

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Пользовательское приложение для автоматизации
задач оператора в составе информационно-управляющей
платформы**

Студент _____ Жуков Александр Владимирович
(подпись, дата) (ФИО полностью)

Руководитель _____ Шибанов С.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от №)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от №)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Жуков Александр Владимирович гр. 21ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Пользовательское приложение для автоматизации задач оператора в составе информационно-управляющей платформы

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Шибанов С.В.

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

1) Назначение: Пользовательское приложение для автоматизации задач оператора в составе информационно-управляющей платформы

2) Среда выполнения: Windows 11

Языковые средства разработки: TypeScript, JavaScript, UML

Инструментальные средства: Node.js, Visual Studio Code, Visual Paradigm 16.3.

Технологии разработки: ООП, сервисно-ориентированный подход.

3) Основные функции разрабатываемого приложения:

а) регистрация и авторизация пользователей;

б) просмотр устройств, событий, локаций и пользователей;

в) редактирование и удаление устройств систем, пользователей, событий;

г) просмотр и редактирование событий в табличном, и графическом представлении;

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и планирование процесса разработки;

2) Анализ требований и постановка задачи на разработку;

3) Разработка;

4) Контроль качества программного обеспечения;

5) Оформление пояснительной записки.

6. Календарный график выполнения работы

п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и планирование процесса разработки	10%	08.04.2025 – 14.04.2025	
2	Анализ требований и постановка задачи на разработку	20%	15.04.2025 – 25.04.2025	
3	Разработка программного обеспечения	30%	26.04.2025 – 12.05.2025	
4	Контроль качества программного обеспечения	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 160 листов, 26 рисунков, 36 таблицы, 23 использованный источник и 7 приложений.

WEB-ПРИЛОЖЕНИЕ, УПРАВЛЕНИЕ УСТРОЙСТВАМИ, ИНФОРМАЦИОННО-УПРАВЛЯЮЩАЯ ПЛАТФОРМА, FRONTEND, УПРАВЛЕНИЕ СОБЫТИМИ, REACT JS, UML.

Объект исследования – пользовательское приложение для автоматизации задач оператора в составе информационно-управляющей платформы.

Цель выпускной квалификационной работы – разработать пользовательское приложение для автоматизации задач оператора в составе информационно-управляющей платформы

Технологии разработки – разработка проводилась на языках программирования JavaScript и TypeScript с использованием фреймворка React JS. В качестве среды разработки использовался редактор программного кода Visual Studio Code.

Среда исполнения – Windows 11, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы было создано пользовательское приложение для автоматизации задач оператора в составе информационно-управляющей платформы.

Все требования к разработке приложения соблюдены.

					ПГУ 09.03.02 – 08BP211.13 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Пользовательское приложение задач оператора в составе информационно-управляющей платформы Пояснительная записка	Лит.	Лист	Листов
Разраб.		Жуков А.В.					4	160
Пров.		Шибанов С.В.				Группа 21ВИ1		
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и планирование процесса разработки приложения для автоматизации задач оператора.....	9
1.1 Low-code/No-code платформа для развертывания, эксплуатации и сопровождения информационно-управляющих систем	9
1.1.1 Назначение и основные возможности платформы.....	9
1.1.2 Концептуальная модель информационно-управляющей системы на основе платформы.....	12
1.1.2.1 Внешние информационно-управляющие системы.....	13
1.1.2.2 События.....	13
1.1.2.3 Активные правила.....	15
1.1.3 Обобщенная программная архитектура платформы.....	16
1.2 Принципы организации и функционирования приложения для автоматизации задач оператора.....	19
1.2.1 Классы отображаемых объектов и их свойства.....	19
1.2.2 Бизнес-процессы управления отображаемыми объектами.....	21
1.2.3 Принципы модульности и расширяемости при построении приложения для автоматизации задач оператора.....	24
1.3 Сравнительный анализ современных приложений для автоматизации задач оператора.....	27
1.3.1 Обоснование сравнительного анализа.....	27
1.3.2 Обзор приложений для анализа.....	30
1.3.3 Результаты сравнительного анализа и выводы.....	34
1.4 Планирование процесса разработки программных средств.....	36
1.4.1 Цели разработки и назначение программных средств.....	36
1.4.2 Классы пользователей программных средств.....	37
1.4.3 План-график разработки и планирование бюджета.....	38
2 Анализ требований и постановка задачи на разработку приложения для автоматизации задач оператора.....	41

2.1 Анализ функциональных требований.....	41
2.2 Анализ нефункциональных требований.....	49
2.3 Анализ требований к пользовательскому интерфейсу	52
2.4 Анализ требований к аппаратно-программному обеспечению	56
2.4.1 Требования к аппаратному обеспечению	56
2.4.2 Требования к системному программному обеспечению	58
2.4.3 Требования к средствам разработки	59
2.5 Постановка задачи на разработку приложения	61
3 Разработка программных средств приложения для автоматизации задач оператора.....	62
3.1 Разработка архитектуры развертывания программных средств.....	62
3.2 Разработка сценариев взаимодействия пользователя с компонентами приложения.....	65
3.3 Разработка пользовательского интерфейса приложения.....	68
3.3.1 Разработка структуры пользовательского интерфейса.....	68
3.3.2 Разработка визуальных прототипов пользовательского интерфейса.....	71
3.4 Разработка программного обеспечения приложения.....	83
3.4.1 Разработка серверной части приложения.....	83
3.4.1.1 Разработка структуры серверной части приложения.....	83
3.4.1.2 Разработка программных модулей уровня модели	86
3.4.1.3 Разработка программных модулей уровня контроллера	90
3.4.2 Разработка клиентской части приложения.....	91
3.4.2.1 Разработка программных модулей уровня представления.....	91
4 Контроль качества программных средств взаимодействия с объектами мониторинга и управления.....	94
4.1 Планирование процесса контроля качества приложения	94
4.1.1 Мероприятия по контролю качества.....	94
4.1.2 Обоснование выбора модели тестирования	95
4.1.3 Обоснование выбора метрик программного кода	96

4.1.4 Обоснование выбора методик оценки качества пользовательского интерфейса	98
4.2 Тестирование приложения	99
4.2.1 Функциональное тестирование	99
4.2.2 Конфигурационное тестирование	103
4.3 Оценка качества программного кода	105
4.3.1 Метрики устойчивости и абстрактности	105
4.3.2 Метрики Холстеда	109
4.4 Оценка качества пользовательского интерфейса	112
Заключение	114
Список использованных источников	116
Приложение А (справочное) Глоссарий	119
Приложение Б (обязательное) Реализация моделей	121
Приложение В (обязательное) Реализация контроллера компонента системы	123
Приложение Г (обязательное) Реализация представления компонента системы	129
Приложение Д (рекомендуемое) Визуальные прототипы пользовательского интерфейса	139
Приложение Е (рекомендуемое) Результаты функционального тестирования	144
Приложение Ж (рекомендуемое) Результаты конфигурационного тестирования	154

Введение

Информационно-управляющие системы (ИУС) играют ключевую роль в обеспечении эффективного функционирования технических и технологических объектов. Современные ИУС представляют собой комплекс программных и аппаратных средств, предназначенных для сбора, обработки, хранения, отображения и передачи информации, необходимой для принятия решений оператором. Одним из важнейших компонентов ИУС является операторский интерфейс, обеспечивающий визуализацию данных, мониторинг текущего состояния систем и взаимодействие с пользователем.

В условиях роста объёмов обрабатываемой информации, повышения требований к скорости реакции на события и необходимости удалённого управления возрастает значимость разработки адаптируемых и расширяемых программных решений для визуализации и управления данными. Программные средства, обеспечивающие оперативный доступ к событиям, отображение критических сообщений, сортировку и фильтрацию информации, а также возможность настройки интерфейса под задачи конкретного пользователя, становятся необходимыми в промышленности, энергетике, транспорте, а также в сфере умных зданий и интернета вещей.

Целью данной работы является разработка программного решения для визуализации событий и параметров устройств в рамках информационно-управляющей системы. Особое внимание уделяется гибкости архитектуры, адаптации интерфейса под требования оператора, поддержке плагинов и расширяемости системы.

Разрабатываемая система ориентирована на гибкость, масштабируемость, расширяемость и кроссплатформенность.

Таким образом, работа направлена на создание современной, удобной и надёжной информационно-управляющей системы, способной эффективно поддерживать деятельность операторов и специалистов технической поддержки в реальном времени.

1 Анализ предметной области и планирование процесса разработки приложения для автоматизации задач оператора

1.1 Low-code/No-code платформа для развертывания, эксплуатации и сопровождения информационно-управляющих систем

1.1.1 Назначение и основные возможности платформы

Разработка информационно-управляющей платформы (ИУП) ведется в рамках инициативной научно-исследовательской работы (НИР) «Методы и алгоритмы повышения эффективности оперативной обработки событий в системах активных правил», выполняемой на кафедре «Математическое обеспечение и применение ЭВМ». ИУП предназначена для создания, внедрения и эксплуатации информационно-управляющих систем (ИУС) на базе технологии Интернета вещей (IoT).

В законченном варианте ИУП может применяться для комплексной разработки безопасных, надежных, масштабируемых и высокопроизводительных приложений в режиме low-code/no-code без написания кода (либо с минимальным объемом кодирования) на всех этапах разработки, внедрения и сопровождения ИУС.

Одним из главных преимуществ Low-code и No-code платформы является скорость разработки. Благодаря визуальному интерфейсу и готовым модулям разработчики могут сосредоточиться на функционале и бизнес-логике, не тратя время на написание кода с нуля.

Low-code и No-code платформа должна обеспечить снижение расходов на разработку, так как требуется меньше времени и усилий по сравнению с традиционными методами разработки. Средствами платформы можно создавать MVP-решения (минимально жизнеспособный продукт) или прототипы, что позволит проверять идеи на жизнеспособность и доработать первоначальные продукты до нужного уровня.

Платформа даст больше возможностей для кастомизации и масштабирования, так как поддерживает использование кода для настройки специфических функций. Это позволит компаниям создавать более сложные и функциональные приложения, которые могут адаптироваться к растущим требованиям.

Low-code/No-code платформа позволит инженерам и техническим специалистам с хорошим знанием индустрии, но небольшим опытом программирования, создавать и разворачивать как рыночно-ориентированные продукты, так и сервисы внутренней цифровизации предприятий. Это обеспечит привлечение команд с высокой бизнес-экспертизой к процессу создания программного обеспечения.

ИУП позволит разработчикам в интерактивном режиме формировать метаданные, описывающие аспекты организации и поведения дискретно-событийных реагирующих ИУС (ДСРИУС) [1]:

- инфраструктуру – IoT-устройства (датчики и устройства управления);
- обрабатываемые данные – входные и хранимые данные;
- логику мониторинга – правила обнаружения событий во входных данных, поступающих от устройств наблюдения (датчиков);
- логику управления – активные правила, описывающие реакцию на обнаружение событий.

На основе метаданных создается и настраивается программное обеспечение, непосредственно отвечающее за обработку входных данных, обнаружение событий и реагирование на обнаруженные события.

Платформа позволяет выполнять потоковую обработку больших объемов входных данных с высокой скоростью, чтобы обеспечивать оперативное обнаружение событий, мониторинг и управление.

Для реагирования на обнаруженные события в ИУП используются активные правила (Event-Condition-Action, ECA).

Обнаруженные события, состояния физических и логических объектов управления, другие данные сохраняются в памяти. Поддерживается три уровня хранения данных:

- оперативный – хранение данных непосредственно в потоках;
- промежуточный – хранение данных в промежуточном хранилище в формате key-value, а качестве которого используется in-memory система управления данными;
- долговременный – хранение данных долговременном хранилище, в качестве которого используется реляционная система управления данными.

Оперативный уровень хранения используется в процессе обработки данных при их поступлении в систему, на этапе обнаружения событий и в ходе исполнения активных правил.

Промежуточный уровень хранения используется в основном пользовательскими приложениями для быстрого доступа к архивным данным, а также для восстановления оперативных данных в случае их потери и различных сбоев.

Долговременный уровень хранения используется для архивирования и журнализации накопленных данных. Долговременные данные в дальнейшем планируется использовать для интеллектуального анализа, разбора нештатных ситуаций и пр.

Глубина хранения данных настраивается для каждого проекта отдельно.

Основными функциями информационно-управляющей платформы являются [2]:

- определение и актуализация инфраструктуры датчиков и устройств управления;
- конструирование и управление событиями и активными правилами;
- обработка потоков данных от устройств мониторинга, первичная обработка простых и обнаружение сложных событий;

- исполнение активных правил по отношению к объектам управления;
- журнализация и анализа событий и активных правил;
- аудит событий и активных правил;
- настройка.

В процессе дальнейших исследований и разработки функционал информационно-управляющей платформы будет развиваться и совершенствоваться.

1.1.2 Концептуальная модель информационно-управляющей системы на основе платформы

В основе организации и функционирования информационно-управляющей платформы лежит концептуальная модель, включающая классы информационных объектов и отношений между ними (рисунок 1) [3].

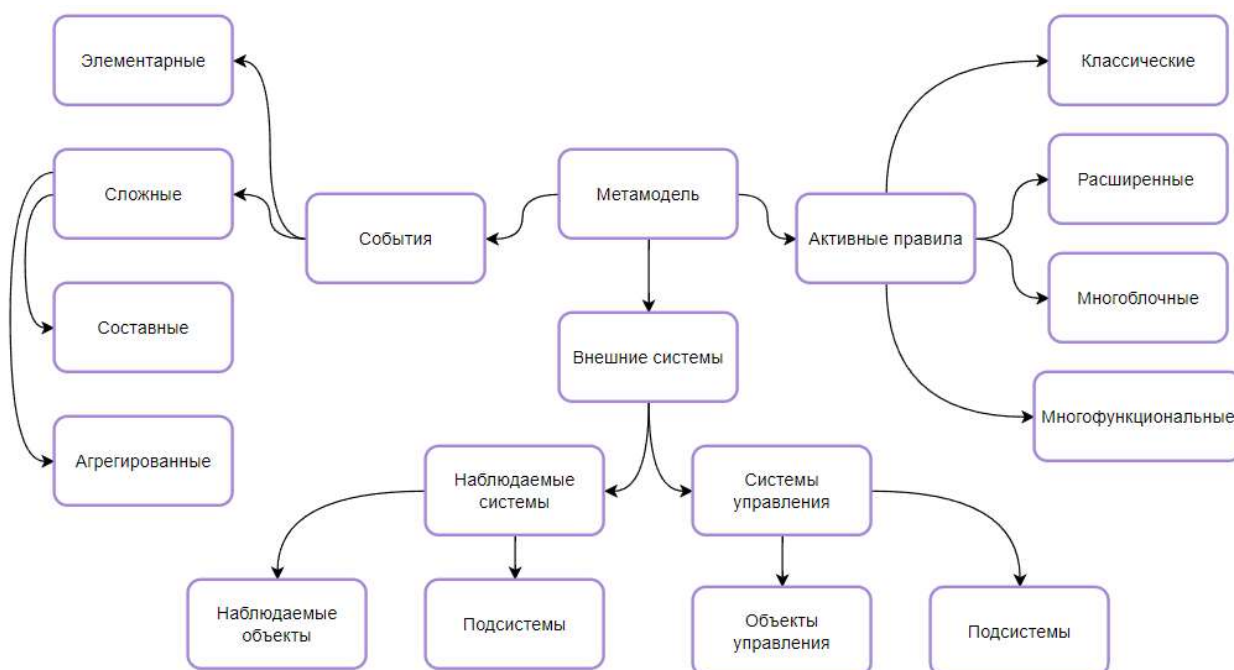


Рисунок 1 – Обобщенная концептуальная модель информационно-управляющей платформы

Концептуальная модель включает следующие классы объектов:

- системы мониторинга и управления;

- объекты мониторинга и объекты управления;
- события, поступающие от объектов мониторинга;
- активные правила для обработки событий и выработки управляющих воздействий;
- другие виды объектов.

В дальнейшем набор классов информационных объектов концептуальной модели может быть расширен и видоизменен.

1.1.2.1 Внешние информационно-управляющие системы

Информационно-управляющая платформа может взаимодействовать с множеством внешних информационно-управляющих систем, которые могут подразделяться на системы мониторинга и системы управления.

Любая внешняя ИУС может иметь внутреннюю, иногда достаточно сложную, структуру и состоять из множества подсистем. Каждая ИУС или ее подсистема могут включать множество объектов мониторинга и объектов управления.

Объекты мониторинга являются источниками данных. Данные, поступающие от объектов мониторинга, интерпретируются как базовые (или первичные) события и образуют входной поток событий.

К объектам управления могут быть применены различные управляющие воздействия как реакция на возникающие события. Управляющие действия, применяемые к объектам управления, создают выходной поток действий.

1.1.2.2 События

Дискретно-событийные ИУС (ДСИУС) могут порождать, передавать и обрабатывать большое количество событий в единицу времени, поступающих от внешних источников в дискретные моменты времени, а также события, возникающие внутри системы в результате выполнения различных операций. Они могут быть как запланированными, так и возникать спонтанно.

ДСИУС играют ключевую роль в обеспечении эффективной работы информационных систем, позволяя им реагировать на изменения внешней и внутренней среды, а также обеспечивать выполнение необходимых функций.

События делятся на элементарные (простые) и комплексные (сложные), которые могут быть либо агрегированными, либо составными. (рисунок 2).



Рисунок 2 – Классификация событий

Элементарные события поступают от внешних наблюдаемых систем, объекты наблюдения которых отправляют различную информацию в определенные моменты времени. Элементарное событие описывается уникальным идентификатором события, контекстом события – набором параметров, описывающих природу события, идентификатором источника события – объекта мониторинга, типом события, временем возникновения события.

Сложные события являются вычисляемыми событиями, которые фиксируются платформой в результате выполнения операций над элементарными и/или другими сложными событиями. Эти сложные события можно дополнительно классифицировать на агрегированные и составные события.

Агрегированное событие формируется на основе агрегирующего запроса к входному потоку элементарных событий.

Составное событие представляет собой комбинацию элементарных и/или сложных событий, с возможным использованием ограничений и логических операций. Контекст составного события определяется параметрами контекста входящих событий.

1.1.2.3 Активные правила

В основе поведения информационно-управляющей платформы при реагировании на события, возникающие во внешних системах мониторинга и управления, лежит модель активных правил (Event-Condition-Action, ECA). Активное правило включает:

- событийную часть, которая описывает тип события, инициирующего данное правило;
- условную часть, представляющую собой логическое выражение;
- активную часть, состоящая из операций над объектами управления.

Реакция на возникающие события вырабатывается в процессе исполнения активных правил, ассоциированных с типами событий. Активные правила можно классифицировать на классические и расширенные (рисунок 3).

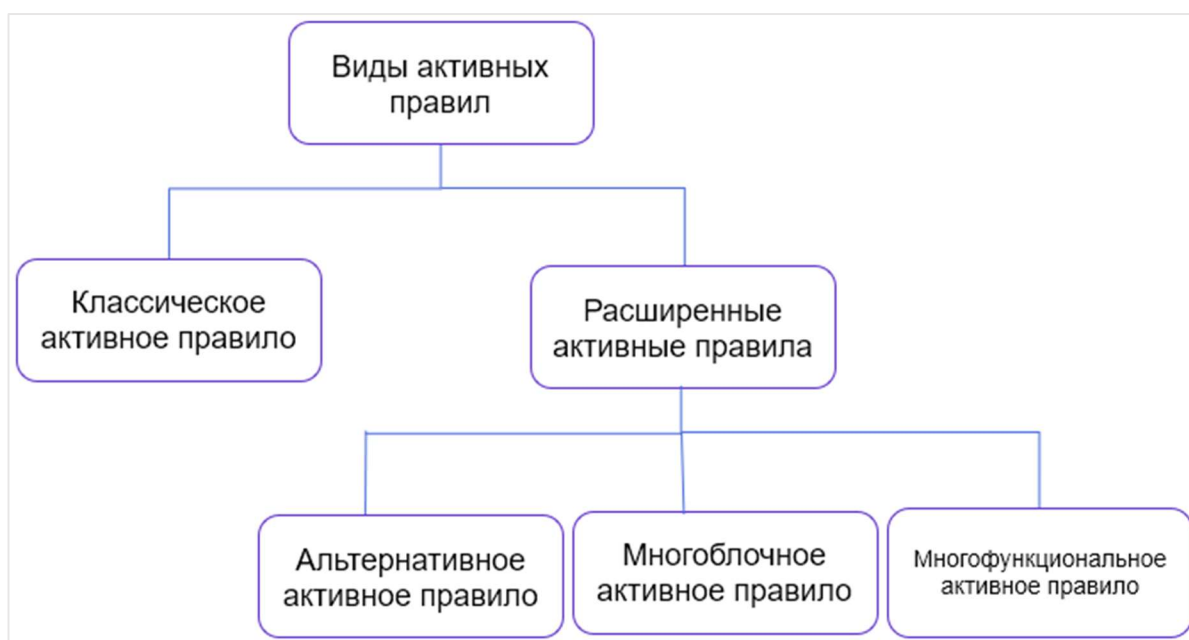


Рисунок 3 – Классификация активных правил

Классическое активное правило описывается кортежем $r = \langle e, c, a \rangle$, где e – тип события, обрабатываемого системой, c – условие, логическое выражение над контекстом события, при истинности которого для экземпляра события выполняется исполнение активного правила, a – действие, набор операций, выполняемые над объектами управления в случае истинности условия c .

Появление расширенных активных правил обусловлено растущей сложностью бизнес-процессов и необходимостью повышения быстродействия систем, вследствие минимизации множества активных правил.

Альтернативное активное правило представляется как $r = \langle e, c, \{a_{base}, a_{alt}\} \rangle$, где a_{base} – действие, выполняемое в случае истинности условия, a_{alt} – альтернативное действие, выполняемое в случае, если условие ложно.

Для описания реакции на сложное событие с множеством возможных результирующих событий применяется многоблочные активные правила, для которых определены несколько вариантов обнаружения событий. Такие правила представляются как $r = \langle \{e_1, c_1, a_1\}, \dots, \{e_n, c_n, a_n\} \rangle$, где $e_1, \dots, e_n$ – множество результирующих событий, $c_1, \dots, c_n$ – множество условий, соответствующих множеству результирующих событий, $a_1, \dots, a_n$ – множество действий, соответствующих множеству условий.

Многофункциональное активное правило применяется в случае, когда в качестве реакции на возникновения события и истинности условия необходимо выполнить множество реагирующих воздействий. Многофункциональное правило представляется кортежем $r = \langle e, c, A \rangle$, где $A = \{a_1, \dots, a_k\}$ – множество экземпляров результирующих воздействий, применяемых к управляемым объектам.

1.1.3 Обобщенная программная архитектура платформы

Архитектурно информационно-управляющая платформа состоит из нескольких достаточно крупных программных компонент, которые взаимодействуют между собой по протоколу RESTful API (рисунок 4).

Программные средства управления метаданными реализованы в виде отдельного портала, который позволяет создавать логические элементы информационно-управляющих систем и сохранять их описание в центральном репозитории метаданных. Метаданные из центрального репозитория тиражируются в другие компоненты платформы и размещаются в локальных репозиториях.

Центральный репозиторий представляет собой глобальное хранилище метаданных объектов информационно-управляющих систем, разворачиваемых на платформе.

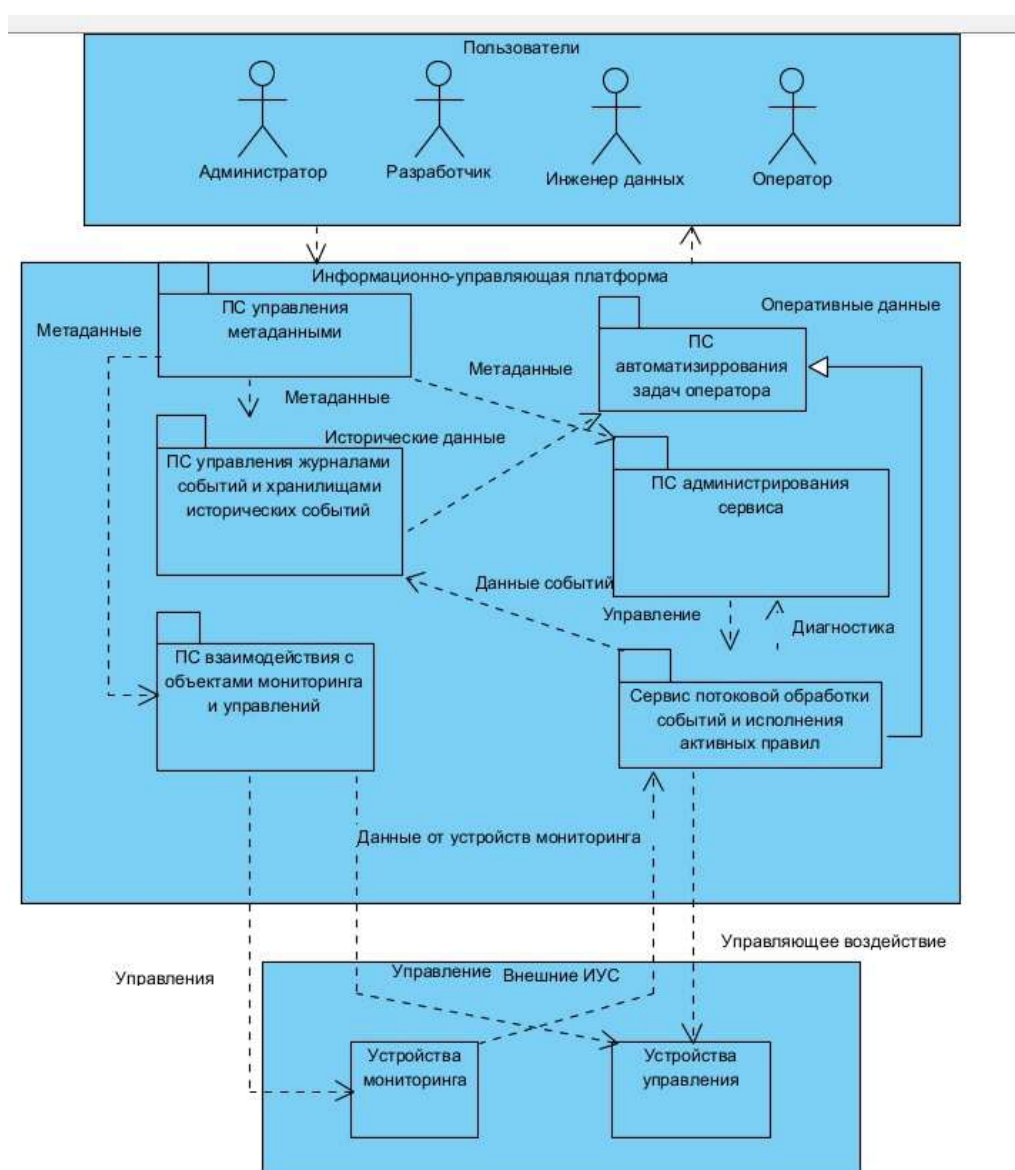


Рисунок 4 – Обобщенная программная архитектура информационно-управляющей платформы

REST-сервер в составе программных средств управления метаданными реализует операции ведения и верификации метаданных в центральном репозитории, также синхронизацию и тиражирование метаданных в локальные репозитории других компонент платформы.

Сервис потоковой обработки событий и исполнения активных правил предназначен для обработки данных, поступающих от IoT-устройств, обнаружения заданных событий и исполнения активных правил в ответ на возникновение событий. Сервис построен на основе микросервисной и событийно-ориентированной архитектуры.

Микросервисная архитектура представляет собой подход к разработке единого приложения в виде набора небольших сервисов, работающих в отдельных процессах, и взаимодействующих с применением упрощенных механизмов. Событийно-ориентированная архитектура состоит из независимых и специализированных компонентов обработки событий, которые асинхронно получают и обрабатывают события. Использование микросервисной и событийно-ориентированной архитектуры предполагает независимое масштабирование сервисов, высокую отказоустойчивость, ускоренную разработку.

Программные средства взаимодействия с объектами мониторинга и управления реализуют основные операции с IoT-устройствами – создание и развертывание устройств, контроль работоспособности, моделирование их поведения. В настоящее время поддерживается взаимодействие по протоколу MQTT через облачный сервис Yandex IoT Cloud. Для непосредственного управления IoT-устройствами и моделирования их поведения применяется платформа NODE-Red.

Программные средства управления журналами событий и хранилищами исторических данных предназначены организации оперативного, промежуточного и долговременного хранения данных об обнаруженных событиях, состояниях физических и логических объектов управления и другие. Сохраненные и накопленные данные доступны для просмотра и анализа.

В состав платформы входят пользовательские приложения с графическим интерфейсом, ориентированные на различные классы пользователей. Для разработчиков предназначен портал управления метаданными, для операторов и диспетчеров – приложение для автоматизации задач оператора, для администратора устройств – панель управления устройствами, для инженера данных – панель управления данными.

1.2 Принципы организации и функционирования приложения для автоматизации задач оператора

1.2.1 Классы отображаемых объектов и их свойства

В рамках функционирования приложения для автоматизации задач оператора основными элементами пользовательского интерфейса являются отображаемые объекты, представляющие собой логически обособленные единицы информации, доступные для визуального представления, анализа и управления.

Каждый отображаемый объект принадлежит к определённому классу, который задаёт его структуру, семантику и поведение в системе. Ниже приведены основные классы отображаемых объектов и их свойства.

Класс пользователь (User) необходим для отображения информации о зарегистрированных участниках системы.

Свойства пользователя:

- id – уникальный идентификатор пользователя;
- email – адрес электронной почты;
- username – имя пользователя;
- role – роль (оператор, администратор);
- status – активен/удален.

Класс устройство (Device) предназначен для представления физических или виртуальных устройств IoT.

Свойства устройства:

- id – уникальный идентификатор устройства;
- name – наименование;
- ssid – серийный номер;
- type – тип (датчик, контроллер и прочее);
- location – привязка к локации;
- status – активность (вкл/выкл);
- userPermissions – список пользователей с правами доступа.

Класс событий (Event) предназначен для визуализации и контроля произошедших инцидентов или изменений состояния устройств.

Свойства событий:

- id – уникальный идентификатор события;
- timestamp – время возникновения;
- deviceId – идентификатор связанного устройства;
- type – тип события (например, перегрев, ошибка подключения);
- severity – уровень серьёзности;
- message – описание сообщения от устройства;
- parameters – дополнительные параметры (значение, единица измерения, и прочее);
- acknowledged – статус признания события.

Класс локаций (Location) необходим для организации иерархии объектов по физическому или логическому размещению.

Свойства локаций:

- id – уникальный идентификатор;
- name – название локации (комната, блок, зона);
- parentId – родительская локация (для дерева вложенности);
- depth – уровень вложенности.

Класс виджетов (Widget) нужен для отображения данных в виде графиков, таблиц или карточек.

Свойства виджетов:

- id – уникальный идентификатор виджета;
- type – тип (график, таблица, карточка);
- dataSource – источник данных (устройства, события);
- dataValue – значения виджета.

Таким образом, структура отображаемых объектов обеспечивает модульность и расширяемость пользовательского интерфейса, позволяя оператору выполнять контроль, диагностику и управление объектами системы в интерактивном и интуитивно понятном виде.

1.2.2 Бизнес-процессы управления отображаемыми объектами

Бизнес-процесс – это совокупность взаимосвязанных задач и мероприятий, направленных на достижение определенной цели или результата в рамках организации. Бизнес-процессы могут включать как производственные, так и административные функции и являются основой эффективного функционирования предприятия. Правильное управление и оптимизация бизнес-процессов позволяет значительно улучшить производительность, качество, гибкость, прозрачность и управляемость предприятия. Использование современных методов и инструментов анализа и автоматизации бизнес-процессов, помогает достигать высоких результатов и обеспечивать конкурентоспособность на рынке [5].

Для отображения бизнес процесса управления отображаемыми объектами была выбрана диаграмма деятельности.

Диаграмма деятельности (или диаграмма активностей) в UML – это графическое представление потока управления между различными действиями, часто используемое для моделирования бизнес-процессов или алгоритмов. Она показывает последовательность шагов и условий, которые приводят к изменению состояния системы или выполнению определенной задачи.

Диаграмма управления отображаемыми объектами изображена на рисунке 5.

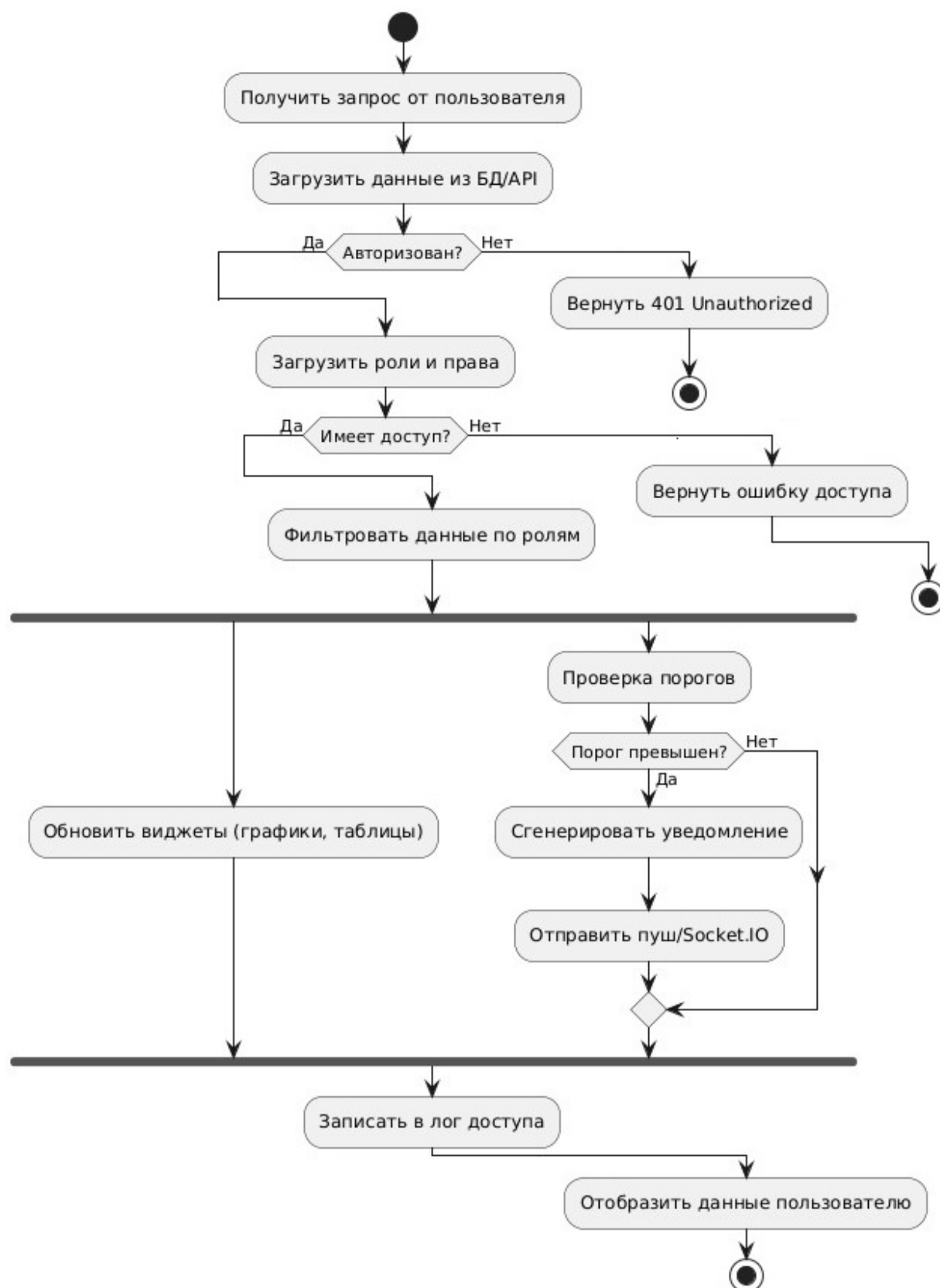


Рисунок 5 – Управление отображаемыми объектами

Процесс охватывает цикл обработки пользовательского запроса – от первичной аутентификации до отображения отфильтрованных данных с параллельной логикой обработки пороговых значений и генерации уведомлений.

Бизнес-процесс можно условно разделить на следующие функциональные блоки:

- инициация запроса;
- аутентификация и авторизация;
- проверка прав доступа;
- обновление визуальных компонентов;
- генерация уведомлений;
- логирование событий.

Инициация запроса – событие начала инициируется действием пользователя (например, загрузка дашборда или панели мониторинга). Запускается процесс извлечения данных из источников хранения, например база данных, кэш, API.

Аутентификация и авторизация – производится проверка наличия валидного токена (например, JWT). В случае успеха происходит извлечение информации о пользователе, его ролях и уровнях доступа (на основе RBAC-модели). В случае неуспешной аутентификации – генерация ответа 401 Unauthorized.

Проверка прав доступа – система определяет, имеет ли пользователь право на доступ к конкретным устройствам или данным. При наличии доступа – данные фильтруются в соответствии с разрешённым уровнем доступа. При отсутствии доступа – пользователю возвращается сообщение об ошибке, и процесс завершается.

Обновление визуальных компонентов – после фильтрации данных запускается параллельное выполнение подзадач, таких как обновление визуальных компонентов интерфейса – графики, таблицы, индикаторы.

Проверка пороговых значений – сравнение полученных параметров с допустимыми границами.

Генерация уведомлений – в случае превышения установленных порогов параметров. Формируется уведомление для пользователей, имеющих соответствующие права. Уведомление доставляется через механизм Web-сокета.

Логирование событий – все действия регистрируются в журнале событий.

Данная модель бизнес-процесса необходима для:

- формализации логики работы интерфейса с отображаемыми объектами;
- обеспечения безопасности путём строгой авторизации и фильтрации данных;
- повышения отказоустойчивости и наблюдаемости системы за счёт логирования;
- поддержки динамических уведомлений и автоматической реакции на события.

Программа отражает логически завершённую последовательность операций, обеспечивающих целостность, актуальность и безопасность данных, отображаемых в пользовательском интерфейсе. Модель легко масштабируется и может быть адаптирована под интеграцию дополнительных подсистем (например, управление пользователями, редактирование виджетов, подключение внешних плагинов).

1.2.3 Принципы модульности и расширяемости при построении приложения для автоматизации задач оператора

Разработка информационно-управляющих приложений для автоматизации задач оператора требует соблюдения архитектурных принципов, обеспечивающих гибкость, масштабируемость и адаптацию

системы к изменяющимся условиям эксплуатации. Ключевыми среди них являются модульность и расширяемость.

Модульность предполагает структурную декомпозицию приложения на независимые и переиспользуемые компоненты, каждый из которых реализует строго определённый функционал. Такой подход упрощает сопровождение, тестирование и повторное использование кода, а также обеспечивает чёткое разделение ответственности между слоями и модулями. В контексте проектируемой системы модульность реализуется на компонентном, сервисном и плагинном уровнях.

На компонентном уровне интерфейса – каждый виджет (таблица, график, карта, панель событий и др.) реализуется как самостоятельный UI-модуль с собственным состоянием и API.

На сервисном уровне логики бизнес-логика приложения разнесена по сервисам, отвечающим за авторизацию, маршрутизацию событий, хранение и обработку данных.

На плагинном уровне расширения – архитектура позволяет подключать новые компоненты (например, пользовательские графики или интеграции) без изменения основного кода приложения.

Расширяемость обеспечивает возможность адаптации и наращивания функциональности без нарушения стабильности уже реализованных компонентов. Это критически важно для приложений, эксплуатируемых в условиях многопрофильной среды и динамически меняющихся требований. Расширяемость достигается за счёт:

- внедрения плагинной архитектуры, позволяющей подключать внешние модули (например, новые типы виджетов, драйверы интеграции, логики уведомлений);
- использования унифицированных интерфейсов взаимодействия между модулями, в частности через событийную модель и механизмы подписки/рассылки;

- поддержки настройки интерфейса через конфигурационные файлы или панели управления, что обеспечивает быструю адаптацию без перекомпиляции;
- изоляции плагинов в собственных пространственных и логических контекстах, минимизирующих влияние на остальную систему.

При разработке приложения для автоматизации задач оператора ключевым требованием, определяющим архитектурные решения, стала модульность и расширяемость как на уровне клиентского, так и серверного компонентов. Эти принципы обеспечивают возможность постепенного наращивания функциональности, упрощают сопровождение и способствуют адаптации системы к специфике различных отраслей.

На стороне backend-модуля используется сервисно-ориентированная архитектура (SOA), где каждый функциональный блок (аутентификация, обработка событий, управление пользователями, маршрутизация уведомлений и пр.) реализуется как независимый плагин, регистрируемый в общем приложении. Основными реализованными подходами является плагиновая модель загрузки модулей – каждый функциональный компонент может быть добавлен в систему без необходимости модификации ядра приложения.

Контракты между модулями – интерфейсы взаимодействия между компонентами стандартизированы, что обеспечивает их взаимозаменяемость. Механизм расширения моделей данных через адаптеры позволяет подключать дополнительные источники (например, БД, брокеры сообщений) без нарушения логики обработки.

Таким образом, бэкенд-платформа выступает как конструктор, в котором каждый сервис может быть разработан, протестирован и внедрён независимо.

В клиентской части применены принципы компонентной архитектуры с поддержкой плагинов.

Виджеты как независимые модули – таблицы, графики, карты устройств, панели событий реализуются как обособленные компоненты, которые можно подключать и отключать без изменения основного интерфейса.

Динамическая регистрация компонентов – виджеты загружаются через централизованный реестр и могут поставляться как внешние пакеты.

Использование React и JavaScript – композиционный подход позволяют гибко управлять зависимостями между элементами интерфейса.

Таким образом, клиентское приложение построено по принципу ядра и расширяемых виджетов, что позволяет разрабатывать новые интерфейсные элементы отдельно от основного приложения и интегрировать их через публичный API.

В совокупности, описанные архитектурные подходы формируют прочный фундамент для построения гибкой, расширяемой системы, способной адаптироваться к динамическим требованиям и задачам операторского управления.

1.3 Сравнительный анализ современных приложений для автоматизации задач оператора

1.3.1 Обоснование сравнительного анализа

Сравнительный анализ полученных данных – один из самых популярных и основополагающих методов анализа, предполагающий выявление определенных характеристик у объекта исследования и сравнение его по этим параметрам с другими объектами или с самим собой, но, например, в разные периоды времени.

В результате сравнения можно понять и увидеть что-то общее в явлениях и выявить закономерность или, наоборот, узнать, что никаких закономерностей нет, и все меняется хаотично, но это бывает намного реже.

Методология сравнительного анализа предполагает наличие объекта или объектов анализа, информации по ним, параметра, по которому будет

производиться сравнение и проведение анализа по одному из возможных методов, а также создание рекомендаций на основании полученных данных.

Среди методов big data для анализа больших данных выделяют:

- общенаучные методы (аналогия, анализ, синтез);
- логические методы (выделение категорий, сегментов).

Сравнительный анализ полученных данных в ходе работы компании можно проводить как по первой группе методов, так и по второй. Аналогия предполагает перенесение опыта, например, предыдущего сезона или конкурентной компании на этот сезон или рассматриваемую организацию. Синтез – объединение разных частей в нечто новое, например, создание новой линейки товаров.

Целью сравнительного анализа является выбор наилучшего программного решения или платформы для автоматизации задач оператора в контексте мониторинга, визуализации и управления событиями в распределённых системах. Результатом анализа является обоснованный выбор архитектурного и технологического решения, удовлетворяющего требованиям по функциональности, удобству использования, масштабируемости и интеграции с существующими компонентами системы.

Основные задачи сравнительного анализа:

- определить возможности современных решений по визуализации и управлению событиями;
- выявить преимущества и недостатки приложений по ключевым характеристикам;
- оценить соответствие функционала требованиям типового оператора;
- выбрать наилучшее решение в контексте планируемого программного комплекса [6].

Критерии оценки и расшифровки сравнительного анализа приведены в таблице 1.

Таблица 1 – Критерии оценки и их расшифровка

№	Критерий	Расшифровка
1	Функциональная полнота	Наличие всех ключевых функций: фильтрация, сортировка, уведомления, работа с графиками и таблицами.
2	Удобство пользовательского интерфейса	Оценка логики взаимодействия, адаптивности, доступности и UX-дизайна.
3	Масштабируемость	Способность платформы обрабатывать увеличивающийся объём данных и пользователей без деградации производительности.
4	Интеграция с внешними системами	Наличие готовых API, поддержка подключаемых плагинов, работа с базами данных и брокерами сообщений.
5	Конфигурируемость и расширяемость	Возможности настройки интерфейса, поддержки тем, языков, загрузки пользовательских компонентов.
6	Лицензирование и стоимость	Наличие открытого исходного кода, условия использования, стоимость коммерческих версий.
7	Активность сообщества/поддержка	Регулярность обновлений, наличие документации, активность разработчиков и пользователей.

Шкалы оценки критериев (по 5-балльной шкале):

5 – отлично, реализовано полностью, гибко и без существенных ограничений;

4 – хорошо, реализовано с незначительными ограничениями или компромиссами;

3 – удовлетворительно, функционал присутствует, но с заметными ограничениями;

2 – плохо, реализовано частично или нестабильно;

1 – неудовлетворительно, отсутствует или требует значительных доработок.

Для проведения анализа были выбраны наиболее популярные и широко применяемые на практике решения, охватывающие как open-source, так и коммерческие платформы.

Критерии отбора платформ для сравнительного анализа:

- высокая популярность и зрелость в своих сегментах;
- открытость к доработке и расширению;
- наличие активного сообщества и документации;

– применимость для задач визуализации событий и работы оператора.

Выбранные приложения и их описание представлено в таблице 2.

Таблица 2 – Выбор приложений для анализа

№	Название платформы	Тип лицензии	Язык/технологии	Основное назначение	Возможность расширения
1	Grafana	Open Source (AGPL)	GO, React, TypeScript	Визуализация временных рядов	Плагины, панель
3	ThingsBoard	Open Source / Pro	Java, Angular	IoT-платформа, визуализация и управление устройствами	Расширяемые виджеты
3	Home Assistant	Open Source (Apache 2.0)	Python, YAML, Jinja2	Автоматизация и визуализация в умных домах	Пользовательские дашборды

Для оценки применимости современных решений были выбраны как общепринятые промышленные стандарты визуализации (Grafana, ThingsBoard), так и специализированные платформы с развитой автоматизацией (Home Assistant).

Результаты сравнительного анализа данных платформ будут использоваться в последующих разделах для обоснования архитектурных и технологических решений при проектировании собственной системы управления отображаемыми объектами.

1.3.2 Обзор приложений для анализа

В рамках сравнительного анализа были выбраны 3 современных приложений, каждое из которых представляет собой решение, применяемое для автоматизации задач оператора, мониторинга событий и визуализации данных. Приложения различаются по назначению, архитектурным принципам, способам конфигурации, а также по доступности расширений и уровням поддержки. Ниже представлен подробный обзор каждого из них.

- 1) Grafana;
- 2) ThingsBoard;

3) Home Assistant.

Grafana – широко используемая open source-платформа для построения интерактивных дашбордов, ориентированная на визуализацию временных рядов и данных мониторинга. Поддерживает подключение к множеству источников данных (Prometheus, InfluxDB, PostgreSQL и др.), предоставляет гибкие механизмы фильтрации и отображения данных, а также систему плагинов. Существует коммерческая версия Grafana Enterprise с дополнительными возможностями, которая также доступна как для самостоятельной установки, так и из облачной службы Grafana Labs. Интерфейс Grafana изображен на рисунке 6.

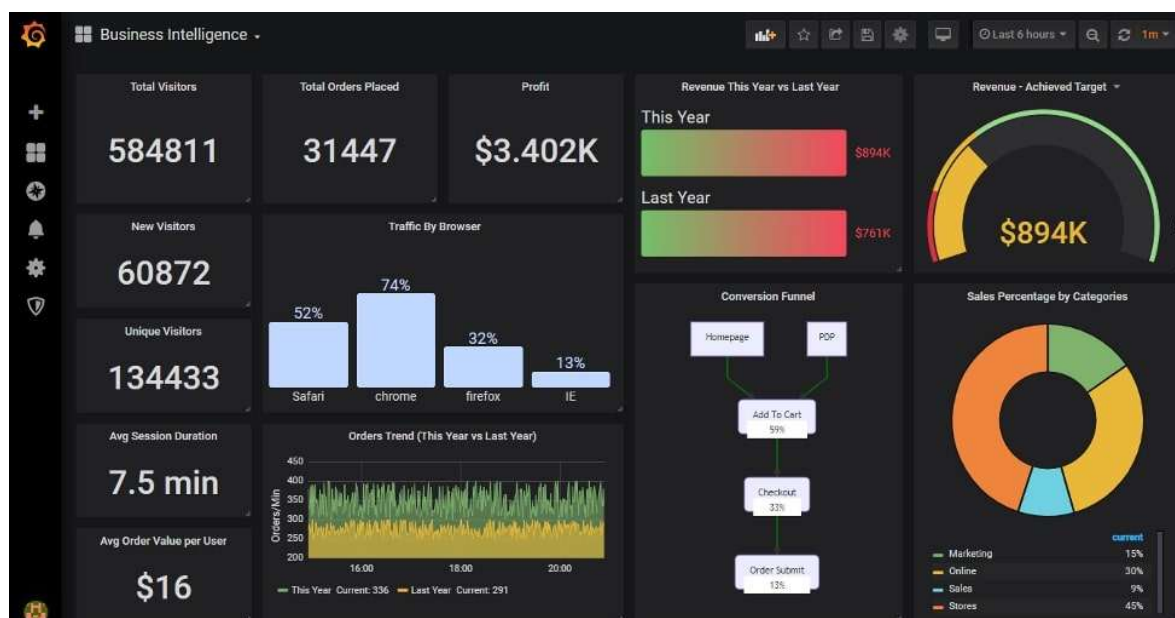


Рисунок 6 – Итерфейс Grafana

Ключевые особенности системы:

- высокая настраиваемость дашбордов и панелей;
- поддержка графиков, таблиц, heatmap, алертов;
- богатая экосистема плагинов;
- web-интерфейс с удобным UI;
- возможность создания пользовательских плагинов и панелей визуализации.

Grafana активно используется в DevOps-практиках, системах мониторинга серверов, IoT и аналитике телеметрии.

ThingsBoard – платформа IoT-уровня для сбора, обработки, визуализации и управления данными от подключённых устройств. Предлагает обширный инструментарий для создания дашбордов, построения правил обработки событий и реализации многоуровневой архитектуры доступа. Интерфейс ThingsBoard изображен на рисунке 7.

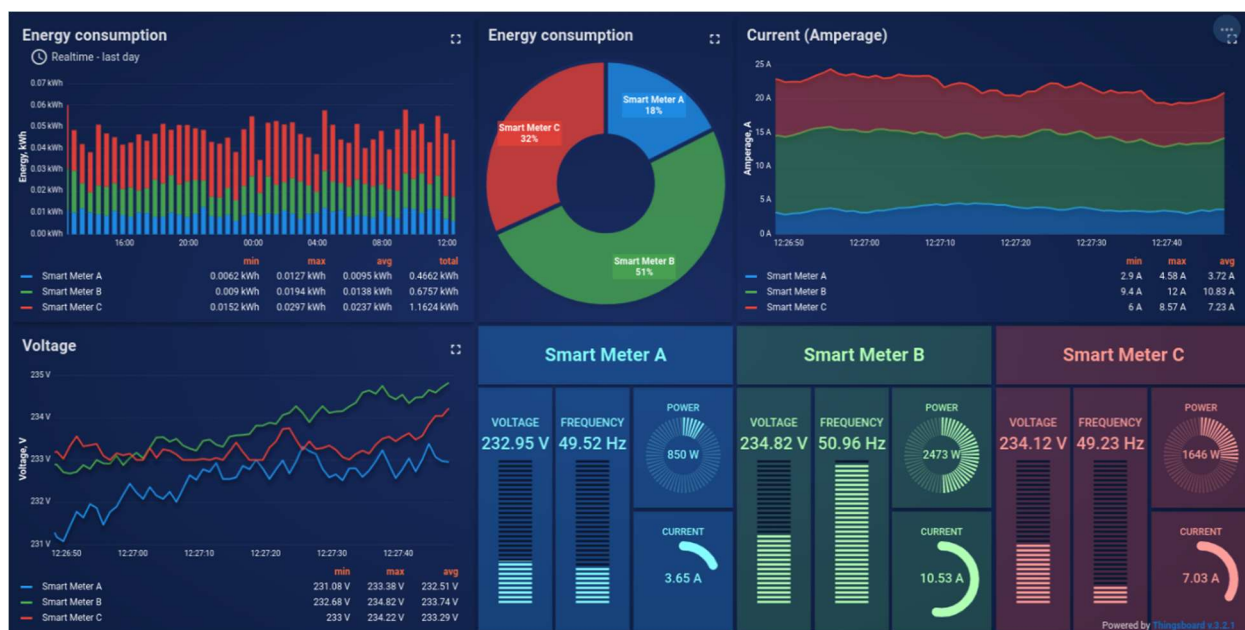


Рисунок 7 – Итерфейс ThingsBoard

Ключевые особенности:

- поддержка телеметрии в реальном времени;
- встроенные механизмы работы с устройствами, пользователями и правами доступа;
- механизм правил и алертов (Rule Engine);
- расширяемая архитектура дашбордов и виджетов;
- коммерческая версия с поддержкой кластеризации и multi-tenancy.

ThingsBoard ориентирован на промышленные приложения, умные города, агропромышленность и энергоэффективные системы.

Home Assistant – open source-платформа для автоматизации и управления устройствами умного дома. Обеспечивает возможность интеграции с большим числом устройств, настройку автоматизаций и визуализацию состояния систем. По умолчанию для базы данных исторических данных используется SQLite. Интерфейс с базами данных реализован на SQLAlchemy, благодаря чему поддерживается достаточно широкий набор реляционных СУБД, в частности, MySQL, MariaDB, PostgreSQL, Microsoft SQL Server. Интерфейс Home Assistant изображен на рисунке 8.



Рисунок 8 – Итерфейс Home Assistant

Ключевые особенности:

- поддержка более 1000 интеграций с внешними устройствами и сервисами;
- конфигурация через YAML-файлы или визуальный интерфейс;
- визуальные дашборды;
- механизмы автоматизации, сценариев и условной логики;
- возможность работы offline.

Несмотря на ориентацию на домашнюю автоматизацию, Home Assistant активно используется в проектах, где требуется гибкая настройка событий и интерфейсов без написания кода.

Каждое из рассмотренных приложений обладает рядом преимуществ и имеет определённую область применения. Наиболее универсальными по степени конфигурируемости и визуализации являются Grafana и ThingsBoard. Home Assistant – лидер в пользовательской автоматизации. Разрабатываемое решение сфокусировано на специфике корпоративного мониторинга событий и ориентировано на гибкость интерфейса, расширяемость и соответствие бизнес-процессам организации.

1.3.3 Результаты сравнительного анализа и выводы

В рамках проведенной работы был представлен сравнительный анализ трех платформ – Grafana, ThingsBoard, и Home Assistant – по семи критериям, релевантным для нашего IoT-проекта (управление событиями и устройствами с ролями, уведомлениями, графиками и таблицами). Каждая платформа была проанализирована по степени соответствия этим критериям. Особое внимание уделялось возможности расширения с помощью сторонних плагинов или интеграций. Также было важно, чтобы система поддерживала автоматизированную отправку уведомлений в реальном времени. В таблице 3 приведены результаты сравнительного анализа.

Таблица 3 – Результаты сравнительного анализа

№	Критерий	Grafana	ThingsBoard	Home Assistant
1	Функциональная полнота	4	5	4
2	Удобство пользовательского интерфейса	4	4	4
3	Масштабируемость	4	5	4
4	Интеграция с внешними системами	4	5	4
5	Конфигурируемость и расширяемость	4	5	4
6	Лицензирование и стоимость	5	5	5
7	Активность сообщества/поддержка	5	4	5

С точки зрения функциональной полноты Grafana предлагает мощные графики и таблицы, фильтры и оповещения, но требует внешнего хранилища данных, например, Prometheus или InfluxDB. ThingsBoard интегрирует сбор

телеметрии, визуализацию, фильтрацию, сортировку и уведомления из коробки. Home Assistant поддерживает графики и таблицы, фильтры и уведомления, но ориентирован на автоматизацию умного дома, а не на сложный аналитический интерфейс.

Grafana предлагает современный интерфейс с высокой адаптивностью и настройкой дашбордов. ThingsBoard обеспечивает гибкие, но менее узнаваемые UI-виджеты. Home Assistant ориентирован на конечного пользователя, интерфейс дружелюбный, но интерфейс дашбордов менее конфигурируемый.

Grafana работает с масштабируемыми TS базы, выдерживает сотни источников. ThingsBoard поддерживает кластеризацию, микросервисы и Prometheus-мониторинг. Home Assistant масштабируется хорошо в локальной среде, но при больших каталогах устройств может появляться лаг.

Grafana поддерживает сотни источников, включая Home Assistant, MQTT, Prometheus. ThingsBoard встроенные интеграции MQTT, HTTP, CoAP, LoRa, OPC-UA и API. Home Assistant поддерживает множество протоколов (Zigbee, Z-Wave, MQTT, cloud), ориентирован на умный дом.

Конфигурируемость и расширяемость:

- Grafana – дашборды, темы, плагины, переменные;
- ThingsBoard – rule chains, кастомные виджеты, API;
- Home Assistant – тематизация, YAML-автоматика, собственные карты.

С точки зрения лицензирования и стоимости все проекты с открытым исходным кодом и бесплатны. Grafana и Home Assistant предлагают коммерческие облачные/поддерживаемые версии, но базовые версии бесплатны.

Сообщество Grafana и Home Assistant имеют крупные активные сообщества, регулярные обновления и богатую документацию. ThingsBoard также развивается быстро, но сообщество заметно меньше.

1.4 Планирование процесса разработки программных средств

1.4.1 Цели разработки и назначение программных средств

Система представляет собой платформу мониторинга, управления и анализа событий для подключённых устройств (IoT). Система поддерживает прав доступа, событий и уведомлений в реальном времени. Так же система позволяет анализировать архивные данные.

Система позволяет собирать данные от устройств, датчиков и контроллеров. Хранить и анализировать архивные события. Оповещать пользователей в реальном времени. Предоставлять контролируемый доступ к устройствам на основе ролей пользователей.

Система спроектирована гибко и может быть применена в различных отраслях, например, в промышленной автоматизации, для мониторинга производственного оборудования, отслеживание превышений температур, вибраций, давления. Также данный механизм может использоваться в системе так называемого умного дома, для получения уведомления о превышении температуры, дыма, углекислого газа, а также для контроля доступа к системам отопления, вентиляции и безопасности. Еще один вариант применения связан с энергетикой и климат-контролем, в этом случае система может использоваться для мониторинга электросетей, генераторов и климатических систем, например, при перегреве или отключение питания срабатывает тревога. В логистике и транспорте возможно слежение за состоянием автомобиля, например, температура в рефрижераторах, открытие дверей или анализ отчётов о событиях, связанных с транспортом или грузом. В медицине и фармацевтике программа позволяет контролировать температуру хранилищ с лекарствами и в реальном времени присылать уведомления о сбоях в медицинском оборудовании, а также проводить аудит истории событий для соответствия нормативам.

Ключевые функциональные возможности платформы являются приём и регистрация событий от физических устройств, датчиков и контроллеров,

оповещение пользователей в реальном времени о критических событиях, построение графиков и таблиц для аналитики, хранение и последующий анализ архивных событий.

Таким образом, данная система представляет собой универсальное средство диспетчеризации, наблюдения и аналитики, способное адаптироваться под нужды конкретной отрасли за счёт гибкой архитектуры, поддержки плагинов и расширяемого интерфейса.

1.4.2 Классы пользователей программных средств

Классы пользователей – это группы пользователей, объединенные общими признаками, такими как уровень доступа, функциональность, частота использования продукта и другие.

На рисунке 9 представлена диаграмма вариантов использования платформы для интернета вещей.

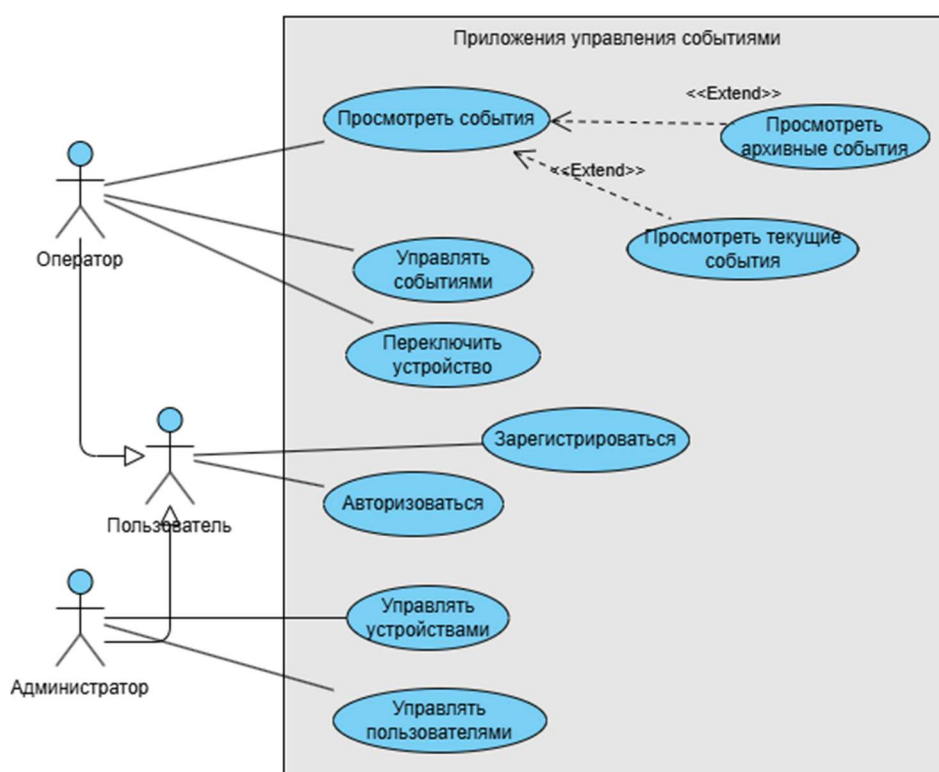


Рисунок 9 – Диаграмма вариантов использования платформы для интернета вещей

Ниже приведена функций сервиса для классов пользователей на таблице 4.

Таблица 4 – Функции сервиса для классов пользователей

Роль	Функция	Описание функции
Пользователь	Зарегистрироваться	Позволяет создать учетную запись в системе.
	Авторизироваться	Позволяет войти в систему под своими учетными данными.
Оператор	Просмотреть текущие события	Позволяет просматривать события, поступившие в систему от устройств в реальном времени или с минимальной задержкой. Используется, чтобы следить за происходящим прямо сейчас.
	Просмотреть архивные события	Предоставляет доступ к истории событий, которые были зафиксированы в прошлом. Полезно для анализа и отчётности.
	Управлять событиями	Оператор может не только просматривать, но и управлять событиями: подтверждать, помечать, фильтровать, принимать меры (например, отправить уведомление или изменить статус).
	Переключить устройство	Оператор может выключить устройство, если оно включено, и включить, если выключено.
Администратор	Управлять устройствами	Добавление, удаление, изменение данных устройств, назначение устройств пользователя (назначенные пользователи смогут управлять устройством).
	Управлять пользователями	Добавление, удаление, изменение пользователей, назначение ролей (наблюдатель, оператор, администратор). Администратор управляет политиками доступа и безопасностью.

В дальнейшем перечень функций сервиса будет уточнении и расширен. Для каждого варианта использования будет построен подробный сценарий.

1.4.3 План-график разработки и планирование бюджета

Функционально-стоимостный анализ (ФСА) – это метод, применяемый для оценки себестоимости продукта или услуги путём детализированного изучения и управления всеми производственными и организационными процессами, участвующими в создании конечного результата.

Для разработки программных средств были определены основные задачи проекта, представленные в таблице 5.

Таблица 5 – Основные задачи проекта

№	Название задачи	Длительность (дней)	Предшественники
1	Анализ требований и постановка задачи	3	—
2	Проектирование архитектуры системы	4	1
3	Разработка структуры базы данных	3	2
4	Реализация уровня модели (Model)	4	3
5	Разработка API и бизнес-логики (Controller)	6	4
6	Настройка Kafka и обработка событий	3	5
7	Реализация механизмов аутентификации и авторизации	3	5
8	Проектирование пользовательского интерфейса (UI/UX)	3	2
9	Разработка клиентской части (представление, View)	7	8, 5
10	Интеграция клиента и сервера	3	9, 6
11	Разработка системы уведомлений через Socket.IO	2	10
12	Тестирование системы (функциональное, нагрузочное)	4	11
13	Разработка документации и оформление отчётов	2	12
14	Деплой и публикация проекта	1	13

Ресурсы, используемые в проекте, представлены на таблице 6.

Таблица 6 – Список ресурсов проекта

Ресурс	Тип ресурса	Ед. изм.	Стоимость за ед., Р	Описание
Ведущий разработчик	Трудовой	день	6 500	Разработка архитектуры, модели, API, Kafka, безопасность
Frontend-разработчик	Трудовой	день	5 000	Интерфейс, React, визуализация
UI/UX-дизайнер	Трудовой	день	4 000	Прототипы, макеты, юзабилити
Системный администратор	Трудовой	день	4 000	Настройка серверов, деплой
PostgreSQL сервер	Инфраструктурный	мес	8 000	Аренда облачного PostgreSQL на весь срок
Сервер приложений	Инфраструктурный	мес	12 000	Backend (Node.js, Kafka, Socket.IO)
Лицензии и ПО	Программный	ед.	10 000	Подписки, домен, SSL, инструменты

Общая продолжительность проекта составила 48 рабочих дней.

Общая сумма бюджетных затрат включает трудовые ресурсы, архитектуру и прочее.

Трудовые ресурсы:

- ведущий разработчик – $6\,500 \times 20$ дней = 130 000 Р;
- Frontend-разработчик – $5\,000 \times 15$ дней = 75 000 Р;
- UI/UX-дизайнер – $4\,000 \times 5$ дней = 20 000 Р;
- системный администратор – $4\,000 \times 3$ дня = 12 000 Р.

Инфраструктура и прочее:

- PostgreSQL сервер – $8\,000 \times 3$ мес = 24 000 Р;
- сервер приложений – $12\,000 \times 3$ мес = 36 000 Р.

Можно заметить, что для реализации проекта потребуется 307 000 рублей и 48 дней работы.

Диаграмма Ганта проекта представлена на рисунке 10.

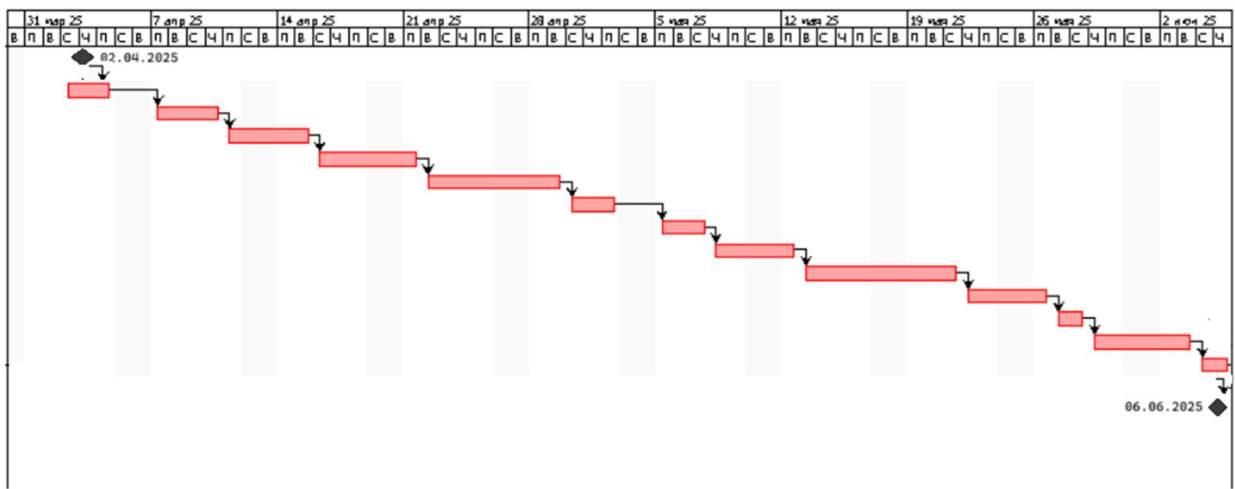


Рисунок 10 – Диаграмма Ганта

Итоговый отчёт по затратам позволяет убедиться, что предложенный вариант реализации программных средств экономически целесообразен и соответствует требованиям эффективности и рентабельности. Затраты на разработку, тестирование и внедрение находятся в пределах допустимого бюджета, что свидетельствует о рациональном распределении ресурсов.

2 Анализ требований и постановка задачи на разработку приложения для автоматизации задач оператора

2.1 Анализ функциональных требований

Анализ требований – часть процесса разработки программного обеспечения, включающая в себя сбор требований к программному обеспечению (ПО), их систематизацию, выявление взаимосвязей, а также документирование. Является частью общинженерной дисциплины «инженерия требований».

В процессе сбора требований важно принимать во внимание возможные противоречия требований различных заинтересованных лиц, таких как заказчики, разработчики или пользователи.

Полнота и качество анализа требований играют ключевую роль в успехе всего проекта. Требования к ПО должны быть документируемые, выполнимые, тестируемые, с уровнем детализации, достаточным для проектирования системы. Требования могут быть функциональными и нефункциональными.

Анализ требований включает три типа деятельности:

- сбор требований – общение с клиентами и пользователями, чтобы определить, каковы их требования для анализ предметной области;
- анализ требований – определение, являются ли собранные требования неясными, неполными, неоднозначными или противоречащими, решение этих проблем, выявление взаимосвязи требований;
- документирование требований – требования могут быть задокументированы в различных формах, таких как простое описание, сценарии использования, пользовательские истории, или спецификации процессов [7].

Функциональные требования – это конкретные руководящие принципы, описывающие поведение, функции и операции, которые должно выполнять программное обеспечение или система. В разработке программного

обеспечения и систем эти требования описывают, что система должна делать для удовлетворения потребностей пользователей и бизнес-целей. Функциональные требования обычно подробно излагаются в документации, например, в спецификации требований, чтобы обеспечить ясность и согласованность между группами разработчиков, заинтересованными сторонами и конечными пользователями [8].

В разработке программного обеспечения и систем функциональные требования определяют основные функции или действия, которые должна выполнять система. Они могут варьироваться от взаимодействия с пользователем, обработки данных, вычислений и интеграции с другими системами до шагов, необходимых для завершения рабочего процесса. Они отличаются от нефункциональных требований, которые фокусируются на системных атрибутах, таких как производительность, удобство использования и безопасность. Функциональные требования отвечают на вопросы «что» и «как» поведения системы, чтобы предоставить план для разработки и тестирования.

Функциональные требования имеют решающее значение для успеха продукта, поскольку они служат основой для процесса проектирования и разработки.

Для описания функциональных требований к системе была выбрана диаграмма вариантов использования.

Диаграмма вариантов использования в UML – диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Основное назначение диаграммы – описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему [9].

Функции, которые должна реализовывать система автоматизации задач оператора, представлены на диаграмме вариантов использования (рисунок 11).

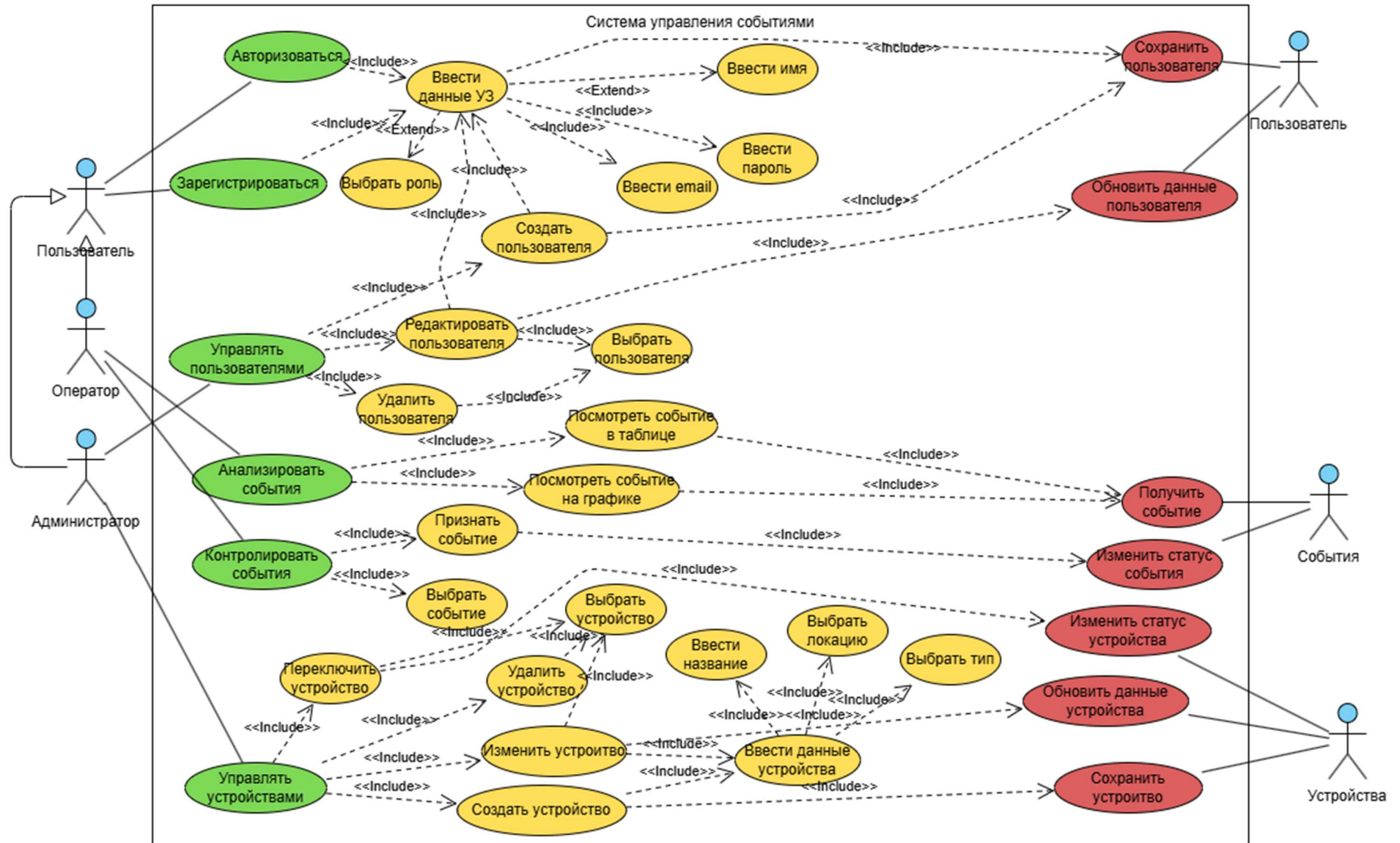


Рисунок 11 – Диаграмма вариантов использования системы автоматизации задач оператора

В таблице 7 отображено подробное описание сценария бизнес-процесса «Регистрация».

Таблица 7 – Сценарии бизнес-процесса «Регистрация»

Поле	Содержание
ID	BP-01
Описание	Регистрация нового пользователя в системе.
Действующие лица	Пользователь (гость)
Предусловие	Пользователь не зарегистрирован.
Основной поток	1. Ввести ФИО 2. Ввести Email 3. Ввести телефон 4. Ввести пароль 5. Ввести дату рождения 6. Если данные валидны, то добавить в БД 7. Войти в систему
Постусловие	Пользователь зарегистрирован и вошел в систему.

Процесс начинается с регистрации клиента, который вводит необходимые данные. Если данные валидны, то все введенные данные пользователя добавляются в базу данных, и он входит в систему. Если данные неверны, то пользователь вводит данные еще раз. Далее клиент может создавать, отменять и анализировать заказы.

В таблице 8 отображено подробное описание сценария бизнес-процесса «Авторизация».

Таблица 8 – Сценарии бизнес-процесса «Авторизация»

Поле	Содержание
ID	BP-02
Описание	Аутентификация пользователя в системе.
Действующие лица	Зарегистрированный пользователь
Предусловие	Учетная запись существует в базе данных.
Основной поток	1. Ввести логин и пароль 2. Проверка в базе данных 3. Пока данные неверны повторять ввод 4. Войти в систему
Постусловие	Пользователь успешно авторизован.

В начале пользователь вводит логин и пароль, которые сравниваются с данными в базе данных. Потом если данные верны, то пользователь входит в систему, если неверны, то возвращается ошибка и пользователь пытается еще раз.

В таблице 9 отображено подробное описание сценария бизнес-процесса «Управлять пользователями».

Таблица 9 – Сценарии бизнес-процесса «Управлять пользователями»

Поле	Содержание
ID	BP-03
Описание	Управление учетными записями пользователей через панель администратора.
Действующие лица	Администратор
Предусловие	Администратор авторизован в системе.
Основной поток	<ol style="list-style-type: none"> 1. Открыть форму создания 2. Ввести имя, email, пароль 3. Выбрать роль 4. Если данные валидны, то отправить на сервер 5. сохранить в БД 6. отобразить сообщение 7. показать ошибки
Постусловие	Пользователь создан и доступен для авторизации.

Процесс управления пользователями осуществляется администратором через панель администратора. Администратор имеет возможность создавать новых пользователей, редактировать их данные, назначать роли и управлять их правами доступа к функциональным модулям и объектам системы (например, устройствам, комнатам или событиям). При создании пользователя система требует ввести необходимые идентификационные данные и выбрать соответствующую роль (например, «Оператор»). Система валидирует введенные данные, проверяет уникальность email и сохраняет информацию в базе данных. При успешном создании пользователь получает возможность входа в систему и выполнения разрешённых операций в зависимости от своей роли. Процесс также охватывает редактирование и удаление существующих пользователей, с обязательной проверкой прав доступа и сохранением истории изменений для последующего аудита.

Управление учетными записями критически важно для обеспечения безопасности, повышения эффективности и оптимизации работы в организации.

В таблице 10 отображено подробное описание сценария бизнес-процесса «Анализировать события».

Таблица 10 – Сценарии бизнес-процесса «Анализировать события»

Поле	Содержание
ID	BP-04
Описание	Анализ событий, поступающих от устройств, с применением фильтров.
Действующие лица	Оператор, администратор
Предусловие	Пользователь имеет доступ к событиям и устройствам.
Основной поток	<ol style="list-style-type: none"> 1. Открыть дашборд событий 2. Установить фильтры 3. Система отправляет запрос и возвращает данные 4. Пользователь анализирует параметры события
Постусловие	События проанализированы, информация учтена.

Оператор или администратор получает доступ к панели мониторинга, где отображаются события, поступившие от устройств. События могут иметь различные уровни критичности, такие как, информационные, предупреждения, ошибки и критические ошибки. Пользователь может применять фильтры по времени, устройствам, уровням и другим параметрам, чтобы сузить область анализа. При выборе события отображаются его параметры, такие как, значение, единица измерения, пороговое значение и метка времени. Анализ событий позволяет выявить отклонения от нормы.

В таблице 11 отображено подробное описание сценария бизнес-процесса «Управлять событиями».

Таблица 11 – Сценарии бизнес-процесса «Управлять событиями»

Поле	Содержание
ID	BP-05
Описание	Пометка событий как признанных для учёта в дальнейшей обработке.
Действующие лица	Оператор, администратор
Предусловие	События отображаются в интерфейсе, пользователь авторизован.
Основной поток	<ol style="list-style-type: none"> 1. Получить список событий 2. Выбрать события 3. Признать событие 4. Система обновляет статус
Постусловие	События помечены как признанные, лог действий сохранён.

События, поступающие в систему, могут требовать дополнительной обработки или классификации. Пользователи, могут выполнять над событиями административные действия, такие как пометить событие как признанное или скрывать его из активного списка. Эти действия фиксируются системой, включая пользователя, выполнившего действие, и временную метку. Подобный подход обеспечивает прозрачность реагирования, помогает в расследованиях и разборе инцидентов. Управление событиями позволяет организовать эффективный поток работ по инцидентам и минимизировать потери от простоев или аварий.

В таблице 12 отображено подробное описание сценария бизнес-процесса «Управлять устройствами».

Таблица 12 – Сценарии бизнес-процесса «Управлять устройствами»

Поле	Содержание
ID	BP-06
Описание	Добавление, редактирование и назначение прав на устройство.
Действующие лица	Администратор
Предусловие	Пользователь имеет административные права.
Основной поток	1. Открыть окно создания 2. Ввести имя, тип, комнату, статус 3. Назначить пользователей 4. Если данные валидны, то сохранить в БД 5.отобразить в дашборде
Постусловие	Устройство добавлено и отображается в интерфейсе.

Администратор имеет доступ к модулю управления устройствами, где может добавлять новые устройства, редактировать параметры существующих, а также управлять их связью с комнатами и пользователями. При добавлении устройства необходимо указать его уникальное имя, тип (датчик, контроллер и так далее), расположение (комната или зона), текущий статус и список пользователей, которые имеют право на доступ или управление этим устройством. Система проверяет корректность ввода и наличие обязательных полей. После успешного сохранения устройство становится доступным в системе, отображается на дашборде и может принимать события через Kafka или другие каналы. Управление устройствами является основным звеном в

администрировании IoT-инфраструктуры и обеспечивает привязку физических объектов к логической модели системы.

Также была составлена спецификация бизнес-операций по бизнес процессам отображенная в таблице 13.

Таблица 13 – Спецификация бизнес операций для пользователя системы

Название	Бизнес-процесс	Описание
1	2	3
Ввести данные УЗ	Авторизация	Вводится все данные пользователей
Ввести имя	Авторизация	Вводится имя пользователя
Ввести email	Авторизация	Вводится email пользователя
Ввести пароль	Авторизация	Вводится пароль пользователя
Выбрать роль	Авторизация	Выбирается роль пользователя
Создать пользователя	Управлять пользователями	Создается учетная запись нового пользователя
Редактировать пользователя	Управлять пользователями	Редактируются данные выбранного пользователя администратором
Выбрать пользователя	Управлять пользователями	Выбирается пользователь из списка для выполнения дальнейших действий
Удалить пользователя	Управлять пользователями	Пользователь помечается как удаленный
Посмотреть события в таблице	Анализировать события	Пользователь может посмотреть текущие и архивные события в таблице
Посмотреть события на графике	Анализировать события	Пользователь может посмотреть текущие и архивные события на графике
Выбрать событие	Контролировать события	Выбирается событие из списка для дальнейшей обработки
Признать событие	Контролировать события	Событие меняет статус на признанное
Выбрать устройство	Управлять устройствами	Выбирается устройство из списка для дальнейшей обработки
Создать устройство	Управлять устройствами	Создается новое устройство с введенными данными
Изменить устройство	Управлять устройствами	Обновляются данные устройства
Ввести данные устройства	Управлять устройствами	Пользователь вводит данные устройства
Ввести название	Управлять устройствами	Пользователь вводит название устройства

Продолжение таблицы 13

1	2	3
Выбрать локацию	Управлять устройствами	Пользователь выбирает локацию устройства из списка
Выбрать тип	Управлять устройствами	Пользователь выбирает тип устройства
Удалить устройство	Управлять устройствами	Устройство меняет состояние на удаленное
Переключить устройство	Управлять устройствами	Устройство меняет состояние на выключено, если было включено и на включено, если было выключено

Таким образом, диаграмма вариантов использования системы автоматизации задач оператора позволяет детально отобразить все возможные взаимодействия пользователя с системой, структурировать и упорядочить функциональные требования.

2.2 Анализ нефункциональных требований

Нефункциональные требования – требования, определяющие свойства, которые система должна демонстрировать, или ограничения, которые она должна соблюдать, не относящиеся к поведению системы. Например, производительность, удобство сопровождения, расширяемость, надежность, факторы эксплуатации.

Нефункциональные требования (NFR) определяют стандарты и качества, которым должна соответствовать система для эффективной работы, уделяя особое внимание – это система работает, а не почему это так. В то время как функциональные требования определяют конкретные задачи или поведение программной системы, NFR описывают такие качества, как скорость, надежность и безопасность, которые определяют производительность, удобство использования и устойчивость системы.

Нефункциональные требования формируют то, как пользователи воспринимают систему. Например, отзывчивый интерфейс (требование производительности) и интуитивная навигация (требование удобства использования) значительно повышают удовлетворенность пользователей. NFR, связанные со стабильностью, такие как надежность и

отказоустойчивость, сокращают время простоя и предотвращают раздражающие сбои. Требования масштабируемости гарантируют, что система может справляться с возросшими нагрузками без ухудшения производительности, что имеет решающее значение для приложений, ориентированных на рост.

Нефункциональные требования часто диктуют архитектуру и выбор технологий, направляя разработчиков на выбор фреймворка, оптимизацию базы данных и конфигурацию сервера. Планирование ресурсов вокруг NFR помогает распределять бюджет и персонал для эффективного соответствия стандартам высокой производительности или безопасности, тем самым избегая дорогостоящих переделок или модернизаций в дальнейшем. Таким образом, NFR являются неотъемлемой частью достижения целей проекта и сроков [10].

Нефункциональные требования системы автоматизации задач оператора:

- 1) производительность;
- 2) масштабируемость;
- 3) надёжность;
- 4) безопасность;
- 5) поддерживаемость;
- 6) удобство использования;
- 7) совместимость.

Производительность – система должна обрабатывать входящие события в реальном времени с задержкой не более 1 секунды между поступлением события и его отображением в пользовательском интерфейсе. Kafka-консьюмер должен обрабатывать не менее 1000 сообщений в секунду при средней нагрузке. Пользовательский интерфейс должен загружаться за ≤ 2 секунд при нормальной скорости интернета (10 Мбит/с).

Масштабируемость – архитектура системы должна поддерживать горизонтальное масштабирование как для компонентов сервера (Kafka,

сервисы, БД), так и клиента (через CDN и балансировщики). Модульная система должна позволять добавление новых модулей без изменения существующего кода и с минимальной перезагрузкой сервиса.

Надёжность – система должна обеспечивать $\geq 99.9\%$ доступности в месяц. Все ключевые компоненты (Kafka, PostgreSQL, backend API) должны иметь механизмы резервирования и автоматического восстановления. При потере соединения с Kafka, система должна автоматически пере подключаться и восстанавливать потребление с последнего смещённого оффсета.

Безопасность – аутентификация осуществляется через JWT с использованием HTTPS. Авторизация реализована по RBAC. Каждый пользователь имеет ограниченный доступ только к тем устройствам, комнатам и событиям, которые ему назначены. Все действия пользователя логируются с указанием времени, IP, user-agent и результата операции. Доступ к административным функциям (например, управление пользователями, настройками системы) ограничен только пользователями с ролью admin.

Поддерживаемость – код на клиенте и на сервере должен быть написан на строго типизированных языках (TypeScript/Node.js) и сопровождаться документацией. Интерфейсы модулей и структур данных должны быть описаны в виде контрактов (например, через JSON Schema или интерфейсы). Настройки темы, языка и поведения UI должны храниться централизованно и быть легко изменяемыми.

Удобство использования (Usability) – пользовательский интерфейс должен быть отзывчивым (responsive) и адаптированным под рабочие экраны (1280px–1920px). Уведомления об ошибках, событиях и системных статусах должны отображаться в пределах 200 миллисекунд от их получения. Все действия должны сопровождаться визуальной обратной связью (спиннер, тост, модальное окно).

Совместимость – система должна корректно работать в актуальных версиях браузеров, например Chrome, Edge, Firefox. Поддерживаются

WebSocket и HTTP/2 для взаимодействия с сервером. Все внешние библиотеки должны быть с открытым исходным кодом.

2.3 Анализ требований к пользовательскому интерфейсу

Интерфейс пользователя, он же пользовательский интерфейс (UI – англ. user interface) – интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы [11].

Веб-интерфейс – веб-страница или совокупность веб-страниц, предоставляющая пользовательский интерфейс для взаимодействия с сервисом или устройством посредством протокола HTTP и веб-браузера. Веб-интерфейсы получили широкое распространение в связи с ростом популярности всемирной паутины и соответственно – повсеместного распространения веб-браузеров.

Одним из основных требований к веб-интерфейсам является их одинаковый внешний вид и одинаковая функциональность при работе в различных браузерах.

Классическим и наиболее популярным методом создания веб-интерфейсов является использование HTML с применением CSS и JavaScript. Однако различная реализация HTML, CSS, DOM и других спецификаций в браузерах вызывает проблемы при разработке веб-приложений и их последующей поддержке[12].

Для описания пользовательского интерфейса системы была выбрана диаграмма навигации.

Назначение схемы навигации – выразить основные пути пользовательского интерфейса в системе. Это основные пути через экраны системы, но не обязательно все возможные пути. Ее можно представить, как карту дорог пользовательского интерфейса системы.

Схема навигации служит в качестве фона и связи между отдельными Раскадровками. Раскадровки описывают навигацию перемещений

пользователя по элементам пользовательского интерфейса для выполнения системных функций, а схема навигации определяет допустимые пути навигации.

Схема навигации облегчает оценку того, сколько «щелчков» должен сделать пользователь, чтобы достичь определенного экрана [13].

Схема навигации web-приложения изображена на рисунке 12.

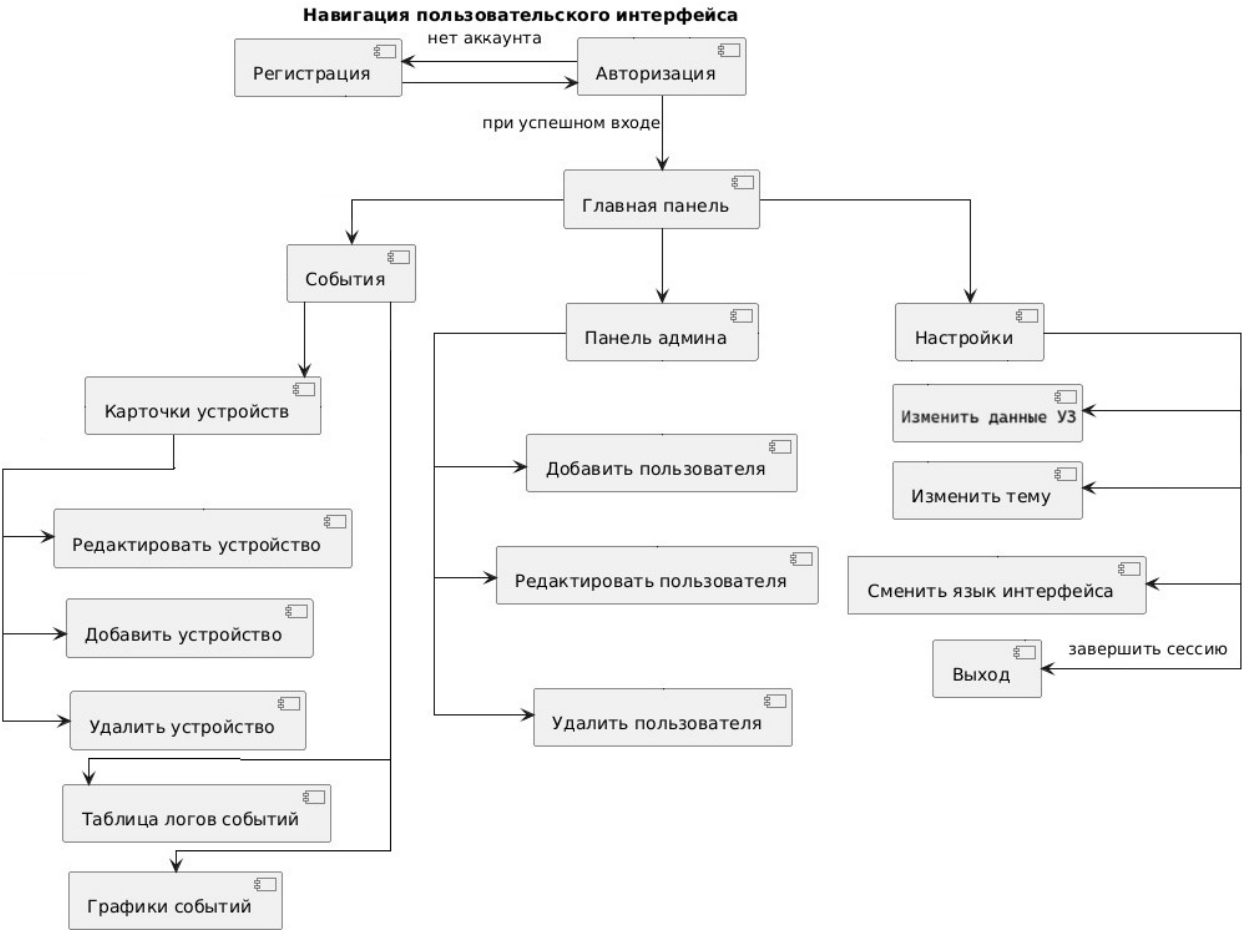


Рисунок 12 – Схема навигации пользовательского интерфейса

Диаграмма навигации, описывает логическую структуру пользовательского интерфейса программной системы, ориентированной на управление событиями, пользователями и устройствами в рамках информационно-аналитической IoT-платформы. Навигация моделирует переходы между ключевыми пользовательскими экранами и действиями, отражая иерархию представления и сценарии взаимодействия пользователя с системой.

Страница «Регистрация» позволяет пользователю ввести и отправить данные для первичного создания учётной записи. Включает поля, обязательные для идентификации пользователя (имя, email, пароль). При успешной валидации данных выполняется переход на главную панель.

Страница «Авторизация» реализует механизм входа в систему с использованием пары «логин–пароль» и токенов (JWT). При подтверждении подлинности пользователя происходит переход к основной точке входа в систему – главной панели.

Страница «Главная панель» Выступает центральной навигационной точкой системы, предоставляющей доступ к ключевым подсистемам, таким как событийно-аналитическому дашборду, административному модулю управления пользователями, а также настройкам пользовательского интерфейса.

Переходы возможны к следующим страницам:

- события;
- панель администратора;
- настройки.

Страница «События» реализует обзор всех событий, поступающих в систему от устройств (например, через Kafka). Включает инструменты фильтрации, сортировки и навигации по событиям. Содержит графики событий – отображают временные тренды показателей устройств, аномалии и пики по параметрам (например, температура, напряжение, и др.), карточки устройств – интерактивные представления устройств с текущими показателями и статусами, таблица логов событий – структурированный журнал всех поступивших сообщений с метаданными (время, устройство, уровень важности, параметры).

Карточки устройств содержат расширенные карточки всех зарегистрированных устройств, позволяющие осуществлять следующие действия:

- 1) добавить устройство – открывает модальное окно или страницу с формой ввода (имя, тип, комната, назначенные пользователи);
- 2) редактировать устройство – даёт возможность модификации параметров устройства;
- 3) удалить устройство – удаляет устройство с подтверждением действия.

Эти функции обеспечивают полную жизненную модель объектов мониторинга в системе.

Панель администратора доступна только пользователям с ролью администратора. Предоставляет интерфейс для управления учётными записями системы. Включает действия:

- добавить пользователя;
- редактировать пользователя;
- удалить пользователя.

Каждое действие сопровождается валидацией данных, проверкой прав доступа и регистрацией действия в логах.

Страница «Настройки» содержит пользовательские параметры отображения интерфейса:

- 1) поменять данные учетной записи – позволяет обновить имя, логин и пароль пользователя;
- 2) изменить тему – позволяет переключение между светлой и тёмной темой;
- 3) сменить язык интерфейса – выбор доступных языков для мультиязычной локализации;
- 4) выход – завершает сессию, удаляя токен доступа и возвращая пользователя на страницу авторизации.

Данная архитектура интерфейса отражает модульную и ролевую организацию интерфейса информационной системы. Она обеспечивает чёткое разграничение функциональности по пользовательским ролям и реализует

принципы удобства, безопасности и масштабируемости. Все страницы связаны логически и поддерживают целостный пользовательский путь, начиная с регистрации и заканчивая выполнением прикладных функций системы (анализ, контроль, управление, настройка).

2.4 Анализ требований к аппаратно-программному обеспечению

2.4.1 Требования к аппаратному обеспечению

Аппаратное обеспечение – совокупность физических частей компьютера, включая центральный процессор (ЦП), оперативная память (ОЗУ), материнская плата, ПЗУ с микропрограммами, внутренние накопители (жёсткие диски, SSD, ленточные накопители, дисководы дискет и оптических дисков и так далее), графическая карта, звуковая карта и компьютерный корпус. Оно также включает в себя внешние устройства, такие как монитор, принтер или МФУ, компьютерную мышь, клавиатуру, мобильные и аварийные источники питания (ИБП) и прочие.

Аппаратное обеспечение обычно направляется программным обеспечением для выполнения любой команды или инструкции. Комбинация аппаратного и программного обеспечения образует пригодную для использования вычислительную систему, хотя существуют и другие системы, состоящие только из аппаратного обеспечения (простейшие из них – счётные палочки и счёты) [14].

Развёртывание и эффективная эксплуатация информационно-аналитического приложения, использующего клиент-серверную архитектуру с поддержкой обработки событий в реальном времени, предъявляет определённые требования к аппаратной конфигурации сервера. Эти требования определяются объёмом данных, интенсивностью взаимодействия между компонентами системы (включая Web-клиент, API-сервер, брокер сообщений и СУБД), а также числом одновременных пользователей.

Минимальные параметры обеспечивают базовую работоспособность системы в среде разработки или в условиях ограниченной нагрузки (до 5 одновременных пользователей, низкая частота поступления событий):

- 1) центральный процессор (CPU) – 4 физических ядра с архитектурой x86–64, тактовая частота не ниже 2.0;
- 2) оперативная память (RAM) – 8 ГБ DDR4 (двухканальный режим рекомендуется, но не обязателен);
- 3) накопитель (Storage) – твердотельный накопитель (SSD), объём не менее 128 ГБ (рекомендуется NVMe-накопитель для снижения латентности при работе с базой данных и логами событий);
- 4) сетевой адаптер – 1 Гбит/с Ethernet или сопоставимый по пропускной способности интерфейс.

Рекомендуемая конфигурация предназначена для стабильной работы приложения в продуктивной среде с умеренной и высокой нагрузкой (до 100 одновременных пользователей, поступление событий в режиме реального времени, интенсивная запись и чтение из базы данных):

- 1) центральный процессор (CPU) – 8 физических ядер / 16 потоков, архитектура x86–64, с поддержкой виртуализации и многозадачности, частота от 2.6 ГГц;
- 2) оперативная память (RAM) – от 32 ГБ DDR4 с возможностью масштабирования (особенно при использовании in-memory кэширования или хранения промежуточных данных в оперативной памяти);
- 3) накопитель (Storage) – SSD NVMe, объём от 512 ГБ, поддержка скоростей чтения/записи не менее 2500 МБ/с;
- 4) сетевой адаптер – высокопроизводительный 1–10 Гбит/с.

Дополнительные рекомендации по аппаратной инфраструктуре:

- резервное питание;
- использование серверных корпусов с резервируемыми вентиляторами и контролем температуры;

– поддержка аппаратного масштабирования (например, возможность установки дополнительных модулей RAM, второго CPU, дополнительных накопителей).

Минимальная конфигурация ориентирована на разработку и тестирование, а рекомендованная – на эксплуатацию в продуктивной среде. Аппаратные ресурсы должны обеспечивать своевременную обработку входящих событий, устойчивое хранение данных и отзывчивость пользовательского интерфейса без деградации производительности при пиковых нагрузках.

2.4.2 Требования к системному программному обеспечению

Настоящий раздел определяет перечень системного программного обеспечения, необходимого для установки, конфигурации, эксплуатации и поддержки информационной системы. Приведены также требования к среде конечного пользователя (клиентской части) и браузерной совместимости.

Для обеспечения стабильности, совместимости с контейнеризацией и инструментами DevOps, рекомендуется использование дистрибутивов операционных систем, таких как Linux Ubuntu версии 20.04 LTS и выше, Debian версии 11 и выше или Windows Server 2022.

Для средств виртуализации и контейнеризации лучше использовать Docker Engine версии 20.10 и выше для контейнеризации компонентов приложения и Kubernetes версии 1.25 и выше для масштабируемых/кластерных установок.

Для системы управления базами данных нужно PostgreSQL версии 13 и выше. Используется в качестве основной реляционной БД для хранения данных системы.

Брокеры сообщений Apache Kafka версии 3.0 и выше для асинхронной обработки событий от устройств и нотификации пользователей в режиме реального времени.

Также браузеры для пользовательского интерфейса, такие как Google Chrome версии 110 и выше, Mozilla Firefox, Microsoft Edge, Safari для macOS или Opera.

Соответствие описанным требованиям к программной инфраструктуре является условием корректной работы всех функциональных и нефункциональных компонентов системы.

2.4.3 Требования к средствам разработки

Разработка системы предполагает использование современного стека технологий, обеспечивающего высокую степень модульности, масштабируемости, кроссбраузерности и удобства сопровождения. Ниже представлены основные инструменты и технологии, используемые на различных этапах жизненного цикла программного обеспечения, от проектирования интерфейсов до реализации бизнес-логики и взаимодействия с внешними сервисами.

Языки программирования и технологии разработки:

- 1) HTML5 – язык разметки для создания структуры веб-страниц пользовательского интерфейса, поддерживает семантические теги, адаптивную верстку, мультимедийные элементы;
- 2) CSS3 – таблица стилей, используемый для визуального оформления элементов, в проекте применяется утилитарный CSS-фреймворк (Tailwind CSS) для ускорения разработки и унификации внешнего вида;
- 3) JavaScript (ES6+) и TypeScript – основные языки программирования клиентской логики, используется для работы с DOM, событийной моделью, сетевыми запросами и подключением сторонних библиотек;
- 4) React.js – библиотека для построения компонентно-ориентированных пользовательских интерфейсов, обеспечивает реактивное обновление данных и поддержку SPA (Single Page Application);

5) Tailwind CSS – утилитарный CSS-фреймворк, позволяющий эффективно создавать адаптивные интерфейсы без необходимости написания кастомных классов;

6) Socket.IO – библиотека для реализации взаимодействия в реальном времени между клиентом и сервером.

Среды разработки и вспомогательные инструменты:

1) Visual Studio Code (VSCode) – основная среда разработки (IDE), обеспечивающая интеграцию с Git, отладку, автодополнение и форматирование кода;

2) Figma – средство проектирования пользовательских интерфейсов. Используется для создания wireframe-макетов, прототипов и передачи спецификаций разработчикам;

3) Postman – тестирование REST API, отладка запросов и анализ ответов сервера;

4) Docker – контейнеризация среды разработки, обеспечение изоляции зависимостей, автоматизация запуска многокомпонентных сервисов;

5) Node.js + npm – установка зависимостей, запуск сборки, линтинг, трансляция TypeScript и управление жизненным циклом проекта;

6) Git + GitHub – система управления версиями, обеспечивающая совместную разработку, контроль изменений и CI/CD-процессы;

7) Prettier – инструменты автоматического форматирования кода в соответствии с заданным стилевым руководством.

Перечисленный технологический стек и инструменты обеспечивают создание надёжной, расширяемой и поддерживаемой информационной системы. Использование типизированного языка (TypeScript), компонентной архитектуры (React), современных средств командной разработки (Git, Docker) позволяет достигнуть высокой производительности, безопасности и удобства сопровождения.

2.5 Постановка задачи на разработку приложения

В данном выпускной квалификационной работе рассматривается структура и функционирование системы Информационная система мониторинга и управления IoT-устройствами.

Разработка приложения обусловлена необходимостью создания гибкой и масштабируемой программной платформы, предназначенной для сбора, обработки и визуализации событий от распределённых устройств, централизованного управления пользователями и правами доступа, обеспечения мониторинга в реальном времени с возможностью фильтрации и анализа данных.

Система ориентирована на промышленные, инфраструктурные и исследовательские сценарии, где требуется отображение текущего состояния объектов (устройств) и обработка критических событий.

Целью разработки является создание web-приложения с администраторской и пользовательской частью, обеспечивающего:

- 1) безопасную авторизацию и разграничение прав доступа;
- 2) регистрацию, просмотр и редактирование устройств и их параметров;
- 3) обработку событий и оповещений от устройств через брокер сообщений;
- 4) ведение логов и истории событий;
- 5) возможность масштабирования и расширения функционала через модульную архитектуру;
- 6) визуальное представление данных в виде виджетов, таблиц и графиков.

Основной целью является создать эффективное и удобное в использовании приложение для пользователя системы управления IoT-устройствами, которое позволит оптимизировать процессы создания и управления устройствами и событиями.

3 Разработка программных средств приложения для автоматизации задач оператора

3.1 Разработка архитектуры развертывания программных средств

Архитектура программного обеспечения (англ. software architecture)—совокупность важнейших решений об организации программной системы. Архитектура включает:

- выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
- соединение выбранных элементов структуры и поведения во всё более крупные системы;
- архитектурный стиль, который направляет всю организацию – все элементы, их интерфейсы, их сотрудничество и их соединение [15].

Для отображения архитектуры программных средств была выбрана диаграмма развертывания.

Диаграмма развёртывания (англ. Deployment diagram) в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт, диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

Целью диаграммы развертывания является представление статического снимка архитектуры развертывания и сетевой топологии. Полное описание системы, скорее всего, будет содержать ряд различных диаграмм развертывания, каждая из которых будет сосредоточена на отдельном аспекте управления системой. Для многих из этих представлений диаграмма

развертывания объединяется с диаграммой компонентов, чтобы показать связь между оборудованием и программным обеспечением. Например, для многоуровневого приложения диаграмма развертывания будет моделировать распределение уровней приложения, их физические соединения и их логические пути связи [16].

Для визуализации физической архитектуры, как приложение развёрнуто на серверах, какие узлы участвуют, как они взаимодействуют между собой, была сделана диаграмма развертывания на рисунке 13.

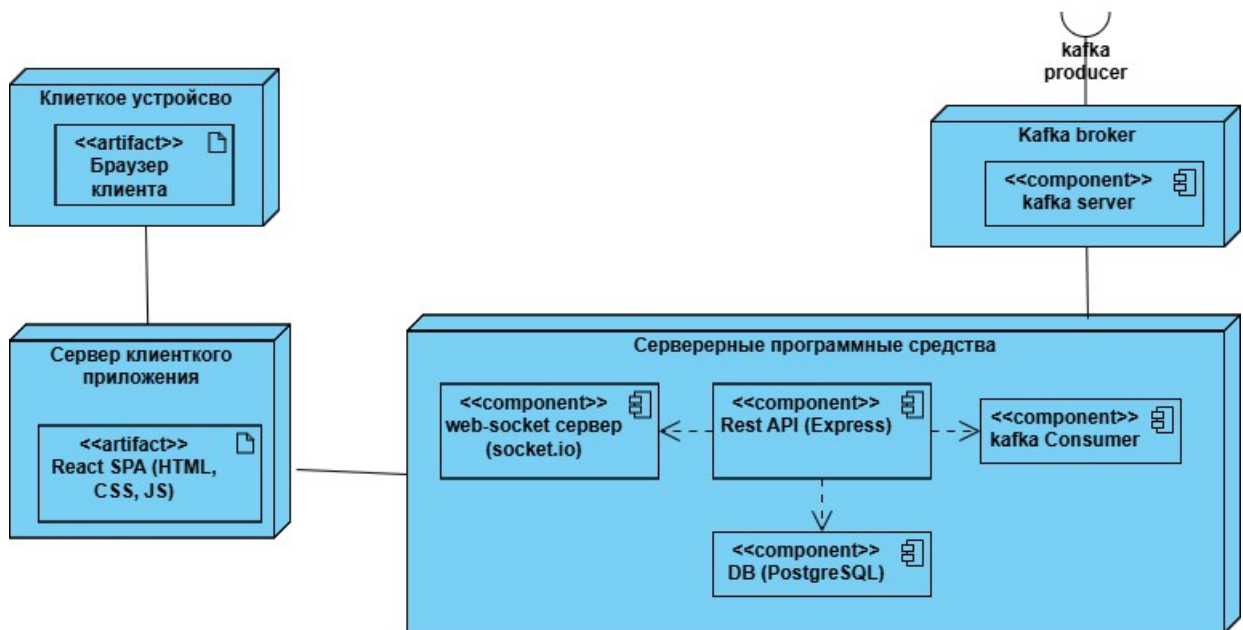


Рисунок 13 – Архитектура приложения

Разрабатываемая система представляет собой многоуровневое web-приложение, реализующее взаимодействие между клиентской и серверной частью, брокером сообщений и реляционной базой данных.

Пользовательский интерфейс реализован как одностраничное приложение (SPA) на основе React, собранное в виде статических файлов HTML/CSS/JS. Эти файлы размещаются на отдельном frontend-сервере.

Пользовательское приложение осуществляет:

- 1) отправку HTTP-запросов к REST API;
- 2) подключение к WebSocket для получения уведомлений в реальном времени;

3) загрузку в браузере из frontend-сервера.

Серверная часть развёртывается на Node.js выполняет функции:

- обрабатывает HTTP-запросы от клиента;
- взаимодействует с базой данных PostgreSQL;
- обеспечивает соединения в реальном времени через WebSocket сервер (Socket.IO);
- отправляет уведомления через WebSocket при возникновении событий.

Kafka Consumer подписан на определённые Kafka-топики, получает сообщения от брокера Kafka, выполняет валидацию и сохраняет события в PostgreSQL, по результатам – инициирует отправку данных через WebSocket клиентам.

База данных (PostgreSQL) используется как основное хранилище данных и работает напрямую через SQL/Query Builder.

Kafka Broker принимает сообщения от устройств (или имитаторов), используется как канал для событийной архитектуры, передаёт сообщения в Kafka Consumer, развернутый на backend-сервере.

Взаимодействие компонентов происходит следующим образом:

- 1) пользователь открывает интерфейс в браузере, загружаемый с frontend-сервера;
- 2) клиент инициирует вход в систему через REST API, получает JWT-токен;
- 3) клиент начинает WebSocket-соединение для получения событий в реальном времени;
- 4) устройства или симуляторы отправляют данные в Kafka;
- 5) kafka Consumer на backend получает сообщение, анализирует его и сохраняет в PostgreSQL;
- 6) уведомляет связанных с устройством, на которое пришло событие, пользователей через WebSocket;

- 7) пользователь на клиенте видит новое событие или оповещение;
- 8) все действия логируются и могут быть просмотрены в таблице логов или графически.

Архитектура приложения обеспечивает надёжную маршрутизацию данных от устройств до интерфейса пользователя. При этом взаимодействие между компонентами организовано по принципу событийно-ориентированной и модульной модели. Такая структура подходит для систем с умеренной нагрузкой, требующих высокой гибкости при развёртывании и отладке.

3.2 Разработка сценариев взаимодействия пользователя с компонентами приложения

Для отображения сценариев взаимодействия пользователя с компонентами приложения была выбрана диаграмма последовательности.

Диаграмма последовательности (англ. sequence diagram) – UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие акторов (действующих лиц) информационной системы в рамках прецедента.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни» (англ. lifeline), отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни» – фокусы контроля, англ. focus of control), и стрелки, показывающие обмен сигналами или сообщениями между объектами [17].

Диаграммы последовательностей используются для более детального описания логики сценариев использования. Это отличное средство документирования проекта с точки зрения сценариев использования!

Диаграмма последовательности использования системы пользователем изображена на рисунке 14.

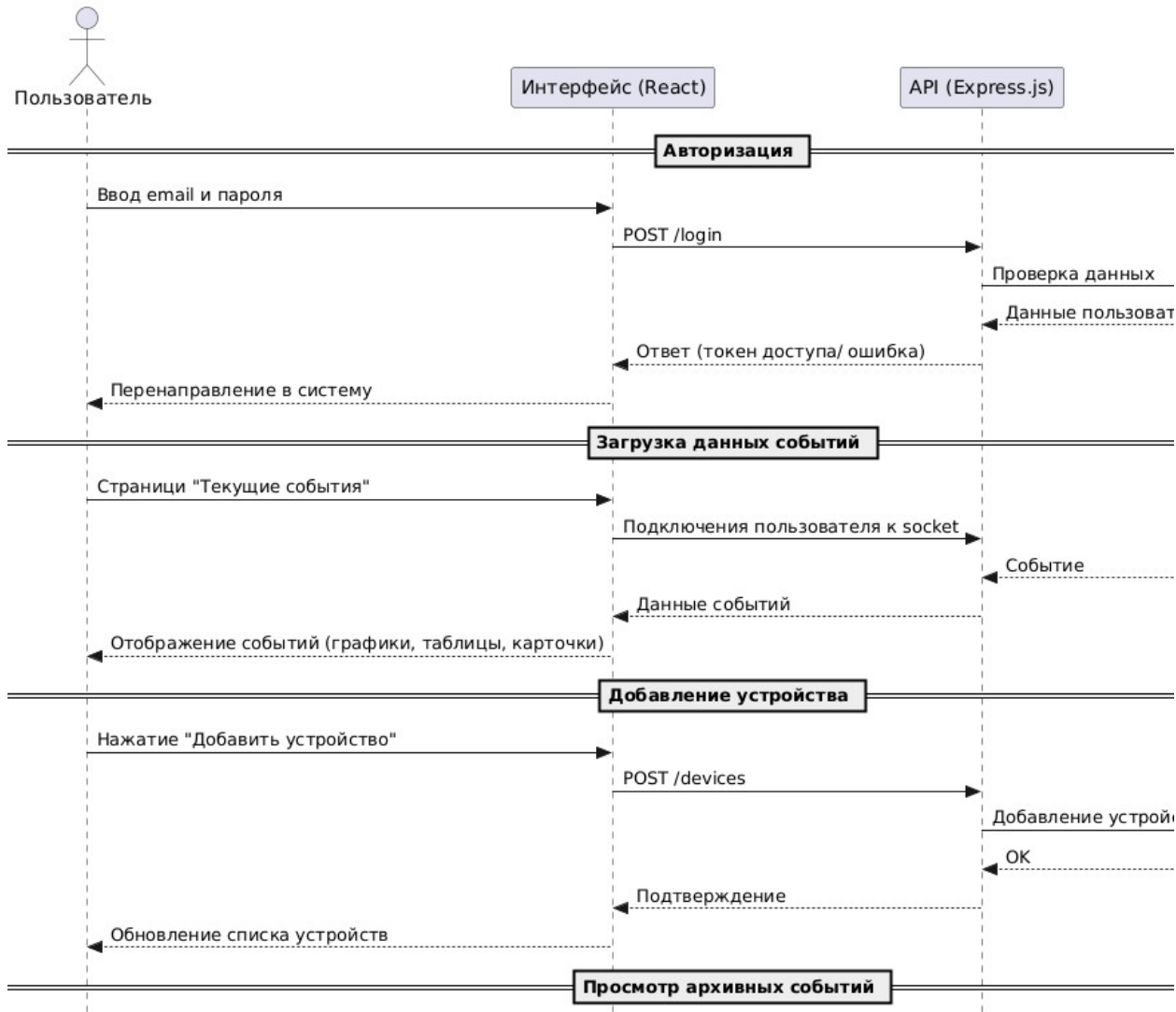


Рисунок 14 – Диаграмма последовательности использования системы

Диаграмма последовательности моделирует последовательность сообщений между основными участниками архитектуры веб-приложения, такими как пользователь, пользовательский интерфейс (React), серверный API (Express.js), база данных, а также внешняя система передачи событий (Kafka). Диаграмма демонстрирует как синхронные (HTTP-запросы), так и асинхронные взаимодействия (через WebSocket и Kafka).

В сценарии авторизации – пользователь вводит свои учетные данные в интерфейсе. Интерфейс (UI) отправляет запрос POST /login на серверный API. API обращается к базе данных для проверки данных пользователя. После

проверки, база возвращает информацию об учетной записи (или сообщает об ошибке). API возвращает на клиент либо JWT-токен доступа, либо сообщение об ошибке. UI в случае успеха перенаправляет пользователя в защищённую часть приложения. Целью является аутентифицировать пользователя и получить токен доступа для последующих запросов.

В сценарии загрузка данных текущих событий – после входа пользователь переходит на страницу «Текущие события». UI инициирует подключение к WebSocket-соединению через API для получения событий в реальном времени. Kafka как промежуточный брокер направляет события в API. API обрабатывает события и передаёт их в UI. UI визуализирует данные для пользователя в виде графиков, таблиц и карточек устройств.

Особенностью является использование асинхронного взаимодействия через сокеты для обновлений в реальном времени.

В сценарии добавления устройства – пользователь инициирует добавление устройства через форму. UI отправляет POST /devices с данными нового устройства. API передаёт данные в базу данных для сохранения. После успешного добавления база данных возвращает подтверждение. API информирует UI о результате. UI обновляет интерфейс, добавляя новое устройство в список.

В сценарии просмотра архивных событий – пользователь выбирает период в интерфейсе. UI отправляет GET запрос на API. API обращается к базе данных для выборки исторических событий. База возвращает агрегированные данные. API передаёт их обратно UI. UI отображает информацию в графическом и табличном виде.

Назначение сценария – предоставление доступа к историческим данным с возможностью аналитики.

Диаграмма отражает четыре ключевых сценария работы пользователя с системой:

- 1) авторизация;
- 2) получение событий в реальном времени;

- 3) управление устройствами;
- 4) анализ архивных данных.

Она демонстрирует комбинацию синхронных REST-запросов и асинхронных механизмов, подчеркивая роль каждого компонента в архитектуре. Такая модель позволяет обеспечить высокую интерактивность, масштабируемость и управляемость пользовательского интерфейса.

3.3 Разработка пользовательского интерфейса приложения

3.3.1 Разработка структуры пользовательского интерфейса

Структура пользовательского интерфейса (UI) определяет организацию и расположение всех визуальных элементов, с которыми пользователь взаимодействует при использовании приложения или сайта. Она включает в себя макет, навигацию, визуальные элементы (кнопки, иконки, поля ввода) и другие компоненты, обеспечивающие удобство и эффективность взаимодействия [18].

Основные элементы структуры пользовательского интерфейса:

- макет (Layout) – определяет расположение и организация элементов на экране, как контент, навигация и другие компоненты выравниваются и располагаются относительно друг друга;
- навигация – обеспечивает перемещение пользователя по приложению или сайту, включает меню, кнопки, ссылки и другие элементы, позволяющие пользователю легко находить нужную информацию;
- визуальные элементы – все графические компоненты, с которыми пользователь взаимодействует, такие как кнопки, иконки, поля ввода, переключатели, ползунки и другие.

Принципы проектирования UI структуры:

- простота и понятность;
- удобство использования;
- единообразие;

- обратная связь;
- эффективность.

Простота и понятность – интерфейс должен быть легким для понимания и использования, даже для неопытных пользователей.

Удобство использования – интерфейс должен быть интуитивно понятным и обеспечивать комфортное взаимодействие.

Единообразие – использование единого стиля и принципов оформления во всем интерфейсе.

Обратная связь – предоставление пользователю информации о текущем состоянии системы и результате его действий.

Эффективность – пользователь должен иметь возможность быстро и легко находить нужную информацию и выполнять задачи.

Разработка пользовательского интерфейса включает в себя следующие этапы:

1. анализ целевой аудитории – определение демографических характеристик, поведения и предпочтений пользователей;
2. конкурентный анализ – изучение интерфейсов конкурентов для выявления сильных и слабых сторон;
3. создание прототипов – разработка макетов и прототипов интерфейса для тестирования и доработки;
4. тестирование – проведение юзабилити-тестирования для оценки удобства и эффективности интерфейса;
5. реализация – разработка и внедрение окончательного варианта интерфейса.

Для отображения структуры приложения была выбрана диаграмма компонентов.

Диаграмма компонентов (Component Diagram) в UML – это структурная диаграмма, которая отображает организацию и взаимосвязи компонентов в

системе. Она показывает, как система разделена на компоненты и как эти компоненты взаимодействуют между собой [19].

Диаграмма компонентов является важным инструментом для проектирования и понимания архитектуры сложных программных систем. Она позволяет визуализировать структуру системы, ее компоненты и взаимосвязи между ними, что способствует более эффективной разработке и поддержке.

Диаграмма компонентов, представляющая структуру пользовательского приложения изображена на рисунке 15.

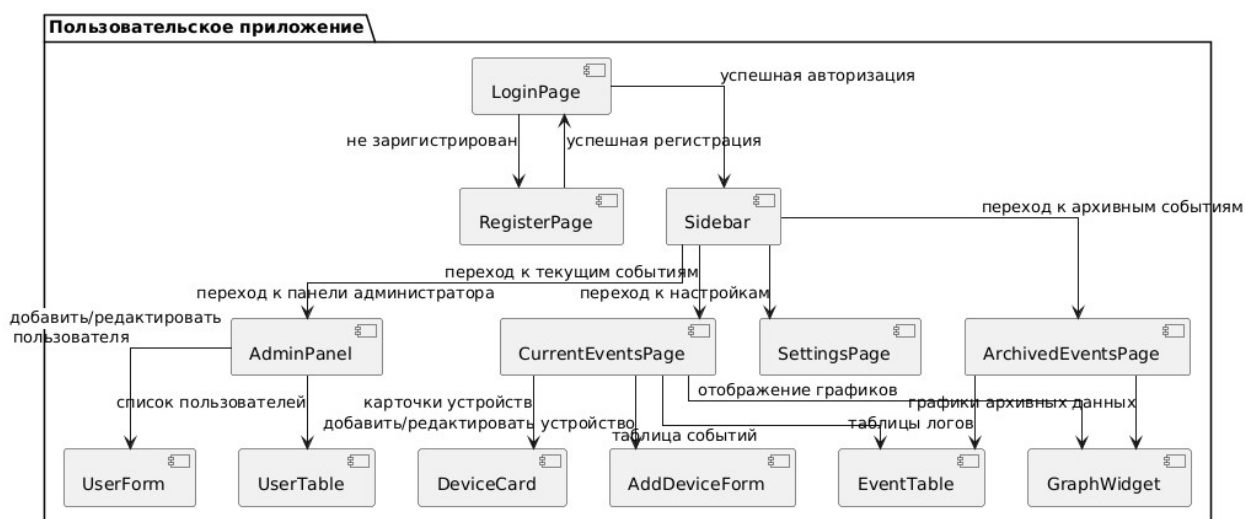


Рисунок 15 – Структуры пользовательского приложения

Web-приложение для задач оператора содержит 6 страниц.

LoginPage (Страница авторизации) – точка входа в систему, после успешной авторизации происходит переход к основному интерфейсу.

RegisterPage (Страница регистрации) – форма создания учетной записи, после регистрации, возврат на страницу входа.

Sidebar (Навигационная панель) – отображается на всех страницах после входа, содержит переходы между основными разделами.

CurrentEventsPage (Текущие события) – отображает данные в реальном времени: карточки устройств, графики и таблицы событий, позволяет добавлять/редактировать устройства.

AdminPanel (Панель администратора) – интерфейс для управления пользователями, содержит таблицу пользователей и форму создания/редактирования.

SettingsPage (Настройки) – позволяет пользователю изменить тему, язык, просмотреть и редактировать данные профиля.

ArchivedEventsPage (Архивные события) – отображает исторические и агрегированные данные по событиям в виде графиков и таблиц.

3.3.2 Разработка визуальных прототипов пользовательского интерфейса

Прототипирование, то есть создание макета продукта, – важный этап в процессе разработки цифровых продуктов. Основная его цель – сэкономить ресурсы команды за счет предварительного тестирования идей.

Прототипирование позволяет оценить юзабилити проекта, проверить предложенные гипотезы и внести необходимые изменения до старта трудоёмкого и требующего много времени программирования. Прототипирование помогает выявить и устранить недостатки на раннем этапе, сэкономив время и ресурсы [20].

Прототип страницы регистрации представлен на рисунке 16.

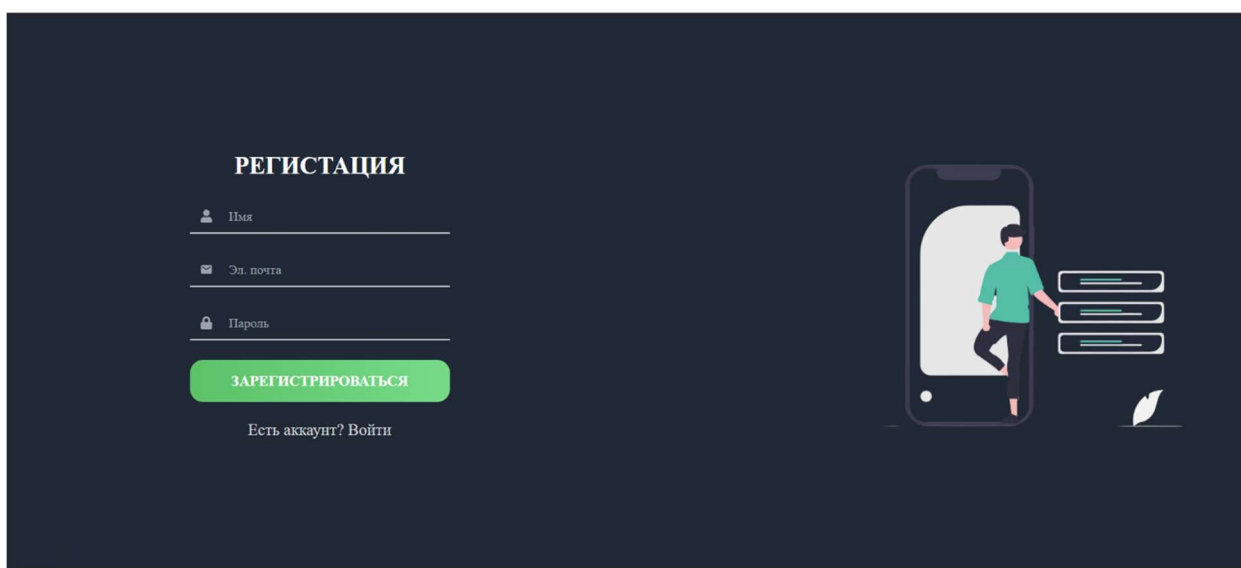


Рисунок 16 – Прототип страницы регистрации

Описание элементов прототипа страницы регистрации представлен в таблице 14.

Таблица 14 – Описание элементов прототипа страницы регистрации

Элемент	Тип компонента	Назначение
Заголовок «Регистрация»	Текст (H1)	Отображение названия формы
Поле «Имя»	Input (text)	Ввод имени пользователя
Поле «Email»	Input (email)	Ввод электронной почты
Поле «Пароль»	Input (password)	Ввод пароля для создания учётной записи
Кнопка «Зарегистрироваться»	Button (primary)	Отправка формы регистрации

При успешной регистрации – автоматический вход в систему и переход на страницу событий.

При ошибке – отображение валидационного сообщения под соответствующим полем.

Прототип страницы входа в систему для аутентификация зарегистрированного пользователя в виде представлен на рисунке 17.

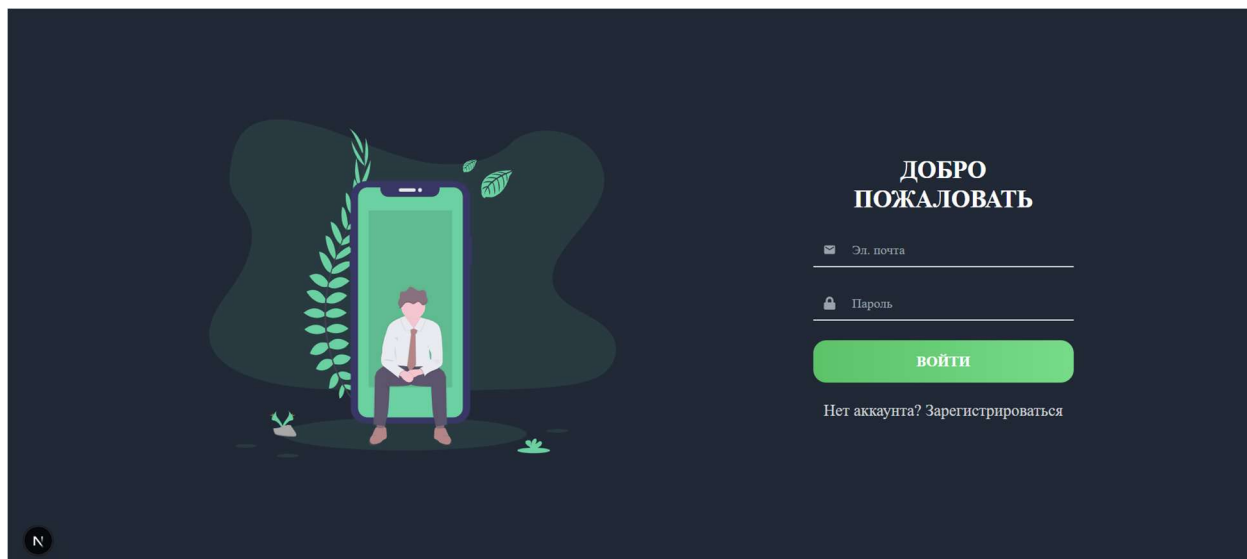


Рисунок 17 – Прототип страницы регистрации

При успешной авторизации – переход на страницу текущих событий. При неудачной попытке – отображение сообщения об ошибке (например, «Неверный email или пароль»). Ссылка – переход на страницу регистрации, если пользователь не зарегистрирован.

Описание элементов прототипа страницы регистрации представлен на таблице 15.

Таблица 15 – Описание элементов прототипа регистрации

Элемент	Тип компонента	Назначение
Заголовок «Добро пожаловать»	Текст (H1)	Название формы входа в систему
Поле «Email»	Input (email)	Ввод логина пользователя
Поле «Пароль»	Input (password)	Ввод пароля
Кнопка «Войти»	Button (primary)	Отправка данных для входа

Страница «Текущие события» предназначена для отображения информации о состоянии устройств в реальном времени, визуализации событийных данных в виде графиков и таблиц, а также для выполнения административных действий – добавления новых элементов визуализации и управления устройствами. Интерфейс предназначен для операторов и администраторов с соответствующими правами доступа и является ключевым элементом системы мониторинга.

Страница текущих событий, как и последующие разделена на три основных структурных области:

- 1) боковая панель навигации (Sidebar);
- 2) верхняя панель заголовка (Header);
- 3) область основного содержимого (Main content).

Структура страницы «Текущие события» обеспечивает логически разделённое отображение такой информации как состояния устройств, событийной аналитики и логов. Прототип предоставляет чёткое разграничение зон ответственности и способствует эффективному UX-дизайну на следующих этапах проектирования.

Прототип предоставляет чёткое разграничение зон ответственности и способствует эффективному UX-дизайну на следующих этапах проектирования. Такая организация компонентов обеспечивает высокую

читаемость интерфейса, удобство восприятия данных и простоту взаимодействия для операторов и администраторов.

Прототип страницы текущих событий представлен на рисунке 18.

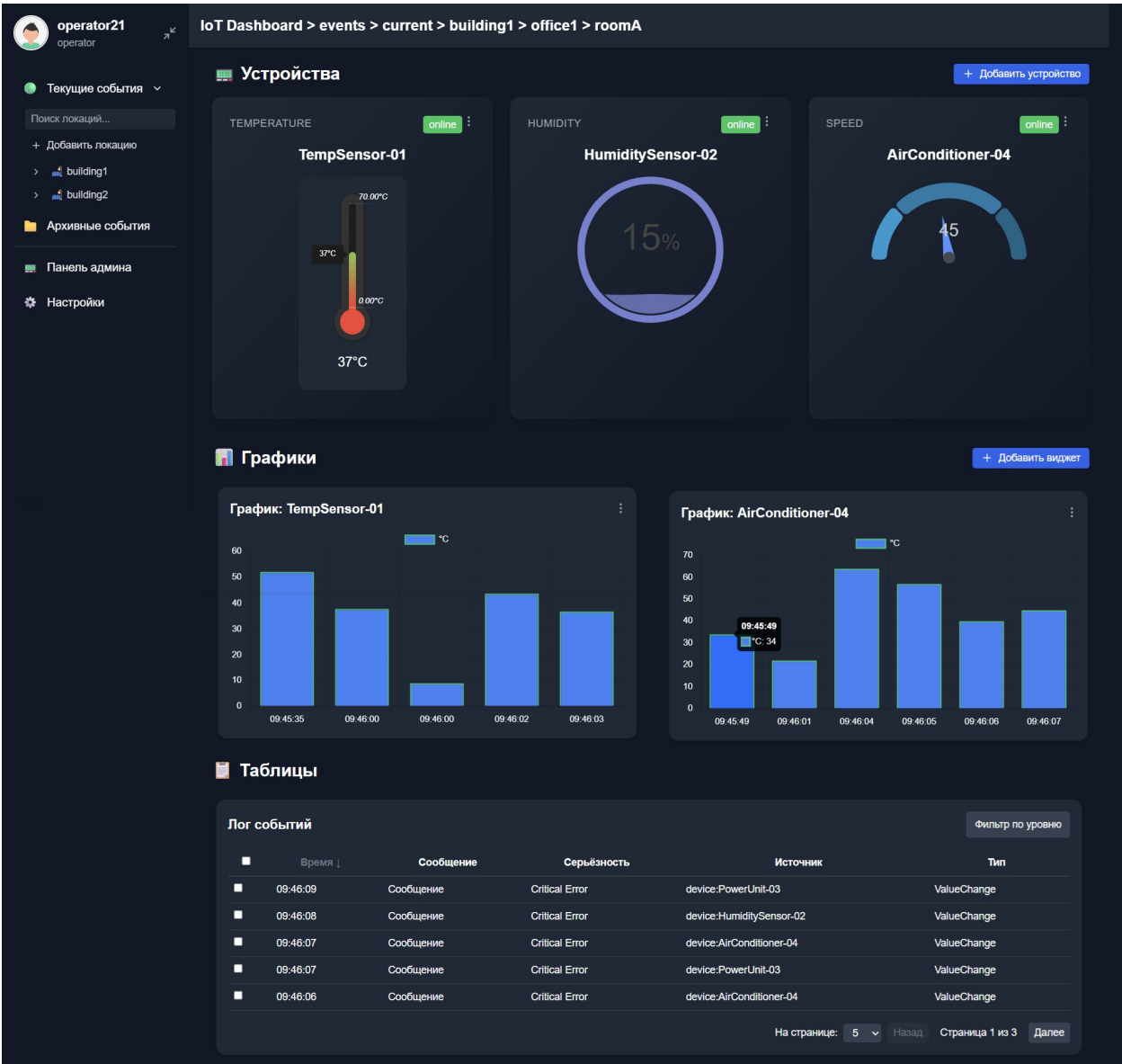


Рисунок 18 – Прототип страницы текущих событий

Боковая панель навигации (Sidebar) расположена в левой части пользовательского интерфейса и представляет собой фиксированный элемент, сохраняющий своё положение при прокрутке основной области страницы. Панель занимает около 20% ширины экрана, обеспечивая баланс между функциональностью и экономией пространства. При необходимости она может быть свернута пользователем, что особенно полезно при работе на

устройствах с ограниченным экранным пространством или при акценте на контентную часть интерфейса.

В таблице 16 представлены описание основных элементов панели навигации.

Таблица 16 – Описание элементов панели навигации

Элемент	Тип	Назначение
Фотография пользователя	Image	Визуальное представление текущего пользователя
Имя пользователя	Text Label	Отображение Имя авторизованного пользователя
Роль пользователя	Text Label	Отображение роли (например «Оператор» или «Администратор»)
Навигационное меню	Nav List	Ссылки на основные разделы
Текущие события	Nav Item	Управление устройствами и событиями
Локации (Комнаты)	Nav Item	Переход к иерархии комнат и страницы событий
Архивные события	Nav Item	Исторические данные по событиям устройств
Панель администратора	Nav Item	Управление пользователями
Настройки	Nav Item	Настройки приложения, например: тема, язык, параметры интерфейса

Верхняя панель заголовка (Header) расположена в верхней горизонтальной области над основным контентом. Назначение панели заголовка – отображение текущего местоположения пользователя в интерфейсе в виде хлебных крошек (breadcrumbs), например «События > текущие события».

В таблице 17 представлены описание основных элементов блока устройств страницы текущих событий.

Таблица 17 – Описание элементов блока устройств текущих событий

Элемент	Тип	Назначение
Заголовок «Устройства»	Text Heading	Название блока
Кнопка «Добавить устройство»	Button (primary)	Кнопка для открытия модального окна для добавления устройства
Карточки устройств	Card Collection	Отображение состояния устройств, краткая информация о устройстве и кнопки управления устройством

В таблице 18 представлены описание основных элементов секции графиков страницы текущих событий.

Таблица 18 – Описание элементов секции графиков текущих событий

Элемент	Тип	Назначение
Заголовок «Графики»	Text Heading	Название секции
Кнопка «Добавить график»	Button (primary)	Кнопка для создания нового графика
Область графиков	Chart Container	Визуализация текущих значений и метрик по событиям

В таблице 19 представлены описание основных элементов блока таблиц страницы текущих событий.

Таблица 19 – Описание элементов блока таблиц текущих событий

Элемент	Тип	Назначение
Заголовок «Таблицы»	Text Heading	Название секции
Кнопка «Добавить таблицу»	Button (primary)	Кнопка открытия конфигуратора таблицы
Компонент таблицы	Data Table	Структурированное представление логов событий

Основная область содержимого страницы (Main Content) расположена в центральной и правая часть экрана. Назначение элемента – отображение событийной информации в формате карточек устройств, графиков и таблиц, с возможностью добавления новых элементов визуализации. Основная область страницы разделена на 3 раздела: блок устройств, блок графиков, блок таблиц.

Клик по пунктам навигации в Sidebar инициирует переход на соответствующую страницу. Кнопки «Добавить» открывают модальные окна с формами ввода и валидацией. Компоненты обновляются в реальном времени через WebSocket. При ошибке загрузки или отсутствии данных – отображаются сообщения с соответствующей ошибкой.

Страница «Архивные события» предназначена для просмотра, анализа и фильтрации событий, зафиксированных в прошлом. Она предоставляет инструменты для интерактивной выборки временного диапазона, построения графиков и таблиц, на основе сохранённых данных. Данный функционал важен для технических специалистов, операторов и администраторов, осуществляющих ретроспективный анализ работы устройств, диагностику инцидентов и построение статистических отчётов.

Страница архивных событий предоставляет пользователю мощный инструмент для анализа накопленных данных, совмещая интерактивные средства визуализации и традиционные таблицы. Благодаря модульной структуре и настраиваемым фильтрам, интерфейс остаётся удобным как для базового просмотра логов, так и для продвинутой аналитики с применением агрегаций и выборок.

Прототип страницы архивных событий представлен на рисунке 19.

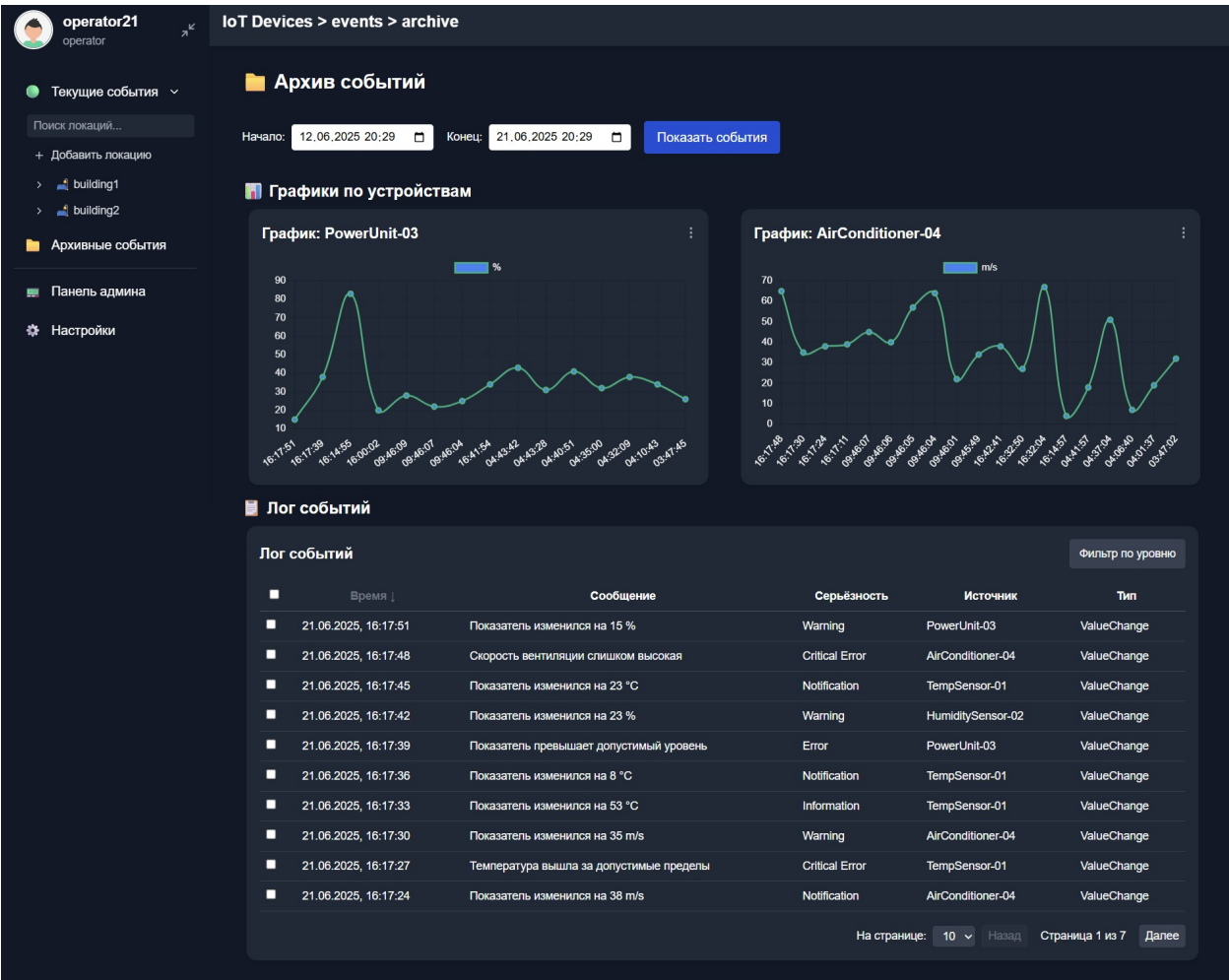


Рисунок 19 – Прототип страницы архивных событий

Страница включает стандартные компоненты:

- 1) боковую панель навигации (Sidebar) с основными пунктами меню;
- 2) верхнюю панель (Header) с хлебными крошками и названием текущего раздела;
- 3) центральную рабочую область (Main Content) – основной блок.

Блок параметров выборки расположен в верхней части основной области. Служит для задания условий фильтрации архивных данных. Блок содержит поля «Начальная дата» и «Конечная дата» для выбора даты и времени начала и конца выборки. Кнопка «Показать» отправляет запрос на сервер для получения и отображения архивных данных. Блок параметров выборки является ключевым элементом взаимодействия пользователя с архивной аналитикой, обеспечивая точность, наглядность и удобство при анализе исторических данных.

Блок графиков архивных данных находится сразу под параметрами фильтрации. Блок представляет собой динамическую визуализацию событий за выбранный период, отображает временное распределение событий.

Таблица логов располагается под графиками. Используется для табличного представления логов событий с возможностью фильтрации, сортировки и поиска.

При изменении параметров выборки и нажатии «Показать», отправляется запрос на backend API. Ответ используется для построения графиков и таблиц. Все визуальные компоненты обновляются синхронно. Ошибки отображаются в виде баннеров или inline-подсказок.

Страница «Панель администратора» предназначена для управления учетными записями пользователей в системе. Интерфейс обеспечивает доступ к информации о текущих пользователях, а также предоставляет средства для их создания, редактирования и удаления. Данный функционал доступен только пользователям с административными правами. При попытке зайти на страницу пользователю без прав администратора выведется сообщение о запрещённом доступе и код 403.

Страница «Панель администратора» реализует структуру управления пользователями с акцентом на наглядность, прямолинейность операций и минимизацию количества переходов. Чёткое разделение интерфейса на блок просмотра и блок редактирования упрощает восприятие и снижает когнитивную нагрузку при работе с административными функциями.

Прототип панели администратора представлен на рисунке 20.

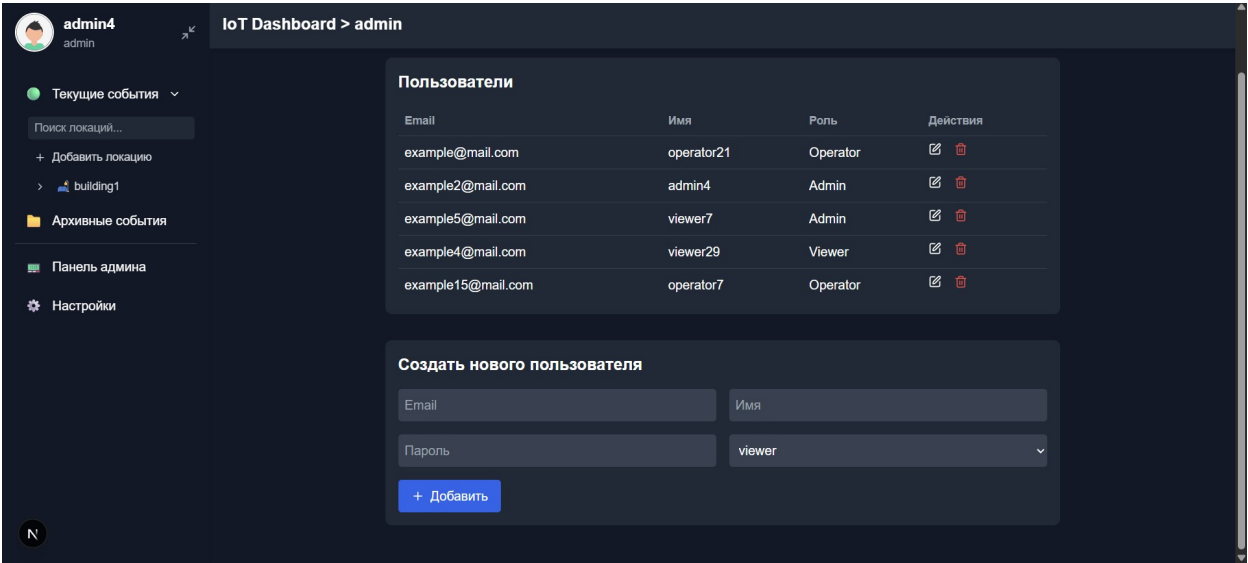


Рисунок 20 – Прототип панели администратора

Компонента страницы являются Sidebar и Header, а так же основное содержимое, которое сосредоточено в блоках таблицы пользователей и формы создания/редактирования пользователя.

Блок просмотра и управления пользователями расположен в верхней части основной области страницы. Элемент необходим для отображения списка всех зарегистрированных пользователей компании и выполнение базовых операций над ними.

В таблице 20 представлены описание основных элементов блока пользователей страницы администратора.

Таблица 20 – Описание элементов блока управления пользователями

Элемент	Тип компонента	Назначение
Заголовок «Пользователи»	Text Heading	Идентификация блока
Таблица пользователей	Data Table	Выводит список пользователей с возможностью редактирования и удаления
Столбец «Email»	Table Column	Отображает адрес электронной почты пользователя
Столбец «Имя»	Table Column	Отображает отображаемое имя пользователя
Столбец «Роль»	Table Column	Отображает роль пользователя в системе (например, «оператор», «админ»)

Продолжение таблицы 20

Элемент	Тип компонента	Назначение
Столбец «Действия»	Action Buttons	Кнопки «Редактировать» и «Удалить» для каждой записи
Кнопка «Редактировать»	Icon/Button	При нажатии открывает форму редактирования в нижнем блоке
Кнопка «Удалить»	Icon/Button	Иницирует подтверждение и удаление пользователя

Блок создания и редактирования пользователя расположен ниже таблицы пользователей для создание новой учетной записи или редактирование выбранной.

Особенностями формы является что при нажатии «Редактировать» данные выбранного пользователя подставляются в форму. После сохранения форма очищается и возвращается в режим создания. Валидация обязательных полей и корректности email и пароля обязательна.

В таблице 21 представлены описание основных элементов блока создания пользователей страницы администратора.

Таблица 21 – Описание элементов блока создания пользователя

Элемент	Тип компонента	Назначение
Заголовок «Создать нового пользователя»	Text Heading	Название формы. При редактировании может заменяться на «Редактировать пользователя»
Поле «Email»	Input (email)	Ввод адреса электронной почты нового/редактируемого пользователя
Поле «Имя»	Input (text)	Ввод отображаемого имени пользователя
Поле «Пароль»	Input (password)	Установка пароля (может быть отключено при редактировании, если не меняется)
Меню выбора роли	Dropdown Select	Выпадающий список всех доступных ролей (например, «Оператор», «Администратор»)
Кнопка «Добавить»	Button (primary)	Сохраняет нового или изменённого пользователя
Кнопка «Отмена»	Button (secondary)	Возвращает форму в режим создания или очищает поля

Клик по кнопке «Редактировать» активирует форму с предзаполненными данными. Кнопка «Удалить» требует подтверждения действия, во избежание потери данных. После добавления/обновления

пользователя, таблица автоматически обновляется. Ошибки отображаются в виде подсказок под соответствующими полями или баннером над формой.

Страница «Настройки» предоставляет пользователю доступ к персонализированным параметрам учетной записи, отображению базовой информации о себе, возможностям управления профилем, а также дополнительным системным настройкам (например, выбор темы или языка интерфейса). Раздел ориентирован на конечного пользователя и фокусируется на автономной настройке поведения приложения без необходимости обращения к администраторам.

Страница «Настройки» обеспечивает пользователю простой и интуитивно понятный интерфейс для работы с персональными и системными параметрами. Четкая иерархия блоков, отделение управляющих функций от визуальных настроек, а также доступность всех ключевых операций в рамках одного интерфейса способствует высокой эффективности взаимодействия.

Прототип страницы настроек представлен на рисунке 21.

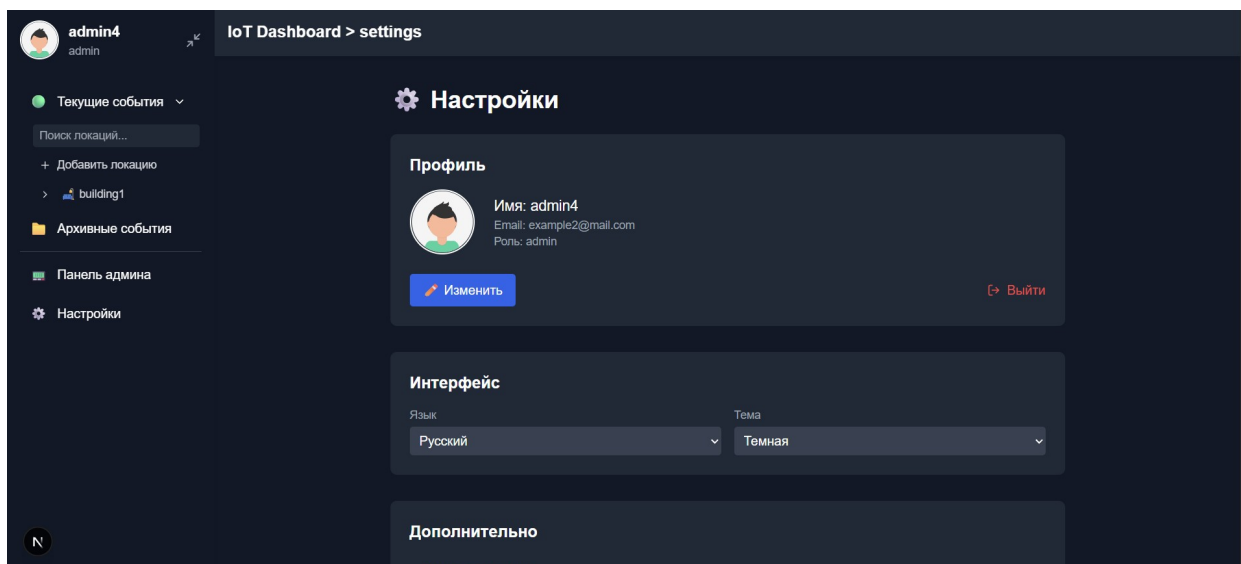


Рисунок 21 – Прототип страницы настроек системы

Основными элементами страницы является Sidebar, Header и основная часть интерфейса, которая разделена на логические блоки управление учетной записью и дополнительные настройки.

Блок «Управление учетной записью» расположен в верхней часть основной области страницы для отображение ключевых пользовательских данных и предоставление функционала управления сессией и профилем.

В таблице 22 представлены описание основных элементов блока управления учетной записью страницы.

Таблица 22 – Описание элементов блока управления учетной записью

Элемент	Тип компонента	Назначение
Заголовок «Настройки»	Text Heading (H1)	Идентификация страницы
Аватар пользователя	Image	Отображение текущей фотографии пользователя
Поле «Email»	Read-only Label	Отображение зарегистрированного email пользователя
Поле «Имя»	Read-only Label	Отображение имени пользователя
Поле «Роль»	Read-only Label	Отображение текущей роли в системе (например «Оператор», «Администратор»)
Кнопка «Редактировать»	Button (secondary)	Переход к редактированию профиля
Кнопка «Выйти»	Button (danger/exit)	Завершение текущей пользовательской сессии

Данные не редактируемы напрямую – при нажатии «Редактировать» открывается форма для редактирования данных пользователя, и кнопка «Сохранить изменения». Кнопка «Выйти» требует подтверждения действия (для предотвращения случайного выхода).

Блок «Дополнительные настройки» расположен ниже блока учетной записи и предназначен для предоставления параметров настройки поведения и внешнего вида пользовательского интерфейса и других настроек системы.

Дополнительные настройки могут содержать переключатель «Тема интерфейса» для выбора между «Светлой» и «Тёмной» темой оформления интерфейса, переключатель «Язык интерфейса» для выбора языка интерфейса (например, Русский, English, Deutsch) и кнопка «Сохранить изменения» для применение выбранных пользовательских настроек.

Все элементы отображаются адаптивно и упорядочены в вертикальной колонке. Выход из аккаунта перенаправляет пользователя на страницу авторизации.

3.4 Разработка программного обеспечения приложения

3.4.1 Разработка серверной части приложения

3.4.1.1 Разработка структуры серверной части приложения

Для отображения архитектуры программы серверной части приложения была выбрана диаграмма пакетов.

Диаграммы пакетов унифицированного языка моделирования(UML) отображают зависимости между пакетами, составляющими модель.

В UML пакеты используются как «механизм для группирования элементов и предоставления пространства имён для этих элементов»

Пакетная диаграмма отображает структурную организацию модели и позволяет группировать классы, компоненты, узлы, проекты в логические структуры, визуализирует зависимости между модулями (импорт, merge).

Диаграмма используется для показа слоистой архитектуры или управления развитием (разделение среды, слоев и подобное) [21].

Пакеты позволяют логически группировать элементы системы, такие как классы, сценарии использования и другие пакеты, и управляют видимостью и доступом к ним.

Диаграммы пакетов используются для организации и структурирования сложной системы, делая ее более понятной и управляемой. Они также помогают в управлении зависимостями между различными частями системы.

Диаграммы пакетов могут быть использованы для представления системы на разных уровнях абстракции.

Диаграмма пакетов является важным инструментом в UML для организации и структурирования сложных систем, обеспечивая логическую группировку элементов, управление видимостью и зависимостями, а также облегчая понимание и управление системой

На рисунке 22 изображена диаграмма пакетов структура серверной части приложения.

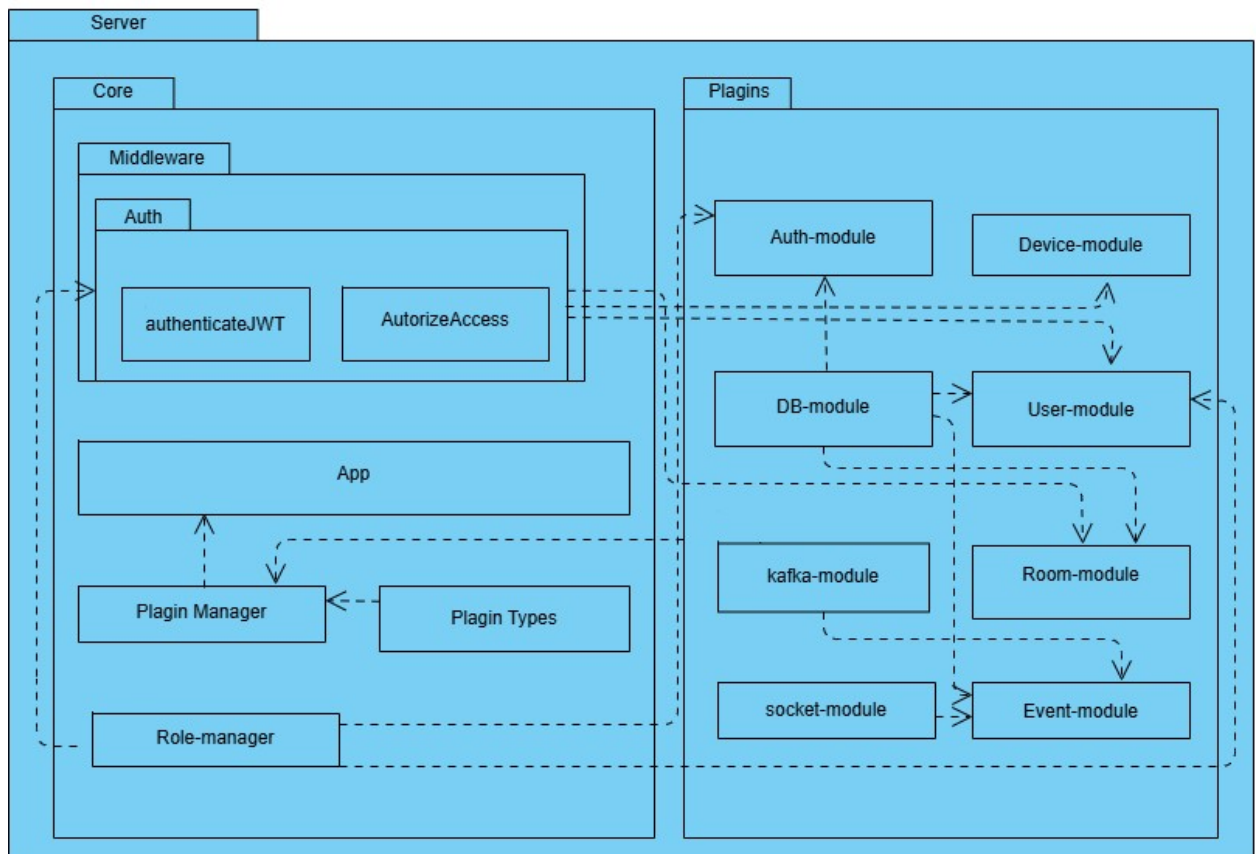


Рисунок 22 – Структура серверной части приложения

Диаграмма пакетов серверной части отражает модульную структуру программной архитектуры, реализованной с использованием сервис-ориентированного и плагино-ориентированного подходов. Основной акцент сделан на разделение ответственности, инверсию зависимостей и расширяемость системы без модификации ядра.

Вся логика сервера организована в виде иерархической структуры, где главный пакет – **server**, включает два ключевых пространства:

- **Core** – ядро системы (инфраструктура, управление плагинами, авторизация, общие механизмы);
- **Plugins** – функциональные модули, подключаемые динамически;
- **Server** – является корневым пространством имён, управляет запуском, регистрацией модулей, инициализацией базовой инфраструктуры и конфигурацией окружения.

Подпакет «core» содержит основные механизмы и абстракции, не привязанные к предметной области. Все элементы plugins используют интерфейсы, определённые в core.

Более детальное описание ядра представлено в таблице 23.

Таблица 23 – Описание компонентов ядра сервера

Модуль	Назначение
app	Точка входа в систему. Управляет загрузкой конфигураций, инициализацией плагинов, зависимостей и запуском серверного приложения.
plugin-manager	Отвечает за загрузку, регистрацию и управление жизненным циклом плагинов. Поддерживает горячую замену и инициализацию зависимостей.
plugin-types	Определяет интерфейсы и контракты, реализуемые всеми плагинами (например, IDevicePlugin, IUserPlugin).
role-manager	Управление ролями и политиками доступа. Включает реализацию RBAC и взаимодействует с auth-плагином.
middleware	Содержит промежуточные обработчики запросов.
middleware.auth	Реализация middleware для проверки JWT, извлечения токенов, контроля доступа и передачи контекста пользователя.

Plugins содержит предметно-ориентированные модули (плагины), реализующие конкретную бизнес-логику. Все плагины реализуют интерфейсы из plugin-types и регистрируются через plugin-manager.

Подробно каждый модуль описан в таблице 24.

Таблица 24 – Описание компонентов ядра сервера

Модуль	Назначение
db-module	Управление подключением к PostgreSQL, инициализация моделей, миграции.
kafka-module	Получение событий от Kafka, парсинг, маршрутизация.
socket-module	Отправка сообщений клиентам в реальном времени через WebSocket (Socket.IO).
auth-module	Реализация авторизации (JWT, refresh tokens, RBAC), интеграция с middleware.
device-module	CRUD-операции над устройствами, обновление значений, проверка прав.
user-module	Управление пользователями, создание пользователей, назначение ролей, обновление данных, профили.
room-module	Работа с комнатами, вложенностью и иерархией помещений.
event-module	Работа с событиями: логирование, хранение, фильтрация, преобразование, оповещение.

Особенности архитектуры является, то что все компоненты слабосвязны, что обеспечивает лёгкую замену/добавление модулей, ядро не знает о реализации модулей, только об интерфейсах, все взаимодействия между модулями происходят через абстракции, возможна изоляция и тестирование плагинов в отрыве от всей системы.

3.4.1.2 Разработка программных модулей уровня модели

Программный модуль – это независимая, функционально законченная часть программы, имеющая определенное назначение и обеспечивающая заданные функции.

На уровне модели реализуется основная бизнес-логика приложения, связанная с обработкой данных, управлением правами доступа и интеграцией с внешними источниками событий. В отличие от применения ORM-библиотек (таких как Prisma или Sequelize), в разработке используется низкоуровневая библиотека pg, обеспечивающая прямое взаимодействие с PostgreSQL. Такой подход позволяет достичь максимального контроля над SQL-запросами и эффективно обрабатывать сложные бизнес-сценарии.

Основные программные модули уровня модели включают:

- модель ролей и прав доступа;
- хранилище и обработка событий;
- модули взаимодействия с базой данных;
- модуль авторизации и аутентификации;
- модуль управление пользователя;
- модуль управления устройствами.

Вся система разграничения доступа реализована через абстрактный класс Role и его конкретные реализации (GuestRole, OperatorRole, AdminRole). Каждая роль определяет, какие ресурсы доступны пользователю через метод `canAccess(resource: string)`. Такой подход соответствует принципам ООП

(наследование, полиморфизм) и позволяет гибко расширять и конфигурировать политику безопасности.

На листинге 1 представлена система ролей пользователей.

Листинг 1 – Система ролей пользователей

```
export abstract class Role {
  constructor(public name: string) {}

  abstract canAccess(resource: string): boolean;

  static fromName(name: string): Role {
    switch (name) {
      case "guest": return new GuestRole();
      case "operator": return new OperatorRole();
      case "admin": return new AdminRole();
      default: throw new Error(`Unknown role: ${name}`);
    }
  }
}

class GuestRole extends Role {
  constructor() { super("guest"); }

  canAccess(resource: string): boolean {
    return resource === "self";
  }
}

class OperatorRole extends Role {
  constructor() { super("operator"); }

  canAccess(resource: string): boolean {
    return ["self", "read", "write", "update",
"delete"].includes(resource);
  }
}

class AdminRole extends Role {
  constructor() { super("admin"); }

  canAccess(resource: string): boolean {
    return true; // доступ ко всему
  }
}
```

В модуле взаимодействия с базой данных каждое взаимодействие осуществляется через ручные SQL-запросы (`db.query(...)`), что требует строгой типизации, обработки ошибок и транзакционного контроля. Данный подход

даёт гибкость при интеграции с существующими схемами базы данных и обеспечивает высокую читаемость бизнес-правил на уровне SQL.

В модули авторизации и аутентификации, аутентификация пользователей реализована через проверку email и пароля, хранимого в виде хэша. При успешной авторизации возвращается JWT-токен, определяющий доступ пользователя на основании его роли.

В модуле хранилища и обработки событий система событий реализована через плагин Kafka. При получении сообщений от брокера Kafka, consumer распаршивает сообщение в структуру AlarmEvent, производит поиск идентификатора устройства в базе данных, определяет тип события и его уровень серьёзности, сохраняет событие в таблицу events, находит всех пользователей с доступом к устройству и инициирует рассылку уведомлений по WebSocket.

Дополнительно реализован REST-интерфейс для получения архивных событий за заданный временной период. Для этого используется функция `get_events_by_date_range`, которая принимает параметры даты начала и конца, а также идентификатор пользователя.

Модуль пользователей реализует такие ключевые функции как администрирование пользователей, изменение данных текущего пользователя, назначение ролей (guest, operator, admin) с соответствующими уровнями доступа, получение данных обо всех пользователях или конкретного пользователя по ID.

Реализация безопасности достигается за счёт использования JWT (JSON Web Token) и middleware-функций `authenticateJWT` и `authorizeAccess`, проверяющих подлинность токена и наличие прав на выполнение действия. Пароли хранятся в зашифрованном виде с использованием алгоритма `bcrypt`.

Модуль устройств предоставляет API-интерфейс для следующих операций:

- добавление и удаление устройств;

- обновление данных устройства (включая статус, тип, расположение, права пользователей);
- отображение доступных пользователю устройств;
- управление иерархией расположения (локаций), включая вложенные комнаты;
- переключение статуса устройства (активен/неактивен).

Доступ к устройствам фильтруется на уровне базы данных функцией `get_user_devices`, которая возвращает только те устройства, к которым пользователь имеет права доступа. Поддерживается вложенность расположений, а глубина и порядок вложенных локаций определяется функцией `get_user_location_tree`.

Структура репозитория данных системы представлена на рисунке 23.

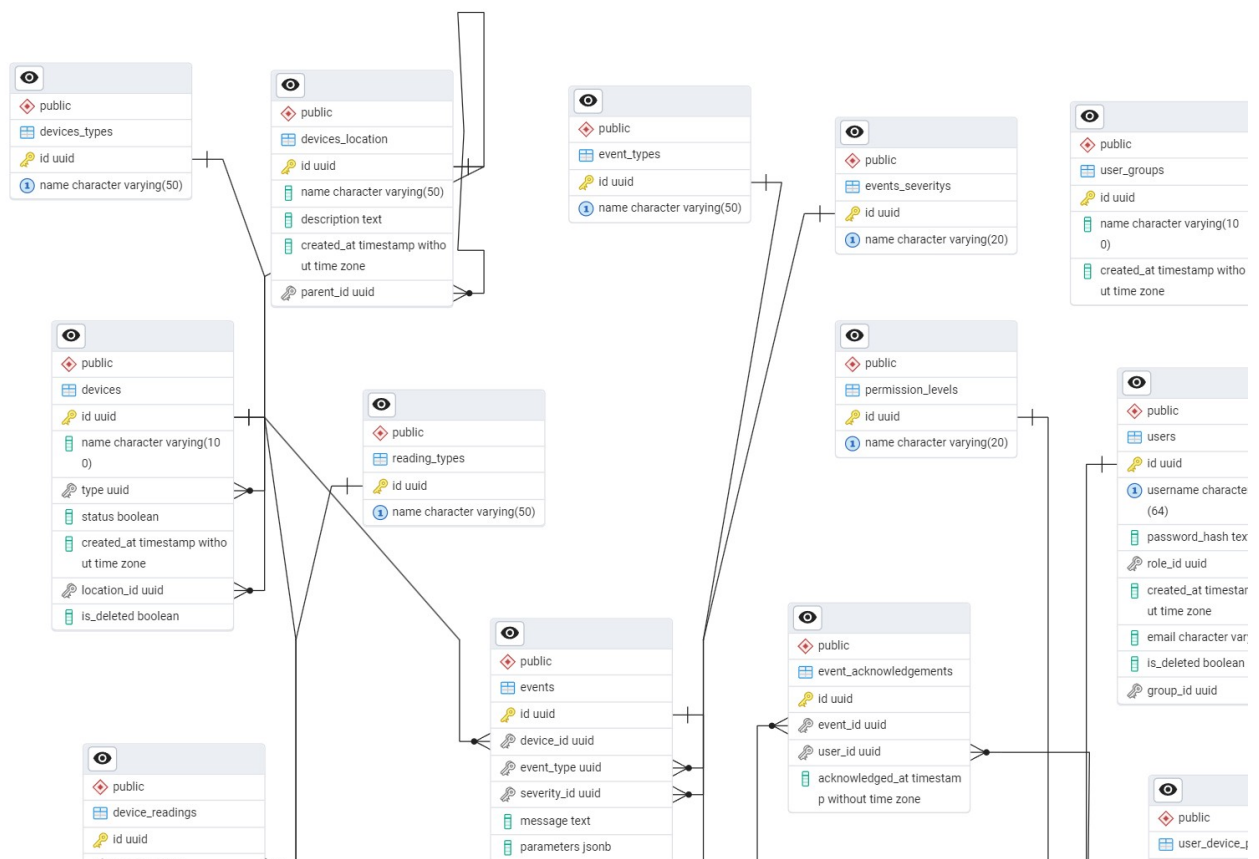


Рисунок 23 – Структура репозитория данных системы

Модельный уровень приложения реализует основную прикладную и предметно-ориентированную логику. Архитектурное решение с явной

реализацией моделей и прямыми SQL-запросами обеспечивает прозрачность, расширяемость и точный контроль над хранением данных и их доступом.

3.4.1.3 Разработка программных модулей уровня контроллера

Контроллеры системы реализуют промежуточный уровень между представлением и моделью. В системе применяются REST-принципы и socket-интерфейсы, что обеспечивает согласованное взаимодействие между frontend и backend.

Контроллеры реализованы в виде модулей-плагинов, подключаемых к экземпляру приложения на базе Express.js. Каждый контроллер обрабатывает определённый набор маршрутов и имеет прямой доступ к экземпляру базы данных через библиотеку pg.

В таблице 25 представлены описание основных REST API эндпоинтов серверного приложения.

Таблица 25 – Описание эндпоинтов серверного приложения

№	Метод	URI	Назначение	Защита / Права
1	2	3	4	5
1	POST	/auth/login	Авторизация пользователя, выдача токена	–
2	POST	/auth/register	Регистрация пользователя, выдача токена	–
3	POST	/user/create-user	Создание нового пользователя	JWT + admin
4	PUT	/user/update-user	Редактирование другого пользователя	JWT + admin
5	PUT	/user/update-user-data	Редактирование своих данных	JWT
6	DELETE	/user/delete-user	Удаление пользователя	JWT + admin
7	GET	/user/get-all-users/:id	Получение списка всех пользователей	JWT + admin
8	GET	/user/get-user-data/:id	Получение информации по конкретному пользователю	JWT + read
9	GET	/devices/userDevices	Получение списка устройств пользователя	JWT
10	GET	/devices/deviceTypes	Получение справочника типов устройств	JWT
11	GET	/devices/deviceLocations	Получение доступной иерархии локаций	JWT
12	POST	/devices/createLocation	Добавление новой локации	JWT + write

Продолжение таблицы 25

13	POST	/devices/create-device	Добавление устройства	JWT + write
14	PUT	/devices/update-device	Обновление данных устройства	JWT + update
15	DELETE	/devices/delete-device	Удаление устройства	JWT + delete
16	PUT	/devices/toggle-device	Активация/деактивация устройства	JWT + update
17	GET	/events/archiveEvents	Получение архивных событий за период	JWT
18	POST	/kafka/send	Отправка события в Kafka вручную	JWT (внутренний вызов)

Особенностью реализации является то, что маршруты защищены middleware-функциями `authenticateJWT` и `authorizeAccess`, что обеспечивает разграничение доступа на основе ролей. Контроллеры не содержат бизнес-логики, а только координируют действия между входными параметрами и вызовами модели. Ошибки обработки данных логируются и возвращаются в стандартизированном формате (`message`, `error`, `statusCode`). Унифицирована структура ответов, как для успешных операций (`{ success: true, data: ... }`), так и для ошибок (`{ success: false, message: ... }`).

3.4.2 Разработка клиентской части приложения

3.4.2.1 Разработка программных модулей уровня представления

Модули представления реализованы с использованием библиотеки `React` и обеспечивают построение пользовательского интерфейса приложения. Компоненты организованы по функциональному признаку и включают страницы и компоненты страниц.

Страницы приложения:

- страница авторизации(/login);
- страница регистрации(/registration);
- текущие события (/events/current);
- архив событий (/events/archive);
- панель администратора (/admin);
- настройки (/settings).

Основные компоненты приложения представлены в таблице 26.

Таблица 26 – Компоненты клиентского приложения

Компонент	Назначение
Layout	Главный макет с Header, Sidebar и областью контента
Sidebar	Навигация по страницам, иерархия комнат с возможностью добавления
Header	Заголовок страницы с хлебными крошками
DeviceCard	Отображение устройства, с действиями: редактировать, включить/выключить, удалить
AddOrEditDeviceModal	Модальное окно для добавления/редактирования устройства и назначения пользователей
Widget	Универсальный компонент для отображения графиков и таблиц
FloatingInput	Интерактивные поля ввода с иконками и анимацией
AnalogGauge, AnalogGauge2, DigitalDisplay, AccelDial, Battery, Temp, LiquidWidget	Визуальные виджеты для отображения данных устройств в различных форматах

Все компоненты используют сервисы API (`api/*.js`), которые обеспечивают асинхронные вызовы и работают с токенами авторизации. Управление глобальными данными, такими как темы и локации, реализовано с помощью React Context API.

Виджеты представляют собой гибко подключаемые визуальные компоненты, используемые для отображения текущих и архивных данных устройств.

Каждый виджет реализован как самостоятельный React-компонент, экспортируемый из структуры `components/widgets/*`. Список поддерживаемых типов виджетов указан в таблице 27.

Таблица 27 – Список поддерживаемых виджетов

Название виджета	Назначение
AnalogGauge	Отображение аналоговых значений в круговой шкале
DigitalDisplay	Цифровое значение параметра (температура, ток и остальное)
AccelDial	Стрелочный индикатор ускорения или отклонения
Battery	Индикатор уровня заряда
Temp	Температурный индикатор с цветовой шкалой
LiquidWidget	Отображение уровня жидкости (с анимацией)

Сначала виджеты регистрируются как плагины и могут быть динамически подключены без изменения кода. На этапе загрузки страницы с графиками или данными, интерфейс получает данные о виджетах и

отображает каждый виджет на основании метаданных (тип, ID устройства, параметры). Компонент `Widget` определяет тип отображения и загружает нужный виджет по ключу `type` (например, `Chart1`, `Analog` и другие). Все виджеты принимают `props`, включая идентификатор устройства, параметры события и стиль отображения.

На листинге 2 представлен механизм загрузки виджетов.

Листинг 2 – Механизм загрузка виджетов

```
// lib/loadWidgets.js
export const loadPlugins = async () => {
  const context = require.context('../plugins', true,
/index\.js$/);
  const plugins = await
Promise.all(context.keys().map(context));
  return plugins.map(mod => mod.default);
};
// lib/WidgetRegistry.js
import { createContext, useContext } from "react";

const WidgetRegistryContext = createContext([]);
export const useWidgets = () =>
useContext(WidgetRegistryContext);
export const WidgetRegistryProvider = ({ children, widgets }) =>
(
  <WidgetRegistryContext.Provider value={widgets}>
    {children}
  </WidgetRegistryContext.Provider>
);
//page
useEffect(() => {
  loadPlugins().then(setWidgets);
}, []);
```

Логика взаимодействия компонентов такова, что после входа пользователя и перехода на страницу `CurrentEvents`, происходит подключение к `WebSocket` (через `Socket.IO`). Входящие события обрабатываются и распределяются по устройствам, отображаясь в соответствующих карточках, таблицах и графиках. Каждое событие может одновременно отображаться в карточке устройства, на графике или в таблице событий.

Пользователь может добавлять виджеты и таблицы через кнопку «Добавить график» – это запускает механизм генерации UI на основе конфигурации, поступающей с сервера или из контекста.

4 Контроль качества программных средств взаимодействия с объектами мониторинга и управления

4.1 Планирование процесса контроля качества приложения

4.1.1 Мероприятия по контролю качества

Контроль качества программного продукта представляет собой систему мероприятий, направленных на выявление дефектов и сопоставление готового функционала с установленными требованиями (функциональными и нефункциональными) углубленного уровня, что подтверждается словарным определением термина как «набор процедур для проверки того, что программный компонент соответствует требованиям и обеспечивает низкий уровень дефектов». В отличие от более широкого понятия обеспечения качества (QA), контроль качества (QC) фокусируется на идентификации дефектов и проверке итогового продукта перед релизом [22].

Основные мероприятия контроля качества:

- планирование контрольных процедур;
- ревью и инспекции кода;
- модульное тестирование;
- интеграционное и системное тестирование;
- приемо-сдаточные тесты;
- тестирование нефункциональных требований;
- регрессия и мониторинг дефектов.

Планирование контрольных процедур – определение критериев приемки по каждому элементу функциональности и производительности и установление ключевых показателей качества (KPI), таких как время ответа, отказоустойчивость, уровень узких мест.

Ревью и инспекции кода – анализ соответствия архитектуре, правилам кодирования, безопасности и читаемости.

Модульное тестирование – проверка каждого программного модуля (функции, класса) в изоляции и обеспечение покрытия граничных условий и обработки ошибок (unit tests).

Интеграционное и системное тестирование – проверка взаимодействия между модулями (при контроллерно-модельных связях), проверка обмена данными с базой данных и сервером, и реализация сценариев реальной работы – от авторизации до вывода карты графиков.

Приемо-сдаточные тесты (Acceptance Tests) – проверка комплексных сценариев и участие представителей заказчика для оценки соответствия ожиданиям и требованиям.

Тестирование нефункциональных требований – нагрузочное тестирование, измерение масштабируемости при нескольких одновременных подключениях, тестирование безопасности и проверка UI (кроссбраузерность, адаптивность, отзывчивость).

Регрессия и мониторинг дефектов – поддержка набора автоматизированных тестов для каждого релиза, отслеживание и измерение плотности дефектов, среднего времени исправления (MTTR), времени до обнаружения (MTTD).

Контроль качества выступает как финальная проверка готового ПО, направленная на выявление и исправление ошибки до публикации, что снижает риск сбоев и повышает доверие пользователей.

Выстроенный процесс контроля – это гарант приемки проекта, минимизации рисков и снижения затрат на исправление дефектов на более поздних этапах жизненного цикла.

4.1.2 Обоснование выбора модели тестирования

Модель тестирования – это концептуальная основа, определяющая методы, цели и этапы тестирования программного обеспечения. Выбор конкретной модели зависит от архитектуры системы, уровня автоматизации, состава функциональности, а также ожидаемой надёжности и удобства

интерфейса. В рамках разработки рассматриваемой системы было принято решение использовать смешанную модель тестирования, включающую элементы функционального, конфигурационного и интерфейсного тестирования.

Основные используемые виды тестирования приведены в таблице 28.

Таблица 28 – Используемые виды тестирования

Вид тестирования	Назначение
Функциональное (черный ящик)	Проверка соответствия поведения системы заявленным требованиям без знания внутренней реализации
Конфигурационное тестирование	Проверка поведения системы в различных средах (браузеры, экраны, ОС)

Обоснование выбора модели является поддержка нескольких сценариев использования – различные роли (администратор, оператор) и возможности (управление устройствами, просмотр событий, модификация профиля) требуют независимой верификации через сценарные и функциональные тесты. Важность интерфейса и UX – система ориентирована на визуальную работу с виджетами, графиками и таблицами – следовательно, особое внимание уделяется оценке интерфейса: скорости, когнитивной нагрузке, эффективности взаимодействия. Метрики и автоматизация оценки качества – модели тестирования также должны быть совместимы с использованием метрик кода и интерфейса.

Выбранная модель тестирования охватывает все аспекты работы системы, от логики компонентов до удобства пользовательского интерфейса. Такой подход позволяет обеспечить всестороннюю валидацию, снизить риски регрессионных ошибок и повысить устойчивость решения в реальной эксплуатации.

4.1.3 Обоснование выбора метрик программного кода

Метрика программного обеспечения (англ. software metric) – мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций [23].

Поскольку количественные методы хорошо зарекомендовали себя в других областях, многие теоретики и практики информатики пытались перенести данный подход и в разработку программного обеспечения.

Набор используемых метрик включает:

- порядок роста (анализ алгоритмов в терминах асимптотического анализа и O-нотации);
- количество строк кода;
- цикломатическая сложность;
- анализ функциональных точек;
- количество ошибок на 1000 строк кода;
- степень покрытия кода тестированием;
- покрытие требований;
- количество классов и интерфейсов;
- метрики программного пакета от Роберта Сесиль Мартина;
- связность.

В таблице 29 приведены выбранные метрики и причины их применения.

Таблица 29 – Выбранные метрики

Метрика	Категория	Обоснование применения
Метрики устойчивости и абстрактности (Instability & Abstractness)	Архитектурные	Анализируют зависимость модулей, их абстрактность, баланс интерфейсов и реализаций. Позволяют выявить уязвимые и перегруженные участки архитектуры.
Метрики Холстеда	Лексические / Сложность	Используются для оценки информационной сложности кода на уровне функций и классов. Позволяют определить, насколько сложно читать, понимать и тестировать модуль.
Плотность комментариев, длина функций, уровни вложенности	Структурные	Вспомогательные показатели для оценки читаемости и соблюдения стиля кодирования.

В условиях разработки расширяемого и многослойного приложения ключевыми задачами являются:

- удержание структурной целостности кода при увеличении числа модулей;

- контроль сложности и связности компонентов;
- обеспечение сопровождаемости и расширяемости без существенного рефакторинга;
- формализация качества для последующего сравнительного анализа версий.

Применение архитектурных и лексических метрик позволяет формализовать оценку качества реализации, выявить зоны риска и технического долга, а также обосновать необходимость оптимизации или рефакторинга. Это делает контроль качества не только субъективной оценкой, но и инструментально подтверждаемым процессом.

4.1.4 Обоснование выбора методик оценки качества пользовательского интерфейса

Оценка качества пользовательского интерфейса (ПИ) – это необходимый этап в контроле над эргономикой, доступностью, эффективностью и когнитивной нагрузкой интерфейса программного продукта. Так как пользовательский интерфейс является точкой взаимодействия между системой и человеком, его качество напрямую влияет на восприятие надёжности, удобства и полноты функционала системы.

Цель оценки интерфейса – выявить и количественно зафиксировать:

- степень удобства использования (usability);
- скорость и количество действий при выполнении задач;
- соответствие принципам пользовательского опыта (UX);
- когнитивную и моторную нагрузку на пользователя;
- эффективность информационного взаимодействия.

Оценка качества ПИ направлена на выявление недостатков и разработку рекомендаций по улучшению интерфейса, чтобы сделать его более удобным, эффективным и приятным для пользователей

В таблице 30 приведены выбранные метрики оценки качества пользовательского интерфейса.

Таблица 30 – Выбранные метрики оценки качества ПИ

Методика	Назначение и обоснование применения
Keystroke-Level Model (KLM)	Позволяет оценить время выполнения пользовательских действий (нажатие клавиш, клики, перемещения) без учёта обучения. Применяется для сравнения эффективности интерфейсных решений (например, редактирование устройства и добавление).
GOMS-модель (Goals, Operators, Methods, Selection rules)	Обеспечивает системный подход к оценке логики взаимодействия. Используется для моделирования пользовательских сценариев и анализа вариантов их реализации в интерфейсе. Позволяет оценить когнитивную нагрузку и оптимальность последовательности действий.

Причиной выбора является применимость к экспертной оценке – выбранные модели не требуют широкого пользовательского тестирования и могут применяться аналитически на этапе разработки или рефакторинга. Фокус на эффективность – Keystroke-Level Model позволяет выявить избыточные шаги или неудобные переходы, а GOMS-модель определяет, насколько логично и понятно устроен интерфейс. Применение аналитических моделей оценки пользовательского интерфейса позволяет объективно формализовать трудоёмкость и удобство использования системы, выявить узкие места интерфейса и принять обоснованные решения по его улучшению. Это особенно актуально для интерфейсов с высокой плотностью информации и сложной логикой взаимодействия.

4.2 Тестирование приложения

4.2.1 Функциональное тестирование

Тестирование «черного ящика» – это функциональное и нефункциональное тестирование без доступа к внутренней структуре компонентов системы. Метод тестирования «черного ящика» – процедура получения и выбора тестовых случаев на основе анализа спецификации (функциональной или нефункциональной), компонентов или системы без ссылки на их внутреннее устройство.

В качестве методики тестирования черным ящиком были выбраны функциональная диаграмма и таблица решений.

Таблица принятия решений – инструмент тест-дизайна, или процесса создания тестов. Таблица помогает придумать, как и что тестировать в программном обеспечении, например, на сайте или в приложении. Её можно использовать для проверки требований, собранных для разработки ПО, например, проверять, что учтены все возможные варианты.

Один из принципов тестирования звучит так – избыточное тестирование невозможно. Поэтому тестировщик вынужден выбирать, на что потратить время и какой метод выбрать, чтобы собрать максимум информации о работоспособности продукта или системы за минимальное время.

Таблица помогает ничего не пропустить и не держать в голове миллион вариантов. Таблица принятия решений полезна там, где есть требования, которые описывают несколько разных исходов ситуации в зависимости от условий. Например, с ней будет удобно рассчитать скидку в магазине, размер бонусов или кэшбэка, процентную ставку по ипотеке. Таблица поможет собрать все условия в одном месте и убедиться, что требования полные и непротиворечивые.

Преимущества таблицы решений:

- 1) наглядность;
- 2) удобство;
- 3) простота.

Наглядность. Это главное преимущество метода. Вместо того, чтобы текстом описывать тест-кейсы и бояться что-то упустить, можно составить матрицу и быть уверенным, что ни одна проверка не потеряется.

Удобство. Один столбец таблицы – один готовый тест-кейс.

Простота. Таблицу решений можно составить в Google-таблице, Excel, на бумаге или даже на салфетке, если хочется. Чтобы использовать метод, не нужно уметь писать код или осваивать специальную программу.

Главный недостаток метода – для составления таблицы нужно время, которого всегда не хватает. Иногда тестировщик думает, что проще потратить полчаса-час на тестирование, а не на составление таблицы. [19]

Таблица причин и следствий представлена на таблице 31.

Таблица 31 – Таблица причин и следствий

Причины	
П1	Корректная длина названия
П2	Корректная длина идентификатора
П3	Название состоит из корректных символов
П4	Идентификатор состоит из корректных символов
П5	Идентификатор свободен
Следствия	
С1	Сообщение: «Некорректная длина названия»
С2	Сообщение: «Некорректная длина идентификатора устройства»
С3	Сообщение: «Некорректные символы в составе названия»
С4	Сообщение: «Некорректные символы в идентификаторе устройства»
С5	Сообщение: «Идентификатор устройства уже используется»
С6	Сообщение: «IoT-устройство успешно зарегистрировано»

Построение функциональной диаграммы на основе таблицы причин и следствий представлено на рисунке 24.

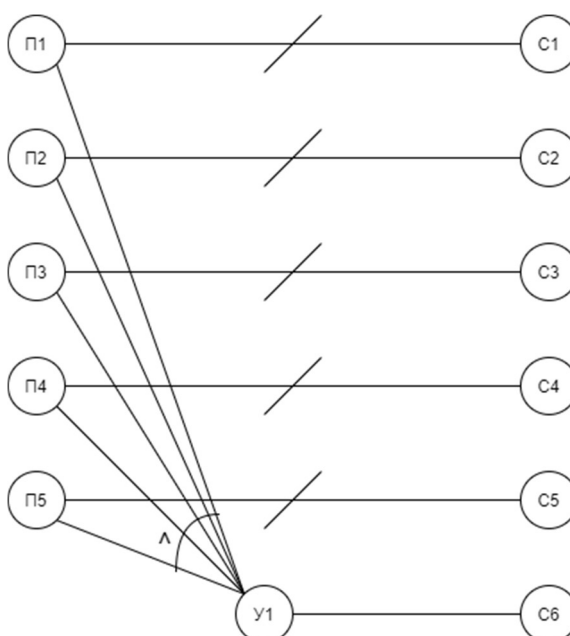


Рисунок 24 – Функциональная диаграмма

Построение таблицы принятия решений на основе таблицы причин и следствий представлено на таблице 32.

Таблица 32 – Таблица принятия решений

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
П1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
П2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1
П3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1
П4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
П5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
С1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
С2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
С3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
С4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
С5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Построение тестов (тест-кейсов) на основе таблицы принятия решений представлено на таблице 33.

Таблица 33 – Таблица тест-кейсов

№	Входные данные	Ожидаемый результат
1	Название: Sr@H Серийный номер: A1B%	Некорректная длина названия, Некорректная длина идентификатора устройства, Некорректные символы в составе названия, Некорректные символы в идентификаторе устройства (рисунок Е.1)
2	Название: Smart#\$Home24 Серийный номер: U*()C3D4	Некорректная длина идентификатора устройства, Некорректные символы в составе названия, Некорректные символы в идентификаторе устройства (рисунок Е.2)
3	Название: CV(^& Серийный номер: A1B\$C3D*E5F6	Некорректная длина названия, Некорректные символы в составе названия, Некорректные символы в идентификаторе устройства (рисунок Е.3)
4	Название: Smart#\$Bomb54 Серийный номер: U*-IC3D4HB5Y	Некорректные символы в составе названия, Некорректные символы в идентификаторе устройства (рисунок Е.4)
5	Название: S-b54 Серийный номер: M%O45Y	Некорректная длина названия, Некорректная длина идентификатора устройства, Некорректные символы в идентификаторе устройства (рисунок Е.5)
6	Название: Smart-Doom Серийный номер: IO%O4	Некорректная длина идентификатора устройства, Некорректные символы в идентификаторе устройства (рисунок Е.6)

Продолжение таблицы 33

№	Входные данные	Ожидаемый результат
7	Название: Br4 Серийный номер: CV%O4&5YP*8W	Некорректная длина названия, Некорректные символы в идентификаторе устройства (рисунок E.7)
8	Название: Smart-Cap Серийный номер: YUPO-4&5YP*8W	Некорректные символы в идентификаторе устройства (рисунок E.8)
9	Название: Sr@H Серийный номер: A1BN	Некорректная длина названия, Некорректная длина идентификатора устройства, Некорректные символы в составе названия (рисунок E.9)
10	Название: Sr&^Home Серийный номер: M1BN	Некорректная длина идентификатора устройства, Некорректные символы в составе названия (рисунок E.10)
11	Название: S#&e Серийный номер: A1B2C3D4E5F6	Некорректная длина названия, Некорректные символы в составе названия (рисунок E.11)
12	Название: *Smart#&eBrt Серийный номер: A1B2C3D4E5F6	Некорректные символы в составе названия (рисунок E.12)
13	Название: Sma Серийный номер: B2CE5F6	Некорректная длина названия, Некорректная длина идентификатора устройства (рисунок E.13)
14	Название: Smart-Milk Серийный номер: E25F6	Некорректная длина идентификатора устройства (рисунок E.14)
15	Название: Silk Серийный номер: A1B2C3D4E5F6	Некорректная длина названия (рисунок E.15)
16	Название: Smart-Camp12 Серийный номер: (занятой идентификатор)	Идентификатор устройства уже используется (рисунок E.16)
17	Название: Smart-Lamp24 Серийный номер: (свободный идентификатор)	IoT-устройство успешно зарегистрировано (рисунок E.17)

Таким образом, было проведено тестирование чёрным ящиком.

4.2.2 Конфигурационное тестирование

Конфигурационное тестирование – это вид тестирования, целью которого является проверка корректной работы приложения в различных программных и аппаратных средах. Данный вид тестирования особенно важен для веб-приложений, так как пользователи могут использовать разные браузеры, устройства и операционные системы.

В рамках проекта конфигурационное тестирование проводилось на уровне пользовательского интерфейса, с целью убедиться в корректной работе всех элементов страницы, включая:

- отрисовку компонентов (графики, таблицы, карточки, формы);
- поддержку адаптивной верстки (в пределах стандартных разрешений);
- поведение интерактивных элементов (выпадающие меню, модальные окна, переключатели и подобные элементы);
- корректную передачу и обработку данных при взаимодействии с API и WebSocket;
- консистентность внешнего вида и иконографии.

Тестирование проводилось вручную, с использованием актуальных версий браузеров. Проверялись корректность верстки, функциональность переходов, взаимодействие с формами, а также общее визуальное поведение компонентов интерфейса.

Результаты конфигурационного тестирования представлены на таблице 34.

Таблица 34 – Таблица конфигурационного тестирования

Браузер	Тестируемая ситуация	Ожидаемый результат	Наблюдаемый результат
1	2	3	4
Google Chrome	Загрузка страницы авторизации	Страница авторизации отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.1)
	Загрузка страницы регистрации	Страница регистрации отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.2)
	Загрузка страницы текущих событий	Страница текущих событий отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.3)
	Загрузка страницы архивных событий	Страница архивных событий отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.4)
	Загрузка панели администратора	Панели администратора отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.5)

Продолжение таблицы 34

1	2	3	4
	Загрузка страницы настроек	Страница настроек отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.6)
Microsoft Edge	Загрузка страницы авторизации	Страница авторизации отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.7)
	Загрузка страницы регистрации	Страница регистрации отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.8)
	Загрузка страницы текущих событий	Страница текущих событий отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.9)
	Загрузка страницы архивных событий	Страница архивных событий отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.10)
	Загрузка панели администратора	Панели администратора отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.11)
	Загрузка страницы настроек	Страница настроек отображается корректно, поля и кнопки активны	Страница отображена корректно, тест пройден (рисунок Ж.12)

На основании проведённого конфигурационного тестирования можно сделать вывод о полной кроссбраузерной совместимости пользовательского интерфейса в современных версиях браузеров Google Chrome и Microsoft Edge на платформе Windows. Все функциональные и визуальные элементы работают корректно, нарушений адаптивности или сбоев отображения не выявлено.

4.3 Оценка качества программного кода

4.3.1 Метрики устойчивости и абстрактности

Метрика устойчивости (Instability, I) и метрика абстрактности (Abstractness, A) являются важными характеристиками архитектурных модулей, отражающими их архитектурный баланс и чувствительность к

изменениям. Они были предложены Робертом Мартином в рамках анализа модульной архитектуры.

Метрики устойчивости и абстрактности являются важными инструментами для оценки качества программного обеспечения. Они позволяют разработчикам принимать обоснованные решения при проектировании, разработке и рефакторинге кода, обеспечивая более надежную и гибкую систему.

Устойчивость рассчитывается по следующей формуле:

$$I (\text{Устойчивость}) = \frac{C_e}{(C_a + C_e)},$$

где I – показатель устойчивости программного компонентов, значение которого изменяется в диапазоне от 0 до 1 (если $I = 0$, то компонент максимально устойчив, если $I = 1$ – максимально неустойчив);

C_a (входящие связи) – количество классов вне данного компонента, зависящих от классов внутри компонента;

C_e (исходящие связи) – количество классов внутри данного компонента, зависящих от классов вне компонента.

Для расчета были выбраны модули системы такие как SocketModule, DatabaseModule, Role, AuthModule, DevicesLogicModule, EventLogicModule, UserLogicModule.

Был выполнен расчет показателя устойчивости для каждого программного компонента приложения:

$$I_1 = \frac{2}{1 + 2} = 0,67;$$

$$I_2 = \frac{1}{6 + 1} = 0,14;$$

$$I_3 = \frac{0}{1} = 0;$$

$$I_4 = \frac{3}{1 + 3} = 0,75;$$

$$I_5 = \frac{2}{1 + 2} = 0,67;$$

$$I_6 = \frac{1}{1+1} = 0,5;$$

$$I_7 = \frac{2}{0+2} = 1;$$

Абстрактность рассчитывается по следующей формуле:

$$A \text{ (Абстрактность)} = \frac{N_a}{N_c},$$

где A – показатель абстрактности программного компонента, значение которого изменяется в диапазоне от 0 до 1 (если $A = 0$, то в компоненте нет абстрактных классов, если $A = 1$ – компонент содержит только абстрактные классы);

N_c – количество классов в компоненте;

N_a – количество абстрактных классов внутри компонента.

Был выполнен расчет показателя абстрактности для каждого программного компонента приложения:

$$A_1 = \frac{0}{1} = 0;$$

$$A_2 = \frac{0}{1} = 0;$$

$$A_3 = \frac{1}{5} = 0,2;$$

$$A_4 = \frac{1}{2} = 0,5;$$

$$A_5 = \frac{1}{2} = 0,5;$$

$$A_6 = \frac{1}{2} = 0,5;$$

$$A_7 = \frac{1}{2} = 0,5;$$

В виде графика можно представить соотношение между устойчивостью и абстрактностью. На графике максимально устойчивые и абстрактные компоненты находятся в левом верхнем углу, а максимально неустойчивые и

конкретные – в правом нижнем углу. График соотношения представлен на рисунке 25.

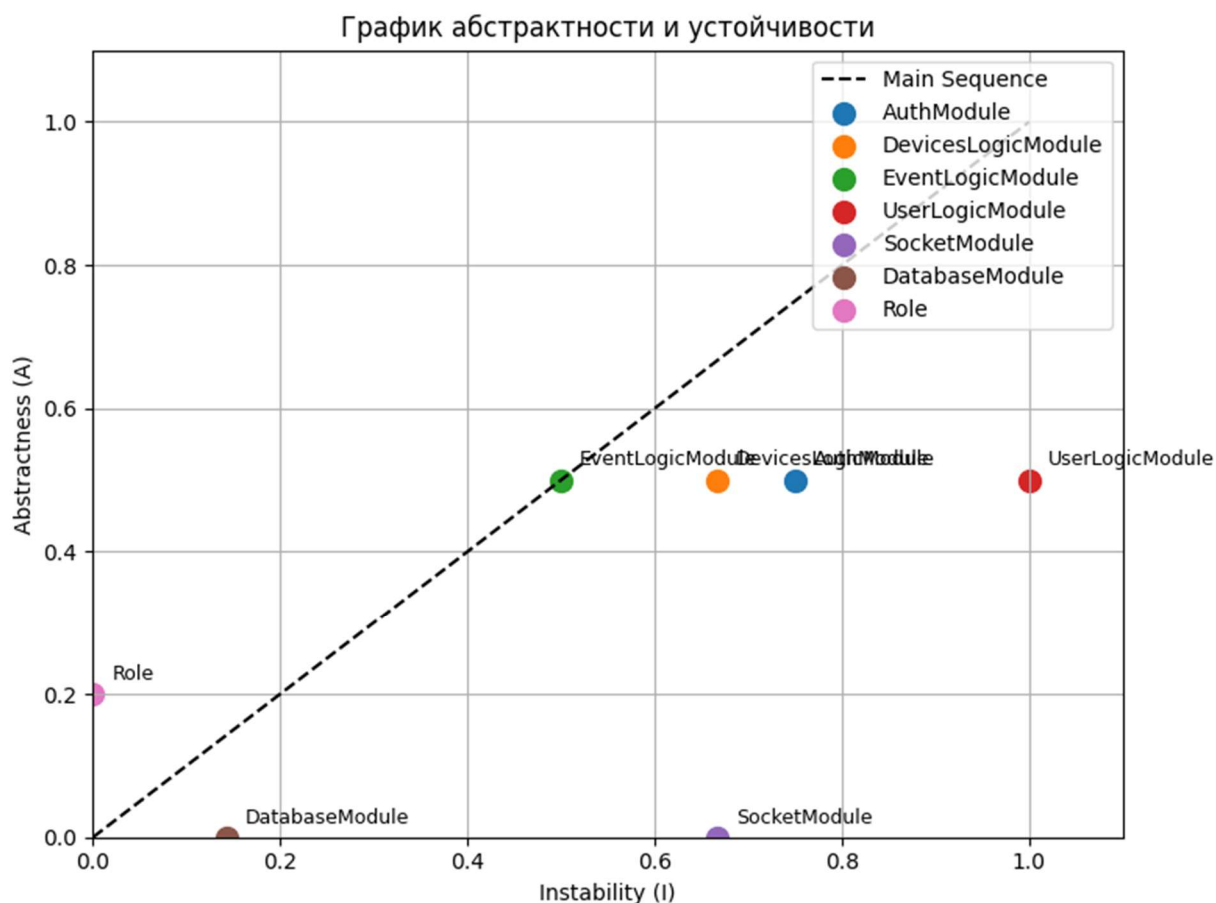


Рисунок 25 – График соотношения между устойчивостью и абстрактностью

Идеально сбалансирован по метрикам модуль EventLogicModule ($I = 0.5$, $A = 0.5$), что соответствует архитектурной главной диагонали Роберта Мартина – баланс между устойчивостью и абстрактностью.

Модуль Role демонстрирует крайне высокую устойчивость ($I = 0.0$), что соответствует его назначению, он предоставляет абстракции, но не зависит от других.

Модуль UserLogicModule наиболее нестабилен ($I = 1.0$), несмотря на высокую абстрактность. Это может свидетельствовать о риске архитектурной уязвимости и необходимости снижения связности.

Модули `SocketIOModule` и `DatabaseModule` не содержат абстракций ($A = 0.0$), что допустимо для низкоуровневых инфраструктурных компонентов, но делает их жёстко связанными с конкретной реализацией.

Архитектура приложения в целом демонстрирует сбалансированное применение интерфейсов и абстрактных компонентов. Однако модули с высокой нестабильностью и нулевой абстрактностью могут потребовать рефакторинга или выделения абстракций для повышения гибкости и расширяемости системы.

4.3.2 Метрики Холстеда

Метрики Холстеда, предложенные Морисом Холстедом – это группа количественных показателей, используемых для оценки сложности программного обеспечения. Они основаны на анализе операторов и операндов в коде, и позволяют оценить такие характеристики, как объем, длина, сложность и уровень программы.

Метрики Холстеда помогают выявить сложные участки кода, которые могут быть трудными для понимания и поддержки. Позволяют сравнивать разные варианты решения одной и той же задачи с точки зрения их сложности. Метрики Холстеда могут использоваться для оценки времени и усилий, необходимых для разработки и поддержки программного обеспечения. Существует корреляция между сложностью кода и вероятностью возникновения ошибок. Метрики Холстеда могут помочь выявить потенциально ненадежные участки кода.

Метрики Холстеда являются полезным инструментом для анализа программного обеспечения, но не стоит рассматривать их как единственное средство оценки. Их следует использовать в сочетании с другими методами анализа и оценки качества кода.

Данные метрики основаны на следующих показателях:

- 1) $n1$ – число уникальных операторов программы, включая символы, разделители, имена процедур и знаки операций (словарь операторов);

2) n_2 – число уникальных операндов программы (словарь операндов);

3) N_1 – общее число операторов в программе;

4) N_2 – общее число операндов в программе.

На основании этих показателей рассчитываются оценки:

- словарь программы (Halstead Program Vocabulary, HPVoc);
- длина программы (Halstead Program Length, HPLen);
- объем программы (Halstead Program Volume, HPVol).

Для подсчета метрик Холстеда была выбрана функция валидации данных создания устройства, представления в листинге 3.

Листинг 3 – Код функции валидации данных создания устройства

```
import { Plugin } from "../../core/types";
import { App } from "../../core/App";
import { authenticateJWT } from "../../core/middleware/auth";

// Типы
interface ArchiveEventQuery {
  userId: string;
  startDate?: string;
  endDate?: string;
}

interface IEventRepository {
  getArchiveEvents(query: ArchiveEventQuery): Promise<any[]>;
}

// Реализация
class PostgresEventRepository implements IEventRepository {
  constructor(private db: any) {}

  async getArchiveEvents({ userId, startDate, endDate }:
ArchiveEventQuery): Promise<any[]> {
    const result = await this.db.query(
      "SELECT * FROM get_events_by_date_range($1, $2, $3)",
      [userId, startDate, endDate]
    );
    return result.rows;
  }
}

const EventLogicPlugin: Plugin = {
  name: "EventLogicPlugin",
```

```

async load(app: App) {
  const db = (app as any).db;
  const eventRepo = new PostgresEventRepository(db);

  app.server.get("/events/archiveEvents", authenticateJWT,
  async (req: any, res: any) => {
    const userId = req.user?.id;
    const { startDate, endDate } = req.query;

    if (!userId) {
      return res.status(400).json({ message: "User ID not
found in token" });
    }

    try {
      const events = await eventRepo.getArchiveEvents({
        userId,
        startDate,
        endDate,
      });

      res.json({ events });
    } catch (err) {
      const message = err instanceof Error ? err.message :
"Internal server error";
      console.error("Failed to fetch archive events:", err);
      res.status(500).json({ message });
    }
  });
},

  async unload() {
    console.log("EventLogicPlugin unloaded");
  },
};

```

export default EventLogicPlugin;

Детальный подсчет показателей для расчета метрик Холстеда представлен на таблице 35.

Таблица 35 – Таблица операторов и операндов

Операторы	Число операторов	Операнды	Число операндов
import	4	Plugin	2
interface	2	App	2
class	1	ArchiveEventQuery	2
constructor	1	IEventRepository	2
async	3	PostgresEventRepository	2
await	2	this.db	5

Продолжение таблицы 35

return	3	query	3
if	1	getArchiveEvents	3
const	4	userId, startDate, endDate	9
=	6	result, events, message	6
.	18	req	4
{}	10	res	4
[]	3		
()	12		
:	6		
throw	1		
try	1		

На рисунке 26 изображен подсчет метрик Холстеда.

$$n1 = 17$$

$$N1 = 72$$

$$n2 = 12$$

$$N2 = 44$$

$$n = n1 + n2 = 29$$

$$N = N1 + N2 = 72 + 44 = 116$$

$$V = N * \log_2(n) = 116 * \log_2(29) = 563,6$$

$$D = (17 / 2) \times (44 / 12) = 31.17$$

$$E = 31.17 \times 673.8 \approx 21\,004.6$$

Рисунок 26 – Подсчет метрик Холстеда

Метрики Холстеда показали, что в программе содержится 29 уникальных операндов и операторов (Словарь программы), общая длина программы составляет 116, а объем программы равен 563,6 бит.

Таким образом, были рассчитаны метрики кода, показавшие корректность программы.

4.4 Оценка качества пользовательского интерфейса

Метод Keystroke-Level Model (KLM) – это модель предсказания времени выполнения стандартных операций человеком при взаимодействии с интерфейсом. Она относится к семейству моделей GOMS (Goals, Operators, Methods, Selection rules) и позволяет оценить производительность интерфейса

без необходимости его запуска, основываясь на моделировании пользовательского поведения.

Метод заключается в том, что все действия пользователя в рамках выполнения сценария можно представить, как набор типовых операций и результат всех действий могут быть подсчитаны.

Метод широко используется при проектировании интерфейсов для критически важных систем, где скорость и точность взаимодействия имеют первостепенное значение.

Для оценки качества пользовательского интерфейса методом Keystroke-level Model использовалась функция создания устройства которая требует ввода имени, выбора типа устройства, выбора локации, нажатия кнопки «создать».

Сценарий создания устройства приведен в таблице 36.

Таблица 36 – План тестирования методом Keystroke-level Model

Шаг	Действие	Операторы	Время (сек)
1	Переместить курсор в поле имени устройства	P + M + BV	$1.1 + 1.35 + 0.2 = 2.65$
2	Ввести имя устройства (например, «Термометр», 9 символов)	$9 \times K$	$9 \times 0.2 = 1.8$
3	Переместить курсор в выпадающее меню типа	P + M + BV	2.65
4	Щёлкнуть и выбрать тип	BV + P + BV	$0.2 + 1.1 + 0.2 = 1.5$
5	Переместить курсор в меню локаций	P + M + BV	2.65
6	Выбрать локацию	BV + P + BV	1.5
7	Переместить курсор на кнопку «Создать»	P + M + BV	2.65
8	Нажать кнопку	BV	0.2

Таким образом, общая оценка времени:

$$T_{total} = 2,65 + 1,8 + 2,65 + 1,5 + 2,65 + 1,5 + 2,65 + 0,2 = 15,6 \text{ сек};$$

По модели KLM, ориентировочное время на выполнение сценария «создание устройства» в вашем интерфейсе составляет ≈ 15.6 секунд для опытного пользователя без ошибок. Такой показатель говорит о правильном построении пользовательского интерфейса.

Заключение

В рамках выпускной квалификационной работы была успешно реализована система автоматизации задач оператора по мониторингу и управлению событиями, связанными с IoT-устройствами. Разработка осуществлялась на основе современных принципов проектирования клиент-серверных приложений с использованием ролевой модели доступа, событийно-ориентированной архитектуры и модульного подхода к логике и интерфейсу.

На начальном этапе был проведён анализ предметной области, охватывающий ключевые сущности, такие как пользователи, устройства, события, локации и взаимодействия между ними. На основании этого анализа были выделены основные бизнес-процессы, определены классы отображаемых объектов, сформированы сценарии использования и определены требования к системе.

В процессе планирования разработки был составлен перечень задач, установлены зависимости и сроки выполнения, подготовлена диаграмма Ганта, таблица ресурсов и рассчитаны бюджетные ограничения. Это позволило систематизировать этапы проектирования и обеспечило управляемость процесса.

Далее была выполнена формализация требований и постановка задачи. Определены функциональные и нефункциональные требования, разработана архитектура приложения, включая структуру базы данных, логику плагинов, модели представления данных, схему API и принципы маршрутизации событий с использованием Kafka и WebSocket (Socket.IO).

В ходе разработки программной системы реализованы все уровни архитектуры:

- модель;
- контроллер;
- представление.

Модель реализована на базе PostgreSQL с использованием хранимых процедур и репозиторного подхода.

Контроллер – реализует REST API, разграничение доступа, обработку событий и аутентификацию.

Представление – выполнено на React с Tailwind CSS, включает адаптивный и интерактивный пользовательский интерфейс с виджетами.

Отдельное внимание уделялось контролю качества программного обеспечения. Была выбрана модель тестирования «черного ящика» для функционального тестирования. Проведена оценка метрик кода (устойчивость, абстрактность, метрики Холстеда), а также выполнено конфигурационное и нагрузочное тестирование. Для интерфейса была применена методика GOMS/KLM, позволившая оценить пользовательский опыт при типовых сценариях.

Таким образом, в результате выполнения работы:

- разработано полнофункциональное веб-приложение с учётом требований безопасности, производительности и удобства;
- реализована масштабируемая архитектура, пригодная для дальнейшего развития и интеграции новых компонентов;
- подтверждено соответствие программного продукта заявленным требованиям посредством многоуровневого тестирования.

Результаты работы могут быть использованы для внедрения в реальную производственную среду, а также служить основой для расширения функциональности в будущих проектах, например, добавление аналитики, машинного обучения, поддержки мобильных клиентов.

Список использованных источников

1. Шибанов С.В., Скоробогатько, Э.В. Лысенко. Интегрированная модель активных правил // Математическое и программное обеспечение систем в промышленной и социальной сферах: междунар. сб. науч. трудов - Магнитогорск: Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2011 - Ч1. - с.41-46.
2. Шибанов, С. В. Сервис потоковой обработки событий и исполнения активных правил / С. В. Шибанов, Я. С. Шлепнев // Математическое моделирование и суперкомпьютерные технологии : Труды XXI Международной конференции, Нижний Новгород, 22–26 ноября 2021 года. – Нижний Новгород: Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского, 2021. – С. 403-407.
3. Шибанов, С. В. Мета модель сервиса потоковой обработки событий / С. В. Шибанов, Я. С. Шлепнев, А. С. Гусаров // Методы, средства и технологии получения и обработки измерительной информации («Шляндинские чтения - 2022»): материалы XIV Международной научно-технической конференции с элементами научной школы и конкурсом научно-исследовательских работ для обучающихся и молодых ученых, Пенза, 24–26 октября 2022 года. – Пенза: Пензенский государственный университет, 2022. – С. 240-243.
4. Шибанов, С. В. Портал управления метаданными сервиса потоковой обработки событий и исполнения активных правил / С. В. Шибанов, А. С. Гусаров, Я. С. Шлепнев // Информационные технологии в моделировании и управлении: подходы, методы, решения: Материалы VI Всероссийской научной конференции с международным участием, Тольятти, 18–20 апреля 2023 года. – Тольятти: Тольяттинский государственный университет, 2023. – С. 481-489.
5. Википедия. Бизнес-процесс [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Бизнес-процесс> (дата обращения: 21.06.2025).

6. Аналитика Плюс. Сравнительный анализ полученных данных [Электронный ресурс]. URL: <https://analytikaplus.ru/sravnitelnyj-analiz-poluchennyh-dannyh/> (дата обращения: 21.06.2025).
7. Википедия. Анализ требований [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Анализ_требований (дата обращения: 21.06.2025).
8. Visure Solutions. Functional Requirements: Guide [Электронный ресурс]. URL: <https://visuresolutions.com/ru/alm-guide/functional-requirements/> (дата обращения: 21.06.2025).
9. Википедия. Диаграмма прецедентов [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_прецедентов (дата обращения: 21.06.2025).
10. Visure Solutions. Non-functional Requirements: Guide [Электронный ресурс]. URL: <https://visuresolutions.com/ru/alm-guide/non-functional-requirements/> (дата обращения: 21.06.2025).
11. Википедия. Интерфейс пользователя [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Интерфейс_пользователя (дата обращения: 21.06.2025).
12. Википедия. Веб-интерфейс [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/веб_интерфейс (дата обращения: 21.06.2025).
13. Rational Unified Process: Навигация по рабочим продуктам [Электронный ресурс]. URL: https://dit.isuct.ru/Publish_RUP/core.base_rup/workproducts/rup_navmap_73AF74E2.html (дата обращения: 21.06.2025).
14. Википедия. Аппаратное обеспечение [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Аппаратное_обеспечение (дата обращения: 21.06.2025).
15. Kruchten, P. The Rational Unified Process: An Introduction / Philippe Kruchten. — Addison-Wesley, 1998. — 320 p.
16. Pender, T. UML Bible / Tom Pender. — Wiley Publishing, Inc., 2003. — 936 p.

17. IBM Developer. The Sequence Diagram [Электронный ресурс]. URL: <https://developer.ibm.com/articles/the-sequence-diagram/> (дата обращения: 21.06.2025).
18. BangBang Education. Пользовательский интерфейс [Электронный ресурс]. URL: <https://bangbangeducation.ru/point/ux-ui-dizain/polzovatelskij-interfejs/> (дата обращения: 21.06.2025).
19. Хабр. Практическое руководство по проектированию интерфейсов [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 21.06.2025).
20. Яндекс Практикум. Что такое прототип сайта [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/prototip-sajta-cto-eto-i-dlya-chego-nuzhen/> (дата обращения: 21.06.2025).
21. Буч, Г., Рамбо, Д., Якобсон, И. Язык UML: руководство пользователя / пер. с англ. — 2-е изд. — М.: ДМК Пресс, 2006. — 496 с. — ISBN 5-94074-334-X.
22. GeeksForGeeks. Differences Between Quality Assurance and Quality Control [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-engineering/differences-between-quality-assurance-and-quality-control/> (дата обращения: 21.06.2025).
23. Metrics for Software Quality [Электронный ресурс] // CMCons.com [архивировано]. URL: https://web.archive.org/web/20081018181130/http://cmcons.com/articles/CC_CQ/dev_metrics/mertics_part_1/ (дата обращения: 21.06.2025).

Приложение А
(справочное)
Глоссарий

Web-приложение – программное обеспечение, работающее на веб-сервере и предоставляющее пользователю доступ через браузер по сети (чаще всего через протокол HTTP/HTTPS). Включает клиентскую (frontend) и серверную (backend) части.

Фреймворк (framework) – набор библиотек, компонентов и соглашений, упрощающий разработку программного обеспечения.

Low-code / No-code – подходы к разработке программных решений с минимальным (или без) ручным программированием, чаще всего с использованием визуальных инструментов.

REST (Representational State Transfer) – архитектурный стиль взаимодействия клиент–сервер, основанный на стандартах HTTP.

Интернет вещей (IoT) – концепция, при которой физические устройства снабжены датчиками, ПО и средствами подключения к сети, что позволяет им обмениваться данными.

API (Application Programming Interface) – набор определений и протоколов для взаимодействия различных программных компонентов.

Wireframe – каркас пользовательского интерфейса, отображающий расположение основных элементов на страницах без оформления.

HTTP-запрос – запрос, отправляемый клиентом на сервер по протоколу HTTP/HTTPS. Используется для получения или отправки данных.

WebSocket – протокол для двусторонней связи между клиентом и сервером в режиме реального времени.

Плагин (Plugin) – модуль, расширяющий функциональность основной системы.

Приложение Б
(обязательное)
Реализация моделей

Листинг Б.1 – Реализация модели компонента пользователя

```
export interface User {  
  id: string;  
  username: string;  
  email: string;  
  passwordHash: string;  
  role: string;  
}
```

Листинг Б.2 – Реализация модели устройства

```
export interface Device {  
  device_id: string;  
  device_name: string;  
  device_type: string;  
  location_id: string;  
  status: boolean;  
  user_permissions: {  
user_id: string, permission_level: string  
  }[];  
}
```

Листинг Б.3– Реализация модели события

```
export interface Event {  
  id: string;  
  timestamp: string;  
  source: string;  
  type: string;  
  severity: string;  
  messages: string;  
  parameters: {  
    currentValue: number;  
    unit: string;  
  };  
  acknowledged: boolean;  
}
```

Приложение В
(обязательное)

Реализация контроллера компонента системы

Листинг В.1 – Реализация контроллера управления устройствами компонента системы

```
import { Plugin } from "../../core/types";
import { App } from "../../core/App";
import { authenticateJWT, authorizeAccess } from
"../../core/middleware/auth";

// Интерфейсы
interface IDeviceRepository {
  getUserDevices(userId: string): Promise<any[]>;
  getDeviceTypes(): Promise<any[]>;
  getUserLocationTree(userId: string): Promise<any[]>;
  createLocation(name: string, desc: string, parentId: string |
null): Promise<string>;
  deleteDevice(deviceId: string): Promise<any>;
  toggleDevice(deviceId: string): Promise<string>;
  createDevice(userId: string, name: string, type: string,
locationId: string): Promise<any>;
  updateDevice(params: UpdateDeviceDTO): Promise<any>;
}

interface UpdateDeviceDTO {
  device_id: string;
  device_name: string;
  device_type: string;
  location_id: string;
  status: boolean;
  user_permissions: { user_id: string, permission_level: string
}[];
}

// Реализация
class PostgresDeviceRepository implements IDeviceRepository {
  constructor(private db: any) {}

  async getUserDevices(userId: string) {
    return (await this.db.query("SELECT * FROM
get_user_devices($1)", [userId])).rows;
  }

  async getDeviceTypes() {
    return (await this.db.query("SELECT * FROM
devices_types")).rows;
  }

  async getUserLocationTree(userId: string) {
    return (await this.db.query("SELECT * FROM
get_user_location_tree($1)", [userId])).rows;
  }
}
```

```

    async createLocation(name: string, desc: string, parentId:
string | null) {
        const result = await this.db.query("SELECT * FROM
add_location($1, $2, $3)", [name, desc, parentId]);
        return result.rows[0].add_location;
    }

    async deleteDevice(deviceId: string) {
        return (await this.db.query("SELECT * FROM
delete_device($1)", [deviceId])).rows;
    }

    async toggleDevice(deviceId: string) {
        return (await this.db.query("SELECT * FROM
toggle_device_status($1)",
[deviceId])).rows[0].toggle_device_status;
    }

    async createDevice(userId: string, name: string, type: string,
locationId: string) {
        return (await this.db.query("SELECT * FROM add_device($1,
$2, $3, $4, $5)", [userId, name, type, locationId,
'admin'])).rows[0];
    }

    async updateDevice({ device_id, device_name, device_type,
location_id, status, user_permissions }: UpdateDeviceDTO) {
        return (await this.db.query("SELECT * FROM update_device($1,
$2, $3, $4, $5, $6)", [
            device_id,
            device_name,
            device_type,
            location_id,
            status,
            JSON.stringify(user_permissions)
        ])).rows[0].update_device;
    }
}

// Плагин
const DevicesLogicPlugin: Plugin = {
    name: "DevicesLogicPlugin",

    async load(app: App) {
        const db = (app as any).db;
        const repo: IDeviceRepository = new
PostgresDeviceRepository(db);

        app.server.get("/devices/userDevices", authenticateJWT,
async (req: any, res: any) => {
            try {
                const userId = req.user?.id;

```

```

        if (!userId) return res.status(400).json({ message:
"User ID not found" });
        const devices = await repo.getUserDevices(userId);
        res.json({ devices });
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: "Failed to fetch user
devices" });
    }
});

app.server.get("/devices/deviceTypes", authenticateJWT,
async (_req: any, res: any) => {
    try {
        const types = await repo.getDeviceTypes();
        res.json({ device_types: types });
    } catch (err) {
        res.status(500).json({ message: "Failed to fetch device
types" });
    }
});

app.server.get("/devices/deviceLocations", authenticateJWT,
async (req: any, res: any) => {
    try {
        const userId = req.user?.id;
        const locations = await
repo.getUserLocationTree(userId);
        const maxDepth = Math.max(...locations.map((loc: any) =>
loc.depth ?? 0));
        const inverted = locations.map((loc: any) => ({ ...loc,
depth: maxDepth - loc.depth }));
        res.json({ location: inverted });
    } catch (err) {
        res.status(500).json({ message: "Failed to fetch
locations" });
    }
});

app.server.post("/devices/createLocation", authenticateJWT,
async (req: any, res: any) => {
    const { locationName, description, parent_id = null } =
req.body;
    try {
        const location_id = await
repo.createLocation(locationName, description, parent_id);
        res.status(201).json({ location_id });
    } catch (err) {
        res.status(500).json({ message: "Failed to create
location" });
    }
});

```

```

    app.server.delete("/devices/delete-device", authenticateJWT,
authorizeAccess("delete"), async (req: any, res: any) => {
    try {
        const result = await
repo.deleteDevice(req.body.device_id);
        res.json({ message: result });
    } catch (err) {
        res.status(500).json({ error: "Internal server error"
});
    }
});

    app.server.put("/devices/toggle-device", authenticateJWT,
authorizeAccess("update"), async (req: any, res: any) => {
    try {
        const status = await
repo.toggleDevice(req.body.device_id);
        res.json({ device_status: status });
    } catch (err) {
        res.status(500).json({ error: "Internal server error"
});
    }
});

    app.server.post("/devices/create-device", authenticateJWT,
authorizeAccess("write"), async (req: any, res: any) => {
        const { user_id, device_name, device_type, location_id } =
req.body;
        try {
            const device = await repo.createDevice(user_id,
device_name, device_type, location_id);
            res.json({ device_id: device });
        } catch (err) {
            res.status(500).json({ error: "Internal server error"
});
        }
    });

    app.server.put("/devices/update-device", authenticateJWT,
authorizeAccess("update"), async (req: any, res: any) => {
        const { device_id, device_name, device_type, location_id,
status, user_permissions = [] } = req.body;
        try {
            const updated = await repo.updateDevice({
                device_id,
                device_name,
                device_type,
                location_id,
                status,
                user_permissions: user_permissions.map((u: any) => ({
user_id: u.user_id, permission_level: "admin" })),

```

```

        });
        res.json({ device: updated });
    } catch (err) {
        res.status(500).json({ error: "Internal server error"
    });
    }
    });
},

async unload() {
    console.log("🔌 DeviceLogicPlugin unloaded");
},
};

export default DevicesLogicPlugin;

```

Приложение Г
(обязательное)

Реализация представления компонента системы

Листинг Г.1 – Реализация представления страницы текущих событий

КОМПОНЕНТА СИСТЕМЫ

```
"use client"
// events/current/page.js
import { useEffect, useRef, useState } from "react";
import { useSearchParams } from "next/navigation";

import Layout from "../../components/Layout";
import DeviceCard from "../../components/DeviceCard";

import { loadPlugins } from "../../lib/loadWidgets";
import { WidgetRegistryProvider } from
"../../lib/WidgetRegistry";
import Widget from "../../components/Widget";

import { Plus } from "lucide-react";
import { connectSocket, getSocket } from "../../lib/socket";
import axios from "axios";
import { createDevice, deleteDevice, toggleDevice, updateDevice
} from "@app/api/deviceService";
import AddOrEditDeviceModal from
"@app/components/AddorEditDeviceModal";
import { useLocations } from "@app/context/LocationsContext";
import toast from 'react-hot-toast';

export default function Dashboard() {
  const searchParams = useSearchParams();
  const room = searchParams.get("room") || "Room A";
  const [widgets, setWidgets] = useState([]);

  const [devices, setDevices] = useState([]);
  const [chartData, setChartData] = useState([]);

  const [isModalOpen, setIsModalOpen] = useState(false);
  const [newType, setNewType] = useState("chart1");
  const [newDevice, setNewDevice] = useState(devices[0]?.type ||
  "");
  const [user, setUser] = useState(null);
  const [accessToken, setAccessToken] = useState(null);

  const { locations, setLocations } = useLocations();
  const [editingDevice, setEditingDevice] = useState(null);

  const openModal = () => setIsModalOpen(true);
  const closeModal = () => setIsModalOpen(false);
```

```

useEffect(() => {
  // Загружаем пользователя из localStorage
  const storedUser = localStorage.getItem('user');
  if (storedUser) {
    setUser(JSON.parse(storedUser));
  }
  const storedAccessToken =
localStorage.getItem('accessToken');
  if (storedAccessToken) {
    setAccessToken(storedAccessToken);
  }

}, []);

useEffect(() => {
  if (!user || !accessToken) return;

  const fetchDevices = async () => {
    try {
      const res = await
axios.get("http://localhost:5731/user/devices", {
        headers: {
          Authorization: `Bearer ${accessToken}`,
        },
      });
    }

    const parsedDevices = res.data.devices.map((d) =>
({
      id: d.device_id,
      name: d.device_name,
      type: d.device_type_name,
      room: d.location,
      status: d.status ? "online" : "offline",
      reading:
        d.last_event?.parameters?.value !== undefined
        ?
`${d.last_event.parameters.value}${d.last_event.parameters.unit
|| ''}`
        : "-",
      lastEvent: d.last_event || null,
    }));

    setDevices(parsedDevices);
  } catch (err) {
    console.error("Ошибка загрузки устройств:", err);
  }
});

fetchDevices();
}, [user, accessToken]);

```

```

useEffect(() => {
  if (!devices.length || chartData.length > 0) return;

  const randomVariant = () => (Math.random() > 0.5 ?
"line" : "bar");

  const generatedChartData = devices.map((device) => ({
    id: device.id,
    name: `График: ${device.name}`,
    type: "chart1",
    variant: randomVariant(),
    unit: device.lastEvent?.parameters?.unit || "",
    sourceType: device.type,
    room: device.room, // ← добавляем!
    dataValue: [],
  }));

  const logTable = {
    name: 'Лог событий',
    type: 'table',
    dataValue: []
  };

  setChartData([...generatedChartData, logTable]);
}, [devices]);

useEffect(() => {
  if (!user) return;
  const socket = connectSocket(user.id);

  socket.on("device-event", (event) => {
    handleEvent(event);
  });

  return () => {
    socket.off("device-event");
    socket.disconnect();
  };
}, [user]);

useEffect(() => {
  loadPlugins().then(setWidgets);
}, []);

const handleEvent = (event) => {
  const { messages, severity, timestamp, parameters,
source, id, type } = event;

  const value = parameters?.value;

```

```

const label = new Date(timestamp).toLocaleTimeString();

if (severity === "Critical Error" || severity ===
"Error") {
  toast((t) => (
    <div>
    <span className="mr-5">
      🗨 {messages}
    </span>
    <button onClick={() => toast.dismiss(t.id)}>
      Ok
    </button>
    </div>
  ));
}

// Обновление таблицы
setChartData((prev) =>
  prev.map((w) =>
    w.type === "table"
    ? {
      ...w,
      dataValue: [
        {
          label,
          value: messages || `Событие: ${id}`,
          severity: severity?.toLowerCase() ||
"information",
          source: source,
          type: type
        },
        ...w.dataValue.slice(0, 49),
      ],
    }
    : w
  )
);

const deviceId = source.replace("device:", "");

// Обновление карточки устройства
setDevices((prev) =>
  prev.map((device) =>
    device.name === deviceId
    ? {
      ...device,
      reading: value !== undefined
        ? `${value}${parameters?.unit || ''}`
        : "-",
      lastEvent: event,
    }
  )
);

```

```

        : device
      )
    );

    // Обновление графика
    if (value !== undefined) {
      setChartData((prev) =>
        prev.map((w) =>
          w.type.startsWith("chart") &&
w.name.includes(deviceId)
            ? {
                ...w,
                dataValue: [...w.dataValue.slice(-6), {
label, value }],
              }
            : w
          )
        );
    }
  };

  const handleDeleteDevice = async (id) => {
    await deleteDevice(id);
    setDevices((prev) => prev.filter((d) => d.id !== id));
    setChartData((prev) => prev.filter((d) => d.id !==
id));
  };

  const handleToggleDevice = async (device) => {
    await toggleDevice(device.id);
    setDevices((prev) =>
      prev.map((d) =>
        d.id === device.id
          ? { ...d, status: d.status === "online" ?
"offline" : "online" }
          : d
        )
      );
  };

  const handleEditDevice = (device) => {
    setEditingDevice(device);
    openModal();
  };

  const handleSaveDevice = async ({ id, name, type, room,
status, users }) => {
    try {
      let deviceData;

```

```

        if (editingDevice) {
            deviceData = await updateDevice(id, name, type,
room, status, users);
            setDevices((prev) =>
                prev.map((d) =>
                    d.id === id
                        ? {
                            ...d,
                            name,
                            type,
                            room: deviceData?.device,
                            status: `${status? 'online': 'offline'}`
                        }
                        : d
                )
            );

            setChartData((prevChartData) =>
                prevChartData.map((w) =>
                    w.id === id
                        ? {...w,
                            name: `График: ${name}`,
                            room: deviceData?.device,
                        }:w
                )
            );
        } else {
            const newDevice = await createDevice(name, type,
room);

            const created = newDevice?.device_id;
            if (!created) {
                console.error("Не удалось создать устройство.");
                return;
            }

            setDevices((prev) => [
                ...prev,
                {
                    id: created.device_id,
                    name,
                    type,
                    room: created.full_location,
                    status: "offline",
                    reading: created.event_parameters.value +
created.event_parameters.unit,
                    lastEvent: null,
                },
            ]);

            // Добавление графика только при создании
устройства

```

```

        const randomVariant = () => (Math.random() > 0.5 ?
"line" : "bar");
        setChartData((prevChartData) => [
            ...prevChartData,
            {
                id: created.device_id,
                name: `График: ${name}`,
                type: "chart1",
                variant: randomVariant(),
                unit: created.event_parameters.unit,
                sourceType: type,
                room: created.full_location, // ← добавляем
                dataValue: [],
            },
        ]);
    }
    } catch (err) {
        console.error("Ошибка при сохранении устройства:",
err);
    }
};

return (
    <>
    <WidgetRegistryProvider widgets={widgets}>
    <Layout>
    <div className="p-4 space-y-8">
        {/* ===== Устройства ===== */}
        <section>
            <div className="flex items-center justify-between
mb-4">
                <h2 className="text-2xl font-semibold text-black
dark:text-white">📱 Устройства</h2>
                <button
                    onClick={() => {
                        setEditingDevice(null);
                        openModal();
                    }}
                    className="text-sm bg-blue-600 hover:bg-blue-700
text-black dark:text-white px-3 py-1 rounded flex items-center
gap-2"
                >
                    <Plus size={16} /> Добавить устройство
                </button>
            </div>
            <div className="grid grid-cols-1 md:grid-cols-3 gap-
6">
                {devices
                    .filter((device) => device.room === room)
                    .map((device) => (
                        <DeviceCard

```

```

        key={device.id}
        device={device}
        onDelete={handleDeleteDevice}
        onToggle={handleToggleDevice}
        onEdit={handleEditDevice}/>
      )}
    </div>
  </section>

  {/* ===== Виджеты: Графики ===== */}
  <section>
    <div className="flex items-center justify-between
mb-4">
      <h2 className="text-2xl font-semibold text-black
dark:text-white">📊 Графики</h2>
      <button
        onClick={openModal}
        className="text-sm bg-blue-600 hover:bg-blue-700
text-black dark:text-white px-3 py-1 rounded flex items-center
gap-2"
        >
        <Plus size={16} /> Добавить виджет
      </button>
    </div>
    <div className="grid grid-cols-1 md:grid-cols-2 gap-
6">
      {chartData
        .filter(w => w.type === 'chart1' && w.room ===
room)
        .map((tdata, index) => (
          <Widget key={`widget-${index}`} data={tdata} />
        ))}
    </div>
  </section>

  {/* ===== Виджеты: Таблицы ===== */}
  <section>
    <h2 className="text-2xl font-semibold text-black
dark:text-white mb-4">📋 Таблицы</h2>
    <div className="grid grid-cols-1 md:grid-cols-1 gap-
6">
      {chartData.filter(w => w.type ===
'table').map((tdata, index) => (
        <Widget key={`table-${index}`} data={tdata} />
      ))}
    </div>
  </section>

```

```

</div>

<AddOrEditDeviceModal
  isOpen={isOpen}
  onClose={() => setIsModalOpen(false)}
  onSubmit={handleSaveDevice}
  device={editingDevice} // или null
  rooms={locations}
/>

</Layout>
</WidgetRegistryProvider>
</>
);
}

```

Приложение Д
(рекомендуемое)

Визуальные прототипы пользовательского интерфейса

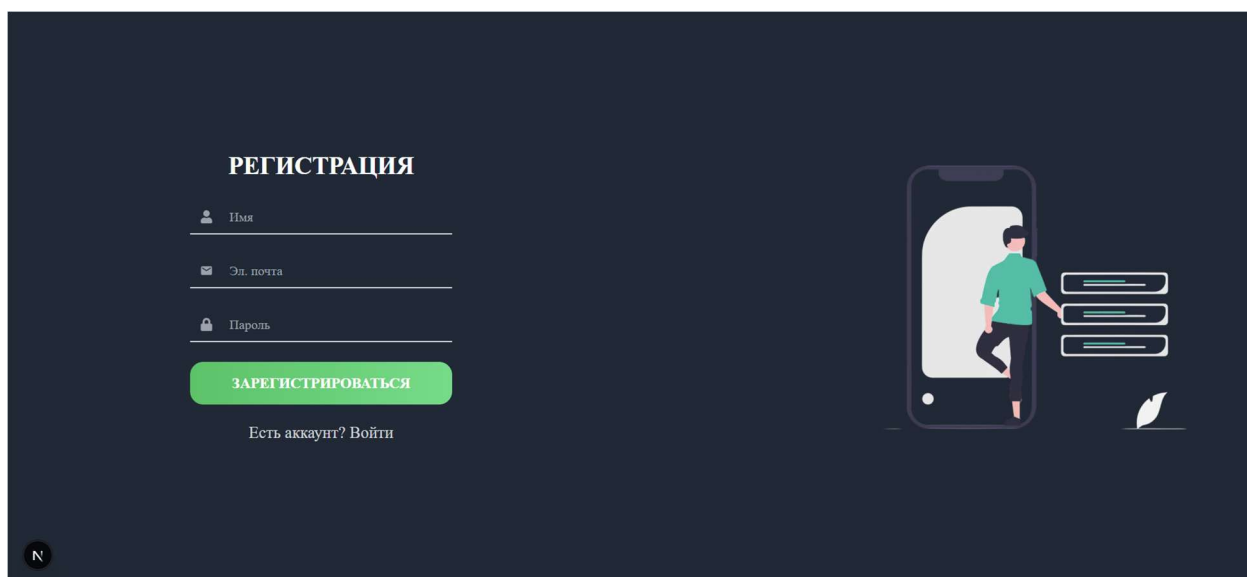


Рисунок Д.1 – Страница Регистрации

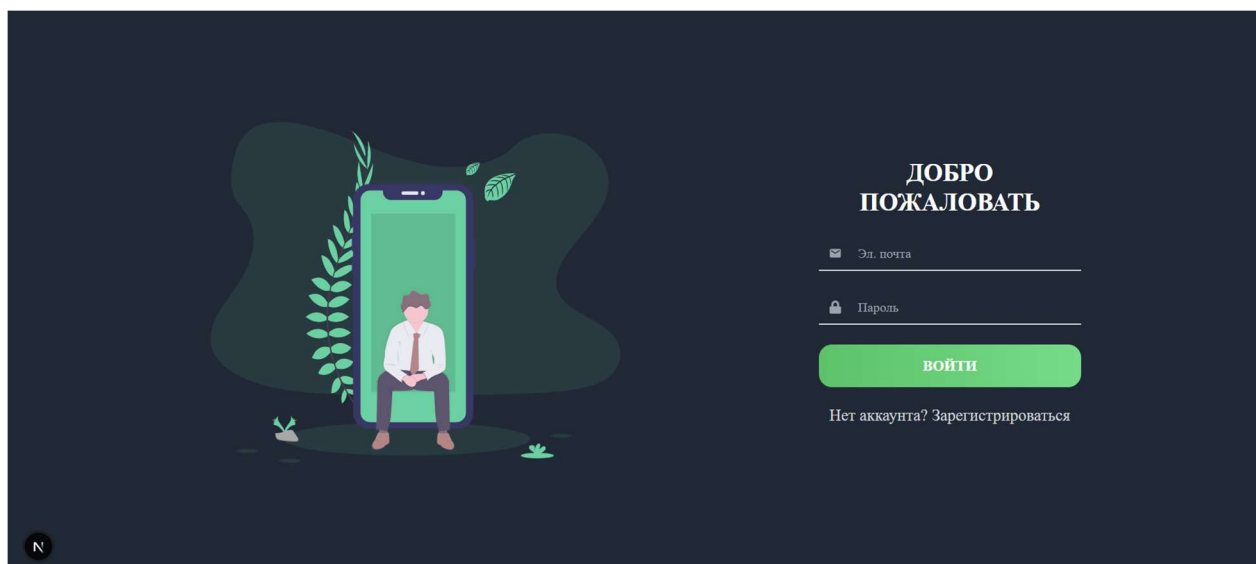


Рисунок Д.2 – Страница входа



Рисунок Д.3 – Страница текущих событий

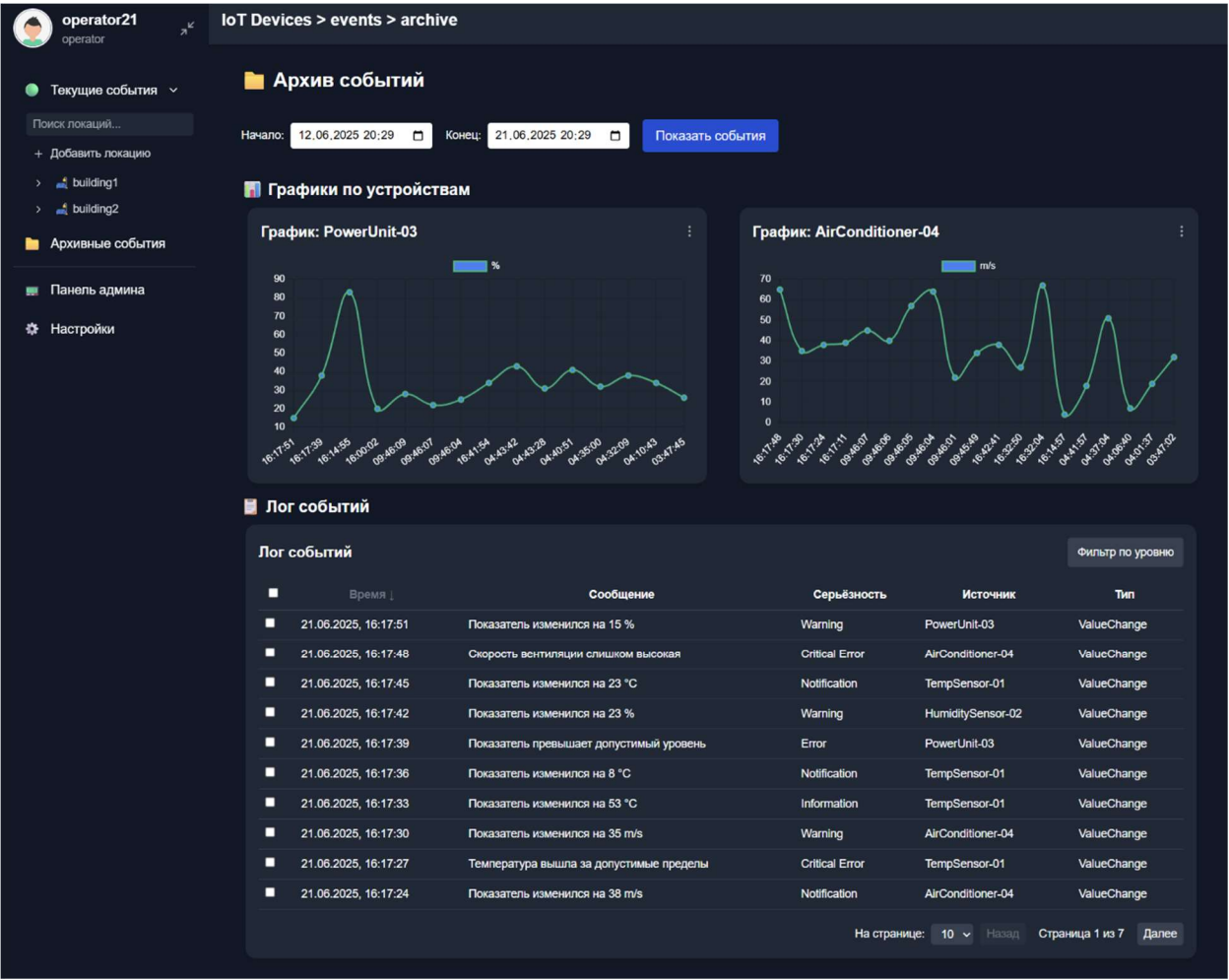


Рисунок Д.4 – Страница архивных событий

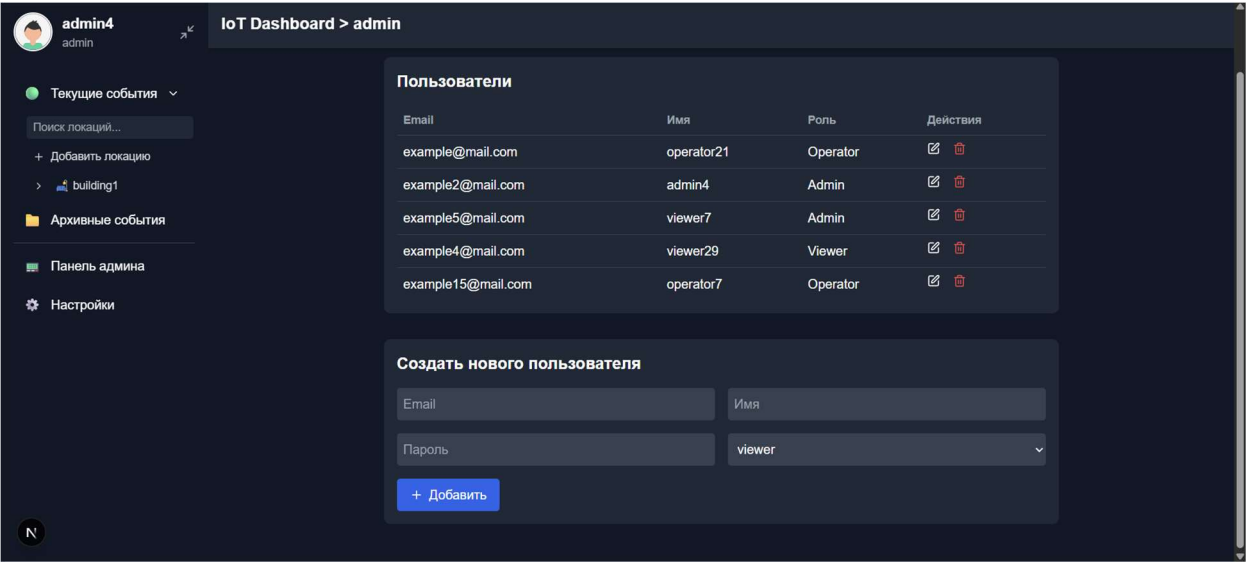


Рисунок Д.5 – Страница администратора

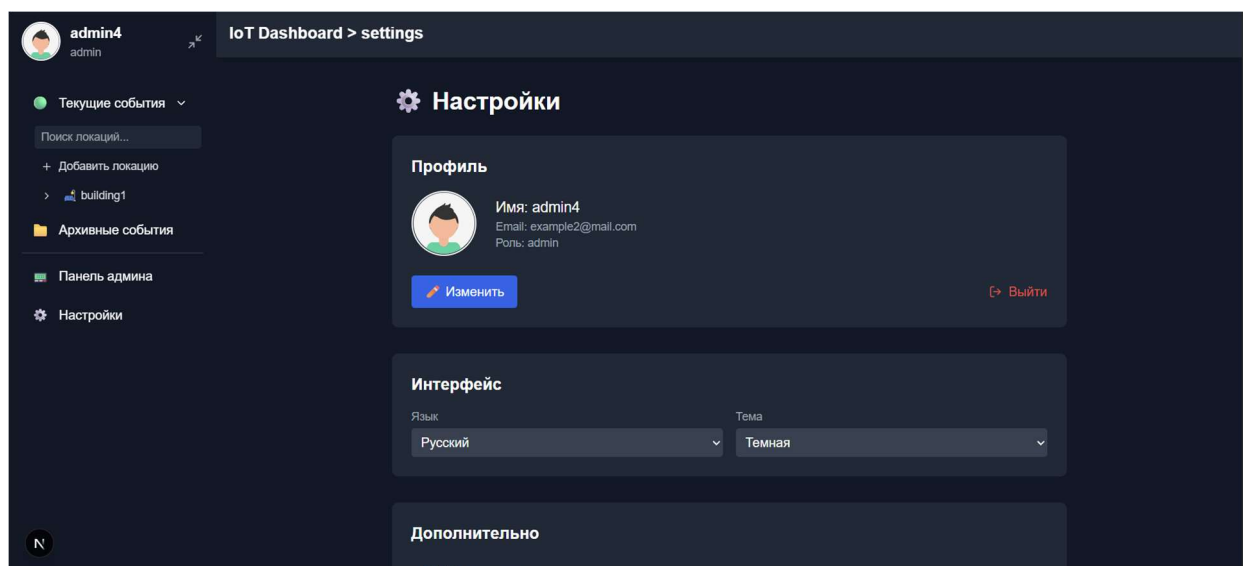


Рисунок Д.6 – Страница настроек системы

Приложение Е
(рекомендуемое)
Результаты функционального тестирования

Добавить устройство

- ⚠️ Некорректная длина названия
- ⚠️ Некорректные символы в составе названия
- ⚠️ Некорректная длина идентификатора устройства
- ⚠️ Некорректные символы в идентификаторе устройства

Имя устройства
Sr@H

Тип устройства
temperature

Комната
roomC

Серийный номер
A1B%|

Добавить устройство

Рисунок Е.1 – Функциональный тест 1

Добавить устройство

- ⚠️ Некорректные символы в составе названия
- ⚠️ Некорректная длина идентификатора устройства
- ⚠️ Некорректные символы в идентификаторе устройства

Имя устройства
Smart#\$Home24

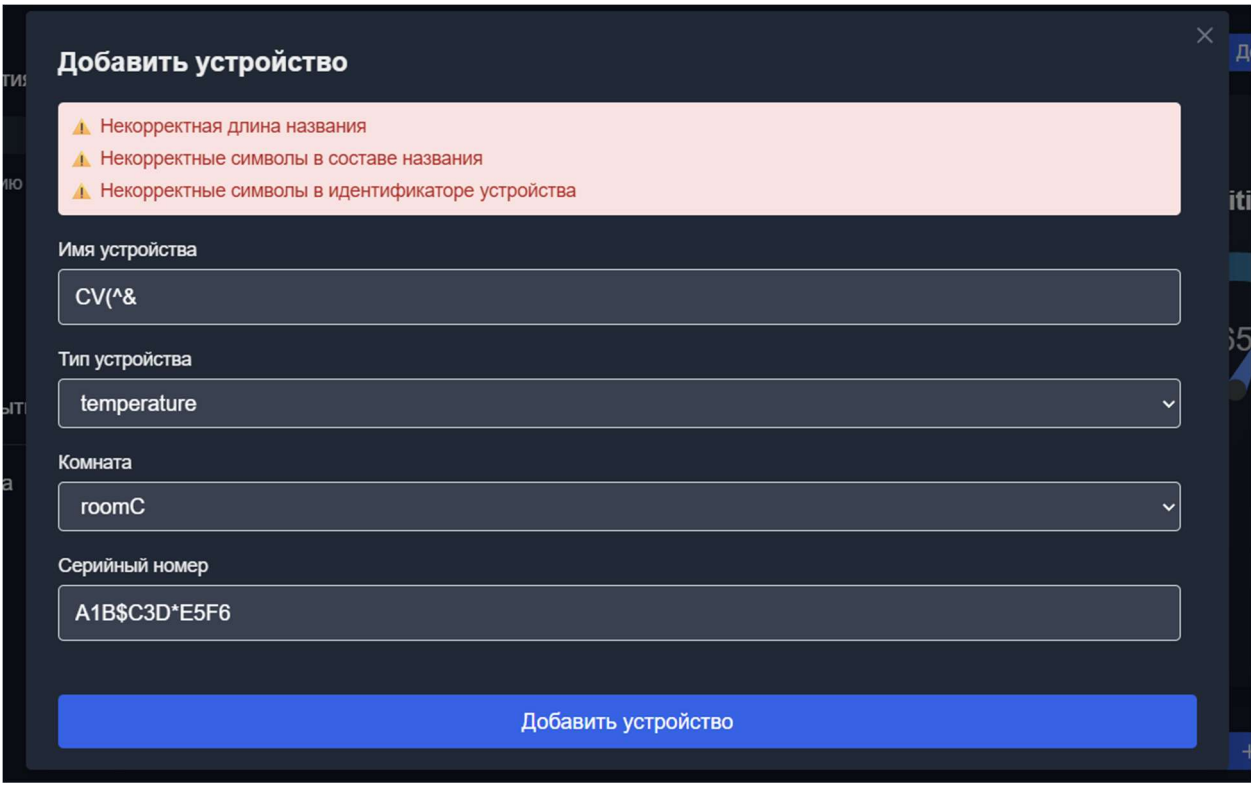
Тип устройства
temperature

Комната
roomC

Серийный номер
U*()C3D4

Добавить устройство

Рисунок Е.2 – Функциональный тест 2



Добавить устройство

- ⚠ Некорректная длина названия
- ⚠ Некорректные символы в составе названия
- ⚠ Некорректные символы в идентификаторе устройства

Имя устройства
CV(^&

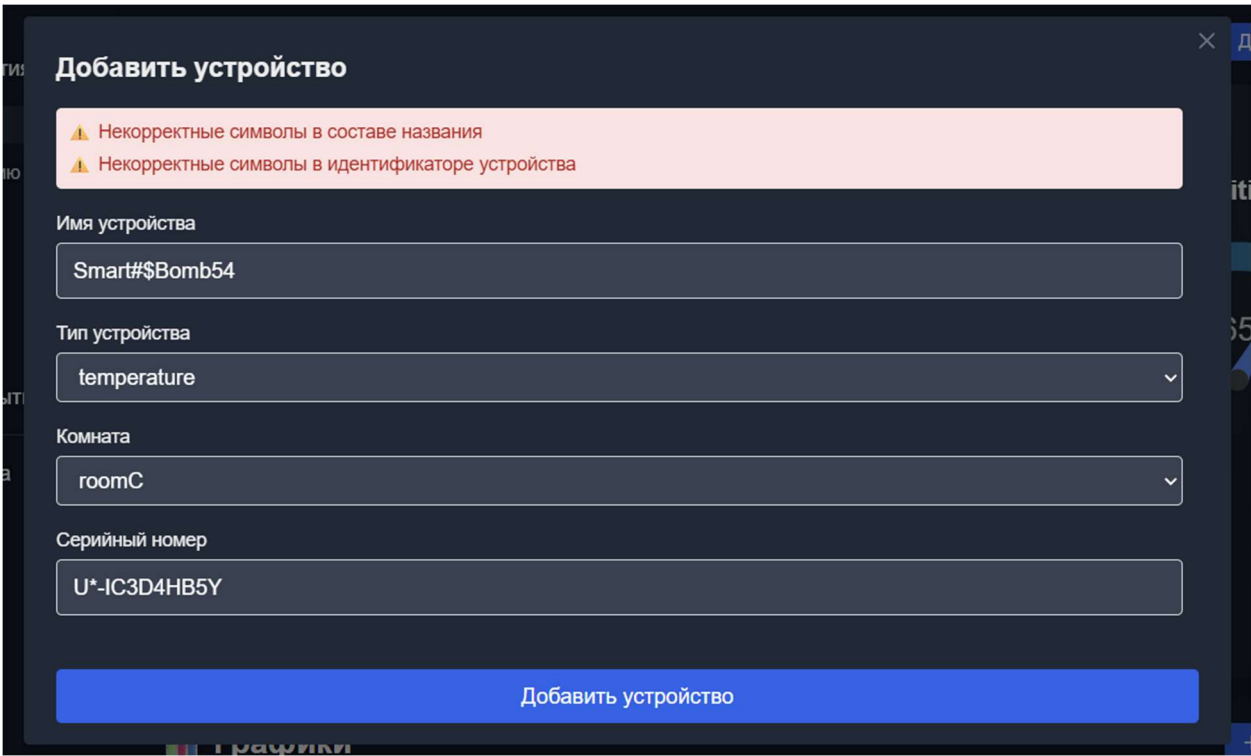
Тип устройства
temperature

Комната
roomC

Серийный номер
A1B\$C3D*E5F6

Добавить устройство

Рисунок Е.3 – Функциональный тест 3



Добавить устройство

- ⚠ Некорректные символы в составе названия
- ⚠ Некорректные символы в идентификаторе устройства

Имя устройства
Smart#\$Bomb54

Тип устройства
temperature

Комната
roomC

Серийный номер
U*-IC3D4HB5Y

Добавить устройство

Рисунок Е.4 – Функциональный тест 4

Добавить устройство

- ⚠ Некорректная длина названия
- ⚠ Некорректная длина идентификатора устройства
- ⚠ Некорректные символы в идентификаторе устройства

Имя устройства
S-b54

Тип устройства
temperature

Комната
roomC

Серийный номер
M%O45Y

Добавить устройство

Рисунок Е.5 – Функциональный тест 5

Добавить устройство

- ⚠ Некорректная длина идентификатора устройства
- ⚠ Некорректные символы в идентификаторе устройства

Имя устройства
Smart-Doom

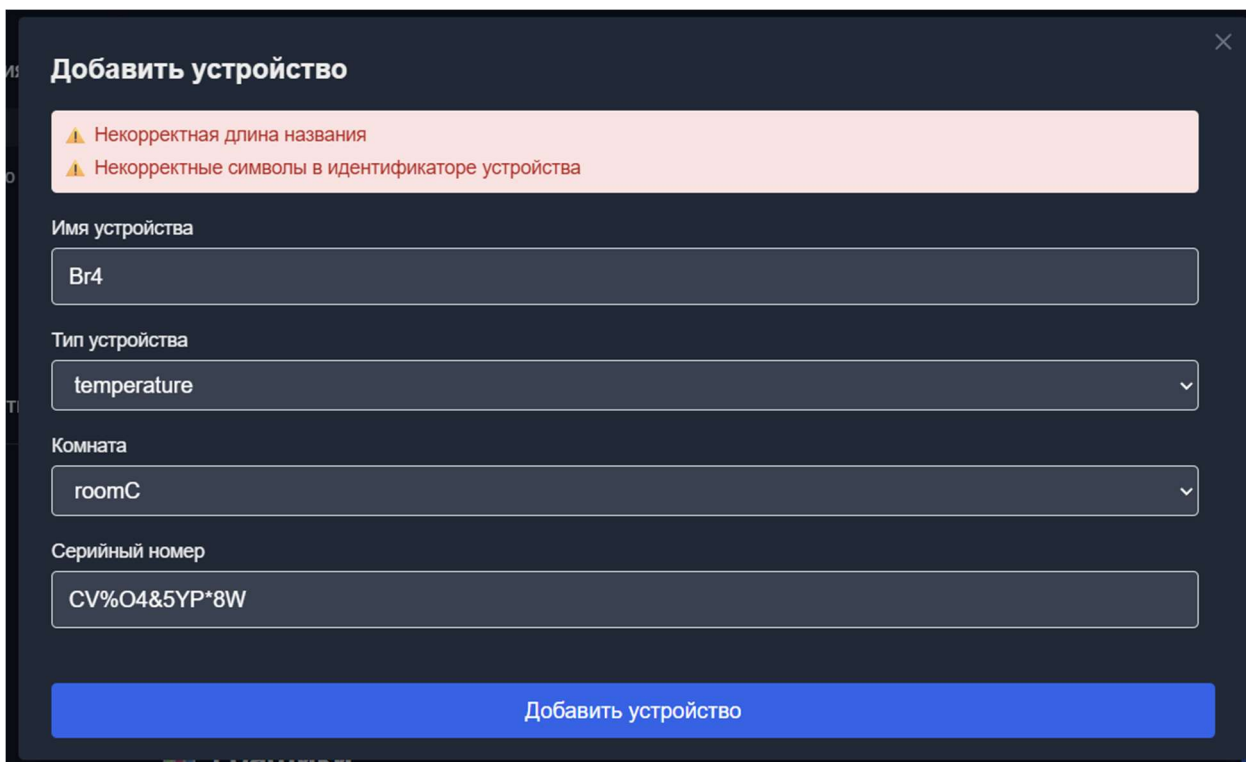
Тип устройства
temperature

Комната
roomC

Серийный номер
IO%O4

Добавить устройство

Рисунок Е.6 – Функциональный тест 6



Добавить устройство

⚠ Некорректная длина названия
⚠ Некорректные символы в идентификаторе устройства

Имя устройства
Br4

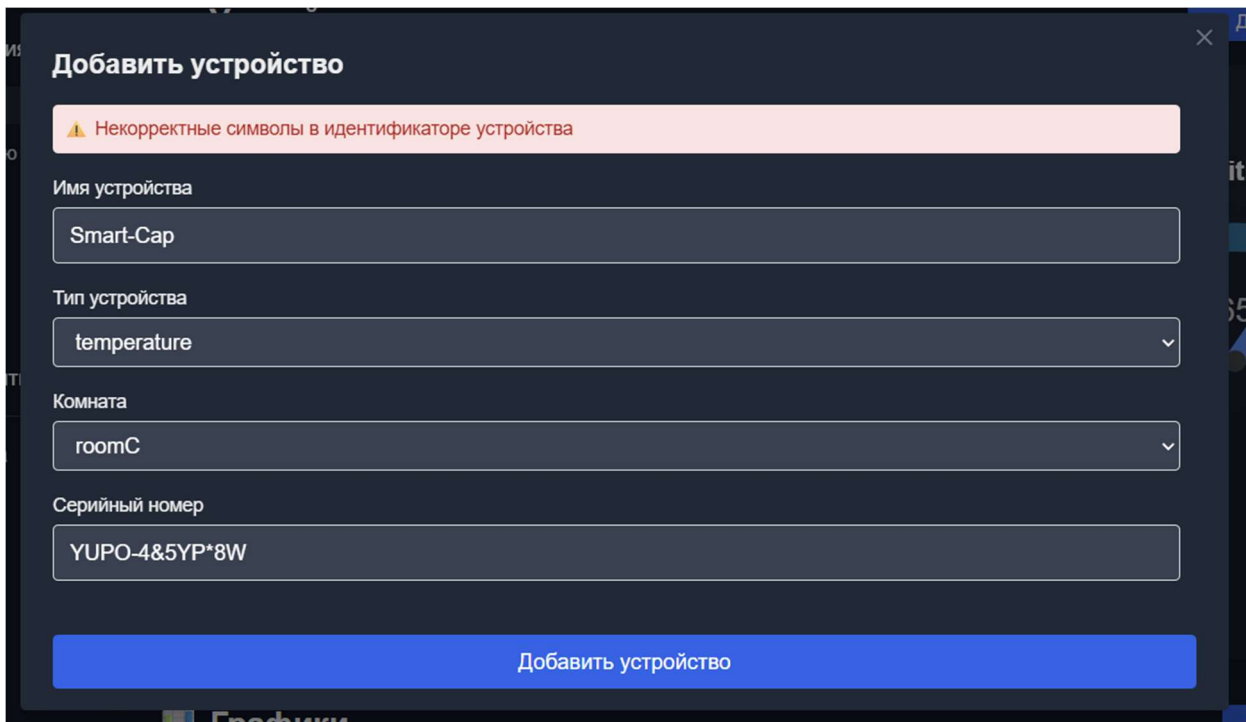
Тип устройства
temperature

Комната
roomC

Серийный номер
CV%O4&5YP*8W

Добавить устройство

Рисунок Е.7 – Функциональный тест 7



Добавить устройство

⚠ Некорректные символы в идентификаторе устройства

Имя устройства
Smart-Cap

Тип устройства
temperature

Комната
roomC

Серийный номер
YUPO-4&5YP*8W

Добавить устройство

Рисунок Е.8 – Функциональный тест 8

Добавить устройство

- ⚠️ Некорректная длина названия
- ⚠️ Некорректные символы в составе названия
- ⚠️ Некорректная длина идентификатора устройства

Имя устройства
Sr@H

Тип устройства
temperature

Комната
roomC

Серийный номер
A1BN

Добавить устройство

Рисунок Е.9 – Функциональный тест 9

Добавить устройство

- ⚠️ Некорректные символы в составе названия
- ⚠️ Некорректная длина идентификатора устройства

Имя устройства
Sr&^Home

Тип устройства
temperature

Комната
roomC

Серийный номер
M1BN

Добавить устройство

Рисунок Е.10 – Функциональный тест 10

Добавить устройство

⚠️ Некорректная длина названия
⚠️ Некорректные символы в составе названия

Имя устройства
S#&e

Тип устройства
temperature

Комната
roomC

Серийный номер
A1B2C3D4E5F6

Добавить устройство

Рисунок Е.11 – Функциональный тест 11

Добавить устройство

⚠️ Некорректные символы в составе названия

Имя устройства
*Smart#&eBrт

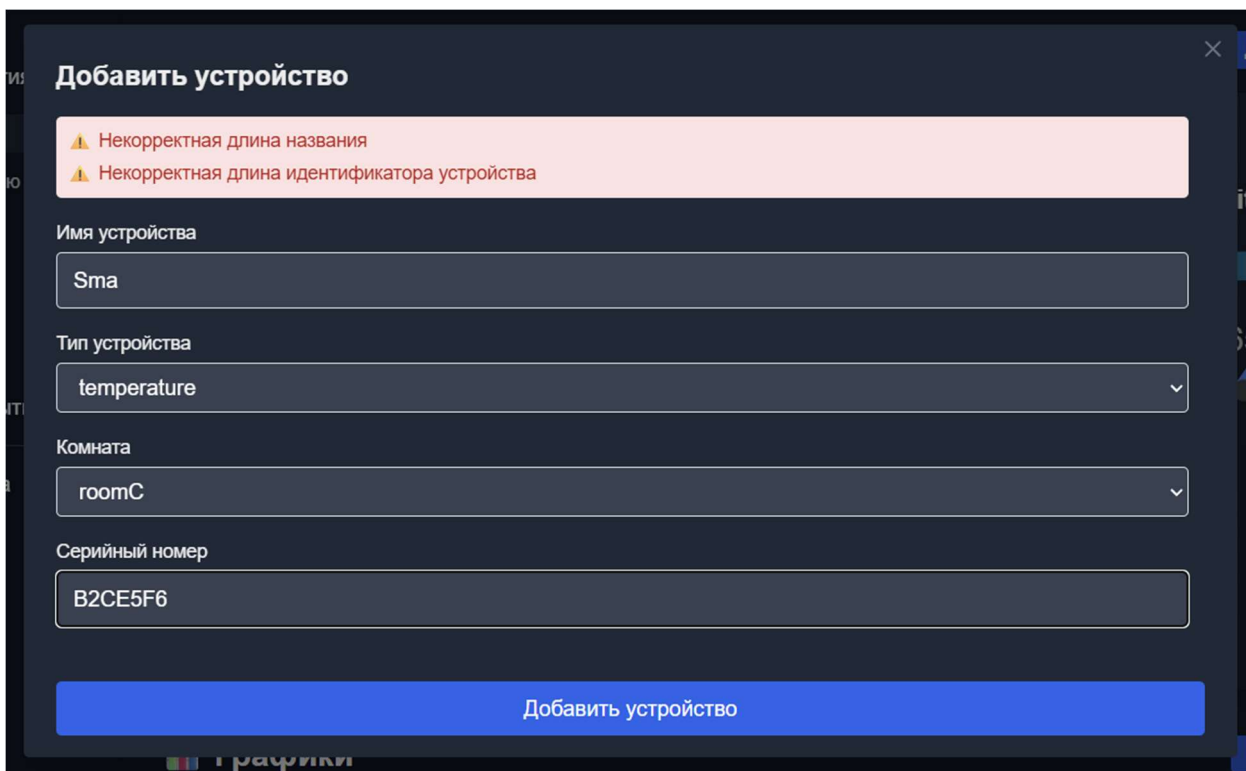
Тип устройства
temperature

Комната
roomC

Серийный номер
A1B2C3D4E5F6

Добавить устройство

Рисунок Е.12 – Функциональный тест 12



Добавить устройство

⚠ Некорректная длина названия
⚠ Некорректная длина идентификатора устройства

Имя устройства
Sma

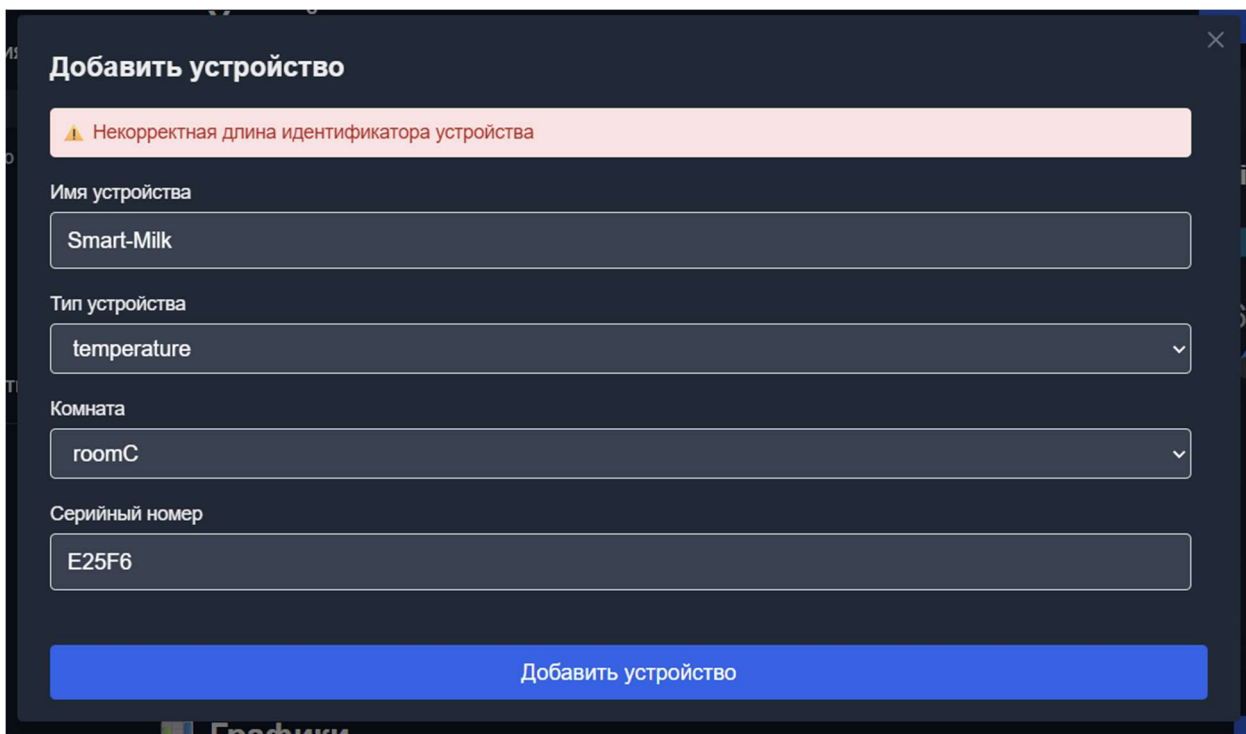
Тип устройства
temperature

Комната
roomC

Серийный номер
B2CE5F6

Добавить устройство

Рисунок Е.13 – Функциональный тест 13



Добавить устройство

⚠ Некорректная длина идентификатора устройства

Имя устройства
Smart-Milk

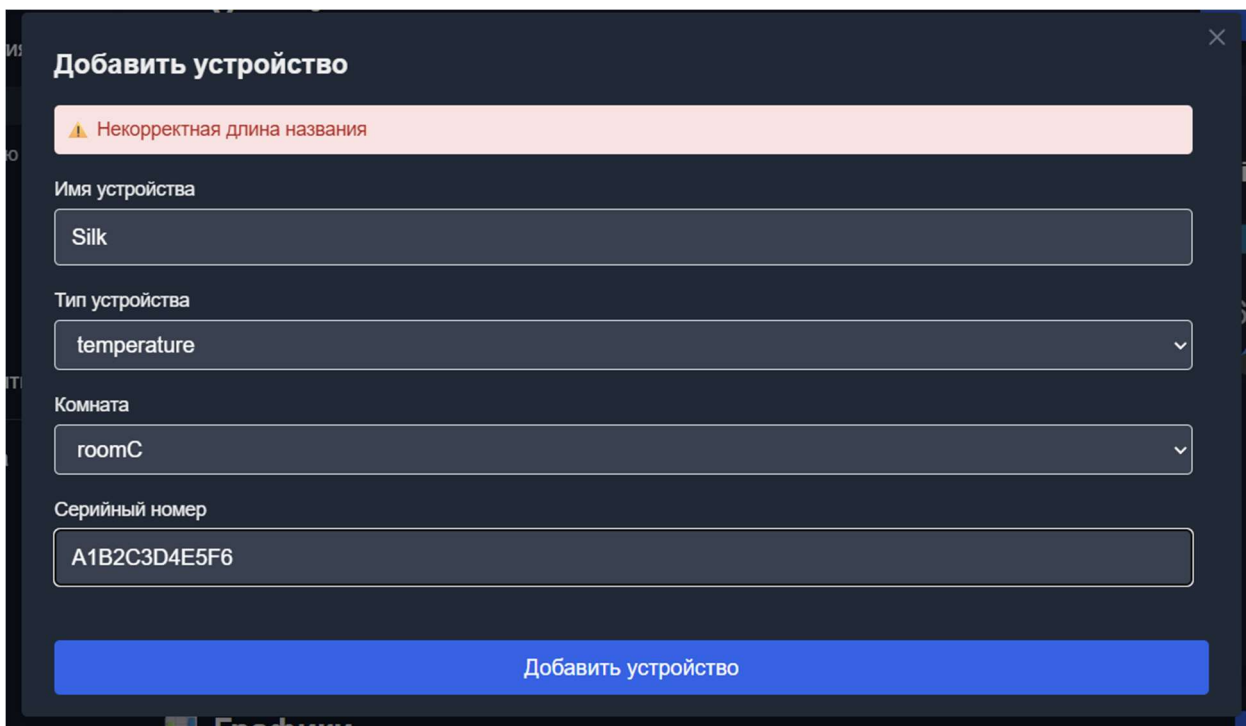
Тип устройства
temperature

Комната
roomC

Серийный номер
E25F6

Добавить устройство

Рисунок Е.14 – Функциональный тест 14



Добавить устройство

⚠ Некорректная длина названия

Имя устройства
Silk

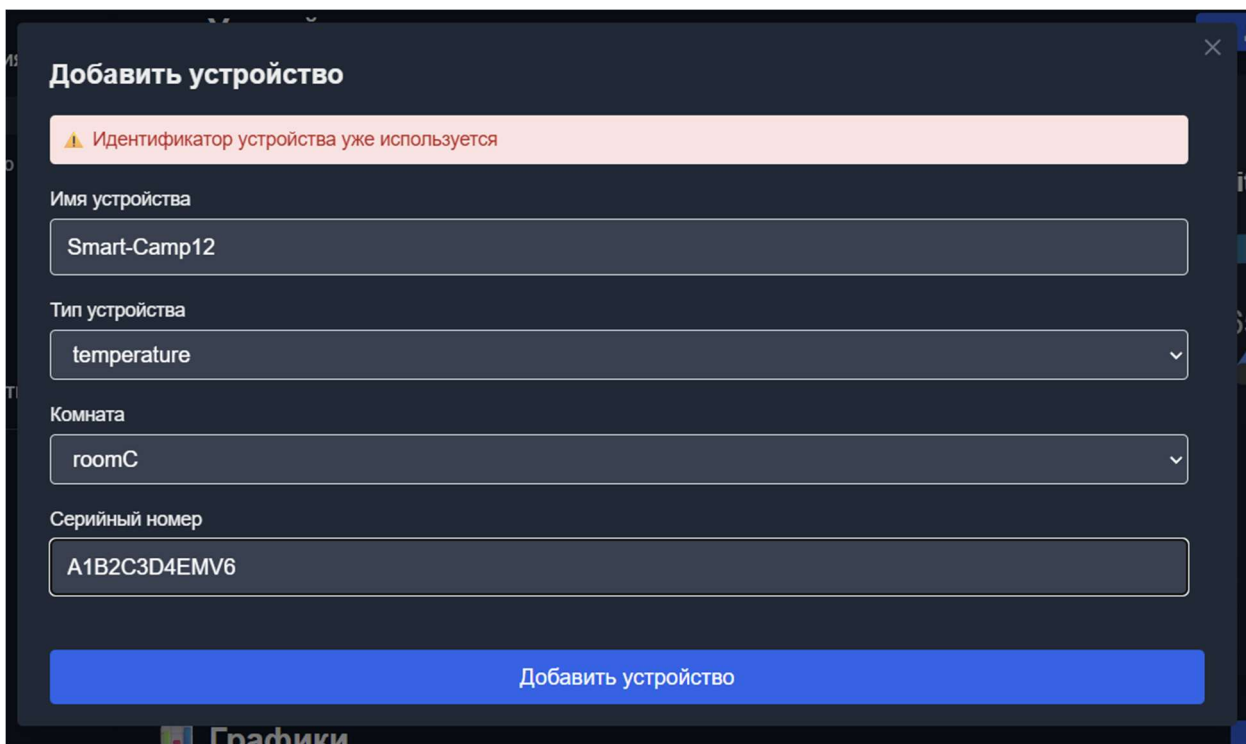
Тип устройства
temperature

Комната
roomC

Серийный номер
A1B2C3D4E5F6

Добавить устройство

Рисунок Е.15 – Функциональный тест 15



Добавить устройство

⚠ Идентификатор устройства уже используется

Имя устройства
Smart-Camp12

Тип устройства
temperature

Комната
roomC

Серийный номер
A1B2C3D4EMV6

Добавить устройство

Рисунок Е.16 – Функциональный тест 16

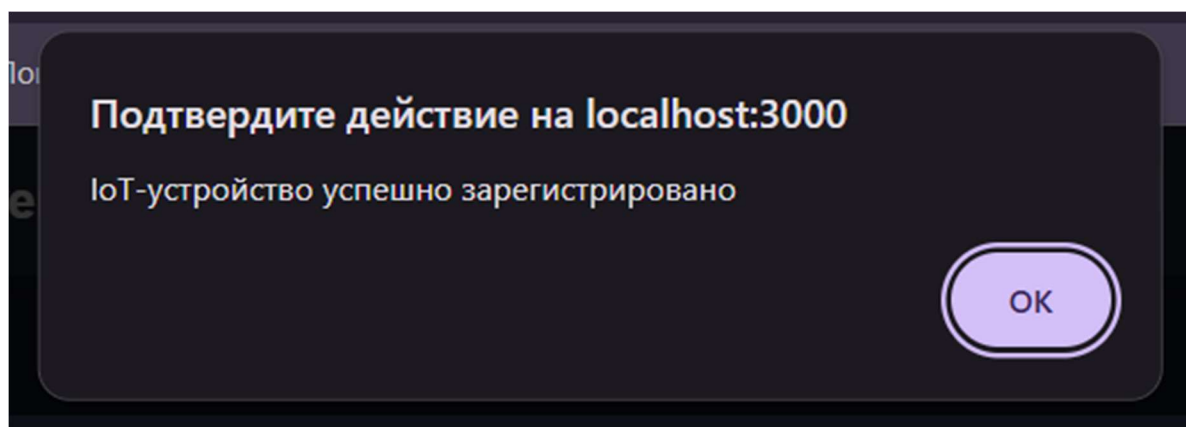


Рисунок Е.17 – Функциональный тест 17

Приложение Ж
(рекомендуемое)
Результаты конфигурационного тестирования

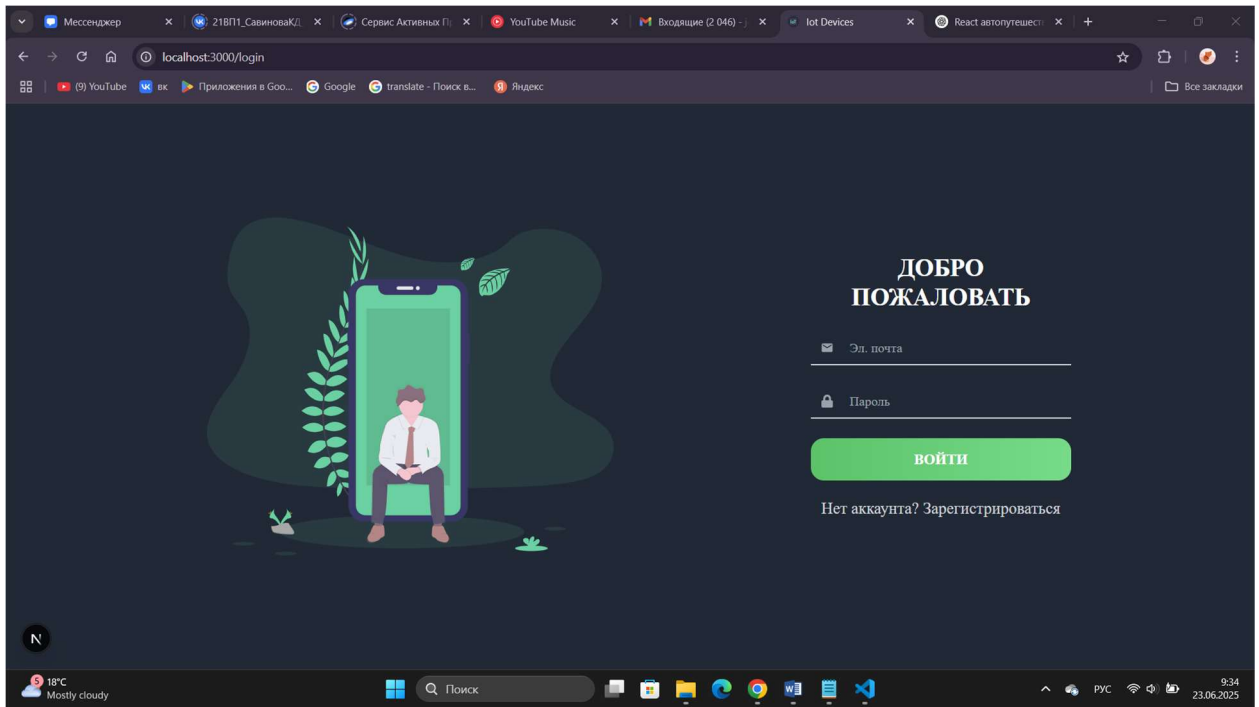


Рисунок Ж.1 – Конфигурационный тест 1

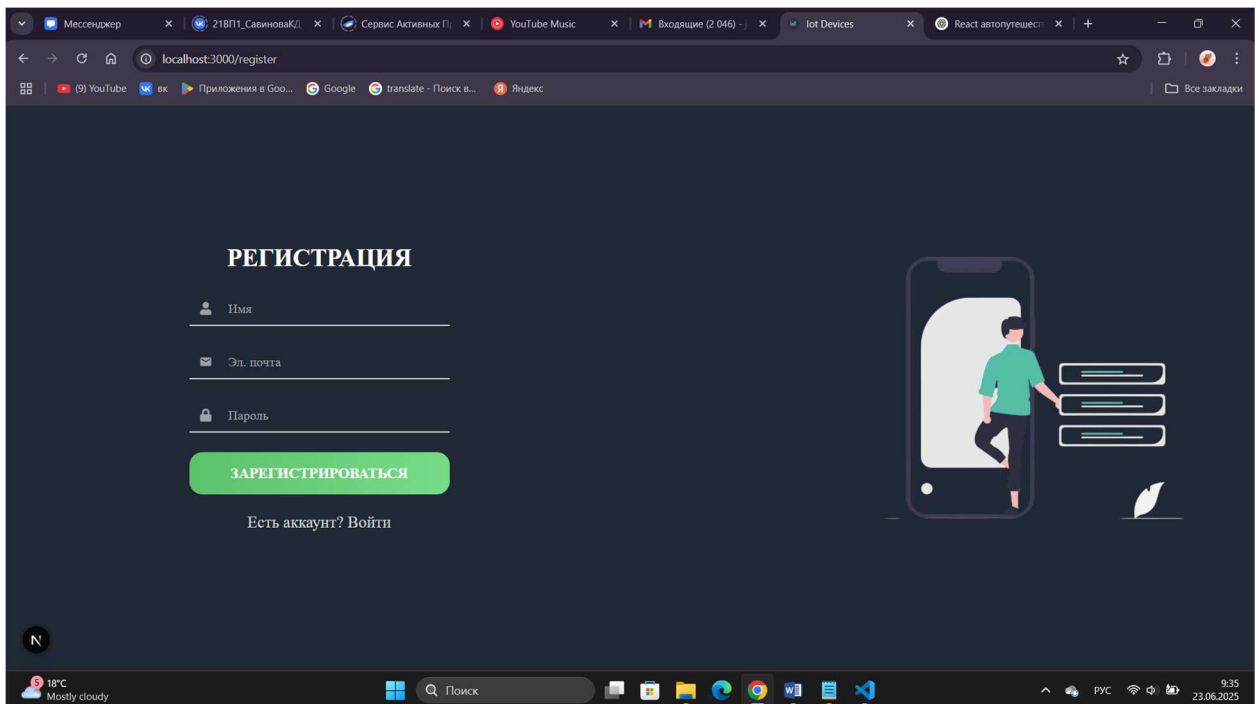


Рисунок Ж.2 – Конфигурационный тест 2

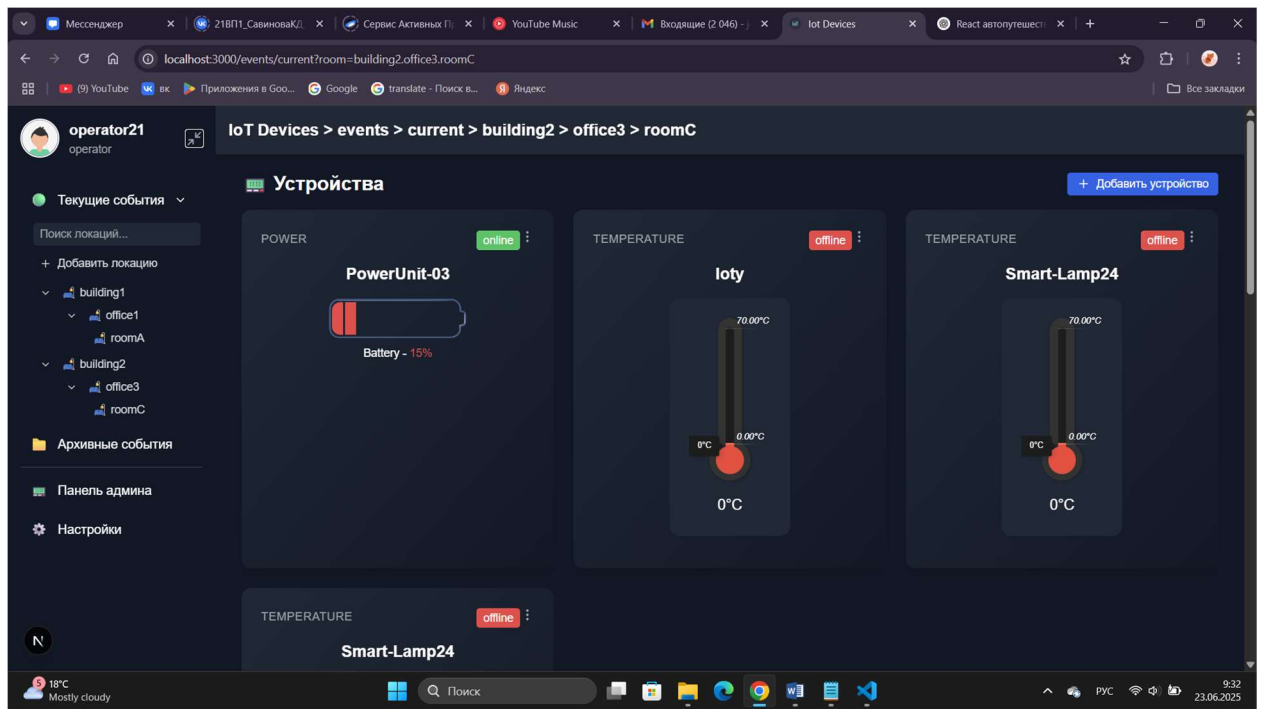


Рисунок Ж.3 – Конфигурационный тест 3

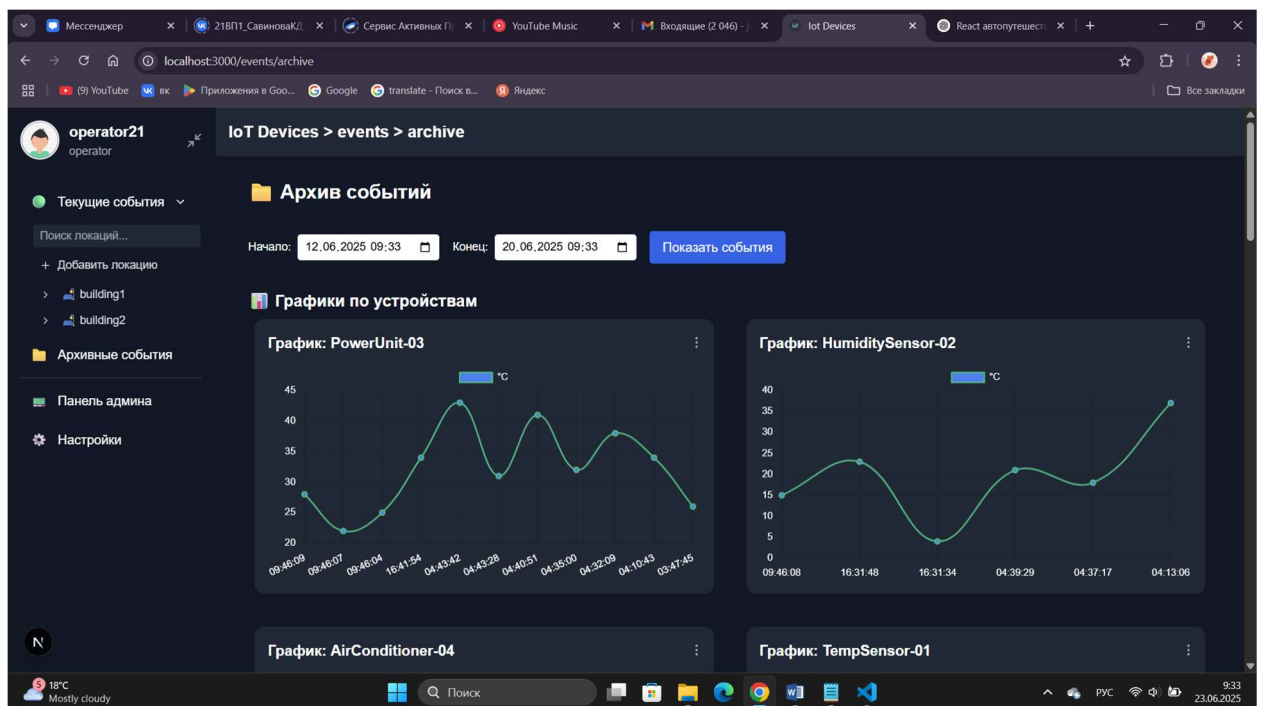


Рисунок Ж.4 – Конфигурационный тест 4

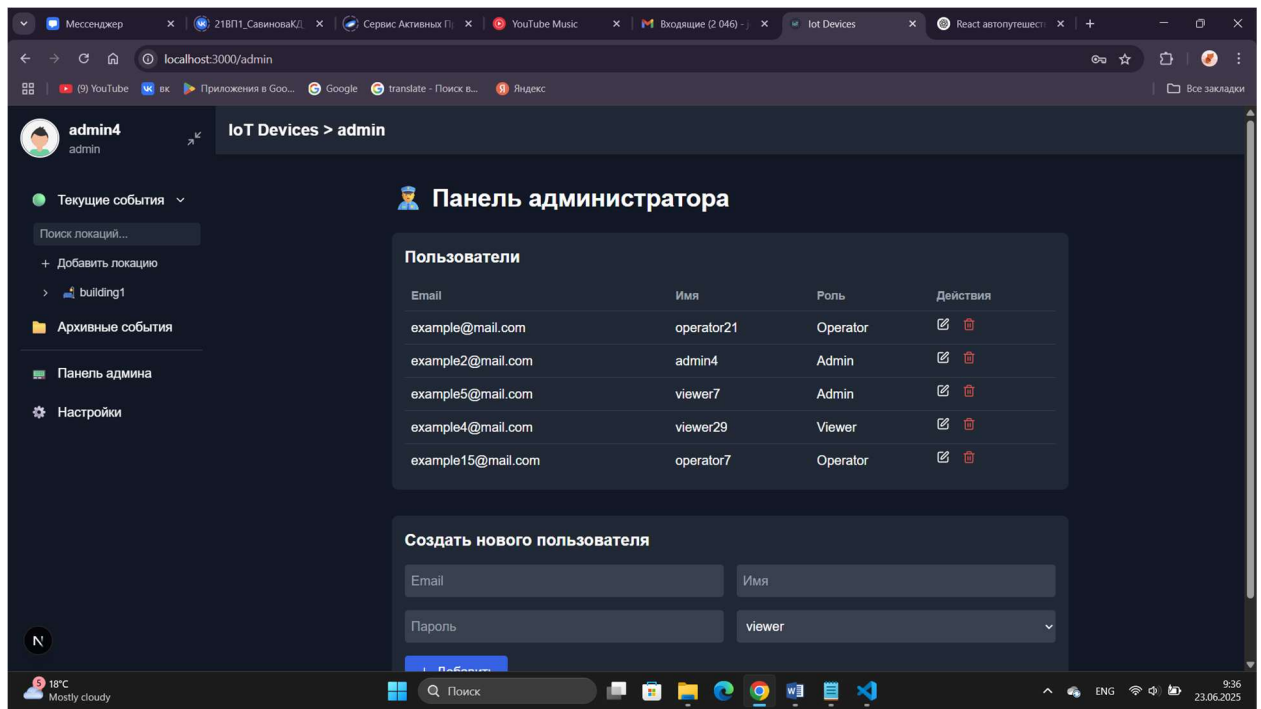


Рисунок Ж.5 – Конфигурационный тест 5

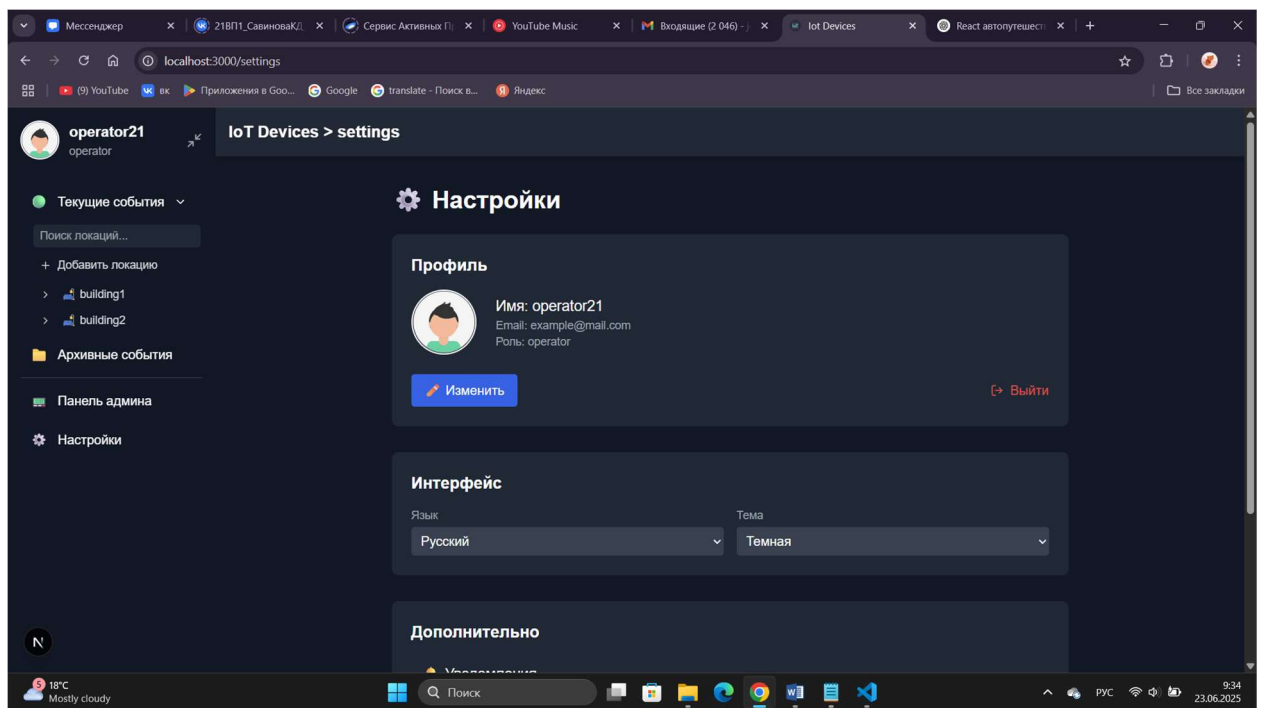


Рисунок Ж.6 – Конфигурационный тест 6

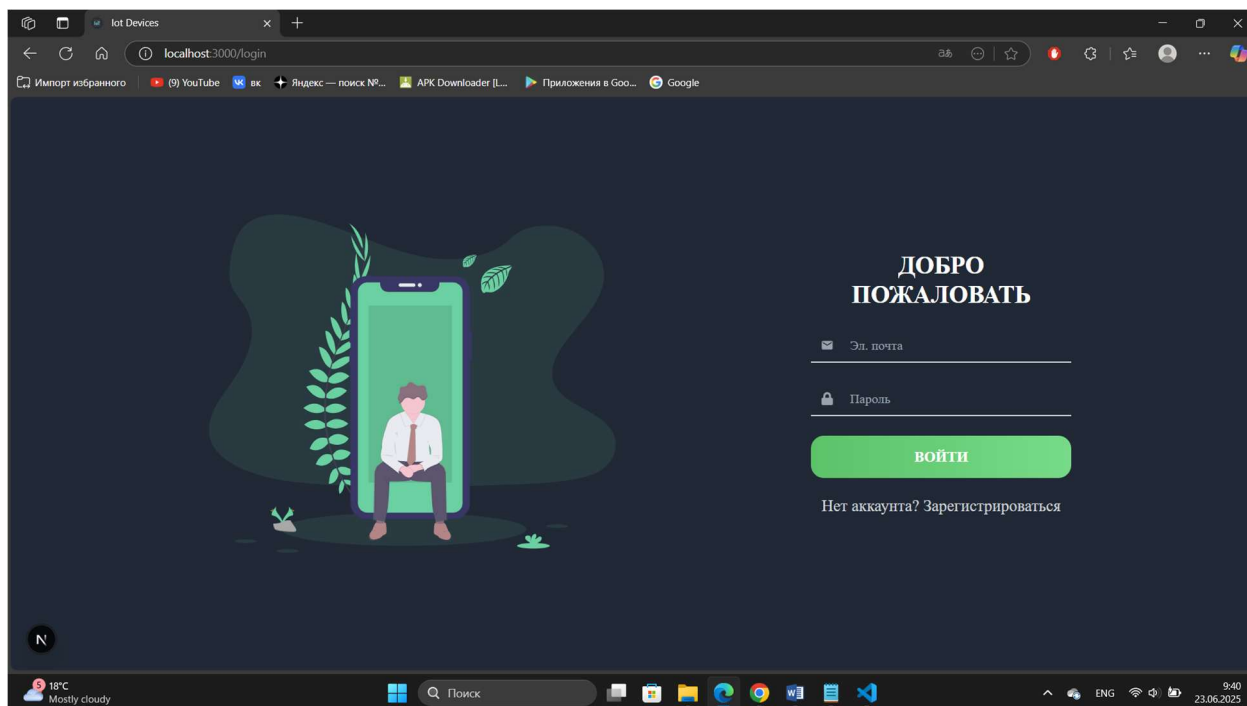


Рисунок Ж.7 – Конфигурационный тест 7

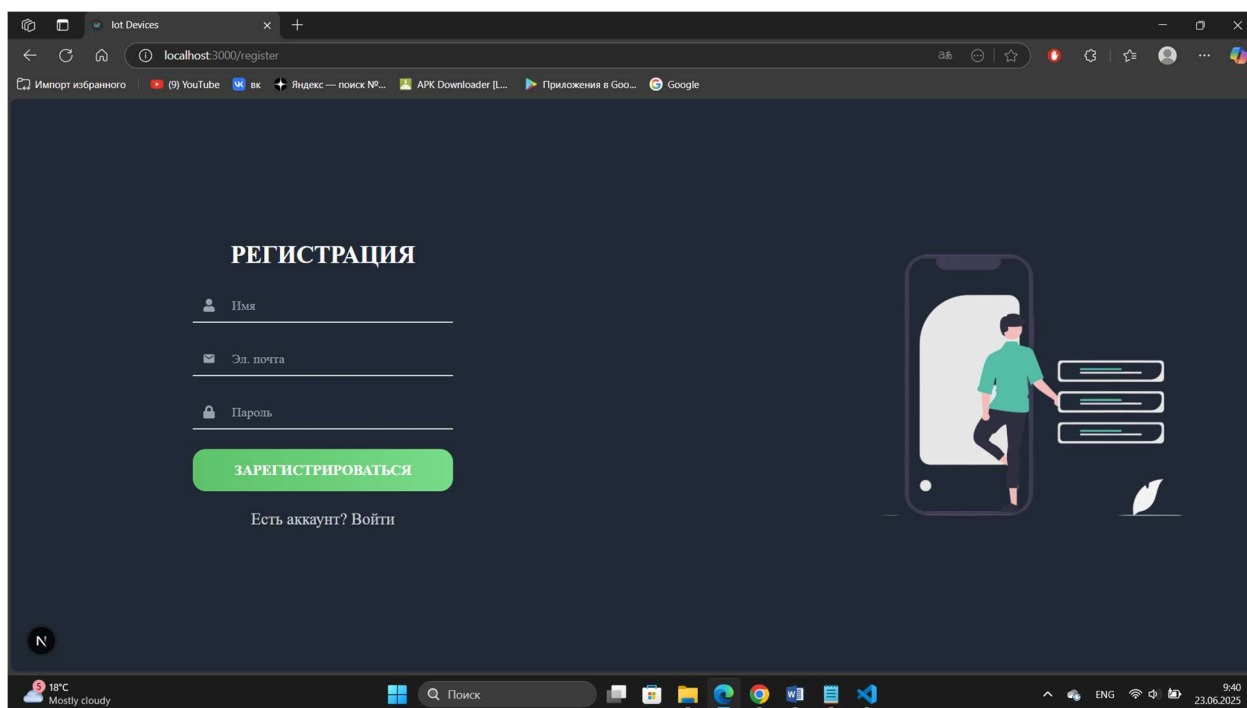


Рисунок Ж.8 – Конфигурационный тест 8

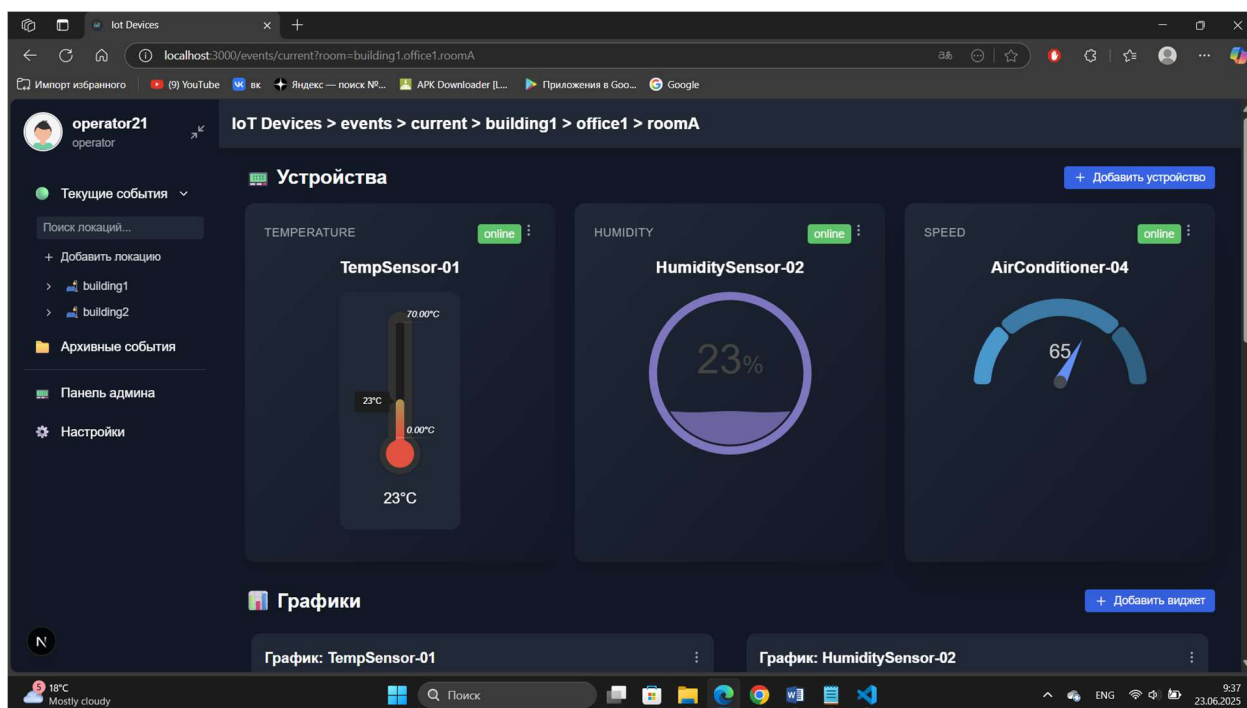


Рисунок Ж.9 – Конфигурационный тест 9

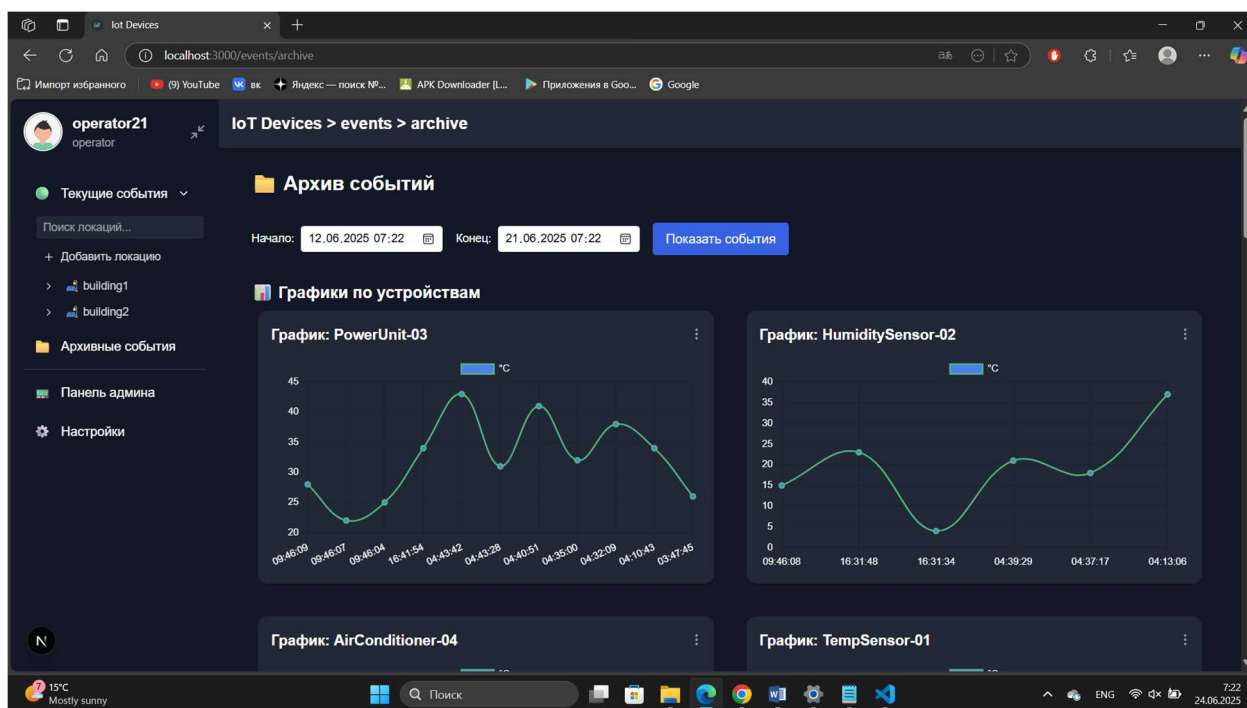


Рисунок Ж.10 – Конфигурационный тест 10

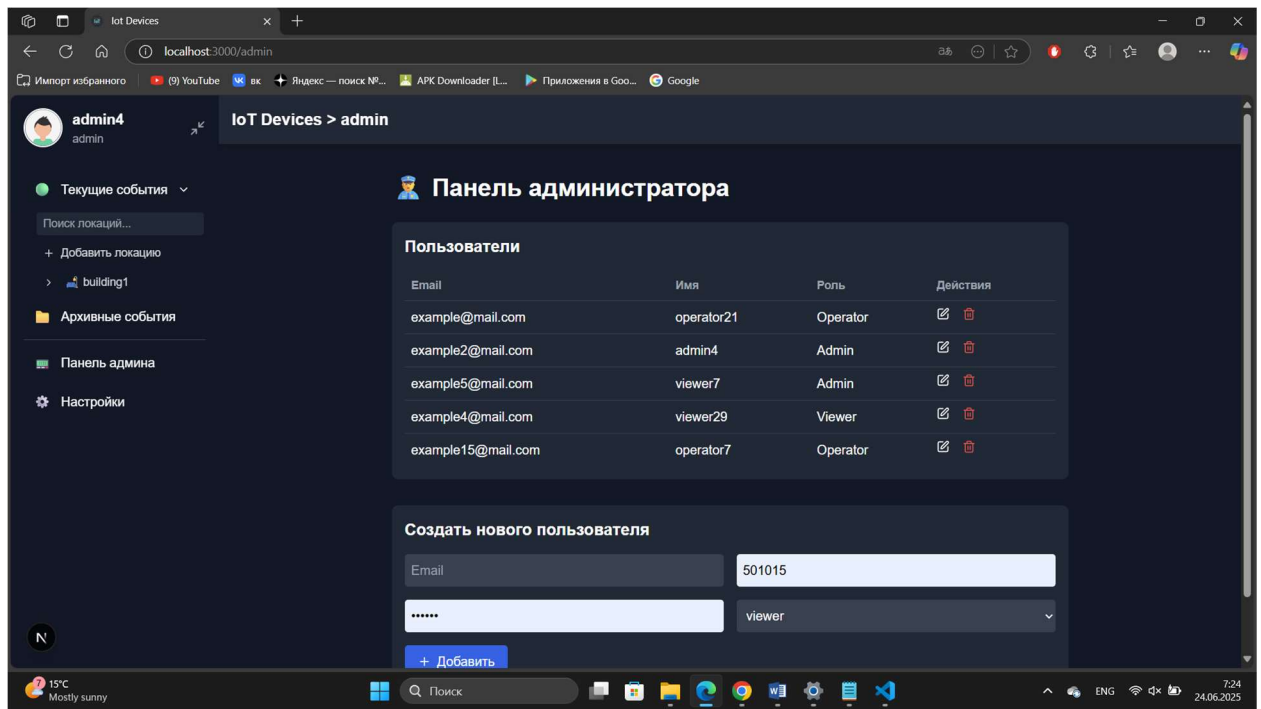


Рисунок Ж.11 – Конфигурационный тест 11

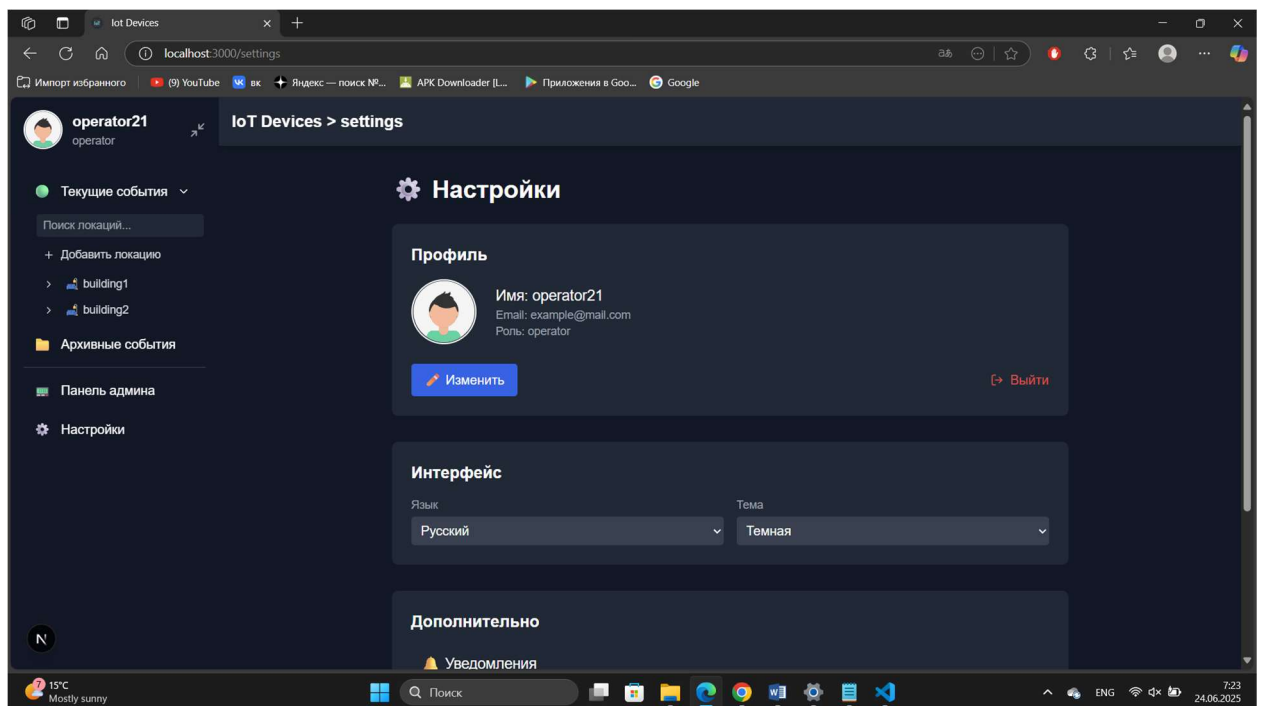


Рисунок Ж.12 – Конфигурационный тест 12