



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ Информатика, искусственный интеллект и системы управления _____

КАФЕДРА _____ Теоретическая информатика и компьютерные технологии _____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

*База данных учета приборов
мониторинга лесных пожаров*

Студент _____ ИУ9-62Б _____
(Группа)

(Подпись, дата) _____ М.А.Пехова _____
(И.О.Фамилия)

Руководитель курсовой работы

(Подпись, дата) _____ Д.П.Посевин _____
(И.О.Фамилия)

Консультант

(Подпись, дата) _____ (И.О.Фамилия)

2025 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	2
1. Обзор предметной области	5
2. Проектирование структуры базы данных	8
2.1. Модель «сущность-связь»	8
2.3. Нормализация базы данных	9
2.3.1. Первая нормальная форма	9
2.3.2. Вторая нормальная форма	10
2.3.3. Третья нормальная форма	10
2.3.4. Бойс-Кодд нормальная форма	12
2.3.5. Четвертая нормальная форма	12
2.3.6. Пятая нормальная форма	13
2.3.7. Шестая нормальная форма	13
2.3.8. Результат нормализации	13
2.4. Преобразование модели «сущность-связь» в реляционную	15
3. Выбор технологий и инструментов	24
4. Разработка схемы базы данных	27
5. Разработка API для взаимодействия с базой данных	29
6. Тестирование и отладка	31
7. Интеграция в пользовательское веб-приложение	33
8. Документация и поддержка	37
ЗАКЛЮЧЕНИЕ	38
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	40
ЛИСТИНГ ПРОГРАММ	41

ВВЕДЕНИЕ

Интеграция баз данных в сервисы и приложения, предназначенные для мониторинга природных рисков, является одной из ключевых задач современной разработки. В эпоху ускоренной цифровизации, когда своевременные данные становятся основой управленческих решений и стратегического планирования, требуется обеспечить быстрый и надёжный доступ к информации, поступающей от различных источников. Это особенно важно в системах мониторинга лесных пожаров, где необходимо обрабатывать большой поток телеметрии от датчиков температуры, влажности, содержания CO_2 и других показателей окружающей среды.

Приложения, выполняющие такие задачи, востребованы в лесном хозяйстве, экологии, страховании, туризме и государственных структурах, ответственных за охрану природных ресурсов. Они позволяют в режиме реального времени обнаруживать аномалии, формировать прогнозы пожарной опасности, оперативно предупреждать ответственных лиц и координировать действия пожарных служб. Интеграция базы данных обеспечивает централизованное хранение, валидацию и создание истории поступающих данных, упрощая последующий анализ и подготовку управленческих отчётов.

Актуальность работы обусловлена двумя основными факторами. Во-первых, объём телеметрии, генерируемой сотнями распределённых приборов, может исчисляться десятками миллионов записей в сутки; без правильно спроектированной БД такие данные невозможно обрабатывать корректно. Во-вторых, для совместной работы лесничеств, диспетчерских центров и мобильных групп требуется единое, согласованное приложение, поддерживающее синхронизацию и разграничение доступа.

Интеграция базы данных в такие системы повышает их стабильность и расширяемость, обеспечивая конкурентные преимущества на рынке информационных технологий для природоохранного сектора.

Таким образом, создание надёжной базы данных в приложении мониторинга лесных пожаров является актуальной и востребованной задачей,

способствующей повышению точности прогнозов, удобства эксплуатации и конкурентоспособности конечного продукта.

Цель данной работы — разработать базу данных, обеспечив эффективное логирование телеметрии, её анализ, хранение исторических срезов и поддержку совместной работы пользователей.

Для достижения цели необходимо решить следующие задачи:

1. Анализ требований к базе данных: выявить функциональные и нефункциональные требования к БД; определить основные сущности; установить требования к производительности, безопасности и надёжности.
2. Проектирование структуры базы данных: Разработать ER-диаграмму, нормализовать таблицы, задать связи «многие-ко-многим» через промежуточные отношения, выбрать подходящие типы данных и индексы.
3. Выбор технологий и инструментов: определить подходящие технологии и инструменты для реализации базы данных, учитывая требования к производительности, масштабируемости и совместимости с веб-интерфейсом.
4. Создание физической схемы базы данных: создать таблицы, первичные и внешние ключи, ограничения CHECK, триггеры для автоматического расчёта показателей, представления для аналитических выборок.
5. Разработка API для взаимодействия с базой данных: спроектировать REST-интерфейс для работы с объектами БД, реализовать энд-пойнты CRUD и агрегирующие запросы, обеспечить сериализацию и валидацию данных.
6. Тестирование и отладка: провести тестирование разработанной базы данных на соответствие требованиям, выявить и устранить возможные ошибки и недочеты в функциональности и производительности.

7. Подключение веб-клиента и сторонних приложений к REST-API, предоставленному Django REST Framework; обеспечить выдачу данных в JSON и возможность CRUD-операций через открытые энд-пойнты.
8. Оптимизация и масштабирование: провести оптимизацию производительности базы данных и ее масштабирование для обеспечения эффективной работы при увеличении объема данных и нагрузки.
9. Документация и поддержка: создать документацию к базе данных, описывающую ее структуру, функциональность и способы использования, а также обеспечить поддержку и сопровождение разработанной базы данных после ее внедрения.

1. Обзор предметной области

Информационная система учета приборов мониторинга лесных пожаров предназначена для сбора, хранения и анализа данных о параметрах окружающей среды, посылаемых устройствами. Основная цель данной системы – учет приборов, которые передают полученные измерения о состоянии окружающей среды, определение их характеристик и географических координат. Это важно для анализа вероятности возникновения пожара на конкретной местности, соответственно и для его своевременного детектирования.

Основные сущности системы:

1. локация (Location):
 - a. location name: название местности;
 - b. description: район мониторинга.
2. экземпляры приборов (Device):
 - a. inventory number: инвентарный номер;
 - b. device type: тип прибора;
 - c. latitude: градус широты;
 - d. longitude: градус долготы;
 - e. installation date: дата установки.
3. параметры окружающей среды (EnvironmentalParameters):
 - a. inventory number: инвентарный номер;
 - b. recorded at: время и дата записи;
 - c. temperature: температура;
 - d. humidity: влажность;
 - e. co2 level: уровень co2;
 - f. processed: флаг обработки.
4. проанализированная информация (AnalyzedInformation):
 - a. number of analysis: номер анализа;
 - b. analyzed at: дата и время анализа;
 - c. fire hazard: вероятность возгорания.

5. оповещения (Alarm):

- a. number of alarm: номер оповещения;
- b. status of alarm: статус;
- c. alarm level: уровень тревоги;
- d. date of alarm: дата;
- e. time of alarm: время.

6. инцидент (Incident):

- a. number of incident: номер инцидента;
- b. status of incident: статус;
- c. description: описание;
- d. time of detection: время обнаружения;
- e. time of resolve: время разрешения инцидента;
- f. location name: имя местности;
- g. time window start: начало временного окна;
- h. time window end: конец временного окна.

Взаимосвязи между сущностями:

- каждое устройство закрепляется за конкретной локацией, то есть в одной местности может находиться несколько устройств, тогда как каждое устройство связано только с одной локацией;
- каждое устройство генерирует множество записей с данными об окружающей среде, фиксируемыми в таблице параметров;
- один и тот же набор исходных данных, зафиксированный в записи параметров окружающей среды, может быть использован для проведения одного конкретного анализа;
- каждое проанализированное событие приводит к формированию максимум одного оповещения;
- каждое оповещение может приводить к созданию одного инцидента, поскольку оно сигнализирует о конкретной

потенциальной угрозе, требующей реагирования; в то время как один инцидент может включать в себя несколько оповещений.

Эта модель позволяет структурировать данные, необходимые для поиска устройств, считывающих информацию о состоянии окружающей среды, и предоставляет возможности для анализа и визуализации полученных данных, что является ключевым требованием для обеспечения надежного мониторинга и детектирования лесных пожаров и впоследствии построении корректной реляционной модели базы данных.

2. Проектирование структуры базы данных

2.1. Модель «сущность-связь»

На основе сформулированных требований к базе данных была построена модель «сущность-связь» (см. рисунок 1).

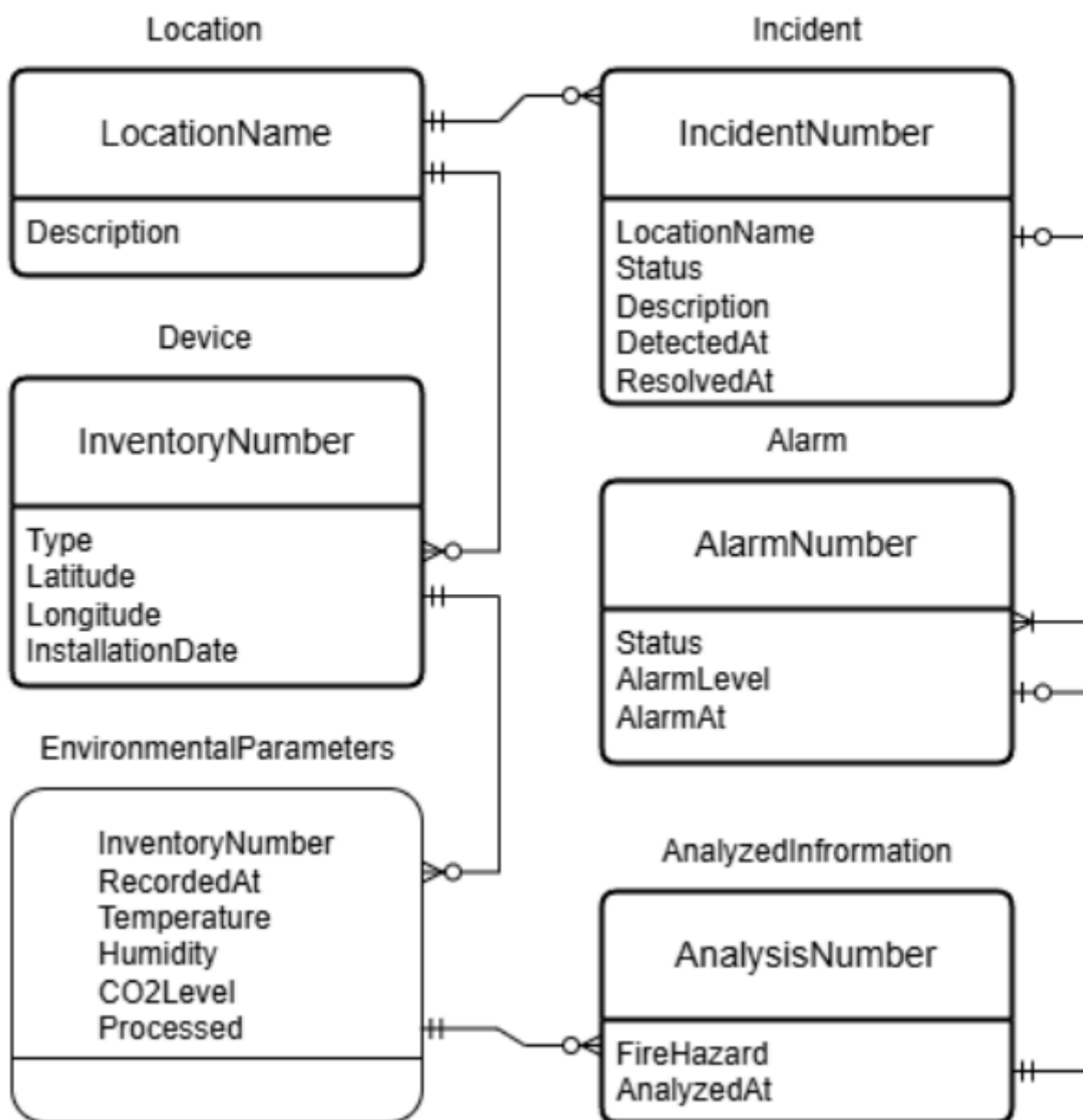


Рисунок 1 – ER-модель приложения мониторинга лесных пожаров

2.2. Нормализация базы данных

Оптимизация структуры базы данных через нормализацию предполагает последовательное преобразование информации в соответствии с требованиями шести уровней нормальных форм (от 1НФ до 6НФ). Каждая ступень этой иерархии решает конкретные проблемы: минимизирует дублирование данных, исключает противоречия и ошибки при операциях обновления или удаления. Рассмотрим, как эти принципы применяются к системе отслеживания лесных пожаров, где точность и согласованность данных критически важны для оперативного анализа угроз.

2.2.1. Первая нормальная форма

Первая нормальная форма требует, чтобы каждая ячейка таблицы содержала атомарное значение, а внутри таблицы не было повторяющихся групп столбцов.

1. Локация `location(id, location_name, description)`:

Таблица `location` уже соответствует первой нормальной форме, так как все поля содержат атомарные данные: `location_id`, `location_name`, `description` — все значения уникальны и не разбиваются на составные части.

2. Девайс `device(id, location_id, inventory_number, type, date_of_installation, latitude, longitude)`:

В таблице `device` все поля также атомарные: `device_id`, `location_id`, `inventory_number`, `type`, `date_of_installation`, `latitude`, `longitude` — никаких повторяющихся групп нет.

3. Параметры окружающей среды `environmental_parameters(id, device_id, temperature, humidity, co2_level, recorded_at, processed)`:

Все поля также соответствуют первой нормальной форме: `recorded_data_id`, `device_id`, `temperature`, `humidity`, `co2_level`, `recorded_at`, `processed` — никаких групп повторяющихся данных нет.

4. Проанализированная информация `analyzed_information(id, recorded_data_id, fire_hazard, analyzed_at)`:

Таблица `analyzed_information` также соответствует первой нормальной форме, все поля атомарные: `analysis_id`, `recorded_data_id`, `fire_hazard`, `analyzed_at`.

5. Оповещения `alarm(id, analysis_id, incident_id, alarm_level, status, alarm_at)`:
Все поля в таблице `alarm` атомарные: `alarm_id`, `analysis_id`, `incident_id`, `alarm_level`, `status`, `alarm_at`.

6. Инциденты `incident(id, location_id, time_window_start, time_window_end, detected_at, resolved_at, status, description)`:
В таблице `incident` все поля атомарные: `incident_id`, `location_id`, `status`, `description`, `detected_at`, `resolved_at`, `time_window_start`, `time_window_end`.

В каждой таблице все поля представляют собой атомарные значения, нет массивов или вложенных структур. Следовательно, все таблицы находятся в 1НФ.

2.2.2. Вторая нормальная форма

Для отношений с составным первичным ключом каждый неключевой атрибут обязан зависеть от всего ключа, а не от его части.

В каждой таблице используется суррогатный первичный ключ `id` (один столбец). Частичных зависимостей быть не может. Дополнительно проверяем, что других кандидатных ключей, состоящих из части РК, нет. Требование второй нормальной формы выполнено для всех таблиц.

2.2.3. Третья нормальная форма

Третья нормальная форма (и более строгая нормальная форма Бойса-Кодда) требует отсутствия транзитивных зависимостей вида «РК → поле1 → поле2», когда поле2 зависит от поля1, а поле1 зависит от первичного ключа. Рассмотрев архитектуру базы данных получим, что в каждой таблице мы имеем зависимость вида `primary_key` → (все остальные поля), и больше никаких нет, поскольку нет лишних атрибутов, которые повторяют данные из других таблиц или дублируют их. Например, если бы в таблице `device` дублировали `location_name` (чтобы быстро смотреть название местности без джойна), то это потенциально могло бы нарушать третью нормальную форму.

1. Локация (location):

В таблице location нет транзитивных зависимостей. Поля location_name и description зависят только от location_id, и больше ничего не зависит от других полей.

2. Устройство (device):

В таблице device нет транзитивных зависимостей. Все атрибуты зависят только от device_id.

3. Параметры окружающей среды (environmental_parameters):

Рассмотрим поле processed. По своей природе это булево поле служит флагом, отмечающим факт «обработки» (анализа) конкретной записи телеметрии. Семантически processed = True устанавливается только после того, как для данной строки создана запись в AnalyzedInformation. Однако эта связь реализована в бизнес-логике (в таске calculate_hazard_batch и в механизме bulk_update), а не декларативно в определении таблицы. С точки зрения схемы БД processed не зависит от другого неключевого поля внутри environmental_parameters. Оно напрямую зависит только от id записи (то есть $A = id \rightarrow B = processed$). Никакой зависимости вида temperature; processed или recorded_at; processed не существует на уровне DDL. В Incident есть бизнес-правило: resolved_at заполняется, когда status = 'closed'. Но это тоже не функциональная зависимость на уровне модели — просто моментальные значения, контролируемые сигналами/логикой приложения. В Alarm флаг status и время alarm_at никак не влияют друг на друга с точки зрения схемы.

4. Проанализированная информация (analyzed_information):

В таблице analyzed_information нет транзитивных зависимостей. Все атрибуты зависят от analysis_id.

5. Оповещения (alarm):

В таблице alarm нет транзитивных зависимостей. Все атрибуты зависят от alarm_id.

6. Инциденты (incident):

В таблице incident нет транзитивных зависимостей. Все атрибуты зависят от incident_id.

Таким образом, несмотря на то что некоторые поля («processed», «resolved_at» и проч.) логически координируются с другими атрибутами, никакой транзитивной зависимости внутри таблиц нет, и схема полностью удовлетворяет требованиям третьей нормальной формы.

2.2.4. Бойс-Кодд нормальная форма

Теория. Для каждой нетривиальной функциональной зависимости $X \rightarrow Y$ детерминанта X должна быть кандидатом в ключи.

Во всех таблицах $X = id$, что, несомненно, кандидатный ключ. В таблице alarm объявлено ограничение UNIQUE (analysis_id, alarm_level) — эта пара тоже становится кандидатом в ключи. Все зависимости, где $X = (analysis_id, alarm_level)$, удовлетворяют критерию «X — кандидатный ключ».

Таким образом, нормальная форма Бойса-Кодда выполнена.

2.2.5. Четвертая нормальная форма

Четвертая нормальная форма требует отсутствия нетривиальных многозначных зависимостей. Чаще всего многозначные зависимости возникают, если в одной таблице хранятся «списки» значений, повторяющихся для одних и тех же ключей. Например, хрестоматийный пример выглядит так, например, «автор книги» и «иллюстратор книги» — одна и та же книга может иметь несколько авторов и нескольких иллюстраторов, и тогда таблицу приходится разносить.

В представленной схеме не просматриваются никакие атрибуты, которые могли бы одновременно принимать множество значений при одном и том же ключе. Все потенциальные «множественные» связи (например, устройство может иметь множество записей environmental_parameters) реализованы через связь один-ко-многим ($device_id \rightarrow$ множество записей в environmental_parameters) при помощи отдельной таблицы.

2.2.6. Пятая нормальная форма

Пятая нормальная форма ориентирована на устранение нетривиальных соединительных зависимостей между более, чем двумя таблицами, которые не следуют из ключей.

То есть, пятая нормальная форма требует, чтобы все возможные связи между элементами таблицы полностью задавались «через ключи» и не было таких «огромных» таблиц, которые можно ещё раз «распилить» на три и более таблиц (не на две) без потери данных и без появления аномалий при обновлении.

Как правило, проблемы с пятой нормальной формой, — это сложные «многие-ко-многим-ко-многим» отношения (когда одна таблица может быть результатом соединения сразу нескольких в каком-то нетривиальном виде). Например, Alarm опосредует потенциальную N:M-связь между AnalyzedInformation и Incident, сама являясь полноценной бизнес-сущностью; дополнительных скрытых зависимостей не возникает.

Таким образом, 5НФ соблюдена.

2.2.7. Шестая нормальная форма

Строгая шестая нормальная форма (один факт — одна таблица) редко применяется в OLTP-части из-за большого числа JOIN-ов, однако при высокочастотной телеметрии и требовании версионирования отдельных атрибутов она становится разумной альтернативой “широкой” таблице. В нашей текущей реализации все показания поступают пакетом, поэтому третьей нормальной формы/нормальной формы Бойса-Кодда достаточно; при переходе к более мелкому шагу времени потребуется рассмотреть гибридное time-series-хранилище.

2.2.8. Результат нормализации

Схема удовлетворяет первой, второй, третьей четвертой, пятой нормальным формам; шестая нормальная форма сознательно не применяется как избыточная для оперативной системы. Такая степень нормализации

минимизирует дублирование и исключает аномалии вставки, удаления и обновления, обеспечивая надёжное хранение данных для мониторинга лесных пожаров.

2.3. Преобразование модели «сущность-связь» в реляционную

Первый этап – при преобразовании ER-диаграммы в реляционную схему мы «разворачиваем» все связи «многие-ко-многим» в отдельные таблицы-связки и одновременно вводим явные первичные ключи у тех сущностей, которые логически зависят от других (т. е. «слабых» сущностей).

Второй этап – для каждой основной (не-связующей) таблицы создаём однозначный идентификатор вида <Сущность>ID (surrogate-РК), выделяем из атрибутов естественные ключи-кандидаты и оставляем их помеченными как UNIQUE, а сам РК называем просто id.

1. Ассоциативные (M:N) связи не использовались (в предметной области таких нет), а все отношения 1:N и N:1 оформлены через FK с on_delete-семантикой.
2. Супрогатные ключи: у каждой модели есть авто-поле id.
3. Кандидаты в ключи: добавлено поле unique=True на поля inventory_number в Device и location_name в Location.

После описанных итераций, включающих в себя все этапы нормализации, мы получаем верную реляционную модель (см. рисунок 2).

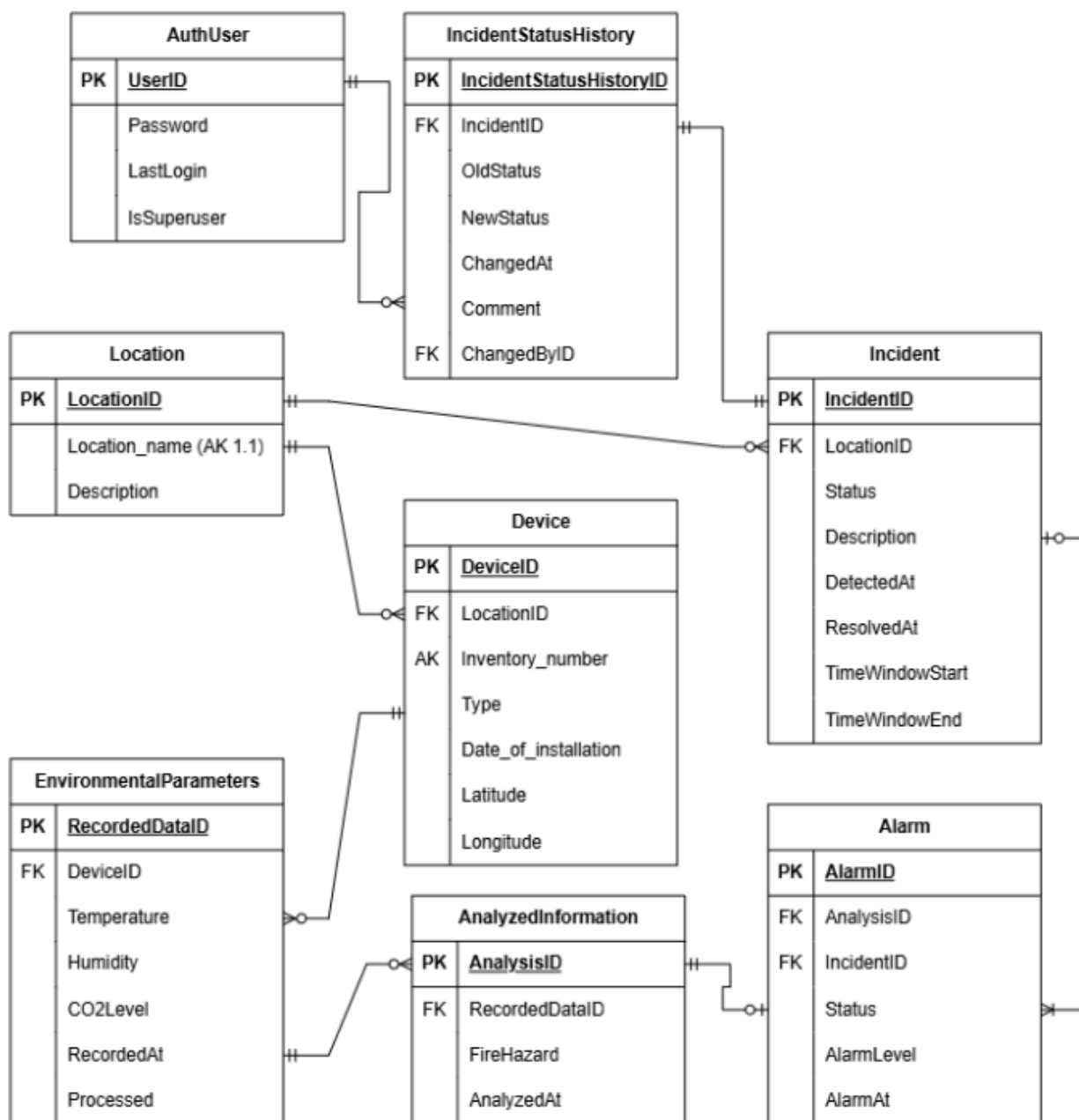


Рисунок 2 – Реляционная модель приложения мониторинга лесных пожаров

Далее для проверки была построена таблица ограничений кардинальностей связей между моделями (таблица 1).

Таблица 1 – Ограничения кардинальных чисел

Зависимости		Кардинальные числа		
Родитель	Ребенок	Тип	Максимум	Минимум

LOCATION	DEVICE	Non identifying	1:N	M-O
DEVICE	ENVIRONMENTAL_PARAMETERS	Non identifying	1:N	M-O
ENVIRONMENTAL PARAMETERS	ANALYZED_INFORMATION	Non identifying	1:N	M-O
ANALYZED INFORMATION	ALARM	Non identifying	1:N	M-O
INCIDENT	ALARM	Identifying	1:N	M-O
INCIDENT	INCIDENT_STATUS_HISTORY	Identifying	1:N	M-O
ALARM	INCIDENT	Identifying	N:1	O-M

Следом рассмотрим, какие действия требуются при вставке, изменении, или удалении того или иного элемента для каждой из таблиц (таблицы 2-9).

Таблица 2 – Действия над таблицами LOCATION и DEVICE 1:N M-O

Операция	Действие над LOCATION	Действие над DEVICE
Вставка	–	подбор существующей записи-родителя
Изменение	запрет	запрет
Удаление	каскадное удаление	–

Таблица 3 – Действия над таблицами DEVICE и ENVIRONMENTAL_PARAMETERS 1:N M-O

Операция	Действие над GEOCOORDS	Действие над ENVIRONMENTAL_PARAMETERS
Вставка	–	подбор родителя
Изменение	запрет	запрет

Удаление	каскадное удаление	–
----------	--------------------	---

Таблица 4 – Действия над таблицами ENVIRONMENTAL_PARAMETERS и ANALYZED_INFORMATION 1:N M-O

Операция	Действие над ENVIRONMENTAL_PARAMETERS	Действие над ANALYZED_INFORMATION
Вставка	–	подбор родителя
Изменение	запрет	запрет
Удаление	каскадное удаление	–

Таблица 5 – Действия над таблицами ANALYZED_INFORMATION и ALARM 1:N M-O

Операция	Действие над ANALYZED_INFORMATION	Действие над ALARM
Вставка	–	подбор родителя
Изменение	запрет	запрет
Удаление	запрет (PROTECT, необходимо предварительно удалить/переназначить alarms)	–

Таблица 6 – Действия над таблицами INCIDENT и ALARM 1:N O-M

Операция	Действие над INCIDENT	Действие над ALARM
Вставка	– (может быть NULL)	родитель указывается опционально
Изменение	запрет	запрет
Удаление	SET NULL — инцидент удаляется, поле incident_id в alarm обнуляется	–

Таблица 7 – Действия над таблицами INCIDENT и INCIDENT_STATUS_HISTORY 1:N M-O

Операция	Действие над INCIDENT	Действие над INCIDENT_STATUS_HISTORY
Вставка	–	создаётся автоматически при первом сохранении инцидента
Изменение	запрет	запрет
Удаление	каскадное удаление	–

Таблица 8 – Действия над таблицами LOCATION и INCIDENT 1:N M-O

Операция	Действие над LOCATION	Действие над INCIDENT
Вставка	–	родитель указывается опционально
Изменение	запрет	запрет
Удаление	запрет (PROTECT, если есть связанные инциденты)	–

После этого для каждого объекта были построены таблицы со следующими полями: имя атрибута, тип данных, тип ключа, статус нулевого значения, замечания. Рассмотрим каждую из них.

Таблица 9 – Атрибуты объекта LOCATION

Название	Тип	Ключ	NULL-статус	Замечания
LocationID	Int	Primary Key	NOT NULL	Surrogate Key IDENTITY (1, 1)
LocationName	Varchar (255)	Alternative Key	NOT NULL	Unique (AK1.1)
Description	Text	—	NULL	

Таблица 10 – Атрибуты объекта DEVICE

Название	Тип	Ключ	NULL-статус	Замечания
DeviceID	Int	Primary Key	NOT NULL	Surrogate Key
LocationID	Int	Foreign Key	NOT NULL	Location
InventoryNumber	Varchar(50)	Alternative Key	NOT NULL	UNIQUE (AK1.1)
Type	Varchar(50)	—	NULL	—
DateOfInstallation	Date	—	NULL	—
Latitude	Decimal(10, 6)	—	NULL	Широта
Longitude	Decimal(10, 6)	—	NULL	Долгота

Таблица 11 – Атрибуты объекта ENVIRONMENTAL_PARAMETERS

Название	Тип	Ключ	NULL-статус	Замечания
EnvParamID	Int	Primary Key	NOT NULL	Surrogate Key
DeviceID	Int	Foreign Key	NOT NULL	Device
Temperature	Decimal(5, 2)	—	NULL	CHECK -50...200 °C
Humidity	Decimal(5, 2)	—	NULL	CHECK 0...100 %
CO2_Level	Decimal(10, 2)	—	NULL	CHECK ≥ 0 ppm
RecordedAt	Datetime	—	NOT NULL	default = NOW()
Processed	Boolean	—	NOT NULL	default = FALSE

Таблица 12 – Атрибуты объекта ANALYZED_INFORMATION

Название	Тип	Ключ	NULL-статус	Замечания
			с	

AnalysisID	Int	Primary Key	NOT NULL	Surrogate Key
EnvParamID	Int	Foreign Key	NOT NULL	EnvironmentalP arameters
FireHazard	Decimal(5, 2)	—	NULL	% опасности 0–100
AnalyzedAt	Datetime	—	NOT NULL	AUTO_TIMES TAMP

Таблица 13 – Атрибуты объекта INCIDENT

Название	Тип	Ключ	NULL-стату с	Замечания
IncidentID	Int	Primary Key	NOT NULL	Surrogate Key
LocationID	Int	Foreign Key	NULL	Location
TimeWindowS tart	Datetime	—	NOT NULL	—
TimeWindowE nd	Datetime	—	NULL	—
DetectedAt	Datetime	—	NOT NULL	AUTO_TIMES TAMP
ResolvedAt	Datetime	—	NULL	—
Status	Varchar(20)	—	NOT NULL	open / investigation / closed
Description	Text	—	NULL	—

Таблица 14 – Атрибуты объекта ALARM

Название	Тип	Ключ	NULL-статус	Замечания
AlarmID	Int	Primary Key	NOT NULL	Surrogate Key
AnalysisID	Int	Foreign Key	NOT NULL	AnalyzedInfor mation; ON DELETE

				PROTECT
IncidentID	Int	Foreign Key	NULL	Incident, ON DELETE SET NULL
AlarmLevel	Varchar(20)		NOT NULL	low / medium / high / critical
Status	Varchar(20)	—	NOT NULL	active / acknowledged / resolved (по умолчанию active)
AlarmAt	Datetime	—	NOT NULL	Время возникновения тревоги

Таблица 15 – Атрибуты объекта INCIDENT_STATUS_HISTORY

Название	Тип	Ключ	NULL-статус	Замечания
IncidentStatus HistoryID	Int	Primary Key	NOT NULL	Surrogate Key
IncidentID	Int	Foreign Key	NOT NULL	Incident
OldStatus	Varchar(20)	—	NULL	—
NewStatus	Varchar (20)	—	NOT NULL	—
ChangedAt	Datetime	—	NOT NULL	AUTO_TIME STAMP
ChangedBy	Int	Foreign Key	NULL	Auth_User
Comment	Text	—	NULL	—

Таблица 17 – Атрибуты объекта AUTH_USER

Название	Тип	Ключ	NULL-статус	Замечания
UserID	Int	Primary Key	NOT NULL	Surrogate Key
Password	Varchar(20)	—	NOT NULL	—

LastLogin	Datetime	—	NOT NULL	—
IsSuperuser	Boolean	—	NOT NULL	—

3. Выбор технологий и инструментов

В процессе проектирования веб-ориентированной системы мониторинга лесных пожаров ключевым этапом являлся выбор оптимального набора технологий для управления данными. Этот выбор обоснован требованиями к производительности, масштабируемости и функциональной полноте системы, а также необходимостью обеспечения надежности в условиях обработки гетерогенных данных в режиме реального времени.

Для реализации системы мониторинга лесных пожаров были выделены следующие критические требования:

- надежное хранение структурированных данных, включая:
 - геокоординаты устройств (широта, долгота);
 - исторические показатели телеметрии (температура, влажность, уровень CO₂);
 - журнал инцидентов и тревог.
- обработка данных в реальном времени с поддержкой периодических задач (генерация данных, анализ уровня пожарной опасности);
- масштабируемость для адаптации к росту числа подключенных устройств и увеличению объема поступающих данных.

Чтобы обеспечить надежную работу веб-приложения был выбран такой технологический стек:

- PostgreSQL — выбрана как основная СУБД:
 - поддержка геопространственных данных через расширение PostGIS (в перспективе интеграции), что позволяет выполнять пространственные запросы (например, поиск устройств в зоне пожара);
 - возможность сложных аналитических запросов, включая агрегацию временных рядов (средние значения параметров за период, определение пиковых уровней CO₂);

- полная интеграция с Django ORM, обеспечивающая декларативное описание моделей данных, автоматическую генерацию миграций и минимизацию ручного написания SQL-кода;
- транзакционная целостность (ACID), гарантирующая корректность данных при параллельной записи показаний от множества устройств.
- Redis — дополняет PostgreSQL, являясь компонентом для обработки временных данных:
 - оркестрация асинхронных задач через Celery (например, периодическая генерация тестовых данных, пакетный анализ уровня опасности);
 - кэширование данных дашборда (KPI, графики последних тревог), что снижает нагрузку на PostgreSQL и ускоряет отклик интерфейса.
- DjangoORM:
 - объектно-ориентированное представление данных, упрощающее манипуляции с сущностями (устройства, местности, инциденты);
 - автоматическая оптимизация запросов через методы `select_related()` и `prefetch_related()`, минимизирующие проблему N+1;
 - гибкая интеграция с REST API (DRF), обеспечивающая сериализацию данных для клиентских приложений.

В контексте обработки больших объемов данных в реальном времени рассматривалась СУБД Realm, демонстрирующая высокую производительность при операциях вставки и чтения. Однако для данного проекта выбор был сделан в пользу PostgreSQL по следующим причинам:

- необходимость сложных JOIN-запросов при анализе взаимосвязей между устройствами, инцидентами и телеметрией;
- требование к транзакционной целостности, критичное для систем мониторинга.

Технологический стек (PostgreSQL, Redis, Django ORM) обеспечивает баланс между производительностью, надежностью и скоростью разработки.

Использование PostgreSQL в качестве основной СУБД позволяет масштабировать систему за счет секционирования таблиц и репликации, а интеграция с Redis решает задачи обработки данных в реальном времени. Дальнейшее развитие системы может включать внедрение PostGIS для расширенного геопространственного анализа и оптимизацию индексов для работы с Big Data.

4. Разработка схемы базы данных

В качестве серверной СУБД для курсового проекта выбран PostgreSQL 16, а доступ к данным реализован через ORM-фреймворк Django 5.2. Такой стек обеспечивает строгую ссылочную целостность, поддержку выразительного SQL-диалекта, расширяемость (TimescaleDB, JSON, индексы) и легко интегрируется с асинхронными задачами Celery.

На концептуальном уровне предметная область описывается шестью сущностями: Location (площадка установки датчиков), Device (конкретный измерительный узел), EnvironmentalParameters (сырые показания датчика), AnalyzedInformation (результаты расчёта индекса пожарной опасности), Alarm (тревожное событие) и Incident (совокупность взаимосвязанных тревог в пределах одной местности и временного окна). Каждая сущность материализована в виде модели Django, а их взаимные связи заданы внешними ключами. Логическая схема была сначала построена в виде ER-диаграммы, где для каждой связи указаны минимальные и максимальные кардинальные значения, а затем преобразована в реляционную, соблюдая 5-ю нормальную форму. Единственным осознанным исключением является булево поле `processed` в таблице `EnvironmentalParameters`: оно дублирует факт наличия соответствующей строки в `AnalyzedInformation`, но радикально ускоряет выборку не-обработанных данных; это пример допустимой денормализации, введённой ради производительности.

Для сохранения предметной целостности в самой базе заданы три `check-constraints`, ограничивающие диапазоны температуры, влажности и содержания CO_2 ; наличие таких ограничений предотвращает занесение физических «невозможных» значений независимо от приложения. Ссылочная целостность поддерживается внешними ключами с различной политикой удаления: для `EnvironmentalParameters` \rightarrow `AnalyzedInformation` применяется `CASCADE`, что гарантирует автоматическое удаление результатов анализа при очистке старой телеметрии, тогда как `AnalyzedInformation` \rightarrow `Alarm` работает через `PROTECT`, исключая случайное «обрушение» тревожных записей.

Существенное внимание уделено индексации. Для таблицы с телеметрией создан композитный индекс (`device_id`, `recorded_at`), обеспечивающий быстрый поиск последних измерений любого устройства, а также частичный индекс по полю `recorded_at` только для строк, где `processed = false`. Такой индекс позволяет Celery-задаче анализа находить «живой хвост» данных без полного просмотра миллионной таблицы. Дополнительный индекс по (`status`, `alarm_at`) в таблице `Alarm` ускоряет построение дашборд-графиков и фильтр активных тревог.

Бизнес-правила, выходящие за рамки декларативного SQL, реализованы на уровне приложения. После сохранения записи `AnalyzedInformation` с опасностью $\geq 50\%$ автоматически срабатывает Django-сигнал `create_alarm_on_hazard`, формирующий объект `Alarm`. В свою очередь, метод `trigger_alarm` у модели `Alarm` либо привязывает тревогу к ещё открытому `Incident`, либо создаёт новый; таким образом поддерживается сквозная бизнес-логика «показания → анализ → тревога → инцидент» без участия пользователя.

Фоновая обработка организована через Celery: задача `generate_random_data` раз в N секунд/минут формирует тестовые показания, `calculate_hazard_batch` рассчитывает индекс опасности пакетами по 1000 строк, а `purge_old_env` ежедневно удаляет устаревшую необработанную телеметрию.

5. Разработка API для взаимодействия с базой данных

Взаимодействие клиентской части с данными реализовано через REST-API на Django REST Framework. Каждая сущность (Location, Device, Alarm, Incident и др.) представлена отдельным ресурсом с доступом через стандартные HTTP-методы. Большинство эндпоинтов (EnvironmentalParameters, AnalyzedInformation) имеют режим read-only, так как данные датчиков автоматически генерируются Celery-задачами. Оператору доступны просмотр карты устройств, активных тревог и статуса инцидентов.

Данные EnvironmentalParameters создаются фоновой задачей generate_random_data, поэтому ручное добавление через API заблокировано. Для изменения состояния инцидентов (например, закрытия) используется IncidentViewSet с кастомной логикой в perform_update(), где статус меняется через метод change_status(), гарантирующий атомарность операций через механизмы Django.

Связь Location → Device реализована через эндпоинты /api/locations/ и /api/devices/. Запрос /api/devices/?has_coords=1 возвращает устройства с заполненными координатами. Для фильтрации используется django-filters: DeviceFilter поддерживает параметр has_coords. AlarmFilter позволяет фильтровать тревоги по alarm_level, status, временным диапазонам, используя индексы (status, alarm_at) и частичный индекс env_processed_false_idx для ускорения выборок.

Внешние ключи настроены на каскадное удаление: AnalyzedInformation.recorded_data → EnvironmentalParameters (CASCADE). Alarm.analysis → AnalyzedInformation (PROTECT для предотвращения случайного удаления). Это исключает «висячие» данные при очистке телеметрии.

Индексы PostgreSQL ускоряют выборки: Составной индекс env_dev_rec_idx (device, recorded_at). Частичный индекс для необработанных данных (processed=False). Автодокументирование через drf-spectacular

(эндпоинты `/api/docs/`, `/api/redoc/`). Добавление новых ресурсов (например, архива графиков) требует только создания сериализатора и `ViewSet`.

Сочетание DRF и транзакционной поддержки Django позволило создать API, которое

- обеспечивает атомарность всех изменений состояния инцидентов;
- отдаёт сводную информацию о тревогах и устройствах за миллисекунды благодаря целевым индексам и отбору только необходимых полей;
- автоматически поддерживает целостность связей за счёт каскадных правил в PostgreSQL;
- остаётся расширяемым — добавление нового ресурса, например архива графиков, сводится к созданию сериализатора и регистрации `ViewSet`, после чего спецификация OpenAPI обновляется автоматически.

Таким образом, разработанное REST-API полностью удовлетворяет задачам курсового проекта: клиент без прямого доступа к базе данных может строить карту устройств, следить за журналом тревог, подтверждать и закрывать инциденты, сохраняя при этом все гарантии согласованности и надёжности данных.

6. Тестирование и отладка

После того как основная логика была реализована, база данных и связанное с ней приложение прошли простое, но достаточное для учебного проекта испытание. Сначала я зашла в администраторский интерфейс Django и вручную создал новый объект Location, несколько Device с координатами и одну строку EnvironmentalParameters с произвольными значениями температуры, влажности и CO₂. Запись корректно сохранилась, что подтвердило работоспособность операций «создать» и «прочитать».

Далее были подняты Celery worker и beat. Я дождалась очередного запуска фоновой задачи генерации телеметрии: в таблице EnvironmentalParameters появились новые строки, а буквально через несколько секунд рядом появились расчёты в AnalyzedInformation. Это означало, что цепочка «показание → анализ» отрабатывает без ошибок. Я открыла страницу `/dashboard/incidents/` и убедилась, что свежие тревоги автоматически склеиваются в инциденты и отображаются оператору.

API проверялось самым прямым способом — через `curl`. Запрос `curl http://127.0.0.1:8000/api/devices/?has_coords=1` вернул корректный JSON-список устройств. Тем самым чтение данных отрабатывает правильно.

В процессе проверки я держала открытыми консоли с логами Django и Celery. Когда появлялись ошибки (например, из-за опечатки в имени поля `co2_level`), трасс-бек сразу был виден, код исправлялся, процессы перезапускались, и повторный прогон проходил уже чисто. Таким же образом отлавливались ошибки целостности при удалении старых записей: после изменения внешних ключей на `CASCADE` сообщения о `ProtectedError` исчезли.

После нескольких итераций «создание данных → просмотр лога → исправление» приложение стабильно выполняло все требуемые операции: данные создавались, обрабатывались, тревоги формировались, инциденты закрывались, а в журналах не оставалось сообщений об ошибках. Для целей курсового проекта такой объём ручного функционального тестирования

оказался достаточным, чтобы подтвердить корректность работы базы данных и связанной с ней бизнес-логики.

7. Интеграция базы данных в пользовательское приложение

Когда база данных и фоновые задачи уже работали стабильно, они были подключены к самой пользовательской части сайта — небольшому веб-приложению на Django. Всё свелось к трём простым шагам:

- Переход на REST-запросы — вместо того чтобы тянуть записи устройств и тревог прямо из моделей Django, страницы теперь запрашивают данные через готовые эндпоинты вида `/api/devices/?has_coords=1`, `/api/alarms/?status=active`. Сервер отвечает чистым JSON, а браузер рисует интерфейс сам. Это упростило шаблоны и убрало лишнюю нагрузку с сервера.
- Обновление карты — карта Яндекс загружает координаты устройств тем же запросом `/api/devices`. Благодаря индексу по полям широты и долготы ответ приходит очень быстро, и маркеры появляются почти мгновенно—even при сотнях датчиков.

Главная страница отображает оперативную сводку: В верхней части — карточки с общим количеством устройств, активных тревог и открытых инцидентов. По нажатию на карточку «Устройства» можно выполнять операции `create`, `read`, `update`, `delete`, обращаясь непосредственно к базе данных. Аналогичный функционал (`update`) предусмотрен внутри карточки «Инциденты».

11
Устройств

2476
Активных тревог

107
Открытых инцидентов

Рисунок 3 – Карточки сводки























Устройства (11)							
+ Новое устройство							
ID	Инв. №	Тип	Широта	Долгота	Локация	Установлено	
14	78788мрлмл	лщощ	64.000000	82.000000	Крапивинский муниципальный округ	20.05.2025	 
4	DEV-001	Датчик температуры	56.139700	90.863020	Ястребовский сельсовет	20.10.2023	 
5	DEV-002	Датчик влажности	56.194360	90.880870	Ястребовский сельсовет	21.10.2023	 
6	DEV-003	Датчик CO2	56.180900	90.416220	Ключинский сельсовет	22.10.2023	 
7	DEV-004	Датчик давления	56.175840	90.420000	Ключинский сельсовет	23.10.2023	 
8	DEV-005	Датчик ветра	56.060610	85.379500	Яшкинский муниципальный округ	24.10.2023	 
9	DEV-006	Комбинированный датчик	55.901560	86.269600	Яйский муниципальный округ	25.10.2023	 
10	DEV-007	Датчик освещенности	55.781490	86.892800	Ижморский муниципальный округ	26.10.2023	 
11	DEV-008	Датчик шума	55.541080	87.347730	Чебулинский муниципальный округ	27.10.2023	 
12	DEV-009	Датчик дождя	55.236760	87.748190	Тисульский муниципальный округ	28.10.2023	 
13	DEV-010	Пожарный датчик	54.890210	87.664540	Крапивинский муниципальный округ	29.10.2023	 

Рисунок 4 – Раскрытая карточка для приборов с возможностями crud-операций

Устройства (11)											
+ Новое устройство											
ID	Инв. №	Тип	Широта	Долгота	Локация	Установлено					
Inventory number:		Type:	Latitude:		Longitude:	Location:					
<input type="text"/>		<input type="text"/>	<input type="text"/>		<input type="text"/>	<input type="text"/>					
Date of installation:											
<input type="text"/>											
14	78788мрлмл	лщощ	64.000000	82.000000	Крапивинский муниципальный округ	<div> <div>-----</div> <div> <div>Участок молодняка</div> <div>Старый лес</div> <div>Крапивинский муниципальный округ</div> <div>Тисульский муниципальный округ</div> <div>Чебулинский муниципальный округ</div> <div>Ижморский муниципальный округ</div> <div>Яйский муниципальный округ</div> <div>Яшкинский муниципальный округ</div> <div>Ключинский сельсовет</div> <div>Ястребовский сельсовет</div> </div> </div>					
4	DEV-001	Датчик температуры	56.139700	90.863020	Ястребовский сельсовет						
5	DEV-002	Датчик влажности	56.194360	90.880870	Ястребовский сельсовет						
6	DEV-003	Датчик CO2	56.180900	90.416220	Ключинский сельсовет						
7	DEV-004	Датчик давления	56.175840	90.420000	Ключинский сельсовет						

Рисунок 5 – Функционал добавления нового устройства

Ниже — линейный график "Alarm / day" за последние 7 дней. Клик по точке открывает модальное окно с событиями за выбранный день.

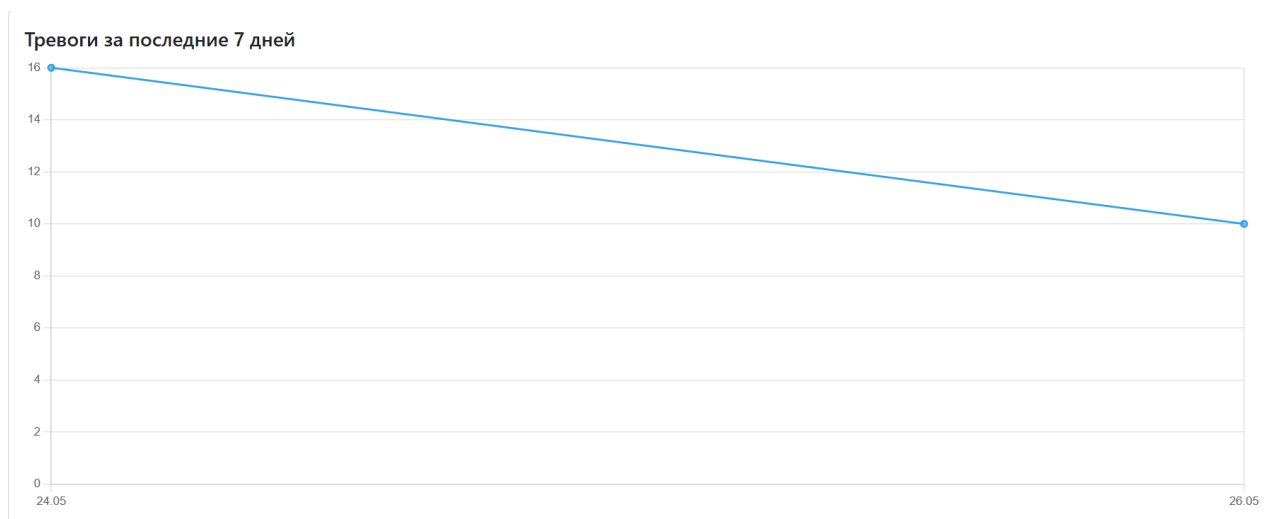


Рисунок 6 – График тревог за последние 7 дней

Тревоги за 2025-05-26

ID	Устройство	Уровень	Время
3239	78788мрлмл	ВЫСОКИЙ	16:59
3238	DEV-002	ВЫСОКИЙ	16:59
3237	DEV-003	ВЫСОКИЙ	16:59
3236	DEV-006	ВЫСОКИЙ	16:59
3235	DEV-010	СРЕДНИЙ	16:58
3234	78788мрлмл	ВЫСОКИЙ	16:58
3233	DEV-004	ВЫСОКИЙ	16:58
3232	DEV-007	ВЫСОКИЙ	16:58
3231	DEV-008	СРЕДНИЙ	16:58
3230	DEV-010	ВЫСОКИЙ	16:58

Рисунок 7 – Модальное окно, отображающее список тревог в конкретный день

Под графиком — таблица "Critical alarms" (последние 10 критических тревог) и кнопка "CSV за 7 дней". При нажатии формируется и скачивается отчет.

Последние критические тревоги				<div> <div>Все критические тревоги</div> <div>CSV за 7 дней</div> </div>
#	Устройство	Уровень	Время	
3239	78788мрлмл	ВЫСОКИЙ	26.05 16:59	
3238	DEV-002	ВЫСОКИЙ	26.05 16:59	
3237	DEV-003	ВЫСОКИЙ	26.05 16:59	
3236	DEV-006	ВЫСОКИЙ	26.05 16:59	
3234	78788мрлмл	ВЫСОКИЙ	26.05 16:58	
3233	DEV-004	ВЫСОКИЙ	26.05 16:58	
3232	DEV-007	ВЫСОКИЙ	26.05 16:58	
3230	DEV-010	ВЫСОКИЙ	26.05 16:58	
3226	DEV-001	ВЫСОКИЙ	24.05 02:19	
3225	DEV-002	ВЫСОКИЙ	24.05 02:19	

[Перейти на карту устройств](#)

Рисунок 6 – Список критических тревог с возможностью выгрузки данных за последние 7 дней

Карта устройств

Фильтр тревог: ☒ Critical ☒ High ☒ Medium ☒ Low

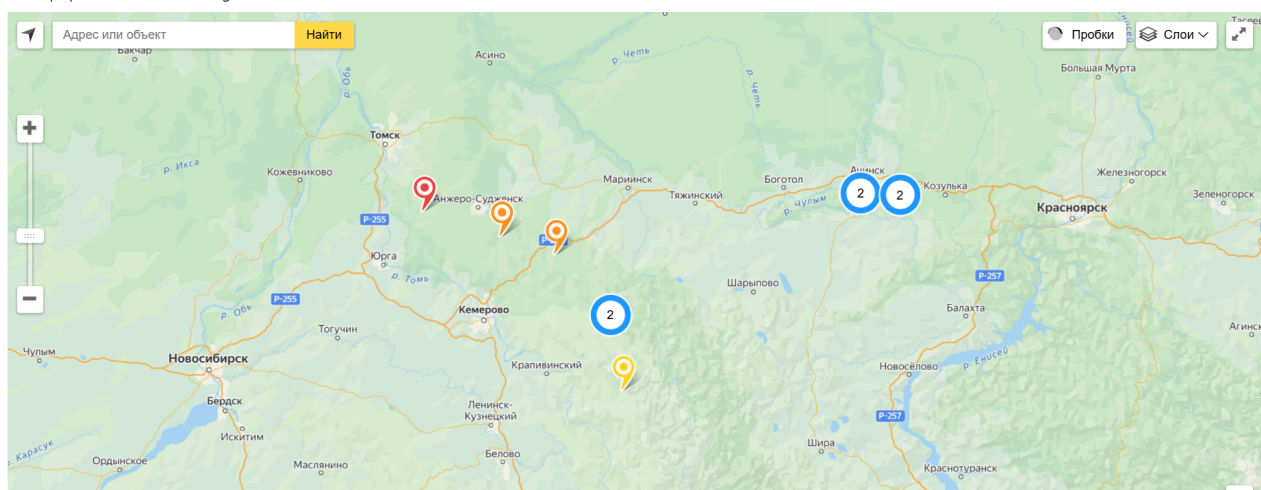


Рисунок 8 – Яндекс-карта устройств расположенных согласно своим координатам

8. Документация и поддержка

Документация к базе данных оформлялась прямо в репозитории, чтобы любому будущему разработчику не приходилось разбираться вслепую. Каждая модель Django снабжена коротким doc-string'ом, где указано назначение сущности, список её полей и заметки о внешних ключах. Над ключевыми методами – такими как расчёт индекса пожарной опасности и автоматическое создание тревоги – находятся пояснения о том, что именно происходит и почему сделано именно так.

В каталоге docs/ лежит ER-диаграмма, экспортированная из Django-плагина Graph Models: картинка сразу показывает, как связаны таблицы. Рядом – README.md с пошаговой инструкцией «как поднять проект», образцом файла .env и примерами curl-запросов к API.

Логи ведутся через стандартный механизм Django: при ошибке в консоли сразу появляется трасс-бек, а при штатной работе в файле видны отметки о созданных тревогах и закрытых инцидентах. Такой минимум комментариев, doc-string'ов и вспомогательных файлов оказался достаточным: чтобы понять проект, нужно открыть код и прочитать пояснения прямо там, а для запуска достаточно следовать инструкциям из README.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была разработана рабочая система мониторинга пожарной обстановки на базе PostgreSQL и Django:

1. Анализ требований: определены сущности (Location, Device, EnvironmentalParameters и др.), выделены ключевые ограничения по целостности и скорости работы.
2. Выбор СУБД: выбран PostgreSQL – он обеспечивает строгие внешние ключи, check-ограничения и удобен в связке с Django ORM.
3. Проектирование и реализация схемы: построена ER-модель, преобразованная в реляционные таблицы; добавлены индексы для быстрых выборок и контрольные ограничители диапазонов значений.
4. Интеграция с веб-приложением: создан REST-API (DRF), через который фронтенд загружает список устройств и активных тревог; карта Яндекс отображает координаты датчиков в реальном времени.
5. Фоновые задачи: Celery-worker генерирует тестовые показания, вычисляет уровень опасности и автоматически формирует тревоги и инциденты.
6. Тестирование и отладка. Ручные CRUD-проверки в админке, запросы к API через curl и просмотр логов подтвердили корректность всех операций; после исправлений система работает стабильно.

Разработка и интеграция базы данных в веб-приложение мониторинга пожарной обстановки стала важным этапом проекта, обеспечивающим надёжное хранение, обработку и отображение данных телеметрии. В качестве основной СУБД была выбрана PostgreSQL, поскольку она обеспечивает высокую надёжность, поддержку внешних ключей, ограничений целостности и

индексирования, что особенно важно при работе с потоками данных от устройств в реальном времени.

Использование фреймворков Django и Celery позволило организовать систему, в которой данные автоматически генерируются, анализируются и приводят к формированию тревог и инцидентов. Интерфейс на основе карты (Яндекс) и API (Django REST Framework) позволил эффективно отобразить актуальное состояние системы и обеспечить доступ операторов к ключевой информации.

Полученные результаты подтверждают, что реализованная архитектура базы данных подходит для задач оперативного мониторинга и может быть адаптирована под различные сценарии — как в учебных, так и в практических проектах в области информационных технологий.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Документация Django [Электронный курс — URL: [Django documentation | Django documentation | Django](#)] (Дата обращения 20.03.2025)
2. Документация Celery [Электронный курс — URL: [Introduction to Celery — Celery 5.5.2 documentation](#)] (Дата обращения 12.04.2025)
3. Документация PostgreSQL [Электронный курс —] URL: [PostgreSQL: Documentation](#) (Дата обращения 20.03.2025)
4. Документация Python [Электронный курс —] URL: [Our Documentation | Python.org](#) (Дата обращения 20.03.2025)

ЛИСТИНГ ПРОГРАММ

Листинг 1 – config/__init__.py

```
from .celery import app as celery_app
__all__ = ("celery_app",)
```

Листинг 2 – config/celery.py

```
import os
from celery import Celery
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "config.settings")
app = Celery("config")
app.config_from_object("django.conf:settings", namespace="CELERY")
app.autodiscover_tasks()
```

Листинг 3 – config/settings.py

```
from pathlib import Path
import environ
from celery.schedules import crontab
BASE_DIR = Path(__file__).resolve().parent.parent
env = environ.Env(
    DEBUG=(bool, False),
)
environ.Env.read_env(BASE_DIR / ".env")
SECRET_KEY = env("DJANGO_SECRET_KEY")
DEBUG = env.bool("DEBUG")
ALLOWED_HOSTS = env.list("ALLOWED_HOSTS")

if not DEBUG:
    SESSION_COOKIE_SECURE = True
    CSRF_COOKIE_SECURE = True
    SECURE_HSTS_SECONDS = 31536000
    SECURE_HSTS_INCLUDE_SUBDOMAINS = True
    SECURE_HSTS_PRELOAD = True
    SECURE_BROWSER_XSS_FILTER = True
```

```

    SECURE_CONTENT_TYPE_NOSNIFF = True
INSTALLED_APPS = [
    "rest_framework_simplejwt",
    "channels",
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "monitoring",
    "dashboard"
    "rest_framework",
    "drf_spectacular",
    "django_filters",
    "django_celery_results",
    "django_celery_beat",
]
MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
ROOT_URLCONF = "config.urls"
#WSGI_APPLICATION = "config.wsgi.application"
ASGI_APPLICATION = "config.asgi.application"
TEMPLATES = [
    {
        "BACKEND":
        "django.template.backends.django.DjangoTemplates",

```

```

"DIRS": [BASE_DIR / "templates"],
"APP_DIRS": True,
"OPTIONS": {
    "context_processors": [
        "django.template.context_processors.debug",
        "django.template.context_processors.request",
        "django.contrib.auth.context_processors.auth",

"django.contrib.messages.context_processors.messages",

    ],
},
],
STATIC_URL = "static/"
STATICFILES_DIRS = [BASE_DIR / "static"]
DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": env("POSTGRES_DB"),
        "USER": env("POSTGRES_USER"),
        "PASSWORD": env("POSTGRES_PASSWORD"),
        "HOST": env("POSTGRES_HOST"),
        "PORT": env("POSTGRES_PORT"),
    }
}
LANGUAGE_CODE = "en-us"
TIME_ZONE = "Europe/Moscow"
USE_I18N = True
USE_TZ = True
REST_FRAMEWORK = {
    "DEFAULT_SCHEMA_CLASS": "drf_spectacular.openapi.AutoSchema",
    "DEFAULT_AUTHENTICATION_CLASSES": [

```

```

"rest_framework_simplejwt.authentication.JWTAuthentication",
    "rest_framework.authentication.SessionAuthentication",
],
"DEFAULT_PERMISSION_CLASSES": [
    "rest_framework.permissions.IsAuthenticatedOrReadOnly",
],
"DEFAULT_PAGINATION_CLASS":
"rest_framework.pagination.PageNumberPagination",
"PAGE_SIZE": 100,
"DEFAULT_FILTER_BACKENDS":
["django_filters.rest_framework.DjangoFilterBackend"],
}

```

```

REDIS_URL = env("REDIS_URL")
CELERY_BROKER_URL = f"{REDIS_URL}/0"
CELERY_RESULT_BACKEND = f"{REDIS_URL}/1"
CELERY_TIMEZONE = TIME_ZONE
CELERY_IMPORTS = ("monitoring.tasks",)

```

```

CELERY_BEAT_SCHEDULE = {
    "generate_env_every_10min": {
        "task": "monitoring.tasks.generate_random_data",
        "schedule": crontab(minute="*/10"),
    },
    "purge_old_env_daily": {
        "task": "monitoring.tasks.purge_old_env",
        "schedule": crontab(hour=3, minute=0),
    },
}

```

```

CHANNEL_LAYERS = {

```

```

    "default": {
        "BACKEND": "channels_redis.core.RedisChannelLayer",
        "CONFIG": {"hosts": [REDIS_URL]},
    }
}

```

Листинг 4 – config/urls.py

```

from django.contrib import admin
from django.urls import path, include
from drf_spectacular.views import (
    SpectacularAPIView,
    SpectacularSwaggerView,
    SpectacularRedocView
)
from rest_framework_simplejwt.views import (
    TokenObtainPairView, TokenRefreshView,
)
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('monitoring.urls')),
        path("api/token/",      TokenObtainPairView.as_view(),
name="token_obtain_pair"),
        path("api/token/refresh/",  TokenRefreshView.as_view(),
name="token_refresh"),
        path('api/schema/',      SpectacularAPIView.as_view(),
name='schema'),
                                path('api/docs/',
SpectacularSwaggerView.as_view(url_name='schema'),
name='swagger-ui'),
                                path('api/redoc/',
SpectacularRedocView.as_view(url_name='schema'), name='redoc'),
        path("dashboard/", include("dashboard.urls")),
]

```

Листинг 5 – Класс dashboard/apps.py

```

from django.apps import AppConfig
class DashboardConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'dashboard'

```

Листинг 6 – Класс dashboard/forms.py

```

from django import forms
from monitoring.models import Device, Incident

class DeviceForm(forms.ModelForm):
    class Meta:
        model = Device
        fields = [
            "inventory_number",
            "type",
            "latitude",
            "longitude",
            "location",
            "date_of_installation",
        ]
        widgets = {
            "inventory_number": forms.TextInput(
                attrs={"class": "form-control form-control-sm"}
            ),
            "type": forms.TextInput(
                attrs={"class": "form-control form-control-sm"}
            ),
            "latitude": forms.NumberInput(
                attrs={"class": "form-control form-control-sm",
"step": "0.000001"}
            ),
            "longitude": forms.NumberInput(

```

```

        attrs={"class": "form-control form-control-sm",
"step": "0.000001"}
    ),
    "location": forms.Select(
        attrs={"class": "form-select form-select-sm"}
    ),
    "date_of_installation": forms.DateInput(
        attrs={"class": "form-control form-control-sm",
"type": "date"}
    ),
}

```

```

class IncidentStatusForm(forms.ModelForm):
    class Meta:
        model = Incident
        fields = ["status"]
        widgets = {
            "status": forms.Select(attrs={"class": "form-select
form-select-sm"})
        }

```

Листинг 7 – Класс dashboard/urls.py

```

from django.urls import path
from .views import (
    DashboardHomeView, DeviceMapView,
    DeviceListView, AlarmListView, IncidentListView,
    AlarmCriticalListView, AlarmByDayView
)
from .views import CriticalAlarmCSV
from .views import DeviceInlineEdit, DeviceInlineSave,
DeviceInlineRow, DeviceInlineCreate, DeviceInlineDelete
from django.http import HttpResponse
from .views import (

```



```

        IncidentInlineStatusEdit,    IncidentInlineStatusSave,
IncidentInlineRow,
)

def device_form_cancel(request):
    return HttpResponse("", status=204)

app_name = "dashboard"

urlpatterns = [
    path("", DashboardHomeView.as_view(), name="home"),
    path("map/", DeviceMapView.as_view(), name="device-map"),
        path("devices/", DeviceListView.as_view(),
name="device-list-page"),
        path("alarms/", AlarmListView.as_view(),
name="alarm-list-page"),
        path("incidents/", IncidentListView.as_view(),
name="incident-list-page"),
    path("alarms/critical/", AlarmCriticalListView.as_view(),
        name="critical-alarm-list-page"),
    path(
        "alarms/day/<slug:iso_date>/",
        AlarmByDayView.as_view(),
        name="alarms-by-day",
    ),
    path(
        "alarms/critical/csv/", CriticalAlarmCSV.as_view(),
        name="critical-alarm-csv"),
        path("devices/<int:pk>/edit/", DeviceInlineEdit.as_view(),
name="device-edit"),
        path("devices/<int:pk>/save/", DeviceInlineSave.as_view(),
name="device-save"),
        path("devices/<int:pk>/row/", DeviceInlineRow.as_view(),
name="device-row"),

```

```

        path("devices/new/form/",          DeviceInlineCreate.as_view(),
name="device-new"),
        path("devices/<int:pk>/delete/", DeviceInlineDelete.as_view(),
name="device-delete"),
        path("devices/form/cancel/",      device_form_cancel,
name="device-cancel"),
    path(
        "incidents/<int:pk>/status/edit/",
        IncidentInlineStatusEdit.as_view(),
        name="incident-edit-status",
    ),
    path(
        "incidents/<int:pk>/status/save/",
        IncidentInlineStatusSave.as_view(),
        name="incident-save-status",
    ),
    path(
        "incidents/<int:pk>/row/",
        IncidentInlineRow.as_view(),
        name="incident-row",
    ),
]

```

Листинг 8 – Класс dashboard/views.py

```

from datetime import timedelta
from urllib import request
from django.utils import timezone
from django.db import models
from django.views.generic import TemplateView
from django.views.generic import TemplateView, ListView
from django.core.paginator import Paginator
from datetime import date

```

```

from django.http import HttpResponseRedirect
import csv
from django.utils.timezone import now, timedelta
from django.db.models.functions import TruncDate
from django.views import View
from django.views.generic import UpdateView
from django.http import HttpResponseRedirect
from django.template.loader import render_to_string
from django.views import View, generic
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from monitoring.models import Device, Alarm, Incident
from django.views.generic import DeleteView
from .forms import DeviceForm, IncidentStatusForm
from django.http import HttpResponseRedirect, HttpResponseRedirectBadRequest

```

```

class DashboardHomeView(TemplateView):
    """Главная панель с KPI и графиком тревог."""
    template_name = "dashboard/home.html"

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)

        ctx["device_total"] = Device.objects.count()
        ctx["alarm_active"] = Alarm.objects.filter(status="active").count()
        ctx["incident_open"] = Incident.objects.filter(
            status__in=["open", "investigation"]
        ).count()

        today = timezone.now().date()

```

```

start = today - timedelta(days=6)
rows = (
    Alarm.objects
        .filter(alarm_at__date__gte=start)
        .annotate(d=TruncDate("alarm_at"))
        .values("d")
        .order_by("d")
        .annotate(cnt=models.Count("id"))
)

ctx["chart_labels"] = [r["d"].strftime("%d.%m") for r in
rows]

ctx["chart_data"] = [r["cnt"] for r in rows]

# Последние критические тревоги
ctx["latest_alarms"] = (
    Alarm.objects
        .filter(alarm_level__in=["high", "critical"])
        .select_related("analysis__recorded_data__device")
        .order_by("-alarm_at")[:10]
)

return ctx


class DeviceMapView(TemplateView):
    """Интерактивная карта устройств."""
    template_name = "dashboard/device_map.html"


class DeviceListView(ListView):
    model = Device
    template_name = "dashboard/device_list.html"
    paginate_by = 25
    ordering = "inventory_number"

```

```

class AlarmListView(ListView):
    model = Alarm
    template_name = "dashboard/alarm_list.html"
    paginate_by = 25

    def get_queryset(self):
        return (super().get_queryset()
                .filter(status="active")
                .select_related("analysis__recorded_data__device")
                .order_by("-alarm_at"))

class IncidentListView(ListView):
    model = Incident
    template_name = "dashboard/incident_list.html"
    paginate_by = 25

    def get_queryset(self):
        return (super().get_queryset()
                .filter(status__in=["open", "investigation"])
                .select_related("location")
                .order_by("-detected_at"))

class AlarmCriticalListView(ListView):
    """BCE тревоги уровня critical, независимо от статуса."""
    model = Alarm
    template_name = "dashboard/alarm_critical_list.html"
    paginate_by = 25

    def get_queryset(self):
        return (super()
                .get_queryset()
                .filter(alarm_level="critical"))

```

```

        .select_related("analysis__recorded_data__device")
        .order_by("-alarm_at"))

class AlarmByDayView(ListView):
    """AJAX-страница: все тревоги за указанный день (iso_date =
    YYYY-MM-DD)."""
    template_name = "dashboard/_alarm_day_table.html"
    context_object_name = "alarms"
    paginate_by = 40

    def get_queryset(self):
        iso = self.kwargs["iso_date"]
        picked_date = date.fromisoformat(iso)
        return (Alarm.objects
                .filter(alarm_at__date=picked_date)
                .select_related("analysis__recorded_data__device")
                .order_by("-alarm_at"))

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        ctx["picked_date"] = self.kwargs["iso_date"]
        return ctx

class CriticalAlarmCSV(View):
    """Отдаёт CSV-файл со всеми critical-тревогами за последние 7
    дней."""

    def get(self, request):
        week_ago = now() - timedelta(days=7)
        qs = (Alarm.objects
              .filter(alarm_level="critical",
alarm_at__gte=week_ago)

```

```

.select_related("analysis__recorded_data__device")
                .order_by("-alarm_at"))

    response = HttpResponse(content_type="text/csv")
    response["Content-Disposition"] = (
                                                f'attachment;
filename="critical_alarms_{now():%Y%m%d}.csv"')

    writer = csv.writer(response)
    writer.writerow(["ID", "Устройство", "Время", "Hazard %",
"Статус"])
    for a in qs:
                                                device      =
a.analysis.recorded_data.device.inventory_number
        writer.writerow([
            a.id, device, a.alarm_at.strftime("%Y-%m-%d
%H:%M"),
            a.analysis.fire_hazard, a.status
        ])
    return response

class DeviceInlineEdit(generic.UpdateView):
    model = Device
    fields = ["inventory_number", "latitude", "longitude"]
    template_name = "dashboard/_device_form.html"

class DeviceInlineSave(generic.UpdateView):
    model = Device
    fields = ["inventory_number", "latitude", "longitude"]

    def form_valid(self, form):
        obj = form.save()

```

```

        html = render_to_string("dashboard/_device_row.html",
{"d": obj})
        return HttpResponse(html)

    def form_invalid(self, form):
        return HttpResponse(form.errors.as_ul(), status=400)

class DeviceInlineRow(View):
    def get(self, request, pk):
        d = Device.objects.get(pk=pk)
        return
        HttpResponse(render_to_string("dashboard/_device_row.html", {"d":
d})))

class DeviceInlineCreate(generic.CreateView):
    model = Device
    form_class = DeviceForm
    template_name = "dashboard/_device_form_add.html"

    def form_valid(self, form):
        obj = form.save()
        html = render_to_string(
            "dashboard/_device_row.html",
            {"d": obj},
            request=self.request
        )
        return HttpResponse(html, status=201)

    def form_invalid(self, form):
        html = render_to_string(self.template_name,
                                {"form": form},
                                request=self.request)
        return HttpResponse(html, status=422)

```



```

class DeviceInlineDelete(DeleteView):
    model = Device

    # При реальном DELETE-запросе
    def delete(self, request, *args, **kwargs):
        self.object = self.get_object()
        self.object.delete()
        return HttpResponseRedirect("", status=204)

    # HTMX по умолчанию шлёт POST с _method=DELETE
    def post(self, request, *args, **kwargs):
        return self.delete(request, *args, **kwargs)

class IncidentInlineRow(View):
    def get(self, request, pk):
        inc = Incident.objects.get(pk=pk)
        html = render_to_string("dashboard/_incident_row.html",
{"inc": inc})
        return HttpResponseRedirect(html)

#          —          форма          редактирования


---



class IncidentInlineStatusEdit(generic.UpdateView):
    model = Incident
    form_class = IncidentStatusForm
    template_name = "dashboard/_incident_form_status.html"

#          —          сохранение          статуса


---



class IncidentInlineStatusSave(generic.UpdateView):
    model = Incident
    form_class = IncidentStatusForm
    template_name = "dashboard/_incident_form_status.html"

```

```

def form_valid(self, form):
    obj = self.get_object() # инцидент из базы
    new_status = form.cleaned_data["status"]
    obj.change_status(new_status, user=self.request.user) #
аудит + save
    html = render_to_string(
        "dashboard/_incident_row.html", {"inc": obj},
request=self.request
    )
    return HttpResponse(html)

def form_invalid(self, form):
    html = render_to_string(
        self.template_name, {"form": form},
request=self.request
    )
    return HttpResponseBadRequest(html)

```

Листинг 9 – Конструкция для создания нового объекта Location с заданными параметрами

```

Realm realm = Realm.getDefaultInstance();
realm.executeTransaction(r -> {
    Location location = r.createObject(Location.class, "New
Location");
    location.setDescription("This is a new location");
});
realm.close();

```

Листинг 10 – Способ получения всех объектов Device с типом Station

```

RealmResults<Device> devices = realm.where(Device.class)
    .equalTo("type", "Station")
    .findAll();

```

Листинг 11 – Функция имитации поведения программно-определяемой радиосистемы, записывающая данные в базу данных

```

private void generateData() {
    Sample sample = realm.createObject(Sample.class, -1);

sample.setGeoCoords(realm.where(GeoCoords.class).findFirst());
    sample.setRecordTime(LocalDateTime.now());

    Random r = new Random();
    float time = 107.0f;
    for (float i = 104.8f; i < 107.7f; i+=.1f) {
        Integer x = (int)(i * 1_000_000);
        Float y;
        if (Math.abs(time - i) < .1 * 2) {
            y = -20 + r.nextFloat() * 5;
        } else {
            y = -45 + r.nextFloat() * 5;
        }

        FreqRecord freqRecord =
realm.createObject(FreqRecord.class, -1);
        freqRecord.setSample(sample);
        freqRecord.setFreqValue(x);
        freqRecord.setSignalStrength(y);
    }
}

```

Листинг 12 – Метод, считывающий последний обработанный программно-определяемой радиосистемой пакет, записанный в базу данных

```
private ArrayList<BarEntry> getLastSampleFromBD() {
    GeoCoords geoCoords =
realm.where(GeoCoords.class).findFirst();
    Sample sample = realm
        .where(Sample.class)
        .equalTo("geoCoords", geoCoords.toString())
        .equalTo("recordTime",
            realm
                .where(Sample.class)
                    .equalTo("geoCoords",
geoCoords.toString())
                        .max("recordTime")
                            .intValue()).findFirst());
    ArrayList<BarEntry> data = new ArrayList<>();

    for (FreqRecord freqRecord : sample.getFreqRecords()) {
        data.add(new BarEntry(freqRecord.getFreqValue(),
freqRecord.getSignalStrength()));
    }

    return data;
}
```

Основные сущности системы:

7. локация (location):

- a. location name: название местности;
- b. description: описание местности.

8. экземпляры приборов (device):

- a. inventory number: инвентарный номер;
- b. location name: название локации;
- c. device name: название прибора;
- d. device type: тип прибора;
- e. hemisphere of latitude: полушарие широты (N или S);
- f. degrees of latitude: градусы широты;
- g. hemisphere of longitude: полушарие долготы (E или W);
- h. degrees of longitude: градусы долготы;
- i. installation date: дата установки.

9. параметры окружающей среды (environmental parameters):

- a. parameter name: название параметра;
- b. inventory number: инвентарный номер;
- c. temperature: температура;
- d. humidity: влажность;
- e. wind speed: скорость ветра;
- f. co2 level: уровень co2;
- g. time: время.

10. проанализированная информация (analyzed information):

- a. number of analysis: номер анализа;
- b. parameter name: название параметра;
- c. inventory number: инвентарный номер;

- d. time of analysis: время анализа;
- e. fire hazard: вероятность возгорания.

11. оповещения (alarm):

- a. number of alarm: номер оповещения;
- b. number of analysis: номер анализа;
- c. status: статус;
- d. date: дата;
- e. time: время.

12. инцидент (incident):

- a. number of incident: номер инцидента;
- b. number of alarm: номер оповещения;
- c. status: статус;
- d. description: описание;
- e. time of detection: время обнаружения;
- f. time of resolve: время разрешения инцидента.