10))

```

```

queryset = RusGuardEvent.objects.all().order_by('-event_time')

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

    if page < 1:
        page = 1
    if page > total_pages:
        page = total_pages

    start = (page - 1) * per_page
    end = start + per_page

    serializer = RusGuardEventSerializer(queryset[start:end],
many=True)

    return Response({
        'count': total_count,
        'page': page,
        'per_page': per_page,
        'total_pages': total_pages,
        'results': serializer.data
    })

class RusGuardEventsSearchView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        employee_name = request.query_params.get('employee_name',
        '').strip()
        event_type = request.query_params.get('event_type', '')
        date_from = request.query_params.get('date_from', '')
        date_to = request.query_params.get('date_to', '')
        ordering = request.query_params.get('ordering', '-
event_time')

        page = int(request.query_params.get('page', 1))
        per_page = int(request.query_params.get('per_page', 10))

        queryset = RusGuardEvent.objects.all()

        if employee_name:
            queryset = queryset.filter(employee_name__icontains=employee_name)

        if event_type and event_type != 'all':
            queryset = queryset.filter(event_type=event_type)

        if date_from:
            try:
                from_date = datetime.strptime(date_from, '%Y-%m-%d')

```

```

queryset = queryset.filter(event_time__date__gte=from_date.date())
    except ValueError:
        pass

    if date_to:
        try:
to_date = datetime.strptime(date_to, '%Y-%m-%d')
queryset = queryset.filter(event_time__date__lte=to_date.date())
    except ValueError:
        pass

allowed_ordering = [
    'employee_name', '-employee_name',
    'event_type', '-event_type',
    'event_time', '-event_time',
]

    if ordering in allowed_ordering:
queryset = queryset.order_by(ordering)
    else:
queryset = queryset.order_by('-event_time')

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

    if page < 1:
        page = 1
    if page > total_pages:
        page = total_pages

    start = (page - 1) * per_page
    end = start + per_page
page_queryset = queryset[start:end]

    serializer = RusGuardEventSerializer(page_queryset,
many=True)

    return Response({
        'employee_query': employee_name,
        'event_type': event_type,
        'date_from': date_from,
        'date_to': date_to,
        'ordering': ordering,
        'count': total_count,
        'page': page,
        'per_page': per_page,
        'total_pages': total_pages,
        'results': serializer.data
    })

@api_view(['GET'])

```

```

def UniversalSearchView(request):
    """
    Универсальный поиск по номеру или ФИО
    Пример: ?q=Иванов или ?q=A123BC
    """
    query = request.GET.get('q', '').strip()

    if not query:
        return Response({'results': [], 'count': 0})

    is_number = any(c.isdigit() for c in query)

    if is_number:
        vehicles =
Vehicle.objects.select_related('employee').filter(
    number__icontains=query.upper()
)[:10]
    else:
        name_parts = query.split()
        from django.db.models import Q

        if len(name_parts) == 1:
            vehicles =
Vehicle.objects.select_related('employee').filter(
    Q(employee__surname__icontains=name_parts[0]) |
    Q(employee__name__icontains=name_parts[0])
)[:10]
        else:
            vehicles =
Vehicle.objects.select_related('employee').filter(
    Q(employee__surname__icontains=name_parts[0]) &
    Q(employee__name__icontains=name_parts[1])
)[:10]

    owners = {}
    for vehicle in vehicles:
        emp = vehicle.employee
        if emp:
            key = f"{emp.surname} {emp.name}"
            if key not in owners:
                owners[key] = {
                    'employee': {
                        'id': emp.id,
                        'surname': emp.surname,
                        'name': emp.name,
                        'company': emp.company,
                    },
                    'vehicles': []
                }
            owners[key]['vehicles'].append({
                'id': vehicle.id,

```

```

        'number': vehicle.number,
        'brand': vehicle.brand,
    })

    results = list(owners.values())

    return Response({
        'query': query,
        'count': len(results),
        'results': results
    })

```

Файл django\website\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')

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/', include(router.urls)),
path('api/user/permissions/', UserPermissionsView.as_view(),
name='user_permissions'),
]

```

```

if settings.DEBUG:
urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)

```

Файл django\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 django\website\permissions.py

```

from rest_framework import permissions

class CameraAccessPermission(permissions.BasePermission):

    def has_permission(self, request, view):
        return request.user.is_authenticated

    def has_object_permission(self, request, view, obj):
        return obj.user_has_access(request.user)

```

Файл django\website\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'
        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'
        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('Типмодели',max_length=500,choices=TYPE_CHOICES,
default='other', blank=True )
    number = models.IntegerField('Номермодели', null=True,
blank=True)
    class Meta:
verbose_name = 'Загрузкамоделей'
verbose_name_plural = 'Загрузкамоделей'

    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 = 'Транспортноесредство'
verbose_name_plural = 'Транспортныесредства'

```

```

        ordering = ['number']
        managed = True

    def save(self, *args, **kwargs):
# Конвертация русских букв
ru_to_lat = {
            'A': 'A', 'B': 'B', 'E': 'E', 'K': 'K', 'M': 'M',
            'H': 'H', 'O': 'O', 'P': 'P', 'C': 'C', 'T': 'T',
            'Y': 'Y', 'X': '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 = 'Зоныкамер'
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='Типсобытия'
)

event_time = models.DateTimeField(verbose_name='Времясобытия')

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 = 'Событиечеловека'
verbose_name_plural = 'Событиялюдей'
ordering = ['-event_time']
managed = True

def __str__(self):
target = self.zone.name if self.zone else self.line.name if
self.line else 'Безобъекта'
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='Времявхода')

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

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='')
created_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)

class Meta:
db_table = 'website_faceembedding'

    def __str__(self):

```

```

        return self.person_name

class FaceRecognitionLog(models.Model):
    camera_id = models.IntegerField()
    frame_id = models.CharField(max_length=100)
    person_id = models.CharField(max_length=100, null=True)
    person_name = models.CharField(max_length=200, null=True)
    confidence = models.FloatField()
    embedding = JSONField(null=True)
    face_image = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

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

class RusGuardEvent(models.Model):
    """
    Событияот СКУД RusGuard - тольковажное
    """
    EVENT_TYPE_CHOICES = [
        ('enter', 'Вход'),
        ('exit', 'Выход'),
    ]

    employee = models.ForeignKey(
        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='Типсобытия')
    event_time = models.DateTimeField(verbose_name='Времясобытия')
    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 django\website\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',)

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

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

        if not user.is_superuser and not user.is_staff:
            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
    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]

    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']

    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):
        ru_to_lat = {
            'А': 'A', 'Б': 'B', 'В': 'V', 'Г': 'G', 'Д': 'D', 'Е': 'E', 'Ж': 'Z', 'З': 'S', 'И': 'I', 'Й': 'Y', 'К': 'K', 'Л': 'L', 'М': 'M', 'Н': 'H', 'О': 'O', 'П': 'P', 'Р': 'R', 'С': 'C', 'Т': 'T', 'У': 'U', 'Ф': 'F', 'Х': 'X', 'Ц': 'Z', 'Ч': 'S', 'Ш': 'S', 'Щ': 'S', 'а': 'A', 'б': 'B', 'в': 'V', 'г': 'G', 'д': 'D', 'е': 'E', 'ж': 'Z', 'з': 'S', 'и': 'I', 'й': 'Y', 'к': 'K', 'л': 'L', 'м': 'M', 'н': 'H', 'о': 'O', 'п': 'P', 'р': 'R', 'с': 'C', 'т': 'T', 'у': 'U', 'ф': 'F', 'х': 'X', 'ц': 'Z', 'ч': 'S', 'ш': 'S', 'щ': 'S',
        }

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

        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):
    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):
    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': 'Эмбеддинг 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('<imgsrc="{}" 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 "✗ Ошибка формата"
    return "-"
embedding_info.short_description = "Эмбеддинг"

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 django\Dockefile

```
FROM python:3.11

WORKDIR /app

RUN apt-get update && apt-get install -y \
gcc \
postgresl-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"]
```

Файл django\requirements.txt

```
Django==5.2.7
django-restframework==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
django-restframework-simplejwt>=5.3.0
```

Приложение В

(обязательное)

Система видеоаналитики автоматического контроля и учета рабочего времени
сотрудников с использованием компьютерного зрения

Презентация

					ПГУ1.090301.22.001 ПЗ	Лист
						141
Изм.	Лист	№ докум.	Подпись	Дата		

Система видеоаналитики автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения

Выполнил: студент группы
22ВВП1
Рябов Святослав Игоревич
Руководитель: к.т.н, доцент
кафедры "Вычислительная
техника"
Митрохина Наталья Юрьевна

Пенза 2026

Актуальность

- повсеместное использование видеонаблюдения на предприятиях .
- высокая стоимость коммерческих систем видеоаналитики.
- доступность открытых технологий компьютерного зрения.
- неэффективность ручных методов учета рабочего времени.

2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.22.001 ПЗ

Лист

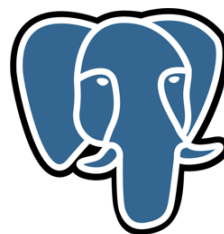
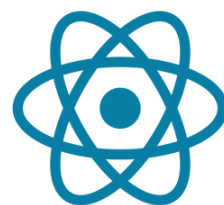
142

Постановка задачи

- обнаружение людей в видеопотоке, и их последующее отслеживание и идентификация.
- интеграция с системой контроля и управления доступом для сверки данных.
- формирование информации по времени нахождения сотрудников в заданных зонах
- отображение обработанных видеопотоков и аналитических данных через веб-интерфейс

3

Используемые технологии



docker®

django

4

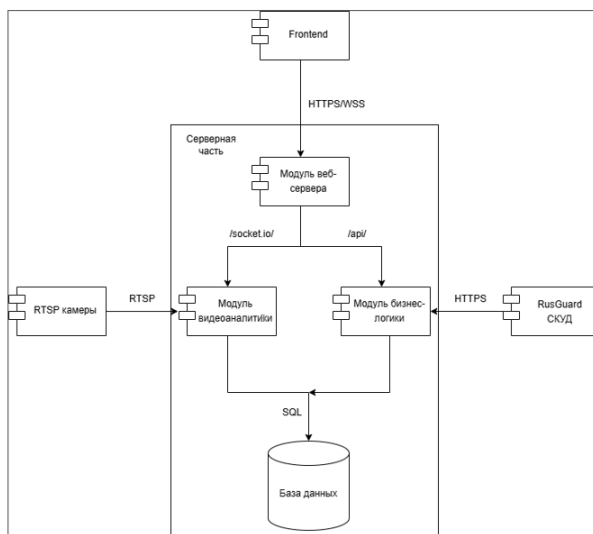
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.22.001 ПЗ

Лист

143

Архитектура модулей



5

Диаграмма вариантов использования



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.22.001 ПЗ

Лист

144

[Главная страница с выбором камер](#)



Страница видеоаналитики



Изм.	Лист	№ докум.	Подпись	Дата

Логирование СКУД

Логи сотрудников

Поиск по сотруднику: Все события

От: До:

№	Сотрудник	Тип события	Время	Детали
1	Охрана	Выход	27.05.2026, 20:40:48	По ключу Номер ключа 1600738823 (0x00005F695607)
2	Охрана	Вход	27.05.2026, 20:38:15	По ключу Номер ключа 1600738823 (0x00005F695607)
3	Каюкова Анастасия Александровна @КаюковА	Выход	27.05.2026, 18:40:44	По ключу Номер ключа 1585994151 (0x00005E8859A7)
4	Фиомина Галина Николаевна	Выход	27.05.2026, 17:27:32	По ключу Номер ключа 2662390736 (0x00009EB0DBD0)
5	Попыгина Ярослава	Выход	27.05.2026, 17:23:41	По ключу Номер ключа 2654449520 (0x00009E37AF70)
6	Елистратов Даниил Дмитриевич	Выход	27.05.2026, 17:20:09	По ключу Номер ключа 2655667664 (0x00009E4A45D0)
7	Елистратов Даниил Дмитриевич	Вход	27.05.2026, 17:19:47	По ключу Номер ключа 2655667664 (0x00009E4A45D0)

1 из 331 (2315 записей)

9

Административная панель

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, BOOT, ОТКРЫТЬ САРТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

Администрирование сайта

WEBSITE

- Faca embeddings
- Загрузка моделей
- Зоны камер
- Камеры
- Линии камер
- Сотрудники
- Транспортные средства

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

- Группы
- Пользователи

Последние действия

Мои действия

- <https://videanalytics.xenus.dev/m...>
- [Administrator](#)
- [Administrator](#)
- <https://videanalytics.xenus.dev/m...>
- <https://videanalytics.xenus.dev/m...>
-
-
-
-
-

10

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.22.001 ПЗ

Лист

146

Перспективы развития

- внедрение распознавания лиц для распознавания сотрудников.
- масштабирование системы на большее количество камер.
- автоматическое формирование отчетов.
- неэффективность ручных методов учета рабочего времени.

11

11

Заключение

В ходе дипломной работы была разработана система автоматического контроля и учета рабочего времени сотрудников с использованием компьютерного зрения. Решены следующие задачи: проанализирована предметная область и существующие аналоги, спроектирована модульная архитектура, реализован модуль видеоаналитики с детекцией и трекингом людей, разработан модуль бизнес-логики с REST API, создан клиентский веб-интерфейс на React, настроена интеграция с СКУД. Все компоненты завернуты в Docker-контейнеры. Проведено тестирование.

12

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.22.001 ПЗ

Лист

147

Спасибо за внимание!

13

					ПГУ1.090301.22.001 ПЗ	<i>Лист</i>
						148
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		