4. МОДЕЛИРОВАНИЕ И ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Проектирование информационной системы мониторинга безопасности периметра базируется на функциональных и нефункциональных требованиях, сформулированных в разделе 2. Основной задачей данного этапа является конкретизация архитектурных, технологических и структурных решений, необходимых для построения работоспособной, устойчивой и масштабируемой системы, соответствующей целевым сценариям применения.

Выбор технологического стека обусловлен спецификой задач, поставленных перед системой, и особенностями её целевой среды эксплуатации. Основной акцент сделан на локальной работе, устойчивости к сбоям, использовании алгоритмов компьютерного зрения и временного прогнозирования, а также на минимальной зависимости от внешних облачных сервисов.

4.1. Аппаратно-ОС-платформа

Система развёртывается в виде настольного программного обеспечения на рабочих станциях под управлением ОС семейства Windows и Linux. При этом приоритет отдается ОС на базе ядра Linux, в связи с более гибкой и детализированной настройкой безопасности [8]. В качестве ОС на базе ядра Linux рекомендуются дистрибутивы: Debian, Ubuntu, Fedora, Astra Linux и Arch Linux. В качестве аппаратной базы используются стандартные промышленные компьютеры и серверы с поддержкой многопоточности и аппаратного ускорения графики.

Минимальные требования к аппаратной составляющей представлены в таблице 4.1.1.

Таблица 4.1.1 – Минимальные требования к аппаратной составляющей (составлено автором)

Сервер обработки данных	CPU	Intel Core i5 / AMD Ryzen 5 (4–6 ядер, ≥2.5 ГГц)
	GPU	NVIDIA GTX 1650 / RTX 2060 (4–6 ГБ VRAM) для запуска ML-моделей
	RAM	16 ГБ DDR4
	HDD	512 ГБ SSD + 1 ТБ HDD для хранения данных
Рабочая станция пользователя	CPU	Intel i5 / AMD Ryzen 5 (4–6 ядер, ≥2.5 ГГц)
	RAM	16 ГБ
	Монитор	Full HD (1920×1080), опционально с поддержкой сенсорного ввода

Взаимодействие с БПЛА осуществляется через интерфейс наземной станции управления и каналы телеметрии. Бортовая платформа дронов использует прошивку PX4 с открытым API. Регламентация вектора информационного взаимодействия, между системой и БПЛА, осуществляется по техническим протоколам MAVLink [9].

4.2. Языки и библиотеки

Разработка системы осуществляется на языке C++ с применением библиотеки Qt версии 6+ для построения графического интерфейса и межмодульного взаимодействия [10]. Для работы с изображениями и видеопотоком используется OpenCV, эффективная библиотека для обработки видео и изображений в реальном времени. Создание и обучение нейросетевого модуля реализуется на Python с использованием библиотек TensorFlow и PyTorch [11]. Обмен данными между компонентами, написанными на различных языках, реализуется через сериализованные структуры и локальные каналы взаимодействия. Взаимодействие между компонентами системы и нейросетевыми модулями, конвертированными в формат ONNX, производится с помощью библиотеки ONNX Runtime [12]. Такая связка обеспечивает высокую производительность в критичных участках и гибкость при работе с нейросетевыми моделями.

4.3. СУБД и формат данных

Для хранения событий, телеметрии, маршрутов и аналитических результатов используется встроенная реляционная СУБД (SQLite), обеспечивающая достаточную производительность при локальном использовании и простоту интеграции с приложением [13]. При увеличении объемов данных, возможен переход на более вместительную, но более “тяжелую” реляционную СУБД (PostgreSQL). Для временного обмена между модулями применяются сериализованные JSON-объекты, а также XML и бинарные форматы для исторических выборок и логирования.

Диаграмма принятия решений представлена на рисунке 4.3.1.

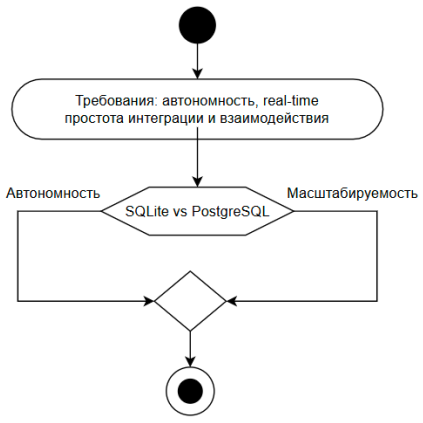


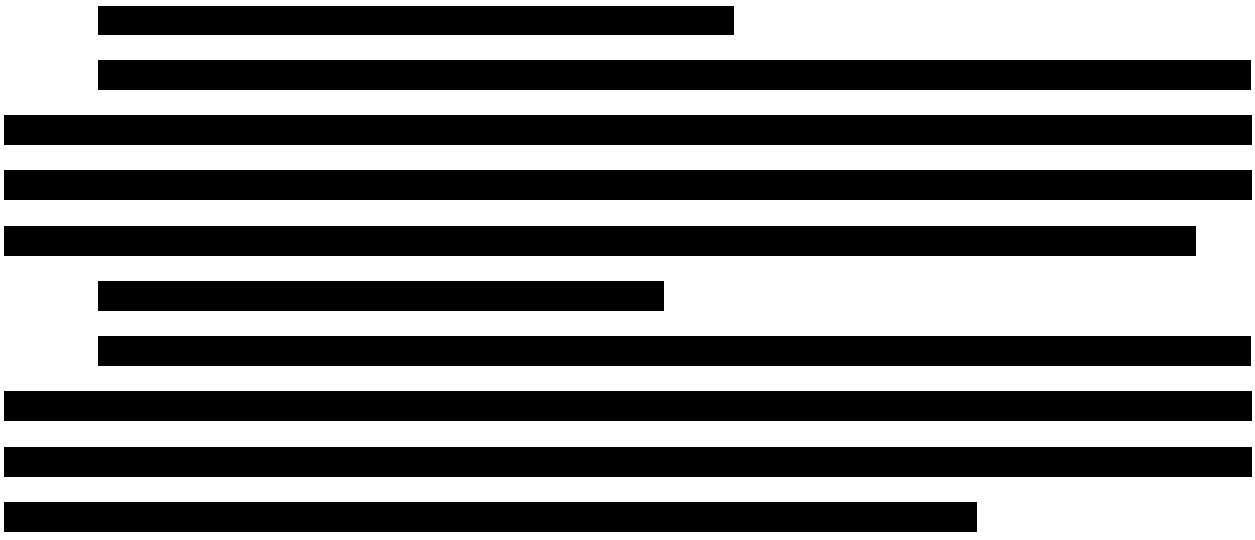
Рисунок 4.3.1 – Диаграмма принятия решений (разработано автором)

4.4. ML-модели

В архитектуре системы предполагается использование двух типов моделей машинного обучения: модели для пространственной детекции объектов YOLO и модели для анализа временных рядов (например, архитектуры типа LSTM). Их задача заключается соответственно в обнаружении угроз на видеопотоке и прогнозировании рисков по накопленным телеметрическим данным. Модели конвертируются и интегрируются в систему в формате ONNX, в связи с универсальностью и независимостью от фреймворка, взаимодействие осуществляется посредством межпроцессорного обмена данными с системой.

4.5. Инструменты тестирования

Для отладки и тестирования системы используются встроенные модули логирования, мониторинга и визуализации состояния компонентов. Функциональное тестирование выполняется через имитацию пользовательских сценариев, тестирование ML-моделей — через метрики precision, recall, AUC, визуальный анализ. Также применяются юнит-тесты для отдельных классов C++ и нагрузочные тесты для оценки поведения системы при высоком объёме видеопотока и событий. Для финального тестирования перед физическим развертыванием системы используется Gazebo Simulator в симбиозе с PX4-AutoPilot, позволяющие эмулировать работу системы с гибкой настройкой окружения, близкого к реальным условиям [14].



5. ВЫБОР СРЕДСТВ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ, РАЗРАБОТКИ, ТЕСТИРОВАНИЯ И СОПРОВОЖДЕНИЯ, ФОРМАТЫ ДАННЫХ

Для успешной реализации проекта необходимы инструменты, охватывающие все этапы жизненного цикла программного обеспечения — от проектирования до сопровождения. В данном разделе описываются инструменты, использованные при создании системы, их функционал и обоснование выбора.

5.1.Средства автоматизированного проектирования

На этапе проектирования системы был использован следующий инструмент:

— CA Erwin Process Modeler

Инструмент для проектирования процессов в нотации IDEF0. Использовался для моделирования бизнес-логики и структуры данных [15].

— Draw.io

Бесплатный онлайн-сервис для создания диаграмм (классов, потоков данных, архитектуры системы). Выбор обусловлен простотой интерфейса, поддержкой разнообразных диаграмм и возможностью совместной работы.

5.2. Средства разработки

Разработка системы велась с использованием следующих инструментов:

— VS Code

Кроссплатформенный редактор с поддержкой Python, интеграцией с системами контроля версий и встроенным терминалом.

— Qt Creator

Среда для разработки графического интерфейса на C++ с использованием библиотеки Qt, обеспечивающая быструю реализацию пользовательских сценариев.

— DB Browser for SQLite

Графический инструмент для работы с базами данных SQLite. Использовался для создания, редактирования таблиц, выполнения SQL-запросов и проверки целостности данных.

5.3. Выбор СУБД и форматов данных

В качестве основной системы хранения используется реляционная СУБД SQLite, интегрированная непосредственно в приложение. Выбор данной базы данных обусловлен следующими факторами:

— Локальность и автономность — отсутствие зависимости от внешних серверов, что критично для систем безопасности.

— Производительность: эффективная работа с данными даже на ограниченных аппаратных ресурсах.

— Простота интеграции: встраиваемость в приложение без необходимости дополнительной настройки серверов.

— Поддержка ACID-транзакций: гарантия целостности данных при записи событий, телеметрии и результатов анализа [16].

Для временного обмена данными между модулями (например, между компонентами на C++ и Python) применяются:

— Сериализованные JSON-объекты: обеспечивают гибкость и читаемость при передаче параметров конфигурации и метаданных.

— XML-файлы: используются для логирования исторических выборок и экспорта данных.

— Бинарные форматы: применяются для хранения видеопотоков и больших объёмов телеметрии с минимальными накладными расходами.

При масштабировании системы для работы с крупными массивами данных предусмотрен переход на PostgreSQL, что позволит сохранить реляционную модель, но повысить производительность за счёт распределённой архитектуры.

5.4. Конфигурация ML-моделей и интеграция

Модели машинного обучения (YOLO для детекции объектов, LSTM для анализа временных рядов) хранятся и используются в формате ONNX. Это обеспечивает:

— Кросс-фреймворковую совместимость: независимость от библиотек (TensorFlow, PyTorch) на этапе эксплуатации.

— Оптимизацию производительности: ускорение инференса за счёт универсальной среды выполнения.

— Лёгкость обновления моделей: замена версий без изменения кода системы.

Взаимодействие между модулями (например, передача результатов детекции из Python-модели в C++-ядро) реализовано через межпроцессорный обмен данными с использованием сериализованных структур и локальных сокетов.

Преимущества выбранного подхода :

1. Устойчивость к сбоям: локальная СУБД и бинарные форматы минимизируют риски потери данных при разрывах связи.

2. Гибкость: поддержка JSON и XML позволяет адаптировать систему под сторонние инструменты анализа.

3. Масштабируемость: переход на PostgreSQL и модульность форматов данных обеспечивают рост системы без перепроектирования.

4. Совместимость с ОС: SQLite и ONNX поддерживаются как в Windows, так и в Linux, что соответствует требованиям к кросс-платформенности.

5.5. Средства автоматизированного тестирования

Для тестирования системы использовались:

— Отладчик Qt Creator

Инструмент для пошаговой отладки графического интерфейса и модулей на C++.

— Встроенные тесты SQLite

Механизмы проверки целостности данных, валидации транзакций и работы триггеров. Тесты выполнялись через DB Browser for SQLite для обеспечения корректности операций с базой данных.

5.6. Средства сопровождения

Для управления версиями и сопровождения выбраны Git и GitLab. Это система контроля версий для отслеживания изменений в коде и платформа для хостинга репозиториев. Обеспечивает совместную работу, ревью кода и автоматизацию задач CI/CD [17].

6. ОПИСАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И СТРУКТУР ДАННЫХ

Разрабатываемая система мониторинга безопасности периметра реализована как модульное приложение, сочетающее управление БПЛА, обработку данных в реальном времени и интеграцию с нейросетевыми модулями. Архитектура системы представлена через ключевые UML-диаграммы, отражающие её структуру, взаимодействие компонентов и физическое развертывание.

6.1. Архитектура системы

Система построена по модульному принципу, что обеспечивает гибкость и масштабируемость. Основные компоненты:

1. БПЛА:

— Оснащены камерами (H.264, 1080p), датчиками и прошивкой PX4.

— Передают видео через RTMP (UDP 5600) и телеметрию через MAVLink (UDP 14550).

2. Наземная станция:

— Qt-интерфейс: отображает карту объекта, тревоги, видео и телеметрию.

— Сервис управления БПЛА: обрабатывает команды (взлёт/посадка), маршруты патрулирования.

— Сервис аналитики: анализирует исторические данные, генерирует отчёты в PDF.

— ML-модуль: Детекция объектов (YOLO) и прогнозирование рисков (LSTM) в формате ONNX.

— Локальная БД (SQLite): хранит события, телеметрию, конфигурации.

Для концептуального отображения была разработана Архитектура системы в виде UML диаграммы, представленная в Приложении А, рисунок 1.

6.2. Диаграмма вариантов использования

Диаграмма, представленная на рисунке 6.2.1 демонстрирует, как система адаптируется под потребности разных ролей:

— Оператор получает инструменты для управления БПЛА в реальном времени, включая запуск патрулей и мониторинг телеметрии, что соответствует требованию к минимальной задержке видео (≤2 сек).

— Аналитик использует прогнозные модели (LSTM) и тепловые карты для анализа рисков, что реализует задачу предиктивного анализа из раздела 2.

— Администратор настраивает геозоны и распределяет доступ, обеспечивая соблюдение требований к безопасности (разграничение ролей, ГОСТ Р 56939-2022).

Связь сценариев, таких как Сбор информации и Прогнозирование рисков, подчёркивает интеграцию данных от БПЛА, сенсоров и ML-моделей в единый аналитический контур.

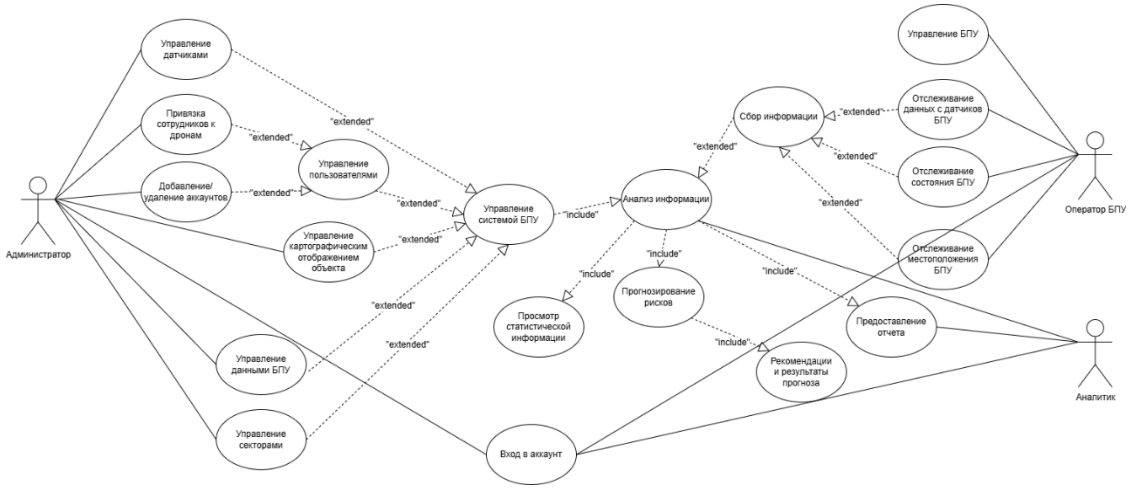


Рисунок 6.2.1 – Диаграмма вариантов использования (разработано автором)

6.3. Диаграмма пакетов

Группировка компонентов в пакеты (Рисунок 6.3.1) отражает модульность системы:

— Пакет данных о БПЛА обеспечивает хранение маршрутов и телеметрии, что поддерживает требование к масштабируемости (до 30 дронов).

— Пакет аналитики объединяет ML-модели (YOLO, LSTM) и инструменты генерации отчётов, выполняя задачу автоматизации анализа из раздела 2.1.

— Зависимость Пакета уведомлений от Пакета прогнозирования показывает, как система автоматически оповещает оператора о рисках, сокращая время реагирования.

Такая структура упрощает обновление компонентов, например, замену SQLite на PostgreSQL без изменения логики других модулей.

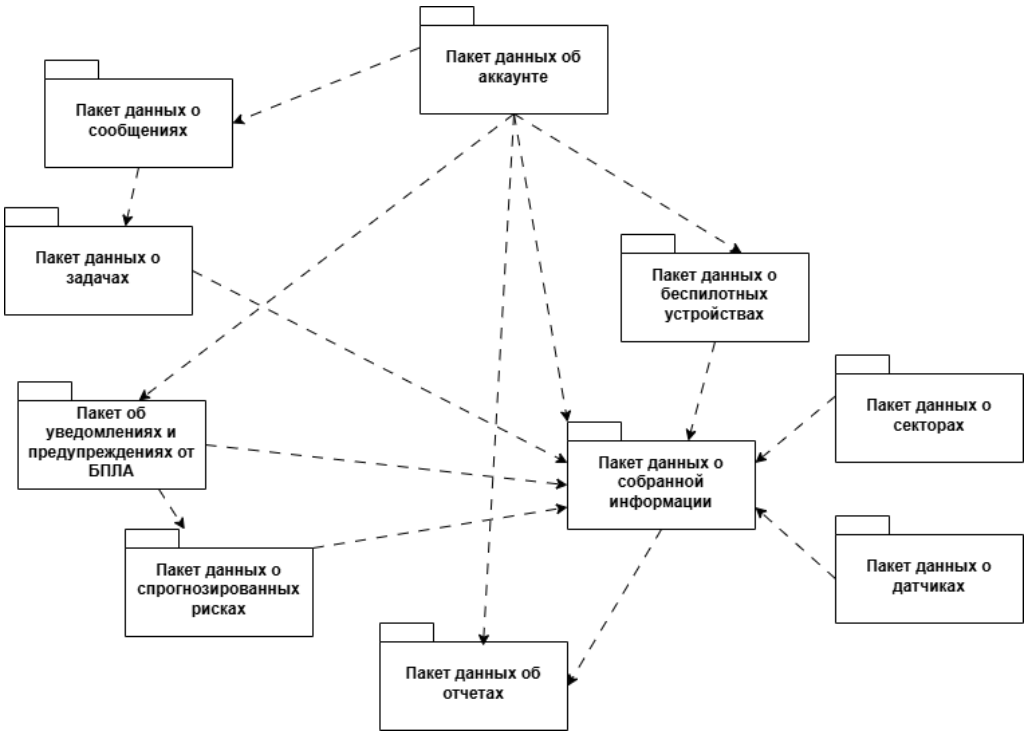


Рисунок 6.3.1 – Диаграмма пакетов (разработано автором)

6.4. Диаграмма компонентов

Взаимодействие компонентов, представленное в виде диаграммы на рисунке 6.4.1, раскрывает, как система достигает производительности 25 FPS:

— ML-модуль обрабатывает видео через YOLO, интегрированный в C++-ядро через ONNX, что соответствует требованию к скорости детекции.

— Сервис управления БПЛА использует MAVLink для передачи команд, обеспечивая стабильную связь даже при потере сигнала (автоматический возврат дронов).

— База данных синхронизирует данные между модулями, используя JSON для конфигураций и бинарные форматы для телеметрии, что минимизирует накладные расходы.

Интерфейсы (например, REST API на порту 8080) позволяют интегрировать систему со сторонними SCADA-решениями через OPC UA.

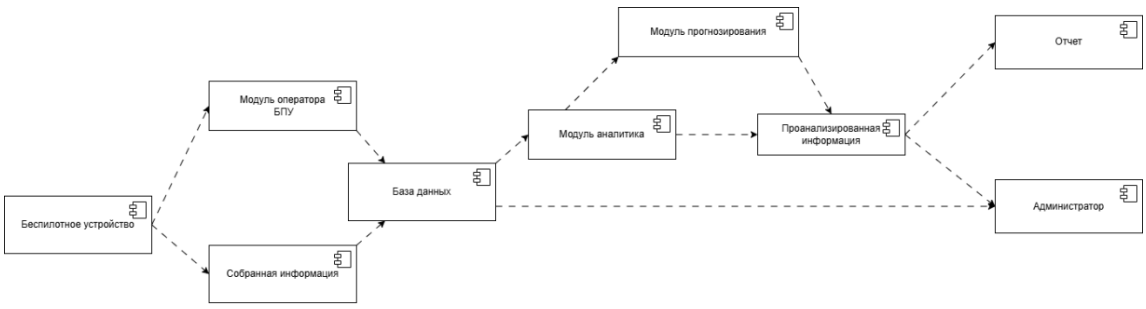


Рисунок 6.4.1 – Диаграмма компонентов (разработано автором)

6.5. Диаграмма развертывания

Физическая архитектура обеспечивает работу в агрессивных условиях:

— БПЛА передают видео по RTMP (битрейт 6 Мбит/с), что соответствует разрешению 1280×720 из требований к данным.

— Наземная станция развернута на оборудовании с GPU NVIDIA GTX 1660+, что гарантирует обработку ML-моделей без задержек.

— Локальная БД (SQLite) и внешнее хранилище резервных копий обеспечивают автономность, критичную для объектов без стабильного интернета.

Использование UDP для видео и MAVLink для телеметрии снижает зависимость от качества канала связи [18].

Для ознакомления с разработанной диаграммой развертывания см. Приложении А, рисунок 2.

Архитектура системы, отражённая в диаграммах и фрагментах кода, обеспечивает:

— Автономность: Локальная БД и обработка данных без облачных зависимостей.

— Производительность: Обработка видео 25 FPS на GPU, прогнозирование в реальном времени.

— Масштабируемость: Возможность подключения до 30 БПЛА, переход на PostgreSQL для больших объёмов данных. Диаграммы демонстрируют соответствие требованиям к безопасности, включая шифрование данных (AES-256) и разграничение прав доступа.

6.6. Описание разработки базы данных

В разрабатываемой системе мониторинга безопасности периметра ключевым аспектом является проектирование эффективной базы данных, обеспечивающей надёжное хранение, обработку и обмен информацией между компонентами [19]. Основные решения в этой области базируются на требованиях к локальности работы, производительности и масштабируемости, сформулированных в разделе 2. Для визуального представление базы данных была разработана ER-диаграмма базы данных.

На рисунке 6.6.1 представлена ER-диаграмма базы данных, отражающая ключевые сущности системы и их взаимосвязи:

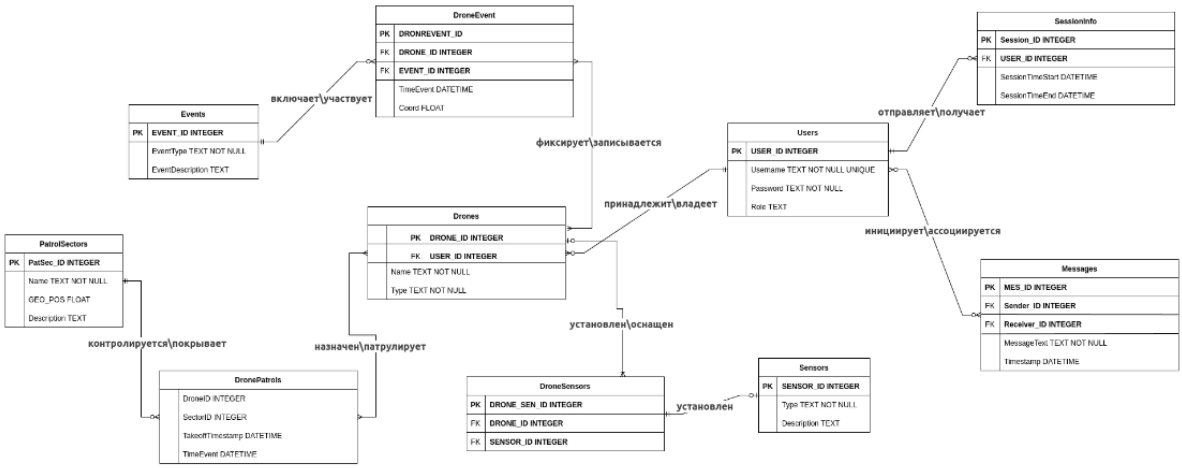


Рисунок 6.6.1 - ER-диаграмма базы данных (разработано автором)

В таблице 6.6.1 приведен словарь основных таблиц базы данных, отражающих сущности и связи, описанные в логической модели.

Таблица 6.6.1 - Словарь основных таблиц базы данных (составлено автором)

Таблица	*Поле*	*Описание*
Users	USER_ID	Целочисленный тип до 4 байт. Уникальный идентификатор пользователя.
	Username	Строковый тип, длина до 255 символов. Имя пользователя.
	Password	Строковый тип, длина до 255 символов. Пароль пользователя.
	Role	Строковый тип, длина до 50 символов. Роль пользователя.
Drones	DRONE_ID	Целочисленный тип до 4 байт. Уникальный идентификатор дрона.
	Name	Строковый тип, длина до 255 символов. Имя дрона.
	Type	Строковый тип, длина до 255 символов. Тип дрона.
	USER_ID	Целочисленный тип до 4 байт. Идентификатор пользователя, закреплённого за дроном.

Продолжение таблицы 6.6.1

PatrolSectors	SECTOR_ID	Целочисленный тип до 4 байт. Уникальный идентификатор сектора.
	Name	Строковый тип, длина до 255 символов. Имя сектора.
	Description	Строковый тип, длина до 65535 символов. Описание сектора.
DronePatrols	DRONE_ID	Целочисленный тип до 4 байт. Идентификатор дрона.
	SECTOR_ID	Целочисленный тип до 4 байт. Идентификатор сектора.
	TakeoffTimestamp	Дата и время взлёта дрона. Формат: YYYY-MM-DD HH:MM:SS.
	TimeEvent	Дата и время события. Формат: YYYY-MM-DD HH:MM:SS.
Sensors	SENSOR_ID	Целочисленный тип до 4 байт. Уникальный идентификатор сенсора.
	Type	Строковый тип, длина до 255 символов. Тип сенсора.
	Description	Строковый тип, длина до 65535 символов. Описание сенсора.
DroneSensors	DRONE_ID	Целочисленный тип до 4 байт. Идентификатор дрона.
	SENSOR_ID	Целочисленный тип до 4 байт. Идентификатор сенсора.
Events	ID	Целочисленный тип до 4 байт. Уникальный идентификатор события.
	EventType	Строковый тип, длина до 255 символов. Тип события.
	EventDescription	Строковый тип, длина до 65535 символов. Описание события.
DroneEvent	EventID	Целочисленный тип до 4 байт. Идентификатор события.
	DRONE_ID	Целочисленный тип до 4 байт. Идентификатор дрона.

Окончание таблицы 6.6.1

	TimeEvent	Дата и время события. Формат: YYYY-MM-DD HH:MM:SS.
SessionInfo	SESSION_ID	Целочисленный тип до 4 байт. Уникальный идентификатор сессии.
	USER_ID	Целочисленный тип до 4 байт. Идентификатор пользователя.
	SessionTimeStart	Дата и время начала сессии. Формат: YYYY-MM-DD HH:MM:SS.
	SessionTimeEnd	Дата и время окончания сессии. Формат: YYYY-MM-DD HH:MM:SS.
Messages	MESSAGE_ID	Целочисленный тип до 4 байт. Уникальный идентификатор сообщения.
	SenderID	Целочисленный тип до 4 байт. Идентификатор отправителя.
	ReceiverID	Целочисленный тип до 4 байт. Идентификатор получателя.
	MessageText	Строковый тип, длина до 65535 символов. Текст сообщения.
	Timestamp	Дата и время отправки. Формат: YYYY-MM-DD HH:MM:SS.

6.7. Внутреннее устройство реализованной программы

Для реализации программного обеспечения была выбрана связка языка программирования C++ и фреймворка Qt версии 6, что подробно обосновано в разделе 4.2. C++ предоставляет мощные средства объектно-ориентированного программирования (ООП) [20], включая абстракцию, инкапсуляцию, наследование и полиморфизм, что способствует созданию модульного и масштабируемого кода. Qt6 дополняет стандартные возможности C++ собственной объектной моделью, основанной на классе QObject и механизме сигналов и слотов, что обеспечивает эффективную коммуникацию между объектами и поддержку событийно-ориентированной архитектуры. Использование Meta-Object Compiler (moc) позволяет реализовать динамическую типизацию, систему свойств и другие расширения, необходимые для построения гибких и расширяемых пользовательских

интерфейсов [21]. Таким образом, сочетание C++ и Qt6 обеспечивает высокую производительность, кроссплатформенность и удобство разработки сложных приложений с графическим интерфейсом.

В процессе проектирования архитектуры системы особое внимание было уделено модульности и разделению ответственности между компонентами. Это позволило создать гибкую структуру, в которой каждый модуль выполняет строго определённые функции, обеспечивая при этом лёгкость сопровождения и расширения системы. Основные модули включают в себя:

— MainISODWindow: главное окно приложения, управляющее отображением различных пользовательских интерфейсов в зависимости от роли пользователя.

— DBManager: модуль для взаимодействия с базой данных, обеспечивающий хранение и извлечение информации о пользователях, дронах, секторах и событиях.

— ExtraModules: вспомогательные модули, включая обработку данных, детекцию объектов с использованием YOLO и предсказание аномалий с помощью модели LSTM.

— LoginManager: модуль, отвечающий за аутентификацию пользователей и управление сессиями.

— MapBuilder: компонент для отображения карты и взаимодействия с географическими данными, включая построение и валидацию периметра охраняемой зоны.

— UsersWindow: интерфейсы для различных ролей пользователей, таких как администратор, оператор, аналитик и обычный пользователь.

Для наглядного представления архитектуры системы были разработана модульная диаграмма классов, отражающие взаимодействие между основными компонентами и их внутреннюю структуру. Диаграмма представлена на рисунке 6.7.1.

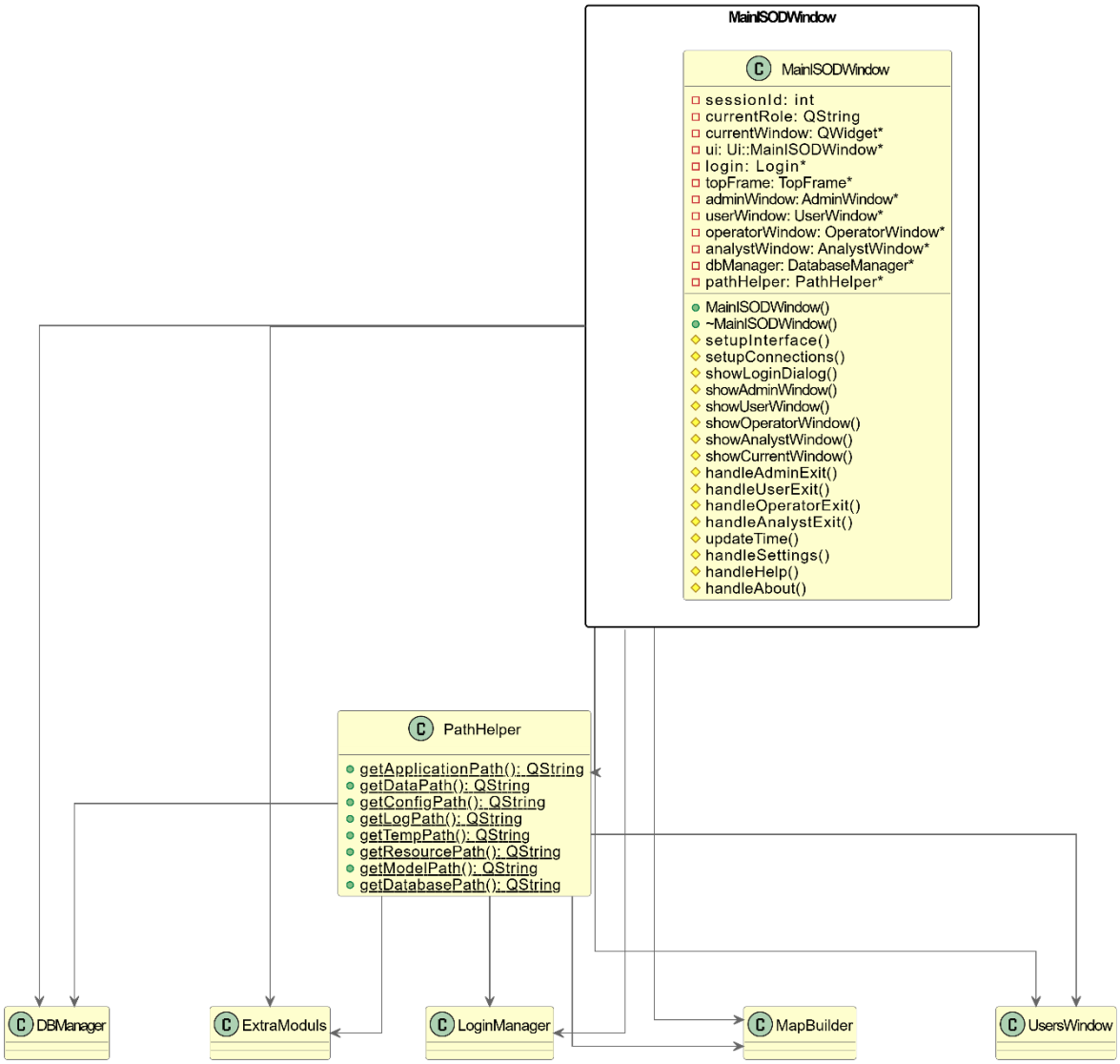


Рисунок 6.7.1 – Модульная диаграмма классов (разработано автором)

[redacted]

6.8. Разработка пользовательских интерфейсов

Для обеспечения эффективного взаимодействия пользователей с системой мониторинга безопасности периметра, разработан графический пользовательский интерфейс, адаптированный к различным ролям пользователей, что соответствует требованиям, указанным в разделе 1.3. Основное окно приложения предоставляет доступ к функциональным модулям в зависимости от уровня доступа, включая административные панели, операторские интерфейсы, аналитические инструмент. Интерфейс построен с использованием Qt6 и C++, что обеспечивает кроссплатформенность, высокую производительность и гибкость в разработке.

Визуальные компоненты интерфейса реализованы с применением Qt Widgets, что позволяет создавать сложные и настраиваемые элементы управления, соответствующие требованиям каждого типа пользователя. Интерфейс обеспечивает интуитивно понятную навигацию, отображение данных в реальном времени и доступ к аналитическим инструментам, что способствует повышению эффективности работы пользователей. На рисунках 6.8.1, 6.8.2, 6.8.3, 6.8.4, 6.8.5, 6.8.6, 6.8.7, 6.8.8 показаны основные пользовательские интерфейсы разработанной программы.

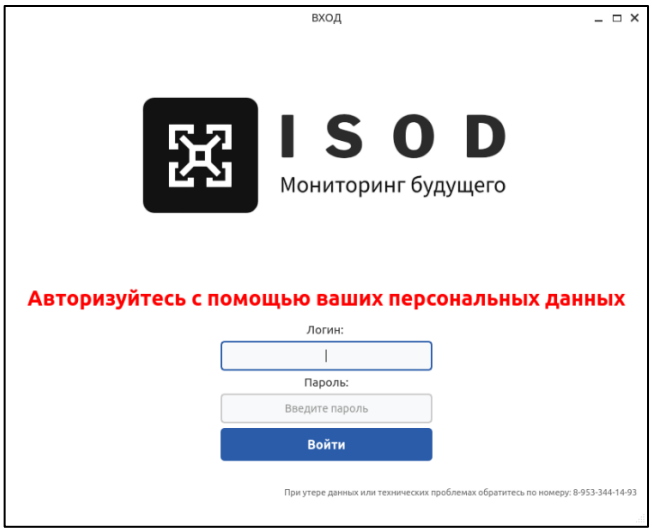


Рисунок 6.8.1 – Окно авторизации в системе (разработано автором)

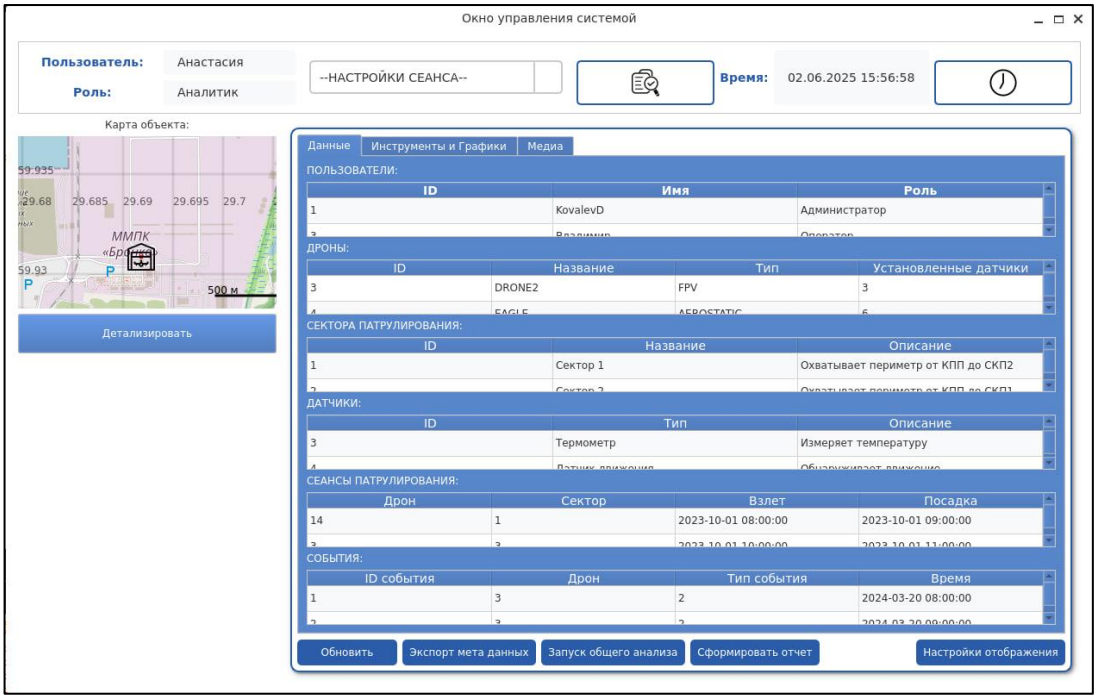


Рисунок 6.8.2 – Главное окно системы: роль аналитик, первая страница(разработано автором)

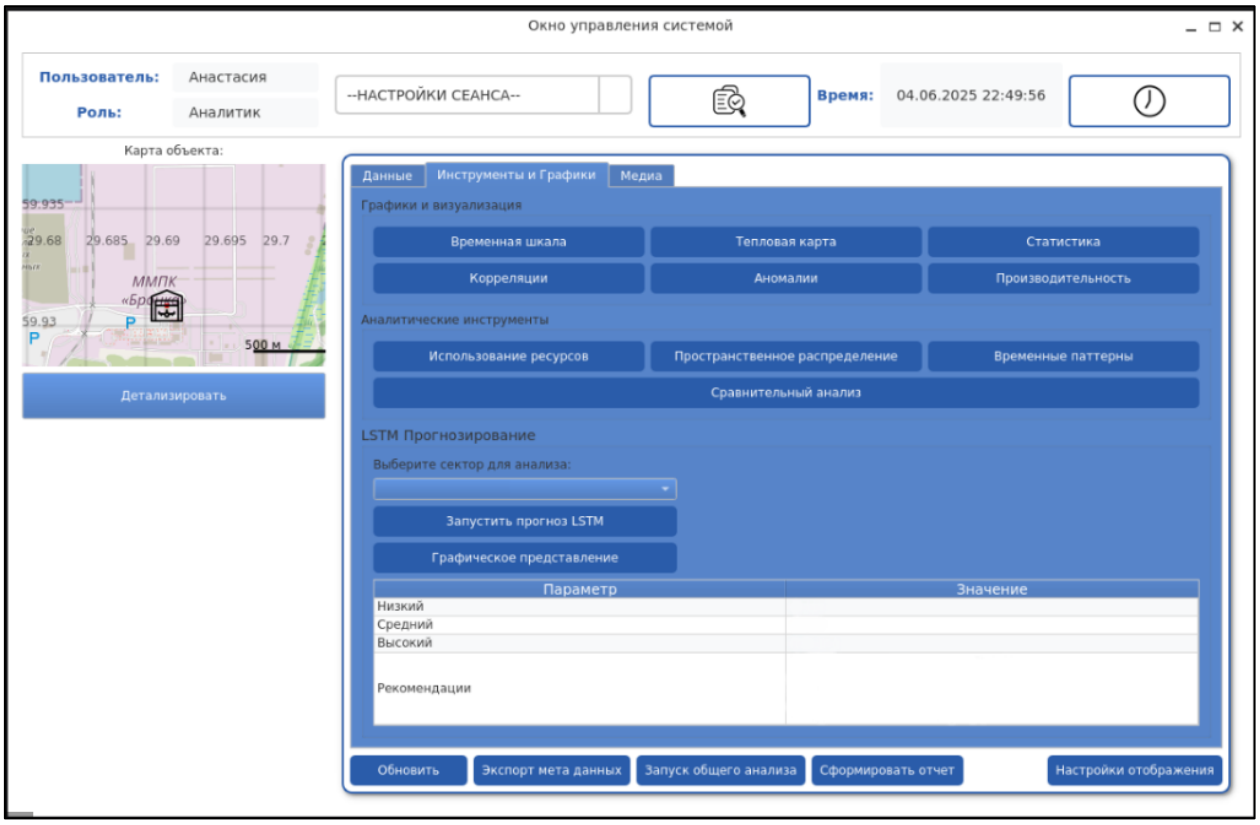


Рисунок 6.8.3 – Главное окно системы: роль аналитик, вторая страница(разработано автором)

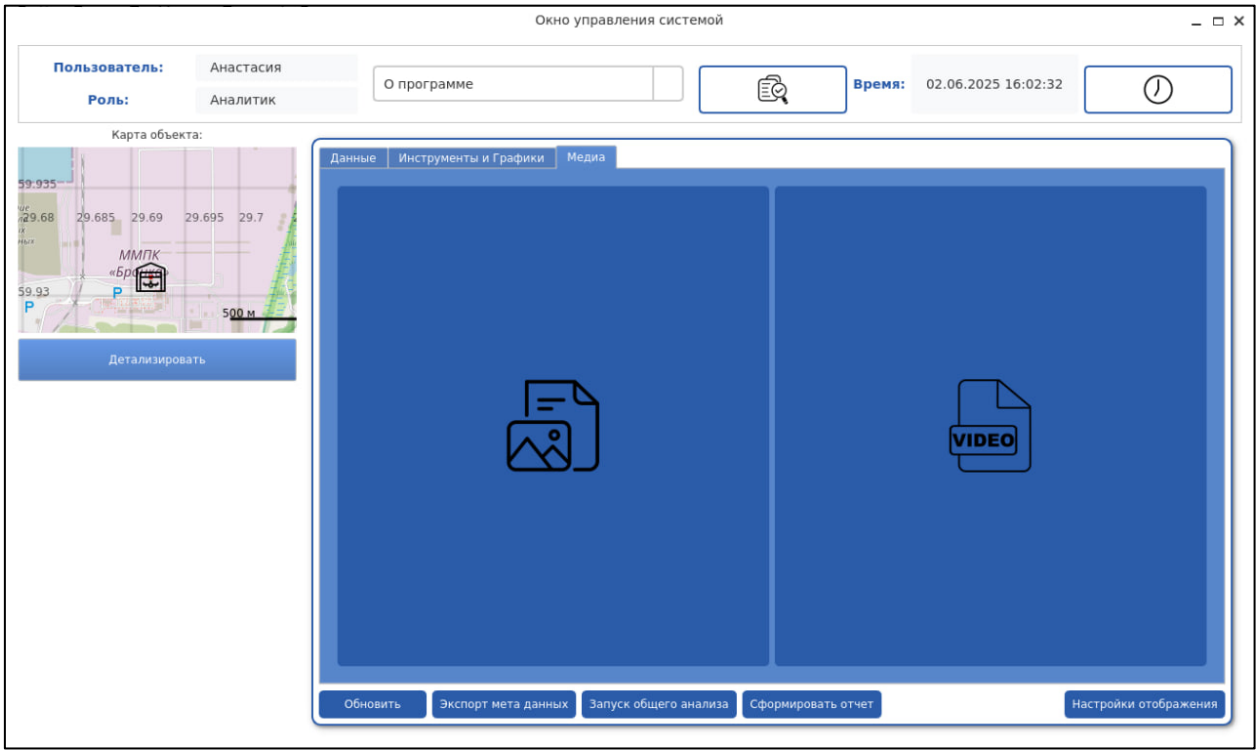


Рисунок 6.8.4 – Главное окно системы: роль аналитик, третья страница(разработано автором)

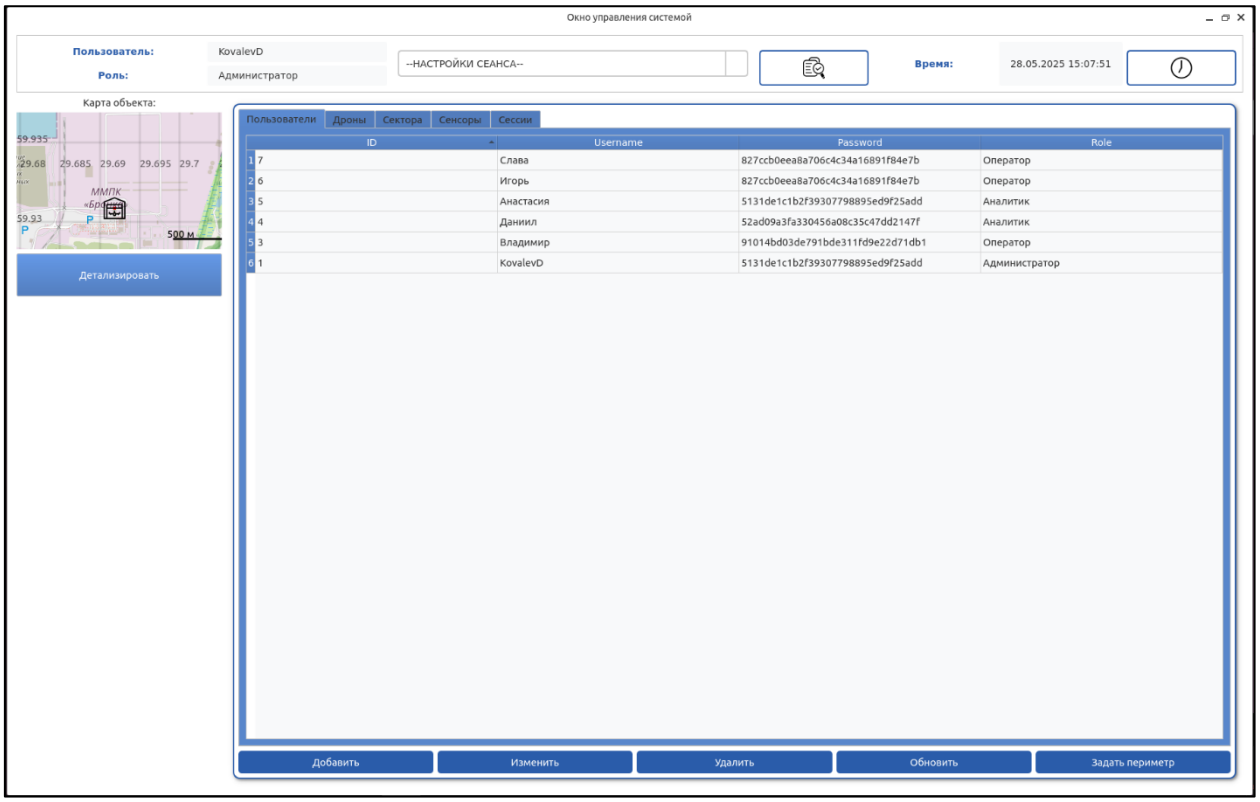


Рисунок 6.8.5 – Главное окно системы: роль администратор (разработано автором)

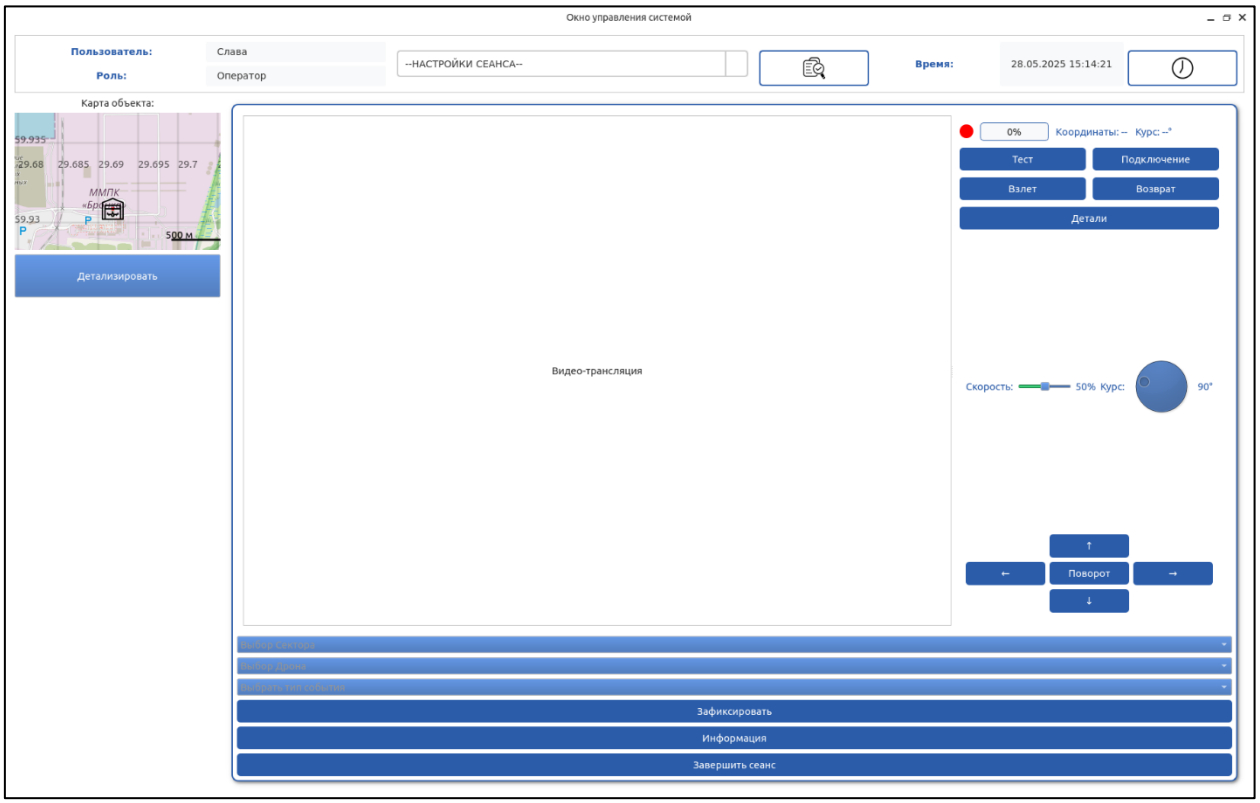


Рисунок 6.8.6 – Главное окно системы: роль оператор (разработано автором)

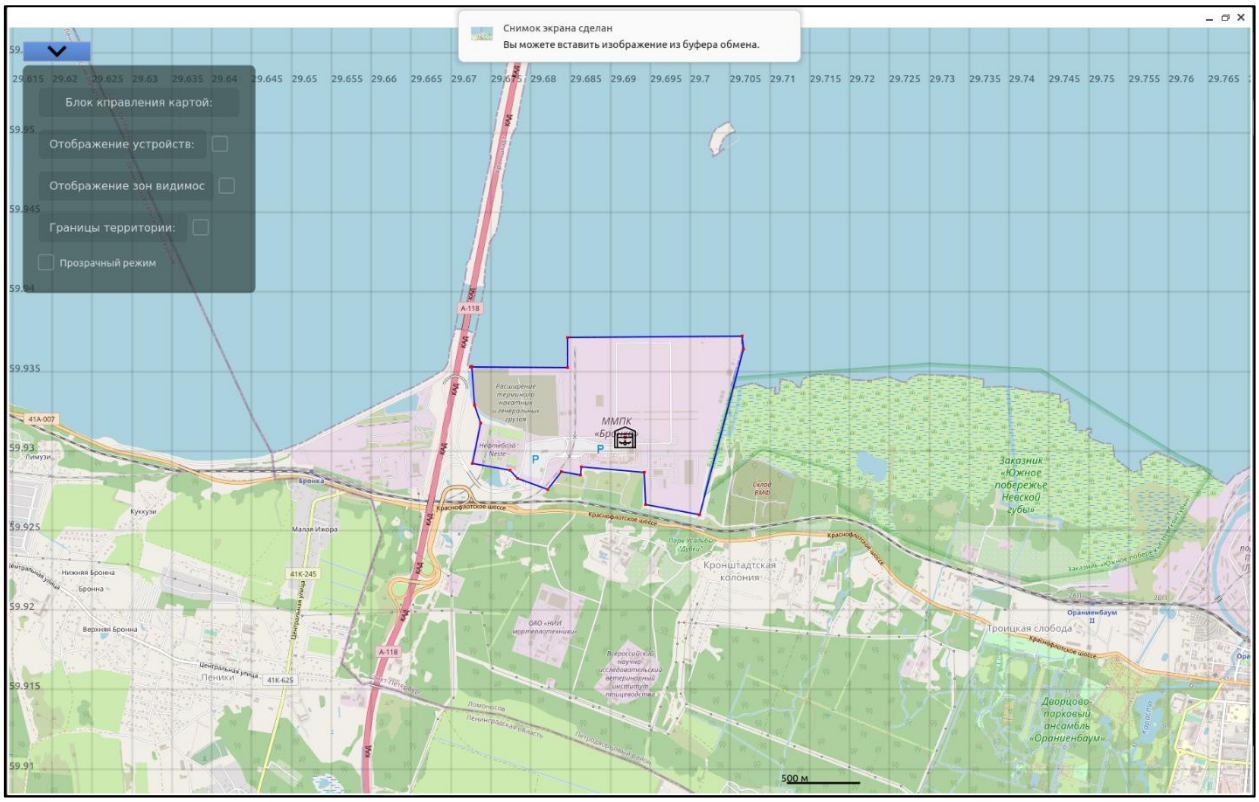


Рисунок 6.8.7 – Окно карты (разработано автором)

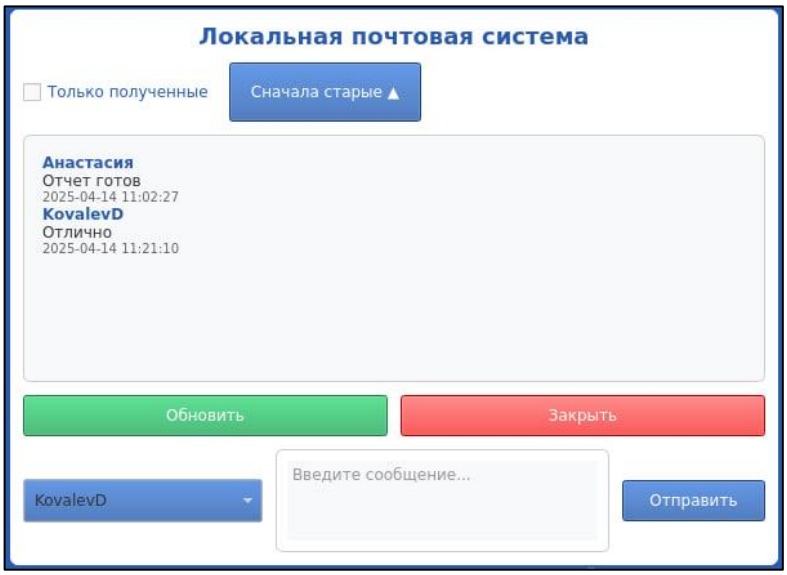


Рисунок 6.8.8 – Окно почтового клиента (разработано автором)

[redacted]