

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 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Web-приложения контроля за показаниями
датчиков умного дома**

Студент _____ Мишихин Владислав Юрьевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Дорофеева О.С.
(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
«25» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Мишихин Владислав Юрьевич гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Web-приложения контроля за показаниями датчиков умного дома

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

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

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

1) *Назначение: программные средства для контроля за показаниями датчиков умного дома и получению рекомендаций.*

2) *Среда выполнения: Windows 11.*

Языковые средства разработки: JavaScript, Java, SQL, C++, UML.

Инструментальные средства: JetBrains IntelliJ idea, Swagger, npm,

Docker, PostgreSQL, Draw.io, Visual Paradigm 17.0. Arduino IDE

Технологии разработки ООП, микросервисная архитектура.

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) настройка системы в соответствии с предпочтениями пользования;

в) хранение и систематизация информации от датчиков умного дома;

г) разделение функциональности на ролевые модели;

д) анализ безопасности и комфорта условий проживания пользователя;

е) визуализация аналитических данных для лучшего понимания результатов.

ж) генерация аналитических и рекомендательных отчетов для пользователя

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;

2) Проектирование;

3) Реализация;

4) Контроль качества программного обеспечения;

5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 13.04.2025	
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 115 листа, 44 рисунка, 22 таблицы, 20 использованных источников и 4 приложения.

УМНЫЙ ДОМ, WEB-ПРИЛОЖЕНИЕ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, REST, UML, POSTGRESQL, JAVA, JS, БАЗА ДАННЫХ, SPRING BOOT, ARDUINO, C++.

Объект исследования – программные и технические средства для наблюдения за показаниями умного дома.

Цель выпускной квалификационной работы – разработать программные и аппаратные средства для наблюдения за показаниями умного дома и информирования пользователя о качестве жизни и способов ее улучшения.

Технологии разработки – разработка клиентской части приложения проводилась на языке программирования JavaScript в среде разработки JetBrains IntelliJIdea. Разработка серверной части приложения производилась на языке программирования Java, с использованием фреймворка Spring Boot в среде разработки JetBrains IntelliJIdea. Разработка аппаратной части производилась на языке программирования C++, с использованием библиотеки esp-32 dev в среде разработки Arduino IDE. Разработка базы данных проводилась в СУБД PostgreSQL.

Среда исполнения – Windows 11, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская, серверная и аппаратная части программных средств для наблюдения за показаниями умного дома. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08ВР211.17 ПЗ						
Изм.	Лист	№ докум.	Подпись	Дата	Web-приложения контроля за показаниями датчиков умного дома. Пояснительная записка			Лит.	Лист	Листов	
Разраб.		Мишихин В.Ю.									
Пров.		Дорофеева О.С.							4	115	
Реценз.								Группа 21ВП1			
Н. Контр.		Такташкин Д.В									
Утв.		Козлов А.Ю.									

Содержание

Введение.....	7
1 Анализ программных средств для контроля за показаниями датчиков умного дома.....	9
1.1 Исследование предметной области.....	9
1.2 Обзор и анализ программных средств системы контроля показаний датчиков умного дома.....	10
1.2.1 Система умного дома «Ajax starter kit»	10
1.2.2 Система умного дома «Ростелеком Безопасность».....	11
1.2.3 Система умного дома «Xiaomi Smart Home Security kit»	12
1.2.4 Система умного дома «Sber умный дом»	13
1.2.5 Сравнение программных средств контроля показаний датчиков умного дома	14
1.3 Анализ функциональных требований.....	17
1.4 Анализ нефункциональных требований.....	21
1.5 Планирование разработки и оценка бюджета.....	22
2 Проектирование программных средств для системы контроля показаний датчиков умного дома.....	26
2.1 Архитектура приложения.....	26
2.2 Проектирование структур данных	28
2.2.1 Концептуальная модель базы данных.....	28
2.2.2 Логические модели баз данных	29
2.3 Проектирование пользовательского интерфейса.....	34
2.4 Проектирование классов программных средств для контроля за показаниями датчиков умного дома.....	35
3 Реализация программных средств для системы контроля за показаниями датчиков умного дома.....	37
3.1 Реализация базы данных	37

3.2 Реализация серверной части приложения	40
3.3 Реализация клиентской части приложения	43
3.4 Реализация интерфейса приложения	44
3.5 Реализация аппаратной части приложения	51
3.6 Реализация взаимодействия частей приложения.....	53
3.7 Реализация подготовки отчетов в приложении	57
4 Контроль качества программного обеспечения.....	60
4.1 Обоснование выбора режима тестирования и метрик кода	60
4.2 Тестирование программных средств для контроля показаний датчиков умного дома	62
4.2.1 Функциональное тестирование черным ящиком.....	62
4.2.2 Модульное тестирование	66
4.3 Оценка качества программного кода	69
4.3.1 Метрика Холстеда.....	69
4.3.2 Метрика Чепина	71
Заключение	73
Список используемых источников.....	75
Приложение А (обязательное) Листинг.....	77
Приложение Б (рекомендуемое) Графическая часть.....	101
Приложение В (рекомендуемое) Атрибуты сущностей базы данных.....	106
Приложение Г (рекомендуемое) Результаты тестирования	114

Введение

Современные технологии «умного дома» активно развиваются, предоставляя пользователям возможность удаленного мониторинга и управления домашними системами. Одной из ключевых задач в этой области является контроль за показаниями датчиков, таких как температура, влажность, освещенность, шума, протечки воды и другие. Грамотная организация сбора и анализа этих данных позволяет повысить энергоэффективность, безопасность и комфорт в жилом пространстве [1].

В настоящее время пользователи стремятся к автоматизации процессов мониторинга домашней среды, что требует удобных и надежных инструментов для визуализации и анализа данных с датчиков. Однако без специализированного программного обеспечения эффективное управление этими показателями затруднено.

Разработка Web-приложения для контроля показаний датчиков умного дома является актуальной задачей, так как подобные системы позволяют: автоматизировать сбор и хранение данных с различных датчиков, визуализировать изменения параметров в реальном времени, настраивать уведомления о критических отклонениях (например, утечке газа или повышении температуры), анализировать статистику для оптимизации энергопотребления и безопасности.

Целью выпускной квалификационной работы является разработка Web-приложения, обеспечивающего удобный мониторинг и управление показаниями датчиков умного дома.

Системы умного дома, подобные разрабатываемой, объединяют различные типы датчиков для комплексного мониторинга окружающей среды. В данной системе используются: датчик уровня воды для обнаружения протечек, датчик угарного газа MQ7 для контроля безопасности воздуха, цифровой датчик DHT11, измеряющий температуру и влажность, высокоточный термометр DS18B20, а также фоторезистор для отслеживания

уровня освещенности. Такое сочетание цифровых и аналоговых сенсоров позволяет не только фиксировать текущее состояние домашней среды, но и оперативно реагировать на потенциально опасные ситуации – от превышения концентрации СО до протечек воды. Каждый датчик играет важную роль в создании безопасного и комфортного пространства, а их совместная работа обеспечивает всесторонний контроль за ключевыми параметрами жилого помещения.

Исходя из цели данной работы, были поставлены следующие задачи:

- проанализировать предметную область и аналоги похожего программного обеспечения, поставить задачу на разработку;
- спроектировать программные средства, а также структуры данных разрабатываемого приложения;
- разработать модули программных средств;
- проконтролировать качество программного обеспечения.

Успешное решение поставленных задач позволит создать эффективный инструмент для контроля за показаниями датчиков умного дома.

1 Анализ программных средств для системы контроля за показаниями датчиков умного дома

1.1 Исследование предметной области

Программные средства для контроля показаний датчиков умного дома – это инструменты, позволяющие пользователям в режиме реального времени отслеживать состояние домашней среды, получать уведомления о критических изменениях и анализировать статистику для оптимизации энергопотребления и безопасности [2].

Датчики умного дома охватывают различные аспекты жизнеобеспечения жилого пространства, включая контроль температуры, влажности, уровня CO, протечек воды и шума. Каждый из этих параметров требует точного мониторинга и настройки индивидуальных порогов срабатывания. Например, датчик температуры может использоваться для управления климатом, а датчик протечки – для предотвращения аварийных ситуаций.

Подобные системы разрабатываются для широкого круга пользователей: от владельцев квартир и частных домов до управляющих компаний, ответственных за эксплуатацию многоквартирных зданий. Обязательным условием работы приложения является регистрация пользователя, что обеспечивает персонализацию настроек и безопасный доступ к данным [3].

Таким образом, современные решения для мониторинга умного дома не только предоставляют информацию о состоянии жилья, но и помогают автоматизировать контроль за критически важными параметрами, повышая уровень комфорта и безопасности.

Современные системы умного дома строятся на взаимодействии с разнообразными датчиками, которые условно можно разделить на два типа: цифровые и аналоговые. Их ключевое отличие заключается в способе передачи данных. Цифровые датчики, такие как DHT11 или DS18B20, обмениваются информацией с микроконтроллером (например, ESP32) с помощью дискретных

сигналов, что обеспечивает высокую помехоустойчивость и простоту обработки. Аналоговые датчики, напротив, передают непрерывный сигнал, например, уровень освещенности или концентрацию газа, который требует преобразования в цифровой формат с помощью АЦП (аналого-цифрового преобразователя).

Микроконтроллеры, такие как ESP32, оснащены универсальными GPIO-пинами (General Purpose Input/Output), которые могут работать как с цифровыми, так и с аналоговыми устройствами. Это позволяет гибко подключать различные датчики, настраивая их под конкретные задачи – от контроля температуры до обнаружения протечек воды. Умение корректно работать с обоими типами датчиков становится важным навыком, поскольку каждый из них играет свою роль в создании надежной и точной системы мониторинга. Например, цифровые датчики часто используются для точных измерений, а аналоговые – для отслеживания плавно меняющихся параметров, таких как уровень освещенности или качество воздуха.

1.2 Обзор и анализ программных средств системы контроля показаний датчиков умного дома

1.2.1 Система умного дома «Ajax starter kit»

Система «AJAX StarterKit» представляет собой комплексное решение для мониторинга безопасности и контроля параметров домашней среды. Комплект системы приведен на рисунке 1.



Рисунок 1 – Система умного дома «Ajax starter kit»

Основным преимуществом системы является надежность работы и стабильность соединения с датчиками, что обеспечивает своевременное оповещение пользователя о критических событиях. Приложение поддерживает базовые функции мониторинга, включая контроль за открытием дверей/окон и движением в помещении.

Ключевым недостатком системы является ограниченный набор поддерживаемых датчиков – отсутствует возможность мониторинга уровня СО, шума и других параметров домашней среды. Кроме того, функционал приложения не предусматривает углубленного анализа данных и построения детализированных отчетов, что снижает его полезность для комплексного контроля умного дома.

1.2.2 Система умного дома «Ростелеком Безопасность»

Система «Ростелеком Безопасность» представляет собой комплексное решение для мониторинга жилого пространства. Комплект системы представлен на рисунке 2.



Рисунок 2 – Система умного дома «Ростелеком Безопасность»

Основным преимуществом системы является широкий спектр поддерживаемых датчиков, включая датчики температуры, влажности и протечки воды. Приложение предоставляет пользователям возможность

настройки индивидуальных сценариев реагирования на различные события, а также предлагает интеграцию с другими сервисами компании. Особого внимания заслуживает система оповещений, которая включает как push-уведомления, так и SMS-предупреждения.

Однако система имеет ряд существенных ограничений. Наиболее значимым недостатком является отсутствие поддержки датчиков уровня СО и шума, что снижает ее эффективность для комплексного мониторинга домашней среды. Кроме того, функционал приложения не предусматривает детализированного анализа исторических данных и формирования расширенных отчетов, что ограничивает возможности для долгосрочного анализа изменений параметров домашней среды.

1.2.3 Система умного дома «Xiaomi Smart Home Security kit»

Система «Xiaomi Smart Home Security Kit» представляет собой современное решение для комплексного мониторинга домашнего пространства, комплект которого представлен на рисунке 3.



Рисунок 3 – Система умного дома «Xiaomi Smart Home Security kit»

Ключевым преимуществом данной системы является глубокая интеграция с экосистемой Xiaomi и поддержка широкого спектра датчиков, включая устройства для измерения температуры, влажности и контроля протечек воды. Приложение обеспечивает надежную синхронизацию данных

между устройствами и предлагает гибкую систему уведомлений о критических изменениях параметров домашней среды.

Однако система обладает определенными ограничениями. Наиболее существенным недостатком является отсутствие поддержки датчиков уровня СО и шума, что снижает ее эффективность для всестороннего анализа состояния жилого помещения. Также в приложении не реализованы функции для глубокого анализа исторических данных и формирования комплексных отчетов, что ограничивает возможности для долгосрочного мониторинга и выявления тенденций изменения параметров домашней среды.

1.2.4 Система умного дома «Sber умный дом»

Система «Sber Умный дом» представляет собой современное решение для автоматизации жилых помещений, разработанное при участии Сбербанка. Комплект системы представлен на рисунке 4.



Рисунок 4 – Система умного дома «Sber умный дом»

Основным преимуществом системы является тесная интеграция с экосистемой Сбербанка и поддержка голосового управления через ассистента «Салют». Приложение предлагает базовый набор функций для мониторинга температуры, влажности и контроля протечек воды, а также обеспечивает стабильную работу через мобильное приложение.

Однако система имеет ряд существенных ограничений. Главным недостатком является узкий перечень поддерживаемых датчиков – отсутствует возможность подключения датчиков уровня СО и шума. Также в приложении отсутствуют расширенные функции анализа данных и формирования детализированных отчетов, что снижает его полезность для комплексного мониторинга параметров домашней среды. Эти ограничения особенно заметны при попытке использовать систему для решения комплексных задач умного дома, где требуется не просто сбор данных, но и их содержательный анализ с выработкой прогностических моделей. Отсутствие API для интеграции со сторонними аналитическими сервисами дополнительно сужает потенциальные сценарии использования платформы.

1.2.5 Сравнение программных средств контроля показаний датчиков умного дома

Проведем сравнительный анализ рассмотренных выше систем умного дома, таких как «AJAX», «Ростелеком», «Xiaomi», «Sber», по ключевым функциям.

Для оценки приложений будет использоваться трехбалльная шкала соответствия. В таблице 1 представлены характеристики оценки.

Таблица 1 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) полностью отсутствует
1	существует частичная реализация функции (качества)
2	функция (качество) полностью реализована

Результаты сравнения программных средств для контроля за показаниями датчиков умного дома представлены в таблице 2.

Таблица 2 – Сравнение программных средств аналогичного назначения

Критерий \ Приложение	«AJAX»	«Ростелеком»	«Xiaomi»	«Sber»
1. Поддержка датчиков				

Продолжение таблицы 2

Критерий \ Приложение	«AJAX»	«Ростелеком»	«Xiaomi»	«Sber»
- Температуры	1	2	2	2
- Влажности	0	2	2	2
- СО (угарный газ)	0	1	1	0
- Уровня воды (протечка)	2	2	2	2
- Шума	0	0	0	0
- Освещенности	1	1	1	1
2. Локальная работа	2	1	1	1
3. Уведомления				
- Telegram	0	0	0	0
- Почта	0	1	0	1
- Внутри приложения	1	1	1	1
- SMS	0	1	0	0
4. Визуализация данных				
- Графики изменений	1	1	2	2
- Экспорт данных	0	0	1(CSV)	0
5. Стоимость базового комплекта	0	1	2	0
ИТОГО	8	14	15	12

Для наглядности, полученные результаты отображены на диаграмме, приведенной на рисунке 5.

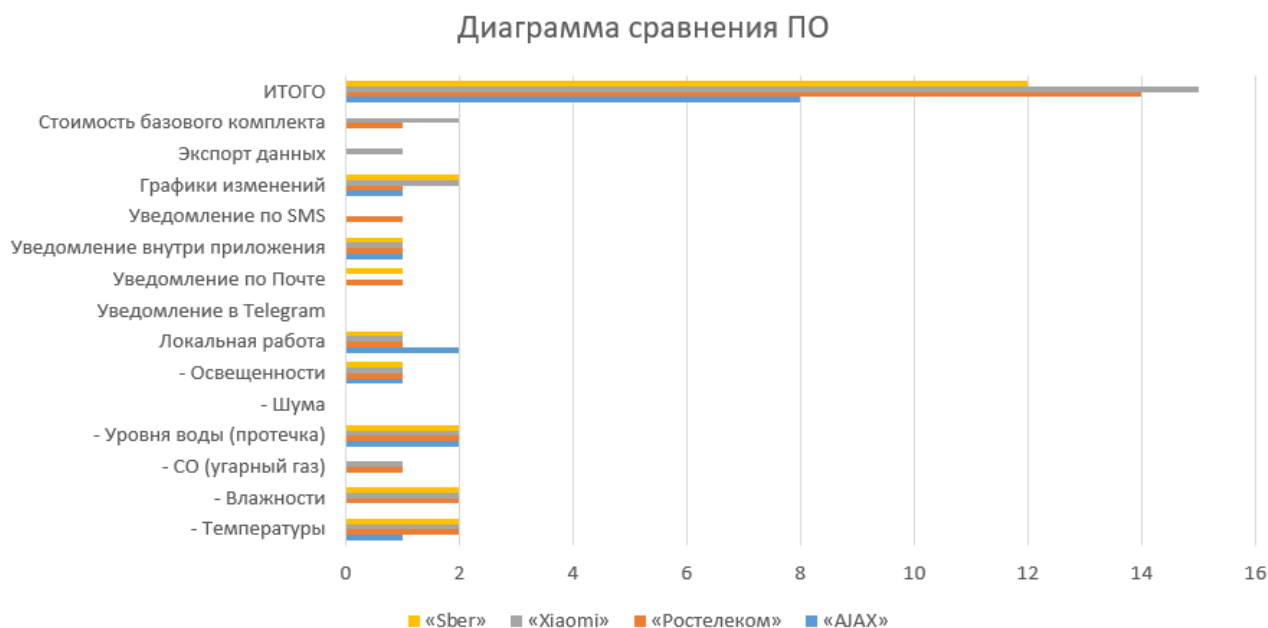


Рисунок 5 – Результаты сравнения приложений-аналогов

Проведенный обзор существующих решений показал, что ни одна из рассмотренных систем не обеспечивает полноценного контроля всех ключевых параметров домашней среды. Хотя решение Xiaomi предлагает наиболее широкий функционал среди аналогов, оно все же имеет существенные ограничения в части поддержки специализированных датчиков и возможностей анализа данных.

Основной проблемой существующих систем остается их узкая специализация – они либо ориентированы на безопасность, либо предлагают базовый мониторинг, но не обеспечивают комплексного подхода к управлению домашней средой. Особенно заметен недостаток поддержки датчиков уровня CO и шума, которые критически важны для полноценного контроля качества воздуха и акустического комфорта.

На основании выявленных ограничений было принято решение о разработке специализированного web-приложения, которое объединит все необходимые функции для комплексного мониторинга домашней среды. Новая

система будет отличаться расширенными возможностями анализа данных, гибкой системой оповещений и поддержкой всех типов датчиков, что позволит пользователям получить полный контроль над параметрами своего жилого пространства.

1.3 Анализ функциональных требований

В процессе анализа задачи на разработку программных средств для контроля за показаниями датчиков умного дома были выделены следующие функциональные требования:

- регистрация и авторизация пользователя;
- редактирование аккаунта пользователя;
- настройка отображение: кастомизация визуальной информации приложения по предпочтениям пользователя;
- получения показаний датчиков (температуры, влажности, CO, уровня освещенности, протечки, уровня шума);
- получение уведомлений;
- проверка состояния системы;
- получение отчета об условиях жизни;
- получение диагностики системы.

В соответствии с выявленными требованиями была разработана диаграмма вариантов использования UML, представленная на рисунке 6.

Диаграмма включает ключевые функции, такие как регистрация и авторизация, позволяющие пользователю получить доступ к персонализированным настройкам и данным. Важным аспектом системы является возможность редактирования аккаунта, что обеспечивает гибкость и удобство при изменении персональных данных. Настройка отображения позволяет адаптировать интерфейс под индивидуальные предпочтения, делая работу с приложением более комфортной. Получение данных с датчиков играет ключевую роль в мониторинге условий окружающей среды.

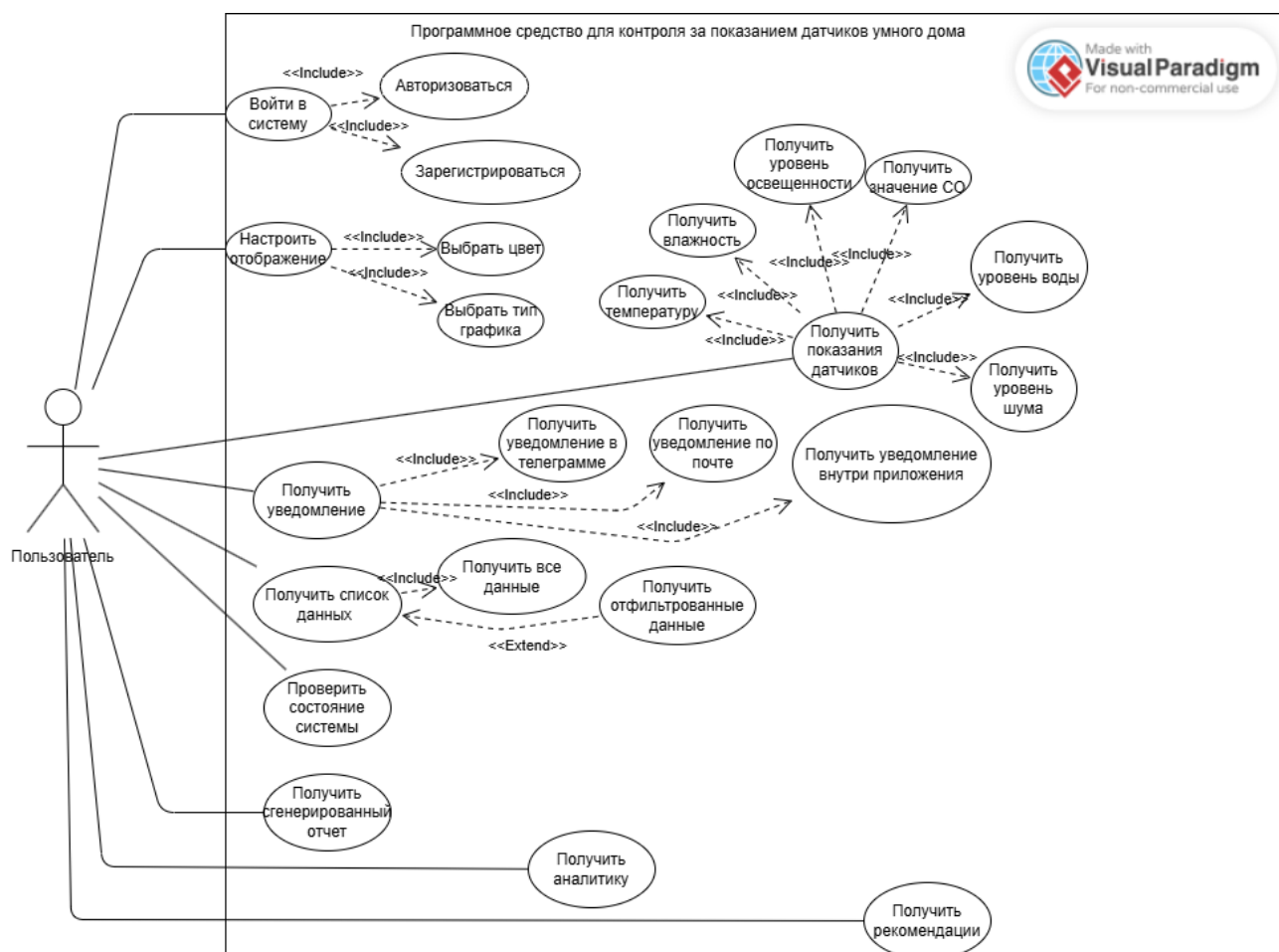


Рисунок 6 – Диаграмма вариантов использования

Ниже приведены спецификации наиболее важных бизнес-процессов.

Сценарий варианта использования «Войти в систему» представлен в таблице 3.

Таблица 3 – Вариант использования «Войти в систему»

Прецедент: Войти в систему
ID: 1
Краткое описание: Пользователь регистрируется или авторизуется в приложении
Главные актеры: Пользователь
Второстепенные актеры: Нет

Продолжение таблицы 3

<p>Предусловие:</p> <ol style="list-style-type: none"> 1. Пользователь запустил систему 2. Пользователь зашел в приложение 3. Пользователь не был зарегистрирован/авторизован
<p>Основной поток:</p> <ol style="list-style-type: none"> 1. Пользователь переходит на страницу входа в приложение 2. Пользователь заполняет форму входа в приложение 3. Пользователь получает доступ к функционалу приложения <p>Альтернативные потоки:</p> <ol style="list-style-type: none"> 2.а Пользователь вводит данные для регистрации (пароль, адрес электронной почты)
<p>Постусловия:</p> <ol style="list-style-type: none"> 1. Пользователь вошел в приложение 2. Пользователю доступен функционал приложения

Сценарий варианта использования «Настроить отображение» изображен в таблице 4.

Таблица 4 – Вариант использования «Настроить отображение»

Прецедент: Настроить отображение
ID: 2
<p>Краткое описание:</p> <p>Пользователь настраивает отображение данных, согласно своим предпочтениям</p>
<p>Главные актеры:</p> <p>Пользователь</p>
<p>Второстепенные актеры:</p> <p>Нет</p>
<p>Предусловие:</p> <ol style="list-style-type: none"> 1. Пользователь запустил систему 2. Пользователь авторизован

Продолжение таблицы 4

<p>Основной поток:</p> <ol style="list-style-type: none"> 1. Пользователь переходит на страницу настройки отображения 2. Пользователь выбирает параметры по своему усмотрению 3. Пользователь сохраняет настройки отображения 3. Пользователь видит измененное отображение <p>Альтернативные потоки:</p> <ol style="list-style-type: none"> 2.а Пользователь оставляет настройки отображения по умолчанию
<p>Постусловия:</p> <ol style="list-style-type: none"> 1. Пользователю постоянно доступно настроенное им отображение

В таблице представлен прецедент входа пользователя в систему. Пользователь взаимодействует с приложением через сценарий авторизации или регистрации. Основной поток показывает последовательность действий: переход на страницу входа, заполнение данных и получение доступа. Альтернативный поток (2.а) отражает вариант регистрации нового пользователя. Диаграмма наглядно отображает предусловия (запуск системы) и постусловия (успешный вход и доступ к функционалу).

Сценарий варианта использования «Получить показание датчика» представлен в таблице 5.

Таблица 5 – Вариант использования «Войти в систему»

Прецедент: Получить показание датчика
ID: 3
<p>Краткое описание:</p> <p>Пользователь получает показание интересующего датчика системы</p>
<p>Главные актеры:</p> <p>Пользователь</p>
<p>Второстепенные актеры:</p> <p>Нет</p>

Продолжение таблицы 5

Предусловие: 1. Пользователь запустил систему 2. Пользователь авторизован
Основной поток: 1. Пользователь переходит на страницу получения данных 2. Пользователь выбирает нужные датчики
Постусловия: 1. Пользователю доступно отображение показаний датчика в формате, приближенному к реальному времени

Данная таблица иллюстрирует процесс персонализации интерфейса пользователем. После авторизации пользователь может изменить параметры отображения данных согласно своим предпочтениям через специальный раздел настроек. Основной сценарий включает выбор и сохранение параметров, альтернативный вариант предполагает использование настроек по умолчанию. Успешное выполнение прецедента гарантирует постоянное применение выбранных параметров отображения для данного пользователя.

В результате анализа функциональных требований были выделены варианты использования программы. Для некоторых вариантов использования, которые являются основными для работы разрабатываемого программного средства, были представлены их спецификации, позволяющие отобразить последовательность действий пользователя и системы.

1.4 Анализ нефункциональных требований

Разрабатываемое web-приложение для контроля показаний датчиков умного дома строится на микросервисной архитектуре с использованием современных технологических решений.

Серверная часть реализована на Java (Spring boot) и включает четыре микросервиса: шлюза, работы с данными, управления уведомлениями и мониторинга. Каждый микросервис использует отдельную базу данных

PostgreSQL, что обеспечивает надежное хранение и быстрый доступ к данным. Взаимодействие между ними происходит через gRPC. Развертывание осуществляется с помощью Docker-контейнеров, что упрощает масштабирование и обновление системы

Использование PostgreSQL как базы данных обеспечивает высокую производительность, масштабируемость и надежность. JS, в сочетании с HTML и CSS, позволяет разработать эффективное и масштабируемое клиентское приложение. Микросервисная архитектура позволяет разделять функциональность приложения на независимые модули, что упрощает разработку, тестирование и обслуживание. Взаимодействие серверной и клиентской части приложения осуществляется через WebSocket [4].

Построение диаграмм осуществляется с помощью языка UML 2.0 в среде draw.io. Это бесплатный онлайн-сервис, позволяющий создавать диаграммы, схемы и прочие представления данных: диаграммы классов, последовательности, вариантов использования, развертывания и т. д. [5].

Разрабатываемое web-приложение должно быть оптимизировано для работы в современных браузерах, таких как Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge.

1.5 Планирование разработки и оценка бюджета

Эффективное управление проектом требует тщательного планирования сроков разработки и точного расчета бюджета, что является залогом успешной реализации программного продукта. В рамках данного проекта для организации работ была использована специализированная система управления проектами с открытым исходным кодом Project Libre, предоставляющая комплексный инструментарий для планирования, мониторинга и контроля всех этапов разработки.

После занесения данных об этапах создания приложения в программу было выполнено планирование разработки программных средств, а также была произведена оценка бюджета. Для визуального отображения проведения работ

используется диаграмма Ганта, отображающая вехи разработки программного обеспечения [20].

Полученная диаграмма Ганта, представлена на рисунке 10.

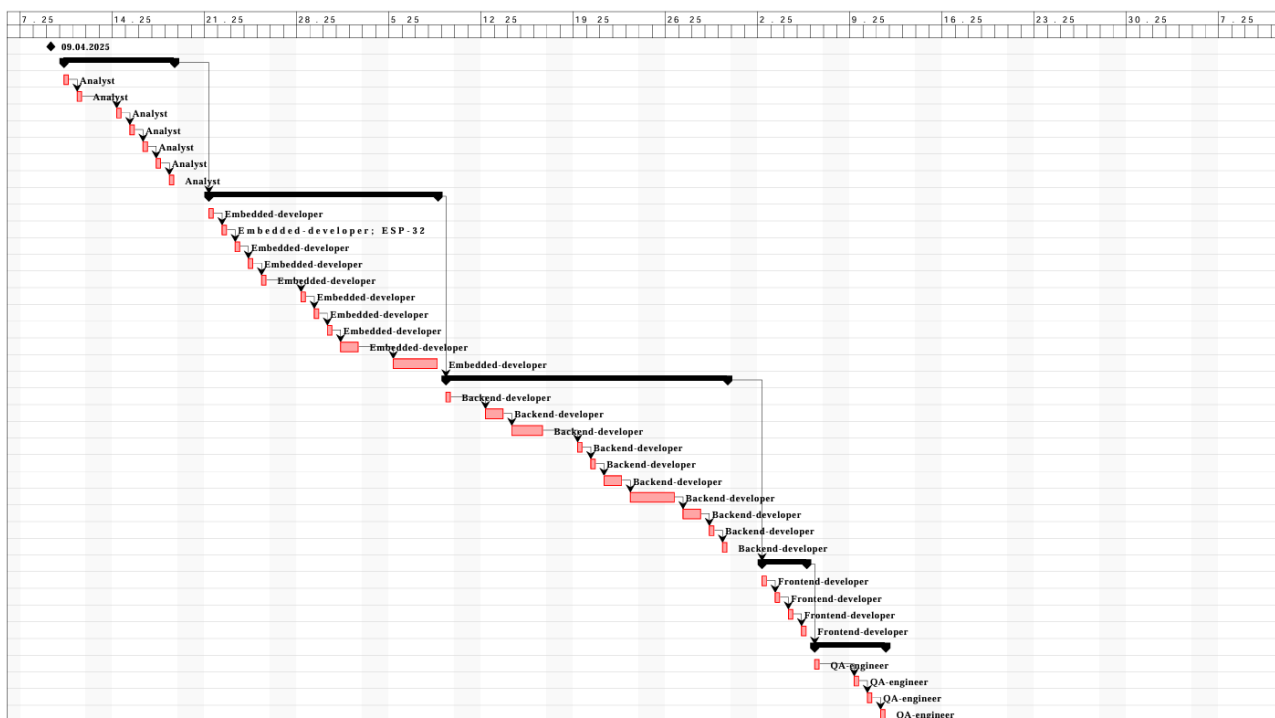


Рисунок 10 – Диаграмма Ганта для системы умного дома

При планировании разработки было выделено 41 задача и 5 этапов. За дату начала работ было взято 9 апреля 2025. Согласно полученной диаграмме, окончание работ по проекту запланировано на 13 июня. Такой подход обеспечивает четкое соблюдение сроков и позволяет контролировать прогресс на каждом этапе, что позволило структурировать процесс и обеспечить последовательное выполнение работ. Анализ данных отчета позволяет своевременно корректировать планы и обеспечивать выполнение проекта в установленные сроки без потери качества.

Для оценки занятости трудовых ресурсов и этапов производства, программой был сгенерирован отчет, представленный на рисунке 11, наглядно демонстрирующий распределение задач между участниками проекта, что помогает избежать перегрузки и оптимизировать рабочий процесс.

Код	Название	Продолжите	Начало	Окончание	Предшествующие	Название ресурса
1	Начало проекта	0 дней?	09.04.2025, 08:00	09.04.2025, 08:00		
2	Анализ предметной области	7 дней?	10.04.2025, 08:00	18.04.2025, 17:00		
3	Анализ рынка	1 день?	10.04.2025, 08:00	10.04.2025, 17:00		Analyst
4	Анализ технологических	1 день?	11.04.2025, 08:00	11.04.2025, 17:00	3	Analyst
5	Цели и задачи сравнительного	1 день?	14.04.2025, 08:00	14.04.2025, 17:00	4	Analyst
6	Изучение систем-аналогов	1 день?	15.04.2025, 08:00	15.04.2025, 17:00	5	Analyst
7	Анализ функциональных	1 день?	16.04.2025, 08:00	16.04.2025, 17:00	6	Analyst
8	Анализ нефункциональных	1 день?	17.04.2025, 08:00	17.04.2025, 17:00	7	Analyst
9	Планирование разработки и	1 день?	18.04.2025, 08:00	18.04.2025, 17:00	8	Analyst
10	Проектирование аппаратного	14 дней?	21.04.2025, 08:00	08.05.2025, 17:00	2	
11	Оценка процесса разработки	1 день?	21.04.2025, 08:00	21.04.2025, 17:00		Embedded-developer
12	Создание аппаратной части	1 день?	22.04.2025, 08:00	22.04.2025, 17:00	11	Embedded-developer, Модуль
13	Программирование датчиков	1 день?	23.04.2025, 08:00	23.04.2025, 17:00	12	Embedded-developer
14	Программирование датчика	1 день?	24.04.2025, 08:00	24.04.2025, 17:00	13	Embedded-developer
15	Программирование датчика	1 день?	25.04.2025, 08:00	25.04.2025, 17:00	14	Embedded-developer
16	Программирование датчика	1 день?	28.04.2025, 08:00	28.04.2025, 17:00	15	Embedded-developer
17	Программирование датчика	1 день?	29.04.2025, 08:00	29.04.2025, 17:00	16	Embedded-developer
18	Программирование датчика СО	1 день?	30.04.2025, 08:00	30.04.2025, 17:00	17	Embedded-developer
19	Разработка программного	2 дней?	01.05.2025, 08:00	02.05.2025, 17:00	18	Embedded-developer
20	Разработка программного	4 дней?	05.05.2025, 08:00	08.05.2025, 17:00	19	Embedded-developer
21	Проектирование серверной	16 дней?	09.05.2025, 08:00	30.05.2025, 17:00	10	
22	Оценка процесса разработки	1 день?	09.05.2025, 08:00	09.05.2025, 17:00		Backend-developer
23	Выбор технологий реализации	2 дней?	12.05.2025, 08:00	13.05.2025, 17:00	22	Backend-developer
24	Разработка серверной	3 дней?	14.05.2025, 08:00	16.05.2025, 17:00	23	Backend-developer
25	Разработка API системы	1 день?	19.05.2025, 08:00	19.05.2025, 17:00	24	Backend-developer
26	Разработка баз данных	1 день?	20.05.2025, 08:00	20.05.2025, 17:00	25	Backend-developer
27	Разработка сервиса "Шлюз"	2 дней?	21.05.2025, 08:00	22.05.2025, 17:00	26	Backend-developer
28	Разработка сервиса "Логика"	2 дней?	23.05.2025, 08:00	26.05.2025, 17:00	27	Backend-developer
29	Разработка сервиса	2 дней?	27.05.2025, 08:00	28.05.2025, 17:00	28	Backend-developer
30	Разработка сервиса	1 день?	29.05.2025, 08:00	29.05.2025, 17:00	29	Backend-developer
31	написание документации	1 день?	30.05.2025, 08:00	30.05.2025, 17:00	30	Backend-developer
32	Проектирование клиентской	4 дней?	02.06.2025, 08:00	05.06.2025, 17:00	21	
33	Проектирование UI/UX	1 день?	02.06.2025, 08:00	02.06.2025, 17:00		Frontend-developer
34	Разработка двунаправленного	1 день?	03.06.2025, 08:00	03.06.2025, 17:00	33	Frontend-developer
35	Интеграция с API сервера	1 день?	04.06.2025, 08:00	04.06.2025, 17:00	34	Frontend-developer
36	Разработка и интеграция	1 день?	05.06.2025, 08:00	05.06.2025, 17:00	35	Frontend-developer
37	Тестирование системы	4 дней?	06.06.2025, 08:00	11.06.2025, 17:00	32	
38	Тестирование и оценка	1 день?	06.06.2025, 08:00	06.06.2025, 17:00		QA-engineer
39	Тестирование и оценка	1 день?	09.06.2025, 08:00	09.06.2025, 17:00	38	QA-engineer
40	Тестирование и оценка	1 день?	10.06.2025, 08:00	10.06.2025, 17:00	39	QA-engineer
41	Итоговая проверка системы в	1 день?	11.06.2025, 08:00	11.06.2025, 17:00	40	QA-engineer

Рисунок 11 – Отчет этапов производства

По данным отчета об использовании трудовых ресурсов можно сделать вывод, что общая занятость трудовых ресурсов составляет 360 часа с 09.04.2025 по 11.06.2025 с учетом пятидневной рабочей недели. Такой подход обеспечивает четкое соблюдение сроков и позволяет контролировать прогресс на каждом этапе.

Для оценки финансовых затрат необходимо назначить цены ресурсов. Стоимость используемых ресурсов представлена на рисунке 12.

Название	Тип	Ед. изм. материалов	Инициалы	Максимальные единицы	Стандартная ставка
Backend-developer	Работа		B	100 %	250,00 Р/час
Frontend-developer	Работа		F	100 %	250,00 Р/час
Embedded-developer	Работа		E	100 %	250,00 Р/час
Модуль ESP-32 с датчиками	Материал	шт	Esp		2000,00 Р
Analyst	Работа		A	100 %	200,00 Р/час
QA-engineer	Работа		T	100 %	150,00 Р/час

Рисунок 12 – Стоимость задействованных ресурсов

По отчету об общих затратах проекта можно сделать вывод, что для разработки программных средств для системы контроля за показаниями датчиков умного дома потребуется 86000 рублей.

2 Проектирование программных средств для системы контроля показаний датчиков умного дома

2.1 Архитектура приложения

Архитектура программных средств для системы умного дома представляет собой микросервисную архитектуру в сочетании с шаблоном проектирования MVC. Контроллер и модель реализованы внутри микросервиса шлюза, он отвечает за перенаправление и валидации запросов от пользователя по микросервисам. Клиентское приложение обеспечивает отображение данных и служит основным интерфейсом для получения команд от пользователей, которые впоследствии направляются на сервер посредством WebSocket [6].

На рисунке 13 и 14 изображены диаграмма развертывания сервиса и схема взаимодействия подсистем, которые отображают архитектуру программных средств для системы контроля показаний датчиков умного дома.

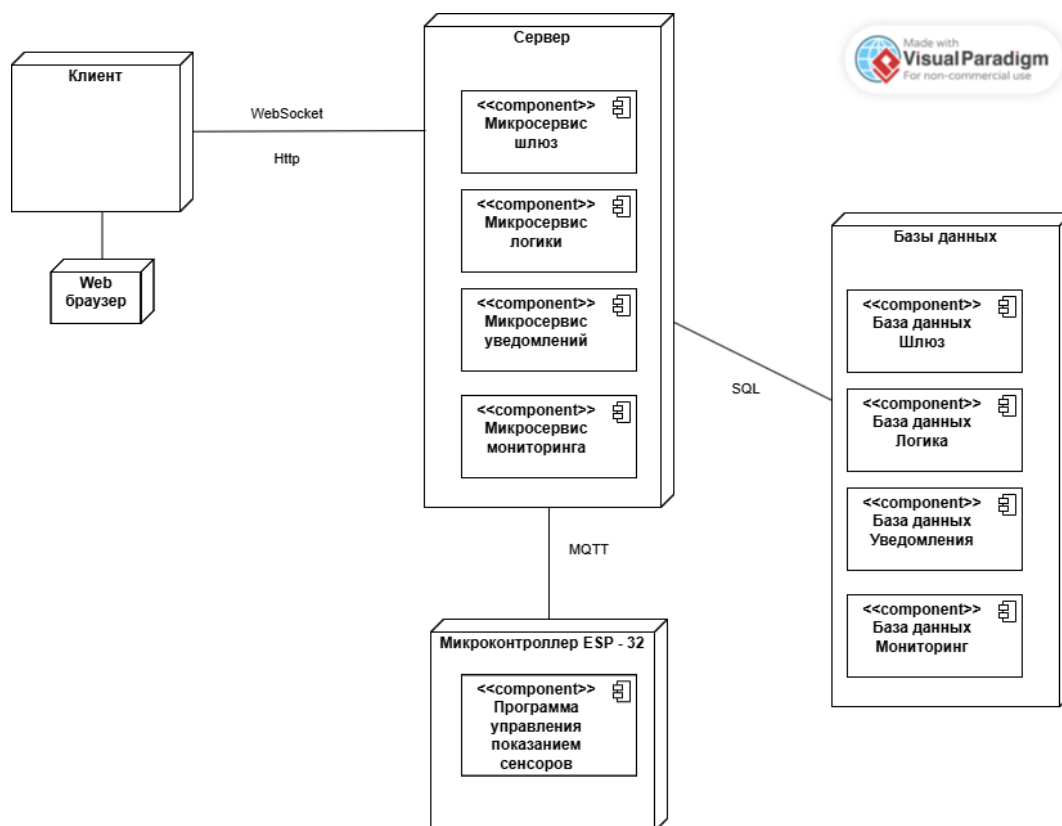


Рисунок 13 – Диаграмма развертки системы считывания показаний датчиков умного дома

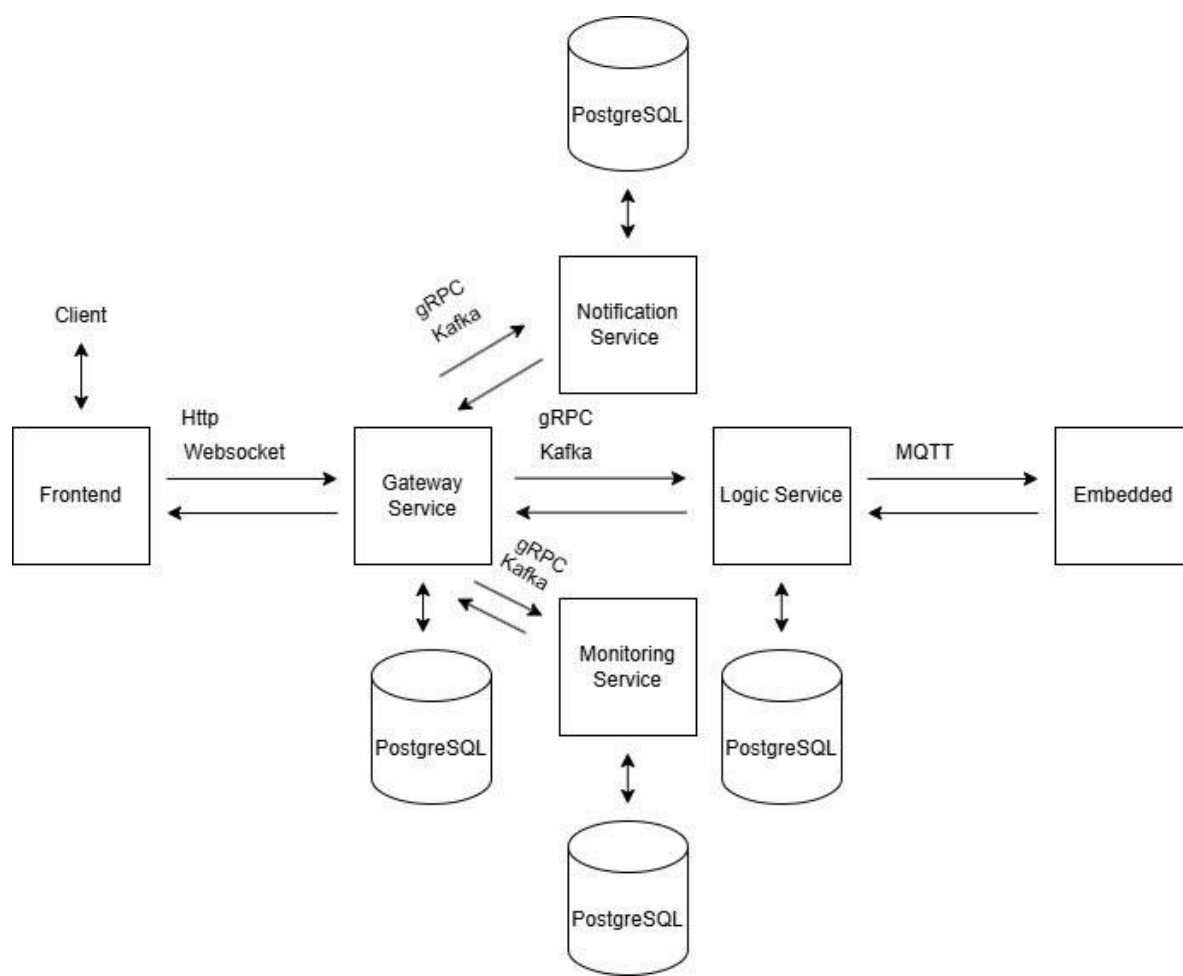


Рисунок 14 – Схема взаимодействия подсистем считывания показаний датчиков умного дома

Серверная часть приложения представлена в виде четырех независимых микросервисов, каждый из которых имеет свой функционал, отвечающий за определенную часть целостного приложения, каждый сервис взаимодействует с отдельной базой данных. Микросервисы могут обращаться друг к другу через API для совместной обработки данных. Таким образом, серверная часть выступает промежуточным слоем между запрашиваемыми ресурсами, базами данных и аппаратной частью [7].

После получения данных микросервисом от БД, он их анализирует, создает ответ в виде Proto или сообщением брокера, и отправляет его микросервису шлюзу, тот в свою очередь отправляет клиенту информацию в формате JSON. Этот процесс подчеркивает адаптивность и возможность

масштабирования системы, обеспечивая автономную и эффективную работу каждого из элементов.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель – это модель, отображающая данные и связи предметной области с учетом их семантического содержания, независимо от представления в компьютере [8].

Особенностью концептуального моделирования является его ориентация на содержательную интерпретацию данных.

Для составления концептуальной модели данных требуется выделить список всех основных сущностей предметной области разрабатываемого приложения. Описание сущностей предметной области представлено в таблице 5.

Таблица 5 – Сущности БД

Название сущности	Идентификатор сущности	Назначение сущности
Пользователь	User	Содержит сведения о пользователе
Девайс	Devices	Содержит сведения о устройстве
Информация датчика	SensorData	Содержит информацию о показании датчика
Пользовательские настройки	UserCardSettings	Содержит информацию о пользовательских настройках отображения
Сервис	Service	Содержит сведения о сервисе системы
Состояние сервиса	ServiceHealthLog	Содержит сведения о логах сервиса
Состояние девайса	DeviceHealth	Содержит сведения логах и состоянии девайса
Уведомление	Notification	Содержит информацию о уведомлении
Настройки уведомления	NotificationSettings	Содержит сведения о настройках, необходимых для отправки уведомления

Графическое представление концептуальных моделей данных каждого микросервиса представлены на рисунке 15.

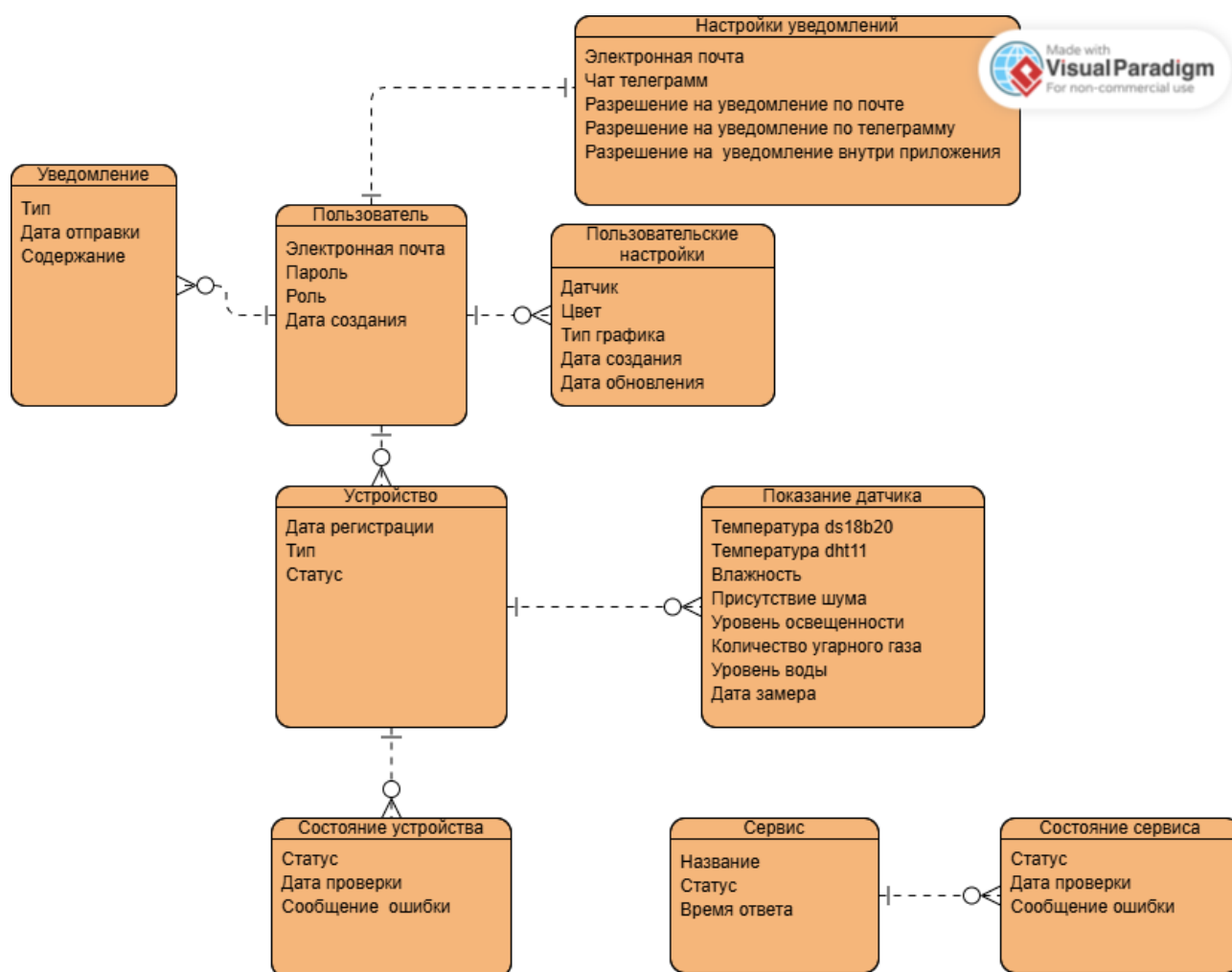


Рисунок 15 – Концептуальная модель базы данных предметной области
«Система показаний датчиков умного дома»

Данное представление построено в среде Visual Paradigm 17.0 в нотации IDEF1X.

Наборы атрибутов для каждой сущности предметной области и их характеристики приведены в приложении Б.

2.2.2 Логические модели баз данных

В разрабатываемых программных средствах для контроля за показаниями датчиков умного дома была выделена разработка четырех баз данных: «Шлюз», «Логика», «Мониторинг», «Уведомления».

Разделение было произведено по следующим причинам:

- разделяя данные по функциональным областям, удобнее управлять их хранением и обработкой;
- использование нескольких баз данных позволяет сбалансировать нагрузку;
- разделение данных по нескольким БД позволяет эффективнее организовывать резервное копирование и восстановление данных.

Общая логическая модель проектируемых баз данных представлена на рисунке 16.

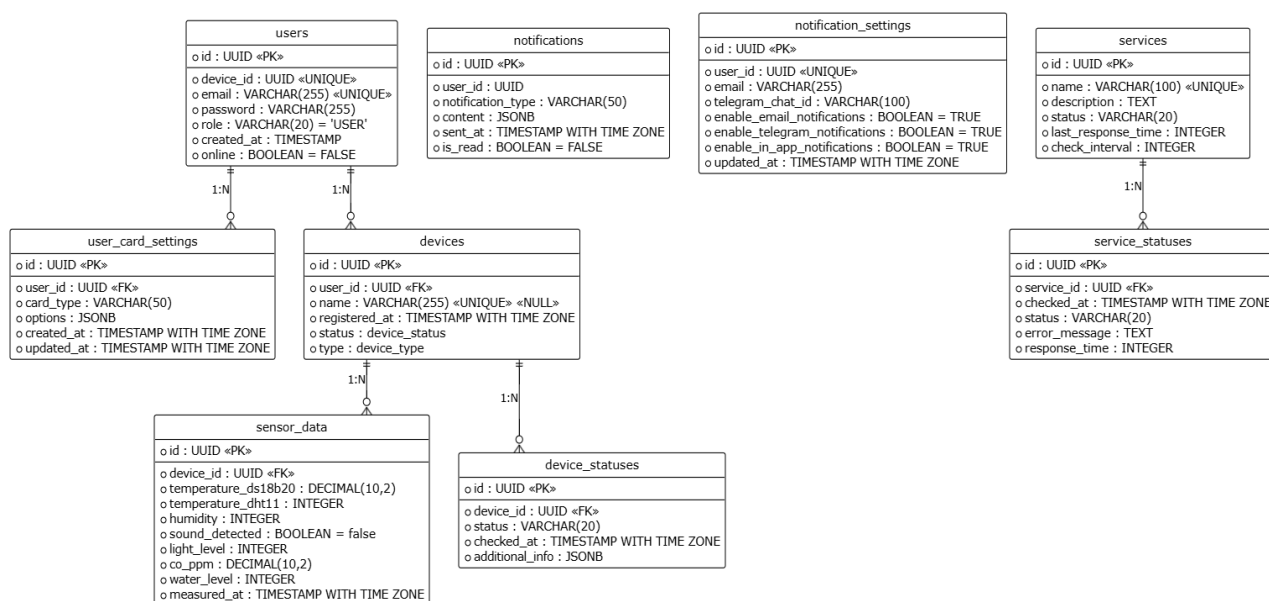


Рисунок 16 – Логическая модель базы данных всей системы

Логическая схема БД сервиса пользователей отражает ключевые сущности для управления учетными записями и персональными настройками. Основная таблица `users` содержит базовые атрибуты пользователей, включая уникальные идентификаторы и учетные данные. Таблица `user_card_settings` реализует отношение «один-ко-многим» с пользователями, храня персонализированные настройки интерфейса в формате JSON. Схема обеспечивает целостность данных за счет внешнего ключа и уникальных ограничений, что соответствует принципам реляционного моделирования. Логическая модель базы данных сервиса «Шлюз» продемонстрирована на рисунке 17.

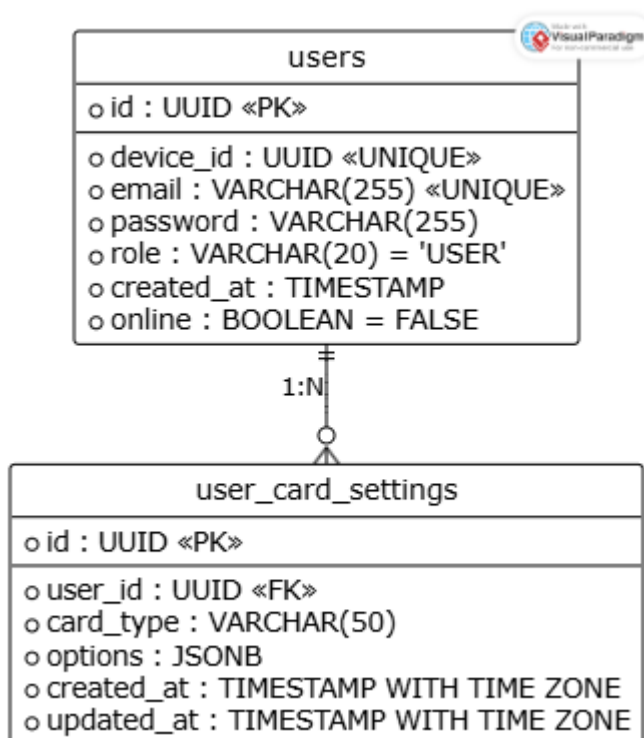


Рисунок 17 – Логическая модель базы данных сервиса Шлюз

Логическая модель БД для сервиса «Логика» построена вокруг двух взаимосвязанных сущностей. Таблица `devices` регистрирует метаданные подключенных устройств с поддержкой различных статусов и типов. Таблица `sensor_data` агрегирует показания датчиков по принципу «один-ко-многим» (одно устройство – множество измерений). Схема предусматривает индексацию по полю `device_id` и временным меткам для оптимизации запросов исторических данных, сохраняя при этом нормализованную структуру. Логическая модель данного сервиса отображена на рисунке 18. Для повышения производительности запросов, особенно при работе с историческими данными, в схеме предусмотрена индексация по полю `device_id` и временным меткам. Это позволяет быстро фильтровать и агрегировать данные, не нарушая нормализованную структуру базы. Логическая модель, представленная на рисунке 18, наглядно отображает взаимосвязи между сущностями и ключевые атрибуты, обеспечивая прозрачность проектирования и удобство дальнейшей разработки.

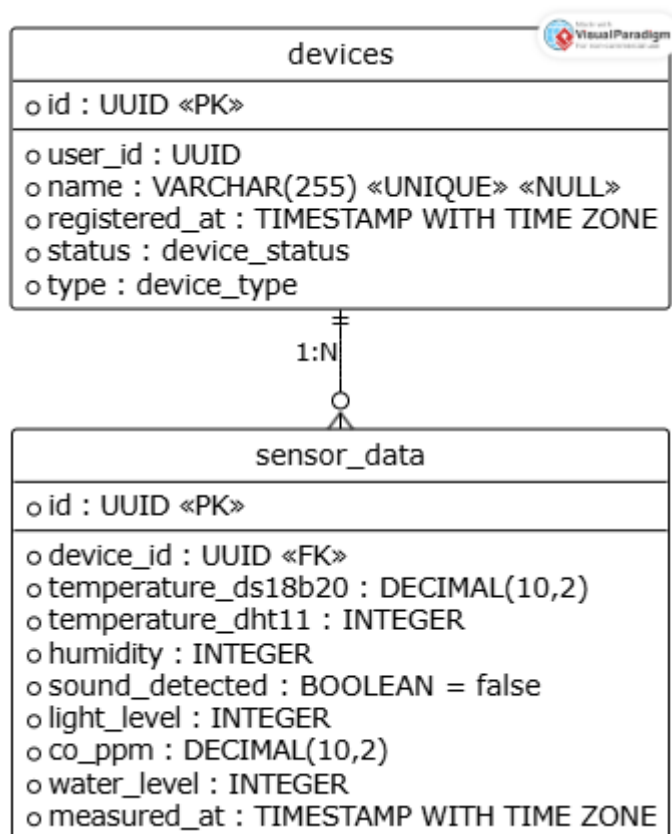


Рисунок 18 – Логическая модель базы данных сервиса Логики

Логическая структура БД сервиса «Уведомления» разделена на управление настройками и обработку событий. Таблица `notification_settings` централизует пользовательские предпочтения для различных каналов оповещения с гарантией уникальности через ограничение на `user_id`. Таблица `notifications` фиксирует факт отправки сообщений без жесткой привязки к другим микросервисам. Отсутствие явных внешних ключей в схеме обусловлено распределенной природой системы, при этом сохраняется логическая связь через общие идентификаторы пользователей. Логическая схема представлена на рисунке 19. Несмотря на отсутствие явных внешних ключей, связь между таблицами поддерживается через единые идентификаторы пользователей, что обеспечивает целостность данных без потери гибкости. Логическая схема, представленная на рисунке 19, отражает эту структуру, демонстрируя баланс между нормализацией данных и требованиями распределенной системы. Подобная архитектура позволяет сервису

«Уведомления» эффективно интегрироваться с другими компонентами платформы, обеспечивая надежную и своевременную доставку сообщений пользователям.

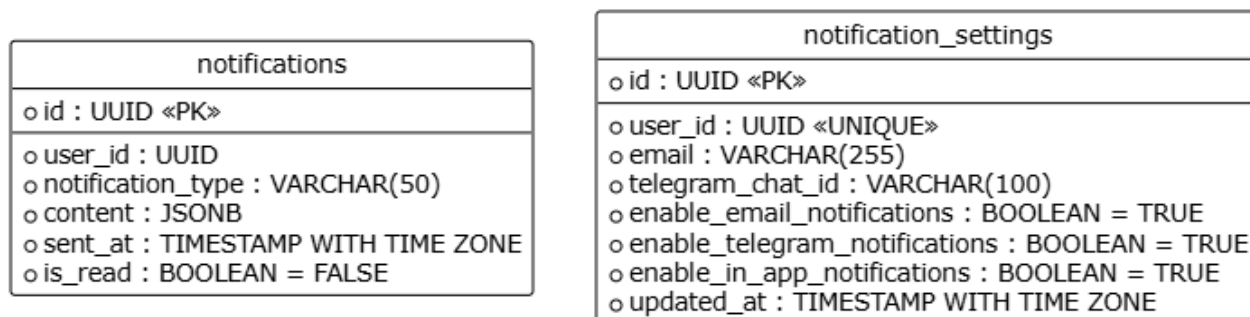


Рисунок 19 – Логическая модель базы данных сервиса Уведомления

Логическая схема мониторинга реализует трекинг состояния компонентов системы. Центральная таблица `services` описывает наблюдаемые сервисы, а `service_statuses` фиксирует их историю состояний по принципу «один-ко-многим». Отдельная таблица `device_statuses` логически связана с сервисом устройств через общие идентификаторы, но без декларативного внешнего ключа. Схема предусматривает временную индексацию для эффективного анализа временных рядов и выявления аномалий в работе системы. Логическая модель указана на рисунке 20.

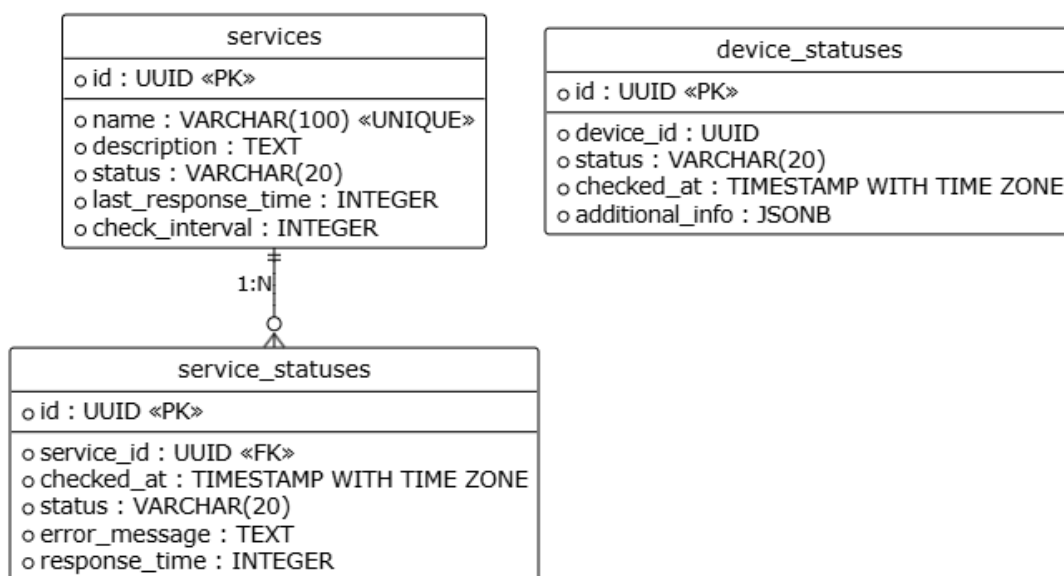


Рисунок 20 – Логическая модель базы данных сервиса Мониторинг

Анализ функциональных зависимостей показал, что база данных находится в третьей нормальной форме, так как все таблицы содержат атомарные значения и нет ни одного неключевого атрибута, зависящего от другого неключевого атрибута, который, в свою очередь, зависит от первичного ключа.

2.3 Проектирование пользовательского интерфейса

Интерфейс web-приложения должен отвечать всем функциональным требованиям, а также должен быть удобным в использовании.

Проектируемый интерфейс включает в себя страницы входа, регистрации и главной страницы с навигационным меню по приложению. На главной странице должны располагаться настраиваемые объекты отображения данных сенсоров.

Кроме вкладок и основных страниц в приложении должны быть реализованы страница личного кабинета, страница получения отчетов, страница мониторинга состояния модулей, страница получения отчетов и рекомендательной информации.

Шаблон визуального представления информации с датчиков приведен на рисунке 21.

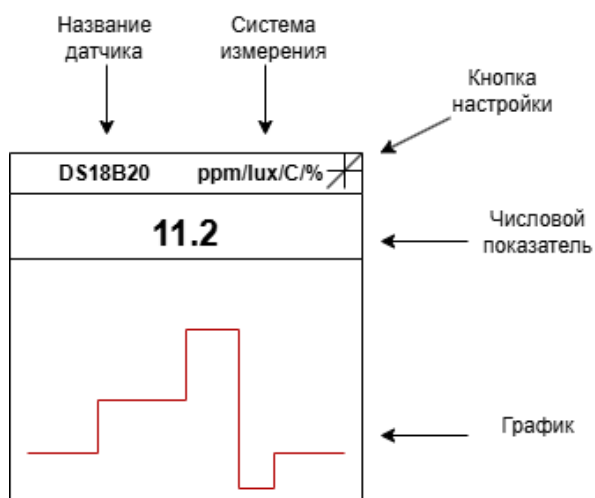


Рисунок 21 – Шаблон визуального представления информации с датчиков

Каждая составляющая интерфейса будет реализована с использованием компонентов стандартных библиотек языка программирования JavaScript, а также вспомогательном шаблонизаторе Spring Thymeleaf. Применение подобной архитектуры упрощает расширение приложения и использование кода повторно [10].

2.4 Проектирование классов программных средств для контроля за показаниями датчиков умного дома

В ходе проектирования серверной части приложения, для каждого из микросервисов были разработаны диаграммы классов согласно шаблону проектирования MVC.

Диаграммы классов для микросервисов «Мониторинга», «Шлюза», «Логики» и «Уведомлений» представлены на рисунках Б.1, Б.2, Б.3 и Б.4 в приложении Б.

Краткое описание функционального назначения наиболее значимых классов микросервиса «Планирование» приведено в таблице 6.

Таблица 6 – Компоненты диаграммы классов микросервиса «Шлюз»

Компонент	Назначение
DiplomaGatewayApplication	Точка входа в программу
MainController	Класс, реализующий обработку пользовательских запросов, взаимодействие между моделями и представлениями.
MqttController	Класс, представляющий логику взаимодействия запросов через брокер.
UserController	Класс, представляющий логику взаимодействия запросов пользователей
UserRepository	Класс, содержащий механизмы для обработки данных, связанных с пользователем.
UserService	Класс, представляющий бизнес логику работы с пользователем.

Продолжение таблицы 6

Компонент	Назначение
SensorDataMapper	Класс, предоставляющий логику преобразования данных сенсоров из разных форматов.
LogicService	Класс, представляющий логику трансляции запросов от класса контроллера на другой микросервис посредством gRPC.

Компоненты диаграммы классов микросервисов «Логика», «Мониторинга» и «Уведомлений» имеют идентичное функциональное назначение классов и придерживаются такой же логики построения программного кода сервисов, как и в микросервисе «Шлюз».

3 Реализация программных средств для системы контроля за показаниями датчиков умного дома

3.1 Реализация базы данных

Физическая модель базы данных — это самое детальное описание ее реализации, включая типы данных, индексы, физическое размещение файлов.

На основе концептуальной модели базы данных и логических были построены физические модели базы данных для баз, использующихся в разрабатываемом программном средстве [11].

Физическая модели баз данных «Шлюз», «Мониторинг», «Логика» и «Уведомления» были построены в среде PgAdmin 4 и представлены на рисунке 22.

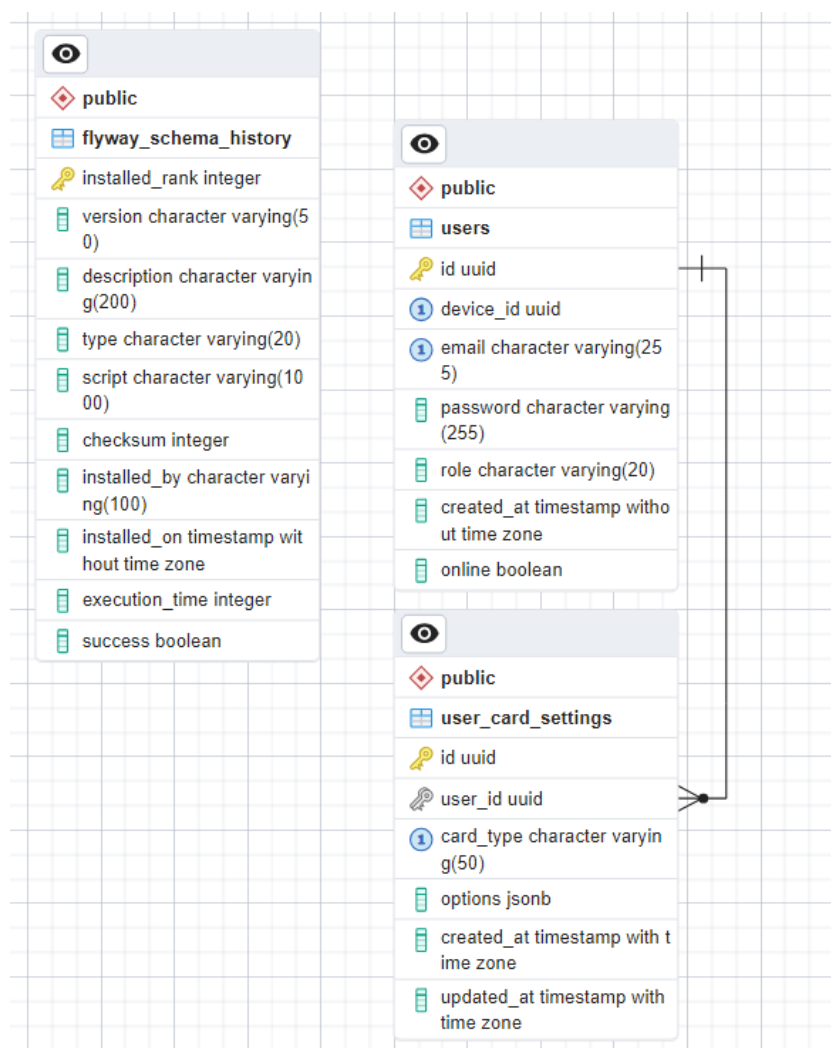


Рисунок 22 – Физическая модель базы данных «Шлюз»

База данных, соответствующая разработанной физической модели, была развернута с помощью механизма миграций. Механизм миграций необходим для версионирования и синхронизации структуры БД. Физическая модель «Логика» представлен на рисунке 23.

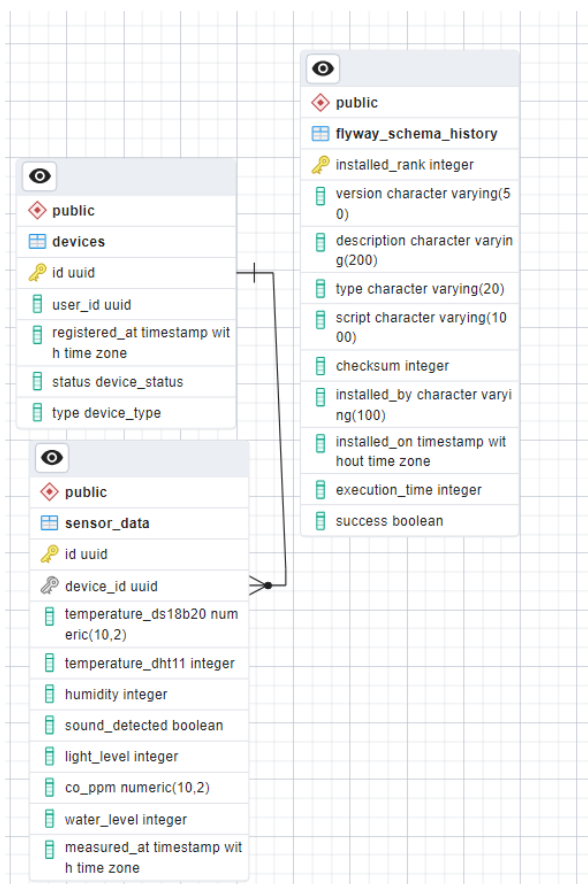


Рисунок 23 – Физическая модель базы данных «Логика»

Разработанная база данных, соответствующая физической модели, была успешно развернута с использованием механизма миграций. Миграции обеспечивают контроль версий структуры БД, позволяя последовательно применять изменения схемы данных, сохраняя ее согласованность на разных этапах разработки и эксплуатации системы. Данный подход гарантирует надежное управление эволюцией базы данных, включая добавление новых таблиц, модификацию существующих полей и настройку индексов без потери целостности данных. Реализованный механизм обеспечивает прозрачность изменений, позволяя отслеживать историю модификаций и при необходимости откатывать обновления. Модель «Мониторинг» представлена на рисунке 24.

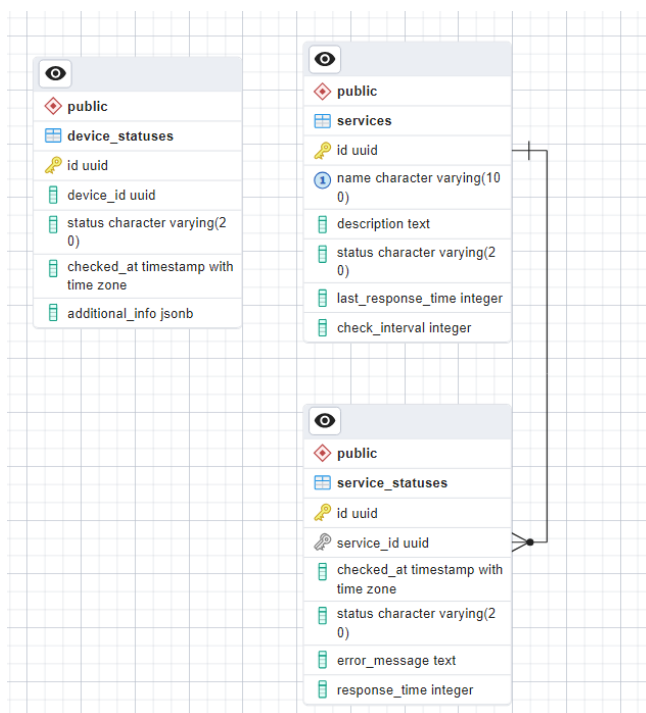


Рисунок 24 – Физическая модель базы данных «Мониторинг»

Модель «Уведомления» представлена на рисунке 25.

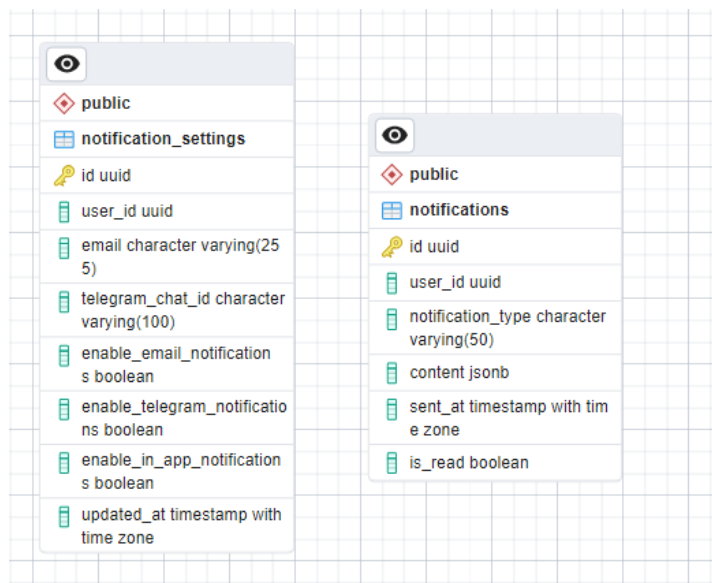


Рисунок 25 – Физическая модель базы данных «Уведомления»

В данной системе каждый микросервис обладает собственной изолированной базой данных, что обеспечивает независимость и отказоустойчивость компонентов. Базы данных синхронизируются через общие идентификаторы, позволяя сервисам согласованно работать с данными, не нарушая принципы слабой связанности. Например, сервис шлюза фиксирует

сырые данные с датчиков, сервис логики обрабатывает и сохраняет аналитику, сервис мониторинга отслеживает метрики работы, а сервис уведомлений использует информацию о пользователях для отправки предупреждений – все это происходит через скоординированное взаимодействие независимых БД. Такая архитектура гарантирует масштабируемость: каждый сервис может развиваться и менять схему своей базы, не затрагивая другие компоненты системы.

3.2 Реализация серверной части приложения

Серверная часть программных средств для системы получения данных датчиков умного дома разрабатывалась в среде программирования IntelliJ Idea, на языке программирования Java. Код серверной части приложения приведен в приложении Г.

Система реализована как набор из четырех микросервисов, где каждый сервис включает модели данных, отражающие структуру базы, и сериализаторы для конвертации данных при обмене между клиентом и сервером. Для безопасного и гибкого управления изменениями структуры данных предусмотрены механизмы миграций, которые автоматизируют процесс обновления схемы, минимизируют риски потери информации и упрощают развертывание новых версий в рабочей среде, обеспечивая бесперебойную адаптацию системы к эволюционирующим требованиям.

Разработанные сервисы предоставляют REST API, WebSocket и gRPC для взаимодействия. Реализованные методы описаны в таблице 7.

Таблица 7 – Методы API программных средств для контроля за показаниями датчиков умного дома

URL запроса	Http метод	Описание
/main	GET	Получение главной страницы приложения
/api/auth	POST	Авторизация пользователя

Продолжение таблицы 7

URL запроса	Http метод	Описание
/api/register	POST	Регистрация пользователя
URL запроса	Http метод	Описание
/statistic	GET	Получение страницы статистики
/api/sensor-data/filter	GET	Получение списка показаний датчиков по фильтрации
/api/sensor-data/report	GET	Получение сгенерированного отчета по нужным параметрам
/api/users	GET	Получение списка пользователей
/api/users/register/device	POST	Зарегистрировать устройство
/api/user-settings/{userId}	GET	Получить настройки пользователя
/api/user-settings/{userId}/{cardType}	POST	Сохранить настройки пользователя
/topic/sensor-data	WS	Получение данных датчиков с сервера
SaveSensorData	RPC	Сохранение данных
SaveAggregatedSensorData	RPC	Сохранение агрегированных данных
GetSensorData	RPC	Получение данных
RegisterDevice	RPC	Регистрация устройства
GenerateReport	RPC	Генерация отчета

Разработанные микросервисы обернуты в docker контейнеры, это сделано с целью дальнейшего разворачивания приложения на других серверах с платформами, поддерживаемыми docker [12]. Код развертки сервиса шлюза в контейнер представлен в листинге 1. Полный код развертки всех сервисов, СУБД и брокеров приведены в приложении Б.

Листинг 1 – Код развертки приложения

```
version: '3.8'
services:
  diploma-gateway:
    build:
      context: ../diploma-gateway
      dockerfile: Dockerfile
    container_name: diploma-gateway
```

```

    ports:
      - "8080:8080"
      - "9091:9091"
    environment:
      -
    SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/diploma-
gateway
      - SPRING_DATASOURCE_USERNAME=postgres
      - SPRING_DATASOURCE_PASSWORD=root
      - MQTT_BROKER_URL=tcp://mqtt-broker:1883
    depends_on:
      postgres:
        condition: service_healthy
      mqtt-broker:
        condition: service_started
    networks:
      - diploma-network
volumes:
  postgres_data:
  grafana_data:

networks:
  diploma-network:
    driver: bridge

```

Конфигурация взаимодействия между всеми компонентами системы описана в файле `docker-compose.yml`, который обеспечивает согласованное развертывание всей инфраструктуры. Процесс начинается с инициализации PostgreSQL-контейнера, где создаются отдельные базы данных для каждого микросервиса (`diploma-gateway`, `diploma-logic`, `diploma-notification` и `diploma-monitoring`) с едиными учетными данными доступа. Параллельно запускается MQTT-брокер на базе Eclipse Mosquitto для обработки сообщений от IoT-устройств. После успешного старта этих сервисов последовательно развертываются микросервисы: API-шлюз (`diploma-gateway`), обрабатывающий входящие запросы и взаимодействующий с MQTT; сервис бизнес-логики (`diploma-logic`) для анализа данных; `notification-сервис` (`diploma-notification`), отвечающий за рассылку уведомлений через почту, Telegram и внутри приложения; а также `monitoring-сервис` (`diploma-monitoring`) для отслеживания состояния системы. Для обеспечения полноценного мониторинга дополнительно поднимаются контейнеры с Prometheus (сбор метрик), Grafana

(визуализация), Loki (логи) и Promtail (отправка логов в Loki), которые автоматически настраиваются на сбор данных со всех микросервисов через их actuator-эндпоинты. Сетевой мост diploma-network обеспечивает изолированное взаимодействие между контейнерами, а healthcheck-и PostgreSQL гарантируют запуск микросервисов только после готовности базы данных. Все сервисы используют единую схему именования и стандартные порты, что упрощает масштабирование и диагностику системы.

3.3 Реализация клиентской части приложения

Клиентская часть программных средств для системы контроля за показаниями датчиков умного дома разрабатывалась в среде программирования IntelliJ Idea, на языке программирования JavaScript, языке разметки HTML и языке стилей CSS. Код клиентской части приложения приведен в приложении Д.

На рисунке 26 представлена диаграмма компонентов клиентской части разрабатываемого приложения.

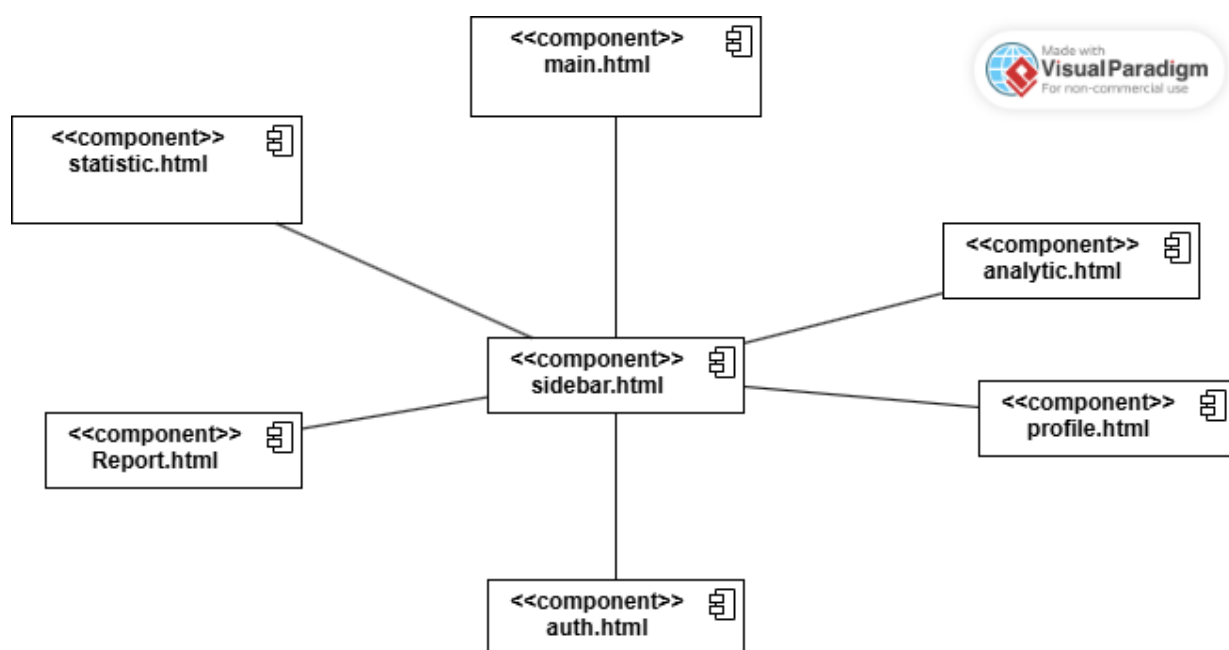


Рисунок 26 – Диаграмма компонентов клиентской части приложения

Описание компонентов клиентской части приложения представлено в таблице 8.

Таблица 8 – Компоненты клиентской части приложения

Компонент	Назначение
Main.html	Главная страница приложения
Sidebar.html	Компонент, содержащий перенаправление и навигацию через страницы
Statistic.html	Страница получения статистики
Report.html	Страница генерации отчетов
Компонент	Назначение
Profile.html	Страница профиля пользователя, с его настройками
Analytic.html	Страница получения аналитики, советов и заключений по данным
Auth.html	Страница авторизации и регистрации

Для передачи данных между сервером и клиентом используются стандартные функции языка JS, а также библиотеки axios, stompjs и sockjs. Выбор данных библиотек обусловлен удобством использования и быстроты выполнения.

3.4 Реализация интерфейса приложения

В соответствии с требованиями к пользовательскому интерфейсу, предоставленными в пункте 2.3, были реализованы следующие страницы web-приложения для системы показаний датчиков умного дома:

- страница авторизации;
- страница регистрации;
- страница личного кабинета пользователя;
- главная страница приложения;
- страница получения статистики;
- страница генерации отчетов;
- страница получения аналитики, советов и заключений по данным.

При открытии приложения пользователь попадает на страницу авторизации, представленную на рисунке 27.

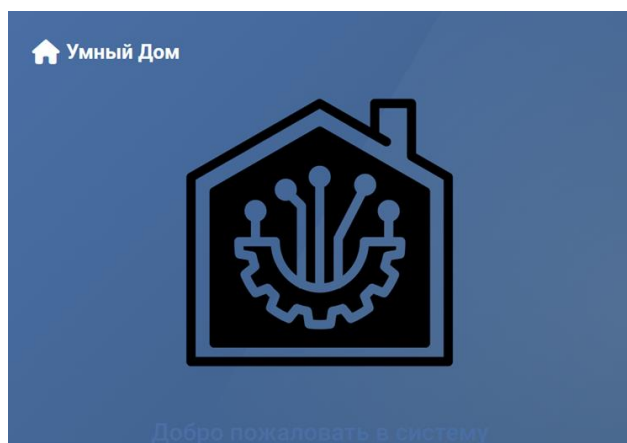


Рисунок 27 – Страница авторизации

Основные элементы страницы авторизации:

- 1 – поле ввода почты пользователя;
- 2 – поле ввода пароля пользователя;
- 3 – поле выбора запоминания пользователя;
- 4 – кнопка входа;
- 5 – кнопка перехода на страницу регистрации.

При нажатии на кнопку «Войти» система проверяет правильность введенных данных пользователем и проверяет наличие зарегистрированного пользователя в системе. Если обработка данных произведена успешно, то пользователь переходит на главную страницу. При активации кнопки входа система выполняет валидацию введенных данных, включая проверку формата электронной почты и соответствие пароля требованиям безопасности. Далее осуществляется поиск пользователя в базе данных и сверка учетных данных. В случае успешной аутентификации происходит перенаправление на главную страницу приложения. При обнаружении ошибок (неверный логин/пароль или отсутствующая учетная запись) пользователь получает соответствующее уведомление с рекомендациями по устранению проблемы.

Страница регистрации, на которую пользователь может перейти со страницы авторизации, представлена на рисунке 28.

Умный Дом

Добро пожаловать в систему управления умным домом

Управляйте всеми устройствами вашего дома из одного места

Регистрация

Email
Введите ваш email

Пароль
Создайте пароль

Подтвердите пароль
Повторите пароль

ID устройства
Введите ID вашего устройства

☐ Я согласен с [условиями использования](#)

☐ Я согласен с [условиями использования](#)

[Зарегистрироваться](#)

Уже есть аккаунт? [Войти](#)

Рисунок 28 – Страница регистрации

Основные элементы страницы регистрации:

- 1 – поле ввода электронной почты пользователя;
- 2 – поле повторного ввода электронной почты пользователя;
- 3 – поле ввода даты рождения пользователя;
- 4 – поле ввода идентификатора устройства пользователя;
- 5 – флаг выбора согласия с условиями пользования;
- 6 – флаг выбора согласия с условиями пользования;
- 7 – кнопка возврата на страницу входа.

При нажатии на кнопку «Зарегистрироваться» система проверяет правильность введенных данных пользователем, а также проверяет наличие пользователя с таким логином в системе, если данного пользователя в системе еще нет и данные корректны, то пользователь заносится в систему и происходит переход на страницу авторизации. При обнаружении ошибок (неверный логин/пароль или отсутствующая учетная запись) пользователь получает соответствующее уведомление с рекомендациями по устранению проблемы. Реализованный механизм обеспечивает надежную защиту от несанкционированного доступа, сохраняя при этом удобство взаимодействия.

Страница личного кабинета, на которую можно попасть после авторизации в приложении представлена на рисунке 29.

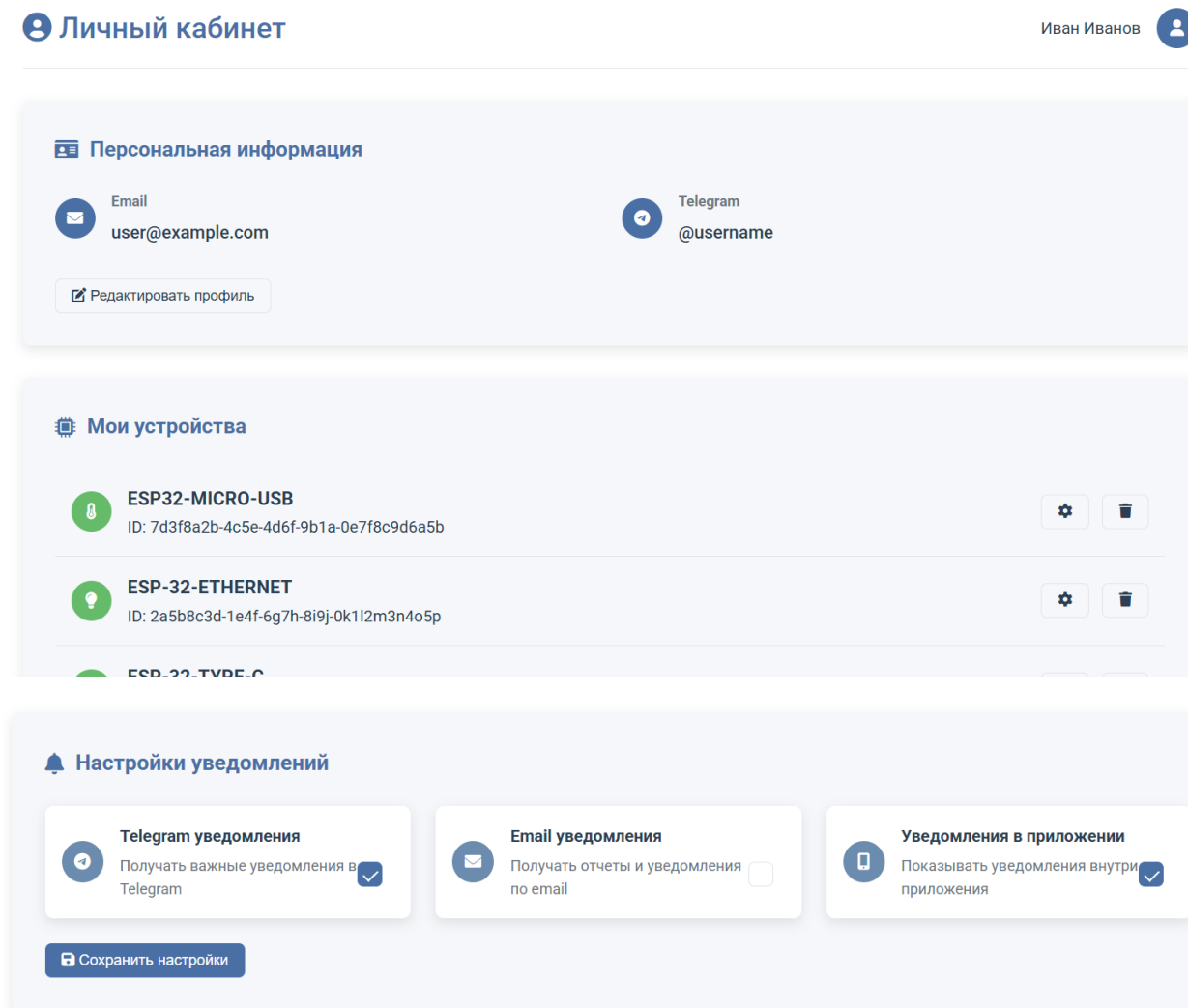


Рисунок 29 – Страница личного кабинета пользователя

Страница личного кабинета пользователя содержит информацию о пользователе, а также его устройствах, так же имеется настройка того, где пользователь хочет получать уведомления.

Страница с главным функционал представляет собой 7 настраиваемых карточек, отвечающих за визуализацию данных каждого конкретного датчика. Интерфейс реализует принцип централизованного доступа к учетной информации и параметрам системы, обеспечивая персонализированное взаимодействие с платформой.

Содержимое главной страницы системы представлено на рисунке 30.

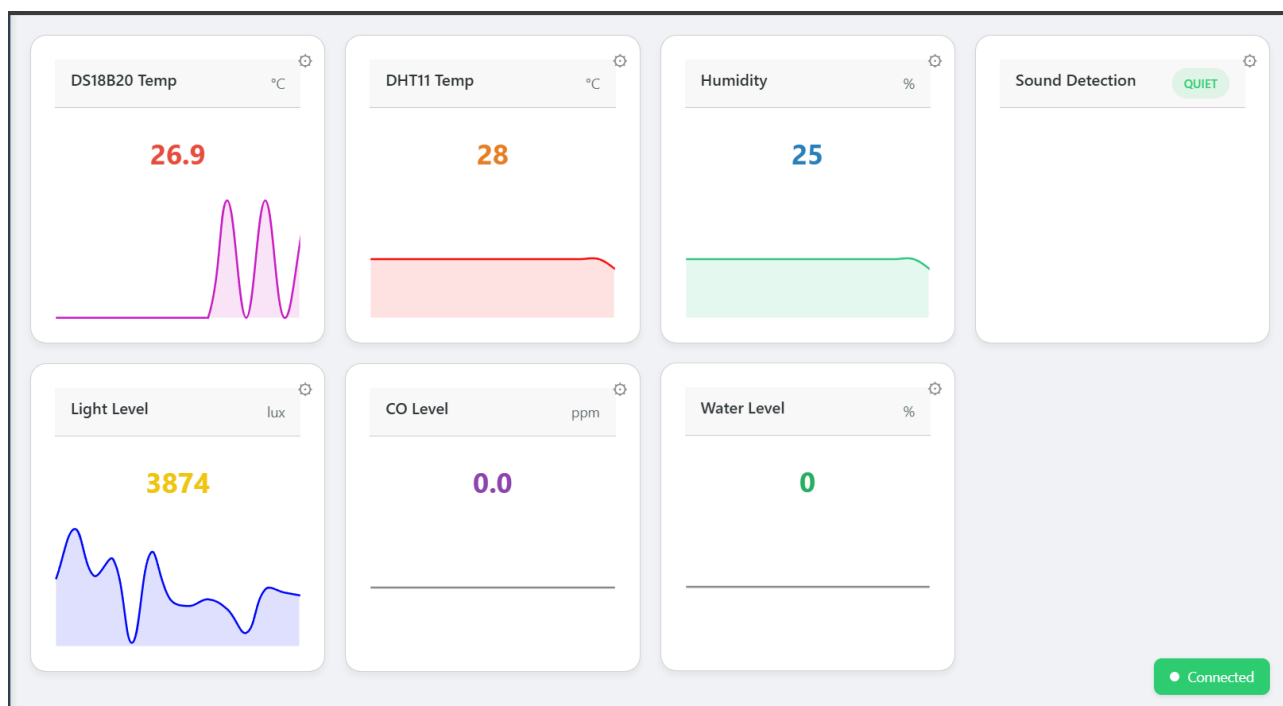


Рисунок 30 – Главная страница системы

На данной страницы выводятся объекты, связанные с показаниями датчиков, установленных в системе умного дома.

При клике на кнопку шестеренки появляется небольшое окно позволяющее настроить предпочтения по отображению: цвет и тип графика.

Окно статистики показаний датчиков представлено на рисунке 31.

Sensor Data Dashboard

Sensor IDs (comma-separated UUIDs)
e.g. 123e4567-e89b-12d3-a456-426614174000

Device IDs (comma-separated UUIDs)

DS18B20 Min Temp	DS18B20 Max Temp	DHT11 Min Temp
DHT11 Max Temp	Min Humidity %	Max Humidity %
Sound Detected Any	Min Light Level	Max Light Level
Min CO PPM	Max CO PPM	Min Water Level
Max Water Level	Measured At Start	
	ДД.ММ.ГГГГ --:--	
Measured At End		
ДД.ММ.ГГГГ --:--		

Рисунок 31 – Страница статистики

На странице размещены поля, вводя данные, в которые пользователь сам выбирает для себя способ фильтрации данных.

В процессе эксплуатации системы пользователь может сгенерировать себе отчет о показаниях используя гибкий фильтр, схожий с фильтром, применяемым на странице получения статистики. Пример сгенерированного отчета представлен на рисунке 32.

Smart House - Report									
Licensed report: Date of generation: 05.06.2025									
AVERAGE VALUES Temperature: 24,1°C Humidity: 31,2% CO: 0,0 ppm									
SENSOR DATA									
Date	Time	Temp.1	Temp.2	Humid.	Sound	Light	CO	Water	Device.
27.05.25	21:44	24,6	27	26	Yes	3639	0,0	0	a1b2c3d4
27.05.25	22:07	24,6	26	27	Yes	3988	0,0	0	a1b2c3d4
27.05.25	23:03	24,6	27	26	Yes	3952	0,0	0	a1b2c3d4
27.05.25	23:09	24,6	27	26	Yes	4001	0,0	0	a1b2c3d4
27.05.25	20:09	24,6	27	26	Yes	3997	0,0	0	a1b2c3d4
27.05.25	23:13	24,6	27	26	Yes	3951	0,0	0	a1b2c3d4
27.05.25	23:16	24,6	27	26	Yes	3946	0,0	0	a1b2c3d4

Рисунок 32 – Пример оформления PDF-отчета

Основные элементы отчета:

- 1 – дата генерации отчета;
- 2 – блок средних показателей основных датчиков;
- 3 – таблица данных, с записями, отфильтрованными по предпочтениям пользователя.

Дизайн документа создавался, следуя двум основным критериям: функциональности и современности. Для этого была выбрана библиотека

Apache PDFbox, позволяющая генерировать подобные отчеты быстро, задавать нужный дизайн и не перегружать пользователя информацией.

Система также предусматривает альтернативу веб интерфейсу, представленную в виде Telegram бота. Это решение добавляет вариативности и удобства пользователю при работе, а также позволяет взаимодействовать с приложением используя мобильный телефон. Интерфейс Telegram бота представлен на рисунке 33.

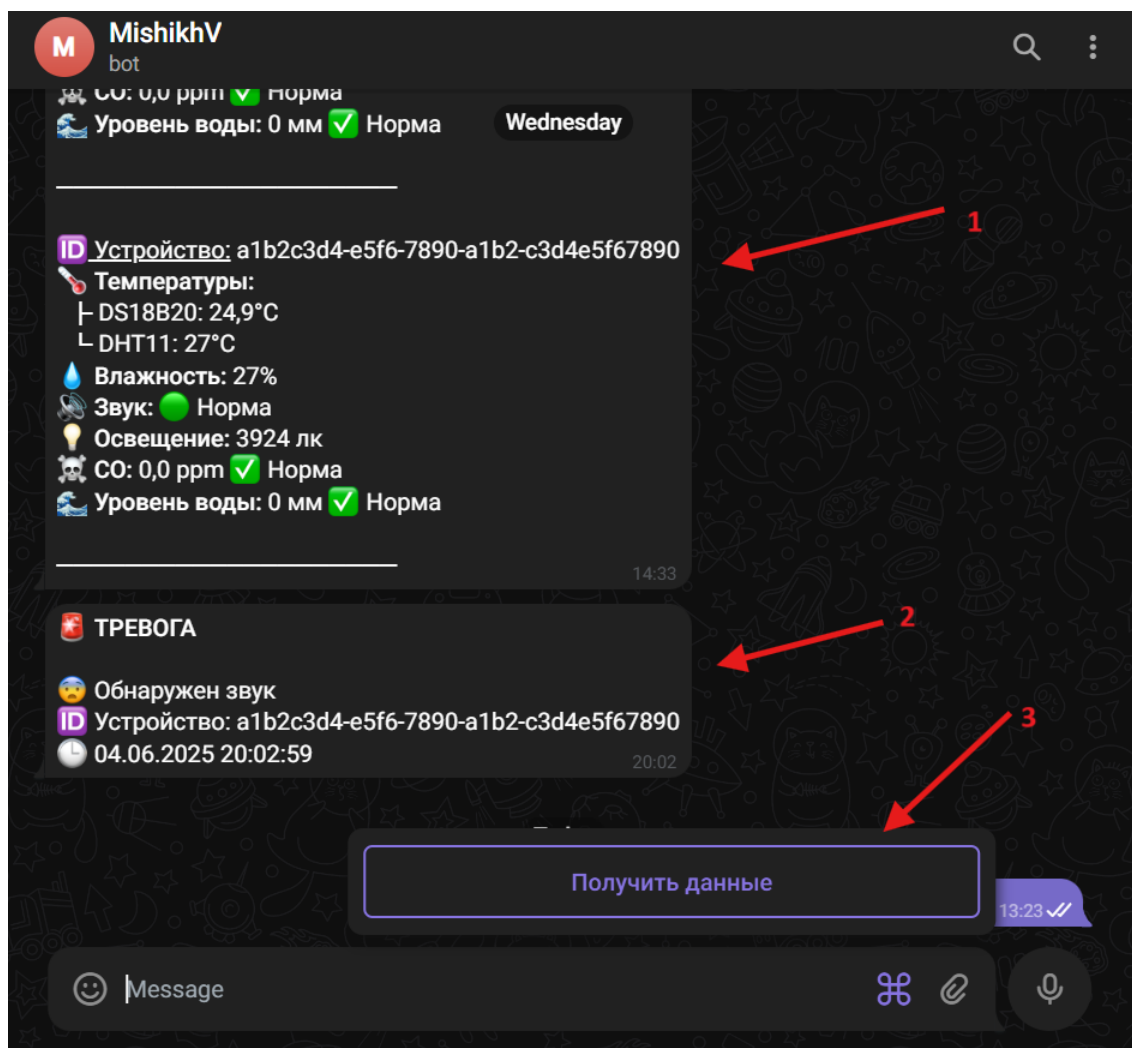


Рисунок 33 – Интерфейс Telegram бота

Основные элементы интерфейса Telegram бота:

- 1 – сообщение о последних показаний системы;
- 2 – уведомление о тревожных показаниях;
- 3 – кнопка получения последних показаний системы.

Telegram бот реализован через использование официального API Telegram, а также интегрируемую с фреймворком Spring boot библиотеку Telegram starter.

Разработка графического интерфейса производилась с использованием каскадной таблицы стилей CSS3 и гипертекстовой разметки HTML5. HTML5 служит для формирования структуры веб-страниц. CSS3, в свою очередь, применяется для настройки дизайна, включая параметры шрифтов, цветовую гамму и многое другое.

Графическое представление аналитических данных достигнуто с помощью библиотеки chart.js. Эта библиотека предназначена для построения разнообразных типов диаграмм, что позволяет наглядно демонстрировать различные аналитические показатели. Благодаря ее гибкости и адаптации под разные устройства, она эффективно интегрируется в общий интерфейс, облегчая представление и анализ комплексных данных [13].

3.5 Реализация аппаратной части приложения

Аппаратная часть приложения была разработана на микроконтроллере ESP-32 с встроенным модулем программирования и питания, с разъемом type-c. Программный код, разворачиваемый на данном микроконтроллере написан на языке программирования C++ в среде Arduino IDE.

Помимо микроконтроллера, была использована беспаячная макетная плата, нужная для крепления устройств, провода типа папа-папа, а также резисторы сопротивлением 1 кОм. В основе системы лежат 6 датчиков:

- 1 – цифровой датчик температуры ds18b20;
- 2 – цифровой датчик температуры dht11;
- 3 – цифровой датчик влажности dht11;
- 4 – аналоговый фоторезистор;
- 5 – аналоговый датчик уровня воды;
- 6 – цифровой датчик звука;
- 7 – цифровой датчик угарного газа.

Для грамотной разработки аппаратной части системы была создана электротехническая схема, представленная на рисунке 34.

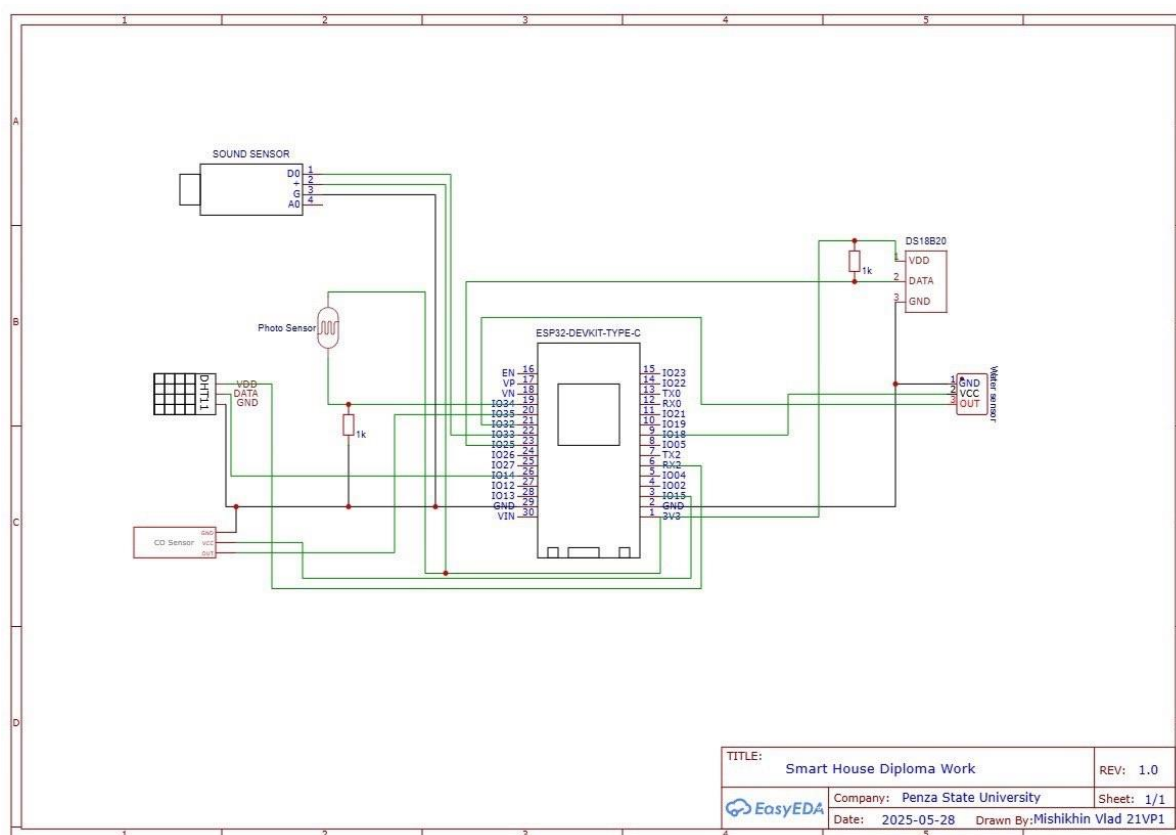


Рисунок 34 – Электротехническая схема системы умного дома

Помимо микроконтроллера, была использована беспаячная макетная плата, нужная для крепления устройств, провода типа «папа – папа», а также резисторы сопротивлением 1 кОм.

В микроконтроллерных системах считывание сигналов с датчиков осуществляется с помощью двух принципиально разных подходов: цифрового (`digitalRead`) и аналогового (`analogRead`) чтения данных. Функция `digitalRead` предназначена для работы с дискретными сигналами, которые могут принимать только два значения – высокий или низкий уровень напряжения, что соответствует логическим состояниям 1 и 0. Этот метод используется для детектирования простых событий, таких как срабатывание кнопки или обнаружение звукового импульса, где важна лишь факт наличия или отсутствия сигнала. В отличие от этого, функция `analogRead` позволяет измерять непрерывно изменяющееся напряжение в аналоговом виде с определённой

разрешающей способностью (обычно 10 – 12 бит), что необходимо для работы с датчиками, выдающими пропорциональные сигналы, такими как фоторезисторы, потенциометры или датчики уровня жидкости.

В представленном проекте оба метода чтения данных находят своё применение в зависимости от типа подключаемых сенсоров. Цифровое чтение (`digitalRead`) используется для обработки сигналов от датчика звука, где важно зафиксировать момент возникновения звукового события, что реализовано через прерывание для мгновенного реагирования. Аналоговое чтение (`analogRead`) применяется для получения количественных показаний с датчиков освещённости и уровня воды, которые изменяют своё сопротивление в зависимости от внешних условий, формируя непрерывный сигнал. Такое разделение подходов позволяет оптимально использовать ресурсы микроконтроллера – быстро реагировать на критические события через цифровые входы и одновременно точно измерять параметры окружающей среды через аналоговые каналы.

3.6 Реализация взаимодействия частей приложения

Для взаимодействия написанного для аппаратной части программного модуля и серверной части приложения используется внешний брокер сообщений, использующий протокол MQTT. MQTT – это сетевой протокол, представляющий собой надстройку над привычной сетевой моделью TCP/IP, ориентирован на работу с телеметрией устройств, и реализует обмен сообщениями по принципу «издатель – подписчик».

Реализация подключения и работы с брокером в программном обеспечении, развернутом на микроконтроллере ESP32, происходит за счет использования библиотеки `PubSubClient`. Код работы с данной библиотеки приведен в листинге 2.

Листинг 2 – Код работы взаимодействия с библиотекой `PubSubClient`

```
#include <PubSubClient.h>
const char* mqttServer = "mqtt.eclipseprojects.io";
```

```

const int mqttPort = 1883;
const char* mqttTopic = "sensors/room1";
PubSubClient mqttClient(espClient);
mqttClient.setServer(mqttServer, mqttPort);
mqttClient.setKeepAlive(90);
bool connectToMQTT() {
    if (mqttClient.connected()) return true;

    if (millis() - lastMqttAttempt < MQTT_RETRY_INTERVAL) return
false;

    lastMqttAttempt = millis();
    Serial.print("Connecting to MQTT...");

    if (mqttClient.connect("ESP32Client")) {
        Serial.println("connected!");
        return true;
    } else {
        Serial.println("failed!");
        return false;
    }
}
mqttClient.publish(mqttTopic, json.c_str());

```

Со стороны сервера подключение к брокеру осуществляется благодаря дополнительной зависимости в модуле фреймворка Spring boot integration, включающий в себя поддержку MQTT. Код настройки конфигурации протокола для серверной части представлен в листинге 3.

Листинг 3 – Код настройки конфигурации протокола

```

@Configuration
public class MqttConfig {
    @Bean
    public MqttPahoClientFactory mqttClientFactory() {
        DefaultMqttPahoClientFactory factory = new
DefaultMqttPahoClientFactory();
        MqttConnectOptions mqttConnectOptions = new
MqttConnectOptions();
        mqttConnectOptions.setServerURIs(new
String[]{"tcp://mqtt.eclipseprojects.io:1883"});
        mqttConnectOptions.setAutomaticReconnect(true);
        factory.setConnectionOptions(mqttConnectOptions);
        return factory;
    }
    @Bean
    public MqttMessageConverter mqttMessageConverter() {
        return new DefaultPahoMessageConverter();
    }
    @Bean
    public MqttPahoMessageDrivenChannelAdapter inboundAdapter(

```

```

        MqttPahoClientFactory factory) {
    MqttPahoMessageDrivenChannelAdapter adapter =
        new MqttPahoMessageDrivenChannelAdapter(
            "spring-server",
            factory,
            "sensors/room1");
    adapter.setCompletionTimeout(5000);
    adapter.setQos(1);
    adapter.setOutputChannelName("mqttInputChannel");
    adapter.setConverter(mqttMessageConverter());
    return adapter;
}
}

```

Данные технологии позволяют установить устойчивое соединение для быстрого и легковесного общения компонентов системы.

Для получения доступа микроконтроллера к локальной сети, используется стандарт передачи цифровых потоков данных по радиоканалам Wi-Fi, использующий встроенный в плату модуль.

Двустороннее общение сервера и клиента выполнено благодаря протоколу WebSocket, представляющий собой технологию связи, работающую поверх TCP-соединения и использующую постоянное соединение. В процессе разработки был выбран формат данных – JSON. Коммуникационная инфраструктура системы реализована через многоуровневую архитектуру взаимодействия компонентов, где каждый технологический выбор обусловлен требованиями эффективности и масштабируемости. На аппаратном уровне интеграция в сетевую среду обеспечивается стандартизированными беспроводными интерфейсами, позволяющими микроконтроллерным устройствам участвовать в обмене данными без необходимости сложной физической коммутации. Поверх транспортного уровня организован механизм двунаправленного обмена сообщениями, основанный на современных веб-технологиях, обеспечивающих минимальную задержку передачи и постоянную сессию взаимодействия.

Взаимосвязь между микросервисами серверной части выстроена за счет технологии gRPC, являющейся системой удалённого вызова процедур с

открытым исходным кодом, разработанной компанией Google. Данная система использует транспортный протокол HTTP/2 и интерфейс Protocol Buffer.

Последовательность выполнения считывания данных с датчиков на примере цифрового датчика DS18B20 и дальнейшая передача их клиенту, с использованием разных технологий и библиотек представлена на рисунке 35.

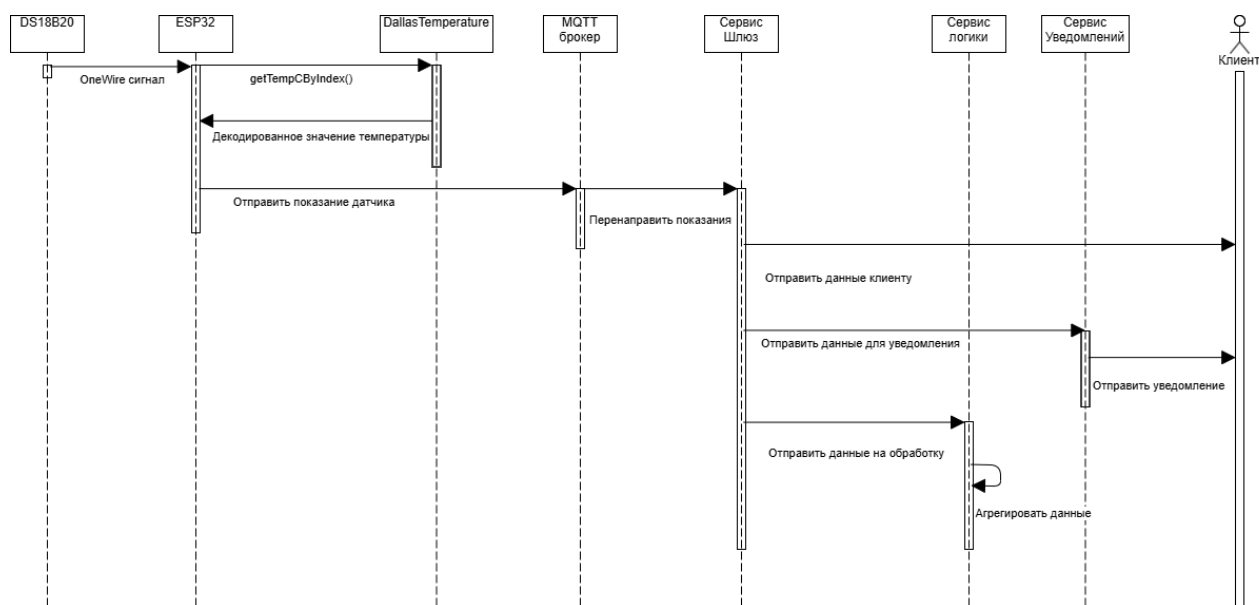


Рисунок 35 – Диаграмма последовательности для системы контроля показаний датчиков умного дома

Данная диаграмма предоставляет схематическое пояснение работы части функционала системы на конкретном примере. В начале происходит считывание температуры с цифрового датчика DS18B20, который отсылает данные в виде OneWire сигнала, в программе микроконтроллера происходит декодирование этой информации и приведение ее в стандартную систему измерения градусов Цельсия. Декодированием занимается библиотека DallasTemperature. Затем данные формируют JSON строку и передаются в брокер MQTT. Сервис шлюза считывает из брокера данное сообщение и передает его на клиент посредством WebSocket, а так же отправляет эти данные в сервис логики, для последующей агрегации и сервис уведомлений, для отправки оповещений, используя протокол gRPC.

3.7 Реализация подготовки отчетов в приложении

В системе реализован комплексный подход к обработке данных с датчиков, включающий как формирование наглядных отчетов, так и интеллектуальный анализ показателей. Процесс начинается с автоматической генерации PDF-отчетов с использованием библиотеки Apache PDFBox, где данные структурируются в удобном для восприятия виде. Каждый отчет содержит не только таблицу с подробными показаниями всех датчиков за выбранный период, но и визуальные элементы, такие как логотип системы и цветовые акценты для выделения критических значений. Особое внимание уделяется оформлению – отчет имеет профессиональный дизайн с четкой иерархией информации, что позволяет пользователю быстро оценить состояние всех параметров в помещении.

На основе собранных данных система проводит глубокий анализ, используя статистические методы для выявления аномалий и потенциально опасных ситуаций. Алгоритмы анализируют последние 100 показаний, вычисляя средние значения и стандартные отклонения по каждому параметру. Также происходит определение индекса теплового комфорта. Это позволяет системе обнаруживать резкие отклонения от нормы, такие как скачки температуры или повышенный уровень CO, и оперативно предупреждать пользователя. При этом учитывается не только абсолютное значение показателей, но и их взаимосвязь – например, система может определить, что повышенный уровень шума в ночное время при низкой освещенности требует особого внимания. Для дополнительной аналитической точности используется выявление аномалий методом Z-критериев.

Важной особенностью системы является ее практическая ориентированность – она не просто фиксирует данные, но и предлагает конкретные рекомендации по улучшению. Например, при обнаружении повышенной влажности система посоветует проветрить помещение или включить осушитель, а при резком росте уровня CO – немедленно принять

меры. Такой подход превращает систему из простого регистратора показаний в интеллектуального помощника, который помогает поддерживать комфортные и безопасные условия в доме, даже если пользователь не обладает специальными техническими знаниями. Схема алгоритма аналитики представлена на рисунке 36.

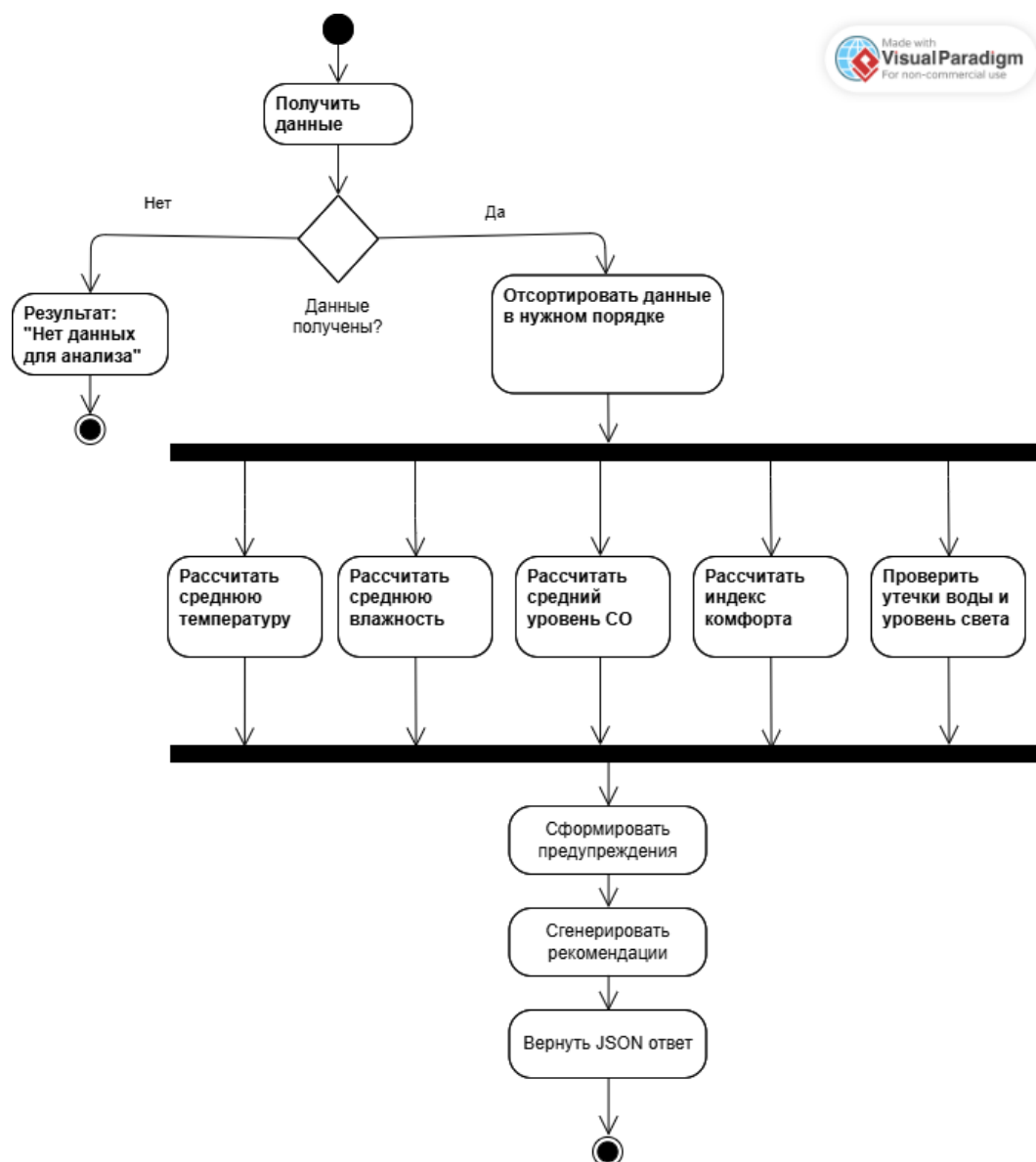


Рисунок 36 – Диаграмма активностей для алгоритмов аналитики

Алгоритм работы аналитического модуля демонстрирует продуманную логику обработки данных и формирования управляющих решений. Практическая ценность решения заключается в автоматизированных рекомендациях, помогающих пользователю оперативно реагировать на изменения микроклимата. Таким образом, система выступает не просто как

сборщик данных, а как полноценный интеллектуальный помощник, обеспечивающий безопасность и комфорт в доме без необходимости глубоких технических знаний от пользователя.

4 Контроль качества программного обеспечения

4.1 Обоснование выбора режима тестирования и метрик кода

Для создания надежного и удобного программного продукта необходимо проводить комплексное тестирование. Эта процедура позволяет выявить потенциальные уязвимости, улучшить стабильность работы системы и повысить уровень удовлетворенности конечных пользователей [14].

Наиболее популярны следующие виды тестирования:

- модульное тестирование;
- тестирование производительности;
- функциональное тестирование;
- интеграционное тестирование.

Модульное тестирование направлено на анализ работы минимальных логических единиц кода в изоляции от других частей системы. Такой подход обеспечивает быструю верификацию базовых функций.

Тестирование производительности представляет собой измерение скорости отклика системы при различных нагрузках, что особенно важно для ресурсоемких приложений.

Функциональное тестирование – это методика тестирования программного обеспечения, направленная на проверку соответствия поведения разработанного продукта заданным функциональным требованиям [15].

Интеграционное тестирование проверяет корректность взаимодействия модулей и сервисов программного обеспечения.

При проведении тестирования программного обеспечения был использован комплексный подход, сочетающий ручные и автоматизированные методы. Основным методом ручного тестирования стало функциональное тестирование, которое было выбрано благодаря его способности комплексно оценивать соответствие системы установленным требованиям. Этот метод позволяет проверить работоспособность приложения в условиях, максимально

приближенных к реальным сценариям использования, что обеспечивает достоверность результатов проверки.

Для автоматизированного тестирования был применен метод модульного тестирования, доказавший свою эффективность при проверке отдельных компонентов системы. Такой подход значительно сокращает временные затраты на разработку и отладку, позволяя быстро выявлять и устранять потенциальные проблемы на уровне отдельных сервисов и модулей приложения. Использование данного подхода позволило оптимизировать процесс разработки за счет сокращения временных затрат на идентификацию и локализацию дефектов. Каждый тестируемый модуль проверяется независимо от других компонентов системы, что значительно упрощает процесс отладки и минимизирует вероятность возникновения каскадных ошибок. Особое внимание уделено тестированию критически важных сервисов, где каждый тестовый сценарий воспроизводит как штатные, так и пограничные условия работы функциональных блоков.

Особое внимание было уделено анализу качества программного кода с использованием специализированных метрик. Эти инструменты предоставляют разработчикам ценные данные для оценки технических характеристик продукта, прогнозирования трудозатрат и анализа сложности реализованных решений. Применение метрик позволяет объективно оценивать надежность и производительность системы, что в конечном итоге способствует созданию более стабильного и качественного программного продукта.

В рамках проекта для оценки кода были выбраны две ключевые метрики. Метрика Холстеда, основанная на анализе операторов и операндов, дает возможность оценить сложность программы, определить объем выполненных работ и потенциал для дальнейшего расширения функционала. Метрика Чепина сосредоточена на анализе внутренней логики модулей, оценивая используемые переменные и их типы, что позволяет повысить качество и читаемость кода. Совместное применение этих метрик обеспечивает всесторонний анализ качества программного обеспечения [17].

4.2 Тестирование программных средств для системы контроля за показаниями датчиков умного дома

4.2.1 Функциональное тестирование черным ящиком

По завершению создания аппаратной и программной части было проведено функциональное тестирование системы. Результаты тестирования приведены в приложении Г.

Все функциональные тесты были завершены положительно. Ради обеспечения более тщательного контроля системы, был выбран тест кейс основного функционала приложения – получение данных, для которого отдельно была создана таблица причин и следствий, диаграмма функциональная диаграмма и таблица решений. Причинно-следственные связи представлены в таблице 9.

Таблица 9 – Таблица причин и следствий

Причины	
П1	Пользователь авторизован в системе
П2	MQTT брокер подключен
П3	Домашняя сеть Wi-Fi работает корректно
П4	Система подключена к питанию
П5	WebSocket соединение с клиентом установлено
Следствия	
С1	Сообщение: «Вы не авторизованы в системе. Войдите или зарегистрируйтесь»
С2	Сообщение: «MQTT брокер не подключен»
С3	Сообщение: «Домашняя сеть Wi-Fi работает не корректно»
С4	Сообщение: «Система не подключена к питанию»
С5	Сообщение: «WebSocket соединение не установлено»
С6	Получение данных датчиков пользователем

Используя данную таблицу, была составлена функциональная диаграмма, представленная на рисунке 37.

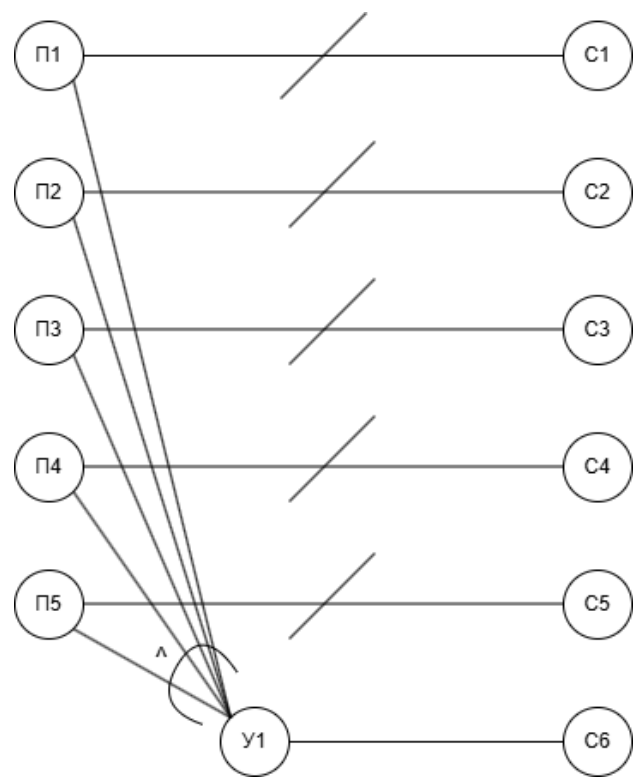


Рисунок 37 – Функциональная диаграмма

На основе диаграммы составляем схему решений. В верхней части – комбинация причин, в нижней – соответствующее следствие на каждую комбинацию причин. Схема решений представлена в таблице 10.

Таблица 10 – Таблица решений

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

Продолжение таблицы 10

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
C4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
C5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
C6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Из таблицы решений удалены несуществующую комбинации причин, затем были построены наборы тестов, расположенные в таблице 11.

Таблица 11 – Таблица тест-кейсов

№	Входные данные	Ожидаемый результат
1	Email: Пароль: Питание: 0V MQTT topic: Wi-Fi пинг: 300ms WebSocket:	Вы не авторизованы в системе, Войдите или зарегистрируйтесь, MQTT брокер не подключен, Домашняя сеть Wi-Fi работает не корректно, Система не подключена к питанию, WebSocket соединение с клиентом не установлено
2	Email: user@mail.ru Пароль: 1234 Питание: 0V MQTT topic: Wi-Fi пинг: 300ms WebSocket:	MQTT брокер не подключен, Домашняя сеть Wi-Fi работает не корректно, Система не подключена к питанию, WebSocket соединение с клиентом не установлено
3	Email: Пароль: Питание: 0V MQTT topic: sensors/room Wi-Fi пинг: 300ms WebSocket:	Вы не авторизованы в системе, Войдите или зарегистрируйтесь, Домашняя сеть Wi-Fi работает не корректно, Система не подключена к питанию, WebSocket соединение с клиентом не установлено
4	Email: user@mail.ru Пароль: 1234 Питание: 0V MQTT topic: sensors/room Wi-Fi пинг: 300ms WebSocket:	Домашняя сеть Wi-Fi работает не корректно, Система не подключена к питанию, WebSocket соединение с клиентом не установлено
5	Email: Пароль: Питание: 0V MQTT topic: Wi-Fi пинг: 20ms WebSocket:	Вы не авторизованы в системе, Войдите или зарегистрируйтесь, MQTT брокер не подключен, Домашняя сеть Wi-Fi работает не корректно, Система не подключена к питанию, WebSocket соединение с клиентом не установлено

Продолжение таблицы 11

№	Входные данные	Ожидаемый результат
6	Email: user@mail.ru Пароль: 1234 Питание: 0V MQTT topic: Wi-Fi пинг: 20ms WebSocket:	MQTT брокер не подключен, Система не подключена к питанию, WebSocket соединение с клиентом не установлено
7	Email: Пароль: Питание: 0V MQTT topic: sensors/room Wi-Fi пинг: 20ms WebSocket:	Вы не авторизованы в системе, Войдите или зарегистрируйтесь, Система не подключена к питанию, WebSocket соединение с клиентом не установлено
8	Email: user@mail.ru Пароль: 1234 Питание: 0V MQTT topic: sensors/room Wi-Fi пинг: 20ms WebSocket:	Система не подключена к питанию, WebSocket соединение с клиентом не установлено
9	Email: Пароль: Питание: 5V MQTT topic: Wi-Fi пинг: 300ms WebSocket:	Вы не авторизованы в системе, Войдите или зарегистрируйтесь, MQTT брокер не подключен, Домашняя сеть Wi-Fi работает не корректно, WebSocket соединение с клиентом не установлено
10	Email: user@mail.ru Пароль: 1234 Питание: 5V MQTT topic: Wi-Fi пинг: 300ms WebSocket:	MQTT брокер не подключен, Домашняя сеть Wi-Fi работает не корректно, WebSocket соединение с клиентом не установлено
11	Email: Пароль: Питание: 5V MQTT topic: sensors/room Wi-Fi пинг: 300ms WebSocket:	Вы не авторизованы в системе, Войдите или зарегистрируйтесь, Домашняя сеть Wi-Fi работает не корректно, WebSocket соединение с клиентом не установлено
12	Email: user@mail.ru Пароль: 1234 Питание: 5V MQTT topic: sensors/room Wi-Fi пинг: 300ms WebSocket:	Домашняя сеть Wi-Fi работает не корректно, WebSocket соединение с клиентом не установлено
13	Email: Пароль: Питание: 5V MQTT topic: Wi-Fi пинг: 20ms WebSocket:	Вы не авторизованы в системе, Войдите или зарегистрируйтесь, MQTT брокер не подключен, WebSocket соединение с клиентом не установлено

Завершение таблицы 11

№	Входные данные	Ожидаемый результат
14	Email: user@mail.ru Пароль: 1234 Питание: 5V MQTT topic: Wi-Fi пинг: 20ms WebSocket:	MQTT брокер не подключен, WebSocket соединение с клиентом не установлено
15	Email: Пароль: Питание: 5V MQTT topic: sensors/room Wi-Fi пинг: 20ms WebSocket:	Вы не авторизованы в системе, WebSocket соединение с клиентом не установлено
16	Email: user@mail.ru Пароль: 1234 Питание: 5V MQTT topic: sensors/room Wi-Fi пинг: 20ms WebSocket:	WebSocket соединение с клиентом не установлено
17	Email: user@mail.ru Пароль: 1234 Питание: 5V MQTT topic: sensors/room Wi-Fi пинг: 20ms WebSocket: app/topic/sensor-data	Данные успешно получены

Проведенное функциональное тестирование подтвердило полную работоспособность системы в соответствии с заявленными требованиями. Успешное выполнение всех тестовых сценариев, включая критически важный кейс получения данных, демонстрирует стабильность работы как аппаратной, так и программной составляющих. Детальный анализ причинно-следственных связей позволил дополнительно укрепить надежность обработки данных и реакцию системы на различные события.

4.2.2 Модульное тестирование

После завершения разработки системы было проведено модульное тестирование с использованием техники мок-объектов (mock testing). Данный подход позволил проверить работу отдельных компонентов системы в изоляции от внешних зависимостей. В ходе тестирования были созданы заглушки для датчиков, MQTT-брокера и WebSocket-соединения, что дало

возможность имитировать различные состояния системы и проверить реакцию программного обеспечения на разные сценарии без необходимости использования реального оборудования.

Мок-тестирование особенно полезно на ранних этапах разработки, когда некоторые компоненты системы еще не готовы. Оно позволяет выявить логические ошибки в обработке данных и проверить корректность взаимодействия между модулями. В нашем случае с помощью мок-объектов удалось проверить все возможные состояния системы – от нормальной работы до различных аварийных ситуаций, таких как потеря соединения или некорректные данные от датчиков. Результаты такого тестирования помогли улучшить обработку ошибок и сделать систему более устойчивой. Данный подход особенно ценен в условиях частично готовой инфраструктуры, когда отсутствие некоторых компонентов не должно препятствовать верификации уже реализованной функциональности.

Код мок тестов приведен в приложении Ж. Результаты модульного тестирования приведены в приложении И. План тестирования представлен в таблице 12.

Таблица 12 – План модульного тестирования

Тестируемый компонент	Тест	Ожидаемый результат	Актуальный результат
SensorData	Должны придти данные с брокера	Пройден	Пройден (рисунок И.1)
	Должны корректно сериализоваться данные	Пройден	Пройден (рисунок И.1)
	Должны корректно передаться данные на клиент	Пройден	Пройден (рисунок И.1)
	Должны отобразиться карточки датчиков	Пройден	Пройден (рисунок И.1)
	Должны накапливаться и усредняться не экстренные данные	Пройден	Пройден (рисунок И.1)

Продолжение таблицы 12

Тестируемый компонент	Тест	Ожидаемый результат	Актуальный результат
NotificationComponent	Должен создаваться	Пройден	Пройден (рисунок И.2)
	Должно отправиться уведомление в телеграм	Пройден	Пройден (рисунок И.2)
	Должно отправиться уведомление по почте	Пройден	Пройден (рисунок И.2)
	Должно отправиться уведомление внутри приложения	Пройден	Пройден (рисунок И.2)
	Должно сохраниться уведомление	Пройден	Пройден (рисунок И.2)
	Должно быть найдено уведомление по идентификатору	Пройден	Пройден (рисунок И.2)
LogicComponent	Должен создаваться	Пройден	Пройден (рисунок И.3)
	Должен агрегировать данные	Пройден	Пройден (рисунок И.3)
	Должен находить данные по фильтру	Пройден	Пройден (рисунок И.3)
	Должен создавать отчет	Пройден	Пройден (рисунок И.3)
	Должен анализировать данные	Пройден	Пройден (рисунок И.3)
	Должен сохранять данные	Пройден	Пройден (рисунок И.3)

Проведенное модульное тестирование с применением мок-объектов подтвердило корректность работы ключевых компонентов системы в изолированных условиях. Успешное прохождение всех тестовых случаев, описанных в таблице 12, свидетельствует о готовности системы к дальнейшему развертыванию.

4.3 Оценка качества программного кода

4.3.1 Метрика Холстеда

Для определения метрики Холстеда за основу была взята функция спецификации `findByFilter`, компонента `LogicComponent`, необходимая для поиска показаний датчиков по определенным фильтрам. Код данной функции представлен в листинге А.50 приложения А.

Для вычисления данной метрики требуется посчитать число вхождения уникальных операторов (N1) и операндов (N2), а также общее количество операторов (n1) и операндов (n2).

Операторы функции `findByFilter` и число их вхождений приведены в таблице 13.

Таблица 13 – Операторы функции `findByFilter`

№	Операторы	Число вхождений
1	->	1
2	!	19
3	.	114
4	=	18
5	&&	2
6	if	20
7	return	2
8	null	17
9	public	2
10	static	1
11	class	1
12	add	5
13	new	2
14	n1 = 13	N1 = 204

Операнды функции `findByFilter` и число их вхождений приведены в таблице 14.

Таблица 14 – Операнды функции `findByFilter`

№	Операнды	Число вхождений
1	id	3
2	deviceId	3
3	temperatureDs18b20	3
4	temperatureDht11	3

Продолжение таблицы 14

№	Операнды	Число вхождений
5	humidity	4
6	soundDetected	2
7	lightLevel	3
8	coPpm	3
9	waterLevel	3
10	measuredAt	3
11	n2 = 10	N2 = 30

Данные, полученные в ходе обработки программного кода функции findByFilter компонента LogicComponent были использованы для расчета характеристики программы по формулам из метрики Холстеда [1]. Результаты вычислений представлены в таблице 15.

Таблица 15 – Результаты вычислений по метрике Холстеда

Значение	Результат
Длина реализации программы	234
Словарь программы	23
Длина программы	81.3
Объем программы	1057.68
Потенциальный объем программы	28.53
Теоретический уровень программы	0.027
Сложность программы	37.04
Работа программирования	28.56

Анализ метрик программного кода показал, что разработанная система полностью соответствует установленным требованиям качества. Полученные значения свидетельствуют об оптимальной структуре кода – длина реализации оказалась меньше длины программы, что указывает на отсутствие существенных недостатков в архитектуре решения. Низкий теоретический уровень программы, выявленный в ходе оценки, подтверждает эффективность использованных алгоритмов, поскольку фактический объем кода значительно

превышает минимально необходимый. При этом анализ демонстрирует грамотное применение возможностей языка программирования, что обеспечивает баланс между производительностью, читаемостью и функциональностью системы. Все ключевые показатели находятся в пределах нормативных значений, что гарантирует стабильную работу приложения и легкость его дальнейшей поддержки.

4.3.2 Метрика Чепина

Для оценки сложности модуля анализа данных сенсоров была применена метрика Чепина. Объектом исследования стал компонент, отвечающий за обработку показаний датчиков и формирование рекомендаций для системы умного дома, исходный код данного модуля представлен в листинге А.48 приложения А.

Для расчета метрики берется все множество переменных, составляющих список ввода-вывода, а затем разбивается на 4 функциональные группы:

- вводимые переменные для расчетов и для обеспечения вывода (Р);
- модифицируемые внутри программы переменные (М);
- переменные, участвующие в управлении работой программного модуля (С);
- не используемые в программе переменные (Т).

Необходимо учитывать переменную в каждой соответствующей функциональной группе:

$$Q = a_1 * P + a_2 * M + a_3 * C + a_4 * T,$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты.

Метрика Чепина использует систему весовых коэффициентов, которые позволяют оценить вклад каждой группы переменных в общую сложность программы. Наибольший вес (коэффициент 3) присваивается управляющим переменным (группа С), поскольку они непосредственно определяют логику выполнения программы и влияют на поток управления. Остальные группы

переменных имеют следующие коэффициенты влияния: входные данные оцениваются коэффициентом 1, модифицируемые переменные – коэффициентом 2, а неиспользуемые («паразитные») переменные, несмотря на их минимальное влияние на функциональность, получают коэффициент 0.5, так как их наличие может усложнять понимание кода.

При анализе модуля обработки сенсорных данных было выявлено 12 входных переменных, используемых для расчетов и формирования результатов, 8 внутренних переменных, значения которых изменяются в процессе работы алгоритма, и одна управляющая переменная, ответственная за логику выполнения программы. Неиспользуемые переменные в данном модуле отсутствуют. Исходя из этих данных и применяя указанные весовые коэффициенты, расчет метрики Чепина для этого модуля демонстрирует:

$$Q = 1 \times 15 + 2 \times 12 + 3 \times 3 + 0.5 \times 0 = 48.$$

Из результата можно сделать вывод, что вычисленная метрика Чепина сигнализирует то, что данный компонент системы имеет относительно небольшой размер и невысокую сложность исходного кода.

По итогу был получен вывод по тестированию: все ошибки были исправлены, что показало регрессионное тестирование.

Заключение

В рамках выполнения выпускной квалификационной работы была разработана комплексная система мониторинга показаний датчиков умного дома. На первом этапе проведено глубокое исследование предметной области, в ходе которого проанализированы современные решения для автоматизации жилых помещений, включая такие популярные системы как «Ajax starter kit», «Ростелеком Безопасность», «Xiaomi Smart Home Security kit» и «Sber умный дом». Сравнительный анализ этих решений позволил выявить их сильные и слабые стороны, что стало основой для формулирования функциональных и нефункциональных требований к разрабатываемой системе.

Архитектурное проектирование системы включало создание модульной структуры приложения с четким разделением компонентов. Были разработаны концептуальная и логическая модели базы данных, обеспечивающие эффективное хранение и обработку показаний датчиков. Особое внимание уделено проектированию пользовательского интерфейса, который должен предоставлять наглядную информацию о состоянии домашней среды, и классов приложения, отвечающих за обработку данных и формирование отчетов.

Реализация системы осуществлялась с использованием современных технологий и инструментов разработки. База данных была реализована на PostgreSQL, обеспечивая надежное хранение информации. Серверная часть системы разработана как набор микросервисов на Spring boot, что обеспечивает масштабируемость и гибкость решения. Клиентское приложение создано с использованием JavaScript, HTML и CSS, что позволило добиться высокой производительности и отзывчивости интерфейса. Отдельное внимание уделено реализации аппаратной части на базе микроконтроллера ESP32 и организации взаимодействия между всеми компонентами системы через REST API, gRPC, MQTT и WebSocket соединения.

Контроль качества разработанного решения включал комплексное тестирование функционала как методом черного ящика, так и модульное

тестирование отдельных компонентов. Для объективной оценки сложности кода применены метрики Холстеда и Чепина, показавшие оптимальное соотношение между функциональностью и сложностью реализации. Планирование работ и оценка бюджета проводились с использованием профессиональных инструментов управления проектами, что позволило точно оценить трудозатраты (360 часов) и общую стоимость разработки (86 000 рублей).

В результате проделанной работы создана современная система мониторинга умного дома, предоставляющая пользователям полный контроль за показаниями датчиков температуры, влажности, уровня СО, освещенности и других параметров домашней среды. Система отличается высокой надежностью, производительностью и соответствует всем современным требованиям к программному обеспечению подобного класса. Разработанное решение может быть успешно применено как в частных домах и квартирах, так и в коммерческой недвижимости для обеспечения комфорта и безопасности.

Список используемых источников

1. Официальный сайт языка программирования Java [Электронный ресурс]. URL: <https://www.java.com/ru/> (дата обращения 04.06.2024).
2. Аллен Дауни, Крис Мейфилд Think Java: How to Think Like a Computer Scientist / Аллен Дауни, Крис Мейфилд – University of Minnesota, 2004 – 320с.
3. Draw.io // Skillfactory Media. [Электронный ресурс]. URL: <https://blog.skillfactory.ru/draw-io/> (дата обращения: 7.06.2024).
4. Артем Демиденко ESP32 для начинающих: Умный дом своими руками // Артем Демиденко – Litres, 2025 – 258 с.
5. Андрей Дементьев «Умный» дом XXI века // Андрей Дементьев – Litres, 2018 – 142 с.
6. Роберт Мартин Чистый код: создание, анализ и рефакторинг / Роберт Мартин – Питер, 2022 – 389 с.
7. Майк МакГрат JavaScript для начинающих. 6-е издание// Майк МакГрат – Эксмо, 2024 – 370 с.
8. Фримен Эрик, Робсон Элизабет, Берт Бейтс, Кэти Сьерра Паттерны Проектирования // Фримен Эрик, Робсон Элизабет, Берт Бейтс, Кэти Сьерра – Питер, 2021 – 640 с.
9. Андрей Матюшин Программирование микроконтроллеров: стратегия и тактика// Андрей Матюшин – ДМК Пресс, 2022 – 200 с.
10. Юрий Магда Программирование и отладка C/C++ приложений для микроконтроллеров ARM // Юрий Магда – ДМК Пресс, 2014 – 270 с.
11. Программирование устройств на основе модуля ESP32 // Тематический блог «Хабр» [Электронный ресурс]. URL: <https://habr.com/ru/articles/522730/> (дата обращения 08.06.2024).
12. Программирование ESP32 с ESP-IDF // Тематический блог «Хабр» [Электронный ресурс]. URL: <https://habr.com/ru/articles/918434/> (дата обращения 09.06.2024)

13. Разработка электроники. О микроконтроллерах на пальцах// Тематический блог «Хабр» [Электронный ресурс]. URL: <https://habr.com/ru/articles/445936/> (дата обращения 10.06.2024).
14. Что такое gRPC? // Документация Yandex Cloud [Электронный ресурс]. URL: <https://yandex.cloud/ru/docs/glossary/grpc?> (дата обращения 11.06.2024).
15. MQTT – The Standard for IoT Messaging // Онлайн документация [Электронный ресурс]. URL: <https://mqtt.org/> (дата обращения 12.06.2024)
16. Как работает Docker: подробный гайд от техлида// Skillbox media [Электронный ресурс]. URL: <https://skillbox.ru/media/code/kak-rabotaet-docker-podrobnyy-gayd-ot-tekhlista/> (дата обращения 12.06.2024)
17. IntelliJ IDEA: что это за среда разработки// Skillfactory media [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/intellij-idea/> (дата обращения 09.06.2024)
18. REST API // Skillfactory Media. [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/rest-api/> (дата обращения: 12.06.2024).
19. Программный код и его метрики // Тематический блог «Хабр». [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 13.06.2024)
20. Ньюман, С. Строим микросервисы: проектирование масштабируемых систем / С. Ньюман. – М.: ДМК Пресс, 2016. – 384 с.

Приложение А
(обязательное)
Листинг

Листинг A.1 – Файл MqttConfig.java

```

@Configuration
public class MqttConfig {

    @Bean
    public MqttPahoClientFactory mqttClientFactory() {
        DefaultMqttPahoClientFactory factory = new
DefaultMqttPahoClientFactory();
        MqttConnectOptions mqttConnectOptions = new
MqttConnectOptions();
        mqttConnectOptions.setServerURIs(new
String[]{"tcp://mqtt.eclipseprojects.io:1883"});
        mqttConnectOptions.setAutomaticReconnect(true);
        factory.setConnectionOptions(mqttConnectOptions);
        return factory;
    }
    @Bean
    public MqttMessageConverter mqttMessageConverter() {
        return new DefaultPahoMessageConverter();
    }
    @Bean
    public MqttPahoMessageDrivenChannelAdapter inboundAdapter(
        MqttPahoClientFactory factory) {

        MqttPahoMessageDrivenChannelAdapter adapter =
            new MqttPahoMessageDrivenChannelAdapter(
                "spring-server",
                factory,
                "sensors/room1");

        adapter.setCompletionTimeout(5000);
        adapter.setQos(1);
        adapter.setOutputChannelName("mqttInputChannel");
        adapter.setConverter(mqttMessageConverter());
        return adapter;
    }
}

```

Листинг A.2 – Файл SwaggerConfig.java

```

@OpenAPIDefinition(
    info = @Info(
        title = "Diploma Project API",
        version = "1.0",
        description = "API для управления пользователями и
датчиками"
    )
)
public class SwaggerConfig {
}

```

Листинг А.3 – Файл WebSocketConfig.java

```

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements
WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry
config) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry
registry) {
        registry.addEndpoint("/ws")
            .setAllowedOriginPatterns("*")
            .withSockJS();
    }
}

```

Листинг А.4 – Файл MainController.java

```

@Controller
@CrossOrigin(origins = "*", maxAge = 3600)
public class MainController {
    private final SimpMessagingTemplate messagingTemplate;
    private final TestService testService;
    private final MyTelegramBot telegramBot;

    public MainController(SimpMessagingTemplate messagingTemplate,
TestService testService, MyTelegramBot telegramBot) {
        this.messagingTemplate = messagingTemplate;
        this.testService = testService;
        this.telegramBot = telegramBot;
    }

    @PostMapping("/api/sensor-data")
    public ResponseEntity<?> getData(@RequestBody SensorData
sensorData) {
        System.out.println(sensorData);
        messagingTemplate.convertAndSend("/topic/sensor-data",
sensorData);

        return ResponseEntity.ok().body(sensorData);
    }

    @GetMapping("/main")
    public String getMainPage() {
        return "main";
    }
}

```

```

    }
    @GetMapping("/statistic")
    public String getStatisticPage() {

        return "statistic";
    }

    @GetMapping("/notification-settings")
    public String getNotificationSettingsPage() {
        return "notification-settings";
    }

    @GetMapping("/reports")
    public String getReportsPage() {
        return "report";
    }

    @GetMapping("/analysis")
    public String getAnalysisPage() {
        return "analysis";
    }
}

```

Листинг А.5 – Файл MqttMessageListener.java

```

@MessageEndpoint
@RequiredArgsConstructor
public class MqttMessageListener {
    private final MyTelegramBot telegramBot;
    private final ObjectMapper objectMapper;
    private final SimpMessagingTemplate messagingTemplate;
    private final LogicService logicService;
    private final NotificationService notificationService;
    private final Map<UUID, List<SensorData>> minuteDataCache =
new ConcurrentHashMap<>();
    private final ScheduledExecutorService scheduler =
Executors.newScheduledThreadPool(1);
    private static final UUID USERID = UUID.fromString("a1b2c3d4-
e5f6-7890-a1b2-c3d4e5f67890");

    @PostConstruct
    public void init() {

        scheduler.scheduleAtFixedRate(this::sendAggregatedData, 1,
15, TimeUnit.SECONDS);
    }

    @PreDestroy
    public void cleanup() {
        scheduler.shutdown();
    }
}

```

```

    }
    @ServiceActivator(inputChannel = "mqttInputChannel")
    public void handleMessage(Message<?> message) throws
    JsonProcessingException {
        String payload = message.getPayload().toString();
        SensorData sensorData = objectMapper.readValue(payload,
        SensorData.class);

        messagingTemplate.convertAndSend("/topic/sensor-data",
        sensorData);
        System.out.println("Received MQTT message: " + payload);

        boolean isUrgentEvent = false;

        if (sensorData.isSoundDetected() ||
        sensorData.getWaterLevel() > 100) {
            StringBuilder alert = new StringBuilder();

            if (sensorData.isSoundDetected()) {
                notificationService.sendSoundAlert(USERID, sensorData.getDeviceId(),
                sensorData.isSoundDetected());
            }

            if (sensorData.getWaterLevel() > 100) {
                notificationService.sendWaterLevelAlert(USERID, sensorData.getDeviceId(),
                sensorData.getWaterLevel());
            }

            isUrgentEvent = true;
        }

        if (isUrgentEvent) {
            logicService.sendSensorData(sensorData);
        }

        minuteDataCache.computeIfAbsent(sensorData.getDeviceId(),
        k -> new ArrayList<>()).add(sensorData);
    } private void sendAggregatedData() {
        minuteDataCache.forEach((deviceId, dataList) -> {
            if (!dataList.isEmpty()) {
                AggregatedSensorData aggregatedData =
                aggregateData(deviceId, dataList);
                logicService.sendAggregatedData(aggregatedData);
                dataList.clear();
            }
        });
    }

    private AggregatedSensorData aggregateData(UUID deviceId,
    List<SensorData> dataList) {

```

```

double avgTempDs18b20 = dataList.stream()
    .mapToDouble(SensorData::getTemperatureDs18b20)
    .average()
    .orElse(0.0);

double avgTempDht11 = dataList.stream()
    .mapToInt(SensorData::getTemperatureDht11)
    .average()
    .orElse(0.0);

double avgHumidity = dataList.stream()
    .mapToInt(SensorData::getHumidity)
    .average()
    .orElse(0.0);

double avgLightLevel = dataList.stream()
    .mapToInt(SensorData::getLightLevel)
    .average()
    .orElse(0.0);

double avgCoPpm = dataList.stream()
    .mapToDouble(SensorData::getCoPpm)
    .average()
    .orElse(0.0);

double avgWaterLevel = dataList.stream()
    .mapToInt(SensorData::getWaterLevel)
    .average()
    .orElse(0.0);

long soundEvents = dataList.stream()
    .filter(SensorData::isSoundDetected)
    .count();

long waterAlerts = dataList.stream()
    .filter(data -> data.getWaterLevel() > 100)
    .count();

return AggregatedSensorData.builder()
    .deviceId(deviceId)
    .startTimestamp(Instant.now().minusSeconds(60))
    .endTimestamp(Instant.now())
    .avgTemperatureDs18b20(avgTempDs18b20)
    .avgTemperatureDht11(avgTempDht11)
    .avgHumidity(avgHumidity)
    .avgLightLevel(avgLightLevel)
    .avgCoPpm(avgCoPpm)
    .avgWaterLevel(avgWaterLevel)
    .soundEventsCount(soundEvents)
    .waterAlertsCount(waterAlerts)
    .build();
}

```

```
}
```

Листинг A.6 – Файл NotificationsController.java

```
@RestController
@RequestMapping("/api/notification-settings")
@RequiredArgsConstructor
public class NotificationsController {
    private final NotificationService notificationService;

    @GetMapping("/")
    public ResponseEntity<NotificationSettingsDto>
getSettingsForUser(@RequestParam UUID userId) {
        return
ResponseEntity.ok(notificationService.getSettingsForUser(userId));
    }

    @PostMapping("/saveOrUpdate")
    public ResponseEntity<?> saveOrUpdateNotificationSettings(
        @RequestBody NotificationSettingsDto
notificationSettingsDto
    ){

notificationService.saveOrUpdateNotificationSettings(notificationS
ettingsDto);
        return new ResponseEntity<>(HttpStatus.valueOf(200));
    }
}
```

Листинг A.6 – Файл SensorDataController.java

```
@RestController
@RequestMapping("/api/sensor-data")
@RequiredArgsConstructor
public class SensorDataController {

    private final LogicService logicService;
    private final SensorDataMapper filterMapper;
    private final PDFService pdfService;

    @Operation(summary = "Filter sensor data", description = "Get
filtered sensor data")
    @GetMapping("/filter")
    public List<SensorDataDto> filterSensorData(
        @Parameter(name = "ids", description = "List of sensor
IDs", in = ParameterIn.QUERY, schema = @Schema(type = "array",
format = "uuid"))
        @RequestParam(required = false) List<UUID> ids,
```

```

        @Parameter(name = "deviceIds", description = "List of
device IDs", in = ParameterIn.QUERY, schema = @Schema(type =
"array", format = "uuid"))
        @RequestParam(required = false) List<UUID> deviceIds,

        @Parameter(description = "Min temperature (DS18B20)")
        @RequestParam(required = false) Double
temperatureDs18b20Start,

        @Parameter(description = "Max temperature (DS18B20)")
        @RequestParam(required = false) Double
temperatureDs18b20End,

        @Parameter(description = "Min temperature (DHT11)")
        @RequestParam(required = false) Integer
temperatureDht11Start,

        @Parameter(description = "Max temperature (DHT11)")
        @RequestParam(required = false) Integer
temperatureDht11End,

        @Parameter(description = "Min humidity")
        @RequestParam(required = false) Integer humidityStart,

        @Parameter(description = "Max humidity")
        @RequestParam(required = false) Integer humidityEnd,

        @Parameter(description = "Sound detected flag")
        @RequestParam(required = false) Boolean soundDetected,

        @Parameter(description = "Min light level")
        @RequestParam(required = false) Integer
lightLevelStart,

        @Parameter(description = "Max light level")
        @RequestParam(required = false) Integer lightLevelEnd,

        @Parameter(description = "Min CO ppm")
        @RequestParam(required = false) Double coPpmStart,

        @Parameter(description = "Max CO ppm")
        @RequestParam(required = false) Double coPpmEnd,

        @Parameter(description = "Min water level")
        @RequestParam(required = false) Integer
waterLevelStart,

        @Parameter(description = "Max water level")
        @RequestParam(required = false) Integer waterLevelEnd,

        @Parameter(description = "Start time for filtering",
schema = @Schema(type = "string", format = "date-time"))

```

```

        @RequestParam(required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE_TIME) LocalDateTime measuredAtStart,

        @Parameter(description = "End time for filtering",
schema = @Schema(type = "string", format = "date-time"))
        @RequestParam(required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE_TIME) LocalDateTime measuredAtEnd) {

    SensorDataFilterDto filterDto = new SensorDataFilterDto();
    filterDto.setIds(ids);
    filterDto.setDeviceIds(deviceIds);

filterDto.setTemperatureDs18b20Start(temperatureDs18b20Start);
    filterDto.setTemperatureDs18b20End(temperatureDs18b20End);
    filterDto.setTemperatureDht11Start(temperatureDht11Start);
    filterDto.setTemperatureDht11End(temperatureDht11End);
    filterDto.setHumidityStart(humidityStart);
    filterDto.setHumidityEnd(humidityEnd);
    filterDto.setSoundDetected(soundDetected);
    filterDto.setLightLevelStart(lightLevelStart);
    filterDto.setLightLevelEnd(lightLevelEnd);
    filterDto.setCoPpmStart(coPpmStart);
    filterDto.setCoPpmEnd(coPpmEnd);
    filterDto.setWaterLevelStart(waterLevelStart);
    filterDto.setWaterLevelEnd(waterLevelEnd);
    filterDto.setMeasuredAtStart(measuredAtStart);
    filterDto.setMeasuredAtEnd(measuredAtEnd);

    return logicService.filterSensorData(filterDto);
}

@GetMapping("/report")
public ResponseEntity<byte[]> getReport(
    @Parameter(name = "ids", description = "List of sensor
IDs", in = ParameterIn.QUERY, schema = @Schema(type = "array",
format = "uuid"))
    @RequestParam(required = false) List<UUID> ids,

    @Parameter(name = "deviceIds", description = "List of
device IDs", in = ParameterIn.QUERY, schema = @Schema(type =
"array", format = "uuid"))
    @RequestParam(required = false) List<UUID>
deviceIds,

    @Parameter(description = "Min temperature (DS18B20)")
    @RequestParam(required = false) Double
temperatureDs18b20Start,

    @Parameter(description = "Max temperature (DS18B20)")
    @RequestParam(required = false) Double
temperatureDs18b20End,

```

```

        @Parameter(description = "Min temperature (DHT11)")
        @RequestParam(required = false) Integer
temperatureDht11Start,

        @Parameter(description = "Max temperature (DHT11)")
        @RequestParam(required = false) Integer
temperatureDht11End,

        @Parameter(description = "Min humidity")
        @RequestParam(required = false) Integer
humidityStart,

        @Parameter(description = "Max humidity")
        @RequestParam(required = false) Integer
humidityEnd,

        @Parameter(description = "Sound detected flag")
        @RequestParam(required = false) Boolean
soundDetected,

        @Parameter(description = "Min light level")
        @RequestParam(required = false) Integer
lightLevelStart,

        @Parameter(description = "Max light level")
        @RequestParam(required = false) Integer
lightLevelEnd,

        @Parameter(description = "Min CO ppm")
        @RequestParam(required = false) Double coPpmStart,

        @Parameter(description = "Max CO ppm")
        @RequestParam(required = false) Double coPpmEnd,

        @Parameter(description = "Min water level")
        @RequestParam(required = false) Integer
waterLevelStart,

        @Parameter(description = "Max water level")
        @RequestParam(required = false) Integer
waterLevelEnd,

        @Parameter(description = "Start time for filtering",
schema = @Schema(type = "string", format = "date-time"))
        @RequestParam(required = false)
@DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime
measuredAtStart,

        @Parameter(description = "End time for filtering",
schema = @Schema(type = "string", format = "date-time"))

```

```

        @RequestParam(required = false)
        @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime
        measuredAtEnd) {

            SensorDataFilterDto filterDto = new SensorDataFilterDto();
            filterDto.setIds(ids);
            filterDto.setDeviceIds(deviceIds);

            filterDto.setTemperatureDs18b20Start(temperatureDs18b20Start);
            filterDto.setTemperatureDs18b20End(temperatureDs18b20End);
            filterDto.setTemperatureDht11Start(temperatureDht11Start);
            filterDto.setTemperatureDht11End(temperatureDht11End);
            filterDto.setHumidityStart(humidityStart);
            filterDto.setHumidityEnd(humidityEnd);
            filterDto.setSoundDetected(soundDetected);
            filterDto.setLightLevelStart(lightLevelStart);
            filterDto.setLightLevelEnd(lightLevelEnd);
            filterDto.setCoPpmStart(coPpmStart);
            filterDto.setCoPpmEnd(coPpmEnd);
            filterDto.setWaterLevelStart(waterLevelStart);
            filterDto.setWaterLevelEnd(waterLevelEnd);
            filterDto.setMeasuredAtStart(measuredAtStart);
            filterDto.setMeasuredAtEnd(measuredAtEnd);

            byte[] bytes = pdfService.generatePDFReport(filterDto);

            return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION,
                    "attachment; filename=report.pdf")
                .contentType(MediaType.APPLICATION_PDF)
                .body(bytes);
        }

        @GetMapping("/analysis")
        public SensorAnalysisResult
        getAnalysisForDevice(@RequestParam(required = true) UUID
        deviceId) {
            return logicService.analyzeDevice(deviceId);
        }
    }
}

```

Листинг А.7 – Файл UserCardSettingsController.java

```

@RestController
@RequestMapping("/api/user-settings")
public class UserCardSettingsController {
    private final UserCardSettingsService service;

    public UserCardSettingsController(UserCardSettingsService
    service) { this.service = service; }
}

```

```

    @GetMapping("/{userId}")
    public ResponseEntity<List<UserCardSettings>>
getSettings(@PathVariable UUID userId) {
        return ResponseEntity.ok(service.getSettings(userId));
    }

    @PostMapping("/{userId}/{cardType}")
    public ResponseEntity<UserCardSettings> saveSettings(
        @PathVariable UUID userId,
        @PathVariable String cardType,
        @RequestBody String optionsJson) {
        UserCardSettings saved = service.upsertSettings(userId,
cardType, optionsJson);
        return ResponseEntity.ok(saved);
    }
}

```

Листинг A.8 – Файл UserController.java

```

@RestController
@RequestMapping("/api/users")
@Tag(name = "User Management", description = "Управление
пользователями")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;
    private final DeviceService deviceService;
    @GetMapping("/")
    @Operation(summary = "Получить всех пользователей",
description = "Возвращает список пользователей с возможностью
фильтрации")
    public List<User> getUsers(
        @ModelAttribute UserFilter userFilter
    ) {
        return userService.getUserByFilter(userFilter);
    }
    @PostMapping("/")
    @Operation(summary = "Получить всех пользователей",
description = "Возвращает список пользователей с возможностью
фильтрации")
    public List<User> getUsersByPost(
        @RequestBody UserFilter userFilter
    ) {
        return userService.getUserByFilter(userFilter);
    }

    @PostMapping("/register/device")
    @Operation(summary = "Получить всех пользователей",
description = "Возвращает список пользователей с возможностью
фильтрации")
    public void registerDevice(@RequestBody DeviceDataDto
deviceDataDto){

```

```

        deviceService.registerDevice(deviceDataDto);
    }

    @GetMapping("/device/get-by-name")
    @Operation(summary = "Получить девайс по имени", description =
"Возвращает список пользователей с возможностью фильтрации")
    public DeviceDto getDeviceByName(@RequestParam String name){
        return deviceService.getDeviceByName(name);
    }
    @GetMapping("/device/get-by-userId")
    @Operation(summary = "Получить девайсы по userId", description =
"Возвращает список устройств по userId")
    public List<DeviceDto> getDevicesByUserId(@RequestParam UUID
userId){
        return deviceService.getDevicesByUserId(userId);
    }
}

```

Листинг А.9 – Файл UserRole.java

```

public enum UserRole {
    ADMIN,
    USER,
    MODERATOR
}

```

Листинг А.10 – Файл User.java

```

@Entity
@Table(name = "users")
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(columnDefinition = "uuid DEFAULT gen_random_uuid()")
    private UUID id;

    @Column(name = "device_id", unique = true, nullable = false)
    private UUID deviceId;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private UserRole role = UserRole.USER;
}

```

```

        @Column(name = "created_at", nullable = false, updatable =
false)
        private LocalDateTime createdAt = LocalDateTime.now();

        @Column(nullable = false)
        private boolean online;
    }

```

Листинг A.11 – Файл UserCardSettings.java

```

@Entity
@Table(name = "user_card_settings", uniqueConstraints =
@UniqueConstraint(columnNames = {"user_id", "card_type"}))
@Data
public class UserCardSettings {

    @Id
    @GeneratedValue
    private UUID id;

    @Column(name = "user_id", nullable = false)
    private UUID userId;

    @Column(name = "card_type", length = 50, nullable = false)
    private String cardType;

    @Column(columnDefinition = "jsonb", nullable = false)
    private String options;

    @Column(name = "created_at", nullable = false, updatable =
false)
    private OffsetDateTime createdAt = OffsetDateTime.now();

    @Column(name = "updated_at", nullable = false)
    private OffsetDateTime updatedAt = OffsetDateTime.now();

    @PreUpdate
    public void preUpdate() { this.updatedAt =
OffsetDateTime.now(); }

}

```

Листинг A.12 – Файл DeviceDataMapper.java

```

@Mapper(componentModel = "spring", unmappedTargetPolicy =
ReportingPolicy.IGNORE, injectionStrategy =
InjectionStrategy.CONSTRUCTOR)
public interface DeviceDataMapper {

    DeviceDataMapper INSTANCE =
Mappers.getMapper(DeviceDataMapper.class);
}

```

```

    @Mapping(target = "deviceId", source = "deviceId")
    @Mapping(target = "userId", source = "userId")
    @Mapping(target = "status", source = "status", qualifiedByName
= "statusToDto")
    @Mapping(target = "type", source = "type", qualifiedByName =
"typeToDto")
    DeviceDataDto toDto(DeviceData deviceData);

    @Mapping(target = "deviceId", source = "deviceId")
    @Mapping(target = "userId", source = "userId")
    @Mapping(target = "status", source = "status", qualifiedByName
= "statusToProto")
    @Mapping(target = "type", source = "type", qualifiedByName =
"typeToProto")
    DeviceData toProto(DeviceDataDto deviceDataDto);

    @Mapping(target = "registeredAt", source = "registeredAt",
qualifiedByName = "timestampToLocalDateTime")
    DeviceDto toDeviceDto(DeviceResponse deviceResponse);

    @Named("statusToProto")
    default com.example.diploma_logic.grpc.Status
statusToProto(Status status) {
        return switch (status) {
            case ACTIVE ->
com.example.diploma_logic.grpc.Status.ACTIVE;
            case REBOOTING ->
com.example.diploma_logic.grpc.Status.REBOOTING;
            case INACTIVE ->
com.example.diploma_logic.grpc.Status.INACTIVE;
        };
    }
    @Named("timestampToLocalDateTime")
    default LocalDateTime timestampToLocalDateTime(Timestamp
timestamp) {
        return LocalDateTime.ofEpochSecond(timestamp.getSeconds(),
timestamp.getNanos(),
ZoneId.of("Europe/Moscow").getRules().getOffset(LocalDateTime.now(
))) );
    }
    @Named("typeToProto")
    default com.example.diploma_logic.grpc.Type typeToProto(Type
type) {
        return switch (type) {
            case ESP32_TYPE_C ->
com.example.diploma_logic.grpc.Type.ESP32_TYPE_C;
            case ESP32_MICRO_USB ->
com.example.diploma_logic.grpc.Type.ESP32_MICRO_USB;

```

```

        case ESP32_ETHERNET ->
com.example.diploma_logic.grpc.Type.ESP32_ETHERNET;
    };
}

@Named("statusToDto")
default Status
statusToDto(com.example.diploma_logic.grpc.Status status) {
    return switch (status) {
        case ACTIVE -> Status.ACTIVE;
        case REBOOTING -> Status.REBOOTING;
        case INACTIVE -> Status.INACTIVE;
        default -> throw new IllegalArgumentException("Unknown
status: " + status);
    };
}

@Named("typeToDto")
default Type typeToDto(com.example.diploma_logic.grpc.Type
type) {
    return switch (type) {
        case ESP32_TYPE_C -> Type.ESP32_TYPE_C;
        case ESP32_MICRO_USB -> Type.ESP32_MICRO_USB;
        case ESP32_ETHERNET -> Type.ESP32_ETHERNET;
        default -> throw new IllegalArgumentException("Unknown
type: " + type);
    };
}
}

```

Листинг A.13 – Файл NotificationsMapper.java

```

@Mapper(componentModel = "spring", unmappedTargetPolicy =
ReportingPolicy.IGNORE, injectionStrategy =
InjectionStrategy.CONSTRUCTOR)
public interface NotificationMappers {

    @Mapping(target = "userId", source = "userId", qualifiedByName
= "uuidToString")
    NotificationSettingsRequest
toProtoNotificationSettingsRequest(NotificationSettingsDto dto);

    @Mapping(target = "userId", source = "userId", qualifiedByName
= "stringToUuid")
    NotificationSettingsDto
toNotificationSettingsDto(NotificationSettingsResponse response);

    @Named("uuidToString")
    default String uuidToString(UUID uuid) {
        if(uuid == null){
            return null;
        }else {

```

```

        return uuid.toString();
    }

}

@Named("stringToUuid")
default UUID stringToUuid(String uuidStr) {
    try {
        return UUID.fromString(uuidStr);
    } catch (IllegalArgumentException e) {
        return null;
    }
}
}

```

Листинг A.14 – Файл SensorDataMapper.java

```

@Mapper(componentModel = "spring", unmappedTargetPolicy =
ReportingPolicy.IGNORE, injectionStrategy =
InjectionStrategy.CONSTRUCTOR)
public interface SensorDataMapper {

    @Mapping(target = "temperatureDs18B20", source =
"temperatureDs18b20")
    @Mapping(target = "temperatureDht11", source =
"temperatureDht11")
    @Mapping(target = "coPpm", source = "coPpm")
    @Mapping(target = "deviceId", source = "deviceId",
qualifiedByName = "uuidToString")
    @Mapping(target = "humidity", source = "humidity")
    @Mapping(target = "soundDetected", source = "soundDetected")
    @Mapping(target = "lightLevel", source = "lightLevel")
    @Mapping(target = "waterLevel", source = "waterLevel")
    com.example.diploma_logic.grpc.SensorData
toGrpcSensorData(SensorData dto);

    @Mapping(target = "deviceId", source = "deviceId",
qualifiedByName = "uuidToString")
    @Mapping(target = "startTime", source = "startTime",
qualifiedByName = "instantToTimestamp")
    @Mapping(target = "endTime", source = "endTime",
qualifiedByName = "instantToTimestamp")
    @Mapping(target = "avgTemperatureDs18B20", source =
"avgTemperatureDs18b20")
    @Mapping(target = "avgTemperatureDht11", source =
"avgTemperatureDht11")
    @Mapping(target = "avgHumidity", source = "avgHumidity")
    @Mapping(target = "avgLightLevel", source = "avgLightLevel")
    @Mapping(target = "avgCoPpm", source = "avgCoPpm")
    @Mapping(target = "avgWaterLevel", source = "avgWaterLevel")
    @Mapping(target = "soundEventsCount", source =
"soundEventsCount")
}

```

```

    @Mapping(target = "waterAlertsCount", source =
"waterAlertsCount")
    com.example.diploma_logic.grpc.AggregatedSensorData
toGrpcAggregatedData (AggregatedSensorData dto);

    @Mapping(
        target = "startTime",
        source = "measuredAtStart",
        qualifiedByName = "mapStartTime",
        nullValueCheckStrategy = NullValueCheckStrategy.ALWAYS
    )
    @Mapping(
        target = "endTime",
        source = "measuredAtEnd",
        qualifiedByName = "mapEndTime",
        nullValueCheckStrategy = NullValueCheckStrategy.ALWAYS
    )
    SensorDataFilter toProtoSensorDataFilter(SensorDataFilterDto
dto);

    @Named("mapStartTime")
    default Timestamp mapStartTime(LocalDateTime measuredAtStart)
    {
        return measuredAtStart != null ?
localDateTimeToTimestamp(measuredAtStart) : null;
    }

    @Named("mapEndTime")
    default Timestamp mapEndTime(LocalDateTime measuredAtEnd) {
        return measuredAtEnd != null ?
localDateTimeToTimestamp(measuredAtEnd) : null;
    }

    @Mapping(target = "temperatureDs18b20", source =
"temperatureDs18B20")
    //    @Mapping(target = "temperatureDht11", source =
"temperature_dht11")
    //    @Mapping(target = "humidity", source = "humidity")
    //    @Mapping(target = "soundDetected", source = "soundDetected")
    //    @Mapping(target = "lightLevel", source = "lightLevel")
    //    @Mapping(target = "coPpm", source = "co_ppm")
    //    @Mapping(target = "waterLevel", source = "waterLevel")
    //    @Mapping(target = "deviceId", source = "device_id",
qualifiedByName = "stringToUuid")
    //    @Mapping(target = "id", source = "id", qualifiedByName =
"stringToUuid")
    @Mapping(target = "measuredAt", source = "measuredAt",
qualifiedByName = "timestampToLocalDateTime")
    SensorDataDto toSensorDataDto(SensorDataInfo sensorDataInfo);

```

```

    @Mapping(target = "analysisAt", source = "analysisAt",
qualifiedByName = "timestampToLocalDateTime")
    SensorAnalysisResult toSensorAnalysisResult(AnalyzeResponse
analyzeResponse);

    @Named("instantToTimestamp")
    default Timestamp instantToTimestamp(Instant instant) {
        return Timestamp.newBuilder()
            .setSeconds(instant.getEpochSecond())
            .setNanos(instant.getNano())
            .build();
    }

    @Named("uuidToStringList")
    default List<String> uuidToStringList(List<UUID> uuids) {
        if (uuids == null) {
            return null;
        }
        return uuids.stream()
            .map(UUID::toString)
            .collect(Collectors.toList());
    }

    @Named("timestampToInstant")
    default Instant timestampToInstant(Timestamp timestamp) {
        return Instant.ofEpochSecond(timestamp.getSeconds(),
timestamp.getNanos());
    }

    @Named("timestampToLocalDateTime")
    default LocalDateTime timestampToLocalDateTime(Timestamp
timestamp) {
        return LocalDateTime.ofEpochSecond(timestamp.getSeconds(),
            timestamp.getNanos(),
ZoneId.of("Europe/Moscow").getRules().getOffset(LocalDateTime.now(
)));
    }

    @Named("uuidToString")
    default String uuidToString(UUID uuid) {
        if(uuid == null){
            return null;
        }else {
            return uuid.toString();
        }
    }

    @Named("stringToUuid")
    default UUID stringToUuid(String uuidStr) {
        try {
            return UUID.fromString(uuidStr);
        }
    }

```

```

        } catch (IllegalArgumentException e) {
            return null;
        }
    }
    @Named("localDateTimeToTimestamp")
    public static Timestamp localDateTimeToTimestamp(LocalDateTime
localDateTime) {
        if (localDateTime == null) {
            return null;
        }

        long seconds = localDateTime.toEpochSecond(ZoneOffset.UTC);
        int nanos = localDateTime.getNano();

        return Timestamp.newBuilder()
            .setSeconds(seconds)
            .setNanos(nanos)
            .build();
    }
}

```

Листинг A.15 – Файл UserCardSettingsRepository.java

```

@Repository
public interface UserCardSettingsRepository extends
JpaRepository<UserCardSettings, UUID> {
    List<UserCardSettings> findAllByUserId(UUID userId);
    Optional<UserCardSettings> findByUserIdAndCardType(UUID userId,
String cardType);
}

```

Листинг A.16 – Файл UserRepository.java

```

@Repository
public interface UserRepository extends JpaRepository<User, UUID>,
JpaSpecificationExecutor<User> {

}

```

Листинг A.17 – Файл DeviceService.java

```

@Service
@RequiredArgsConstructor
public class DeviceService {
    //private final DeviceDataMapper deviceDataMapper;
    private final ManagedChannel channel = ManagedChannelBuilder
        .forAddress("localhost", 9092)
        .usePlaintext()
        .build();
}

```

```

        private final DeviceServiceGrpc.DeviceServiceBlockingStub
deviceServiceBlockingStub =
DeviceServiceGrpc.newBlockingStub(channel);

        public void registerDevice(DeviceDataDto deviceDataDto) {
            DeviceData request =
DeviceDataMapper.INSTANCE.toProto(deviceDataDto);

GrpcClientUtils.doGrpcCall(request, deviceServiceBlockingStub::regi
sterDevice);
        }

        public DeviceDto getDeviceByName(String name) {
            DeviceNameRequest request =
DeviceNameRequest.newBuilder().setName(name).build();
            DeviceResponse deviceResponse =
GrpcClientUtils.doGrpcCall(request,
deviceServiceBlockingStub::getDeviceByName);
            return
DeviceDataMapper.INSTANCE.toDeviceDto(deviceResponse);
        }

        public List<DeviceDto> getDevicesByUserId(UUID userId) {
            UserIdLogicRequest request =
UserIdLogicRequest.newBuilder().setUserId(userId.toString()).build
();
            DeviceResponseList deviceResponseList =
GrpcClientUtils.doGrpcCall(request,
deviceServiceBlockingStub::getDevicesByUserId);
            return
deviceResponseList.getDeviceListList().stream().map(DeviceDataMapp
er.INSTANCE::toDeviceDto).toList();
        }

    }

```

Листинг A.18 – Файл LogicService.java

```

@Service
@RequiredArgsConstructor
public class LogicService {
    private final SensorDataMapper sensorDataMapper;
    private final ManagedChannel channel = ManagedChannelBuilder
        .forAddress("localhost", 9092)
        .usePlaintext()
        .build();
    private final LogicServiceGrpc.LogicServiceBlockingStub
logicServiceBlockingStub =
LogicServiceGrpc.newBlockingStub(channel);
    public void test() {

```

```

logicServiceBlockingStub.testRpc(Empty.getDefaultInstance());
    }

    public void sendAggregatedData(AggregatedSensorData
aggregatedSensorData) {

GrpcClientUtils.doGrpcCall(sensorDataMapper.toGrpcAggregatedData(a
ggregatedSensorData), logicServiceBlockingStub::saveAggregatedSenso
rData); //возможно исправить на async

    }
    public void sendSensorData(SensorData sensorData) {
        com.example.diploma_logic.grpc.SensorData grpcSensorData =
sensorDataMapper.toGrpcSensorData(sensorData);

GrpcClientUtils.doGrpcCall(grpcSensorData, logicServiceBlockingStub
::saveSensorData);
    }

    public List<SensorDataDto>
filterSensorData(SensorDataFilterDto sensorDataFilterDto) {

        SensorDataFilter protoSensorDataFilter =
sensorDataMapper.toProtoSensorDataFilter(sensorDataFilterDto);
        System.out.println("ccc = " + protoSensorDataFilter);
        SensorDataResponse sensorDataResponse =
GrpcClientUtils.doGrpcCall(protoSensorDataFilter,
logicServiceBlockingStub::getSensorData);

public List<SensorDataDto> filterSensorData(SensorDataFilterDto
sensorDataFilterDto) {

        SensorDataFilter protoSensorDataFilter =
sensorDataMapper.toProtoSensorDataFilter(sensorDataFilterDto);
        System.out.println("ccc = " + protoSensorDataFilter);
        SensorDataResponse sensorDataResponse =
GrpcClientUtils.doGrpcCall(protoSensorDataFilter,
logicServiceBlockingStub::getSensorData);

        return
sensorDataResponse.getSensorDataInfoList().stream().map(sensorData
Mapper::toSensorDataDto).collect(Collectors.toList());
    }

    public SensorAnalysisResult analyzeDevice(UUID deviceId) {
        DeviceIdRequest deviceIdRequest =
DeviceIdRequest.newBuilder()
            .setDeviceId(deviceId.toString()).build();
    }

```

```

        return
        sensorDataMapper.toSensorAnalysisResult(GrpcClientUtils.doGrpcCall
        (deviceIdRequest, logicServiceBlockingStub::analyzeDevice));
    }
}

```

Листинг А.19 – Файл NotificationService.java

```

@Service
@RequiredArgsConstructor
public class NotificationService {
    private final NotificationMappers notificationMappers;
    private final ManagedChannel channel = ManagedChannelBuilder
        .forAddress("localhost", 9093)
        .usePlaintext()
        .build();

    private final
    NotificationServiceGrpc.NotificationServiceBlockingStub
    notificationServiceBlockingStub =
    NotificationServiceGrpc.newBlockingStub(channel);

    private final
    NotificationServiceGrpc.NotificationServiceBlockingStub
    notificationServiceBlockingStub =
    NotificationServiceGrpc.newBlockingStub(channel);

    public void
    saveOrUpdateNotificationSettings(NotificationSettingsDto
    notificationSettingsDto) {

        NotificationSettingsRequest
    protoNotificationSettingsRequest =
    notificationMappers.toProtoNotificationSettingsRequest(notificatio
    nSettingsDto);

    GrpcClientUtils.doGrpcCall(protoNotificationSettingsRequest, notifi
    cationServiceBlockingStub::saveNotificationSettings);
    }

    public NotificationSettingsDto getSettingsForUser(UUID
    userId) {
        UserIdRequest request =
    UserIdRequest.newBuilder().setUserId(userId.toString()).build();
        NotificationSettingsResponse notificationSettingsResponse
    = GrpcClientUtils.doGrpcCall(request,
    notificationServiceBlockingStub::getSettingsForUser);
        NotificationSettingsDto notificationSettingsDto =
    notificationMappers.toNotificationSettingsDto(notificationSettings
    Response);
        return notificationSettingsDto;
    }
}

```

```

        public void sendSoundAlert(UUID userId, UUID deviceId, boolean
isSound) {
            SoundAlertRequest request = SoundAlertRequest.newBuilder()
                .setUserId(userId.toString())
                .setDeviceId(deviceId.toString())
                .setSoundDetected(isSound).build();

GrpcClientUtils.doGrpcCall(request, notificationServiceBlockingStub
::sendSoundAlert);
        }

        public void sendWaterLevelAlert(UUID userId, UUID deviceId, int
waterLevel) {
            WaterLevelAlertRequest request =
WaterLevelAlertRequest.newBuilder()
                .setUserId(userId.toString())
                .setDeviceId(deviceId.toString())
                .setWaterLevel(waterLevel)
                .build();
GrpcClientUtils.doGrpcCall(request, notificationServiceBlockingStub
::sendWaterLevelAlert);
        }
    }
}

```

Приложение Б
(рекомендуемое)
Графическая часть

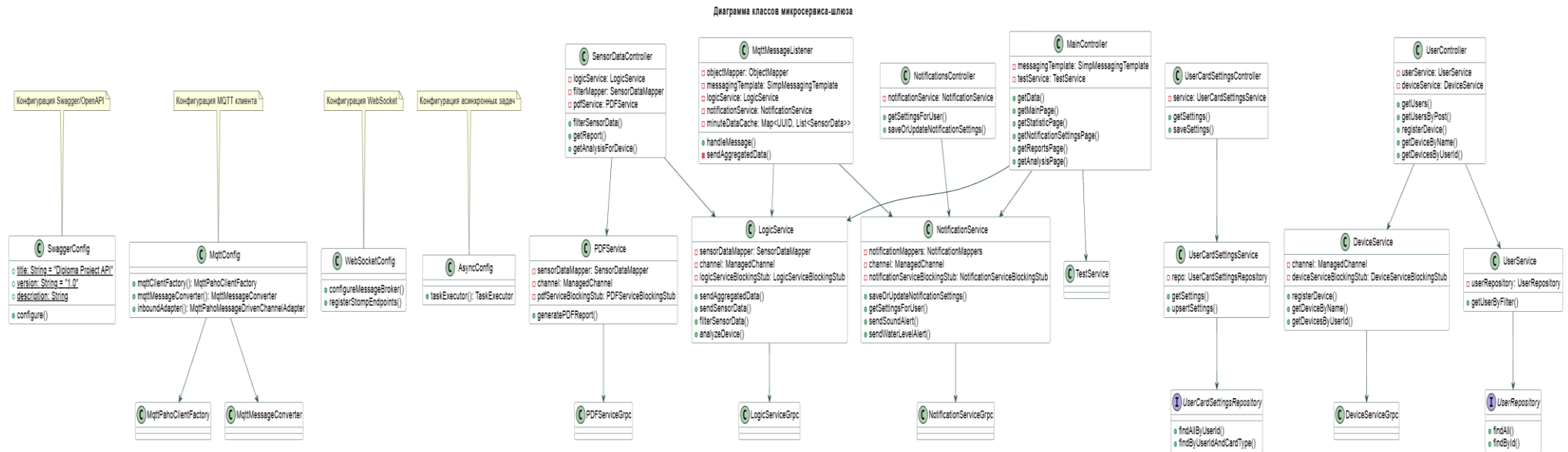


Рисунок Б.1 – Диаграмма классов микросервиса «Шлюз»

Диаграмма классов микросервиса Логика

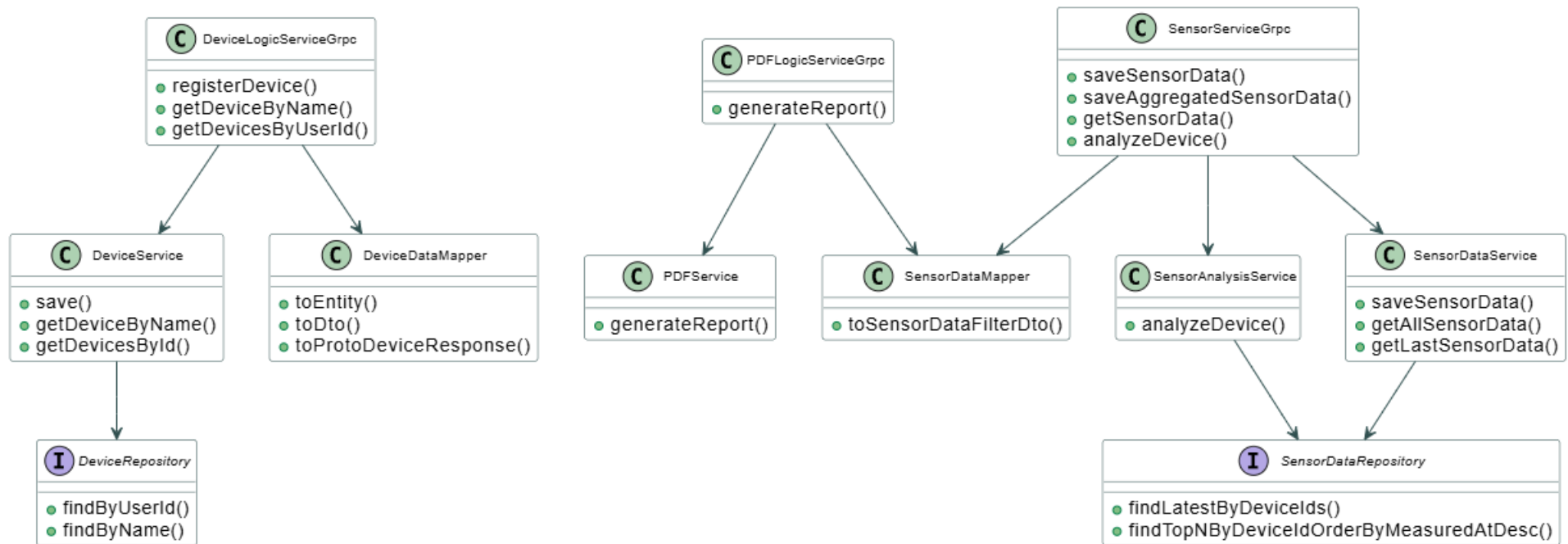


Рисунок Б.2 – Диаграмма классов микросервиса «Логика»

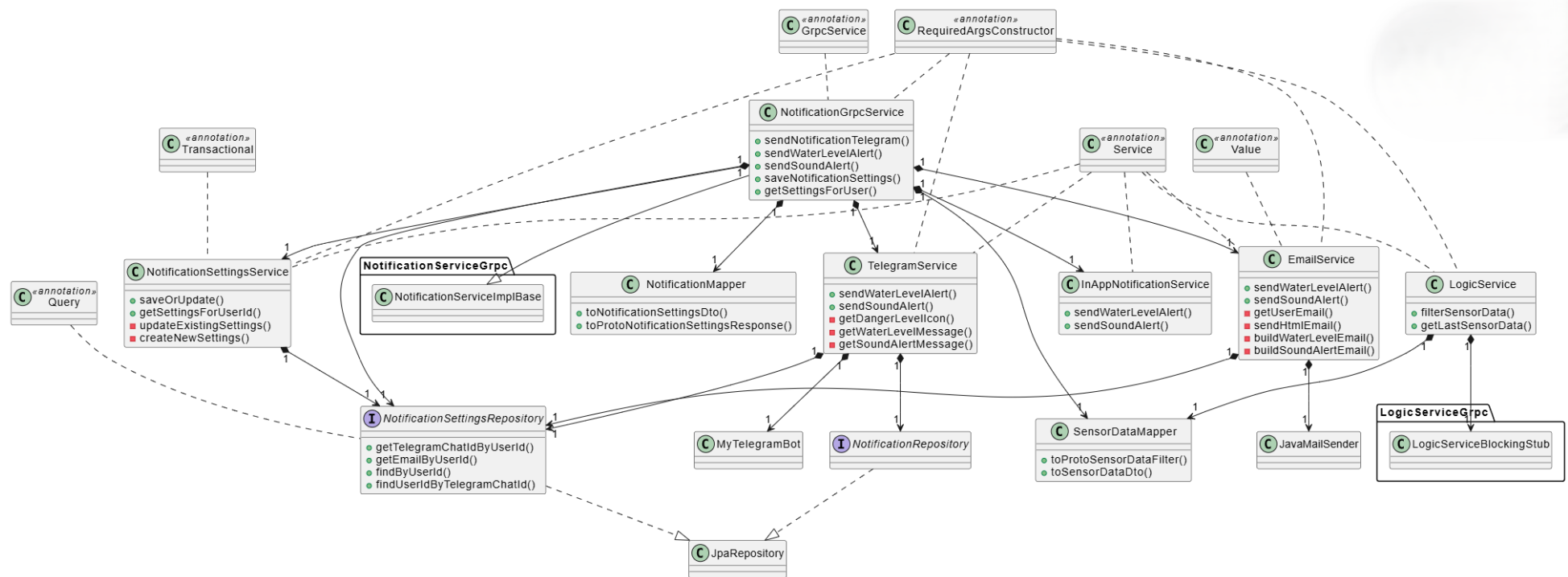


Рисунок Б.3 – Диаграмма классов микросервиса «Уведомления»

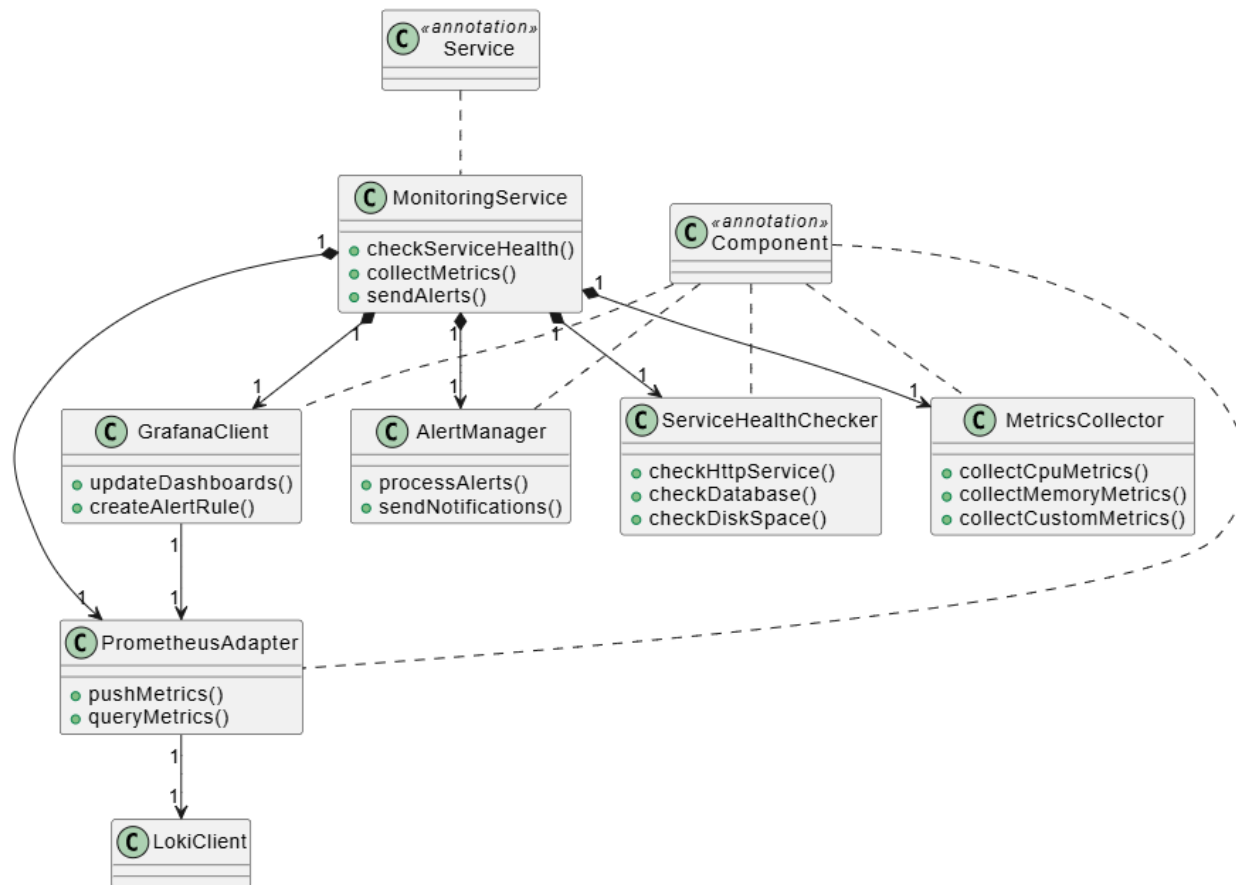


Рисунок Б.4 – Диаграмма классов микросервиса «Мониторинг»

Приложение В
(рекомендуемое)
Атрибуты сущностей базы данных

Таблица В.1 – Атрибуты отношения «users»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор пользователя	UUID	-	RFC 4122	Нет	Первичный ключ	a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
device_id	Привязанный идентификатор устройства	UUID	-	RFC 4122	Нет	Уникальный идентификатор	b1eebc88-8d0a-3df8-cc7d-7cc8ac390b22
email	Электронная почта пользователя	varchar	255	Валидный email-формат	Нет	Уникальный логин	user@example.com
password	Хэш пароля	varchar	255	Любая непустая строка	Нет	Аутентификация	\$2a\$10\$x...
role	Роль пользователя	varchar	20	'USER', 'ADMIN', 'SUPPORT'	Нет	Авторизация	USER
created_at	Дата регистрации	timestamp	-	01.01.2020-текущая дата	Нет	Аудит	2023-05-15 14:30:22
online	Статус активности	boolean	-	true/false	Нет	Мониторинг	true

Таблица В.2 – Атрибуты отношения «user_card_settings»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор настройки	UUID	-	RFC 4122	Нет	Первичный ключ	c2ffd134-1a62-4597-bf28-5e1b9abd1d1a
user_id	Ссылка на пользователя	UUID	-	Существующий users.id	Нет	Внешний ключ	a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
card_type	Тип карточки интерфейса	varchar	50	'DASHBOARD', 'SENSORS', 'REPORTS'	Нет	Классификация	DASHBOARD

Продолжение таблицы В.2

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
options	Настройки карточки	jsonb	-	Валидный JSON	Нет	Конфигурация	{"theme":"dark","columns":3}
created_at	Дата создания	timestamptz	-	01.01.2020-текущая дата	Нет	Аудит	2023-05-16 09:12:45+03
updated_at	Дата обновления	timestamptz	-	01.01.2020-текущая дата	Нет	Аудит	2023-05-20 11:30:15+03

Таблица В.3 – Атрибуты отношения «devices»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор устройства	UUID	-	RFC 4122	Нет	Первичный ключ	d3eebc77-7c0c-5df7-dd6e-8dd7cd491e33
user_id	Владелец устройства	UUID	-	Существующий users.id	Нет	Связь с пользователем	a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
name	Пользовательское имя устройства	varchar	255	Любая строка	Да	Идентификация	"Кухонный датчик"
registered_at	Дата регистрации	timestamptz	-	01.01.2020-текущая дата	Нет	Аудит	2023-05-18 16:45:10+03
status	Текущий статус	device_status	-	'ACTIVE', 'REBOOTING', 'INACTIVE'	Нет	Мониторинг	ACTIVE
type	Тип устройства	device_type	-	'ESP32_TYPE_C', 'ESP32_MICRO_USB'	Нет	Классификация	ESP32_TYPE_C

Таблица В.4 – Атрибуты отношения «sensor_data»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор записи	UUID	-	RFC 4122	Нет	Первичный ключ	e4ffd245-2b73-4698-cf39-6f2a8bce2d2b
device_id	Ссылка на устройство	UUID	-	Существующий devices.id	Нет	Внешний ключ	d3eebc77-7c0c-5df7-dd6e-8dd7cd491e33
temperature_ds18b20	Показание температуры (DS18B20)	decimal	10,2	-55.00..+125.00	Да	Метеоданные	23.50
temperature_dht11	Показание температуры (DHT11)	integer	-	0..50	Да	Метеоданные	24
humidity	Влажность воздуха	integer	-	20..90	Да	Метеоданные	45
sound_detected	Факт обнаружения звука	boolean	-	true/false	Да	Аудиомониторинг	true
light_level	Уровень освещенности	integer	-	0..1023	Да	Фотометрия	560
co_ppm	Концентрация CO	decimal	10,2	0.00..1000.00	Да	Газовый анализ	12.50
water_level	Уровень воды	integer	-	0..500	Да	Гидрометрия	0
measured_at	Время измерения	timestamp tz	-	01.01.2020-текущая дата	Нет	Временная метка	2023-05-20 14:25:30+03

Таблица В.5 – Атрибуты отношения «notifications»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор уведомления	UUID	-	RFC 4122	Нет	Первичный ключ	f5ffd356-3c84-5799-dg40-7g3b9cdf3e4c
user_id	Получатель уведомления	UUID	-	Любой валидный UUID	Нет	Адресация	a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
notification_type	Тип уведомления	varchar	50	'TELEGRAM', 'EMAIL', 'TN_APP'	Нет	Классификация	TELEGRAM
content	Содержимое уведомления	jsonb	-	Валидный JSON	Нет	Данные события	{"message": "Высокий CO", "value": 50}
sent_at	Время отправки	timestampz	-	01.01.2020-текущая дата	Нет	Временная метка	2023-05-21 08:15:22+03
is_read	Флаг прочтения	boolean	-	true/false	Нет	Статус	false

Таблица В.6 – Атрибуты отношения «notification_settings»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор настройки	UUID	-	RFC 4122	Нет	Первичный ключ	g6eed447-4d95-6800-eh51-8h4c0deg4f5d
user_id	Ссылка на пользователя	UUID	-	Любой валидный UUID	Нет	Уникальная привязка	a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
email	Email для уведомлений	varchar	255	Валидный email-формат	Да	Контактные данные	alert@example.com
telegram_chat_id	ID чата в Telegram	varchar	100	Числовой ID (строка)	Да	Контактные данные	"123456789"
enable_email_notifications	Флаг email-уведомлений	boolean	-	true/false	Нет	Настройка	true
enable_telegram_notifications	Флаг Telegram	boolean	-	true/false	Нет	Настройка	false

Продолжение таблицы В.6

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
enable_in_app_notifications	Флаг внутрисистемных уведомлений	boolean	-	true/false	Нет	Настройка	true
updated_at	Время последнего обновления	timestampz	-	01.01.2020-текущая дата	Нет	Аудит	2023-05-20 17:45:10+03

Таблица В.7 – Атрибуты отношения «services»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор сервиса	UUID	-	RFC 4122	Нет	Первичный ключ	h7ffd558-5ea6-6911-fi62-9i5d1efg5h6e
name	Название сервиса	varchar	100	Любая непустая строка	Нет	Идентификация	"MQTT Broker"
description	Описание сервиса	text	-	Любой текст	Да	Документация	"Сервис обработки сообщений IoT"
status	Текущий статус	varchar	20	'UP', 'DOWN', 'DEGRADED', 'MAINTENANCE'	Нет	Мониторинг	UP
last_response_time	Время последнего ответа (мс)	integer	-	0..10000	Да	Производительность	120

Продолжение таблицы В.7

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
check_interval	Интервал проверки (мин)	integer	-	1..1440	Нет	Конфигурация	5
id	Уникальный идентификатор сервиса	UUID	-	RFC 4122	Нет	Первичный ключ	h7ffd558-5ea6-6911-fi62-9i5d1efg5h6e
name	Название сервиса	varchar	100	Любая непустая строка	Нет	Идентификация	"MQTT Broker"

Таблица В.8 – Атрибуты отношения «service_statuses»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор записи	UUID	-	RFC 4122	Нет	Первичный ключ	i8ggd669-6fb7-7022-gj73-0j6e2fgh6i7f
service_id	Ссылка на сервис	UUID	-	Существующий services.id	Нет	Внешний ключ	h7ffd558-5ea6-6911-fi62-9i5d1efg5h6e
checked_at	Время проверки	timestamp	-	01.01.2020-текущая дата	Нет	Временная метка	2023-05-21 12:30:45+03
status	Зафиксированный статус	varchar	20	'UP', 'DOWN', 'DEGRADED', 'MAINTENANCE'	Нет	Мониторинг	UP
error_message	Сообщение об ошибке	text	-	Любой текст	Да	Диагностика	"Connection timeout"
response_time	Время ответа (мс)	integer	-	0..10000	Да	Производительность	150

Таблица В.9 – Атрибуты отношения «device_statuses»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Уникальный идентификатор записи	UUID	-	RFC 4122	Нет	Первичный ключ	j9hhd770-7gc8-8133-hk84-1k7f3ghi7j8g
device_id	Ссылка на устройство	UUID	-	Любой валидный UUID	Нет	Идентификатор устройства	d3eebc77-7c0c-5df7-dd6e-8dd7cd491e33
status	Текущий статус устройства	varchar	20	'ONLINE', 'OFFLINE', 'ERROR', 'LOW BATTERY'	Нет	Мониторинг	ONLINE
checked_at	Время проверки	timestampz	-	01.01.2020- текущая дата	Нет	Временная метка	2023-05-21 13:45:30+03
additional_info	Дополнительная информация	jsonb	-	Валидный JSON	Да	Диагностика	{"battery_level":75, "signal_strength":"good"}

Приложение Г
(рекомендуемое)
Результаты тестирования

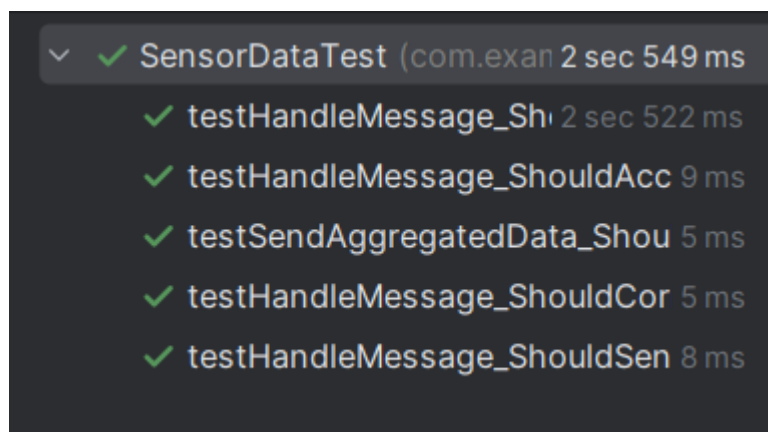


Рисунок Г.1 – Результаты тестирования компонента SensorData

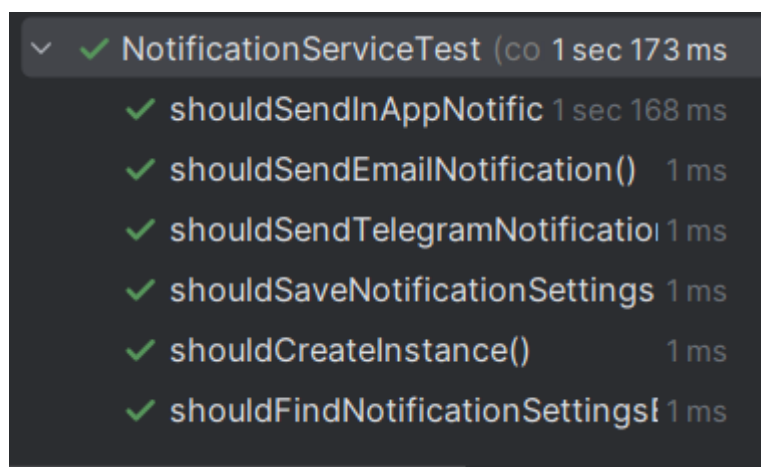


Рисунок Г.2 – Результаты тестирования компонента Notification

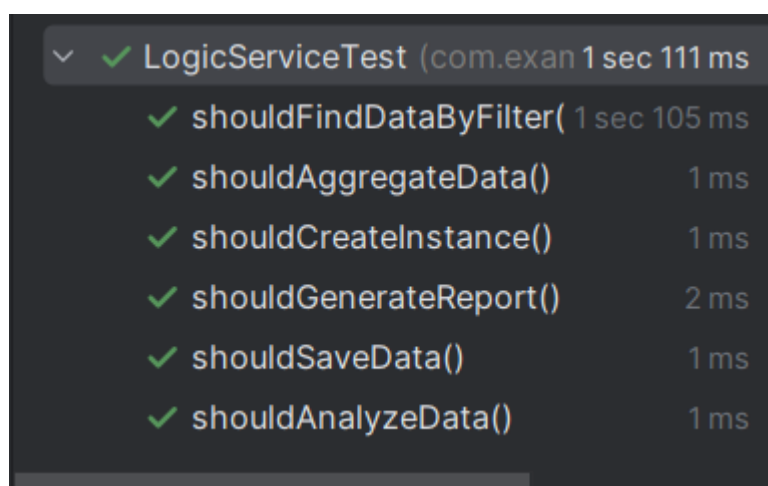


Рисунок Г.3 – Результаты тестирования компонента LogicComponent