

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

Программные средства взаимодействия с объектами мониторинга и управления в составе информационно-управляющей платформы с применением облачных технологий

Студент _____ Астахов Н.С.
(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 Астахов Никита Сергеевич гр. 21ВИ1 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства взаимодействия с объектами мониторинга и управления в составе информационно-управляющей платформы с применением облачных технологий

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 web-приложение управления информационно-управляющей платформы

2) Среда выполнения: Windows 11

Языковые средства разработки: C#, NODE-red, JS, UML

Инструментальные средства: Node-red, Visual Studio, Visual Paradigm 16.3., Yandex cloud

Технологии разработки: ООП, сервисно-ориентированный подход.

3) Основные функции разрабатываемого приложения:

a) Добавление устройств

b) Редактирование устройства

c) Удаление устройства

d) Просмотр значений от устройств

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.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	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 161 лист, 32 рисунка, 26 таблиц, 24 использованных источника и 5 приложений.

IoT-устройства, Yandex cloud, IoT core, устройства мониторинга и управления, C#, RESTful API, Windows forms, NODE-RED, , UML.

Объект исследования – программные средства взаимодействия с объектами мониторинга и управления.

Цель выпускной квалификационной работы – разработать программные средства взаимодействия с объектами мониторинга и управления.

Технологии разработки – разработка проводилась на языках программирования JavaScript и C# с. В качестве среды разработки использовался редактор программного кода Microsoft Visual Studio.

Среда исполнения – Windows 11, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы программные средства взаимодействия с объектами мониторинга и управления.

Все требования к разработке приложения соблюдены.

					ПГУ 09.03.02 – 08ВР211.05 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Астахов Н.С.			Программные средства взаимодействия с объектами мониторинга и управления Пояснительная записка	Лит.	Лист	Листов
Пров.		Шибанов С.В.					4	161
Н. Контр.		Такташкин Д.В				Группа 21ВИ1		
Утв.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и планирование процесса разработки программных средств взаимодействия с объектами мониторинга и управления	9
1.1. Low-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.....	14
1.1.2.3 Активные правила.....	15
1.1.3 Обобщенная программная архитектура платформы	17
1.2 Взаимодействие с объектами мониторинга и управления в информационно–управляющей платформе	20
1.2.1 Классификация объектов мониторинга и управления	20
1.2.2 Технологии, протоколы и программные средства взаимодействия с IoT-устройствами мониторинга и управления.....	21
1.2.3 Бизнес-процессы управления IoT-устройствами мониторинга и управления	22
1.3 Сравнительный анализ программных платформ для проектирования, развертывания и эксплуатации IoT-систем.....	23
1.3.1 Обоснование сравнительного анализа	23
1.3.2 Обзор программных платформ для анализа	24
1.3.3 Результаты сравнительного анализа и выводы	27
1.4 Планирование процесса разработки программных средств.....	29
1.4.1 Цели разработки и назначение программных средств	29
1.4.2 Классы пользователей программных средств	29
1.4.3 План-график разработки и планирование бюджета	29

2 Анализ требований и постановка задачи на разработку программных средств взаимодействия с объектами мониторинга и управления	36
2.1 Анализ функциональных требований.....	36
2.2 Анализ нефункциональных требований	39
2.3 Анализ требований к пользовательскому интерфейсу панели управления.....	40
2.4 Анализ требований к аппаратно-программному обеспечению	46
2.4.1 Требования к аппаратному обеспечению	46
2.4.2 Требования к системному программному обеспечению	47
2.4.3 Требования к средствам разработки	48
2.5 Постановка задачи на разработку программных средств	50
3 Разработка программных средств взаимодействия с объектами мониторинга и управления	51
3.1 Разработка архитектуры развертывания программных средств	51
3.2 Разработка программных шаблонов для реализации функций мониторинга управления.....	54
3.3 Разработка RESTfulAPI-сервера для взаимодействия с приложениями информационно-управляющей платформы	56
3.4 Разработка клиентского приложения	58
3.4.1 Разработка интерфейса	58
3.4.2 Разработка программных средств	64
4 Контроль качества программных средств взаимодействия с объектами мониторинга и управления.....	67
4.1 Планирование процесса контроля качества приложения.....	67
4.1.1 Мероприятия по контролю качества	67
4.1.2 Обоснование выбора модели тестирования	68
4.1.3 Обоснование выбора метрик программного кода	70
4.2 Тестирование приложения.....	71
4.2.1 Функциональное тестирование логики методом черного ящика.....	71
4.2.2 Функционально тестирование интерфейса методом черного ящика	74

4.3 Оценка качества программного кода.....	76
Заключение	79
Список использованных источников	79
Приложение А (обязательное) Глоссарий	84
Приложение Б (обязательное) Шаблоны устройств.....	86
Приложение В (обязательное) Листинг RESTful API	101
Приложение Г (обязательное) Листинг клиентского приложения	125
Приложение Д (обязательное) Тестирование.....	156

Введение

Современные информационно-управляющие системы (ИУС) играют ключевую роль в реализации задач управления. Использование объектов мониторинга и управления в составе ИУС позволяет автоматизировать процессы, повысить эффективность работы и обеспечить оптимальное распределение ресурсов. Это дает возможности для прогнозирования и своевременного реагирования на изменение внешних или внутренних условий.

Целью данной выпускной квалификационной работы является разработка программных средств, обеспечивающих взаимодействие с объектами мониторинга и управления в рамках информационно-управляющей платформы с применением облачных технологий.

Разрабатываемое программное обеспечение должно позволить пользователям создавать объекты мониторинга и управления, редактировать их и просматривать получаемые значения в реальном времени.

Для достижения поставленной цели в работе были определены следующие задачи:

- 1) провести анализ предметной области и постановку задачи;
- 2) выполнить сравнительный анализ существующих программных средств для взаимодействия с объектами мониторинга и управления;
- 3) выполнить проектирование программных средств;
- 4) разработать ПС в соответствие с поставленной задачей;
- 5) провести тестирование разработанного продукта.

В результате выполнения работы были реализованы программные средства, которые позволяют управлять устройствами мониторинга и управления, а также просматривать получаемые от них значения.

В настоящий момент разработан прототип программных средств который в дальнейшем будет расширять свои возможности.

1 Анализ предметной области и планирование процесса разработки программных средств взаимодействия с объектами мониторинга и управления

1.1 Low-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 командам 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 и.

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 [3] (рисунок 1.1).

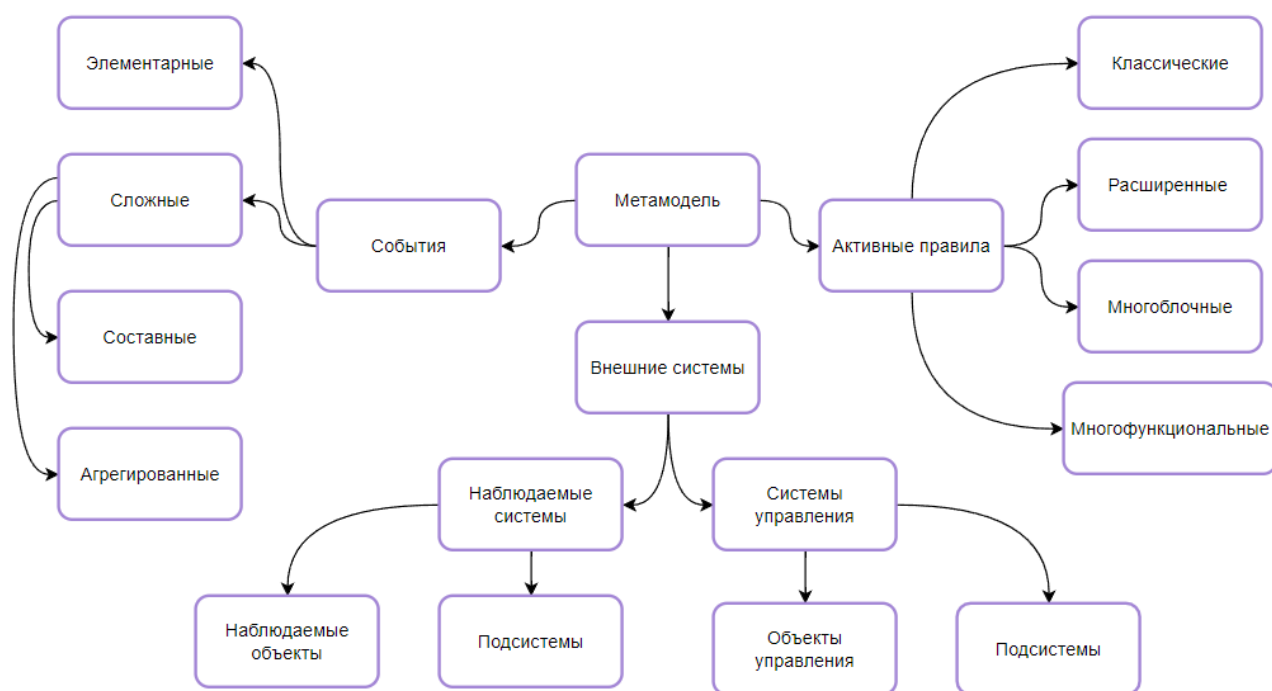


Рисунок 1.1 – Обобщенная концептуальная модель информационно-управляющей платформы

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 таблице 1.1 представлен перечень основных классов объектов концептуальной модели и их краткое описание.

Таблица 1.1 - Перечень основных классов объектов концептуальной модели и их краткое описание

Класс объектов	Описание
Системы мониторинга и управления	Это системы, предназначенные для наблюдения (мониторинга) за состоянием подсистем или объектов, а также для выработки решений и выполнения управляющих воздействий. Они действуют как связующее звено между объектами наблюдения и объектами управления, анализируя данные и реагируя на изменения среды.
Объекты мониторинга и объекты управления	Объекты мониторинга — элементы системы, чьи параметры, состояния или поведение отслеживаются в реальном времени. Они предоставляют данные для анализа, например, датчики или системы с телеметрией. Объекты управления — элементы системы, на которые оказывают непосредственное влияние для изменения их состояния или поведения, например, исполнительные механизмы, устройства или программы.
События, поступающие от объектов мониторинга	Это данные или сигналы, которые указывают на изменения в состоянии объектов мониторинга. События включают информацию о возникновении аномалий, достижении пороговых значений, обновлении параметров, а также других значимых изменениях, требующих реакции.
Активные правила	Это механизм или набор инструкций, определяющий, как система должна интерпретировать поступающие события и вырабатывать решения для управления. Активные правила обеспечивают автоматизацию процессов, включая анализ событий, выбор сценария реакции и выполнение управляющих действий.

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. События делятся на элементарные (простые) и комплексные (сложные), которые могут быть либо агрегированными, либо составными. (рисунок 1.2).



Рисунок 1.2 - 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, состоящую 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 1.3). Расширенное активное правило может быть дополнительно классифицировано на альтернативное, многоблочное и многофункциональное правила.

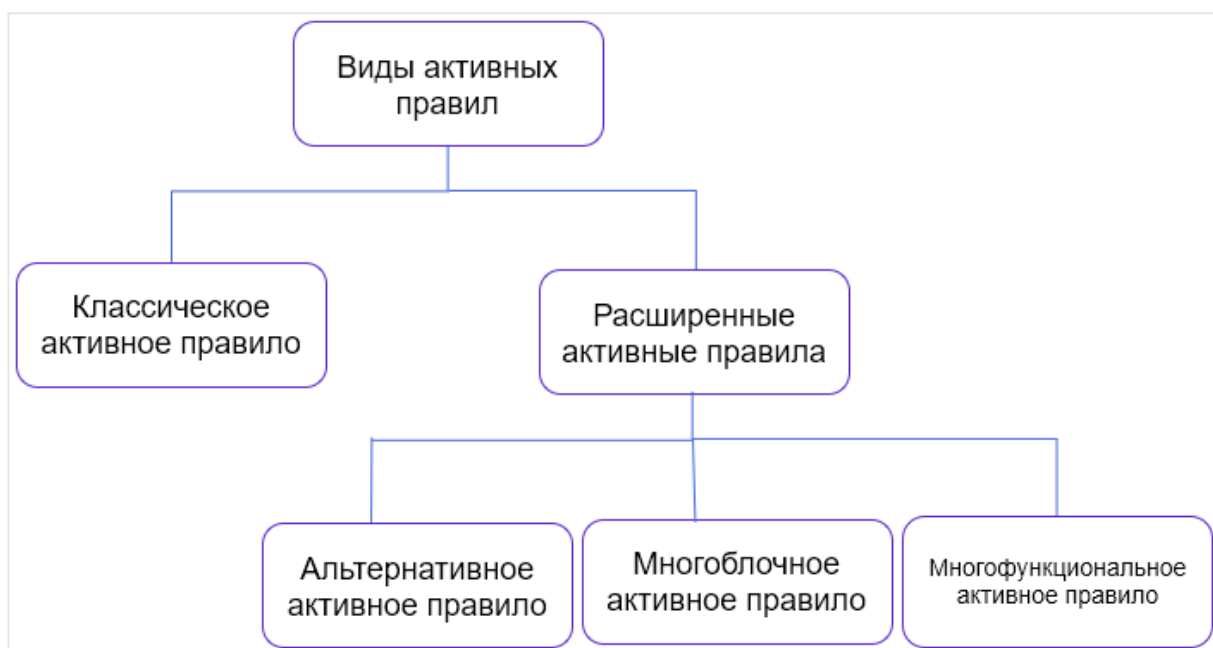


Рисунок 1.3 - Классификация активных правил

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 1.4).

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.

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.

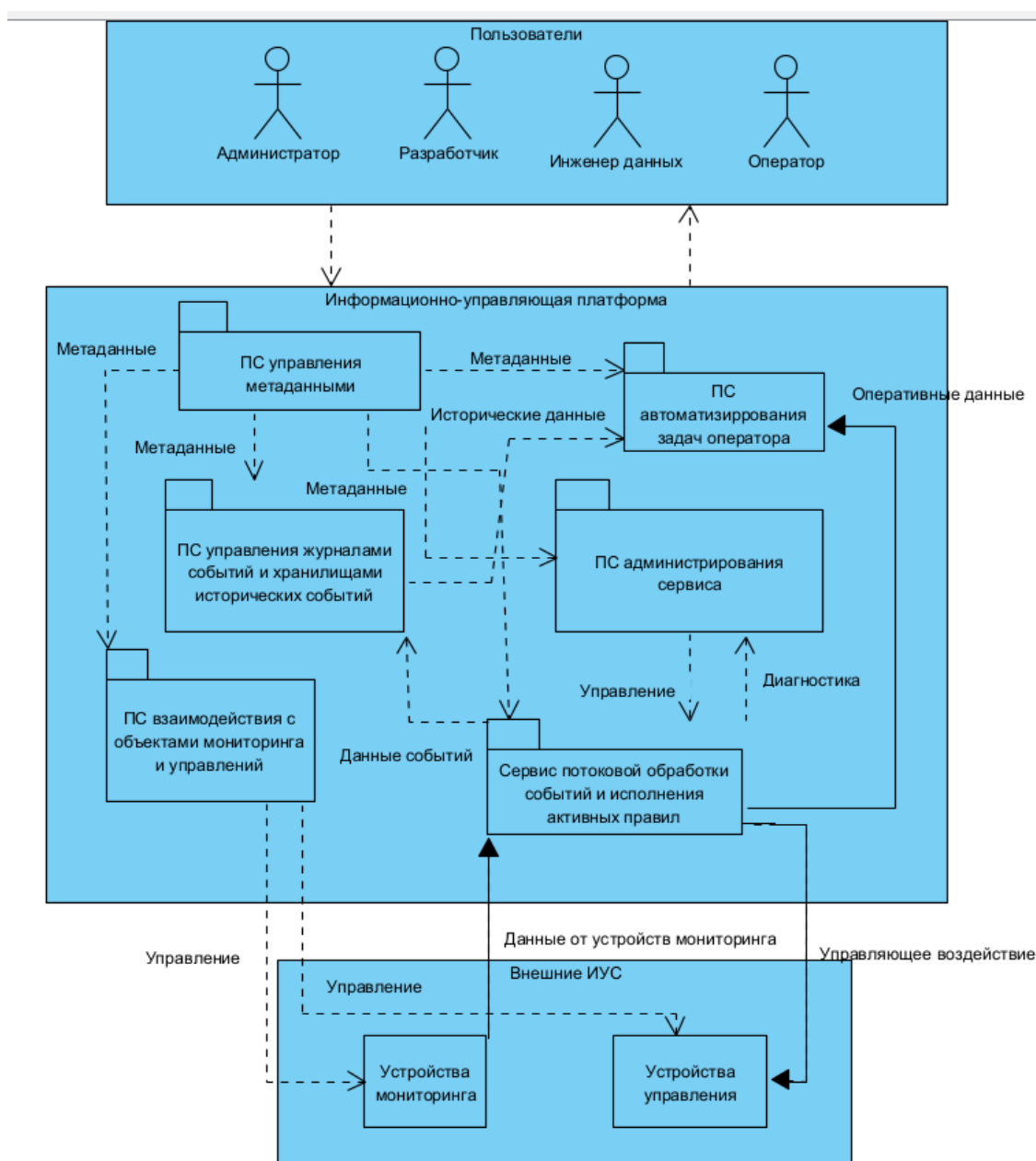


Рисунок 1.4 – Обобщенная программная архитектура информационно-управляющей платформы

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 Взаимодействие с объектами мониторинга и управления в информационно–управляющей платформе

1.2.1 Классификация объектов мониторинга и управления

Классификация объектов мониторинга и управления базируется на нескольких важных критериях. Первым из них является функциональное назначение устройства, которое может быть сосредоточено на сборе данных, выполнение команд или предоставление интерфейса для управления. Вторым критерием являются технические характеристики, а именно параметры, такие как пропускная способность, энергоэффективность и устойчивость к внешним воздействиям. Дополнительно учитываются способы интеграции, которые определяют, насколько быстро и просто устройство можно подключить к информационно-управляющей платформе.

Объекты мониторинга и управления делятся на три основных класса:

- сенсорные устройства;
- исполнительные механизмы;
- интерфейсные модули.

Сенсорные устройства служат для сбора данных окружающей среды, предоставляя платформе показатели для анализа и оптимизации процессов.

Исполнительные механизмы выполняют команды, поступающие от платформы, активируя определенные действия или изменяя параметры работы других объектов.

Интерфейсные модули обеспечивают удобные средства взаимодействия пользователей с системой, предоставляя интуитивно понятные возможности для управления и настройки оборудования.

В настоящее время происходит активный переход к IoT-устройствам, что значительно меняет традиционные методы мониторинга и управления.

IoT-устройство – это концепция, согласно которой различные устройства и объекты могут подключаться к интернету и обмениваться данными между собой. IoT-устройства включают в себя сенсоры, исполнительные механизмы и

интерфейсы модули, которые работают в связке для достижения определенных задач[5].

Переход к IoT-устройствам значительно изменяет методы мониторинга и управления благодаря инновационным технологиям. Такой переход содействует более глубокому взаимодействию с платформой, повышению уровня автоматизации процессов, а также улучшению точности и надежности данных. Основным преимуществом IoT-устройств является возможность удаленного доступа, который предоставляет компании большую гибкость и контроль, а также возможность постоянного сбора данных в режиме реального времени, что существенно увеличивает производительность [6].

1.2.2 Технологии, протоколы и программные средства взаимодействия с IoT-устройствами мониторинга и управления

Эффективная работа с IoT-устройствами требует использования разнообразных технологий и протоколов. Одним из популярных протоколов для передачи данных является MQTT, который обеспечивает легкость обмена сообщениями в ресурсно-ограниченных средах. Протокол CoAP оптимизирован для использования в сетях с ограниченными ресурсами, предлагая надежную и энергоэффективную передачу данных. Стандартные протоколы HTTP и HTTPS также остаются востребованными благодаря удобству интеграции и высокому уровню безопасности. Программные средства, такие как библиотеки и API, предлагают упрощенные методы интеграции и управления устройствами, поддерживая непрерывную связь между устройствами и платформой.

Выбор облачной платформы Yandex IoT Core для взаимодействия с устройствами обоснован рядом преимуществ, такими как безопасность, масштабируемость и надежность. Платформа предлагает инструменты для простого подключения IoT-устройств и их управления через облако, делая сложные локальные конфигурации ненужными. Yandex IoT Core предоставляет мощные средства для анализа данных и визуализации, облегчая мониторинг и содействуя принятию обоснованных решений. Масштабируемость решения

позволяет компании быстро адаптироваться к изменяющимся нуждам и увеличению количества подключаемых устройств без потери производительности. Таким образом, выбор облачного подключения через Yandex IoT Core обусловлен стремлением к улучшению гибкости управления, оперативности и надежности, что способствует значительному повышению эффективности бизнес-процессов компании.

1.2.3 Бизнес-процессы управления IoT-устройствами мониторинга и управления

Управление IoT-устройствами является важнейшим аспектом современного общества, позволяющим эффективно собирать, обрабатывать и использовать данные для оптимизации процессов. IoT-устройства применяются в различных сферах, включая промышленность, здравоохранение и умные дома, с помощью IoT устройств компании могут отслеживать состояние оборудования и прогнозировать его поломки и т. д. [7]

На рисунке 1.5 были отображены бизнес-процессы управления IoT-устройствами в виде диаграммы деятельности [8].

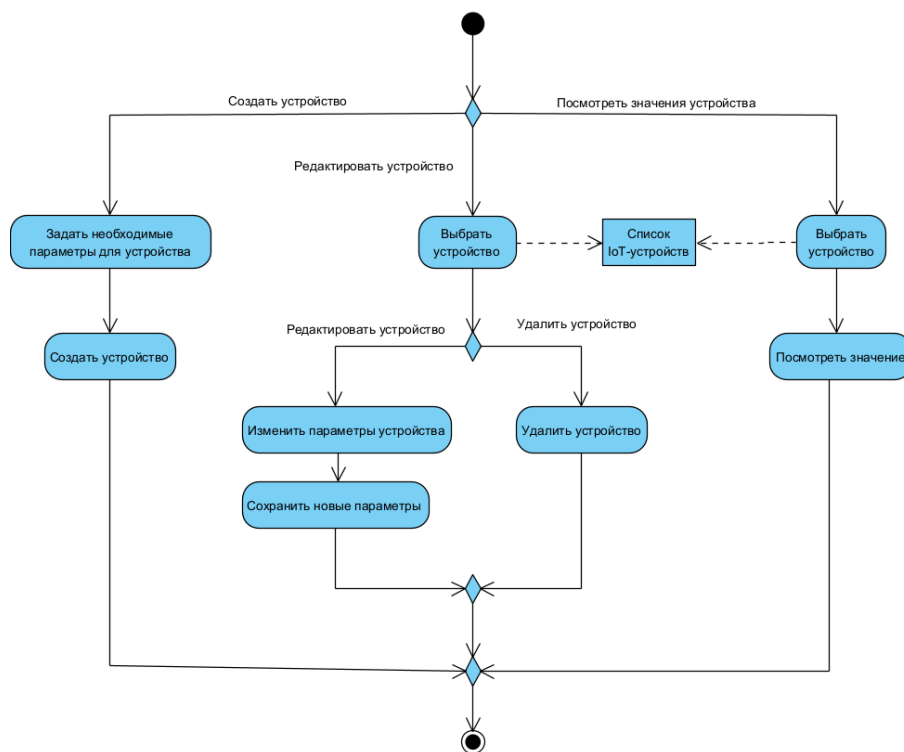


Рисунок 1.5 – Диаграмма деятельности для управления IoT-устройствами

На диаграмме изображены следующие бизнес процессы:

- создать устройство – пользователь задает необходимые параметры и добавляет новое устройство в систему;
- редактировать устройство – пользователь выбирает необходимое устройство и меняет его параметры;
- удалить устройство – пользователь выбирает необходимые параметры и удаляет устройство;
- посмотреть значения устройства – пользователь выбирает устройство и просматривает текущее значение.

Главной ролью для системы является администратор системы, который отвечает за управления IoT-устройствами, обеспечивая выполнения всех процессов.

1.3 Сравнительный анализ программных платформ для проектирования, развертывания и эксплуатации IoT-систем

1.3.1 Обоснование сравнительного анализа

Сравнительный анализ современных средств взаимодействия с объектами мониторинга и управления позволит определить наиболее используемые системы а также получить представление о том, насколько они распространены и актуальны.

Можно выделить следующие цели проведения анализа систем:

- выявление достоинств;
- выявления недостатков;
- выявление характерных базовых функций.

По итогам анализа главной задачей будет определение обязательных функций, а также уникальных возможностей, которые расширят функционал реализуемой системы и приведут к ее высокой конкурентоспособности.

На сегодняшний день существует множество инструментальных средств, которые позволяют выполнять взаимодействия с объектами мониторинга и управления. Далее приведена некоторая часть их:

- Aggregate;
- Rightech;
- ThingSpeak;
- AWS IoT Core;
- Azure IoT Hub;
- Kaa IoT Platform.

Эти платформы ориентированы на сбор и анализ полученных данных, а также на управление IoT-устройствами. Для проведения сравнительного анализа были выбраны следующие системы:

- Aggregate;
- rightech;
- AWS IoT Core.

Подробная информация о сервисах представлена в таблице 1.2.

Таблица 1.2 – Информация о аналоговых системах

Система	Доступность	Наличие онлайн-версии	Фирма производитель
aggregate	Платная	+	Aggregate Inc.
rightech	Платная	+	Rightech
AWS IoT Core	Платная, есть бесплатная версия	+	Amazon Web Services

Таким образом, для дальнейшего анализа были выбраны наиболее используемые инструменты взаимодействия с объектами мониторинга и управления.

1.3.2. Обзор программных платформ для анализа

Aggregate – это платная система, предоставляющая решения для анализа и обработки данных в реальном времени. Она идеально подходит для организаций, которым необходимы надежные инструменты для работы с большими объемами данных. Aggregate предлагает онлайн-версию, что

AWS IoT Core – это услуга от Amazon Web Services, которая предоставляет платный доступ с возможностью использования бесплатного уровня. Платформа предназначена для подключения IoT-устройств и их управления с помощью облачных технологий. Наличие онлайн-версии позволяет пользователям легко масштабировать свои приложения и управлять данными в реальном времени. AWS гарантирует высокую доступность и надежность своих сервисов, что делает эту систему одной из лучших на рынке. Интерфейс AWS IoT Core представлен на рисунке 1.8.

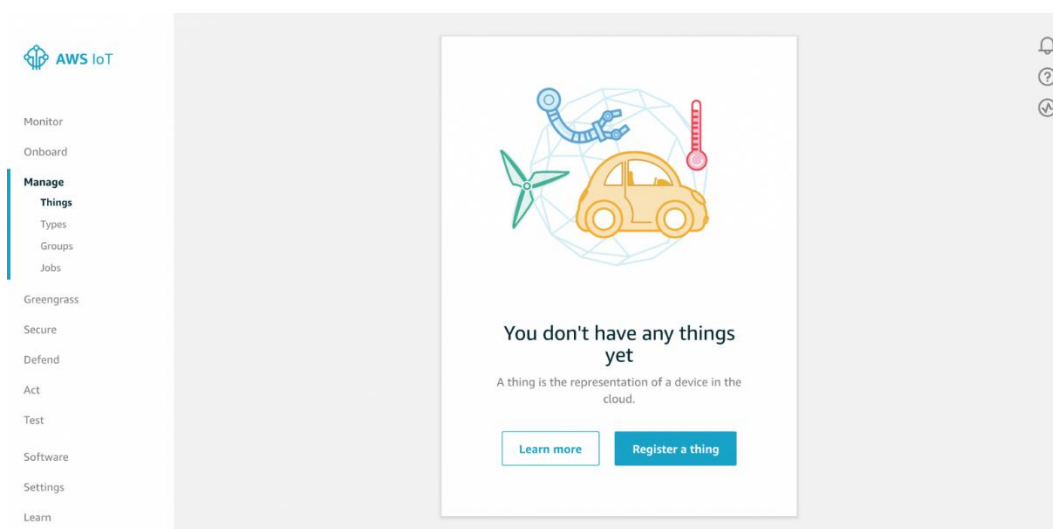


Рисунок 1.8 – Интерфейс платформы AWS IoT Core

Для оценки, анализа и сравнения инструментальных средств были выделены следующие критерии:

- добавление устройства – наличие функции создать устройство с заданными параметрами;
- редактирование устройства – возможность выбрать устройство и отредактировать его параметры;
- удаление устройства – возможность удалить конкретное устройство из системы;
- просмотреть значения в реальном времени – наличие окна с отображением текущих значений устройств;
- обработка больших объемов данных – обеспечение быстрой и эффективной работы с большими данными;

– поддержка клиентов – наличие системы поддержки для решения вопросов и проблем пользователей;

– масштабируемость приложений при помощи облачных технологий – возможность увеличения ресурсов при помощи облачных технологий;

– интеграция с существующими системами – способность взаимодействовать с существующими решениями.

Результаты анализа выделенных инструментальных средств автоматической генерации данных представлены в таблице 1.3.

Таблица 1.3 –Результирующая таблица анализа систем по критериям

Функции	Система		
	Aggregate	Rightech	AWS IoT Core
Добавление устройства	+	+	+
Редактирование устройства	+	+	+
Удаление значения	+	+	+
Просмотреть значение в реальном времени	+	+	+
Обработка больших объёмов данных	+	-	+
Поддержка клиентов	+	-	-
Масштабируемость приложений при помощи облачных технологий	-	-	+
Интеграция с существующими системами	-	+	+

На основе результирующей таблицы наглядно определены сильные и слабые стороны систем взаимодействия с объектами управления и мониторинга.

1.3.3 Результаты сравнительного анализа и выводы

В результате анализа было выявлено, что функция добавления устройства присутствует во всех системах: Aggregate, Rightech и AWS IoT Core. Функции редактирования и удаления устройств, а также возможность просмотра значения в реальном времени реализованы в Aggregate и Rightech, но отсутствуют в AWS IoT Core.

Обработка больших объёмов данных доступна в Aggregate и AWS IoT Core, в то время как Rightech не поддерживает эту функцию. Поддержка клиентов имеется в Aggregate, в то время как Rightech и AWS IoT Core ее не предоставляют.

Масштабируемость приложений с использованием облачных технологий доступна только в AWS IoT Core, тогда как в Aggregate и Rightech эта функция отсутствует.

Интеграция с существующими системами реализована в Rightech и AWS IoT Core, но не поддерживается в Aggregate.

Таким образом, среди трех рассмотренных платформ наибольшее количество функциональных возможностей обладает Aggregate, но AWS IoT Core предлагает лучшие возможности для масштабируемости и обработки больших объемов данных.

Rightech, несмотря на отсутствие некоторых функций, может предоставить удобную интеграцию с существующими системами.

Таким образом, конечный выбор системы будет зависеть от специфических потребностей и предпочтений пользователя.

В рассмотренных средствах взаимодействия с объектами мониторинга и управления присутствуют стандартные функции, которые необходимо поддерживать и в реализуемой системе. К ним относятся:

- добавление устройства;
- редактирование устройств;
- удаление устройств;
- просмотр значений в реальном времени.

Кроме обязательных функций, в разрабатываемую систему необходимо включить ряд дополнительных возможностей, обеспечивающих ее уникальность и конкурентоспособность. В качестве таких особенностей были определены следующие функции:

- масштабируемость систем при помощи облачных технологий;
- интеграция с существующими системами;
- обработка больших объемов данных.

Таким образом разрабатываемая система будет конкурентоспособной благодаря стандартным и уникальным функциям.

1.4 Планирование процесса разработки программных средств

1.4.1 Цели разработки и назначение программных средств

Разрабатываемые программные средства взаимодействия с объектами мониторинга и управления должны обеспечивать пользователя набором удобных инструментов для управления устройствами и просмотра полученных от них значений в реальном времени.

Основные цели разработки:

- реализация моделирования и управления IoT-устройствами;
- разработка хранения и передачи данных от устройств;
- интеграция компонентов для генерации данных;
- интеграция средств для отображения полученных средств.

Кроме того, после окончания основного процесса разработки и публикации сервиса взаимодействия с объектами мониторинга и управления, требуется поддерживать его функционал, проверять на неисправности, проводить анализ работы сервиса с помощью сбора статистики, а также внедрять новый функционал при необходимости, для упрощения работы сервиса.

1.4.2 Классы пользователей программных средств

Существует один класс пользователя сервиса – администратор системы.

Задачи администратора состоят в создании и управлении IoT-устройств, а также просмотра получаемых от них значений.

Создание устройств – это добавление новых устройств в систему с указанными пользователем параметрами.

Управление устройствами включает в себя удаление или редактирование необходимого устройства.

Данные программные средства реализуют функционал, который нацелен на администратора системы.

1.4.3 План-график разработки и планирование бюджета

Планирование процесса разработки включает формирование плана выполнения работ, который отражает порядок действий при реализации проекта с учетом необходимых ресурсов. Основной задачей планирования является определение целей и методов их достижения.

Для демонстрации плана разработки подходит диаграмма Ганта, отображающая длительность и сроки выполнения этапов разработки на шкале времени, а также требуемые для их реализации ресурсы. В качестве инструмента управления проектом была выбрана программа Microsoft Project. Дальнейший расчет будет производиться для ПС взаимодействия с объектами мониторинга и управления.

В таблице 1.4 приведены результаты определения задач проекта, их тип, длительность их выполнения и задачи, предшествующие им.

Таблица 1.4 – Задачи проекта

№	Задача	Тип	Длительность	Предшественники
1	2	3	4	5
1	Начало проекта	Веха	1	
2	Анализ предметной области и постановка задачи на разработку	Фаза	10	
3	Анализ назначения и возможностей ПС	Задача	1	1
4	Анализ архитектуры ПС	Задача	1	3
5	Анализ классификации объектов мониторинга и управления	Задача	1	4
6	Анализ технологий взаимодействия с объектами мониторинга и управления	Задача	1	5
7	Анализ бизнес-процессов управления объектами	Задача	2	6
8	Сравнительный анализ аналогов	Задача	2	7
9	Постановка задач на разработку ПС	Задача	2	8
10	Анализ предметной области и постановка задачи на разработку завершены	Веха	0	9
11	Разработка требований к ПС взаимодействия с объектами мониторинга и управления	Фаза	6	
12	Анализ функциональных требований	Задача	1	9
13	Анализ нефункциональных требований	Задача	1	12
14	Анализ требований к пользовательскому интерфейсу панели управления	Задача	1	13
15	Анализ требования к аппаратному обеспечению	Задача	1	14
16	Анализ требований к системному программному обеспечению	Задача	1	15
17	Анализ требований к средствам разработки	Задача	1	16

Продолжение таблицы 1.4

1	2	3	4	5
18	Разработка требований к ПС взаимодействия с объектами мониторинга и управления завершены	Веха	0	17
19	Разработка ПС взаимодействия с объектами мониторинга и управления	Фаза	23	
20	Разработка архитектуры развертывания программных средств	Задача	4	18
21	Разработка программных шаблонов для реализации функций мониторинга управления	Задача	4	20
22	Разработка ПС создания объектов мониторинга и управления	Задача	5	21
23	Разработка ПС контроля объектов мониторинга и управления	Задача	5	22
24	Разработка ПС для взаимодействия с приложениями информационно-управляющей платформы	Задача	5	23
25	Разработка ПС взаимодействия с объектами мониторинга и управления завершены	Веха	0	24
26	Контроль качества ПС взаимодействия с объектами мониторинга и управления	Фаза	10	
27	Выбор методов тестирования	Задача	2	25
28	Тестирование ПС	Задача	5	27
29	Оценка качества программного кода	Задача	3	28
30	Контроль качества ПС взаимодействия с объектами мониторинга и управления завершен	Веха	0	29
31	Завершение работы над проектом	Фаза	5	
32	Оформление документации	Задача	4	30
33	Сдача проекта	Задача	1	32
34	Проект завершен	Веха	0	33

После составления плана выполнения работ необходимо оценить ресурсы проекта и распределить их по задачам. Информация об имеющихся в проекте ресурсах и их распределение представлены в таблицах 1.5 и 1.6 соответственно.

Таблица 1.5 – Ресурсы проекта

Название ресурса	Тип ресурса	Затраты		Начисление
		Стандартная ставка	Ставка сверхурочных	
Программист	Трудовой	600 р/ч	1000 р/ч	Пропорциональное
Аналитик	Трудовой	500 р/ч	800 р/ч	Пропорциональное
Тестировщик	Трудовой	500 р/ч	800 р/ч	Пропорциональное
Компьютер	Материальный			Пропорциональное
Интернет	Затраты			Пропорциональное
Печать	Затраты			Пропорциональное

Таблица 1.6 – Распределение ресурсов по задачам

Название задачи	Ресурсы	Единицы (затраты)
1	2	3
Начало проекта	Компьютер	1
Анализ предметной области и постановка задачи на разработку		
Анализ назначения и возможностей ПС	Аналитик	100%
Анализ архитектуры ПС	Аналитик	100%
Анализ классификации объектов мониторинга и управления	Аналитик, Интернет	100%
Анализ технологий взаимодействия с объектами мониторинга и управления	Аналитик, Интернет	100%
Анализ бизнес-процессов управления объектами	Аналитик	100%
Сравнительный анализ аналогов	Аналитик, Интернет	100%
Постановка задач на разработку ПС	Аналитик, программист	100%
Анализ предметной области и постановка задачи на разработку завершены		
Разработка требований к ПС взаимодействия с объектами мониторинга и управления		
Анализ функциональных требований	Аналитик	100%
Анализ нефункциональных требований	Аналитик	100%
Анализ требований к пользовательскому интерфейсу панели управления	Аналитик	100%
Анализ требования к аппаратному обеспечению	Аналитик, программист	100%
Анализ требований к системному программному обеспечению	Аналитик, программист	100%
Анализ требований к средствам разработки	Аналитик, программист	100%
Разработка требований к ПС взаимодействия с объектами мониторинга и управления завершены		
Разработка ПС взаимодействия с объектами мониторинга и управления		
Разработка архитектуры развертывания программных средств	Программист	100%
Разработка программных шаблонов для реализации функций мониторинга управления	Программист	100%
Разработка ПС создания объектов мониторинга и управления	Программист	100%
Разработка ПС контроля объектов мониторинга и управления	Программист	100%
Разработка ПС для взаимодействия с приложениями информационно-управляющей платформы	Программист	100%
Разработка ПС взаимодействия с объектами мониторинга и управления завершены		
Контроль качества ПС взаимодействия с объектами мониторинга и управления		
Выбор методов тестирования	Тестировщик	100%

по проекту. На рисунках 1.10 и 1.11 представлены обзор о затратах ресурсов и обзор по проекту соответственно.

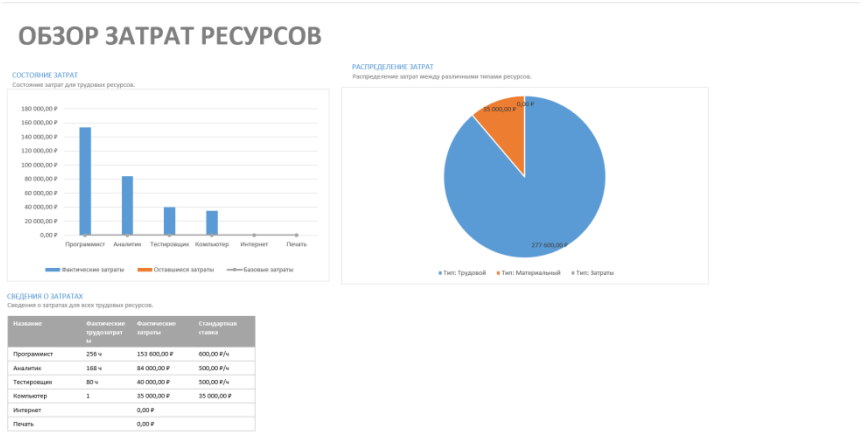


Рисунок 1.10 – Обзор затрат ресурсов проекта

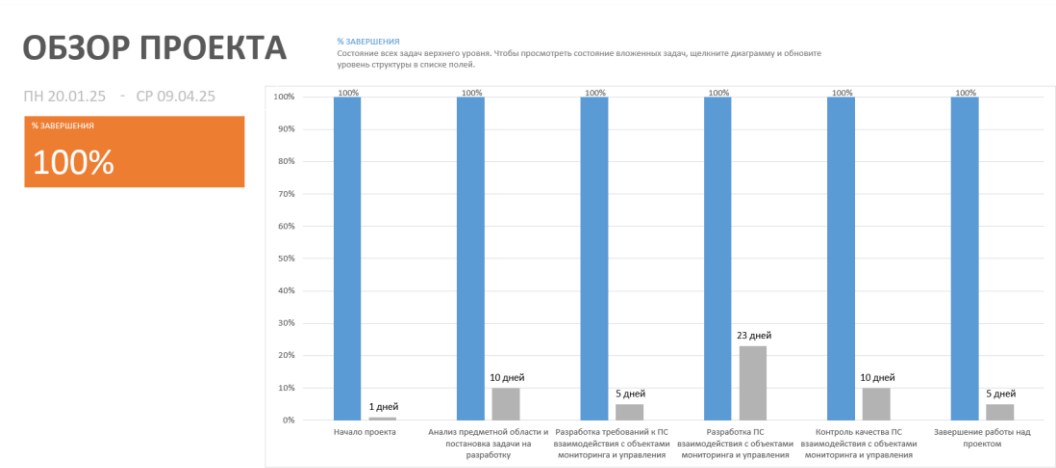


Рисунок 1.11 – Обзор проекта

Построенный обзор о затратах ресурсов показал, что общая стоимость разработки ПС взаимодействия с объектами мониторинга и управления составила 312600 рублей. Исходя из обзора по проекту, следует, что в ходе реализации проекта было выполнено 34 задачи. Проект был выполнен на 100% за 54 дня.

2 Анализ требований и постановка задачи на разработку программных средств взаимодействия с объектами мониторинга и управления

2.1 Анализ функциональных требований

Основной ролью ПС взаимодействия с объектами мониторинга и управления является администратор системы, который создает IoT-устройства, редактирует их параметры и просматривает получаемых от них данные. В рамках создания устройства администратор может задать его название, описание, метки и параметры, после чего сохранить устройство. Также администратор может редактировать или удалить необходимое устройство. Для анализа данных пользователь может просмотреть получаемые значения от устройств, которые поделены на группы.

Результаты анализа функциональных требований представлены в виде диаграммы вариантов использования на рисунке 2.1 [9].

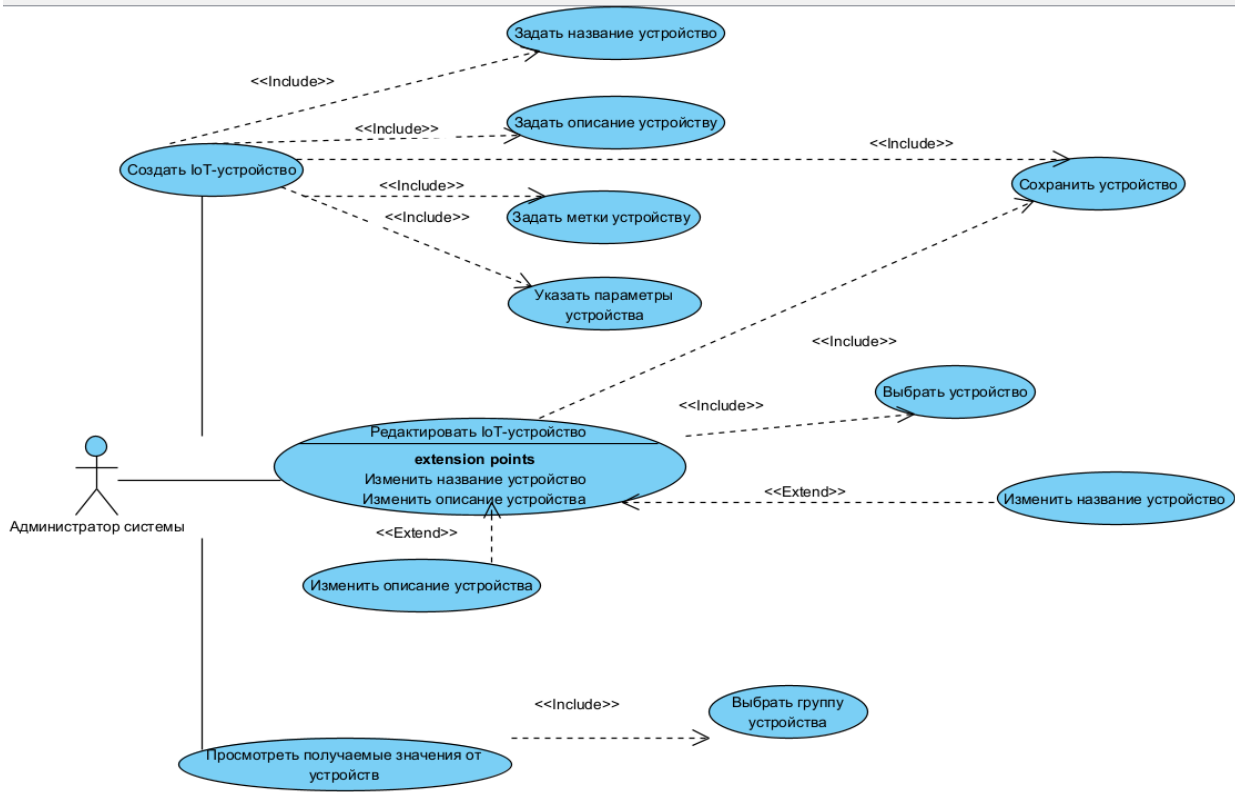


Рисунок 2.1 – Диаграмма вариантов использования для ПС взаимодействия с объектами мониторинга

Диаграмма вариантов использования позволяет легко визуализировать взаимодействие между пользователем и системой, а также выявить ключевые функции, которыми они пользуются.

В таблице 2.1 были описаны все спецификации бизнес-процессов администратора системы.

Таблица 2.1 – Спецификации бизнес-процессов для администратора системы

Актёр	Бизнес-процесс	Описание
Администратор системы	Создать IoT-устройство	Пользователь задает необходимые параметры для устройств, такие как имя, описание, тип устройства, минимальные максимальные границы, расположение и сохраняет его на сервере
Администратор системы	Редактировать IoT-устройство	Пользователь выбирает необходимое устройство и изменяет нужные параметры
Администратор системы	Просмотреть получаемые значения от устройств	Пользователь выбирает группу устройств и просматривает получаемые значения

В таблице 2.2, 2.3 и 2.4 приведены сценарии вариантов использования для каждого бизнес-процесса администратора системы.

Таблица 2.2 – Сценарий варианта использования «Создать IoT- устройство»

Прецедент:	Создать IoT-устройство
Краткое описание:	Пользователь создает устройство мониторинга и управления
Главные актеры:	Администратор системы
Второстепенные актеры:	Нет
Предусловие:	Пользователь открыл приложение
Основной поток:	1) пользователь переходит на страницу создания устройств; 2) пользователь вводит имя устройства; 3) пользователь вводит описание устройства; 4) пользователь добавляет метки устройству; 5) пользователь сохраняет устройство
Постусловие:	Новое IoT-устройство

Таблица 2.3 – Сценарий варианта использования «Редактировать устройство»

Прецедент:	Редактировать устройство
Краткое описание:	Пользователь изменяет свойства IoT-устройства
Главные актеры:	Администратор системы
Второстепенные актеры:	Нет
Предусловие:	Пользователь открыл приложение
Основной поток:	1) пользователь переходит на страницу редактирования устройств; 2) пользователь выбирает необходимое устройство; 3) пользователь изменяет свойства устройства; 4) пользователь обновляет устройство. Альтернативные потоки: 2.а пользователь удаляет устройство
Постусловие:	Отредактированное IoT-устройство

Таблица 2.4 – Сценарий варианта использования «Просмотреть получаемые значения от устройств»

Прецедент:	Просмотреть получаемые значения от устройств
Краткое описание:	Пользователь просматривает полученные значения
Главные актеры:	Администратор системы
Второстепенные актеры:	Нет
Предусловие:	Пользователь открыл приложение
Основной поток:	1) пользователь переходит на страницу просмотра значений; 2) пользователь выбирает необходимую группу устройств; 3) пользователь просматривает устройства
Постусловие:	Нет

В разрабатываемой системе администратор выполняет полный цикл управления устройствами, включая их создание, редактирование и просмотр поступающих от них значений.

2.2 Анализ нефункциональных требований

Нефункциональные требования это критерии, которые описывают аспекты функционирования системы, не связанные с её конкретным поведением или особенностями. Они представляют собой характеристики, влияющие на общую производительность, стабильность и удобство использования приложения.

Существует несколько нефункциональных требований:

- производительность;
- масштабируемость;
- надежность;
- безопасность;
- удобство использования;
- сопровождаемость;
- совместимость;
- конфиденциальность;
- энергоэффективность.

Производительность – это способность системы обрабатывать большое количество запросов, за минимальное время отклика системы.

Масштабируемость – это возможность приложения поддерживать увеличение количества пользователей и объёма данных без значительных потерь производительности.

Надёжность – это способность системы обеспечивать высокий уровень отказоустойчивости и доступности, минимизируя вероятность простоев или сбоев.

Безопасность – это гарантия того, что приложение защищает данные от несанкционированного доступа и соответствует современным стандартам безопасности.

Удобство использования – это интуитивно понятный интерфейс системы, доступный для пользователей с разным уровнем подготовки.

Сопровождаемость – это возможность легко обновлять и поддерживать программное обеспечение, обеспечивая минимальные затраты ресурсов на усовершенствование и исправление ошибок.

Совместимость – это способность приложения работать на всех основных платформах и интегрироваться с другими системами и сервисами.

Конфиденциальность – это строгое соблюдение правил конфиденциальности для гарантии защиты личных и корпоративных данных.

Энергоэффективность – это оптимальное использование ресурсов оборудования, снижающее потребление энергии и уменьшающее нагрузку на инфраструктуру [10].

Для ПС взаимодействия с объектами мониторинга и управления были установлены следующие нефункциональные требования, представленные в таблице 2.5.

Таблица 2.5 – Нефункциональные требования для системы взаимодействия с объектами мониторинга и управления

Требование	Значение
Производительность	Время отклика на запрос не более 2 секунды
Надежность	98% времени система должна работать корректно
Удобство использования	Порог обучения нового пользователя до 30 минут
Конфиденциальность	Гарантия защиты личных данных пользователей
Сопровождаемость	Минимальные затраты времени и ресурсов на обновления, улучшения и исправления ошибок

Данный набор нефункциональных требований позволит разработать надежную и функциональную систему взаимодействия с объектами мониторинга и управления.

2.3 Анализ требований к пользовательскому интерфейсу панели управления

Пользовательский интерфейс – это важный компонент любой системы, обеспечивающий взаимодействие между пользователями и программой. Его основная задача заключается в предоставлении пользователю удобного и интуитивно понятного инструмента для выполнения необходимых действий.

Правильно разработанный интерфейс помогает минимизировать количество ошибок, ускоряет процесс работы и повышает удовлетворяемость пользователей [11].

В пользовательском интерфейсе панели управления необходимо предусмотреть набор из 6 окон.

Главное окно, должно содержать таблицу, в которой будут отображены устройства пользователя их местоположения и статус. В верхней части окна должно быть раскрывающееся меню, позволяющее пользователю перемещаться между другими окнами. Прототип главного окна представлен на рисунке 2.2 [12].



Рисунок 2.2 – Главное окно приложения

Окно добавления устройства, должно содержать поля для ввода значений или выпадающий список из которого пользователь будет выбирать нужное значение, также необходимо предусмотреть переключение выпадающих

списков на форме на поля для свободного ввода, на случай если них не окажется нужного значения, с помощью радиокнопок (radiobutton) каждое поле должно быть помечено меткой, подписывая поле. В нижней части окна необходимо добавить кнопки: «Справка» «Создать» и «Отменить» для подтверждения действий пользователя. Прототип окна добавления устройства представлен на рисунке 2.3.

Прототип окна «добавления устройств» представляет собой графический интерфейс с заголовочной панелью, содержащей кнопки «-», «□» и «x». Основное содержимое окна разделено на две секции: «Расположение» и «Устройство».

В секции «Расположение»:

- Поле «Здание» с выпадающим списком «Список домов».
- Поле «Комната» с выпадающим списком «Список комнат в доме».
- Три радиокнопки: «Добавить здание и комнату», «Добавить только комнату» и «Выбрать из предложенных» (выбранная).

В секции «Устройство»:

- Поле «Выберите устройство» с выпадающим списком «Список устройств».
- Поле «Имя устройства».
- Поле «Описание устройства».
- Чекбокс «Включить сразу».
- Секция «Параметры» с полями «Минимум» и «Максимум», каждое со значением «3» и кнопками «↑» и «↓».

В нижней части окна расположены три кнопки: «Справка», «Создать» и «Отменить».

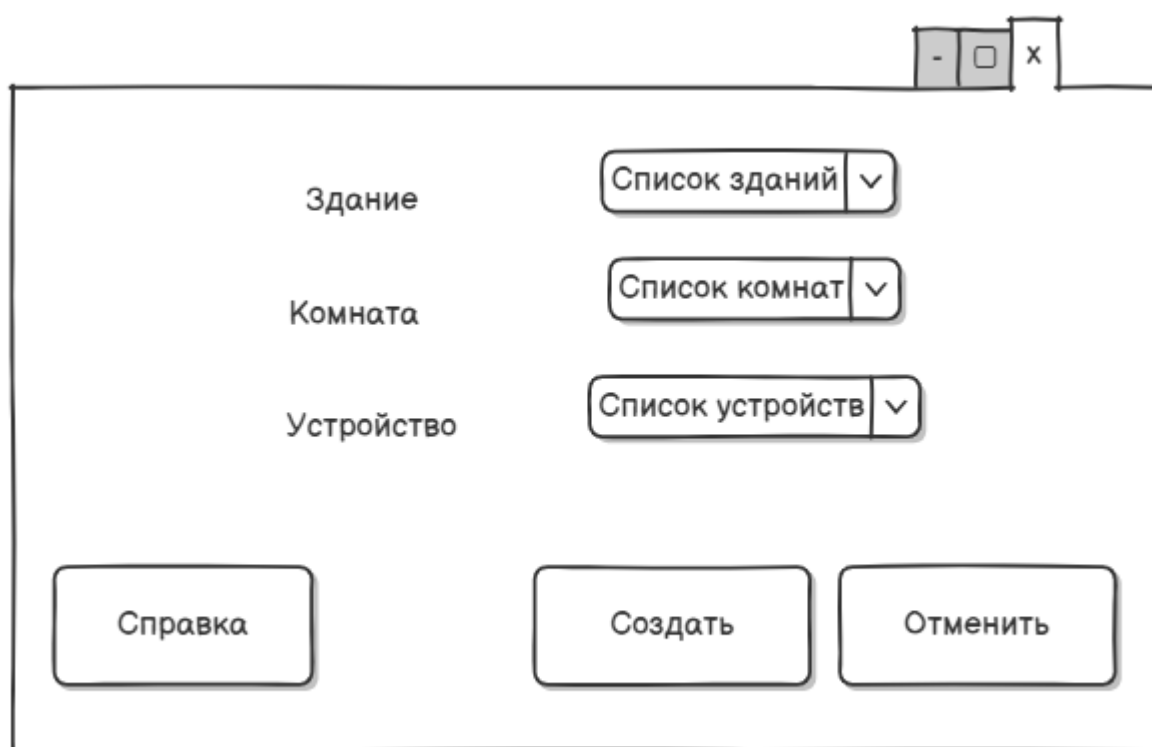
Рисунок 2.3 – Прототип окна «добавления устройств»

Окна редактирования и удаления устройства, должны иметь выпадающие списки, благодаря которым пользователям будет находить необходимое устройство. В окно редактирования устройства также необходимо добавить свойства устройства, которые пользователь будет менять при необходимости. Окна должны иметь кнопку для получения справочной информации по окну, кнопку отмены действия, и кнопку подтверждения изменения или удаления соответственно. Прототип окон редактирования и удаления устройств представлены на рисунке 2.4 и рисунке 2.5 соответственно.



The image shows a window titled "редактирования устройств" (Device Editing). It features a title bar with standard window controls (minimize, maximize, close). The main content area contains four labels on the left and their corresponding controls on the right: "Здание" (Building) with a dropdown menu "Список зданий" (Building list); "Комната" (Room) with a dropdown menu "Список комнат" (Room list); "Устройство" (Device) with a dropdown menu "Список устройств" (Device list); and "Описание устройства" (Device description) with a text input field. Below these is a checkbox labeled "Состояние устройства (вкл/выкл)" (Device status (on/off)). At the bottom, there are three buttons: "Справка" (Help), "Создать" (Create), and "Отменить" (Cancel).

Рисунок 2.4 – Прототип окна «редактирования устройств»



The image shows a window titled "Удаления устройств" (Device Deletion). It features a title bar with standard window controls (minimize, maximize, close). The main content area contains three labels on the left and their corresponding controls on the right: "Здание" (Building) with a dropdown menu "Список зданий" (Building list); "Комната" (Room) with a dropdown menu "Список комнат" (Room list); and "Устройство" (Device) with a dropdown menu "Список устройств" (Device list). At the bottom, there are three buttons: "Справка" (Help), "Создать" (Create), and "Отменить" (Cancel).

Рисунок 2. 5 – Прототип окна «Удаления устройств»

Окно подключения к серверу должно иметь раскрывающийся список с адресами имеющихся серверов, а также поле для свободного ввода, если нужный сервер отсутствует в списке. Необходимо добавить кнопку «Назад» и кнопку подключения к серверу. Прототип окна подключения к серверу устройства представлен на рисунке 2.6.

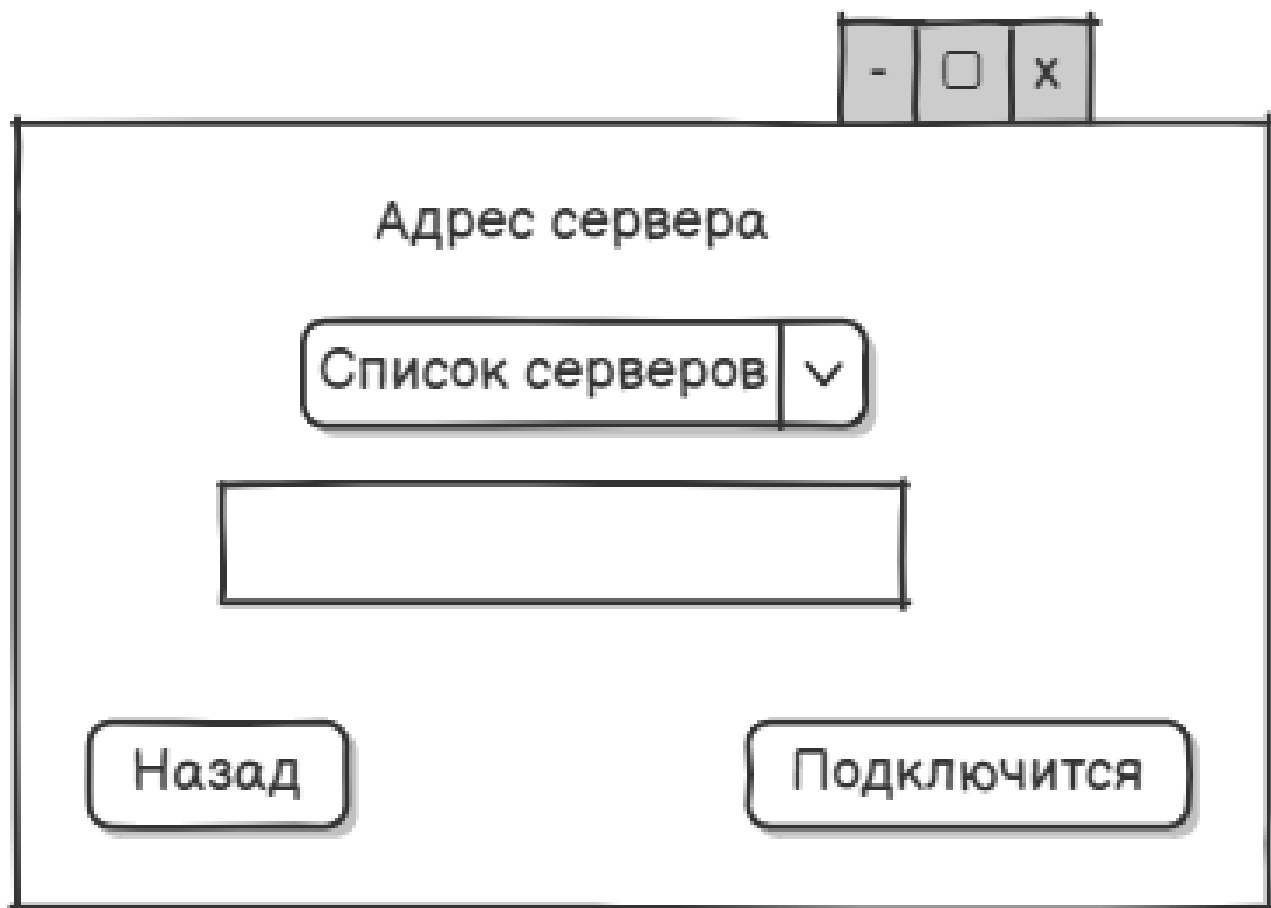


Рисунок 2.6 – Прототип окна «Подключение к серверу»

Окно просмотра значений от устройств, должно иметь окно с отображением сгруппированных значений по комнатам, которые сгруппированы по зданиям. Необходимо добавить кнопку для получения справочной информации по странице, а также кнопку для выхода на главное окно приложения. Прототип просмотра значений от устройств представлен на рисунке 2.7.

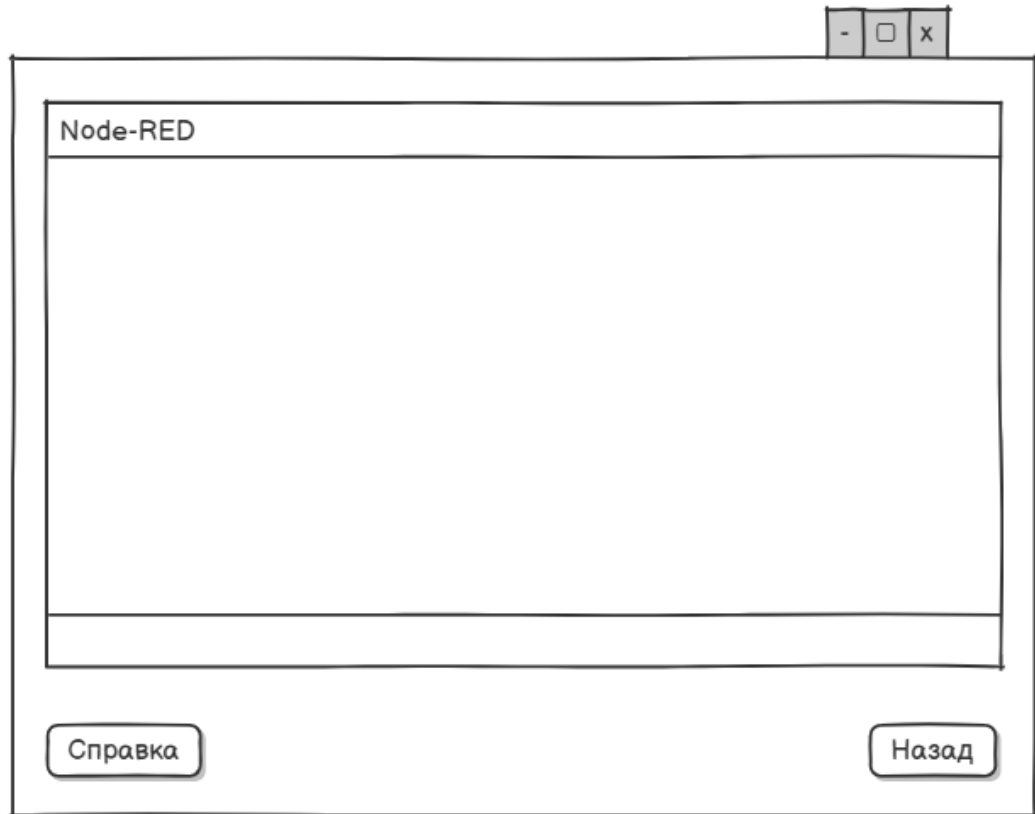


Рисунок 2.7 – Прототип окна «Просмотреть значения»

После завершения работы в окнах пользователь обязательно должен возвращаться на главное окно. Схема навигации по приложению отображена на рисунке 2.8.

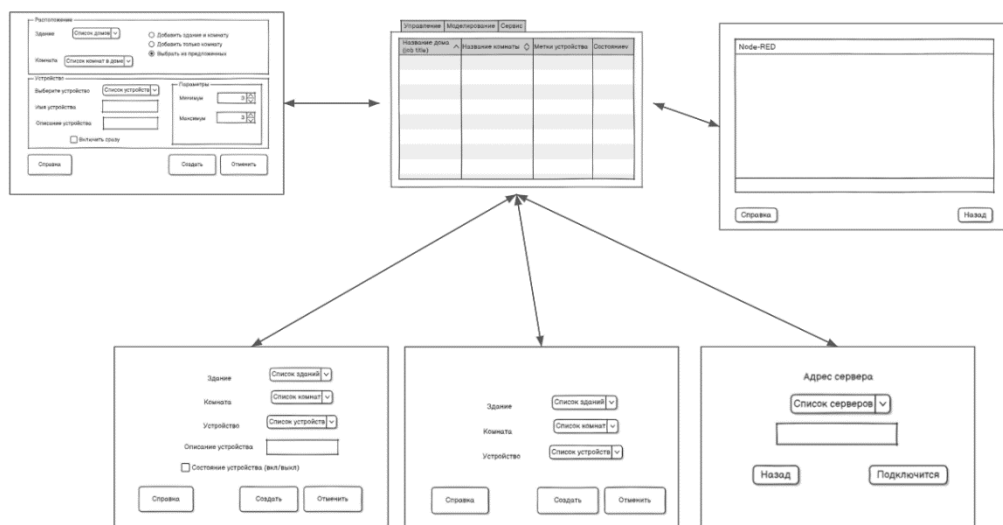


Рисунок 2.8 – Схема навигации приложения

Данная схема обеспечивает интуитивно понятный интерфейс для новых пользователей, позволяя легко ориентироваться в системе, находить нужные разделы для работы с устройствами и эффективно перемещаться между окнами. Такое структурированное представление упрощает выполнение операций, минимизирует риск ошибок и повышает общую производительность приложения.

2.4 Анализ требований к аппаратно-программному обеспечению

2.4.1 Требования к аппаратному обеспечению

Аппаратное обеспечение – это совокупность физических компонентов и устройств, которые составляют компьютерную систему и обеспечивают ее функциональность. Оно включает в себя как электронные, так и механические части, которые работают совместно для выполнения различных задач [13].

Аппаратное обеспечение состоит из:

- процессор;
- оперативная память;
- накопитель данных;
- графическая карта;
- блок питания.

Процессор выполняет основные вычисления и обрабатывает команды.

Оперативная память служит временным хранилищем для данных.

Накопитель данных предназначены для длительного хранения информации.

Графическая карта обрабатывает визуальные данные и отвечает за рендеринг графики.

Для разработки приложения взаимодействия с объектами мониторинга и управления были установлены минимальные и рекомендуемые требования.

Минимальные требования к аппаратному обеспечению отображены в таблице 2.6

Таблица 2.6 – Минимальные требования к аппаратному обеспечению

Компонент	Значение
Процессор	Intel core i3 или аналогичный с частотой не менее 2.0 гГц
Оперативная	8 гб
Накопитель	1 гб свободного места на жестком диске
Видеокарта	Интегрированная видеокарта с поддержкой opengl 2.0
Операционная система	Windows 7

Рекомендуемые требования к аппаратному обеспечению отображены в таблице 2.7.

Таблица 2.7 – Рекомендуемые требования к аппаратному обеспечению

Компонент	Значение
процессор	Intel Core i5 или аналогичный с частотой не менее 2.4 ГГц
оперативная	16 ГБ
накопитель	5 ГБ свободного места на жестком диске
видеокарта	Интегрированная видеокарта с поддержкой OpenGL 3.0
Операционная система	Windows 10

Такие требования позволяют обеспечить стабильную и надежную работу ПС взаимодействия с объектами мониторинга и управления с использованием облачных технологий.

2.4.2 Требования к системному программному обеспечению

Программное обеспечение (ПО) –это совокупность программ, баз данных, файлов, а также описывающих их документов, составляющих систему, для решения группы связанных задач на одном или нескольких взаимодействующих компьютерах.

Программное обеспечение делится на несколько категорий:

- системное;
- прикладное;
- утилиты.

Системное ПО управляет аппаратными ресурсами компьютера и предоставляет базовые функции, необходимые для работы других программ.

Прикладное ПО позволяет выполнять конкретные задачи, такие как тестовая обработка, создание таблиц, графический дизайн и т. д.

Утилиты служат для управления и обслуживания системы, обеспечивая оптимизацию ее работы.

Для развертывания и использования приложения взаимодействия с объектами мониторинга и управления было поставлено следующее необходимое системное программное обеспечение [14].

Системное программное обеспечения для развертывание приложения отображено в таблице 2.8.

Таблица 2.8 — Системное программное обеспечение для развертывания

Компонент	Значение
Операционная система	Centos, которая обеспечивает стабильную и надежную платформу для работы node-red
Node-RED	Инструмент для визуального программирования, который используется для создания и управления потоками данных
Yandex cloud	Облачная платформа, предоставляющая ресурсы для развертывания и масштабирования приложения
Visual studio	Интегрированная среда разработки для создания и отладки rest api и windows forms на c#.

Для использования приложения будет необходим браузер.

Такие требования к программному обеспечению позволяют развернуть клиентское приложение и использовать его.

2.4.3 Требования к средствам разработки

Средства разработки – это набор программных инструментов и приложений, которые помогают разработчикам создавать, тестировать, отлаживать, развёртывать и поддерживать программные продукты.

К программным инструментам относятся редакторы кода, которые позволяют писать и редактировать исходный код, а также компиляторы и интерпретаторы, преобразующие код из его исходной формы в машинный код или выполняющие его напрямую.

Отладчики необходимы для нахождения и исправления ошибок в коде, позволяя разработчиком анализировать выполнение программ и проверять

значения переменных, средства разработки интерфейсов упрощают создания графических интерфейсов пользователя. Фреймворки и библиотеки предлагают наборы компонентов и решений для ускорения разработки избавляя от необходимости писать код с нуля. Инструменты для тестирования позволяют проверять качество программного обеспечения.

Средства разработки играют ключевую роль в процессе создания ПО, способствуя более эффективной организации работы разработчиков и повышение производительности и качества конечного продукта.

Для разработки системы были использованы языки программирования:

- C#;
- JavaScript.

C# был выбран для создания REST API и клиентского приложения благодаря большим возможностям для разработки серверной части и интеграции с различными сервисами. JavaScript используется для симуляции IoT-устройств в системе Node-RED, поскольку он хорошо подходит для работы с событиями и асинхронными запросами, что критично для IoT-среды.

В качестве среды разработки была выбрана Microsoft Visual Studio, так как она дает широкий выбор набора инструментов для упрощения написания кода и отладки, что значительно повышает эффективность работы.

В ходе разработки также были выбраны следующие фреймворки:

- ASP.NET;
- Windows Forms.

ASP.NET был выбран для создания RESTfull сервиса, так как позволяет обрабатывать большое количество запросов [15].

Windows Forms для разработки настольного приложения с графическим интерфейсом, что позволяет создать удобный интерфейс для пользователей.

Выбор в сторону данных средств разработки основан на анализе альтернативных технологий, таких как Python или Java, потому что в ходе анализа было решено, что C# и Javascript обеспечат более высокую производительность и удобство в данной предметной области проекта.

Применение данных технологий позволяет эффективно реализовать поставленные задачи в рамках проекта.

2.5 Постановка задачи на разработку программных средств

Разрабатываемая система предназначена для моделирования и управления IoT-устройствами, эмулирующими работу датчиков. Она будет обеспечивать эффективное хранение и передачу данных, а также управление устройствами в реальном времени.

Компоненты системы включают в себя Node-RED, который представляет собой визуальную среду для создания потоков данных и моделирования работы датчиков. REST API служит интерфейсом для обмена данными между клиентами и сервером, обеспечивая доступ к функционалу системы. Облачное хранилище Yandex Cloud используется для сохранения данных, полученных от IoT-устройств, с возможностью их анализа и обработки. JavaScript будет языком программирования для добавления логики и обработки данных устройств. C# будет использоваться для разработки клиентских приложений и управления устройствами.

Node-RED позволит создавать потоки обработки данных и визуализировать статусы и значения устройств, а также настраивать различные сценарии для них. Rest API будет выполнять функции создания, редактирования и удаления устройств с сервера, а также выполнять подключение к серверу и отображение необходимых данных клиенту. Облачное хранилище позволит хранить устройства и их данные.

Система должна учитывать возможность добавления новых устройств и пользователей без снижения производительности. Интерфейс должен быть удобным и интуитивно понятным для новых пользователей.

3 Разработка программных средств взаимодействия с объектами мониторинга и управления

3.1 Разработка архитектуры развертывания программных средств

Для отображение основных компонентов приложения была выбрана диаграмма развертывания, которая представляет собой графическое отображение архитектуры системы, иллюстрирующее компоненты программного обеспечения и их распределение по разным узлам или физическим машинам. Она показывает, какие программные модули разворачиваются на каждом узле сети и как налажено взаимодействие между этими модулями, Диаграмма развертывания системы взаимодействия с объектами мониторинга и управления отображена на рисунке 3.1.

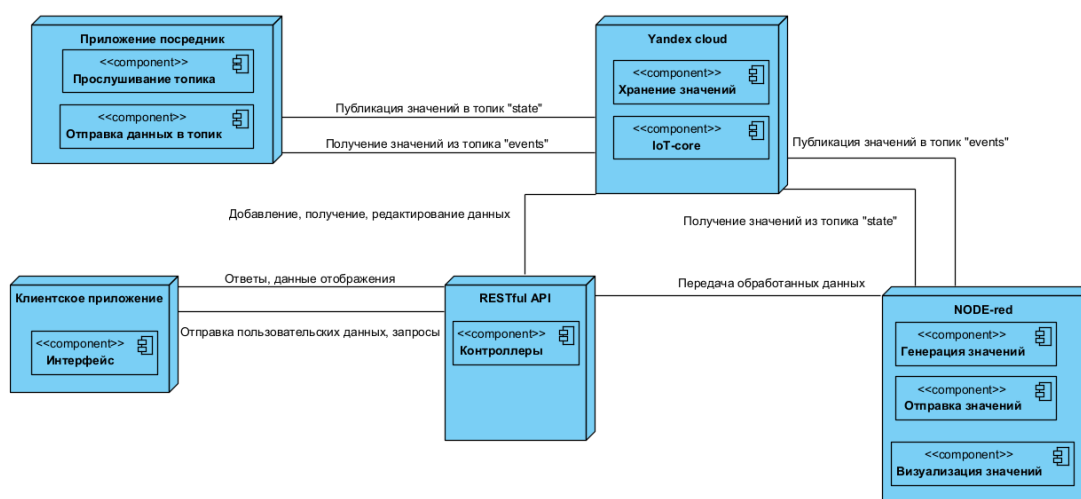


Рисунок 3.1 – Диаграмма развертывания системы

На диаграмме развертывания отображены следующие компоненты:

- клиентское приложение;
- RESTful API;
- NODE-Red;
- Yandex cloud;
- приложение посредник.

Клиентское приложение – это пользовательский интерфейс, через который пользователь взаимодействует с системой. Оно отвечает за отправку пользовательских данных или запросов и получения ответов от RESTful API для отображения данных.

RESTful API – компонент, обеспечивающий взаимодействие клиентского приложения с остальными частями системы. Его основные задачи включают обработку входящих запросов от клиентского приложения, добавления, получение и редактирование данных, а также передачу данных между клиентом и другими компонентами системы.

NODE-Red – компонент для обработки данных. Он состоит из генерации значений, что подразумевает создание исходных данных для последующей обработки и отправки, и визуализации значений, предлагая визуальное представление данных в виде графиков, диаграмм или других форматов.

Yandex Cloud – это центральный облачный компонент, отвечающий за управление данными и их перемещение по топикам. Он включает функции хранения значений, обеспечивая долговременное хранение данных, и IoT-core, управляющий обменом информацией через топики «state» и «events».

Приложение посредник – это консольное приложение, которое выступает связующим звеном между Yandex Cloud и другим оборудованием или приложениями. Оно выполняет функции прослушивания топика, получая данные из топика «events», опубликованного в Yandex Cloud, и отправки данных в топик «state», который обрабатывается дальше другими компонентами.

Все компоненты работают синхронно, чтобы обеспечить регулярный обмен данными, их обработку и визуализацию.

Для компонента RESTful API была выполнена подробная диаграмма классов. Это один из видов структурных диаграмм, используемых в языке моделирования UML (Unified Modeling Language), диаграмма классов системы представлена на рисунке 3.2.

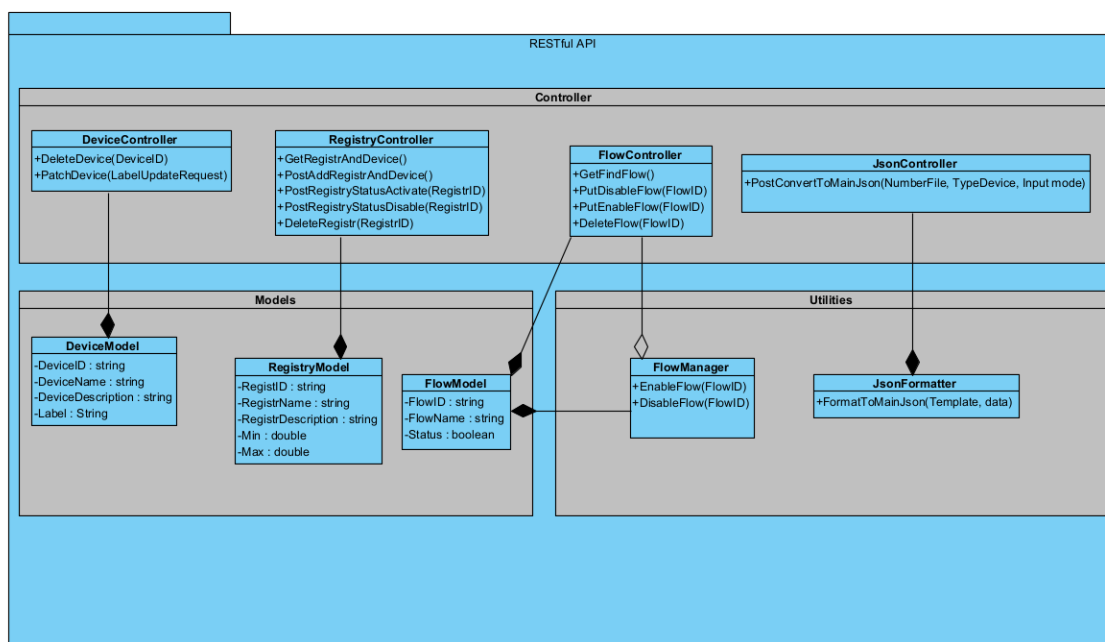


Рисунок 3.2 – Диаграмма классов компонента Restful API

Диаграмма разделена на несколько модулей (пакетов), такие как Controller, Models и Utilities. Каждый из этих модулей содержит группы классов, выполняющие определённые функции.

Классы предоставляющие основной функционал взаимодействия пакета Controller:

- DeviceController – управляет устройствами с помощью методов (DeleteDevice, PatchDevice), работающих с идентификаторами устройств;
- FlowController – отвечает за операции над потоками данных (GetFindFlow, PutEnableFlow);
- RegistryController – обрабатывает реестры, предоставляя методы добавления, активации и удаления;
- JsonController – обеспечивает преобразование данных в нужный формат.

Models содержит классы, описывающие сущности системы:

- DeviceModel – хранит идентификатор устройства, описание и метки;
- RegistryModel – описывает атрибуты реестров, такие как идентификатор, описание, минимальные и максимальные значения;

– FlowModel – отражает структуру потоков данных.

Utilities данный раздел представляет вспомогательные классы:

– FlowManager – позволяет включать или отключать потоки (EnableFlow, DisableFlow);

– JsonFormatter – ответственный за форматирование данных в JSON.

Таким образом система является микросервисной архитектурой, состоит из отдельных компонентов, каждый из которых выполняет определенную задачи, что позволяет обеспечить гибкость и масштабируемость системы.

3.2 Разработка программных шаблонов для реализации функций мониторинга управления

Программные шаблоны представляют собой предварительно определенные структуры кода, которые можно использовать для упрощения процессов. Они содержат основные элементы, необходимые для создания конкретной функциональности.

Для реализации функциональности IoT-устройств были разработаны шаблоны в формате JSON-файлов, [16] которые содержат «placeholders» для вставки необходимых параметров. Выбор конкретного шаблона зависит от типа IoT-устройства, которое требуется создать. Каждый шаблон включает в себя необходимые узлы для симуляции в Node-RED, а также значения, заданные пользователем. Подробное описание всех доступных шаблонов представлено в таблице 3.1.

Таблица 3.1 – Набор шаблонов функциональности устройств

Шаблон	Описание
Шаблон для формирования сообщения от целочисленного датчика	Генерирует и отправляет сообщения, с целочисленным значением, представляющим измеряемый параметр
Шаблон для формирования сообщения от вещественного датчика	Генерирует и отправляет сообщения, с вещественным значением, представляющим измеряемый параметр
Шаблон для формирования сообщения от логического датчика	Генерирует и отправляет сообщения, с логическим значением, представляющим измеряемый параметр

Продолжение таблицы 3.1

Шаблон	Описание
Шаблон для обработки сообщения от целочисленного датчика	Обрабатывает входящие сообщения со значениями целочисленного типа и отображает их на форме
Шаблон для обработки сообщения от вещественного датчика	Обрабатывает входящие сообщения со значениями вещественного типа и отображает их на форме
Шаблон для обработки сообщения от логического датчика	Обрабатывает входящие сообщения со значениями логического типа и отображает их на форме

Фрагмент кода для формирования JSON файла на основе шаблона представлен на листинг 3.1.

Листинг 3.1 – Код замены переменных в шаблоне

```
// Чтение JSON файла
var jsonData =
System.IO.File.ReadAllText(filePath);
JToken jsonObject = JToken.Parse(jsonData);

// Замена значений по плейсхолдерам
jsonData = jsonData
    .Replace("{{MESSAGE}}", input.Message)
    .Replace("{{REGISTRY-NAME}}",
input.RegistryName)
    .Replace("{{REGISTRY-ID}}", input.RegistryId)
    .Replace("{{Zdanie}}",
input.LabelsReg.Location)
    .Replace("{{ROMM}}", input.LabelsDev.Location)
    .Replace("{{DEVICE-NAME}}", input.DeviceName)
    .Replace("{{MIN}}", input.MinValue)
    .Replace("{{MAX}}", input.MaxValue)
    .Replace("{{DEVICE-ID}}", input.DeviceID);

jsonObject = JToken.Parse(jsonData);

// Путь для нового JSON файла
string newFilePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates", "new-
node-red-template.json");

// Запись обновленного JSON в новый файл
System.IO.File.WriteAllText(newFilePath,
jsonObject.ToString());

// Возврат нового JSON файла
return PhysicalFile(newFilePath,
"application/json", "new-node-red-template.json");
```

Код считывает содержимое JSON файла по указанному пути, заменяет плейсхолдеры в данных на значения из объекта «input». Затем обновленный JSON сохраняется в новый файл по заданному пути и возвращается в ответе.

При помощи этих шаблонов пользователь может легко развернуть IoT устройства, упростив процесс настройки и интеграции. Листинг шаблонов для IoT устройств представлен в приложение Б.

3.3 Разработка RESTfulAPI-сервера для взаимодействия с приложениями информационно-управляющей платформы

REST (Representational State Transfer) – это архитектурный стиль для разработки веб-сервисов, который использует стандартные HTTP-методы для взаимодействия между клиентом и сервером. Основная идея REST заключается в том, чтобы предоставлять доступ к ресурсам через определенные URL-адреса, называемые endpoints.

Endpoint представляют собой точные местоположения на сервере, куда могут отправляться запросы. Каждый endpoints соответствует определенному ресурсу или функции, что позволяет клиентам взаимодействовать с сервером, используя стандартные методы, такие как GET, POST, PUT и DELETE. Это обеспечивает простоты использования и масштабируемость интерфейсов [17].

В рамках разработки RESTful API-сервера для взаимодействия с приложениями информационно-управляющей платформы был разработан набор endpoints представленный в таблице 3.2.

Таблица 3.2 – Список разработанных эндпоинтов

Метод	Название	Входные данные	Описание
Get	FindIDFlow	Label	Поиск идентификатора потока
Get	RegistrAndDevice	-	Поиск всех регистров и устройств и них
Post	ConvertToMainJson	NumberFile,TypeDevice,InputModel	Форматирование файла JSON на основе шаблон
Post	AddFlowFromTemplate	-	Добавление потока в node-red

Продолжение таблицы 3.2

Метод	Название	Входные данные	Описание
Post	AddRegistrAndDevice	RegistrName,RegistrDescription,DeviceName,DeviceDescription,LabelRegistr,LabelDevice,Min,Max	Создание реестра и устройства в yandex cloud
Post	RegistryStatus/activate	RegistrID	Выключение реестра
Post	RegistryStatus/disable	RegistrID	Включение реестра
Put	DisableFlow	FlowID	Выключение потока
Put	EnableFlow	FlowID	Включение потока
Delete	DeletDevice	DeviceID	Удаление устройства
Delete	DeletRegistr	RegistrID	Удаление реестра
Delete	DeletFlow	FlowID	Удаление потока
Patch	Device	LabelUpdateRequest	Изменения меток устройства

Код эндпоинта «GetTabIdAsync», который необходим для поиска нужного потока в Node-RED, отображен в листинг 3.2.

Листинг 3.2 – Код endpoints «GetTabIdAsync»

```
[HttpGet("{label}")]
public async Task<IActionResult> GetTabIdAsync(string
label)
{
    try
    {
        // Запрашиваем данные от Node-RED
        var response = await
_httpClient.GetStringAsync("http://127.0.0.1:1880/flows");

        // Преобразуем ответ в JSON
        var flows = JSONArray.Parse(response);

        // Ищем tab с нужной меткой
        foreach (var flow in flows)
        {
            if ((string)flow["type"] == "tab" &&
(string)flow["label"] == label)
            {
                var tabId = (string)flow["id"];
                return Ok(new { Id = tabId });
            }
        }
    }
}
```

```

        return NotFound("Tab с указанной меткой не
найден.");
    }
    catch (HttpRequestException e)
    {
        return StatusCode(503, "Ошибка подключения к Node-
RED: " + e.Message);
    }
}

```

В данном листинге представлен метод «GetTabIdAsync», который отвечает за получение идентификатора потока из Node-RED по заданной метке. Метод выполняет HTTP GET-запрос к локальному серверу Node-RED для получения списка потоков. Затем, преобразуя полученный ответ в JSON-формат, он ищет соответствующий поток с указанной меткой. В случае нахождения нужного потока возвращается его идентификатор в формате JSON с статусом 200. Если поток не найден, возвращается сообщение о ненахождении с кодом 404.

Данный набор эндпоинтов позволяет пользователю взаимодействовать с системой мониторинга и управления в полном объеме. Полный листинг всех эндпоинтов представлен в приложение В.

3.4 Разработка клиентского приложения

3.4.1 Разработка интерфейса

Клиентское приложение, которое обеспечивает эффективное взаимодействие с RESTful API-сервером обладает интерфейсом для пользователей, позволяя им получать доступ к функционалу информационно-управляющей платформы [18]. Всего клиентское приложение включает в себя 6 окон.

Главное окно выполняет роль основного интерфейса для пользователей. В верхней части окна находится главное меню, которое состоит из пунктов «Устройство», «Моделирование» и «Сервис».

В меню «Устройства» находятся следующие кнопки:

- добавить – позволяют пользователю перейти на окно добавления устройства;
- редактировать – позволяет перейти на окно редактирования устройства;
- удалить – позволяет перейти на окно удаления устройств;
- выход – кнопка для выхода из приложения.

В меню «Моделирование» находятся следующие кнопки:

- просмотреть значения – открывает страницу с текущими значениями от устройств;
- подключиться к серверу – открывает страницу на которой пользователь может подключиться к необходимому серверу.

В меню «Сервис» находится кнопка для получения справочной информации о странице.

Центральную часть окна занимает таблица, содержащая информацию о текущих устройствах, а именно «Здание», «Комната», «Имя устройства» и «Статус», что позволяет пользователю посмотреть состояние устройств и его расположение. (Рисунок 3.3)

Устройство Моделирование Сервис				
	Здание	Комната	Имя устройства	Статус
▶	manufacture	workshop3	humidity	ACTIVE
	store	zone1	temperatur	ACTIVE
	office	room2	light	ACTIVE
	manufacture	workshop2	temperatur	ACTIVE
	office	room1	temperature	ACTIVE
	office	room1	light	ACTIVE
	store	zone2	temperatur	ACTIVE
	store	zone1	countervisitor	ACTIVE
	manufacture	workshop1	temperatur	ACTIVE
*				

Рисунок 3.3 – Главное окно приложения

На окне для добавления устройств, в верхней части окна, расположен блок «Расположение» где располагаются поля для выбора здания и комнаты, если в них нет нужных значений, то пользователь переключает режим, и поля с выпадающим списком меняются на поля свободного ввода.

Ниже находится группа «Устройства» которая включает в себя следующие поля:

- названия устройства – выпадающий список где можно выбрать необходимое устройство. (например: чайник, выключатель, датчик температуры);

- имя устройства – поле куда пользователь укажет необходимое имя для нового устройства;

- описание – поле куда пользователь по желанию напишет описание нового устройства;

– включить сразу – метка для указания следует ли включить устройство сразу или оставить выключенным;

– группа элементов «Параметры», которая содержит поля для ввода числовых значений минимума и максимума, применяется только для устройств, которые измеряют числовые значения (например: датчик температуры, датчик влажности).

В нижней части окна располагаются кнопки:

- справка – для получения справочной информации об окне;
- создать – для создания устройства с указанными значениями;
- отменить – кнопка для возвращения на главное окно.

Благодаря группировкам элементов на окне, интерфейс получился легко воспринимаемым для пользователя, что упрощает процесс взаимодействия, позволяет интуитивно находить нужные функции и повышает общую эффективность работы с приложением. Интерфейс окна отображен на рисунке 3.4.

Расположение

Здание

Комната

☐ Добавить новое здание и комнату

☐ Добавить новую комнату

☒ Оставить

Устройство

Устройство

Имя устройства

Описание

Включить сразу ☐

Параметры

Минимальное значение

Максимальное значение

Справка

Создать

Отменить

Рисунок 3.4 – Окно добавления устройства

На рисунке 3.5 изображено окно редактирования устройств, которое предоставляет пользователю возможность изменить свойства уже существующих в системе устройств.

На окне отображены следующие элементы:

Выпадающие списки для выбора здания, комнаты и устройства, что позволяет пользователю отфильтровать не нужные устройства, путем последовательного выбора значений. Когда пользователь выбирает здание ему, показываются только соответствующие ему комнаты, когда выбирает комнату то отображаются только соответствующие устройства.

Ниже находится описание устройство и отметка о состоянии устройства пользователь может ввести новое значение или изменить отметку и значение новое значение сохранится у устройства.

В нижней части окна располагаются кнопки:

- справка – для получения справочной информации об окне;
- сохранить – для сохранения новых значений устройства;
- отменить – кнопка для возвращения на главное окно.

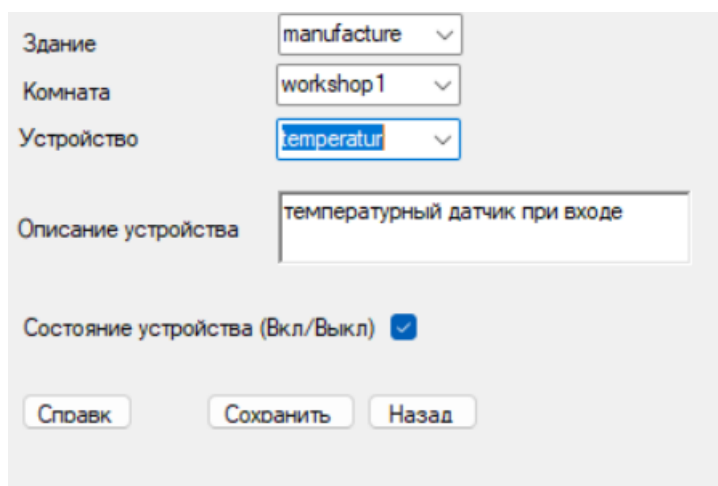


Рисунок 3.5 – Окно редактирования устройства

На рисунке 3.6 изображено окно удаления устройств, которое предоставляет пользователю возможность удалить необходимое устройство из системы.

На окне отображены следующие элементы:

Выпадающие списки для выбора здания, комнаты и устройства, что позволяет пользователю отфильтровать не нужные устройства, путем выбора наводящих значений.

В нижней части окна располагаются кнопки:

- справка – для получения справочной информации об окне;
- удалить – для удаления устройств;
- отменить – для возвращения на главное окно.

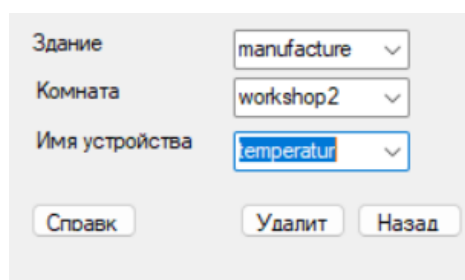


Рисунок 3.6 – Окно удаления устройства

На рисунке 3.7 отображено окно для подключения к серверу. Окно необходимо для подключения пользователя к имеющемуся или к новому серверу

Для выбора сервера расположен выпадающий список, если в нем не оказалось нужного сервера, то пользователь выбирает «Нет подходящего», и поле для свободного ввода адреса сервера становится доступно.

Снизу расположены кнопки назад для возвращения на главную страницу и кнопка подключить которая начинает подключение приложение к указанному серверу.

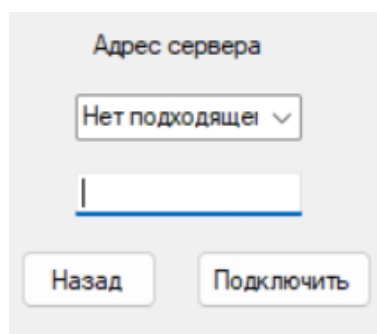


Рисунок 3.7 – Окно подключения к серверу

Окно просмотра значений отображено на рисунке 3.8.

Основную часть окна занимает отображение значений от устройств при помощи подключения к указанному ранее серверу. Значения отображаются группировкой по комнатам, которые в свою очередь сгруппированы по зданиям, выбор здания происходит выплывающим списком слева.

Снизу расположены кнопка «Справка» для получения справочной информации о окне и кнопка «Назад» для возвращения на главную страницу.

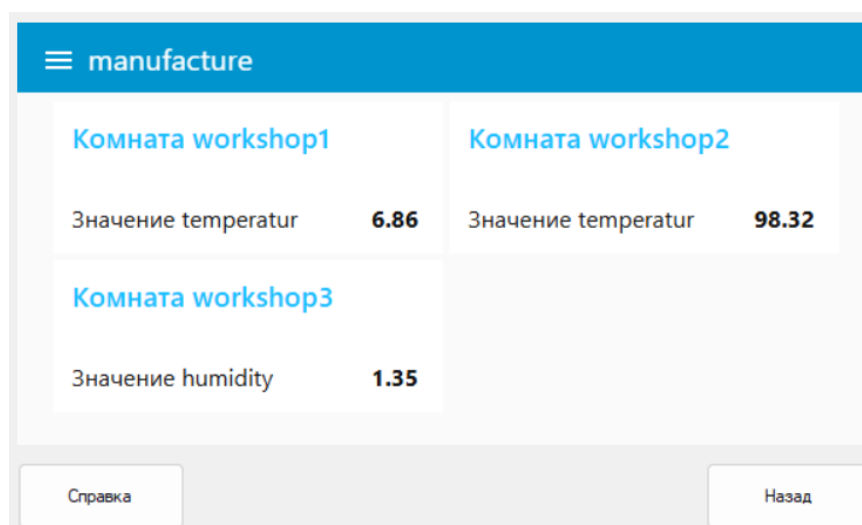


Рисунок 3.8 – Окно просмотра значений от устройств

Таким образом клиентской приложения для взаимодействия с объектами мониторинга и управления получилось структурированным, легким для восприятия и простым для пользования как для новых пользователей, так и для опытных.

3.4.2 Разработка программных средств клиентского приложения

Для клиентского приложения была разработана диаграмма компонентов, которая демонстрирует структуру системы и взаимодействие ключевых модулей. Диаграмма компонент представлена на рисунке 3.9.

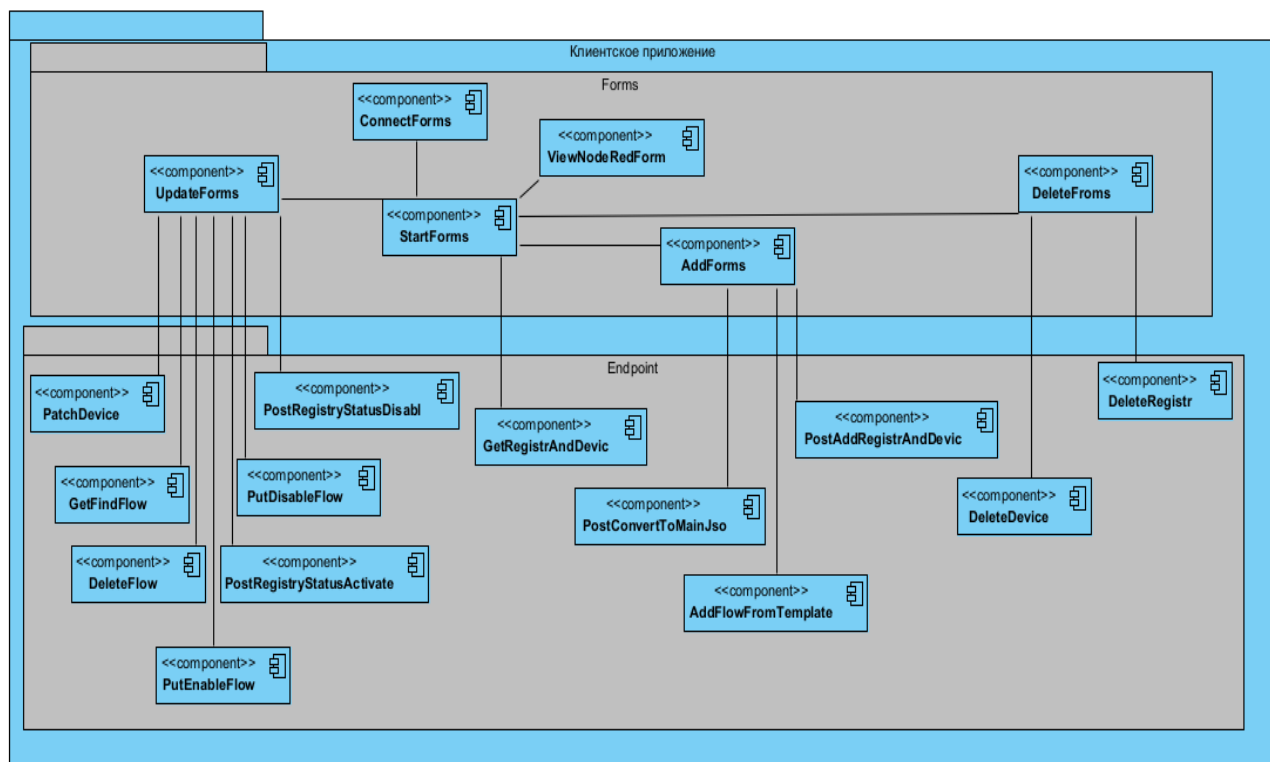


Рисунок 3.9 – диаграмма компонентов для клиентского приложения

На диаграмме изображены два основных модуля в виде пакетов. Модуль Forms содержит компоненты форм приложения, обеспечивая их создание, обновление и управление. Модуль Endpoint включает в себя точки вызова API, которые обеспечивают возможность взаимодействия с сервисами и реализацию бизнес-логики приложения.

Подробное описание компонентов представлено в таблице 3.3.

Таблица 3.3 – Спецификации диаграммы компонентов клиентского приложения

Модуль	Компонент	Описание
Forms	Updateforms	Форма приложения для изменения устройств
Forms	Connectforms	Форма приложения для подключения к серверу
Forms	Startforms	Стартовая форма приложения
Forms	Viewnoderedforms	Форма отображение значений от устройств
Forms	Addforms	Форма добавления устройств в систему
Forms	Deletefroms	Форма удаления устройств из системы
Endpoint	Deleteregistr	API-запрос для удаления реестра из Yandex Cloud
Endpoint	Deletedevice	API-запрос для удаления устройства из Yandex Cloud
Endpoint	Postaddregistranddevice	API-запрос для Добавление регистра и устройства в Yandex Cloud
Endpoint	Addflowfromtemplate	API-запрос для Развертывания устройства в NODE-red

Endpoint	Postconverttomainjson	API-запрос для Форматирование шаблона для NODE-red
Endpoint	Getregistranddevice	API-запрос для Получение списка реестров и устройств
Endpoint	Postregistrystatusdisable	API-запрос для Выключение реестра в Yandex Cloud
Endpoint	Putdisableflow	API-запрос для Выключение потока в NODE-red
Endpoint	Postregistrystatusactivate	API-запрос для Включение реестра в Yandex Cloud
Endpoint	Putenableflow	API-запрос для Включение потока в NODE-red
Endpoint	Deleteflow	API-запрос для Удаление потока из NODE-red
Endpoint	Getfindflow	API-запрос для Получение списка потоков
Endpoint	Patchdevice	API-запрос для Изменение меток устройств

Благодаря такой структуре компонентов, приложение обеспечивает полный цикл управления устройствами и их мониторинг.

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 – это система мероприятий, направленных на обеспечение соответствия продукции или услуг установленным стандартам и требованиям. Он включает в себя планирование, осуществление и анализ результатов проверок, что позволяет выявить и устранить дефекты на различных этапах производства.

Главная цель контроля качества приложений – убедиться, что программа соответствует всем требованиям и корректно функционирует [19].

Основные показатели контроля качества программных средств представляют собой количественные характеристики свойств программного обеспечения, используемые для оценки его качества в конкретных условиях разработки, эксплуатации и сопровождения

Показатели подразделяются на единичные, относящиеся к отдельным свойствам, и комплексные, охватывающие сразу несколько характеристик.

Примерами являются показатели надежности, определяющие устойчивость ПО к внешним факторам, такие как сбои оборудования и некорректность входных данных, показатели сопровождения, обеспечивающие удобство исправления ошибок и поддержание актуальности продукта, показатели удобства применения, отражающие легкость освоения и доступности документации, показатели эффективности, характеризующие производительность ПО относительно ресурсов и временных ограничений; и показатели переносимости, оценивающие возможность адаптации ПО к новым условиям среды исполнения.

Методы контроля качества ПО можно разделить на несколько категорий:

1) методы, связанные с выявлением реального функционирования ПО: Эти методы включают тестирование, профилирование и измерение

количественных показателей качества, а также использование различных метрик;

2) методы, нацеленные на обнаружение нарушений стандартов: Эти методы направлены на выявление отклонений от стандартов создания программного кода, проектных моделей и документации;

3) методы обычного или формализованного анализа проектной документации и исходного кода: Сюда входят методы анализа архитектуры ПО, а также методы формального доказательства свойств ПО и формального анализа эффективности используемых алгоритмов;

4) методы определения показателей качества на основе симуляции работы ПО: Эти методы используют модели разного рода для оценки качества принимаемых решений.

В рамках данной выпускной квалификационной работы в качестве мероприятий по контролю качества было решено провести тестирование и подсчет метрик программного кода [21].

4.1.2 Обоснование выбора модели тестирования

Тестирование программного обеспечения (Software Testing) представляет собой важнейший этап разработки ПО, направленный на выявление и устранение ошибок. Оно заключается в проверке совпадения реальных результатов выполнения программы с ожидаемыми, что происходит посредством проведения тестов. Каждый набор тестов формируется с учетом особенностей программного обеспечения, функциональных требований и возможных сценариев использования [21].

Основная цель тестирования состоит в том, чтобы удостовериться, что программное обеспечение удовлетворяет предъявленным требованиям и заявленным характеристикам качества. Этот процесс помогает разработчикам и тестировщикам убедиться в надежности продукта и его готовности к использованию с конечными пользователями. Также тестирование направлено

на выявление уязвимостей и дефектов до их попадания в конечную версию продукта, что минимизирует риск их обнаружения реальными пользователями.

Все виды тестирования описаны в Таблице 4.1.

Таблица 4.1 – Виды тестирования

Название	Описание	Пример
Модульн ые тесты	Метод тестирования программного обеспечения, при котором проверяются отдельные модули или компоненты программы (функции, методы, классы).	Тестирование функции, которая вычисляет сумму двух чисел. Проверяется, что функция возвращает правильный результат для различных входных данных.
Интеграц ионные тесты	Уровень тестирования, который направлен на проверку взаимодействия нескольких модулей или компонентов системы	Тестирование взаимодействия между двумя модулями, например, проверка того, что модуль базы данных корректно взаимодействует с модулем пользовательского интерфейса.
Функцио нальные тесты	Тип проверки программного обеспечения, который проверяет, соответствует ли функциональность приложения его требованиям и спецификациям.	Тестирование функции регистрации пользователя. Проверяется, что пользователь может успешно зарегистрироваться с корректными данными и получить ошибку при вводе некорректных данных.
Нагрузоч ные тесты	Процесс проверки производительности программного обеспечения или системы при увеличивающейся нагрузке	Тестирование веб-приложения под нагрузкой в 1000 одновременных пользователей. Проверяется, как система справляется с таким количеством запросов и не падает ли она.
Приемочн ое тестирова ние	Этап в процессе разработки программного обеспечения, на котором проверяется, соответствует ли продукт заранее определённым требованиям и спецификациям.	Тестирование приложения на соответствие требованиям заказчика. Проверяется, что все функции работают так, как было оговорено в техническом задании.
Тестирова ние производ ительност и	Процесс оценки скорости, устойчивости и масштабируемости системы под различными нагрузками.	Измерение времени отклика системы при выполнении различных операций, таких как поиск по базе данных или загрузка страницы.
Регрессио нные тесты	Проверка работоспособности приложения после внесения модификаций и доработок.	Повторное тестирование функций приложения после внесения изменений в код. Проверяется, что новые изменения не сломали ранее работавшие функции.

Поиск и устранение ошибок на стадии тестирования помогает предотвратить серьезные последствия, такие как функциональные сбои,

проблемы с пользовательским опытом или угроза кибербезопасности. Кроме того, тестирование способствует повышению доверия к программному продукту, так как гарантирует его стабильность и соответствие требованиям.

Для приложения взаимодействия устройств мониторинга и управления было выбрано функциональное тестирование логики и интерфейса методом черного ящика [22].

4.1.3 Обоснование выбора метрик программного кода

Метрики тестирования – это количественные показатели, которые позволяют оценить качество программного продукта, эффективность тестирования и уровень соответствия заявленным требованиям.

Контроль качества предполагает наличие метрик, которые помогают оценить уровень качества программного продукта. Эти метрики могут измерять различные аспекты программного обеспечения, такие как сложность кода, функциональная надежность, производительность и другие. Метрики могут быть применены ко всей кодовой базе или к отдельным частям, таким как классы, методы или даже отдельные блоки кода [23].

В настоящее время существует множество различных метрик для оценки программ. Их можно разделить на несколько категорий:

- 1) измерение сложности программ: оценка сложности кода, которая может включать в себя количество строк, циклов и других структурных элементов;
- 2) оценка функциональной надежности: проверка того, насколько надежно работает программное обеспечение, например, как часто возникают ошибки или сбои;
- 3) измерение производительности ПО: оценка того, насколько быстро и эффективно работает программа, например, время выполнения операций или использование ресурсов;
- 4) измерение уровня языковых средств: оценка использования различных языковых конструкций и их влияния на качество кода;

5) измерение восприятия и понимания программных текстов: оценка того, насколько легко программисты могут читать и понимать код;

6) измерение производительности труда программистов: оценка эффективности работы программистов, например, количество кода, написанного за определенное время.

Для оценки качества программного кода были выбраны метрики сложности программ, которые представлены в таблице 4.2.

Таблица 4.2 – Метрики сложности программ

Название	Описание
Метрики Холстеда	Эта метрика помогает оценить сложность программы, анализируя количество уникальных слов и общее количество слов в коде. Она также использует словарь программы, чтобы определить, сколько слов нужно для написания программы.
Метрики МакКейба	Эта метрика оценивает сложность программы, подсчитывая количество базовых путей в управляющем графе. Базисные пути –это пути, которые можно пройти через программу, и маккейб предлагает использовать цикломатическую сложность как основную метрику, которая показывает, насколько сложно тестировать программу.
Метрики Спена	Эта метрика основана на анализе местоположения ссылок на данные в каждом разделе программы. Спен показывает, как часто в коде упоминается конкретный идентификатор, и если он упоминается много раз, это может затруднить процесс отладки и тестирования.
Метрики Чепина	Эта метрика направлена на оценку информационной прозрачности конкретного программного модуля. Она анализирует, как часто используются переменные из списка ввода–вывода, чтобы понять, насколько легко понять и тестировать этот модуль.

Из приведенного перечня метрик для оценки качества программного кода разрабатываемой системы была выбрана метрика Холстеда.

4.2 Тестирование приложения

4.2.1 Функциональное тестирование логики методом черного ящика

Функциональное тестирование логики методом черного ящика – это подход к тестированию программного обеспечения, при котором оценивается функциональность приложения без знания внутренней структуры и кода [24].

Для проверки логики приложения взаимодействия с объектами мониторинга и управления были составлены следующие тесты, представленные в таблице 4.3.

Таблица 4.3 – Функциональное тестирование логики приложения методом черного ящика

№	Название теста	Ожидаемый результат	Фактические результат
1	Добавления устройства измерения температуры со значениями от –30 до 100, находящаяся в доме номер 1 в комнате 1	Устройство добавится в систему с заданными параметрами. Пользователь получит сообщение об успешном завершение.	Устройство добавилось в систему с заданными параметрами. Пользователь получил сообщение об успешном завершение. Тест пройден Рисунок Д.1
3	Выключение устройства	Устройство выключится и будет хранить последнее полученное значение	Устройство выключилось и хранит последнее полученное значение Тест пройден Рисунок Д.2
4	Удаление устройства	Устройство удалится из системы полностью	Устройство удалилось из системы. Тест пройден Рисунок Д.3
5	Подключится к серверу хранения значений от устройств	Отобразится окно с текущими значениями	Отобразилось окно с текущими значениями. Тест пройден Рисунок Д.4
7	Просмотреть полученные значения измерения от устройств	Окно текущих значений будет динамически изменяться с учетом получения данных	Окно текущих значений динамически изменяется с учетом получения данных. Тест пройден Рисунок Д.5
8	Вставить в границы значений при создание устройства числовое значение	Приложение не даст вставить не числовое значение	Приложении не позволило вставить не числовое значения в границы устройства. Тест пройден Рисунок Д.6
8	Выключить выключенное устройство	Отобразится сообщения что устройство уже выключено	Отобразилось сообщение что выбранное устройство уже выключено. Тест пройден Рисунок Д.7
9	Удалить не существующее устройство	Отобразится сообщение что такого устройства не существует в системе	Отобразилось сообщение что выбранного устройства в систему не существует. Рисунок Д.8
	Подключится к несуществующему серверу	Отобразится сообщение что к серверу не удалось подключиться по причине его отсутствия	Отобразится сообщение что к серверу не удалось подключиться, по причине его отсутствия. Тест пройден Рисунок Д.9

Некоторые результаты тестирования представлены на рисунке 4.1 и рисунке 4.2.

Рисунок 4.1 – Тест «Вставить не числовое значение в поле границ числового датчика»

Рисунок 4.2 – Тест «Удалить не существующее устройство»

В ходе проведения тестирования логики ПС взаимодействия с устройствами мониторинга и управления, было выявлено что система работает корректно.

Все результаты тестирования отображены в приложение Г.

4.2.2 Функционально тестирование интерфейса методом черного ящика

Функциональное тестирование интерфейса методом черного ящика – это процесс проверки пользовательского интерфейса приложения без анализа внутренней структуры и кода. Основное внимание уделяется взаимодействию пользователя с интерфейсом, его функциональности и удобству.

Для тестирования интерфейса были проведены следующие тесты представленные в таблице 4.4.

Таблица 4.4 – Функциональное тестирования интерфейса приложения методом черного ящика

№	Название теста	Ожидаемый результат	Фактические результат
1	Переход на форму «Добавить устройство»	Главное окно скроется, а окно «Добавить устройство» отобразится	Главное окно скрылось, а окно «Добавить устройство» отобразилось. Тест пройден Рисунок Д.10
2	Переход на форму «редактировать устройство»	Главное окно скроется, а окно «редактировать устройство» отобразится	Главное окно скрылось, а окно «редактировать устройство» отобразилось Тест пройден Рисунок Д.11
3	Переход на форму «удалить устройство»	Главное окно скроется, а окно «удалить устройство» отобразится	Главное окно скрылось, а окно «удалить устройство» отобразилось Тест пройден Рисунок Д.12
4	Переход на форму «подключения к серверу»	Главное окно скроется, а окно «подключения к серверу» отобразится	Главное окно скрылось, а окно «подключения к серверу» отобразилось Тест пройден Рисунок Д.13
5	Переход на форму «просмотреть значения»	Главное окно скроется, а окно «просмотреть значения» отобразится	Главное окно скрылось, а окно «просмотреть значения» отобразилось Тест пройден Рисунок Д.14
6	Получения справки на главном окне	Отобразится сообщения со справкой о главном окне	Отобраилось сообщения со справкой о главном окне Тест пройден Рисунок Д.15
7	Нажатие кнопки «Отмена» со страницы «Добавить устройство»	Текущее окно закроется, откроется главное окно со списком текущих устройств	Текущее окно закрылось, открылось главное окно со списком текущих устройств Тест пройден Рисунок Д.16
8	Нажатие кнопки «Отмена» со страницы «редактировать устройство»	Текущее окно закроется, откроется главное окно со списком текущих устройств	Текущее окно закрылось, открылось главное окно со списком текущих устройств Тест пройден Рисунок Д.16

Продолжение таблицы 4.4

№	Название теста	Ожидаемый результат	Фактические результат
9	Нажатие кнопки «Отмена» со страницы «удалить устройство»	Текущее окно закроется, откроется главное окно со списком текущих устройств	Текущее окно закрылось, открылось главное окно со списком текущих устройств Тест пройден Рисунок Д.16
10	Нажатие кнопки «Отмена» со страницы «подключения к серверу»	Текущее окно закроется, откроется главное окно со списком текущих устройств	Текущее окно закрылось, открылось главное окно со списком текущих устройств Тест пройден Рисунок Д.16
11	Нажатие кнопки «Отмена» со страницы «просмотреть значения»	Текущее окно закроется, откроется главное окно со списком текущих устройств	Текущее окно закрылось, открылось главное окно со списком текущих устройств Тест пройден Рисунок Д.16

Некоторые результаты тестирования интерфейса представлены на рисунке 4.4 и рисунок 4.5.

Рисунок 4.3 – Тест «Переход на окно добавления устройства»

Рисунок 4.4 – Тест «Переход на окно подключения к серверу»

В ходе проведения тестирования интерфейса ПС взаимодействия с устройствами мониторинга и управления, было выявлено что система работает корректно.

Все результаты тестирования отображены в приложение Г.

4.3 Оценка качества программного кода

Для оценки качества кода была применена метрика Холстеда. Данная метрика позволяет оценить общее количество операторов и операндов, а также вычислить сложности программы на основе их уникальных значений.

Метрика помогает лучше понять сложность кода и его структуру, что в свою очередь может повлиять на дальнейшую его оптимизацию и поддержку.

Для тестирования был использован метод «GetTabIdAsync», предназначенный для поиска идентификатора потока по заданному имени. Код представлен в листинге 4.1.

Листинг 4.1 – Код метода GetTabIdAsync

```
public async Task<IActionResult> GetTabIdAsync(string
label)
{
    try
    {
        // Запрашиваем данные от Node-RED
        var response = await
_httpClient.GetStringAsync("http://127.0.0.1:1880/flows");

        // Преобразуем ответ в JSON
        var flows = JSONArray.Parse(response);

        // Ищем tab с нужной меткой
        foreach (var flow in flows)
        {
            if ((string)flow["type"] == "tab" &&
(string)flow["label"] == label)
            {
                var tabId = (string)flow["id"];
                return Ok(new { Id = tabId });
            }
        }

        return NotFound("Tab с указанной меткой не
найден.");
    }
}
```

```

        catch (HttpRequestException e)
        {
            return StatusCode(503, "Ошибка подключения к Node-
RED: " + e.Message);
        }
    }
}

```

Для оценки сложности необходимо подсчитать количество общих и уникальных операторов и операндов. Операторы представлены в таблице 4.5

Таблица 4.5 – Операторы метода GetTabIdAsync

Операторы	Число вхождений
public	1
async	1
Task	1
IActionResult	1
try	1
catch	1
await	1
return	2
if	1
foreach	1
var	2
=	4
==	2
&&	1
.	5
()	6
{}	4
[]	2
"	6
;	5

Операнды данного метода представлены в таблице 4.6.

Таблица 4.6 – Операнды метода GetTabIdAsync

Операнд	Число вхождений
GetTabIdAsync	1
label	2
_httpClient	1
GetStringAsync	1
"http://127.0.0.1:1880/flows"	1
response	2
flows	2
JArray.Parse	1
flow	3
"type"	1
"tab"	1

Продолжение таблицы 4.6

Операнд	Число вхождений
"label"	1
"id"	1
e	1
HttpRequestException	1
StatusCode	1
"Ошибка подключения к Node-RED: "	1
NotFound	1
"Tab с указанной меткой не найден."	1

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 42 + 22 = 64,$$

где $N1$ — общее число операторов;

$N2$ — общее число операндов.

Словарь программы вычисляется по формуле:

$$n = n1 + n2 = 16 + 18 = 34$$

где $n1$ — число уникальных операторов;

$n2$ — число уникальных операндов.

Объем программы рассчитывается по формуле:

$$HV = N * \log_2(n) = 64 * \log_2(34) = 326$$

где N — длина программы;

n — словарь программы.

Сложность программы определяется по формуле:

$$HD = (n1/2) * (N2/2) = (16/2) * (22/18) = 9.78$$

где $n1$ — число уникальных операторов,

$N2$ — общее число операндов,

$n2$ — число уникальных операндов.

Сложность процедуры приблизительно равна 9,78, поэтому её можно считать средней сложности.

Заключение

Целью данного дипломного проекта была разработка программных средств взаимодействия с объектами мониторинга и управления с использованием облачных технологий. На текущей стадии разработки созданы программные средства:

- симуляции устройств, которые имитируют работу реальных объектов мониторинга и позволяют взаимодействуют с облачными сервисами;
- управление устройствами, включая функции удаленного конфигурирования и мониторинга текущего состояния и управления действиями подключенных объектов;
- облачного взаимодействия, обеспечивающие обмен данными между облачными сервисами, включая обработку данных и хранение информации;
- визуализации и анализа данных, дающие возможность наблюдать за состоянием объектов в режиме реального времени.

В рамках дипломного проекта для создания программных средств было разработано приложение, позволяющее полностью использовать функционал ПС. Интерфейс был согласован с разработчиком приложений для взаимодействия с объектами мониторинга и управлений и разработан в едином стиле.

Приложение взаимодействия с объектами мониторинга и управления позволяет:

- просматривать имеющиеся устройства на сервере;
- добавлять новые устройства;
- редактировать устройства;
- удалять устройства;
- подключаться к необходимому серверу;
- просматривать текущие значения от устройств в реальном времени.

Полученное программное средство является функциональным завершением и может расширяться в направлениях функционала программных

средств. Планируется добавить функции создания сложных устройств управления.

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. Метамодель 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) Кучерявый А. Е. Интернет вещей как новая концепция развития сетей связи / А. Е. Кучерявый // Информационные технологии и коммуникации : электронный научный журнал. - 2014. - № 3. - С. 7-29.
- 6) Грингард С. Интернет вещей: будущее уже здесь / С. Грингард ; пер. с англ. - URL: https://litfond.net/r/146426?p_=98 (дата обращения: 01.05.2025).
- 7) Ли П. Архитектура интернета вещей / П. Ли ; пер. с англ. М. А. Райтмана. - М. : ДМК Пресс, 2019. - 454 с
- 8) Буч Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. - 2-е изд. - М. : ДМК Пресс, 2017. - 496 с
- 9) Леоненков А. В. Самоучитель UML 2 / А. В. Леоненков. - СПб. : БХВ-Петербург, 2007. - 576 с.
- 10) Адамс К. М. Нефункциональные требования к программному обеспечению / К. М. Адамс. - М. : Вильямс, 2014. - 448 с
- 11) Головач В. В. Дизайн пользовательского интерфейса / В. В. Головач. - СПб. : Usethics, 2000. - 146 с.
- 12) Саяпин О. В. Разработка интерфейсов прикладных программ: макетирование или прототипирование / О. В. Саяпин, О. В. Тиханычев, С. В. Чискидов, И. А. Быстракова // Прикладная информатика. - 2020. - Т. 85, № 1. - С. 49-56.
- 13) Ключев А. О. Аппаратные и программные средства встраиваемых систем : учебное пособие / А. О. Ключев, П. В. Кустарев, А. Е. Платунов. - СПб. : СПбГУ ИТМО, 2010. - 290 с.
- 14) Вигерс К. Разработка требований к программному обеспечению / К. Вигерс, Дж. Битти ; пер. с англ. - 3-е изд. - СПб. : Питер, 2014. - 736 с.
- 15) Магдануров Г. И. ASP.NET MVC Framework / Г. И. Магдануров, В. А. Юнев. - СПб. : БХВ-Петербург, 2010. - 320 с.
- 16) Бассет Л. JSON за 5 минут / Л. Бассет ; пер. с англ. - М. : Эксмо, 2020. - 160 с.

- 17) Лоре А. Проектирование веб-API / А. Лоре ; пер. с англ. Д. А. Беликова. - М. : ДМК Пресс, 2020. - 440 с.
- 18) Абрамян М. Э. User interface development based on Windows Forms class library : textbook / М. Е. Abramyan. - Rostov-on-Don : Southern Federal University Press, 2021. - 278 p.
- 19) Парфенова А. Ю. Стандартизация, сертификация и управление качеством программного обеспечения / А. Ю. Парфенова. - Самара : Изд-во Самарского университета, 2023. - 84 с.
- 20) Гаффорова Е. Б. Управление качеством : учебное пособие / Е. Б. Гаффорова, Т. Ю. Шкарина, Ж. С. Гаффорова, М. М. Кулеш. - Владивосток : Изд-во ДВГАЭУ, 2001. - 224 с.
- 21) Щербак А. В. Тестирование программного обеспечения : учебник для вузов / А. В. Щербак. - Москва : Юрайт, 2025. - 145 с.
- 22) Куликов С. В. Тестирование программного обеспечения. Базовый курс / С. В. Куликов. - Минск : Четыре четверти, 2020. - 300 с.
- 23) Проскуряков А. В. Качество и тестирование программного обеспечения. Метрология программного обеспечения : учебное пособие / А. В. Проскуряков. - Ростов-на-Дону : Изд-во Южного федерального университета, 2022. - 197 с.
- 24) Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения / Б. Бейзер ; пер. с англ. - М. : Лори, 2004. - 320 с.

ПРИЛОЖЕНИЕ А
(ОБЯЗАТЕЛЬНОЕ)

Глоссарий

ИУС – информационно–управляющие системы.

IoT (Internet of Things) – интернет вещей.

MQTT (Message Queuing Telemetry Transport) – протокол обмена данными для интернета вещей.

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста.

HTTPS ((HyperText Transfer Protocol Secure) – протокол защищённой передачи данных.

API (Application Programming Interface, программный интерфейс приложения) – это набор правил и способов, с помощью которых разные сервисы обмениваются друг с другом данными.

Yandex IoT Core – сервис для интернета вещей.

ПС – программные средства.

ПО – программное обеспечение.

Топик – канал отправки и получения сообщений.

UML (Unified Modeling Language) – языке моделирования.

REST (Representational State Transfer) – это архитектурный стиль для разработки веб-сервисов, который использует стандартные HTTP-методы для взаимодействия между клиентом и сервером.

Endpoint – конечная точка веб-сервиса, к которой клиентское приложение обращается для выполнения определённых операций или получения данных.

ПРИЛОЖЕНИЕ Б
(ОБЯЗАТЕЛЬНОЕ)
Шаблоны устройств

Листинг Б.1 – Шаблон node-red-template-int

```

{
  "id": "bbf6d7c6.937b78 {{DEVICE-ID}}",
  "label": "Device {{DEVICE-ID}}",
  "nodes": [
    {
      "id": "a8a052eb.958af {{DEVICE-ID}}",
      "type": "mqtt out",
      "name": "Device {{DEVICE-ID}}",
      "topic": "$registries/{{REGISTRY-ID}}/events",
      "qos": "1",
      "retain": "",
      "broker": "66c9fe0f.fa37 {{DEVICE-ID}}",
      "x": 610,
      "y": 140,
      "wires": []
    },
    {
      "id": "dd3eaa26.d18508 {{DEVICE-ID}}",
      "type": "inject",
      "name": "Send Data {{DEVICE-ID}}",
      "repeat": "3",
      "crontab": "",
      "once": false,
      "onceDelay": "",
      "topic": "",
      "payload": "true",
      "payloadType": "bool",
      "x": 110,
      "y": 140,
      "wires": [
        [
          "46feee67.7959b {{DEVICE-ID}}"
        ]
      ]
    },
    {
      "id": "46feee67.7959b {{DEVICE-ID}}",
      "type": "function",
      "name": "payload {{DEVICE-ID}}",
      "func": "// Указываем минимальное и максимальное
значения\nvar min = {{MIN}}; // минимальное значение\nvar max =
{{MAX}}; // максимальное значение\n\n// Генерируем случайное число
в заданном диапазоне\nvar speed1 = Math.floor(Math.random() * (max
- min + 1)) + min;\n\n// Create MQTT message in JSON\nmsg = {\n
payload: JSON.stringify(\n      {\n        d:\n          \"speed\" :
speed1\n      }\n    )\n};\nreturn msg;",
      "outputs": 1,
      "noerr": 0,
      "x": 360,
      "y": 140,
      "wires": [

```

```

        [
            "a8a052eb.958af {{DEVICE-ID}}"
        ]
    ]
}
],
"configs": [
    {
        "id": "66c9fe0f.fa37 {{DEVICE-ID}}",
        "type": "mqtt-broker",
        "name": "{{REGISTRY-ID}}",
        "broker": "mqtt.cloud.yandex.net",
        "port": "8883",
        "tls": "",
        "clientid": "",
        "usetls": true,
        "username": "{{DEVICE-ID}}",
        "password": "tESTbROKER1234",
        "compatmode": false,
        "keepalive": "60",
        "cleansession": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthPayload": "",
        "closeTopic": "",
        "closeQos": "0",
        "closePayload": "",
        "willTopic": "",
        "willQos": "0",
        "willPayload": ""
    }
]
}

```

Листинг Б.2 – Шаблон node-red-secondary-int

```

{
    "id": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
    "label": "House ROOM {{REGISTRY-ID}}",
    "nodes": [
        {
            "id": "78544752.ec5418 {{REGISTRY-ID}}",
            "type": "json",
            "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
            "name": "",
            "property": "payload",
            "action": "",
            "pretty": false,
            "x": 270,
            "y": 240,
            "wires": [
                [
                    "a8c0a13e.ee82a {{REGISTRY-ID}}"
                ]
            ]
        }
    ]
}

```

```

    ]
  ],
},
{
  "id": "a8c0a13e.ee82a {{REGISTRY-ID}}",
  "type": "function",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "name": "Parsing JSON {{REGISTRY-ID}}",
  "func": "\nvar speed = {}; \n\n\nspeed.payload =
msg.payload.d.speed\n\nreturn [speed];",
  "outputs": 1,
  "timeout": "",
  "noerr": 0,
  "initialize": "",
  "finalize": "",
  "libs": [],
  "x": 480,
  "y": 240,
  "wires": [
    [
      "a7e75a63.e4a438 {{REGISTRY-ID}}"
    ]
  ]
},
{
  "id": "a7e75a63.e4a438 {{REGISTRY-ID}}",
  "type": "ui_text",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "group": "265b61b2.7845de {{ROMM}} {{REGISTRY-ID}}",
  "order": 3,
  "width": 0,
  "height": 0,
  "name": "Значение {{DEVICE-NAME}}",
  "label": "Значение {{DEVICE-NAME}}",
  "format": "{{msg.payload}}",
  "layout": "row-spread",
  "x": 720,
  "y": 240,
  "wires": []
},
{
  "id": "838b8dd1.3c933 {{REGISTRY-ID}}",
  "type": "debug",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "name": "",
  "active": true,
  "tosidebar": true,
  "console": false,
  "tostatus": false,
  "complete": "true",
  "targetType": "full",
  "statusVal": "",
  "statusType": "auto",

```

```

        "x": 210,
        "y": 120,
        "wires": []
    },
    {
        "id": "100de22f.1d886e {{REGISTRY-ID}}",
        "type": "mqtt in",
        "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
        "name": "registry {{REGISTRY-ID}}",
        "topic": "$registries/{{REGISTRY-ID}}/state",
        "qos": "1",
        "datatype": "auto",
        "broker": "da49c30d.b7fb2 {{REGISTRY-ID}}",
        "nl": false,
        "rap": false,
        "inputs": 0,
        "x": 70,
        "y": 240,
        "wires": [
            [
                "838b8dd1.3c933 {{REGISTRY-ID}}",
                "78544752.ec5418 {{REGISTRY-ID}}"
            ]
        ]
    }
],
"configs": [
    {
        "id": "265b61b2.7845de {{ROMM}} {{REGISTRY-ID}}",
        "type": "ui_group",
        "name": "Комната {{ROMM}}",
        "tab": "9eb927d1.564038 {{REGISTRY-ID}}",
        "order": 3,
        "disp": true,
        "width": "5",
        "collapse": false,
        "className": ""
    },
    {
        "id": "da49c30d.b7fb2 {{REGISTRY-ID}}",
        "type": "mqtt-broker",
        "name": "iot-core-subscription {{REGISTRY-ID}}",
        "broker": "mqtt.cloud.yandex.net",
        "port": "8883",
        "tls": "",
        "clientId": "",
        "autoConnect": true,
        "usetls": true,
        "protocolVersion": 4,
        "keepalive": "60",
        "cleansession": true,
        "autoUnsubscribe": true,
        "birthTopic": "",

```

```

    "birthQos": "0",
    "birthPayload": "",
    "birthMsg": {},
    "closeTopic": "",
    "closeQos": "0",
    "closePayload": "",
    "closeMsg": {},
    "willTopic": "",
    "willQos": "0",
    "willPayload": "",
    "willMsg": {},
    "userProps": "",
    "sessionExpiry": ""
  },
  {
    "id": "9eb927d1.564038 {{REGISTRY-ID}}",
    "type": "ui_tab",
    "name": "{{Zdanie}}",
    "icon": "star",
    "order": 1,
    "disabled": false,
    "hidden": false
  }
]
}

```

Листинг Б.3 – Шаблон node-red-template-float

```

{
  "id": "bbf6d7c6.937b78 {{DEVICE-ID}}",
  "label": "Device {{DEVICE-ID}}",
  "nodes": [
    {
      "id": "a8a052eb.958af {{DEVICE-ID}}",
      "type": "mqtt out",
      "name": "Device {{DEVICE-ID}}",
      "topic": "$registries/{{REGISTRY-ID}}/events",
      "qos": "1",
      "retain": "",
      "broker": "66c9fe0f.fa37 {{DEVICE-ID}}",
      "x": 610,
      "y": 140,
      "wires": []
    },
    {
      "id": "dd3eaa26.d18508 {{DEVICE-ID}}",
      "type": "inject",
      "name": "Send Data {{DEVICE-ID}}",
      "repeat": "3",
      "crontab": "",
      "once": false,
      "onceDelay": "",

```

```

    "topic": "",
    "payload": "true",
    "payloadType": "bool",
    "x": 110,
    "y": 140,
    "wires": [
      [
        "46feee67.7959b {{DEVICE-ID}}"
      ]
    ]
  },
  {
    "id": "46feee67.7959b {{DEVICE-ID}}",
    "type": "function",
    "name": "payload {{DEVICE-ID}}",
    "func": "// Указываем минимальное и максимальное
значения\nvar min = {{MIN}}.0; // минимальное значение\nvar max =
{{MAX}}.0; // максимальное значение\n\n// Генерируем вещественное
число в заданном диапазоне\nvar speed1 = Math.random() * (max -
min) + min;\n\n// Округляем до 2 знаков после запятой
(опционально)\nspeed1 = parseFloat(speed1.toFixed(2));\n\n//
Create MQTT message in JSON\nmsg = {\n  payload: JSON.stringify(\n
{\n    d:{\n      \"speed\" : speed1\n    }\n  }\n
)\n};\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "x": 360,
    "y": 140,
    "wires": [
      [
        "a8a052eb.958af {{DEVICE-ID}}"
      ]
    ]
  }
],
"configs": [
  {
    "id": "66c9fe0f.fa37 {{DEVICE-ID}}",
    "type": "mqtt-broker",
    "name": "{{REGISTRY-ID}}",
    "broker": "mqtt.cloud.yandex.net",
    "port": "8883",
    "tls": "",
    "clientId": "",
    "usetls": true,
    "username": "{{DEVICE-ID}}",
    "password": "tESTbROKER1234",
    "compatmode": false,
    "keepalive": "60",
    "cleansession": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthPayload": "",

```

```

        "closeTopic": "",
        "closeQos": "0",
        "closePayload": "",
        "willTopic": "",
        "willQos": "0",
        "willPayload": ""
    }
}
]
}

```

Листинг Б.4 – Шаблон node-red- secondary –float

```

{
  "id": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "label": "House ROOM {{REGISTRY-ID}}",
  "nodes": [
    {
      "id": "78544752.ec5418 {{REGISTRY-ID}}",
      "type": "json",
      "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
      "name": "",
      "property": "payload",
      "action": "",
      "pretty": false,
      "x": 270,
      "y": 240,
      "wires": [
        [
          "a8c0a13e.ee82a {{REGISTRY-ID}}"
        ]
      ]
    },
    {
      "id": "a8c0a13e.ee82a {{REGISTRY-ID}}",
      "type": "function",
      "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
      "name": "Parsing JSON {{REGISTRY-ID}}",
      "func": "\nvar speed = {}; \n\nspeed.payload =\nmsg.payload.d.speed \n\nreturn [speed];",
      "outputs": 1,
      "timeout": "",
      "noerr": 0,
      "initialize": "",
      "finalize": "",
      "libs": [],
      "x": 480,
      "y": 240,
      "wires": [
        [
          "a7e75a63.e4a438 {{REGISTRY-ID}}"
        ]
      ]
    }
  ],
}

```

```

{
  "id": "a7e75a63.e4a438 {{REGISTRY-ID}}",
  "type": "ui_text",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "group": "265b61b2.7845de {{ROMM}}",
  "order": 3,
  "width": 0,
  "height": 0,
  "name": "Значение {{DEVICE-NAME}}",
  "label": "Значение {{DEVICE-NAME}}",
  "format": "{{msg.payload}}",
  "layout": "row-spread",
  "x": 720,
  "y": 240,
  "wires": []
},
{
  "id": "838b8dd1.3c933 {{REGISTRY-ID}}",
  "type": "debug",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "name": "",
  "active": true,
  "tosidebar": true,
  "console": false,
  "tostatus": false,
  "complete": "true",
  "targetType": "full",
  "statusVal": "",
  "statusType": "auto",
  "x": 210,
  "y": 120,
  "wires": []
},
{
  "id": "100de22f.1d886e {{REGISTRY-ID}}",
  "type": "mqtt in",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "name": "registry {{REGISTRY-ID}}",
  "topic": "$registries/{{REGISTRY-ID}}/state",
  "qos": "1",
  "datatype": "auto",
  "broker": "da49c30d.b7fb2 {{REGISTRY-ID}}",
  "nl": false,
  "rap": false,
  "inputs": 0,
  "x": 70,
  "y": 240,
  "wires": [
    [
      "838b8dd1.3c933 {{REGISTRY-ID}}",
      "78544752.ec5418 {{REGISTRY-ID}}"
    ]
  ]
}

```

```

    }
  ],
  "configs": [
    {
      "id": "265b61b2.7845de {{ROMM}}",
      "type": "ui_group",
      "name": "Комната {{ROMM}}",
      "tab": "9eb927d1.564038 {{REGISTRY-ID}}",
      "order": 3,
      "disp": true,
      "width": "5",
      "collapse": false,
      "className": ""
    },
    {
      "id": "da49c30d.b7fb2 {{REGISTRY-ID}}",
      "type": "mqtt-broker",
      "name": "iot-core-subscription {{REGISTRY-ID}}",
      "broker": "mqtt.cloud.yandex.net",
      "port": "8883",
      "tls": "",
      "clientId": "",
      "autoConnect": true,
      "usetls": true,
      "protocolVersion": 4,
      "keepalive": "60",
      "cleansession": true,
      "autoUnsubscribe": true,
      "birthTopic": "",
      "birthQos": "0",
      "birthPayload": "",
      "birthMsg": {},
      "closeTopic": "",
      "closeQos": "0",
      "closePayload": "",
      "closeMsg": {},
      "willTopic": "",
      "willQos": "0",
      "willPayload": "",
      "willMsg": {},
      "userProps": "",
      "sessionExpiry": ""
    },
    {
      "id": "9eb927d1.564038 {{REGISTRY-ID}}",
      "type": "ui_tab",
      "name": "{{Здание}}",
      "icon": "star",
      "order": 1,
      "disabled": false,
      "hidden": false
    }
  ]
}

```

```
}
```

Листинг Б.5 – Шаблон node-red-template-bool

```
{
  "id": "bbf6d7c6.937b78 {{DEVICE-ID}}",
  "label": "Device {{DEVICE-ID}}",
  "nodes": [
    {
      "id": "a8a052eb.958af {{DEVICE-ID}}",
      "type": "mqtt out",
      "name": "Device {{DEVICE-ID}}",
      "topic": "$registries/{{REGISTRY-ID}}/events",
      "qos": "1",
      "retain": "",
      "broker": "66c9fe0f.fa37 {{DEVICE-ID}}",
      "x": 610,
      "y": 140,
      "wires": []
    },
    {
      "id": "dd3eaa26.d18508 {{DEVICE-ID}}",
      "type": "inject",
      "name": "Send Data {{DEVICE-ID}}",
      "repeat": "3",
      "crontab": "",
      "once": false,
      "onceDelay": "",
      "topic": "",
      "payload": "true",
      "payloadType": "bool",
      "x": 110,
      "y": 140,
      "wires": [
        [
          "46feee67.7959b {{DEVICE-ID}}"
        ]
      ]
    },
    {
      "id": "46feee67.7959b {{DEVICE-ID}}",
      "type": "function",
      "name": "payload {{DEVICE-ID}}",
      "func": "var battery = ['true','false']\n\nvar counter1 =
context.get('counter1')||0;\nncounter1 = counter1+1;\n\nif(counter1 >
9) counter1 = 0;\n\ncontext.set('counter1',counter1);\n\nmsg = {\n
payload: JSON.stringify(\n      {\n        d:{\n          \"battery\" :
battery[counter1]\n        }\n      }\n    )\n};\n\nreturn msg;\n",
      "outputs": 1,
      "noerr": 0,
      "x": 360,
      "y": 140,
      "wires": [
```

```

        [
            "a8a052eb.958af {{DEVICE-ID}}"
        ]
    ]
}
],
"configs": [
    {
        "id": "66c9fe0f.fa37 {{DEVICE-ID}}",
        "type": "mqtt-broker",
        "name": "{{REGISTRY-ID}}",
        "broker": "mqtt.cloud.yandex.net",
        "port": "8883",
        "tls": "",
        "clientid": "",
        "usetls": true,
        "username": "{{DEVICE-ID}}",
        "password": "tESTbROKER1234",
        "compatmode": false,
        "keepalive": "60",
        "cleansession": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthPayload": "",
        "closeTopic": "",
        "closeQos": "0",
        "closePayload": "",
        "willTopic": "",
        "willQos": "0",
        "willPayload": ""
    }
]
}

```

Листинг Б.6 – Шаблон node-red- secondary -bool

```

{
    "id": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
    "label": "House ROOM {{REGISTRY-ID}}",
    "nodes": [
        {
            "id": "78544752.ec5418 {{REGISTRY-ID}}",
            "type": "json",
            "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
            "name": "",
            "property": "payload",
            "action": "",
            "pretty": false,
            "x": 270,
            "y": 240,
            "wires": [
                [
                    "a8c0a13e.ee82a {{REGISTRY-ID}}"
                ]
            ]
        }
    ]
}

```

```

    ]
  ],
},
{
  "id": "a8c0a13e.ee82a {{REGISTRY-ID}}",
  "type": "function",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "name": "Parsing JSON {{REGISTRY-ID}}",
  "func": "var battery = {};\\n\\nbattery.payload =
msg.payload.d.battery\\n\\nreturn [battery];",
  "outputs": 1,
  "timeout": "",
  "noerr": 0,
  "initialize": "",
  "finalize": "",
  "libs": [],
  "x": 480,
  "y": 240,
  "wires": [
    [
      "a7e75a63.e4a438 {{REGISTRY-ID}}"
    ]
  ]
},
{
  "id": "a7e75a63.e4a438 {{REGISTRY-ID}}",
  "type": "ui_text",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "group": "265b61b2.7845de {{ROMM}}",
  "order": 3,
  "width": 0,
  "height": 0,
  "name": "Значение {{DEVICE-NAME}}",
  "label": "Значение {{DEVICE-NAME}}",
  "format": "{{msg.payload}}",
  "layout": "row-spread",
  "x": 720,
  "y": 240,
  "wires": []
},
{
  "id": "838b8dd1.3c933 {{REGISTRY-ID}}",
  "type": "debug",
  "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
  "name": "",
  "active": true,
  "tosidebar": true,
  "console": false,
  "tostatus": false,
  "complete": "true",
  "targetType": "full",
  "statusVal": "",
  "statusType": "auto",

```

```

        "x": 210,
        "y": 120,
        "wires": []
    },
    {
        "id": "100de22f.1d886e {{REGISTRY-ID}}",
        "type": "mqtt in",
        "z": "55574a5c.e6e6e4 {{REGISTRY-ID}}",
        "name": "registry {{REGISTRY-ID}}",
        "topic": "$registries/{{REGISTRY-ID}}/state",
        "qos": "1",
        "datatype": "auto",
        "broker": "da49c30d.b7fb2 {{REGISTRY-ID}}",
        "nl": false,
        "rap": false,
        "inputs": 0,
        "x": 70,
        "y": 240,
        "wires": [
            [
                "838b8dd1.3c933 {{REGISTRY-ID}}",
                "78544752.ec5418 {{REGISTRY-ID}}"
            ]
        ]
    }
],
"configs": [
    {
        "id": "265b61b2.7845de {{ROMM}}",
        "type": "ui_group",
        "name": "Комната {{ROMM}}",
        "tab": "9eb927d1.564038 {{REGISTRY-ID}}",
        "order": 3,
        "disp": true,
        "width": "5",
        "collapse": false,
        "className": ""
    },
    {
        "id": "da49c30d.b7fb2 {{REGISTRY-ID}}",
        "type": "mqtt-broker",
        "name": "iot-core-subscription {{REGISTRY-ID}}",
        "broker": "mqtt.cloud.yandex.net",
        "port": "8883",
        "tls": "",
        "clientId": "",
        "autoConnect": true,
        "usetls": true,
        "protocolVersion": 4,
        "keepalive": "60",
        "cleansession": true,
        "autoUnsubscribe": true,
        "birthTopic": ""
    }
]

```

```

    "birthQos": "0",
    "birthPayload": "",
    "birthMsg": {},
    "closeTopic": "",
    "closeQos": "0",
    "closePayload": "",
    "closeMsg": {},
    "willTopic": "",
    "willQos": "0",
    "willPayload": "",
    "willMsg": {},
    "userProps": "",
    "sessionExpiry": ""
  },
  {
    "id": "9eb927d1.564038 {{REGISTRY-ID}}",
    "type": "ui_tab",
    "name": "{{Zdanie}}",
    "icon": "star",
    "order": 1,
    "disabled": false,
    "hidden": false
  }
]
}

```

ПРИЛОЖЕНИЕ В
(ОБЯЗАТЕЛЬНОЕ)
Листинг RESTful API

Листинг В.1 – Контроллер AddRegistrAndDevice.cs

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace Node_red_API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class RegistrationController : ControllerBase
    {
        private readonly HttpClient _httpClient;
        private readonly ILogger<RegistrationController> _logger;

        public RegistrationController(IHttpClientFactory
httpClientFactory, ILogger<RegistrationController> logger)
        {
            _httpClient = httpClientFactory.CreateClient();
            _logger = logger;
        }

        [HttpPost("AddRegistrANDDevice")]
        public async Task<IActionResult>
AddRegistrANDDevice(string registryName, string
registryDescription, string deviceName, string deviceDescription,
string labelsRegistr, string labelsDevice, string minValue, string
maxValue)
        {
            _logger.LogInformation("Starting AddRegistrANDDevice
with parameters: " +
                                $"RegistryName: {registryName}, "
+
                                $"RegistryDescription:
{registryDescription}, " +
                                $"DeviceName: {deviceName}, " +
                                $"DeviceDescription:
{deviceDescription}");

            var token = "t1.9euelZrLkpjHxsuLjY-
JyJecksuWyO3rnpWakJWKjpOejsycx82Jj8-Tz8nl9Pd7ZDQ9-
e8mGzDs3fT3OxMyPfnvJhsw7M3n9euelZrHkIqXkImcls2cy5ycip6Pku_8xeuelZr
HkIqXkImcls2cy5ycip6Pkg.EeTAm5Qg2Oz1ZHfRrUts51tuzjZ1V7DJP4TfA_jzSb
vhKDKQ9qpDE4_J0zk4PUT0q7h06RnuD-hEHOMrGsYKAA";

            try
            {
                // Создание реестра

```

```

var registryData = new
{
    name = registryName,
    description = registryDescription,
    folderId = "b1g913lffej8kphm0lm7",
    password = "tESTbROKER1234",
    labels = ParseLabels(labelsRegistr)
};

_logger.LogInformation("Creating registry with
data: {@RegistryData}", registryData);

var registryResponse = await
SendRequestAsync("https://iot-devices.api.cloud.yandex.net/iot-
devices/v1/registries", registryData, token);
if (!registryResponse.IsSuccessStatusCode)
{
    var errorContent = await
registryResponse.Content.ReadAsStringAsync();
    _logger.LogError("Failed to create registry.
StatusCode: {StatusCode}, Error: {Error}",
registryResponse.StatusCode,
errorContent);
    return BadRequest($"Не удалось создать реестр:
{errorContent}");
}

// Извлечение ID реестра из ответа
var registryId = await
ExtractRegistryIdAsync(registryResponse);
if (string.IsNullOrEmpty(registryId))
{
    _logger.LogError("Failed to extract registry
ID from response");
    return BadRequest("Не удалось извлечь ID
реестра");
}

_logger.LogInformation("Registry created
successfully. RegistryId: {RegistryId}", registryId);

// Создание устройства
var deviceData = new
{
    registryId = registryId,
    name = deviceName,
    password = "tESTbROKER1234",
    description = deviceDescription,
    labels = ParseLabels(labelsDevice)
};

_logger.LogInformation("Creating device with data:
{@DeviceData}", deviceData);

```

```

        var deviceResponse = await
SendRequestAsync("https://iot-devices.api.cloud.yandex.net/iot-
devices/v1/devices", deviceData, token);

        if (!deviceResponse.IsSuccessStatusCode)
        {
            var errorContent = await
deviceResponse.Content.ReadAsStringAsync();
            _logger.LogError("Failed to create device.
StatusCode: {StatusCode}, Error: {Error}",
deviceResponse.StatusCode,
errorContent);
            return BadRequest($"Не удалось создать
устройство: {errorContent}");
        }

        _logger.LogInformation("Device created
successfully for registry: {RegistryId}", registryId);

        // Извлечение ID реестра из ответа
        var deviceID = await
ExtractDeviceIdAsync(deviceResponse);
        if (string.IsNullOrEmpty(deviceID))
        {
            _logger.LogError("Failed to extract registry
ID from response");
            return BadRequest("Не удалось извлечь ID
реестра");
        }

        return Ok(new
        {
            Message = "Реестр и устройство успешно
созданы",
            RegistryName = registryName,
            RegistryId = registryId,
            DeviceName = deviceName,
            DeviceID = deviceID,
            LabelsReg = ParseLabels(labelsRegistr),
            LabelsDev = ParseLabels(labelsDevice),
            MinValue = minValue,
            MaxValue = maxValue
        });
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while
processing AddRegistrANDDevice");
        return
StatusCode(StatusCode.Status500InternalServerError, "Произошла
внутренняя ошибка сервера");
    }
}

```

```

        }
    }

    private async Task<HttpResponseMessage>
    SendRequestAsync(string url, object data, string token)
    {
        try
        {
            var jsonData = JsonConvert.SerializeObject(data);
            var content = new StringContent(jsonData,
            Encoding.UTF8, "application/json");

            _httpClient.DefaultRequestHeaders.Authorization =
            new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer",
            token);

            _logger.LogDebug("Sending request to {Url} with
            data: {Data}", url, jsonData);

            var response = await _httpClient.PostAsync(url,
            content);

            _logger.LogDebug("Received response from {Url}.
            StatusCode: {StatusCode}", url, response.StatusCode);

            return response;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error while sending request
            to {Url}", url);
            throw;
        }
    }

    private async Task<string>
    ExtractRegistryIdAsync(HttpResponseMessage response)
    {
        try
        {
            var content = await
            response.Content.ReadAsStringAsync();
            _logger.LogDebug("Extracting registry ID from
            response: {Content}", content);

            // Десериализация JSON-ответа
            var jsonResponse =
            JsonConvert.DeserializeObject<dynamic>(content);

            // Извлечение ID реестра из секции metadata
            var registryId =
            jsonResponse?.metadata?.registryId;

```

```

        return registryId?.ToString();
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error while extracting
registry ID");
        return null;
    }
}

private async Task<string>
ExtractDeviceIdAsync(HttpResponseMessage response)
{
    try
    {
        var content = await
response.Content.ReadAsStringAsync();
        _logger.LogDebug("Extracting device ID from
response: {Content}", content);

        // Десериализация JSON-ответа
        var jsonResponse =
JsonConvert.DeserializeObject<dynamic>(content);

        // Извлечение ID устройства из секции metadata
        var deviceId = jsonResponse?.metadata?.deviceId;

        return deviceId?.ToString();
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error while extracting
device ID");
        return null;
    }
}

private object ParseLabels(string labels)
{
    try
    {
        if (string.IsNullOrEmpty(labels))
        {
            _logger.LogDebug("No labels provided");
            return new
System.Collections.Generic.Dictionary<string, string>();
        }

        var labelsDictionary = new
System.Collections.Generic.Dictionary<string, string>();
        foreach (var label in labels.Split(','))
        {

```

```

        var parts = label.Split('=');
        if (parts.Length == 2)
        {
            labelsDictionary.Add(parts[0].Trim(),
parts[1].Trim());
        }
    }

    _logger.LogDebug("Parsed labels: {@Labels}",
labelsDictionary);
    return labelsDictionary;
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error while parsing labels:
{Labels}", labels);
    return new
System.Collections.Generic.Dictionary<string, string>();
}
}
}
}

```

Листинг В.2 – Контроллер ConvertToMainJsonController.cs

```

using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace Node_red_API.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class ConvertToMainJsonController : ControllerBase
    {
        private readonly ILogger<ConvertToMainJsonController>
_logger;

        public
ConvertToMainJsonController(ILogger<ConvertToMainJsonController>
logger)
        {
            _logger = logger;
        }

        [HttpPost]
        public IActionResult ConvertToMainJson([FromBody]
InputModel input, string NumberFile, string TypeDevice)
        {
            try
            {
                string filePath = null;
                if (TypeDevice == "int")

```

```

        {
            if (NumberFile == "1")
            {
                filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates", "int",
"node-red-template.json");
            }
            else if (NumberFile == "2")
            {
                filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates", "int",
"node-red-secondary.json");
            }
            else
            {
                return NotFound("JSON файл не найден.");
            }
        }
        else if (TypeDevice == "float")
        {
            if (NumberFile == "1")
            {
                filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates",
"float", "node-red-template.json");
            }
            else if (NumberFile == "2")
            {
                filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates",
"float", "node-red-secondary.json");
            }
            else
            {
                return NotFound("JSON файл не найден.");
            }
        }
        else if (TypeDevice == "bool")
        {
            if (NumberFile == "1")
            {
                filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates", "bool",
"node-red-template.json");
            }
            else if (NumberFile == "2")
            {
                filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates", "bool",
"node-red-secondary.json");
            }
            else
            {

```

```

        return NotFound("JSON файл не найден.");
    }
}
else
{
    return StatusCode(500, "Произошла ошибка
такого типа устройств нет");
}
// Проверка существования файла
if (!System.IO.File.Exists(filePath))
{
    return NotFound("JSON файл не найден.");
}

// Чтение JSON файла
var jsonData =
System.IO.File.ReadAllText(filePath);
JToken jsonObject = JToken.Parse(jsonData);

// Замена значений по плейсхолдерам
jsonData = jsonData
    .Replace("{{MESSAGE}}", input.Message)
    .Replace("{{REGISTRY-NAME}}",
input.RegistryName)
    .Replace("{{REGISTRY-ID}}", input.RegistryId)
    .Replace("{{Zdanie}}",
input.LabelsReg.Location)
    .Replace("{{ROMM}}", input.LabelsDev.Location)
    .Replace("{{DEVICE-NAME}}", input.DeviceName)
    .Replace("{{MIN}}", input.MinValue)
    .Replace("{{MAX}}", input.MaxValue)
    .Replace("{{DEVICE-ID}}", input.DeviceID);

jsonObject = JToken.Parse(jsonData);

// Обновление значения location в labels
if (jsonObject.Type == JTokenType.Object &&
jsonObject["labels"] != null)
{
    ((JObject)jsonObject)["labels"]["location"] =
input.LabelsDev.Location;
}

// Путь для нового JSON файла
string newFilePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates", "new-
node-red-template.json");

// Запись обновленного JSON в новый файл
System.IO.File.WriteAllText(newFilePath,
jsonObject.ToString());

// Возврат нового JSON файла

```

```

        return PhysicalFile(newFilePath,
"application/json", "new-node-red-template.json");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Ошибка при обновлении JSON
файла.");
        return StatusCode(500, "Произошла ошибка при
обновлении JSON файла.");
    }
}

public class InputModel
{
    public string Message { get; set; }
    public string RegistryName { get; set; }
    public string RegistryId { get; set; }
    public string DeviceName { get; set; }
    public string DeviceID { get; set; }

    public labelsRegistr LabelsReg { get; set; }
    public labelsDevice LabelsDev { get; set; }
    public string MinValue { get; set; }
    public string MaxValue { get; set; }
}

public class labelsRegistr
{
    public string Location { get; set; }
}

public class labelsDevice
{
    public string Work { get; set; }
    public string Location { get; set; }
}
}

```

Листинг В.3 – Контроллер DeletDevice.cs

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using System.Net.Http;

namespace Node_red_API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class DeletDevice : ControllerBase
    {

```

```

private readonly HttpClient _httpClient;

public DeletDevice(HttpClient httpClient)
{
    _httpClient = httpClient;
}

// DELETE api/devices/{deviceId}
[HttpDelete("{deviceId}")]
public async Task<IActionResult> DeleteDevice(string
deviceId)
{
    var deviceUrl = $"https://iot-
devices.api.cloud.yandex.net/iot-devices/v1/devices/{deviceId}";
    var iamToken = "t1.9euelZrLkpjHxsuLjY-
JyJecksuWy03rnpWakJWKjpOejsycx82Jj8-Tz8nl9Pd7ZDQ9-
e8mGzDs3fT3OxMyPfnvJhsw7M3n9euelZrHkIqXkImcls2cy5ycip6Pku_8xeuelZr
HkIqXkImcls2cy5ycip6Pkg.EeTAm5Qg2Oz1ZHfRrUts51tuzjZ1V7DJP4TfA_jzSb
vhKDKQ9qpDE4_J0zk4PUT0q7h06RnuD-hEHOMrGsYKAA";

    try
    {
        _httpClient.DefaultRequestHeaders.Authorization =
new AuthenticationHeaderValue("Bearer", iamToken);

        var response = await
_httpClient.DeleteAsync(deviceUrl);

        if (response.IsSuccessStatusCode)
        {
            return Ok("Device deleted successfully.");
        }
        else
        {
            return StatusCode((int)response.StatusCode,
"Failed to delete device.");
        }
    }
    catch (HttpRequestException ex)
    {
        return StatusCode(500, $"Internal server error:
{ex.Message}");
    }
}
}

```

Листинг В.4 – Контроллер DeletRegistr.cs

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Net.Http;
using System.Net.Http.Headers;

```

```

using System.Threading.Tasks;

namespace Node_red_API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class DeletRegistr : ControllerBase
    {
        private readonly IHttpClientFactory _httpClientFactory;

        public DeletRegistr(IHttpClientFactory httpClientFactory)
        {
            _httpClientFactory = httpClientFactory;
        }

        [HttpDelete("{registryId}")]
        public async Task<IActionResult> DeleteRegistry(string
registryId)
        {
            // Проверка на присутствие токена авторизации

            var requestUri = $"https://iot-
devices.api.cloud.yandex.net/iot-
devices/v1/registries/{registryId}";

            var client = _httpClientFactory.CreateClient();

            var request = new
HttpRequestMessage(HttpMethod.Delete, requestUri);
            request.Headers.Authorization = new
AuthenticationHeaderValue("Bearer", "t1.9euelZrLkpjHxsuLjY-
JyJecksuWyO3rnpWakJWKjpOejsycx82Jj8-Tz8nl9Pd7ZDQ9-
e8mGzDs3fT3OxMyPfnvJhsw7M3n9euelZrHkIqXkImcls2cy5ycip6Pku_8xeuelZr
HkIqXkImcls2cy5ycip6Pkg.EeTAm5Qg2Oz1ZHfRrUts5ltuzjZ1V7DJP4TfA_jzSb
vhKDKQ9qpDE4_J0zk4PUT0q7h06RnuD-hEHOMrGsYKAA");

            var response = await client.SendAsync(request);

            if (response.IsSuccessStatusCode)
            {
                return Ok("Registry deleted successfully.");
            }
            else
            {
                var errorDetails = await
response.Content.ReadAsStringAsync();
                return StatusCode((int)response.StatusCode,
$"Failed to delete registry. Status: {response.StatusCode},
Details: {errorDetails}");
            }
        }
    }
}

```

```
}
```

Листинг В.5 – Контроллер EditLabelDevice.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Net.Http.Headers;
using System.Text;
using System.Text.Json;

namespace Node_red_API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class DeviceController : ControllerBase
    {
        private readonly IHttpClientFactory _clientFactory;

        public DeviceController(IHttpClientFactory clientFactory)
        {
            _clientFactory = clientFactory;
        }

        [HttpPatch("{deviceId}")]
        public async Task<IActionResult> UpdateLabels(
            string deviceId,
            [FromBody] LabelUpdateRequest request)
        {
            if (string.IsNullOrEmpty(deviceId))
                return BadRequest("deviceId обязателен");
            if (request == null || request.Labels == null ||
                request.Labels.Count == 0)
                return BadRequest("Отсутствуют labels в запросе");

            var iamToken = "t1.9euelZrLkpjHxsuLjY-
JyJecksuWyO3rnpWakJWKjpOejsycx82Jj8-Tz8nl9Pd7ZDQ9-
e8mGzDs3fT3OxMyPfnvJhsw7M3n9euelZrHkIqXkImcls2cy5ycip6Pku_8xeuelZr
HkIqXkImcls2cy5ycip6Pkg.EeTAM5Qg2Oz1ZHfRrUts51tuzjZ1V7DJP4TfA_jzSb
vhKDKQ9qpDE4_J0zk4PUT0q7h06RnuD-hEHOMrGsYKAA";
            var ycEndpoint = $"https://iot-
devices.api.cloud.yandex.net/iot-devices/v1/devices/{deviceId}";
            var client = _clientFactory.CreateClient();

            var bodyObject = new
            {
                labels = request.Labels,
                updateMask = "labels"
            };

            var jsonBody = JsonSerializer.Serialize(bodyObject);
            var content = new StringContent(jsonBody,
                Encoding.UTF8, "application/json");
```

```

        var httpRequest = new
HttpRequestMessage(HttpMethod.Patch, ycEndpoint)
        {
            Content = content
        };
        httpRequest.Headers.Authorization = new
AuthenticationHeaderValue("Bearer", iamToken);

        HttpResponseMessage response;
        try
        {
            response = await client.SendAsync(httpRequest);
        }
        catch (HttpRequestException ex)
        {
            return StatusCode(503, $"Ошибка HTTP запроса:
{ex.Message}");
        }

        var result = await
response.Content.ReadAsStringAsync();

        if (!response.IsSuccessStatusCode)
            return StatusCode((int)response.StatusCode,
result);

        return Ok(result);
    }
}

public class LabelUpdateRequest
{
    public Dictionary<string, string> Labels { get; set; }
}
}

```

Листинг В.6 – Контроллер FindIDFlowController.cs

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json.Linq;

namespace Node_red_API.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class FindIDFlowController : ControllerBase
    {
        private readonly HttpClient _httpClient;

        public FindIDFlowController(HttpClient httpClient)
        {
            _httpClient = httpClient;
        }
    }
}

```

```

    }

    [HttpGet("{label}")]
    public async Task<IActionResult> GetTabIdAsync(string
label)
    {
        try
        {
            // Запрашиваем данные от Node-RED
            var response = await
_httpClient.GetStringAsync("http://127.0.0.1:1880/flows");

            // Преобразуем ответ в JSON
            var flows = JArray.Parse(response);

            // Ищем tab с нужной меткой
            foreach (var flow in flows)
            {
                if ((string)flow["type"] == "tab" &&
(string)flow["label"] == label)
                {
                    var tabId = (string)flow["id"];
                    return Ok(new { Id = tabId });
                }
            }

            return NotFound("Tab с указанной меткой не
найден.");
        }
        catch (HttpRequestException e)
        {
            return StatusCode(503, "Ошибка подключения к Node-
RED: " + e.Message);
        }
    }
}

```

Листинг В.7 – Контроллер NodeRedFlowController.cs

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Text;

namespace Node_red_API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class NodeRedFlowController : ControllerBase
    {

```

```

private readonly HttpClient _httpClient;

public NodeRedFlowController(HttpClient httpClient)
{
    _httpClient = httpClient;
}

private async Task<dynamic> GetFlow(string flowId)
{
    var url = $"http://localhost:1880/flow/{flowId}";

    // Выполнение GET-запроса
    var response = await _httpClient.GetAsync(url);

    if (response.IsSuccessStatusCode)
    {
        var responseData = await
response.Content.ReadAsStringAsync();
        // Сериализация JSON в динамический объект
        return
JsonConvert.DeserializeObject<dynamic>(responseData);
    }

    throw new Exception($"Error retrieving flow:
{response.StatusCode}");
}

[HttpPut("disableFlow")]
public async Task<IActionResult> DisableFlow([FromQuery]
string flowId)
{
    // Получение объекта потока
    var flow = await GetFlow(flowId);

    // Изменение параметра disabled на true
    flow["disabled"] = true;

    var url = $"http://localhost:1880/flow/{flowId}";

    // Сериализация обновленного объекта обратно в JSON
    var json = JsonConvert.SerializeObject(flow);
    var content = new StringContent(json, Encoding.UTF8,
"application/json");

    // Отправка PUT-запроса
    var response = await _httpClient.PutAsync(url,
content);

    if (response.IsSuccessStatusCode)
    {
        return Ok("Flow disabled successfully.");
    }
}

```

```

        // Если ошибка - возвращаем её описание
        return StatusCode((int)response.StatusCode, await
response.Content.ReadAsStringAsync());
    }

    [HttpPut("enableFlow")]
    public async Task<IActionResult> EnableFlow([FromQuery]
string flowId)
    {
        // Получение объекта потока
        var flow = await GetFlow(flowId);

        // Изменение параметра disabled на false
        flow["disabled"] = false;

        var url = $"http://localhost:1880/flow/{flowId}";

        // Сериализация объекта в JSON
        var json = JsonConvert.SerializeObject(flow);
        var content = new StringContent(json, Encoding.UTF8,
"application/json");

        // Отправка PUT-запроса
        var response = await _httpClient.PutAsync(url,
content);

        if (response.IsSuccessStatusCode)
        {
            return Ok("Flow enabled successfully.");
        }

        // Если ошибка - возвращаем её описание
        return StatusCode((int)response.StatusCode, await
response.Content.ReadAsStringAsync());
    }

    // Метод для добавления нового потока
    [HttpPost("addFlowFromTemplate")]
    public async Task<IActionResult> AddFlowFromTemplate()
    {
        // Путь для нового JSON файла
        string newFilePath =
Path.Combine(Directory.GetCurrentDirectory(), "Templates", "new-
node-red-template.json");

        if (!System.IO.File.Exists(newFilePath))
        {
            return NotFound("JSON файл не найден.");
        }

        string flowJsonContent;

        // Чтение JSON из файла

```

```

        try
        {
            flowJsonContent = await
System.IO.File.ReadAllTextAsync(newFilePath);
        }
        catch (Exception ex)
        {
            return
StatusCode(StatusCode.Status500InternalServerError, $"Ошибка при
чтении файла: {ex.Message}");
        }

        var url = "http://localhost:1880/flow";

        // Подготовка контента для отправки
        var content = new StringContent(flowJsonContent,
Encoding.UTF8, "application/json");

        // Отправка POST-запроса для добавления нового потока
        var response = await _httpClient.PostAsync(url,
content);

        if (response.IsSuccessStatusCode)
        {
            // Получение ответа от Node-RED
            var responseData = await
response.Content.ReadAsStringAsync();
            return
Ok(JsonConvert.DeserializeObject<dynamic>(responseData));
        }

        // Если ошибка - возвращаем её описание
        return StatusCode((int)response.StatusCode, await
response.Content.ReadAsStringAsync());
    }

    // Метод для удаления потока
    [HttpDelete("deleteFlow/{flowId}")]
    public async Task<IActionResult> DeleteFlow(string flowId)
    {
        // Проверяем, что flowId не пустой
        if (string.IsNullOrEmpty(flowId))
        {
            return BadRequest("Идентификатор потока не может
быть пустым.");
        }

        var url = $"http://localhost:1880/flow/{flowId}";

        // Отправка DELETE-запроса для удаления потока
        var response = await _httpClient.DeleteAsync(url);

        if (response.IsSuccessStatusCode)

```

```

        {
            // Получение ответа от Node-RED
            var responseData = await
response.Content.ReadAsStringAsync();
            return
Ok(JsonConvert.DeserializeObject<dynamic>(responseData));
        }

        // Если ошибка - возвращаем её описание
        return StatusCode((int)response.StatusCode, await
response.Content.ReadAsStringAsync());
    }
}

```

Листинг В.8 – Контроллер RegistrAndDeviceController.cs

```

using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System.Collections.Generic;
using System.Net.Http;
using System.Security.AccessControl;
using System.Threading.Tasks;

namespace Node_red_API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class RegistrAndDeviceController : ControllerBase
    {
        private readonly HttpClient _httpClient;
        private readonly string _iotUrl = "https://iot-
devices.api.cloud.yandex.net/iot-devices/v1/";
        private readonly string _folderId =
"b1g913lffej8kphm0lm7";
        private readonly string _iamToken =
"t1.9euelZrLkpjHxsuLjY-JyJecksuWyO3rnpWakJWKjpOejsycx82Jj8-
Tz8nl9Pd7ZDQ9-
e8mGzDs3fT3OxMyPfnvJhsw7M3n9euelZrHkIqXkImcls2cy5ycip6Pku_8xeuelZr
HkIqXkImcls2cy5ycip6Pkg.EeTAm5Qg2Oz1ZHfRrUts51tuzjZ1V7DJP4TfA_jzSb
vhKDKQ9qpDE4_J0zk4PUT0q7h06RnuD-hEHOMrGsYKAA";
        public RegistrAndDeviceController()
        {
            _httpClient = new HttpClient();
        }

        [HttpGet]
        public async Task<IActionResult> GetRegistriesAndDevices()
        {
            _httpClient.DefaultRequestHeaders.Authorization = new
System.Net.Http.Headers.AuthenticationHeaderValue("Bearer",
_iamToken);

```

```

try
{
    // Запрашиваем все регистры
    var registriesResponse = await
_httpClient.GetAsync($"{_iotUrl}registries?folderId={_folderId}");
    registriesResponse.EnsureSuccessStatusCode();
    var registriesJson = await
registriesResponse.Content.ReadAsStringAsync();
    dynamic registries =
JsonConvert.DeserializeObject(registriesJson);

    var result = new List<object>();

    if (registries?.registries != null)
    {
        foreach (var registry in
registries.registries)
        {
            if (registry.id != null && registry.name
!= null)
            {
                // Создаем список меток с их
значениями для регистра
                var registryLabels = new
Dictionary<string, string>();

                if (registry.labels != null)
                {
                    foreach (JProperty label in
registry.labels)
                    {
                        string key = label.Name;
                        string value =
label.Value?.ToString() ?? string.Empty;
                        registryLabels.Add(key,
value);
                    }
                }

                var registryId =
registry.id?.ToString();
                var registryName =
registry.name?.ToString();
                var registryState =
registry.status?.ToString() ?? "unknown";

                // Запрашиваем все устройства для
текущего регистра
                var devicesResponse = await
_httpClient.GetAsync($"{_iotUrl}devices?registryId={registryId}");
                devicesResponse.EnsureSuccessStatusCode();
            }
        }
    }
}

```

```

        var devicesJson = await
devicesResponse.Content.ReadAsStringAsync();
        dynamic devices =
JsonConvert.DeserializeObject(devicesJson);

        var deviceList = new List<object>();

        if (devices?.devices != null)
        {
            foreach (var device in
devices.devices)
            {
                if (device.id != null &&
device.name != null)
                {
                    // Создаем список меток с
их значениями для устройства
Dictionary<string, string>();

                    if (device.labels != null)
                    {
                        foreach (JProperty
label in device.labels)
                        {
                            string key =
label.Name;
                            string value =
label.Value?.ToString() ?? string.Empty;
                            deviceLabels.Add(key, value);
                        }
                    }

                    deviceList.Add(new
                    {
                        DeviceId =
device.id?.ToString(),
                        DeviceName =
device.name?.ToString(),
                        DeviceDescription =
device.description?.ToString(),
                        Labels = deviceLabels
                    });
                }
            }

            result.Add(new
            {
                RegistryId = registryId,
                RegistryName = registryName,
                RegistryState = registryState,

```

```

                                Labels = registryLabels, //
Переименовано для единообразия
                                Devices = deviceList
                                });
                                }
                                }
                                }

                                return Ok(result);
                                }
                                catch (HttpRequestException e)
                                {
                                    return
StatusCode(StatusCode.Status500InternalServerError, $"Произошла
ошибка при выполнении запроса: {e.Message}");
                                }
                                }
                                }
                                }

```

Листинг В.9 – Контроллер RegistryStatusController.cs

```

using Microsoft.AspNetCore.Mvc;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Newtonsoft.Json;

namespace Node_red_API.Controllers
{
    [Route("api/registries")]
    [ApiController]
    public class RegistryStatusController : ControllerBase
    {
        private readonly IHttpClientFactory _httpClientFactory;
        private const string YandexCloudIotApiUrl = "https://iot-
devices.api.cloud.yandex.net/iot-devices/v1/registries";
        private const string NodeRedApiUrl =
"http://localhost:1880"; // Update with your Node-RED URL

        public RegistryStatusController(IHttpClientFactory
httpClientFactory)
        {
            _httpClientFactory = httpClientFactory;
        }

        /// <summary>
        /// Активирует реестр устройств
        /// </summary>
        [HttpPost("{registryId}/activate")]
        public async Task<IActionResult> ActivateRegistry(string
registryId)
        {

```

```

        return await ChangeRegistryStatus(registryId,
"enable");
    }

    /// <summary>
    /// Деактивирует реестр устройств
    /// </summary>
    [HttpPost("{registryId}/disable")]
    public async Task<IActionResult> DisableRegistry(string
registryId)
    {
        return await ChangeRegistryStatus(registryId,
"disable");
    }

    private async Task<IActionResult>
ChangeRegistryStatus(string registryId, string action)
    {
        if (string.IsNullOrEmpty(registryId))
            return BadRequest("Registry ID is required");

        try
        {
            var client = _httpClientFactory.CreateClient();
            client.DefaultRequestHeaders.Authorization =
                new AuthenticationHeaderValue("Bearer",
"t1.9euelZrLkpjHxsuLjY-JyJecksuWyO3rnpWakJWKjpOejsycx82Jj8-
Tz8nl9Pd7ZDQ9-
e8mGzDs3fT3OxMyPfnvJhsw7M3n9euelZrHkIqXkImcls2cy5ycip6Pku_8xeuelZr
HkIqXkImcls2cy5ycip6Pkg.EeTAm5Qg2Oz1ZHfRrUts51tuzjZ1V7DJP4TfA_jzSb
vhKDKQ9qpDE4_J0zk4PUT0q7h06RnuD-hEHOMrGsYKAA");

            var requestUrl =
$"{{YandexCloudIotApiUrl}}/{registryId}:{action}";

            // Для POST запроса с пустым телом
            var response = await client.PostAsync(requestUrl,
null);

            if (!response.IsSuccessStatusCode)
            {
                var errorContent = await
response.Content.ReadAsStringAsync();
                return StatusCode((int)response.StatusCode,
$"Yandex Cloud API error:
{errorContent}");
            }

            return Ok(await
response.Content.ReadAsStringAsync());
        }
    }

```

```
        catch (HttpRequestException ex)
        {
            return StatusCode(500, $"Error connecting to
Yandex Cloud API: {ex.Message}");
        }
    }
}
```

ПРИЛОЖЕНИЕ Г
(ОБЯЗАТЕЛЬНОЕ)

Листинг клиентского приложения

Листинг Г.1 – Форма AddDevice.cs

```

using Newtonsoft.Json;
using node_red_forms_two.Properties;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Reflection.Emit;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using static node_red_forms_two.Form1;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace node_red_forms_two
{
    public partial class AddDevice : Form
    {
        private Form _Form1;
        public List<Registry> Registries { get; set; }
        public List<Device> Devices { get; set; }
        private ApiResponse _lastApiResponse;
        public AddDevice(Form Form1, List<Registry> registries,
List<Device> devices)
        {
            InitializeComponent();
            _Form1 = Form1;
            this.Registries = registries; // Сохраняем в свойствах
формы
            this.Devices = devices;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            _Form1.Show();
            this.Close();
        }

        //.....УПРАВЛЕНИЕ
РАСПОЛОЖЕНИЕМ.....//
        private void radioButton1_CheckedChanged(object sender,
EventArgs e)
        {
            if (radioButton1.Checked)
            {
                comboBox1.Visible = false;
                comboBox2.Visible = false;
            }
        }
    }
}

```

```

        textBox2.Visible = true;
        textBox3.Visible = true;
    }
}

private void radioButton2_CheckedChanged(object sender,
EventArgs e)
{
    if (radioButton2.Checked)
    {
        comboBox1.Visible = true;
        comboBox2.Visible = false;
        textBox2.Visible = false;
        textBox3.Visible = true;
    }
}

private void radioButton3_CheckedChanged(object sender,
EventArgs e)
{
    if (radioButton3.Checked)
    {
        comboBox1.Visible = true;
        comboBox2.Visible = true;
        textBox2.Visible = false;
        textBox3.Visible = false;
    }
}

//.....ЗАГРУЗКА
ФОРМЫ.....//

private void AddDevice_Load(object sender, EventArgs e)
{
    //ЗАПОЛНЕНИЕ COMBOBOX
    var comboBoxItemsDevice = new List<(string Item,
string DataType)>
    {
        ("Счетчик посетителей", "int"),
        ("Датчик температуры упрощенный", "int"),
        ("Датчик влажности упрощенный", "int"),
        ("Датчик температуры", "float"),
        ("Датчик освещенности", "float"),
        ("Датчик влажности", "float"),
        ("Датчик газа", "float"),
        ("Датчик света", "float"),
        ("Лампочка", "bool"),
        ("Выключатель", "bool"),
        ("Датчик протечки воды", "bool")
    };

    foreach (var (item, dataType) in comboBoxItemsDevice)
    {

```

```

        comboBox3.Items.Add($"{item}");
    }

    // Получаем все уникальные значения location из всех
устройств
    var locations = Registries
        .Where(Registries => Registries.Labels != null &&
Registries.Labels.ContainsKey("location"))
        .Select(Registries =>
Registries.Labels["location"])
        .Distinct()
        .OrderBy(loc => loc)
        .ToList();

    // Очищаем комбобокс и добавляем значения
comboBox1.Items.Clear();
comboBox1.Items.AddRange(locations.ToArray());

    // Если нужно добавить пустое значение (опционально)
comboBox1.Items.Insert(0, "");
comboBox1.SelectedIndex = 0;
}

private void comboBox3_SelectedIndexChanged(object sender,
EventArgs e)
{
    // Получаем выбранный элемент из ComboBox
    string selectedItem =
comboBox3.SelectedItem?.ToString();

    // Сопоставляем выбранный элемент с его типом данных
var comboBoxItems = new List<(string Item, string
DataType)>
{
    ("Счетчик посетителей", "int"),
    ("Датчик температуры упрощенный", "int"),
    ("Датчик влажности упрощенный", "int"),
    ("Датчик температуры", "float"),
    ("Датчик освещенности", "float"),
    ("Датчик влажности", "float"),
    ("Датчик газа", "float"),
    ("Датчик света", "float"),
    ("Лампочка", "bool"),
    ("Выключатель", "bool"),
    ("Датчик протечки воды", "bool")
};

    // Ищем выбранный элемент в списке и получаем его тип
var selectedSensor = comboBoxItems.FirstOrDefault(x =>
x.Item == selectedItem);

    if (selectedSensor.Item == null)
    {

```

```

        MessageBox.Show("Неизвестный датчик!", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    string sensorType = selectedSensor.DataType;
    if (sensorType == "int" || sensorType == "float")
    {
        groupBox3.Visible = true;
    }
    else
    {
        groupBox3.Visible = false;
    }
}

private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
{
    comboBox2.Items.Clear();

    // 1. Получаем выбранное здание (location из регистра)
    string selectedBuilding =
comboBox1.SelectedItem?.ToString();
    if (string.IsNullOrEmpty(selectedBuilding))
        return;

    // 2. Находим все регистры с этим зданием
    var registriesInBuilding = Registries
        .Where(r => r.Labels != null &&
            r.Labels.TryGetValue("location", out
var loc) &&
                loc == selectedBuilding)
        .ToList();

    // 3. Собираем ВСЕ устройства из этих регистров
    var allDevicesInBuilding = registriesInBuilding
        .SelectMany(r => r.Devices) // "Разворачиваем"
    списки устройств
        .ToList();

    // 4. Извлекаем уникальные комнаты (метка "room")
    var uniqueRooms = allDevicesInBuilding
        .Where(d => d.Labels != null &&
            d.Labels.TryGetValue("location", out
var _))
        .Select(d => d.Labels["location"])
        .Distinct()
        .OrderBy(room => room)
        .ToList();

    // 5. Заполняем comboBox3
    comboBox2.Items.AddRange(uniqueRooms.ToArray());
}

```

```

//кнопка создать
private async void button3_Click(object sender, EventArgs
e)
{
    progressBar1.Visible = true;
    labelStatus.Visible = true;
    //ЗАПОЛНЕНИЕ COMBOBOX
    var comboBoxItemsDevice = new List<(string Item,
string DataType)>
    {
        ("Счетчик посетителей", "int"),
        ("Датчик температуры упрощенный", "int"),
        ("Датчик влажности упрощенный", "int"),
        ("Датчик температуры", "float"),
        ("Датчик освещенности", "float"),
        ("Датчик влажности", "float"),
        ("Датчик газа", "float"),
        ("Датчик света", "float"),
        ("Лампочка", "bool"),
        ("Выключатель", "bool"),
        ("Датчик протечки воды", "bool")
    };

    string House = null;
    string Room = null;
    string TypeDevice = null;
    string selectedName =
comboBox3.SelectedItem?.ToString();

    if (!string.IsNullOrEmpty(selectedName))
    {
        var deviceInfo =
comboBoxItemsDevice.FirstOrDefault(x => x.Item == selectedName);
        if (deviceInfo != default)
        {
            TypeDevice = deviceInfo.DataType;
            // Используйте TypeDevice по вашему усмотрению
        }
    }

    string NameDevice = textBox1.Text;
    string DescriptionDevice = richTextBox1.Text;
    string Min = "0";
    string Max = "0";
    string StateDevice =
checkBox1.Checked.ToString().ToLower(); ;
    if (TypeDevice != "bool")
    {
        Min = numericUpDown1.MIN.Value.ToString();
        Max = numericUpDown2.MAX.Value.ToString();
    }
}

```

```

        if (radioButton1.Checked)
        {
            House = textBox2.Text;
            Room = textBox3.Text;
        }
        else if (radioButton2.Checked)
        {
            House = comboBox1.SelectedItem.ToString();
            Room = textBox3.Text;
        }
        else if (radioButton3.Checked)
        {
            House = comboBox1.SelectedItem.ToString();
            Room = comboBox2.SelectedItem.ToString();
        }
        string DescriptionReg = $"Устройство в доме {House} в
комнате {Room} типа {TypeDevice}";
        string RegistrName = $"{House}z{Room}z{NameDevice}";

        MessageBox.Show($"registrName {RegistrName}\n" +
            $"RegistrDescription {DescriptionReg} \n" +
            $"deviceName {NameDevice}\n" +
            $"deviceDescription {DescriptionDevice}\n" +
            $"labelsreg = location={House}\n" +
            $"labelDev = work={StateDevice},location={Room}\n"
+
            $"minValue {Min}\n" +
            $"maxValue {Max}");

        string labelsRegistr = $"location={House}";
        string labelsDevice =
        $"work={StateDevice},location={Room}";

        // Формирование URL
        var url =
        $"https://localhost:7101/api/Registration/AddRegistrANDDevice" +
        $"?registryName={Uri.EscapeDataString(RegistrName)}" +
        $"&registryDescription={Uri.EscapeDataString(DescriptionReg)}" +
        $"&deviceName={Uri.EscapeDataString(NameDevice)}" +
        $"&deviceDescription={Uri.EscapeDataString(DescriptionDevice)}" +
        $"&labelsRegistr={Uri.EscapeDataString(labelsRegistr)}" +
        $"&labelsDevice={Uri.EscapeDataString(labelsDevice)}" +
            $"&minValue={Min}" +
            $"&maxValue={Max}";

        using (var client = new HttpClient())
        {

```

```

        client.DefaultRequestHeaders.Add("accept", "*/*");
        try
        {
            // 1. Первый API вызов
            progressBar1.Value = 1;
            progressBar1.Update();
            labelStatus.Text = "Создание устройства в
Yandex Cloud...";

            var response = await client.PostAsync(url,
null);

            if (response.IsSuccessStatusCode)
            {
                string responseContent = await
response.Content.ReadAsStringAsync();
                _lastApiResponse =
JsonConvert.DeserializeObject<ApiResponse>(responseContent);

                // 2. Второй API вызов
                progressBar1.Value = 2;
                progressBar1.Update();
                labelStatus.Text = "Конвертация в основной
JSON (шаг 1)...";

                await
CallConvertToMainJsonApi(_lastApiResponse, TypeDevice, 1);

                progressBar1.Value = 3;
                progressBar1.Update();
                labelStatus.Text = "Загрузка в Node-RED
(шаг 1)...";

                HttpResponseMessage nodeRedResponse1 =
await
client.PostAsync("https://localhost:7101/api/NodeRedFlow/addFlowFr
omTemplate", null);

                if (nodeRedResponse1.IsSuccessStatusCode)
                {
                    string nodeRedContent1 = await
nodeRedResponse1.Content.ReadAsStringAsync();
                    MessageBox.Show("Загрузка в Node-RED
успешна: " + nodeRedContent1, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                }
                else
                {
                    MessageBox.Show($"Ошибка загрузки в
Node-RED: {nodeRedResponse1.StatusCode}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
    }
}

```

```

        // 3. Третий API вызов
        progressBar1.Value = 4;
        progressBar1.Update();
        labelStatus.Text = "Конвертация в основной
JSON (шаг 2)...";

        await
        CallConvertToMainJsonApi(_lastApiResponse, TypeDevice, 2);

        progressBar1.Value = 5;
        progressBar1.Update();
        labelStatus.Text = "Загрузка в Node-RED
(шаг 2)...";

        HttpResponseMessage nodeRedResponse2 =
await
client.PostAsync("https://localhost:7101/api/NodeRedFlow/addFlowFromTemplate", null);

        if (nodeRedResponse2.IsSuccessStatusCode)
        {
            string nodeRedContent2 = await
nodeRedResponse2.Content.ReadAsStringAsync();
            MessageBox.Show("Загрузка в Node-RED
успешна: " + nodeRedContent2, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show($"Ошибка загрузки в
Node-RED: {nodeRedResponse2.StatusCode}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    else
    {
        Console.WriteLine($"Ошибка:
{response.StatusCode} - {response.ReasonPhrase}");
    }
}
catch (Exception ex)
{
    MessageBox.Show($"Ошибка: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
finally
{
    progressBar1.Value = progressBar1.Maximum;
    labelStatus.Text = "Готово!";
    await Task.Delay(1000); // Даем пользователю
увидеть завершение
    progressBar1.Visible = false;
    labelStatus.Visible = false;
}

```

```

        }
    }
}

private async Task CallConvertToMainJsonApi(ApiResponse
apiResponse, string deviceType, int numberFile)
{
    try
    {
        using (HttpClient client = new HttpClient())
        {
            // URL для второго API
            string convertApiUrl =
$"https://localhost:7101/ConvertToMainJson?NumberFile={numberFile}
&TypeDevice={deviceType}";

            // Настройка заголовков
            client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("*/*"));

            // Сериализуем тело запроса (ответ первого
API)
            string jsonContent =
JsonConvert.SerializeObject(apiResponse);
            var httpContent = new
StringContent(jsonContent, Encoding.UTF8, "application/json");

            // Отправляем POST-запрос
            HttpResponseMessage response = await
client.PostAsync(convertApiUrl, httpContent);

            if (response.IsSuccessStatusCode)
            {
                string responseContent = await
response.Content.ReadAsStringAsync();
                MessageBox.Show($"Второй API успешно
выполнен!\n", "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show($"Ошибка второго API:
{response.StatusCode}\n{await
response.Content.ReadAsStringAsync()}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при вызове второго API:
{ex.Message}", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}

```

```

    }
    public class ApiResponse
    {
        public string Message { get; set; }
        public string RegistryName { get; set; }
        public string RegistryId { get; set; }
        public string DeviceName { get; set; }
        public string DeviceID { get; set; }
        public Labels LabelsReg { get; set; }
        public Labels LabelsDev { get; set; }
        public string MinValue { get; set; }
        public string MaxValue { get; set; }
    }

    public class Labels
    {
        public string Location { get; set; }
        public string Work { get; set; } // Только для
LabelsDev
    }
}

```

Листинг Г.2 – Форма ConnectServer.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace node_red_forms_two
{
    public partial class ConnectServer : Form
    {
        private Form _Form1;
        public ConnectServer(Form Form1)
        {
            InitializeComponent();
            _Form1 = Form1;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            _Form1.Show();
            this.Close();
        }
    }
}

```

```

        private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (comboBox1.SelectedIndex == 1)
            {
                textBox1.Visible = true;
            } else
            {
                textBox1.Visible = false;
            }
        }

        private void ConnectServer_Load(object sender, EventArgs
e)
        {
        }
    }
}

```

Листинг Г.3 – Форма DeleteServer.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static node_red_forms_two.Form1;
using static
System.Windows.Forms.VisualStyles.VisualStyleElement.Button;

namespace node_red_forms_two
{
    public partial class DeleteDevice : Form
    {
        private Form _Form1;
        public List<Registry> Registries { get; set; }
        public List<Device> Devices { get; set; }
        public DeleteDevice(Form Form1, List<Registry> registries,
List<Device> devices)
        {
            InitializeComponent();
            _Form1 = Form1;
            this.Registries = registries; // Сохраняем в свойствах
формы
            this.Devices = devices;
        }
    }
}

```

```

private void button3_Click(object sender, EventArgs e)
{
    _Form1.Show();
    this.Close();
}

private void DeleteDevice_Load(object sender, EventArgs e)
{
    // Получаем все уникальные значения location из всех
устройств
    var locations = Registries
        .Where(Registries => Registries.Labels != null &&
Registries.Labels.ContainsKey("location"))
        .Select(Registries =>
Registries.Labels["location"])
        .Distinct()
        .OrderBy(loc => loc)
        .ToList();

    // Очищаем комбобокс и добавляем значения
comboBox1.Items.Clear();
comboBox1.Items.AddRange(locations.ToArray());

    // Если нужно добавить пустое значение (опционально)
comboBox1.Items.Insert(0, "");
comboBox1.SelectedIndex = 0;
}

private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
{
    comboBox3.Items.Clear();

    // 1. Получаем выбранное здание (location из регистра)
string selectedBuilding =
comboBox1.SelectedItem?.ToString();
    if (string.IsNullOrEmpty(selectedBuilding))
        return;

    // 2. Находим все регистры с этим зданием
var registriesInBuilding = Registries
    .Where(r => r.Labels != null &&
        r.Labels.TryGetValue("location", out
var loc) &&
        loc == selectedBuilding)
    .ToList();

    // 3. Собираем ВСЕ устройства из этих регистров
var allDevicesInBuilding = registriesInBuilding
    .SelectMany(r => r.Devices) // "Разворачиваем"
списки устройств
    .ToList();

```

```

// 4. Извлекаем уникальные комнаты (метка "room")
var uniqueRooms = allDevicesInBuilding
    .Where(d => d.Labels != null &&
        d.Labels.TryGetValue("location", out
var _))
        .Select(d => d.Labels["location"])
        .Distinct()
        .OrderBy(room => room)
        .ToList();

// 5. Заполняем comboBox3
comboBox2.Items.AddRange(uniqueRooms.ToArray());
}

private void comboBox2_SelectedIndexChanged(object sender,
EventArgs e)
{
    comboBox3.Items.Clear(); // Очищаем перед заполнением

    // 1. Получаем выбранные здание и комнату
    string selectedBuilding =
comboBox1.SelectedItem?.ToString();
    string selectedRoom =
comboBox2.SelectedItem?.ToString();

    if (string.IsNullOrEmpty(selectedBuilding) ||
string.IsNullOrEmpty(selectedRoom))
        return;

    // 2. Находим регистры в выбранном здании
    var registriesInBuilding = Registries
        .Where(r => r.Labels != null &&
            r.Labels.TryGetValue("location", out
var loc) &&
                loc == selectedBuilding)
        .ToList();

    // 3. Находим все устройства в этих регистрах с
выбранной комнатой
    var devicesInRoom = registriesInBuilding
        .SelectMany(r => r.Devices) // Все устройства из
всех подходящих регистров
        .Where(d => d.Labels != null &&
            d.Labels.TryGetValue("location", out
var room) &&
                room == selectedRoom)
        .ToList();

    // 4. Заполняем comboBox1 именами устройств
    comboBox3.Items.AddRange(devicesInRoom.Select(d =>
d.DeviceName).ToArray());
}

```

```

private async void button1_Click(object sender, EventArgs
e)
{
    string House = comboBox1.SelectedItem?.ToString();
    string Room = comboBox2.SelectedItem?.ToString();
    string Device = comboBox3.SelectedItem?.ToString();

    string regID = ShowRegistryId();

    string deviceId = FindDeviceIdByRegistryId(regID,
Registries);

    MessageBox.Show($"{regID}\n{deviceId}");

    string NameFlow = $"Device%20{deviceId}";
    MessageBox.Show(NameFlow);
    string apiUrlFindIDFlow =
$"https://localhost:7101/FindIDFlow/{NameFlow}";
    string FlowID = await
PerformGetApiRequest(apiUrlFindIDFlow);
    MessageBox.Show(FlowID);
    // удаление первого Node-red
    string apiUrlDeletFlowOne =
$"https://localhost:7101/api/NodeRedFlow/deleteFlow/{FlowID}";
    await PerformDeleteApiRequest(apiUrlDeletFlowOne);

    NameFlow = $"House%20ROOM%20{regID}";
    apiUrlFindIDFlow =
$"https://localhost:7101/FindIDFlow/{NameFlow}";
    FlowID = await PerformGetApiRequest(apiUrlFindIDFlow);

    // Удаление второго node-red
    string apiUrlDeletFlowTwo =
$"https://localhost:7101/api/NodeRedFlow/deleteFlow/{FlowID}";
    await PerformDeleteApiRequest(apiUrlDeletFlowTwo);

    ///удаление девайса
    string apiUrlDeletDevice =
$"https://localhost:7101/api/DeletDevice/{deviceId}";
    await PerformDeleteApiRequest(apiUrlDeletDevice);

    //удаление регистра
    string apiUrlDeletReg =
$"https://localhost:7101/api/DeletRegistr/{regID}";
    await PerformDeleteApiRequest(apiUrlDeletReg);
}

private string ShowRegistryId()
{
    // 1. Формируем имя для поиска

```

```

        string registryNameToFind =
$"{comboBox1.SelectedItem}z{comboBox2.SelectedItem}z{comboBox3.Sel
ectedItem}";

        // 2. Ищем реестр с таким именем
Registry foundRegistry = Registries.FirstOrDefault(r
=>
        r.RegistryName.Equals(registryNameToFind,
StringComparison.OrdinalIgnoreCase));

        // 3. Показываем результат
if (foundRegistry != null)
{
    return foundRegistry.RegistryId;
}
else
{
    MessageBox.Show($"Реестр с именем
'{registryNameToFind}' не найден",
                    "Ошибка",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Warning);
    return null;
}
}

public string FindDeviceIdByRegistryId(string regID,
List<Registry> registries)
{
    foreach (var registry in registries)
    {
        // Проверяем соответствует ли текущий RegistryId
искомому
        if (registry.RegistryId == regID)
        {
            // Если соответствие найдено, проверяем
наличие устройств
            if (registry.Devices != null &&
registry.Devices.Count > 0)
            {
                // Предположим нам нужен id первого
устройства
                // или любая другая логика для выбора
устройства.
                return registry.Devices[0].DeviceId;
            }
        }
    }
    // Если устройство не найдено, возвращаем null или
бросаем исключение
    return null;
}

```

```

private async Task<string> PerformGetApiRequest(string
apiUrl)
{
    using (HttpClient client = new HttpClient())
    {
        try
        {
            HttpResponseMessage response = await
client.GetAsync(apiUrl);
            string responseContent = await
response.Content.ReadAsStringAsync();

            if (response.IsSuccessStatusCode)
            {
                // Например, если ответ имеет формат JSON
вида {"id": "значение"}
                var jsonData =
Newtonsoft.Json.JsonConvert.DeserializeObject<Dictionary<string,
string>>(responseContent);

                if (jsonData != null &&
jsonData.ContainsKey("id"))
                {
                    string id = jsonData["id"];
                    return id; // Возвращает ID
                }
                else
                {
                    MessageBox.Show("Ответ сервера не
содержит ID.",
                                "Предупреждение",
                                MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    return null;
                }
            }
            else
            {
                MessageBox.Show("Ошибка выполнения API-
запроса. Код ответа: " + response.StatusCode,
                                "Ошибка",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
                return null;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Произошла ошибка при отправке
API-запроса: " + ex.Message,
                                "Ошибка соединения",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
            return null;
        }
    }
}

```

```

    }

    private async Task<bool> PerformDeleteApiRequest(string
apiUrl)
    {
        using (HttpClient client = new HttpClient())
        {
            try
            {
                HttpResponseMessage response = await
client.DeleteAsync(apiUrl);

                if (response.IsSuccessStatusCode)
                {
                    MessageBox.Show("Запрос DELETE успешно
выполнен.",
                                "Информация",
                                MessageBoxButtons.OK, MessageBoxIcon.Information);
                    return true; // Успешное выполнение
запроса
                }
                else
                {
                    MessageBox.Show("Ошибка выполнения DELETE-
запроса. Код ответа: " + response.StatusCode,
                                "Ошибка",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return false; // Запрос выполнен с
ошибкой
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Произошла ошибка при
выполнении DELETE-запроса: " + ex.Message,
                                "Ошибка соединения",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false; // Возвращает ошибку при
выполнении запроса
            }
        }
    }
}

```

Листинг Г.4 – Форма Form1.cs

```

using Microsoft.Win32;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;

```

```

using System.Drawing;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace node_red_forms_two
{
    public partial class Form1 : Form
    {
        private List<Registry> _registries = new List<Registry>();
        private List<Device> _devices = new List<Device>();

        public Form1()
        {
            InitializeComponent();

            //Кнопки
            private void добавитьToolStripMenuItem_Click(object
sender, EventArgs e)
            {
                var allDevices = _registries.SelectMany(r =>
r.Devices).ToList();

                AddDevice addDevice = new AddDevice(this, _registries,
allDevices);
                addDevice.Show();
                this.Hide();
            }
            private void редактироватьToolStripMenuItem_Click(object
sender, EventArgs e)
            {
                var allDevices = _registries.SelectMany(r =>
r.Devices).ToList();

                UpdateDevice updateDevice = new
UpdateDevice(this, _registries, allDevices);
                updateDevice.Show();
                this.Hide();
            }

            private void удалитьToolStripMenuItem_Click(object sender,
EventArgs e)
            {
                var allDevices = _registries.SelectMany(r =>
r.Devices).ToList();

                DeleteDevice deleteDevice = new DeleteDevice(this,
_registries, allDevices);
                deleteDevice.Show();
            }
        }
    }
}

```

```

        this.Hide();
    }
    private void просмотретьToolStripMenuItem_Click(object
sender, EventArgs e)
    {
        ViewValueNodeRed viewValueNodeRed = new
ViewValueNodeRed(this);
        viewValueNodeRed.Show();
        this.Hide();
    }

    private void
подключениеКСерверуToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        ConnectServer connectServer = new ConnectServer(this);
        connectServer.Show();
        this.Hide();
    }

    //Загрузка формы
    private async void Form1_Load(object sender, EventArgs e)
    {
        await UpdateDataGridView();
    }

//.....ФУНКЦИИ.....
.....//

    private async Task UpdateDataGridView()
    {
        string apiUrl =
"https://localhost:7101/api/RegistrAndDevice";

        using (HttpClient client = new HttpClient())
        {
            try
            {
                // Выполняем GET-запрос к API
                HttpResponseMessage response = await
client.GetAsync(apiUrl);
                response.EnsureSuccessStatusCode();

                // Читаем содержание ответа
                string responseBody = await
response.Content.ReadAsStringAsync();

                // Десериализуем JSON-ответ
                _registries =
JsonConvert.DeserializeObject<List<Registry>>(responseBody);

                // Очищаем текущие данные DataGridView

```

```

dataGridView1.Rows.Clear();

// Устанавливаем заголовки столбцов (если
нужно)
dataGridView1.Columns.Clear();
dataGridView1.Columns.Add("RegistryLocation",
"Здание");
dataGridView1.Columns.Add("DeviceLocation",
"Комната");
dataGridView1.Columns.Add("DeviceName", "Имя
устройства");
dataGridView1.Columns.Add("RegistryState",
"Статус");

// Подготовка данных для отображения
foreach (var registry in _registries)
{
    // Получаем метку location из регистра
    (или пустую строку, если нет)
    string registryLocation = registry.Labels
!= null && registry.Labels.ContainsKey("location")
        ? registry.Labels["location"]
        : "";

    foreach (var device in registry.Devices)
    {
        // Получаем метку location из
устройства (или пустую строку, если нет)
        string deviceLocation = device.Labels
!= null && device.Labels.ContainsKey("location")
            ? device.Labels["location"]
            : "";

        // Добавляем строку в таблицу
dataGridView1.Rows.Add(
            registryLocation,
            deviceLocation,
            device.DeviceName,
            registry.RegistryState
        );
    }
}
}
catch (HttpRequestException e)
{
    MessageBox.Show($"Произошла ошибка при
выполнении запроса: {e.Message}");
}
catch (Exception e)
{
    MessageBox.Show($"Произошла ошибка:
{e.Message}");
}
}

```

```

        }
    }

//.....КЛАССЫ.....
.....//

    public class Registry
    {
        public string RegistryId { get; set; }
        public string RegistryName { get; set; }
        public string RegistryState { get; set; }
        public Dictionary<string, string> Labels { get; set; }
        public List<Device> Devices { get; set; }
    }

    public class Device
    {
        public string DeviceId { get; set; }
        public string DeviceName { get; set; }
        public string DeviceDescription { get; set; }
        public Dictionary<string, string> Labels { get; set; }
    }
}

```

Листинг Г.5 – Форма UpdateDevice.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using static node_red_forms_two.Form1;

namespace node_red_forms_two
{
    public partial class UpdateDevice : Form
    {
        private Form _Form1;
        public List<Registry> Registries { get; set; }
        public List<Device> Devices { get; set; }
        public UpdateDevice(Form Form1, List<Registry> registries,
List<Device> devices)
        {
            InitializeComponent();
            _Form1 = Form1;

```

```

        this.Registries = registries; // Сохраняем в свойствах
формы
        this.Devices = devices;
    }

    private void button3_Click(object sender, EventArgs e)
    {
        _Form1.Show();
        this.Close();
    }
    //.....ЗАГРУЗКА ФОРМЫ.....//
    private void UpdateDevice_Load(object sender, EventArgs e)
    {
        // Получаем все уникальные значения location из всех
устройств
        var locations = Registries
            .Where(Registries => Registries.Labels != null &&
Registries.Labels.ContainsKey("location"))
            .Select(Registries =>
Registries.Labels["location"])
            .Distinct()
            .OrderBy(loc => loc)
            .ToList();

        // Очищаем комбобокс и добавляем значения
comboBox2.Items.Clear();
comboBox2.Items.AddRange(locations.ToArray());

        // Если нужно добавить пустое значение (опционально)
comboBox2.Items.Insert(0, "");
comboBox2.SelectedIndex = 0;
    }
    //.....ОБРАБОТКА КОМБОВОКС.....//
    private void comboBox2_SelectedIndexChanged(object sender,
EventArgs e)
    {
        comboBox3.Items.Clear();

        // 1. Получаем выбранное здание (location из регистра)
        string selectedBuilding =
comboBox2.SelectedItem?.ToString();
        if (string.IsNullOrEmpty(selectedBuilding))
            return;

        // 2. Находим все регистры с этим зданием
        var registriesInBuilding = Registries
            .Where(r => r.Labels != null &&
                r.Labels.TryGetValue("location", out
var loc) &&
                    loc == selectedBuilding)
            .ToList();
    }

```

```

        // 3. Собираем ВСЕ устройства из этих регистров
        var allDevicesInBuilding = registriesInBuilding
            .SelectMany(r => r.Devices) // "Разворачиваем"
списки устройств
            .ToList();

        // 4. Извлекаем уникальные комнаты (метка "room")
        var uniqueRooms = allDevicesInBuilding
            .Where(d => d.Labels != null &&
                d.Labels.TryGetValue("location", out
var _))
            .Select(d => d.Labels["location"])
            .Distinct()
            .OrderBy(room => room)
            .ToList();

        // 5. Заполняем comboBox3
        comboBox3.Items.AddRange(uniqueRooms.ToArray());
    }

    private void comboBox3_SelectedIndexChanged(object sender,
EventArgs e)
    {
        comboBox1.Items.Clear(); // Очищаем перед заполнением

        // 1. Получаем выбранные здание и комнату
        string selectedBuilding =
comboBox2.SelectedItem?.ToString();
        string selectedRoom =
comboBox3.SelectedItem?.ToString();

        if (string.IsNullOrEmpty(selectedBuilding) ||
string.IsNullOrEmpty(selectedRoom))
            return;

        // 2. Находим регистры в выбранном здании
        var registriesInBuilding = Registries
            .Where(r => r.Labels != null &&
                r.Labels.TryGetValue("location", out
var loc) &&
                    loc == selectedBuilding)
            .ToList();

        // 3. Находим все устройства в этих регистрах с
выбранной комнатой
        var devicesInRoom = registriesInBuilding
            .SelectMany(r => r.Devices) // Все устройства из
всех подходящих регистров
            .Where(d => d.Labels != null &&
                d.Labels.TryGetValue("location", out
var room) &&
                    room == selectedRoom)
            .ToList();

```

```

        // 4. Заполняем comboBox1 именами устройств
        comboBox1.Items.AddRange(devicesInRoom.Select(d =>
d.DeviceName).ToArray());
    }

    private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
    {
        richTextBox1.Clear();
        checkBox1.Checked = false; // Сбрасываем состояние
перед проверкой

        string selectedDeviceName =
comboBox1.SelectedItem?.ToString();
        if (string.IsNullOrEmpty(selectedDeviceName)) return;

        // Находим устройство и его регистр
        Registry deviceRegistry = null;
        Device selectedDevice = null;

        foreach (var registry in Registries)
        {
            selectedDevice = registry.Devices.FirstOrDefault(d
=> d.DeviceName == selectedDeviceName);
            if (selectedDevice != null)
            {
                deviceRegistry = registry;
                break;
            }
        }

        // Выводим описание устройства
        if (selectedDevice != null)
        {
            richTextBox1.Text =
(selectedDevice.DeviceDescription ?? "Нет описания");

            // Устанавливаем состояние CheckBox
            checkBox1.Checked = deviceRegistry?.RegistryState
== "ACTIVE";
        }
        else
        {
            richTextBox1.Text = "Устройство не найдено";
        }
    }

    private async void button1_Click(object sender, EventArgs
e)
    {
        string House = comboBox1.SelectedItem?.ToString();
        string Room = comboBox2.SelectedItem?.ToString();
    }

```

```

        string Device = comboBox3.SelectedItem?.ToString();
        string DescriptionDevice = richTextBox1.Text;
        bool State = checkBox1.Checked;

        string regID = ShowRegistryId();

        string deviceId = FindDeviceIdByRegistryId(regID,
Registries);

        if (checkBox1.Checked)
        {
            string apiUrlActivateRegistry =
$"https://localhost:7101/api/registries/{regID}/activate";
            await PerformApiRequest(apiUrlActivateRegistry);

            string NameFlow = $"Device%20{deviceId}";

            string apiUrlFindIDFlow =
$"https://localhost:7101/FindIDFlow/{NameFlow}";
            string FlowID = await
PerformGetApiRequest(apiUrlFindIDFlow);

            string apiUrlEnableFlow =
$"https://localhost:7101/api/NodeRedFlow/enableFlow?flowId={FlowID}";
            await
PerformPutRequestWithoutBody(apiUrlEnableFlow);
        }
        else
        {
            string apiUrlDisableRegistry =
$"https://localhost:7101/api/registries/{regID}/disable";
            await PerformApiRequest(apiUrlDisableRegistry);

            string NameFlow = $"Device%20{deviceId}";

            string apiUrlFindIDFlow =
$"https://localhost:7101/FindIDFlow/{NameFlow}";
            string FlowID = await
PerformGetApiRequest(apiUrlFindIDFlow);

            string apiUrlDisableFlow =
$"https://localhost:7101/api/NodeRedFlow/disableFlow?flowId={FlowID}";
            await
PerformPutRequestWithoutBody(apiUrlDisableFlow);
        }
    }

    // Метод для поиска и отображения ID реестра
    private string ShowRegistryId()
    {

```

```

        // 1. Формируем имя для поиска
        string registryNameToFind =
        $"{comboBox2.SelectedItem}z{comboBox3.SelectedItem}z{comboBox1.Sel
        ectedItem}";

        // 2. Ищем реестр с таким именем
        Registry foundRegistry = Registries.FirstOrDefault(r
=>
            r.RegistryName.Equals(registryNameToFind,
            StringComparison.OrdinalIgnoreCase));

        // 3. Показываем результат
        if (foundRegistry != null)
        {
            return foundRegistry.RegistryId;
        }
        else
        {
            MessageBox.Show($"Реестр с именем
            '{registryNameToFind}' не найден",
                            "Ошибка",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Warning);
            return null;
        }
    }

    public string FindDeviceIdByRegistryId(string regID,
    List<Registry> registries)
    {
        foreach (var registry in registries)
        {
            // Проверяем соответствует ли текущий RegistryId
            // искомому
            if (registry.RegistryId == regID)
            {
                // Если соответствие найдено, проверяем
                // наличие устройств
                if (registry.Devices != null &&
                registry.Devices.Count > 0)
                {
                    // Предположим нам нужен id первого
                    // устройства
                    // или любая другая логика для выбора
                    // устройства.
                    return registry.Devices[0].DeviceId;
                }
            }
        }
        // Если устройство не найдено, возвращаем null или
        // бросаем исключение
        return null;
    }

```

```

private async Task PerformApiRequest(string apiUrl)
{
    using (HttpClient client = new HttpClient())
    {
        try
        {
            HttpResponseMessage response = await
client.PostAsync(apiUrl, null); // Используем POST, второй
параметр для тела запроса

            if (response.IsSuccessStatusCode)
            {
                string responseContent = await
response.Content.ReadAsStringAsync();
                MessageBox.Show("API-запрос успешно
выполнен!\nОтвет сервера: " + responseContent,
                                "Успех",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show("Ошибка выполнения API-
запроса. Код ответа: " + response.StatusCode,
                                "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Произошла ошибка при отправке
API-запроса: " + ex.Message,
                                "Ошибка соединения",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}

private async Task PerformPutRequestWithoutBody(string
apiUrl)
{
    using (HttpClient client = new HttpClient())
    {
        try
        {
            HttpResponseMessage response = await
client.PutAsync(apiUrl, null); // Явно указываем null

            if (response.IsSuccessStatusCode)
            {
                string responseContent = await
response.Content.ReadAsStringAsync();

```

```

        MessageBox.Show("API-запрос успешно
выполнен!\nОтвет сервера: " + responseContent,
                        "Успех",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Ошибка выполнения API-
запроса. Код ответа: " + response.StatusCode,
                        "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
catch (Exception ex)
{
    MessageBox.Show("Произошла ошибка при отправке
API-запроса: " + ex.Message,
                    "Ошибка соединения",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
}

```

```

private async Task<string> PerformGetApiRequest(string
apiUrl)
{
    using (HttpClient client = new HttpClient())
    {
        try
        {
            HttpResponseMessage response = await
client.GetAsync(apiUrl);
            string responseContent = await
response.Content.ReadAsStringAsync();

            if (response.IsSuccessStatusCode)
            {
                // Например, если ответ имеет формат JSON
вида {"id": "значение"}
                var jsonData =
Newtonsoft.Json.JsonConvert.DeserializeObject<Dictionary<string,
string>>(responseContent);

                if (jsonData != null &&
jsonData.ContainsKey("id"))
                {
                    string id = jsonData["id"];
                    return id; // Возвращает ID
                }
            }
            else
            {

```



```

private void button2_Click(object sender, EventArgs e)
{
    _Form1.Show();
    this.Close();
}

private async void ViewValueNodeRed_Load(object sender,
EventArgs e)
{
    string url =
"http://127.0.0.1:1880/ui/#!/0?socketid=XjJUEjVnQaT5rRUiAAAN";
    await webView21.EnsureCoreWebView2Async(); //
Инициализация
    webView21.CoreWebView2.Navigate(url);
}
}
}

```

ПРИЛОЖЕНИЕ Д
(ОБЯЗАТЕЛЬНОЕ)

Тестирование

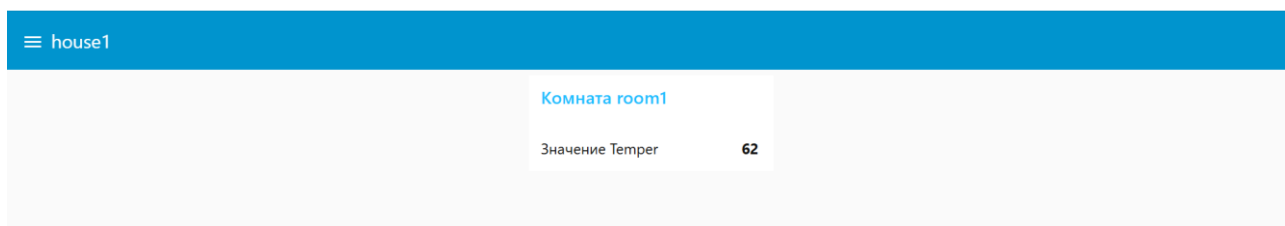


Рисунок Д.1 – Тестирование логики №1

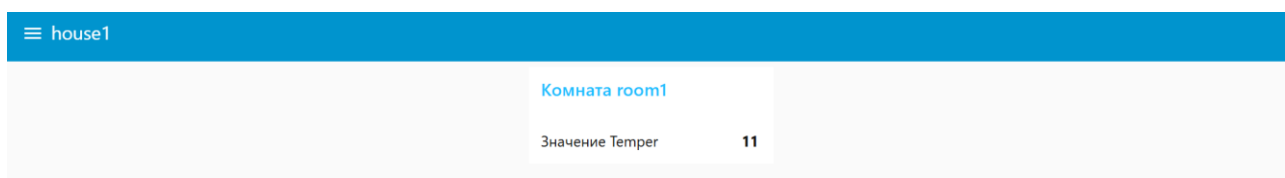


Рисунок Д.2 – Тестирование логики №2

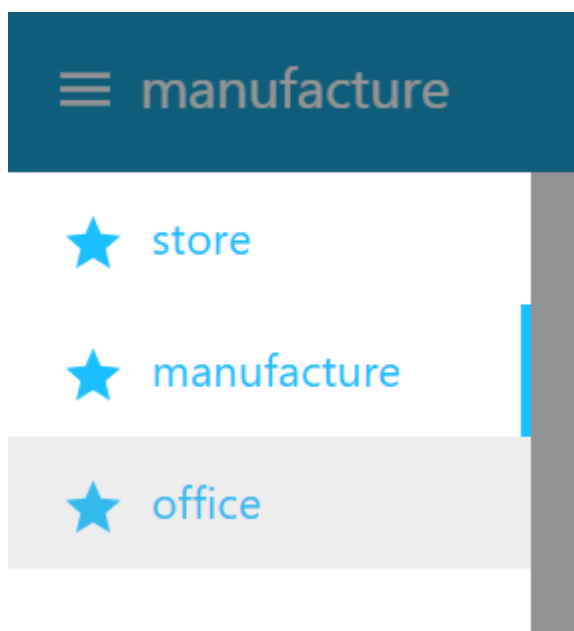


Рисунок Д.3 – Тестирование логики №3

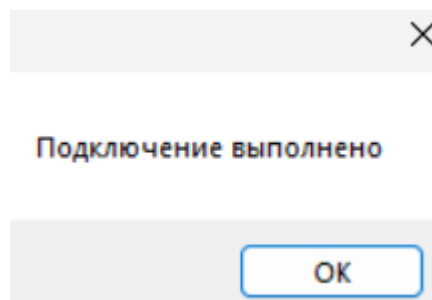
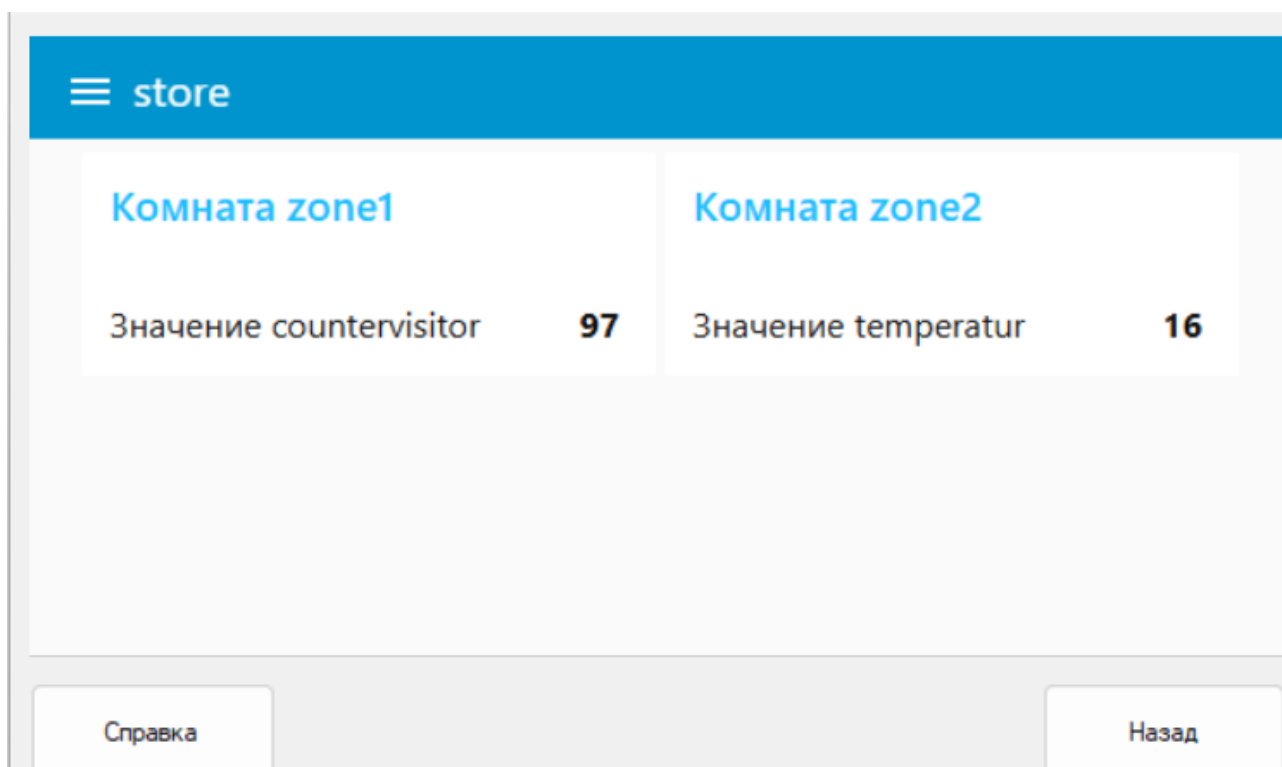


Рисунок Д.4 – Тестирование логики №4



store

Комната zone1

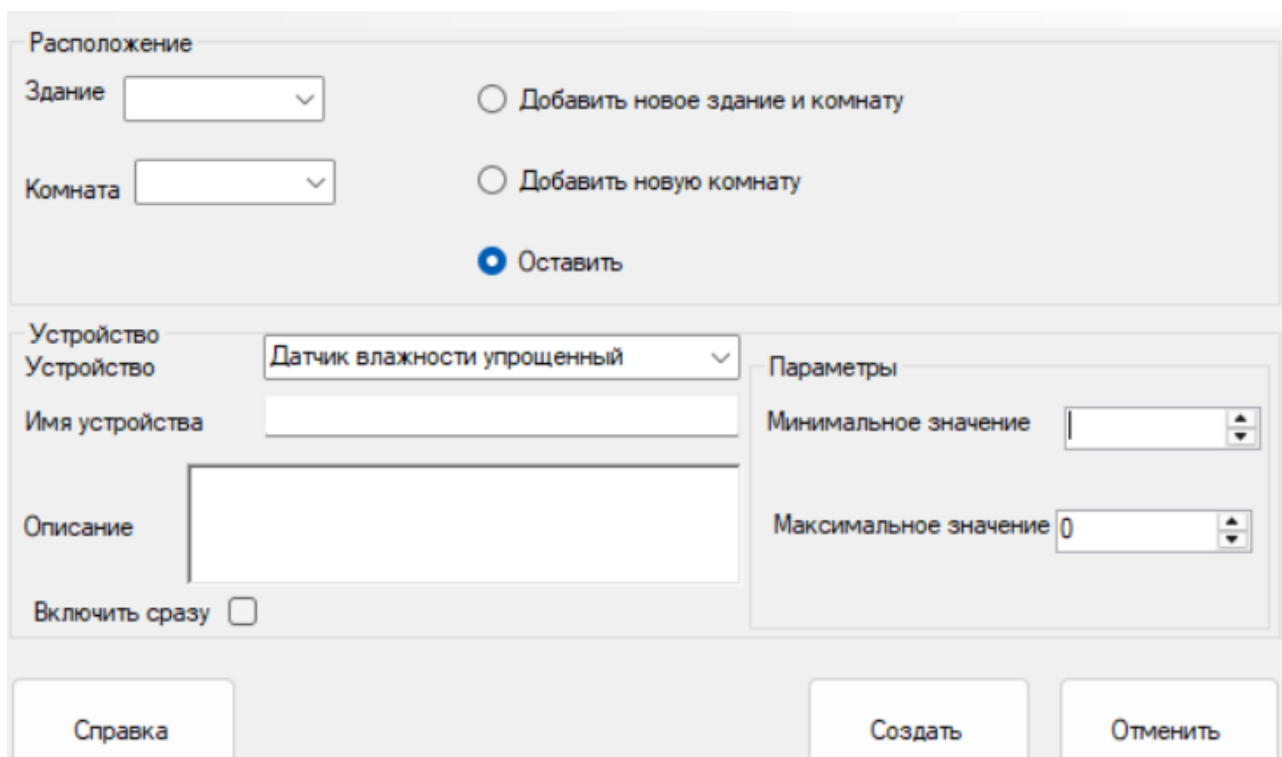
Значение countervisitor 97

Комната zone2

Значение temperatur 16

Справка Назад

Рисунок Д.5 – Тестирование логики №5



Расположение

Здание

Комната

☐ Добавить новое здание и комнату

☐ Добавить новую комнату

☒ Оставить

Устройство

Устройство

Имя устройства

Описание

Параметры

Минимальное значение

Максимальное значение

Включить сразу ☐

Справка Создать Отменить

Рисунок Д.6 – Тестирование логики №6

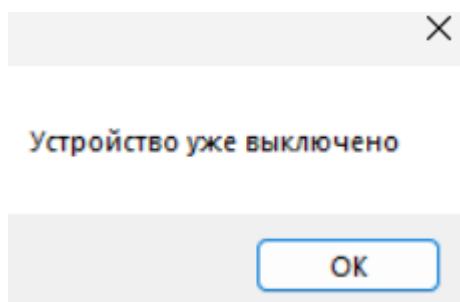


Рисунок Д.7 – Тестирование логики №7

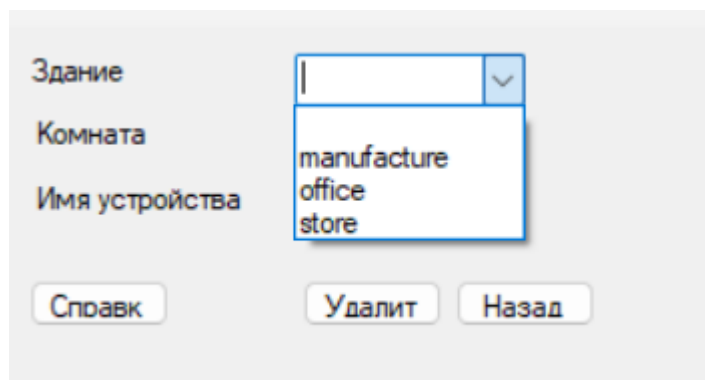


Рисунок Д.8 – Тестирование логики №8

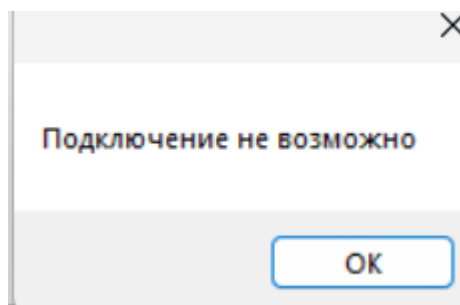
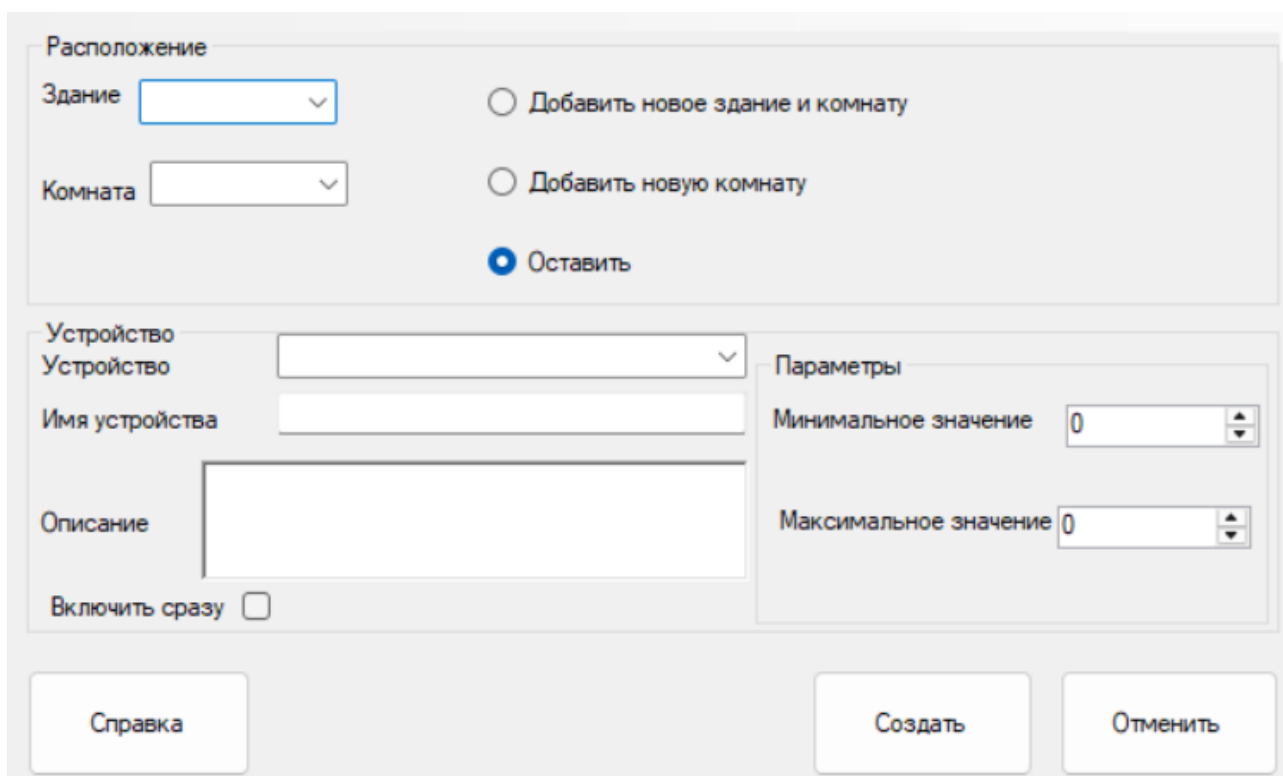


Рисунок Д.9 – Тестирование логики №9



Расположение

Здание

Комната

☐ Добавить новое здание и комнату

☐ Добавить новую комнату

☒ Оставить

Устройство

Устройство

Имя устройства

Описание

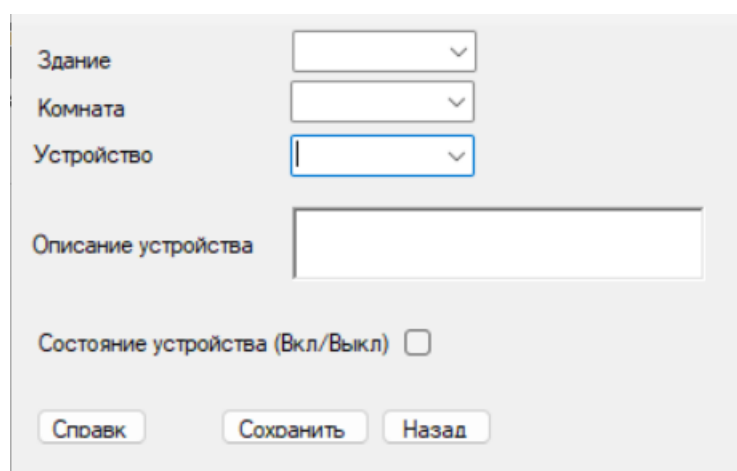
Включить сразу ☐

Параметры

Минимальное значение

Максимальное значение

Рисунок Д.10 – Тестирование интерфейса №1



Здание

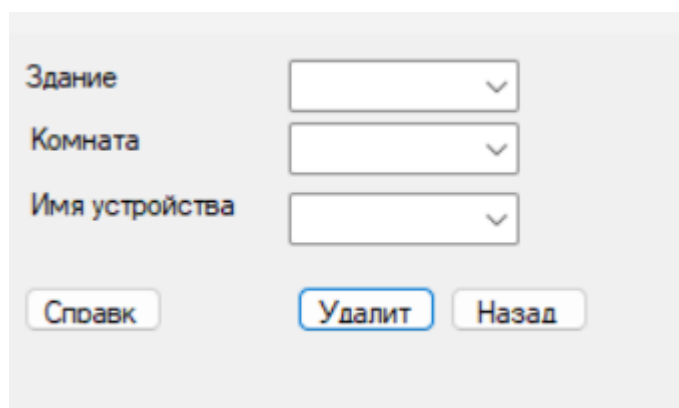
Комната

Устройство

Описание устройства

Состояние устройства (Вкл/Выкл) ☐

Рисунок Д.11 – Тестирование интерфейса №2



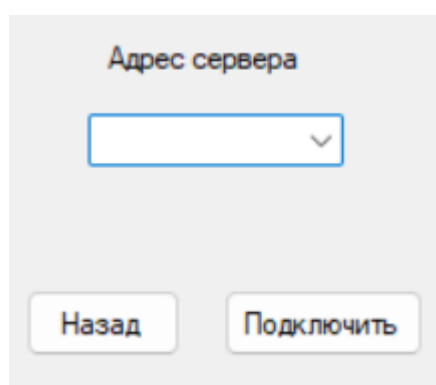
Здание

Комната

Имя устройства

Справка Удалит Назад

Рисунок Д.12 – Тестирование интерфейса №3



Адрес сервера

Назад Подключить

Рисунок Д.13 – Тестирование интерфейса №4

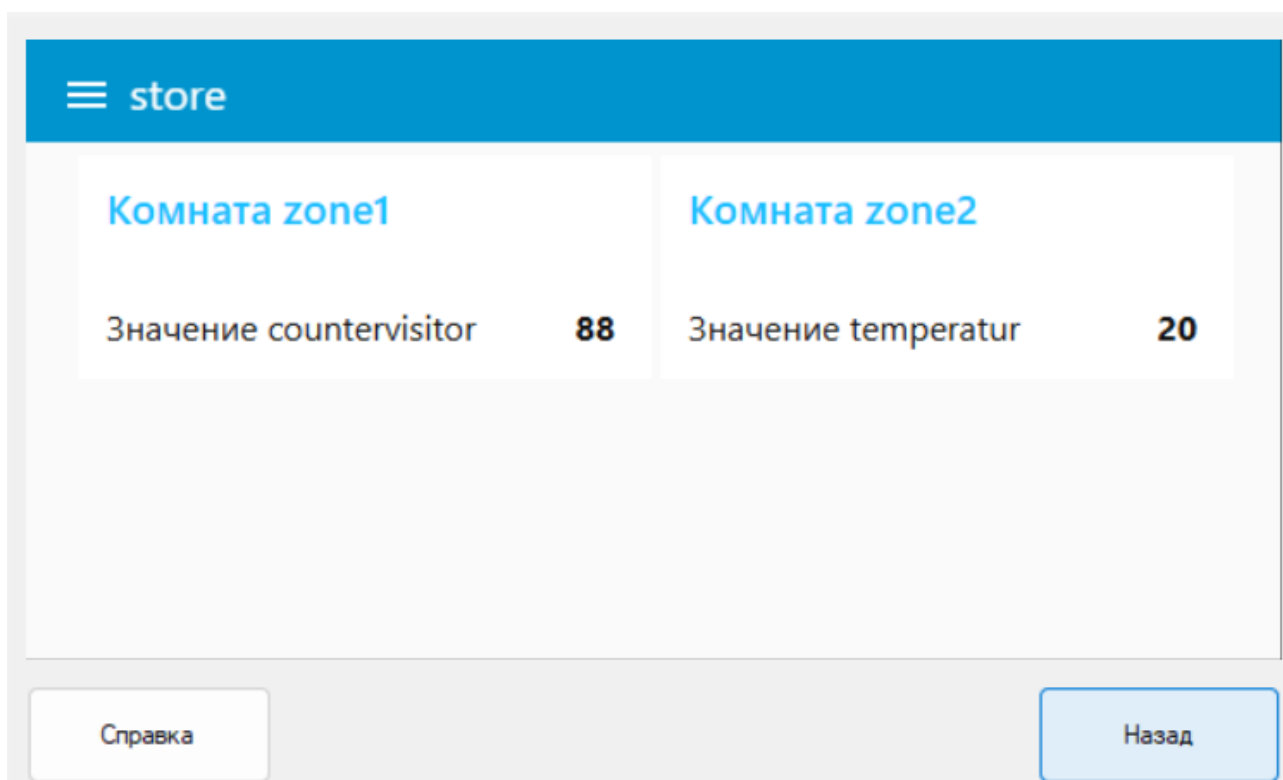


Рисунок Д.14 – Тестирование интерфейса №5

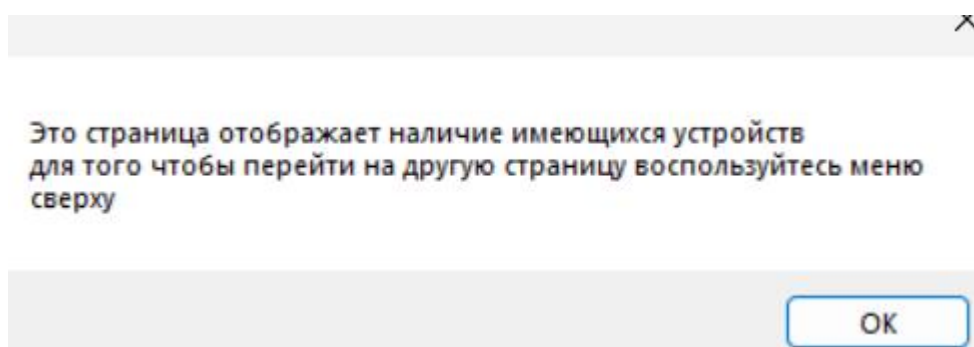


Рисунок Д.15 – Тестирование интерфейса №6

Устройство Моделирование Сервис				
	Здание	Комната	Имя устройства	Статус
▶	manufacture	workshop3	humidity	ACTIVE
	store	zone1	temperatur	ACTIVE
	office	room2	light	ACTIVE
	manufacture	workshop2	temperatur	ACTIVE
	office	room1	temperature	ACTIVE
	office	room1	light	ACTIVE
	store	zone2	temperatur	ACTIVE
	store	zone1	countervisitor	ACTIVE
*				

Рисунок Д.16 – Тестирование интерфейса №7-9