

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

Факультет
вычислительной техники

Кафедра
МО и ПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

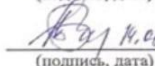
**Комплекс программных и
аппаратных средств мониторинга и
контроля микроклимата теплицы**

Студент


(подпись, дата)

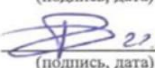
Торопкин Роман Александрович
(ФИО полностью)

Руководитель


(подпись, дата)

Балашова И.Ю.
(фамилия, инициалы)

Нормоконтролёр


(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Макарычев П.П.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2021

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
П.П. Макарычев
« 21 » января 2021 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Торопкин Роман Александрович гр. 17ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Комплекс программных и аппаратных средств мониторинга и
контроля микроклимата теплицы

Тема утверждена приказом ПГУ № 002496 от " 21 " января 2021 г.

2. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Балашова И.Ю.

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

- 1) Назначение: программно-аппаратный комплекс предназначен для контроля и мониторинга микроклимата теплицы путем автоматизации полива.
- 2) Среда выполнения: Unix-подобные операционные системы
Средства разработки: языки программирования Java, Python, JavaScript, SQL
Инструментальные средства: IntelliJ IDEA, PyCharm, DataGrip, Postman
Технологии разработки: ООП, UML
- 3) Основные функции программного обеспечения контроля и мониторинга микроклимата теплицы
 - а) считывать и отправлять показания с датчиков;
 - б) включить/отключить реле питания;
 - в) автоматизировать полив теплицы с помощью системы принятия решений;
 - г) отображать данные датчиков в режиме реального времени;
 - д) проводить настройку системы принятия решений;
 - е) выводить статистическую инфографику по датчикам;
 - ж) отправлять команды на исполнения контроллеру;
 - з) журналировать входящие события.

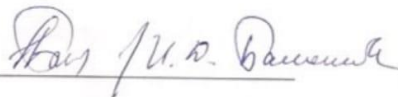
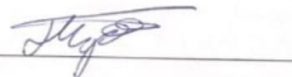
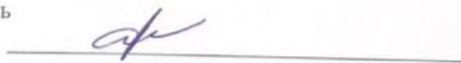
4. Перечень подлежащих разработке вопросов

*Анализ предметной области**Анализ требований**Проектирование**Реализация**Контроль качества ПО:**– Тестирование**– Анализ качества кода**План разработки и оценка бюджета*

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований	10	14.04.2021 – 20.04.2021	
2	Проектирование	25	20.04.2021 – 05.05.2021	
3	Реализация	35	05.05.2021 – 27.05.2021	
4	Контроль качества	15	27.05.2021 – 05.06.2021	
5	План разработки бюджета	5	05.06.2021 – 08.06.2021	
6	Оформление пояснительной записки	10	08.06.2021 – 14.06.2021	

Дата выдачи задания «28» декабря 2021 г.

Руководитель бакалаврской работы
(подпись, дата)Задание к исполнению принял студент
(подпись, дата)Работу к защите допустить
Декан факультета
(подпись, дата)

Реферат

Пояснительная записка содержит 98 листов, 47 рисунков, 15 использованных источников.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, КЛИЕНТ, СЕРВЕР, КОНТРОЛЛЕР, БАЗА ДАННЫХ, RASPBERRY PI 3, SQL, HTML, CSS, JAVASCRIPT, JAVA, SPRING, RABBITMQ, PYTHON, MYSQL, UML, ООП.

Объект разработки – программно-аппаратный комплекс контроля и микроклимата теплицы.

Цель работы – разработать серверное приложение, а также программу для контроллера для контроля и мониторинга микроклимата теплицы с целью автоматизации полива.

Технология разработки – ООП, UML.

Для создания серверной части применялись CASE-средство для проектирования баз данных CA Erwin Data Modeler 7.3, язык SQL и система управления базами данных MySQL 8. Для разработки серверной части использовался язык Java, а для программы контроллера – язык Python. Разработка велась в среде intellij idea и pycharm.

Результаты работы – в ходе выполнения выпускной квалификационной работы было создан программно-аппаратный комплекс контроля и мониторинга микроклимата теплицы. Все требования к разработке приложения соблюдены.

			ПГУ 09.03.04 – 8БР171.21 ПЗ		
№ докум.	Подпись	Дата	Комплекс программных и аппаратных средств мониторинга и контроля микроклимата теплицы	Лит.	Лист
Торошкин Р.А.		14.06.21			
Балишова И. Ю.		14.06.21			
Тактишкин Д. В.		27.06.21			
Макарычев П. П.					
					3
					98

Содержание

Введение	10
1 Анализ предметной области и требований комплекса мониторинга и контроля микроклимата теплицы.....	11
1.1 Анализ программно-аппаратного комплекса.....	11
1.1.1 Описание аппаратных компонентов комплекса	11
1.1.1.1 Описание датчика температуры и влажности.....	11
1.1.1.2 Описание датчика влажности почвы.....	13
1.1.1.3 Описание аналого-цифрового преобразователя	15
1.1.1.4 Описание датчика влажности почвы.....	16
1.1.1.5 Описание контроллера «Raspberry Pi 3 Model B».....	17
1.2 Анализ программных аналогов	18
1.3 Постановка задачи.....	21
1.4 Функциональное моделирование	24
2 Проектирование программных средств контроля и мониторинга микроклиматом теплицы	29
2.1 Базовая архитектура программно-аппаратного комплекса.....	29
2.2 Описание алгоритма автоматизированного управления поливом	30
2.3 Схема подключения и развертывания аппаратных компонентов.....	31
2.4 Средства разработки программного комплекса	32
2.5 Проектирование структуры данных программного комплекса	34
2.5.1 Разработка концептуальной модели базы данных.....	34
2.5.2 Логическое моделирование базы данных	36
2.5.3 Физическое моделирование базы данных.....	37
2.6 Проектирование программных средств	38
2.6.1 Проектирование серверного приложения	38
2.6.1.1 Диаграмма классов серверного приложения	38
2.6.1.2 Диаграмма деятельности серверного приложения	41
2.6.2 Проектирование приложения контроллера.....	42
2.6.2.1 Диаграмма классов приложения контроллера	42
2.6.2.2 Диаграмма деятельности приложения контроллера.....	44
2.6.3 Диаграмма последовательности программного комплекса	46

2.7	Проектирование графического интерфейса комплекса контроля и микроклимата теплицы	47
3	Реализация программного комплекса контроля и мониторинга микроклимата теплицы	53
3.1	Диаграмма компонентов серверного приложения	53
3.2	Представление развертывания серверного приложения	55
3.3	Представление развертывания приложения контроллера.....	58
4	Контроль качества программного комплекса	60
4.1	Модульное тестирование	60
4.2	Интеграционное тестирование	61
4.3	Функциональное тестирование.....	62
4.4	Метрики кода.....	67
4.5	Планирование и оценка бюджета	70
	Заключение	73
	Список использованных источников	74
	Приложение А. Глоссарий.....	76
	Приложение Б. Описание атрибутов сущностей БД.....	79
	Приложение В. Исходный код модульного тестирования.....	84
	Приложение Г. Исходный код серверного приложения	88
	Приложение Д. Исходный код приложения контроллера	92

Введение

В настоящее время все чаще употребляются термины «умный дом» и «интернет вещей», как одно из направлений в развитии современных технологий. Интернет вещей – физические объекты, в которых встроены датчики, программное обеспечение и другие технологии с целью подключения и обмена данными по сети «Интернет» [1].

Одна из перспективных областей применения технологии интернет вещей – контроль и мониторинг микроклимата парников, теплиц, тепличных комбинатов, селекционных комплексов и т.д. Это позволяет автоматизировать процессы контроля влажности, температуры окружающей среды, влажности почвы, уровнем света. Также это позволяет проводить сбор данных от различных датчиков с целью проведения последующего анализа.

Актуальной проблемой при проектировании и реализации таких систем является выбор алгоритма срабатывания различных событий. К таким событиям можно отнести: включение и отключения подачи воды, изменения уровня света, открытие и закрытие форточек, и т.д. При неоптимальном формировании правил может нарушиться экосистема, что приведет к уничтожению выращиваемых культур.

Целью выпускной квалификационной работы является разработка программно-аппаратного комплекса контроля и мониторинга микроклимата теплицы.

Разработка включает в себя анализ требований, этапы проектирования, реализации и тестирования программных и аппаратных средств.

1 Анализ предметной области и требований комплекса мониторинга и контроля микроклимата теплицы

1.1 Анализ программно-аппаратного комплекса

На сегодняшний день системы автоматизированного контроля и мониторинга микроклимата теплиц все чаще применяются как в промышленных, так и в частных масштабах. Такая система позволяет перенести рутинную работу по поливу тепличных культур, по регулированию влажности окружающей среды путем открытия и закрытия форточек и т.д. Также эти системы позволяют просматривать показания с датчиков в режиме реального времени или за определенный временной интервал, что позволяет своевременно принимать решения о состоянии микроклимата в теплице или проводить анализ с целью выявления закономерностей.

Для регулирования и мониторинга климата используются несколько аналого-цифровых датчиков: датчик температуры и влажности окружающей среды «АМ2302» и датчик влажности почвы «TZT Soil moisture sensor»

Показателей с данных датчиков хватит для того, чтобы сделать выводы о состоянии микроклимата и необходимы для формирования правил (триггеров). Для подачи воды в теплицу используется дренажный насос, погруженный в бак. Он управляется с помощью одноканального десяти амперного релейного модуля.

1.1.1 Описание аппаратных компонентов комплекса

1.1.1.1 Описание датчика температуры и влажности

АМ2302 (DHT22) – это комбинированный датчик температуры и влажности (ДТВ), который выдает откалиброванный цифровой сигнал на выходе, представленный на рисунке 1. Датчик включает в себя влагочувствительный компонент, используемый для измерения влажности. Представляет собой два электрода с влагоудерживающей подложкой, зажатой между ними.



Рисунок 1 – Датчик влажности почвы

По мере поглощения влаги подложка высвобождает ионы, которые увеличивают электропроводимость между электродами. Изменение сопротивления между двумя электродами пропорционально относительной влажности [2].

Для измерения температуры используется Negative Temperature Coefficient (NTC) термистор, что означает, что с ростом температуры сопротивление уменьшается. Термистор – это резистор, меняющее свое сопротивление в зависимости от температуры.

Также имеется небольшая печатная плата с восьмизрядной микросхемой в корпусе SOIC-14. Эта микросхема со встроенными аналого-цифровым преобразователем (АЦП), измеряет и обрабатывает аналоговый сигнал, а на выходе выдает цифровой сигнал с данными о температуре и влажности.

Данный датчик легко подключается. У него есть четыре выхода. Схема подключения представлена на рисунке 2.

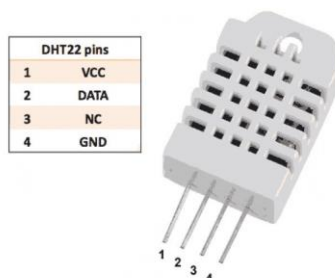


Рисунок 2 – Схема подключения

Ниже представлены обозначения схемы:

- VCC обеспечивает питание датчика, допустимое напряжение от 3.3 до 5 В;
- DATA используется для передачи данных в микроконтроллер;
- NC не используется;
- GND должен быть подключен к земле.

1.1.1.2 Описание датчика влажности почвы

Датчик влажности почвы (ДВП) включает в себя зонд с двумя открытыми проводниками и электронный модуль, который представлен на рисунке 3.



Рисунок 3 – Датчик влажности почвы

Вилка в форме зонда действует как переменный резистор, меняющее свое сопротивление в зависимости от содержания воды в грунте. Это сопротивление обратно пропорционально влажности почвы. Датчик генерирует напряжение в соответствии с сопротивлением, которое можно измерить для определения уровня влажности.

В зависимости от сопротивления датчика модуль выдает выходное напряжение, доступное на выводе аналогового выхода (АО). Этот же сигнал

поступает на высокоточный компаратор LM393 для его оцифровки, с выхода которого сигнал поступает на вывод цифрового выхода (DO).

Модуль содержит встроенный потенциометр для регулировки чувствительности цифрового выхода. С помощью этого потенциометра можно установить пороговое значение; таким образом, когда уровень влажности превышает пороговое значение, модуль выдаст низкий логический уровень, в других случаях высокий логический уровень будет применен к цифровому выходу. Схема подключения представлена на рисунке 4.



Рисунок 4 – Схема подключения ДВП

Ниже представлены обозначения схемы:

- VCC обеспечивает питание датчика, допустимое напряжение от 3.3 до 5 В;
- GND должен быть подключен к земле;
- AO – аналоговый выход;
- DO – цифровой выход;
- вход на зонд – гнездо подключения потенциометра.

Однако этот модуль не имеет встроенного АЦП поэтому к ДВП дополнительно подключается модуль PCF8591.

1.1.1.3 Описание аналого-цифрового преобразователя

Модуль содержит аналоговые датчики, данные с которых выдаются в цифровом виде через АЦП, однако каналы АЦП могут быть отключены от датчиков и подключены к внешним входам.

Модуль имеет двунаправленную шину передачи данных (I²C). Передача данных осуществляется по двум проводам – Serial Data (SDA) и SCL Serial Clock (SCL). Одна такая двухпроводная линия связи может содержать до 127 различных устройств и модулей, которые могут работать с шиной I²C [3].

Шина I²C является синхронной и состоит из двух линий: данных SDA и тактов SCL. Есть ведущий (master) и ведомые (slave). Инициатором обмена всегда является ведущий, обмен между двумя ведомыми невозможен. На одной двухпроводной шине может быть до 127 устройств.

Такты на линии SCL генерирует ведущий линией. Линия SDA может управляться как ведущим, так и ведомым устройством, в зависимости от направления передачи. Единицей обмена информации является пакет, созданный на основе уникальных условий на шин, называемых условиями запуска и остановки. Ведущее устройство в начале каждого пакета передает один байт, который указывает адрес ведомого устройства и направление передачи последующих данных. Данные передаются восьмибитными словами. После каждого слова принимающей стороной передается один бит подтверждения. Схема подключения представлена на рисунке 5.

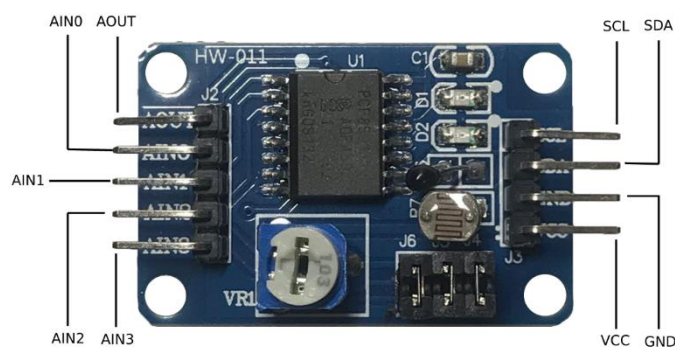


Рисунок 5 – Схема подключения АЦП

Ниже представлены обозначения схемы:

- VCC обеспечивает питание датчика, допустимое напряжение от 3.3 до 5 В;
- GND должен быть подключен к земле;
- SCA – линия данных;
- SCL – линия тактов;
- AOUT – выход цифро-аналогового преобразователя (ЦАП);
- AIN0/AIN3 – входы АЦП.

1.1.1.4 Описание датчика влажности почвы

Следующий аппаратный компонент в системе – одноканальный релейный модуль. Соединение электрических цепей с разными параметрами возможно при использовании шлюза, функцию которого выполняет реле. Принцип устройства – запускать и останавливать механизмы путем замыкания электрической цепи.

Если плата подает сигнал, то внутренний механизм коммутационного устройства выполняет саму работу. Одноканальное реле можно сравнить с пультом дистанционного управления, выполняющим поставленные задачи при слабых сигналах.



Рисунок 6 – Схема подключения реле

Схема подключения представлена на рисунке 6.

Ниже описаны входы и выходы для реле:

- VCC – питание 3.3 В;
- GND должен быть подключен к земле;
- IN – линия входа.

1.1.1.5 Описание контроллера «Raspberry Pi 3 Model B»

Контроллер представляет собой одноплатный компьютер под управлением UNIX подобной операционной системы (ОС) «Raspberry Pi OS» [4]. По программным возможностям он ничем не уступает стационарным компьютерам, а даже превосходит некоторых из них.

Подключение внешней памяти организовано с помощью флеш-накопителя «Micro SD» с максимальным объемом 32 Гб. Для питания контроллера используется 2.1 амперный адаптер с подключением по Micro USB порту. Технические характеристики представлены в таблице 1.

Таблица 1 – Технические характеристики контроллера

Название	Описание характеристики
Процессор	64-битный 4-ядерный процессор Broadcom BCM2837 с тактовой частотой до 1.2 ГГц.
Оперативная память	1ГБ LPDDR2 SDRAM с тактовой частотой 900 МГц
WiFi и Bluetooth адаптер	BCM43438 WIFI 802.11n + Bluetooth 4.1 Low Energy (BLE);
Сетевой интерфейс	100 Мб Ethernet
USB порты	4 USB порта 2.0
Аудио	Стерео 3.5 мм jack
Видео	Графический 2-ядерный сопроцессор Video Core IV® Multimedia, HDMI Full HD выход

Для подключения дополнительных периферийных устройств используется интерфейс ввода-вывода – GPIO, означающая общий интерфейс ввода/вывода. Он содержит цифровые входы и выходы для подключения датчиков, различных контроллеров, реле, дисплеев и прочей периферии. Разъемов питания 4. Другие контакты могут действовать как вход или выход. Кроме того, некоторые из них многофункциональны и могут работать как интерфейс UART, I2C, SPI или ШИМ.

Число пинов на колодке GPIO Raspberry Pi 3 равняется 40. Они пронумерованы и делятся на три группы:

- питающие;
- заземляющие (GND);
- порты (BCM).

Одноплатный компьютер представлен на рисунке 7.



Рисунок 7 – Raspberry Pi 3 model B

Первые необходимы для подачи электричества разных напряжений – 3.3 и 5 В. Вторые обеспечивают безопасность работы платы, отводя электричество. Третьи выступают в роли интерфейсов, получая и отправляя данные. Именно к ним подключаются модули и устройства.

1.2 Анализ программных аналогов

Рассмотрим аналогичные системы, решающие проблемы, поставленные в анализе предметной области. Одной из таких решений является система автоматизации «Терраформ» от Пензенской компании «Бионика» [5]. Они поставляют контролер системы автоматизации климата вместе с пятью различными датчиками и встроенным выходным реле. Контроллер представлен на рисунке 8.



Рисунок 8 – Контроллер «Терраформ»

Также они поставляют мобильное приложение, в котором имеется возможность просматривать показания с датчиков, выставлять предельные значения датчиков, включать и отключать реле питания, регулировать время работы автоматического режима. Технические характеристики представлены в таблице 2.

Таблица 2 – Технические характеристики контроллера «Терраформ»

Название	Описание характеристики
Вес	0.5 кг
Габариты	22 x 8 x 8 см
Напряжение питания контроллера	220 В
Число реле, управляющих устройствами	8
Мощность нагрузки реле	1 кВт, больше через контактор
Число подключаемых датчиков	
Температуры-влажности воздуха	4
Температуры почвы/жидкости	4
Влажности почвы	4
Освещенности	2
Концентрации CO ₂	1
Уровня pH	1

Стоимость одного комплекта составляет 18 000 рублей. Дополнительные датчики покупаются отдельно и стоят от 700 рублей.

Преимущества данной системы:

- быстрая установка и интеграция в теплицу;
- собственное мобильное приложение;
- 5 датчиков в комплекте;
- автоматический режим управления реле.

Недостатки:

- малое количество датчиков;
- отсутствие возможности модифицировать контроллер;
- непрозрачный механизм работы автоматического режима;
- высокая стоимость оборудования.

Следующий продукт, представленный на рынке, это автоматическая теплица от компании «2agrocloud» [6].

На сайте есть возможность выбрать как готовые комплекты, так и собрать собственную конфигурацию оборудования. Пример готовой конфигурации представлен на рисунке 9.

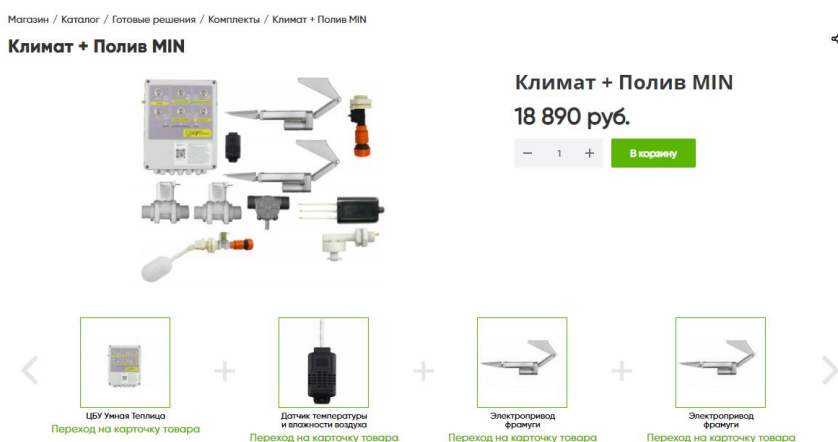


Рисунок 9 – Готовая конфигурация оборудования

Данный готовый комплект содержит:

- ЦБУ со встроенной функцией WiFi доступа – 1 шт;
- датчик температуры и влажности воздуха – 1 шт;
- электропривод фрамуги/форточки – 2 шт;
- клапан подачи воды – 1 шт;
- клапан полива – 1 шт;
- расходомер воды – 1 шт;
- датчик уровня воды – 1 шт;
- датчик температуры и влажности почвы – 1 шт;
- гидроузел подачи воды – 1 шт;
- гидроузел отбора воды – 1 шт.

Их автоматизированная теплица позволяет управлять процессами через приложение на смартфоне или с компьютера, находясь в любой точке мира. Имеется удобная пошаговая инструкция по подключения и монтажу системы.

Преимущества данной системы:

- конструктор выбора оборудования;
- наличие мобильного приложения и web-сайта;
- возможность автоматического полива;

- наличие пошаговой удобной инструкцией;
- возможность расширить функционал.

Недостатки данной системы:

- автоматический режим требует платной подписки от 100 рублей в месяц;
- высокая стоимость оборудования;
- беспроводная передача данных менее надежна, чем проводная;
- фирменные датчики не взаимозаменяемые.

Приведем сравнительную таблицу рассмотренных аналогов, которая представлена в таблице 3.

Таблица 3 – Сравнение аналогов

Характеристика	Бионика	2agrocloud
Наличие мобильного приложения	+	+
Наличие веб-сайта	-	+
Настраиваемый автоматический режим	-	+
Наличие средств анализа показателей	-	-
Возможность расширения функционала	-	+
Учет контроля времени	+	+
Стоимость оборудования	18000 р.	19000 р.

Разрабатываемый комплекс будет учитывать преимущества и недостатки аналогичных систем. Благодаря использованию одноплатного контроллера «Raspberry Pi 3» мы получим выигрыш в вычислительных мощностях, что благоприятно кажется на расширении функционала. Сюда можно включить охранную систему с использованием видеокамер и датчиков движения, управление не только микроклиматом теплицы, но и умным домом. Все эти компоненты будут располагаться в единой экосистеме, что упрощает контроль и использование данного комплекса.

1.3 Постановка задачи

Данная выпускная квалификационная работа посвящена разработке программного и аппаратного комплекса контроля микроклимата теплицы.

Основная задача комплекса – поддержание оптимальных климатических условий путем автоматизации процесса полива.

Комплекс состоит из двух компонентов – аппаратной и программной. Аппаратный компонент включает в себя микроконтроллер, датчики температуры, влажности почвы и окружающей среды, а также реле питания. Программный компонент отвечает за взаимосвязь аппаратных компонентов, а также за пользовательское взаимодействие.

Для надежного хранения и обработки информации необходимо разработать базу данных (БД), содержащую информацию о пользователях, устройствах управления (УУ), привязанных к ним датчикам и реле питания, их показатели, а также команды, подаваемые на микроконтроллер. Разрабатываемый комплекс должен обладать следующим функционалом:

- авторизация и аутентификация пользователей;
- считывание показателей датчиков;
- отправка и прием команд в УУ;
- автоматизированное срабатывание реле питания;
- отображение показателей датчиков в режиме реального времени;
- отображение графиков показателей датчиков.

В процессе проектирования и разработки необходимо разработать архитектуру взаимодействия компонентов комплекса, спроектировать схему подключения аппаратных компонентов.

Для более наглядного и полного представления сформулированных функциональных требований к реализуемой системе была построена диаграмма вариантов использования в нотации UML [7]. Данный вид диаграмм отражает возможности взаимодействия пользователей (актеров) с системой.

Диаграмма вариантов использования представлена на рисунке 10.

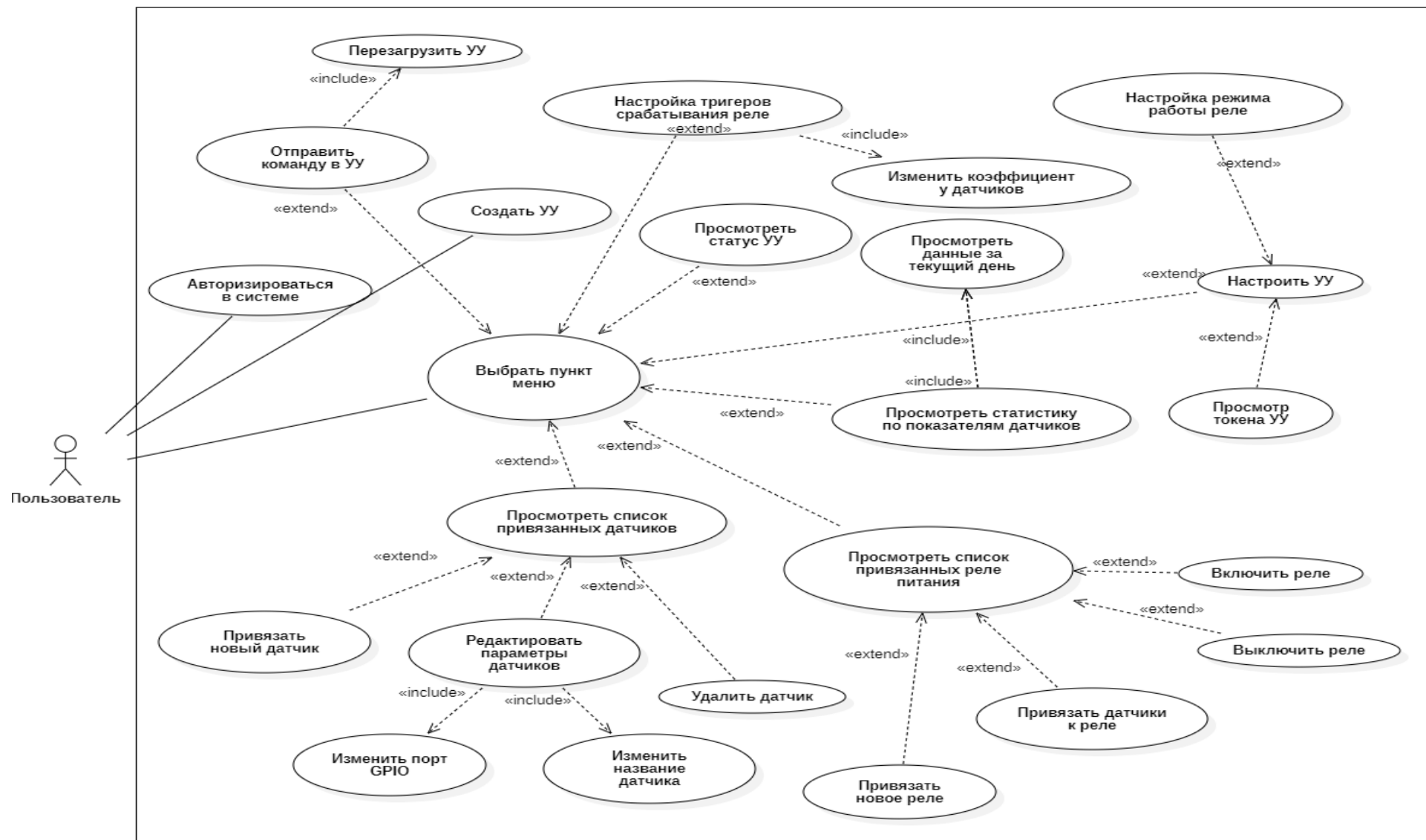


Рисунок 10 – Диаграмма вариантов использования

Актером данной диаграммы выступает пользователь системы. Он имеет возможность просматривать список УУ, анализировать показания с датчиков и управлять реле питания, а также настраивать параметры работы системы принятия решений. Далее описаны основные прецеденты диаграммы вариантов использования.

1. Авторизоваться в системе. Ввод логина и пароля. При верно введенных учетных данных осуществляется вход в систему.
2. Выбрать УУ. Выбрать привязанное УУ для дальнейшей работы с ним.
3. Просмотреть список привязанных датчиков. Просмотр показателей датчиков в режиме реального времени.
4. Просмотреть список привязанных реле. Просмотр списка привязанных к УУ реле.
5. Привязать датчики к реле. В привязанных реле явно указать, какие датчики будут использоваться для расчета коэффициента системы принятия решений.
6. Просмотреть статистику по показателям датчиков. Анализ временных и качественных показателей в виде графиков.
7. Настройка триггеров срабатывания реле. Подбор коэффициентов для настройки системы принятия решений.
8. Отправить команду в УУ. Отправка команды в УУ через веб-интерфейс.

1.4 Функциональное моделирование

Для проектируемой комплекса были выделены бизнес-процессы, которые представлены в виде диаграмм при помощи CASE-средства CA ERwin Process Modeler, поддерживающего такие методологии моделирования бизнес-процессов, как IDEF0, IDEF3 и DFD. CA ERwin Process Modeler является средством визуального моделирования. Диаграммы IDEF0 позволяют понять, какие объекты или информация служат сырьем для

процессов, какие результаты приводят к работе, каковы управляющие факторы и какие ресурсы для этого необходимы. Процесс моделирования системы в IDEF0 начинается с определения контекста, то есть самого абстрактного уровня описания системы в целом [8]. Контекстная диаграмма бизнес-процессов проектируемой системы в нотации IDEF0 представлена на рисунке 11.

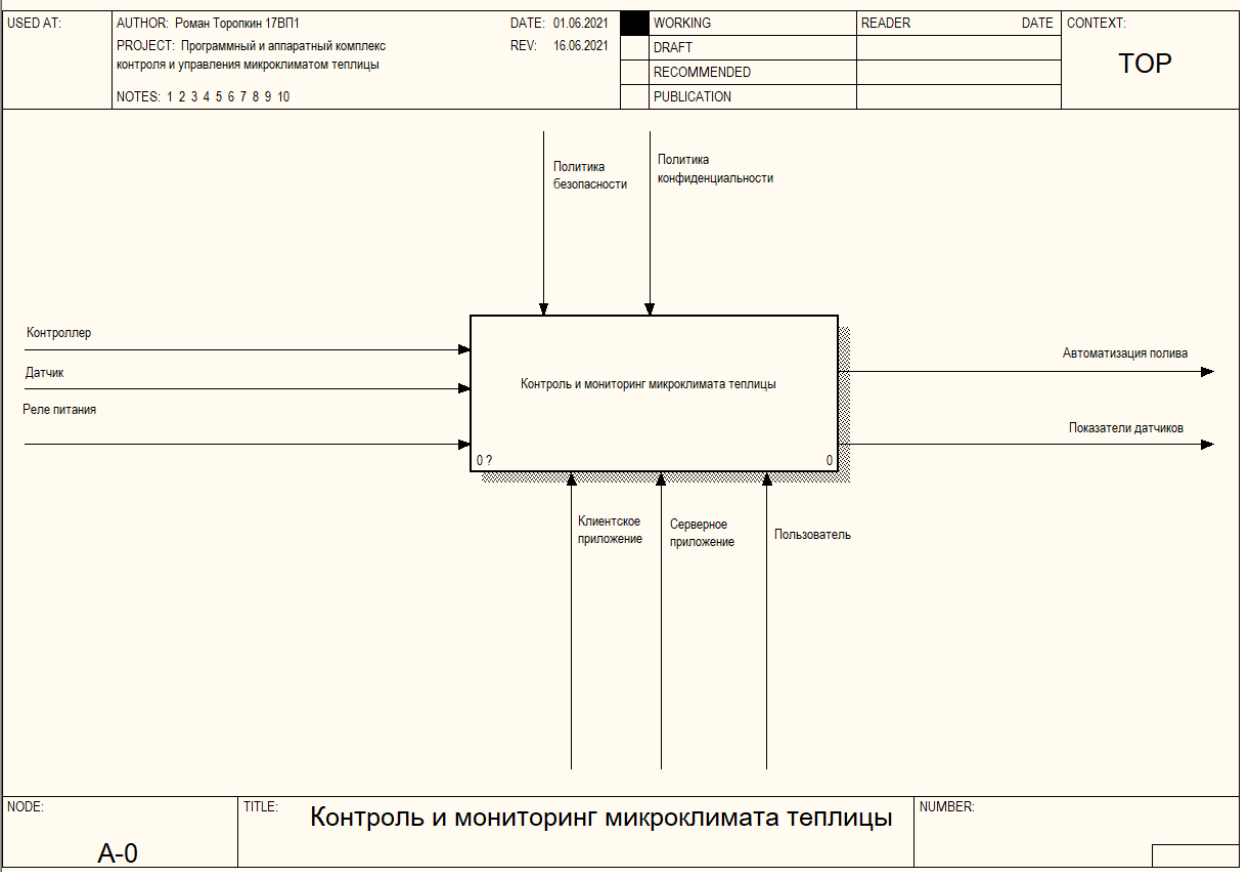


Рисунок 11 – Контекстная диаграмма в нотации IDEF0

На контекстной диаграмме отражается общее описание основного бизнес-процесса (блока) системы – контроля и мониторинга микроклиматом. После описания системы проводится ее разбиение на фрагменты. Этот процесс называется функциональной декомпозицией [9]. Основной бизнес-процесс был декомпозирован на следующие составляющие бизнес-процессы: синхронизировать контроллер, привязать датчики к УУ, привязать реле к УУ

и настроить алгоритм автоматизированного управления поливом. Диаграмма декомпозиции представлена на рисунке 12.

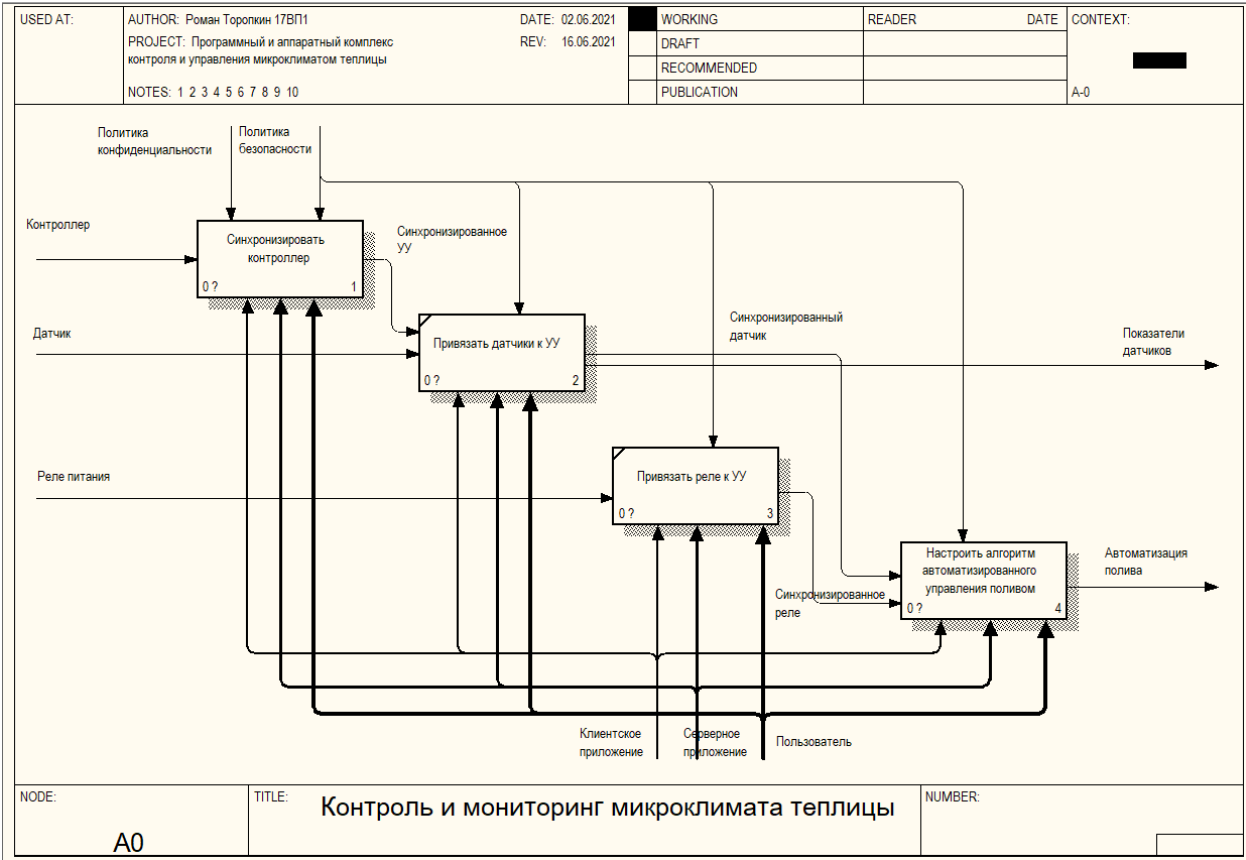


Рисунок 12 – Диаграмма декомпозиции комплекса контроля и микроклимата теплицы в нотации IDEF0

Пользователь синхронизирует контроллер с системой для дальнейшей работы с УУ. Далее необходимо привязать датчики и реле питания к УУ, чтобы в дальнейшем отображались показатели датчиков. После того, как первоначальная синхронизация аппаратных компонентов успешно завершится, необходимо настроить алгоритм автоматизированного управления поливом для автоматизированного контроля микроклимата теплицы.

Бизнес-процесс «Синхронизировать контроллер» декомпозирован в нотации DFD. Диаграммы потоков данных позволяют описать документооборот и обработку данных. В нотации DFD хранилище позволяет

описать и сохранить в памяти данные. Внешние сущности представляют собой источник данных. Стрелки описывают движение объектов из одной части системы в другую [10]. Декомпозиция бизнес-процесса «Синхронизировать контроллер» представлена на рисунке 13.

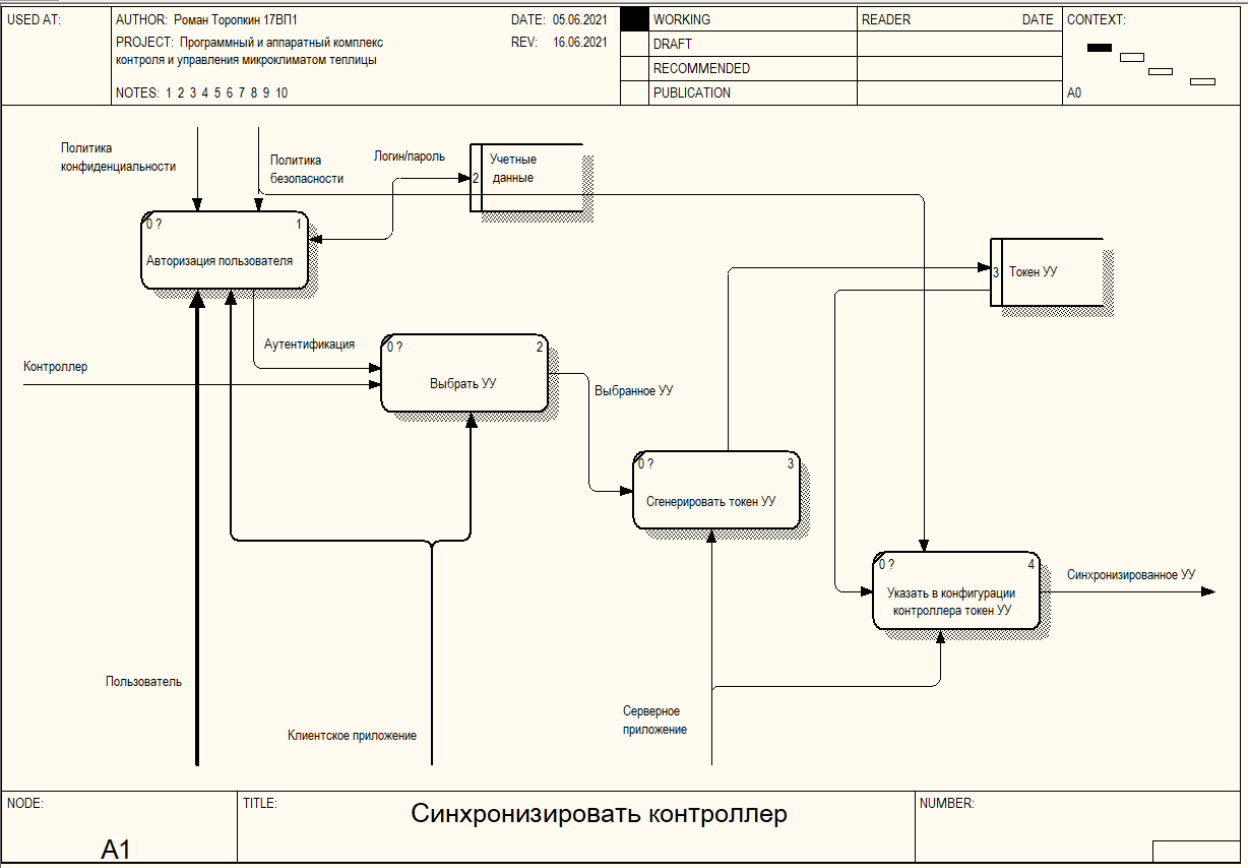


Рисунок 13 – Диаграмма декомпозиции «Синхронизировать контроллер» в нотации DFD

Для синхронизации и работы с УУ пользователю необходимо авторизоваться на web-сайте, введя свои учетные данные (логин и пароль). Происходит переход на страницу со списком доступных УУ. Так как первоначально у пользователя нет синхронизированных УУ, ему необходимо их создать. После создания сервер автоматически генерирует токен доступа управления контроллером, который действует один год. Этот токен необходимо прописать в конфигурационный файл программы по мониторингу и управления для того, чтобы синхронизировать сервер с УУ.

По такому же принципу был декомпозирован бизнес-процесс «Настроить алгоритм автоматизированного управления поливом», диаграмма которого представлена на рисунке 14.



Рисунок 14 – Диаграмма декомпозиции «Настроить алгоритм автоматизированного управления поливом» в нотации DFD

Пользователю необходимо в web-сайте вручную указать, какие датчики будут связаны с реле. Делается это для того, чтобы была возможность устанавливать несколько точек поливов, используя при этом разные показатели датчиков.

Таким образом, можно сказать, что использованное CASE-средство идеально подходит для анализа бизнес-процессов.

2 Проектирование программных средств контроля и мониторинга микроклиматом теплицы

2.1 Базовая архитектура программно-аппаратного комплекса

Разрабатываемый комплекс мониторинга и контроля микроклимата теплицы обладает многоуровневой архитектурой. Архитектурная схема представлена на рисунке 15.

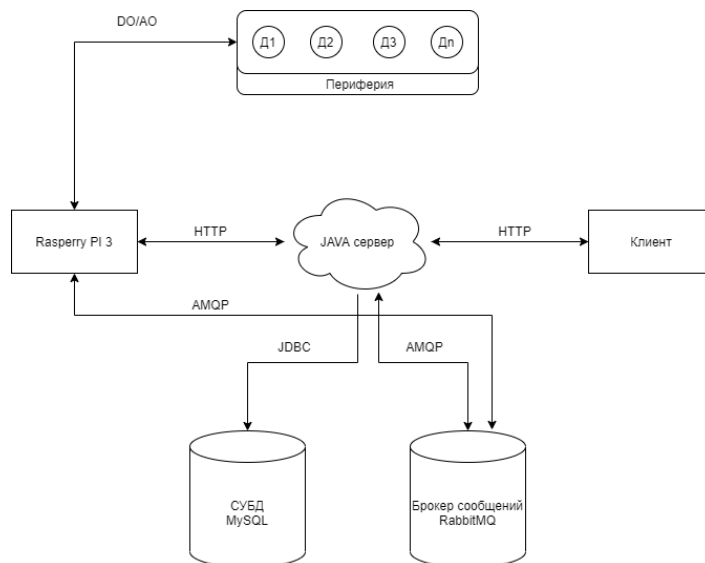


Рисунок 15 – Базовая архитектура программно-аппаратного комплекса

Рассмотрим подробнее каждый компонент системы.

1. Java сервер. Централизованный сервер обрабатывает запросы, поступающих от пользователей (web-клиент) и контроллеров (Raspberry Pi 3). Сетевое взаимодействие между этими компонентами происходит по протоколу HTTP при помощи программного интерфейса приложения (API).

2. Клиентом в данной архитектуре является web-сайт, расположенный на хостинге. Именно на стороне клиента пользователь может взаимодействовать с контроллером с помощью web-обозревателя.

3. Raspberry Pi 3. Контроллер, управляющий периферией по GPIO. На нем в фоновом режиме работает сервис, который обрабатывает показатели с датчиков, управляет реле питания, обрабатывает входящие команды со

стороны сервера, а также отправляет в режиме реально времени текущие показатели.

4. СУБД MySQL. Реляционная база данных MySQL хранит и обрабатывает всю информацию, необходимая для работы программного комплекса. Взаимодействует двухсторонней связью между сервером по протоколу Java DataBase Connectivity (JDBC).

5. Брокер сообщений RabbitMQ. Служит для разгрузки поступающих запросов на сервер и берет на себя передачу команд от сервера в контроллер, а также принимает показатели датчиков от контроллера. Работает по протоколу Advanced Message Queuing Protocol (AMQP).

6. Периферия. Датчики и реле питания, получающие и передающие данные контроллеру по аналого-цифровым выходам.

Таким образом, разработанная архитектура позволяет эффективно разделять функционал программно-аппаратного комплекса. Клиент и контроллер взаимодействуют независимо друг от друга. Разграничение связей между компонентами также обеспечивают высокий уровень безопасности системы. К тому же такая архитектура обладает свойством масштабирования. Достаточно поменять аппаратную конфигурацию сервера для большей пропускной способности системы.

2.2 Описание алгоритма автоматизированного управления поливом

Для обеспечения процесса автоматизации полива было принято решение разработать алгоритм автоматизированного управления поливом. Пошаговое описание алгоритма.

1. Получаем список реле $R_1, R_2, \dots, R_n$.
2. Каждому элементу списка реле $R_i, i = \overline{1, n}$ ставится в соответствие список привязанных датчиков $S^i = \{S_1^i, S_2^i, \dots, S_{l_i}^i\}$ где $S^i < S, S$ – множество всех датчиков.

3. У привязанного датчика необходимо получить весовой коэффициент k_j^i и последний считанный показатель $d_j^i, j = \overline{1, l_i}$.

4. Высчитывается коэффициент срабатывания триггера C_i по формуле:

$$C_i = \sum_{j=1}^{l_i} \frac{d_j^i k_j^i}{100}.$$

5. Высчитывается пороговое значение P_i по формуле:

$$P_i = n_i Y,$$

где n – количество привязанных датчиков;

Y – коэффициент чувствительности триггера.

6. Сравниваются полученные значения P_i и C_i . В случае если $C_i < P_i$, алгоритм подает команду УУ на включение реле питания на 10 секунд.

7. Вернуться к пункту 1.

Алгоритм работает в фоновом режиме с интервалом в 5 секунд. Однако, имеется возможность в любой момент выключить его и управлять реле питания вручную.

2.3 Схема подключения и развертывания аппаратных компонентов

Ниже на рисунке 16 представлена схема подключения и развертывания аппаратных компонентов.

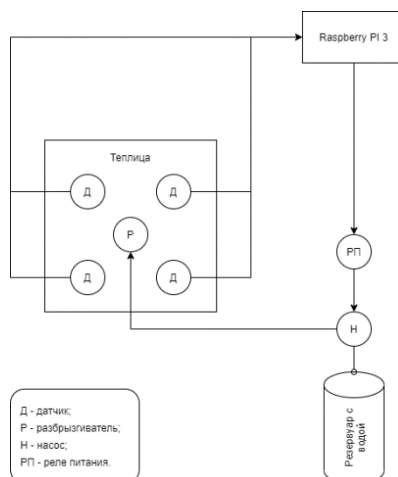


Рисунок 16 – Схема подключения и развертывания аппаратных компонентов

Подключение комплекса начинается с расположения датчиков. Каждый датчик необходимо предварительно обработать влагозащитным лаком с целью повышения срока службы. ДВП рекомендуется устанавливать на равноудаленных участках почвы или на отдельных грядках. ДТВ располагается на самой высокой точке, закрепленный на металлическом профиле рамы. Подключаемый кабель проводится под землей, защищенный гофрированным материалом. Таким образом обеспечивается стабильная и бесперебойная передача данных.

Следующий этап – размещение разбрызгивателя в теплице с последующим подключением к погружному насосу. В месте стыка теплицы шланг прокладывается под землей, а на остальных участках трассы по поверхности грунта. Насос подключается к сети через удлинитель, на конце которого присоединяется реле питания. Это делается для безопасности и упрощения обслуживания релейного компонента. Он же, в свою очередь, подключается к контроллеру аналогичным образом, что и датчики.

Контроллер Raspberry Pi 3 размещается в защитной пластиковом корпусе, окруженный кулерами на вдув и выдув, обшитые сеткой, с целью предотвращения перегрева и попадания пыли. Питание подключается через источник бесперебойного питания с выходным напряжением 12 вольт и максимальным временем работы до 12 часов. Этого достаточно для того, чтобы вовремя приостановить работу фоновой программы и исключить непредсказуемых ситуаций.

При такой схеме, процесс развертывания получается простым и быстрым и не требует больших трудозатрат.

2.4 Средства разработки программного комплекса

Для написания программы под контроллер был выбран язык программирования Python версии 3.7. Выбор данного языка обоснован мощными встроенными и интегрируемыми библиотеками для работы с внешней подключаемой периферией, возможностью использования

парадигмы объектно-ориентированного программирования (ООП), а также скоростью написания кода.

Серверная часть реализуется с помощью фреймворка «Spring» для Java платформы [10]. Фреймворк – это программная платформа, определяющая структуру программной системы. В отличие от использования сторонних библиотек, фреймворк диктует правила построения архитектуры ПО – «каркас», который можно дополнять или изменять, согласно установленным требованиям разработчиков. Основным преимуществом использования платформы «Spring» – встроенный механизм инверсии управления (IoC), благодаря которому написанные модули становятся слабосвязанными друг с другом. Данный фреймворк обладает следующими модулями:

- модуль работы с различными СУБД, использующий JDBC и ORM-средства;
- конфигурационный модуль авторизации и аутентификаций, позволяющий описывать правила доступа к ресурсам;
- модуль, реализующий паттерн программирования MVC;
- модуль написания интеграционных и модульных тестов.

Также использования языка Java обеспечивает высокий уровень безопасности. Это достигается благодаря проверке кода на этапе компиляции, строгой типизации объектов, использовании API безопасности, отсутствии указателей, встроенных механизмов управления памяти, а также загрузчику классов.

Клиентская часть реализуется с помощью наборов web-технологий. В качестве языка разметки используется HTML5, с использованием формального языка описания внешнего вида – CSS, а в качестве языка программирования используется JavaScript. Для упрощения размещения графических компонентов, используется фреймворк Bootstrap версии 5 [11].

Базой данных в программном комплексе является СУБД MySQL версии 8.0 от компании Oracle. MySQL использует реляционную модель данных. Гибкость данного СУБД заключается в использовании разных типов таблиц:

например, таблицы MySQL поддерживают полнотекстовый поиск, а таблицы InnoDB поддерживают транзакции на уровне отдельных записей.

2.5 Проектирование структуры данных программного комплекса

Проектирование структуры данных осуществлялось с помощью Case средства ERwin Data Modeler 7.3 и его методологии IDEF1X. Простота интерфейса, кроссплатформенность и особенность организации физической памяти различных серверных СУБД – все это делает ERwin является наиболее популярным и доступным средством моделирования данных.

Для создания структуры данных в стандарте IDEF1X были рассмотрены два уровня представления модели данных – логический и физический.

2.5.1 Разработка концептуальной модели базы данных

Построение концептуальной моделирование включает в себя описание типов сущностей и описание типов связей между типами сущностей.

В таблице 4 представлено описание типов сущностей.

Таблица 4 – Описание типов сущностей

№	Наименование	Описание
1	Пользователь	Пользователь аппаратно-программного комплекса
2	Роль	Роль пользователя в системе
3	Устройство управления	Программная сущность контроллера
4	Токен устройства	Идентификационный номер УУ, обеспечивающий доступ к управлению УУ
5	Датчик	Аппаратный компонент, считывающий показания окружающего мира
6	Реле питания	Аппаратный компонент, являющийся управляемым ключом замыкания сети
7	Команда	Команда, посылаемая на выполнение УУ
8	Показатель датчика	Числовое значение показания датчика
9	Тип датчика	Тип функциональности датчика
10	Категория команды	Тип посылаемой команды

В таблице 5 представлено описание связей между сущностями

Таблица 5 – Описания связей между сущностями

№	Тип сущности 1	Тип сущности 2	Описание	Мощность	Null-значение
1	Пользователь	Роль	Пользователь может обладать множеством комбинаций ролей	M:M	Нет
2	Датчик	УУ	УУ может иметь несколько датчиков	1:M	Нет
3	Датчик	Тип датчика	Датчики могут быть разных типов	1:M	Нет
4	Реле	УУ	УУ может иметь несколько реле	1:M	Нет
5	Реле	Датчик	У разных реле могут быть разные датчики	M:M	Нет
6	Показатель датчика	Датчик	У датчика может быть несколько показателей	1:M	Нет
7	Команда	УУ	УУ может посылать несколько команд	1:M	Нет
8	Команда	Категория команды	Команда может быть разной категории	1:M	Нет
9	Токен устройства	УУ	УУ может иметь несколько токенов, но только 1 активный	1:M	Нет
10	Токен устройства	Пользователь	Пользователь может иметь несколько токенов	1:M	Нет

Также, необходимо описать каждый атрибут сущностей. Таблицы описания атрибутов представлены в приложении Б таблицы Б.1 – Б.10. Размерность атрибутов задается автоматически фреймворком Spring.

Концептуальная модель представлена на рисунке 17.

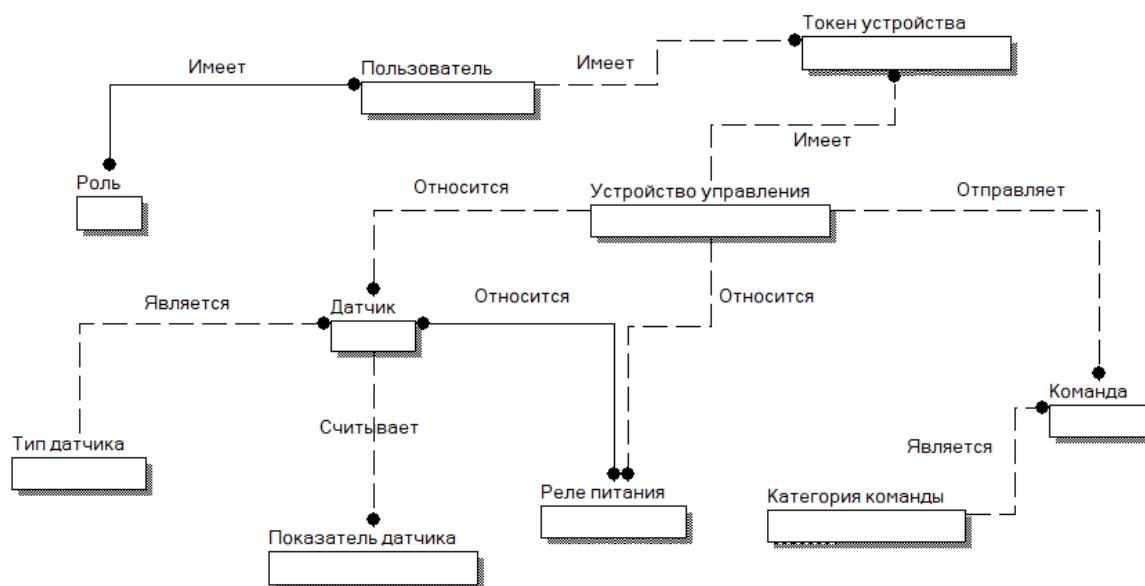


Рисунок 17 – Концептуальная модель БД

Концептуальная модель базы данных – это диаграмма, составленная в принятых обозначениях и подробно демонстрирующая связь между объектами и их характеристиками. Создается концептуальная модель для дальнейшего проектирования БД и ее перевода, например, в реляционную БД. На концептуальной модели в визуально удобной форме записываются связи между объектами данных и их характеристиками.

2.5.2 Логическое моделирование базы данных

Логическая модель – это модель базы данных, выраженная в терминах модели данных. Логическая модель расширяет концептуальную модель, определяя их атрибуты, описания и ограничения для сущностей, проясняет состав сущностей и отношения между ними [12].

Данная модель позволяет получить графическое представление логической структуры предметной области. Модель логической предметной области иллюстрирует сущности, а также их отношения друг с другом. Кроме того, конструкция логической модели не должна зависеть от требований платформы и языка реализации или от способа дальнейшего использования данных.

Сущность – это шаблон, из которого создаются определенные экземпляры сущности. Свойства объектов и субъектов реального мира описываются с помощью атрибутов. Атрибут представляет собой тип характеристик или свойств, связанных с множеством реальных или абстрактных объектов. Экземпляр атрибута определяется типом характеристики и ее значением, называемым значением атрибута. Отношения между сущностями иллюстрируются с помощью отношений. Правила и ограничения отношений описываются с помощью свойств. Обычно отношения определяют либо зависимости между сущностями, либо влияние одной сущности на другую.

На основании концептуальной модели была сконструирована логическая модель базы данных, представленная на рисунке 18.

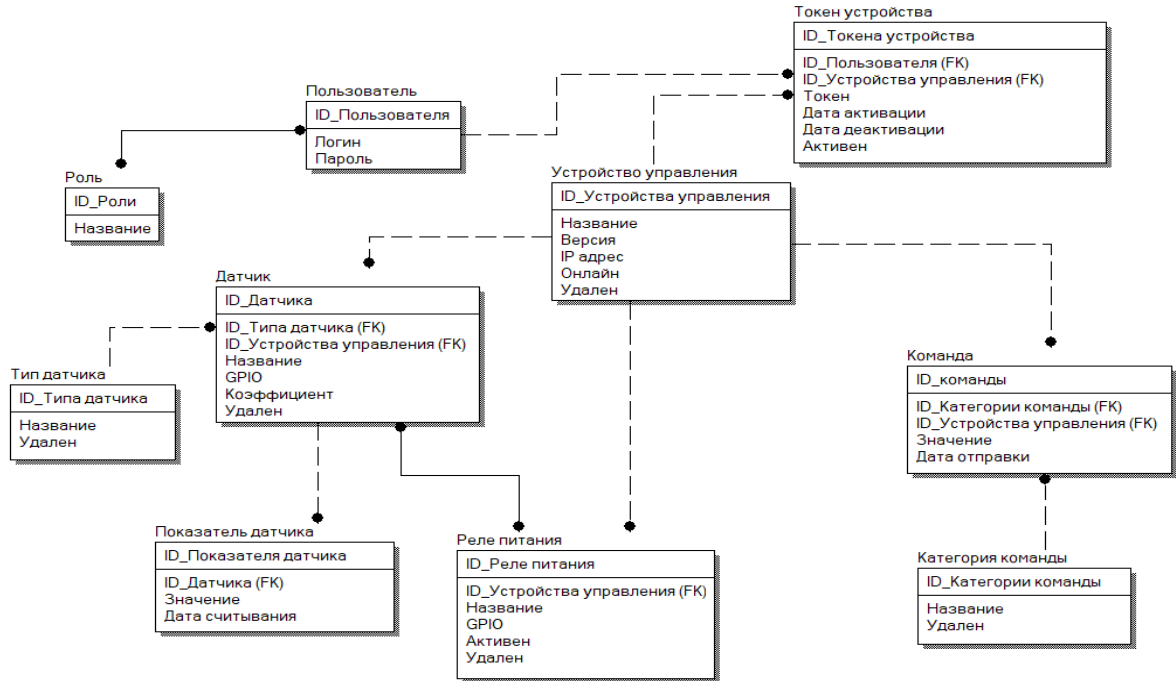


Рисунок 18 – Логическая модель БД

Построенная логическая модель полностью отражает описанные в таблицах Б.1 – Б.10.

2.5.3 Физическое моделирование базы данных

Физическая модель – логическая модель базы данных, выраженная в терминах языка описания конкретной СУБД. Физические модели баз данных определяют, как данные хранятся в среде хранения и как к ним осуществляется доступ, что поддерживается на физическом уровне. Способы физической организации данных в различных СУБД обычно различны и определяются типом используемой ЭВМ, средствами разработки СУБД, а также критериями, которыми руководствуются разработчики СУБД при выборе методов размещения данных и способов доступа к ним.

MySQL – это решение для приложений малого и среднего бизнеса. Он включен в серверы WAMP, AppServ, LAMP и в портативные серверные сборки Denver, XAMPP, VertrigoServ. Обычно MySQL используется в качестве

сервера, к которому имеют доступ локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы [13].

На основании концептуальной и логической модели была разработана физическая модель базы данных 19.

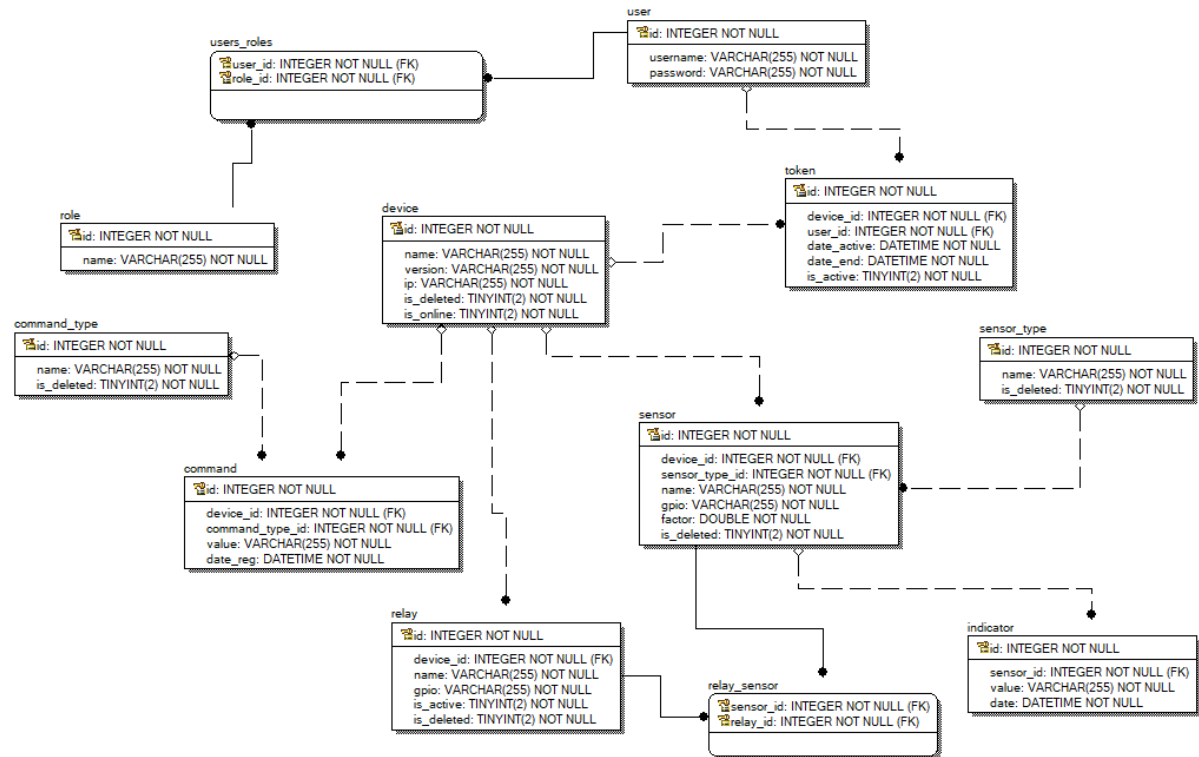


Рисунок 19 – Физическая модель БД

Скрипт для создания таблиц базы данных представлен в приложении А.

2.6 Проектирование программных средств

2.6.1 Проектирование серверного приложения

2.6.1.1 Диаграмма классов серверного приложения

Диаграмма классов описывает типы классов системы и разнообразные связи, которые имеются между ними. Кроме того, в диаграммах классов представляются атрибуты и методы классов. Атрибуты представляют свойства объектов класса. За именованиением атрибута может следовать его тип.

Метод представляет функцию класса. Метод способен обладать параметрами и возвращать значения.

Согласно модели предметной области, была спроектирована диаграмма программных классов, представленная на рисунке 20. В ней выделены только основные модули.

Описание представленных на диаграмме классов приведено в таблице 6.

Таблица 6 – Описание классов

Название класса	Описание
SensorIndicator	ORM представление таблицы «indicator»
SensorType	ORM представление таблицы «sensor_type»
DeviceToken	ORM представление таблицы «token»
CommandType	ORM представление таблицы «command_type»
Relay	ORM представление таблицы «relay»
Sensor	ORM представление таблицы «sensor»
Device	ORM представление таблицы «device»
Command	ORM представление таблицы «command»
RelayRepository	Интерфейс взаимодействия таблицы «relay»
SensorRepository	Интерфейс взаимодействия таблицы «sensor»
DeviceRepository	Интерфейс взаимодействия таблицы «device»
RelayService	Бизнес логика модуля работы с реле питания
SensorService	Бизнес логика модуля работы с датчиками
DeviceService	Бизнес логика модуля работы с УУ
SensorController	Класс, интерпретирующий работу с датчиками
DeviceController	Класс, интерпретирующий работу с УУ

Условно классы в серверном приложении можно разделить на 4 группы: ORM-классы (Entity), интерфейсы взаимодействия с БД (Repository), классы, реализующую бизнес логику (Service), а также классы, интерпретирующие действие пользователей.

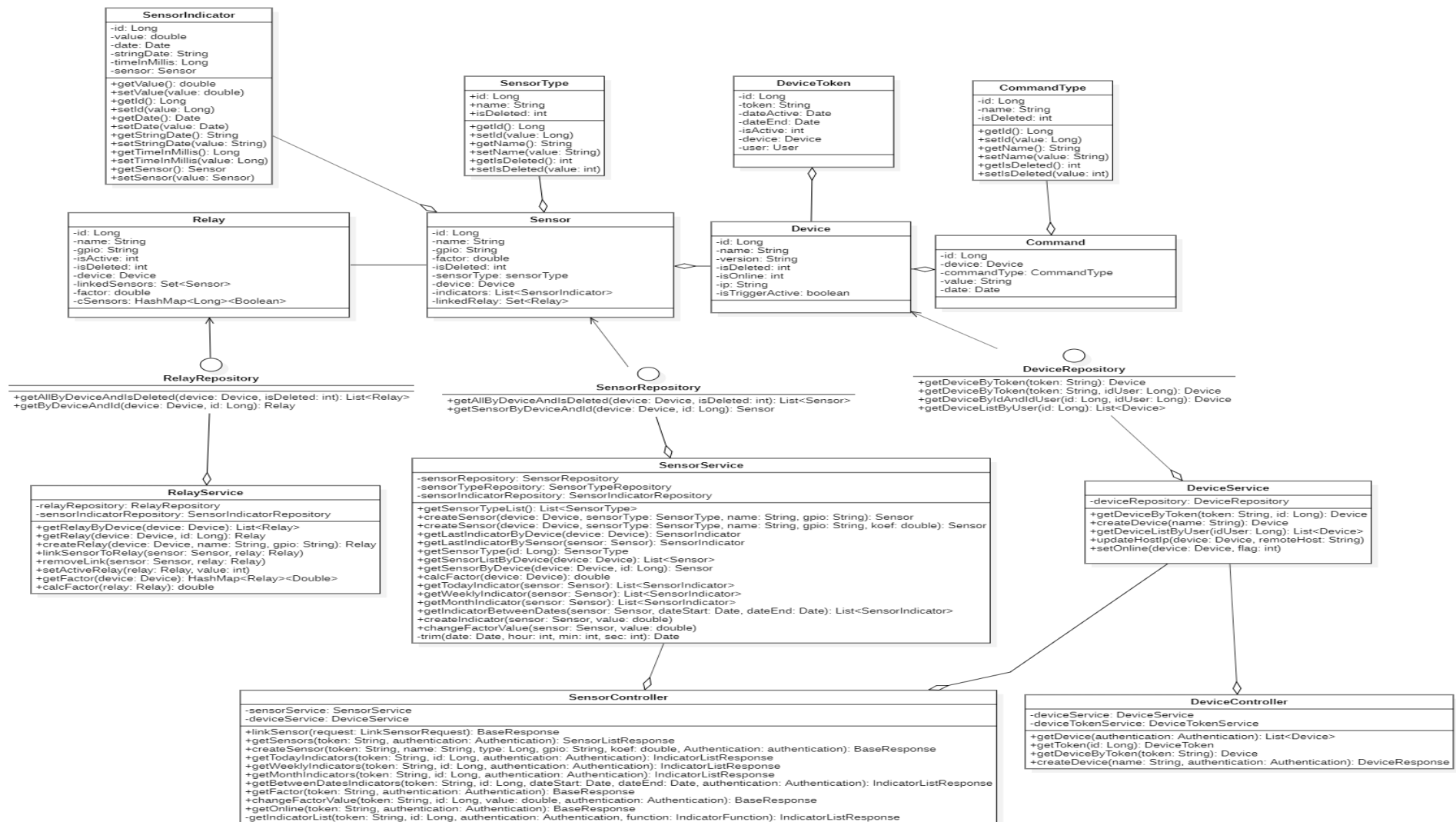


Рисунок 20 – Диаграмма классов серверного приложения

Исходя из диаграммы классов можно сделать вывод, что серверное приложение является многокомпонентным со слабыми связями.

2.6.1.2 Диаграмма деятельности серверного приложения

Диаграмма деятельности показывает разбиение некоторой деятельности на ее составные части. Также на данной диаграмме отображаются логика и последовательность действий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой простой операции, а переход в следующее состояние срабатывает только тогда, когда эта операция завершается в предыдущем состоянии.

В качестве графического представления для выделения основных функций Системы мы применяем диаграмму вариантов использования (use case).

Деятельность может содержать входящие и/или исходящие дуги деятельности, показывающие потоки управления и потоки данных. Если поток соединяет две деятельности, он является потоком управления. Если поток заканчивается объектом, он является потоком данных.

Деятельность выполняется, только тогда, когда готовы все его «входы», после выполнения, деятельность передает управление и(или) данные на свои «выходы». Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали слева направо или сверху вниз.

Последовательность действий при первоначальной настройки и привязывания аппаратных компонентов начинается с авторизацией пользователя на web-сайте, затем ему необходимо выбрать из списка требуемое УУ. После этого, система предлагает пользователю привязать датчики к УУ. Пользователь открывает модальное окно с формой привязки датчика. Аналогичным образом привязывается реле питания. Сценарий работы пользователя с web-сайтом представлен на рисунке 21.

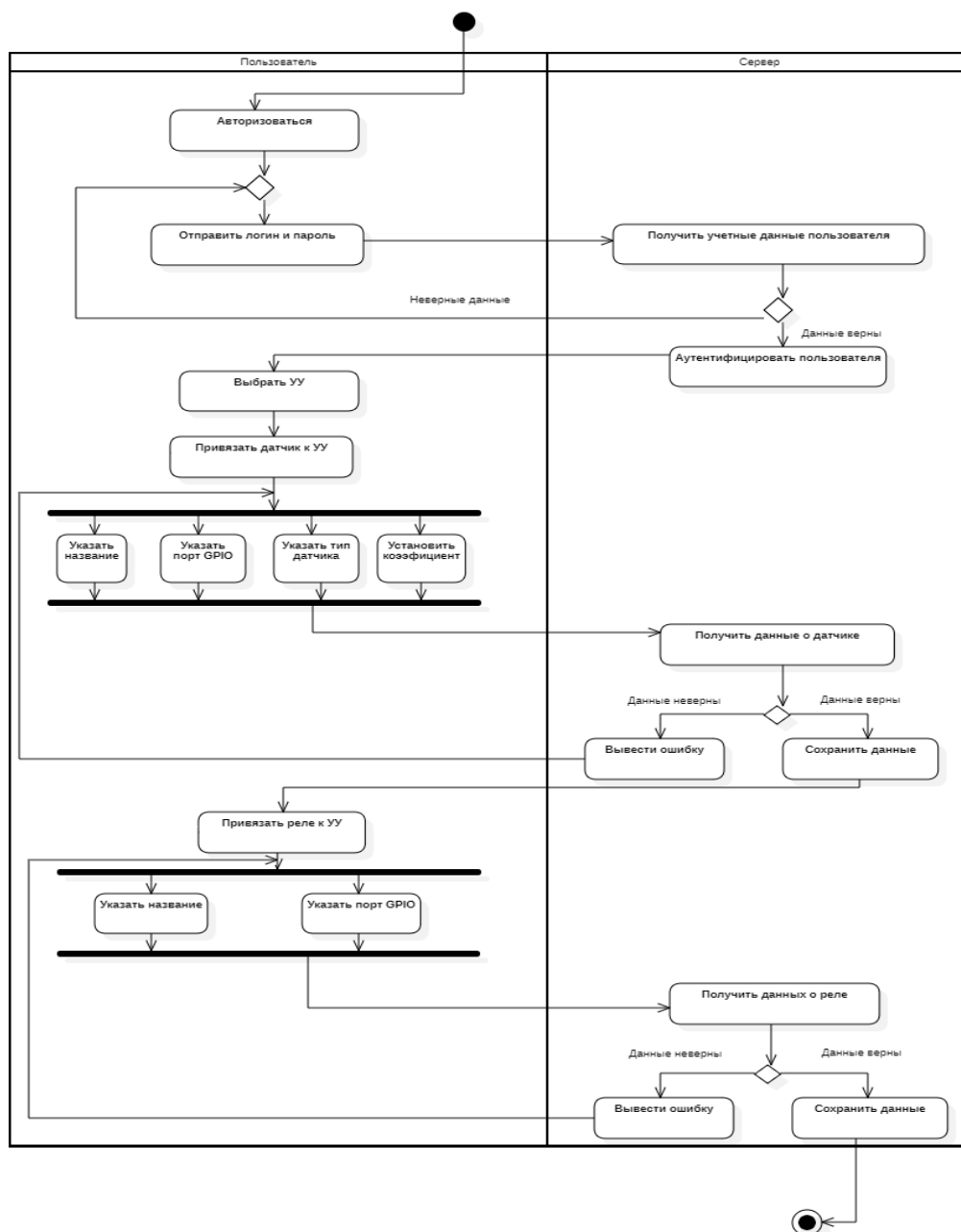


Рисунок 21 – Диаграмма деятельности серверного приложения

Диаграмма описывает процесс привязки датчиков и реле к УУ. Она состоит из двух дорожек, которые представляют собой пользователя и сервер.

2.6.2 Проектирование приложения контроллера

2.6.2.1 Диаграмма классов приложения контроллера

Согласно модели предметной области была спроектирована диаграмма программных классов для приложения контроллера, представленная на рисунке 22.

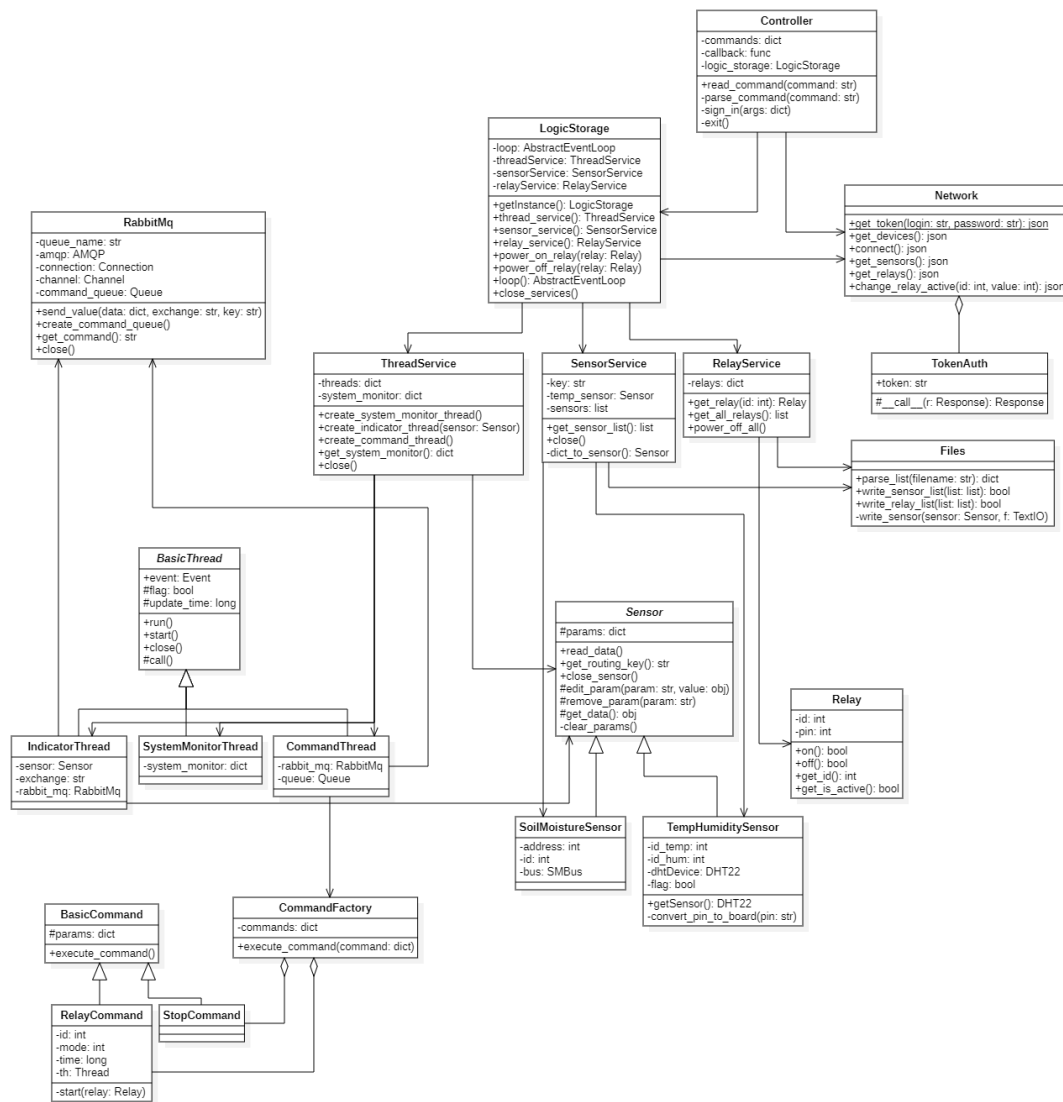


Рисунок 22 – Диаграмма классов приложения контроллера

Описание представленных на диаграмме классов приведено в таблице 7.

Таблица 7 – Описание классов приложения контроллера

Название класса	Описание
Controller	Управляющий класс
LogicStorage	Репозиторий, включающий в себя сервисы бизнес логики
Network	Класс, взаимодействующий с сервером
TokenAuth	Вспомогательный класс, описывающий модель аутентификации
RabbitMq	Класс, взаимодействующий с брокером сообщений
Files	Класс, работающий с файлами
ThreadService	Бизнес логика управления потоками
SensorService	Бизнес логика управления датчиками
RelayService	Бизнес логика управления реле питания
BasicThread	Абстрактный класс, описывающий поведения потоков
IndicatorThread	Поток, взаимодействующий с датчиками
SystemMonitorThread	Поток, взаимодействующий с системной информацией
CommandThread	Поток, взаимодействующий с командами
CommandFactory	Фабрика по созданию команд

Продолжение таблицы 7

Название класса	Описание
BasicCommand	Абстрактный класс, описывающий поведение команд
RelayCommand	Команда, поступающая на реле
StopCommand	Команда, выключающая УУ
Sensor	Абстрактный класс, описывающий поведение датчика
SoilMoistureSensor	Класс, управляющий поведением ДВП
TempHumiditySensor	Класс, управляющий поведением ДТВ
Relay	Класс, управляющий поведением реле питания

Полученную архитектуру можно представить, как слоенную модель, в которой каждый модуль взаимодействует только с соседними. Модель представлена на рисунке 23.



Рисунок 23 – Архитектура приложения контроллера

Она отображает последовательные зависимости от поведенческой модели аппаратных компонентов к контроллеру. Разработанное приложение соответствует всем принципам SOLID (single responsibility, open–closed, Liskov substitution, interface segregation и dependency inversion).

2.6.2.2 Диаграмма деятельности приложения контроллера

Так как данное приложение взаимодействует с пользователем через серверное приложение, путем отправки команд в брокер сообщений, то спроектированная диаграмма будет содержать и этот компонент программного комплекса. На примере отправки команды контроллеру на полив теплицы разработаем диаграмму, которая представлена на рисунке 24.

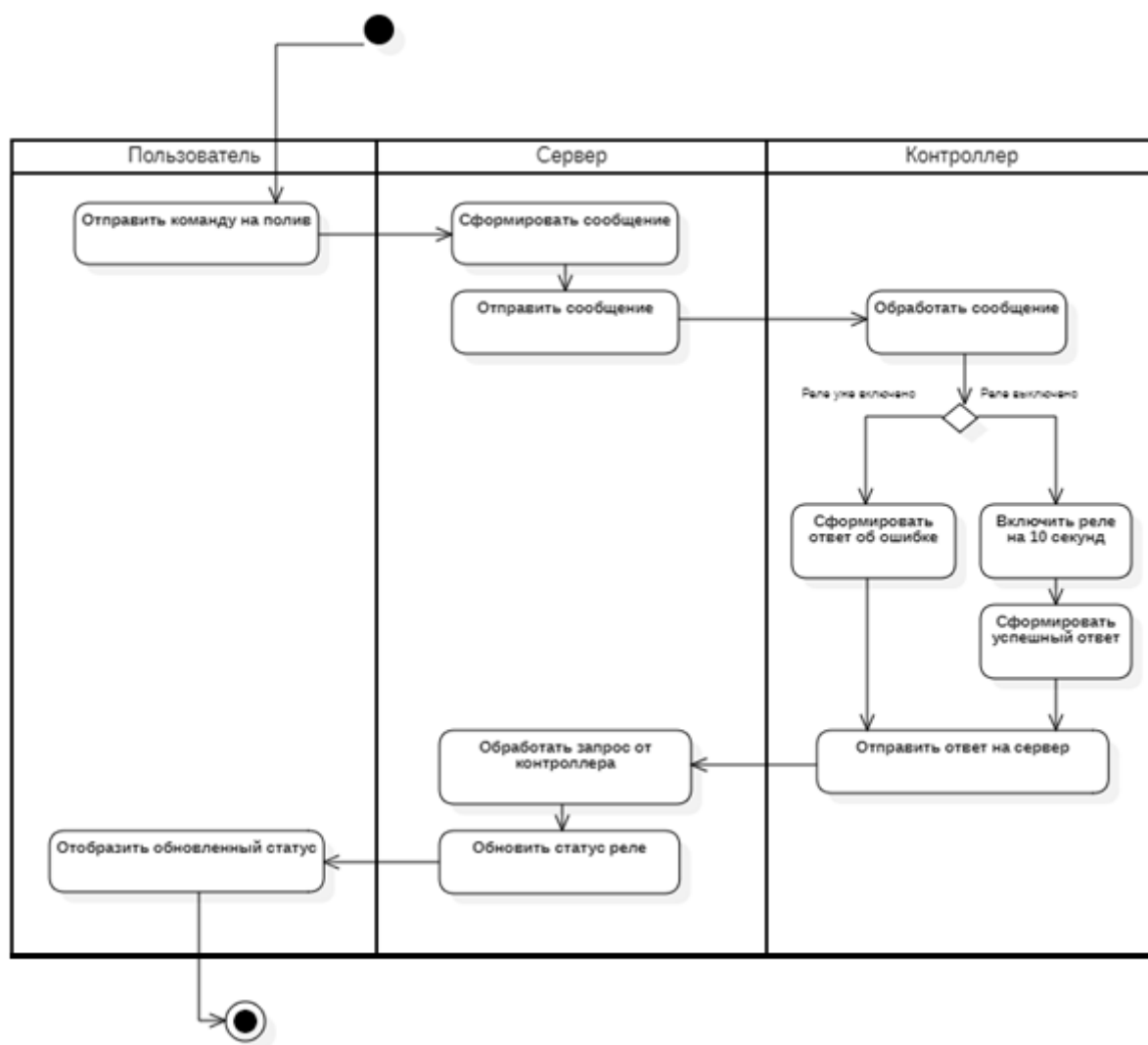


Рисунок 24 – Диаграмма деятельности приложения контроллера на отправку команды

Пользователь с помощью web-интерфейса нажимает на кнопку включения реле питания. Отправляется запрос на сервер, на котором формируется команда на полив. Далее с помощью брокера сообщений она передается на контроллер и обрабатывается. Если реле питание выключено, то оно включается на 10 секунд, иначе формируется ответ с ошибкой. Сформированный ответ отправляется на сервер, где обновляется состояние реле питания и отображается у пользователя.

2.6.3 Диаграмма последовательности программного комплекса

Диаграмма последовательности описывает сценарии системы. В данном виде диаграмм демонстрируются объекты и их коммуникационный аспект. Другими словами, отражаются взаимодействующие объекты, которые обмениваются между собой некоторой информацией [14].

Каждый объект имеет собственную линию жизни, означающую период времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях.

Диаграмма описывает процесс считывания показателя датчика. Она включает в себя 4 объекта: серверное приложение, БД, брокер сообщений, контроллер, которые взаимодействуют в рамках сценария и обмениваются сообщениями. Для отображения процесса «Считывание показателя датчика» была построена диаграмма, представленная на рисунке 25.

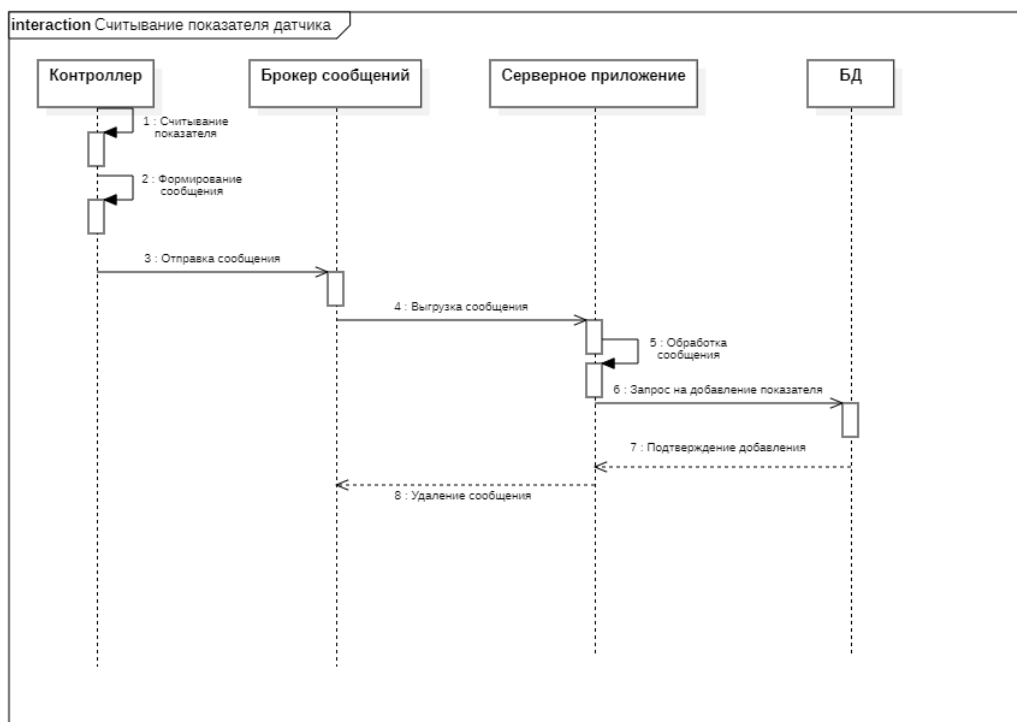


Рисунок 25 – Диаграмма последовательности программного комплекса

Контроллер считывает показания с датчиков и формирует сообщение для отправки в брокер. У каждого датчика есть собственная очередь

сообщений, а все данные отправляются по обменному каналу. Как только поступает новое сообщение, серверное приложение начинает его обработку. Если сообщение не нарушило свою целостность, то отправляется запрос в БД на добавление нового показателя. После этого брокер удаляет прочитанное сообщение.

2.7 Проектирование графического интерфейса комплекса контроля и микроклимата теплицы

Основываясь на диаграмме вариантов использования и анализе требований был реализован пользовательский интерфейс для аппаратно-программного комплекса контроля и микроклимата теплицы. Пользовательский интерфейс позволяет пользователю взаимодействовать с комплексом с помощью web-сайта.

Первой страницей сайта, который увидит пользователь, является форма авторизации. Для входа в систему пользователю необходимо ввести логин и пароль. Экраны авторизации представлен на рисунке 26.

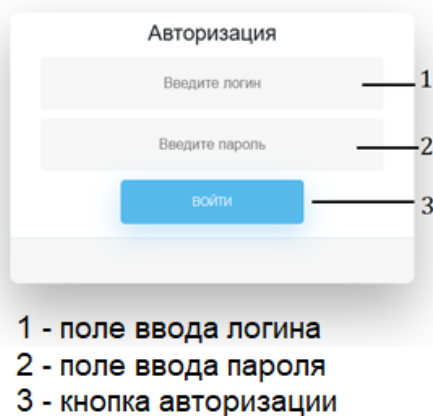


Рисунок 26 – Экран авторизации

После авторизации пользователя перенаправляют на страницу со списком доступных УУ. На этой странице имеется возможность привязать новое УУ или выбрать из списка доступных. Для каждого УУ отображается информация о названии УУ, ip-адресе контроллера, а также дата последней

активности. В целях обеспечения безопасности ip-адрес был скрыт. Интерфейс экрана списка УУ представлен на рисунке 27.



- 1 - пункт меню добавления нового УУ;
- 2 - карточка УУ;
- 3 - название УУ;
- 4 - дата последней активности;
- 5 - ip-адрес контроллера

Рисунок 27 – Экран списка УУ

Кликнув на одну из карточек УУ, пользователь попадает на главную страницу управления УУ. На этой странице происходит весь мониторинг и контроль над контролером УУ. Функционал разделен на 6 блоков: работа с датчиками, работа с реле питанием, просмотр статистики с помощью графиков, настройка триггеров, отправка команд и настройка УУ. Также на шапке сайта отображается статус работоспособности контроллера.

Изначально пользователю открывается блок работы с датчиками. В нем можно просмотреть информацию по датчикам, а также привязать новый. Оформлено в виде карточек и дугообразных индикаторов. Интерфейс главной страницы вместе с блоком работы над датчиками представлен на рисунке 28.



- 1 - навигационное меню;
- 2 - индикация работоспособности контроллера;
- 3 - карточка датчика;
- 4 - название датчика;
- 5 - показание датчика;
- 6 - дата последнего считывания;
- 7 - кнопка привязки нового датчика.

Рисунок 28 – Интерфейс главного меню вместе с блоком работы над датчиками

Если нажать на кнопку «Добавить датчик», то пользователю откроется модальная форма привязки нового датчика, в которой необходимо указать: название датчика, порт GPIO, выбрать тип датчика из выпадающего списка и весовой коэффициент. Интерфейс формы представлен на рисунке 29.

1 - поле ввода названия датчика;
 2 - поле ввода порта GPIO;
 3 - выпадающий список с выбором типа датчика;
 4 - поле ввода весового коэффициента;
 5 - привязка датчика;
 6 - закрытие модального окна.

Рисунок 29 – Интерфейс модальной формы привязки датчика

При выборе пункта меню «Реле» отображается блок в виде таблицы, в котором настраиваются реле питания. Пользователь может увидеть информацию о привязанных реле: название, номер порта GPIO, коэффициент C и его положении (вкл/выкл). У каждой строчки таблицы есть вложенная таблица для привязки датчиков. На рисунке 30 представлен интерфейс работы с реле питания.

1	2	3
Название реле	GPIO	Коэффициент
Тест реле	10	0.78
Статус	2	
Датчик	Привязка	
Темп. комнаты	☒	
Влажность комнаты	☒	
Влажность почвы сектор 1	☒	
4	5	6

1 - таблица списка реле;
 2 - переключатель реле;
 3 - кнопка раскрытия списка датчиков;
 4 - таблица привязанных датчиков;
 5 - название датчика;
 6 - переключатель привязки датчика к реле

Рисунок 30 – Интерфейс работы с реле питания

При выборе пункта меню «Графики» отображается статистическая инфографика для каждого датчика. Пользователь может настроить параметры отображения графиков таких, как: тип датчика, временной промежуток, диапазон дат. На абсциссе расположено время снятия показателя, на ординате его значение. Временной промежуток может быть нескольких видов:

- за текущий день;
- за неделю;
- за месяц;
- по диапазону дат.

На рисунке 31 представлен интерфейс инфографики.



- 1 - выпадающий список выбора датчика;
- 2 - выпадающий список выбора временного диапазона;
- 3 - выпадающий календарь выбора начала диапазона даты;
- 4 - выпадающий календарь выбора окончания диапазона даты;
- 5 - отображение графика;
- 6 - датчик.

Рисунок 31 – Интерфейс инфографики датчиков

При выборе пункта меню «Триггеры» отображается блок для настройки триггеров в виде таблице. Для каждого датчика можно изменить весовой

коэффициент с помощью текстового поля. Данные автоматически отправляются при изменении значения. Интерфейс блока настройки триггеров представлен на рисунке 32.



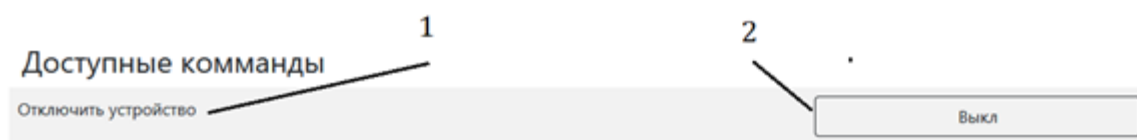
The screenshot shows a table with three columns: 'Название датчика' (Sensor Name), 'Тип датчика' (Sensor Type), and 'Коэффициент' (Coefficient). The table contains three rows of data. A line labeled '1' points to the table header, and a line labeled '2' points to the coefficient input field for the first row.

Название датчика	Тип датчика	Коэффициент
Темп. комната	Датчик температуры	1.1
Влажность комната	Датчик влажности	1.0
Влажность почвы сектор 1	Датчик влажности почвы	1.5

1 - таблица привязанных датчика;
2 - текстовое поле изменения весового коэффициента.

Рисунок 32 – Интерфейс блока настройки триггеров

При выборе пункта меню «Команды» отображается список доступных команд, оформленный в виде кнопок. При нажатии каждой из кнопок сервер отправляет команду на контроллер. Интерфейс отправки команд представлен на рисунке 33.



1 - название команды;
2 - активация команды.

Рисунок 33 – Интерфейс отправки команд

Последний пункт меню «Настройки» позволяет пользователю установить режим работы автоматизации полива. Также в этом блоке отображается информация о токене УУ. На рисунке 34 представлен интерфейс блока «Настройки».

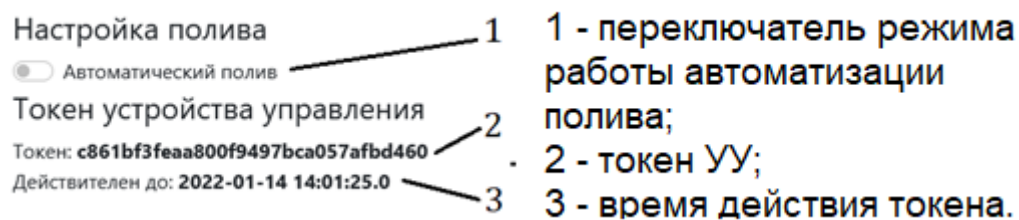


Рисунок 34 – Интерфейс блока настройки УУ

Проектирование интерфейса было выполнено в соответствии с базовыми рекомендациями UX дизайна, дословно означает «опыт пользователя». В более широком смысле это понятие про весь опыт, который получает пользователь при взаимодействии с сайтом или приложением. UX-дизайн отвечает за функции, адаптивность продукта и то, какие эмоции он вызывает у пользователей. Чем понятнее интерфейс, тем легче пользователю получить результат и совершить целевое действие.

Таким образом, графический интерфейс всего приложения состоит из страниц авторизации, списка УУ, а также страницы управления контроллером.

3 Реализация программного комплекса контроля и мониторинга микроклимата теплицы

3.1 Диаграмма компонентов серверного приложения

Все компоненты, за исключением конфигурационного файла, имеют расширение .java и являются классами, написанными на языке Java. Описание классов было представлено выше. Файл с расширением .yml является конфигурационным файлом и задает основные параметры работы и развертывания всего сервера. Пример конфигурационного файла представлен в листинге 1.

Листинг 1 – Пример конфигурационного файла

```
server:
  address: 0.0.0.0
  port: ${PORT}
  fullurl: ${URL}
  apiurl: ${API_URL}

spring:
  profiles:
    active: prod

jpa:
  hibernate:
    ddl-auto: update
  properties:
    hibernate:
      dialect: org.hibernate.dialect.MySQL5Dialect
  show-sql: false

datasource:
  url: ${MYSQL_URL}
  username: ${MYSQL_USERNAME}
  password: ${MYSQL_PASSWORD}
  driver-class-name: com.mysql.cj.jdbc.Driver

rabbitmq:
  host: ${RABBIT_HOST}
  port: ${RABBIT_PORT}
  username: ${RABBIT_USERNAME}
  password: ${RABBIT_PASSWORD}
```

Программные средства серверного приложения состоят из компонентов, которые представлены на рисунке 35.

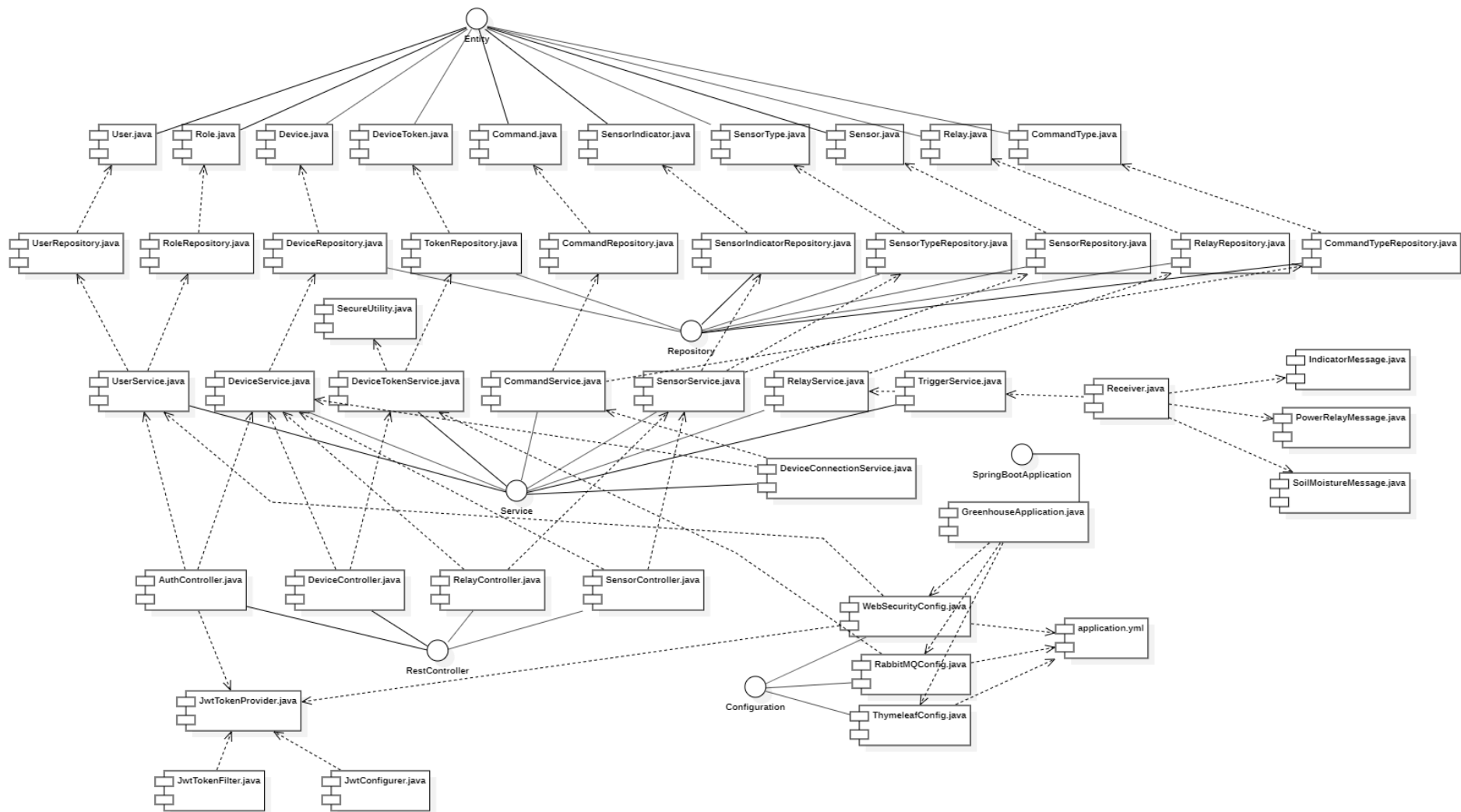


Рисунок 35 – Диаграмма компонентов серверного приложения

Параметры, указанные в фигурных скобках, являются переменными окружения. Переменные окружения в Linux – это специальные переменные, определенные оболочкой и используемые программами во время выполнения. Они могут определяться системой и пользователем. Системные переменные окружения Linux определяются системой и используются программами системного уровня. Благодаря этому легко можно переносить собранное приложение из одного сервера на другой.

3.2 Представление развертывания серверного приложения

Серверное приложение упаковывается в .jar файл с помощью фреймворка для автоматизации сборки пакетов – Maven. Для этого необходимо описать всю структуру приложения в специальный .pom файл. Он обеспечивает декларативную сборку проекта. Файлы описания проекта содержат его спецификацию, а не отдельные команды выполнения. Все задачи обработки файлов, описанные в спецификации, выполняются Maven путем их обработки с помощью последовательности встроенных и внешних плагинов. Для сборки проекта используются специальные команды: *build* и *package*.

После этого необходимо развернуть программный комплекс на ЭВМ (электронно-вычислительная машина). Так как помимо java приложения комплекс включает в себя СУБД MySQL и брокер сообщений RabbitMQ, было принято решение использовать специальный сервис Docker. Это программное обеспечение, предназначенное для автоматизации развертывания и управления приложениями в контейнерных средах. Он позволяет «упаковать» приложение со всей его средой и зависимостями в контейнер, который можно развернуть в любой системе Linux. На рисунке 36 представлена диаграмма развертывания.

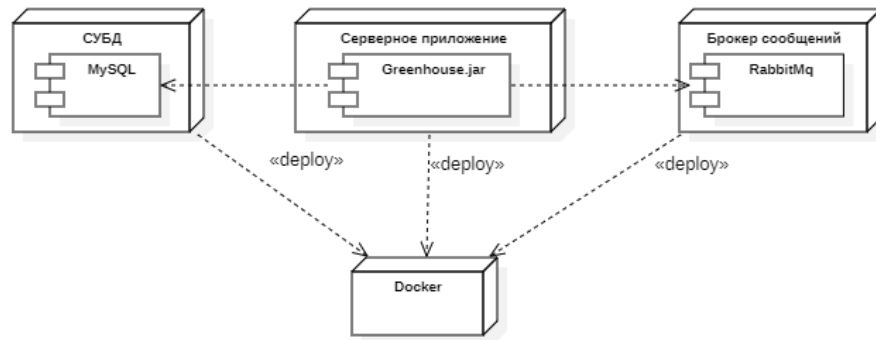


Рисунок 36 – Диаграмма развертывания серверного приложения

Каждый компонент программного комплекса (кроме приложения контроллера) оборачивается в docker-контейнер, который работает независимо друг от друга. Если в какой-либо части системы происходит ошибка, остальные продолжают функционировать.

Контейнеры создаются на основе образов. В стандартном репозитории есть множество готовых образов популярных сервисов. В нашем случае необходимо вручную написать образ для серверного приложения, а образы MySQL и RabbitMQ можно импортировать из облачного хранилища. Для создания образов используются Docker файлы, в которых описывается инструкции по инициализации. В листинге 2 представлено описание Docker файла на создание образа серверного приложения.

Листинг 2 – Инструкция создания образа серверного приложения

```
FROM adoptopenjdk/openjdk11:alpine-jre
ARG JAR_FILE=greenhouse-0.0.1-greenhouse.jar
WORKDIR /opt/app
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","app.jar"]
```

Опишем каждую команду:

- FROM – задает родительский образ;
- ARG – задает переменные для передачи Docker во время сборки образа;
- WORKDIR – задает рабочую директорию для следующей инструкции;

- COPY – копирует в контейнер файлы и папки;
- ENTRYPOINT – предоставляет команду с аргументами для вызова во время выполнения контейнера.

На основе этого образа можно создать и запустить контейнер. Однако, исходя из диаграммы развертывания, разрабатываемое приложение взаимодействует с СУБД и брокером сообщения напрямую. Требуется описать правила взаимодействия и развертывания контейнеров. Это задачу решает встроенный инструмент Docker compose. Это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развёртыванием проектов. Описание инструкций представлено в листинге 3.

Листинг 3 – docker-compose файл

```
version: '3.1'

services:

  db:
    container_name: greenhouse-mysql
    image: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_PASS}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
    ports:
      - 3308:3306
    volumes:
      - ./data/db:/var/lib/mysql

  rabbit:
    container_name: greenhouse-rabbitmq
    image: rabbitmq:3-management-alpine
    restart: always
    environment:
      - RABBITMQ_DEFAULT_USER=${RABBITMQ_USER}
      - RABBITMQ_DEFAULT_PASS=${RABBITMQ_PASS}
    ports:
      - 5673:5672
      - 15673:15672

  app:
    container_name: greenhouse-spring
    build:
      context: ./
      dockerfile: Dockerfile
    restart: always
    environment:
      MYSQL_URL: ${MYSQL_URL}
      MYSQL_USERNAME: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASS}
      RABBIT_USERNAME: ${RABBITMQ_USER}
      RABBIT_PASSWORD: ${RABBITMQ_PASS}
```

```

RABBIT_HOST: rabbit
RABBIT_PORT: 5672
URL: ${FULL_URL}
API_URL: ${API_URL}
PORT: 8081

ports:
- 8080:8081
volumes:
- ./log:/app/logs/

```

В данной инструкции представлено 3 сервиса: db (СУБД MySQL), rabbit (RabbitMQ) и app (Java приложение). Рассмотрим основные атрибуты docker-compose:

- image показывает, какой образ необходимо использовать при сборке.
- restart определяет, в каких случаях требуется перезагрузка контейнера при его аварийном завершении;
- environment определяет переменные окружения для контейнера;
- ports указывает на каком порте работает контейнер, а на каком доступен из вне (хост: контейнер);
- volumes является механизмом для сохранения данных, создаваемых и используемых контейнером (хост: контейнер).

Развертывается комплекс с помощью нескольких команд:

- docker-compose build – сборка контейнеров;
- docker-compose up -d – запуск контейнеров в фоновом режиме.

3.3 Представление развертывания приложения контроллера

Приложение контроллера разворачивается с помощью bash скриптов. Это последовательность команд, которые по очереди считывает и выполняет программа-интерпретатор, в нашем случае это программа командной строки – bash. Так как программа написана на языке python, необходимо первоначально развернуть виртуальную среду со всеми необходимыми библиотеками. Описание создание виртуальной среды python представлено в листинге 4.

Листинг 4 – Скрипт создания виртуальной среды python

```
#!/bin/bash
DIR="venv"
cd ~
if [ ! -d "$DIR" ]; then
    mkdir $DIR
fi
cd $DIR
python3 -m venv env
source env/bin/activate
cd ~
```

Скрипт запуска программы контроллера представлен в листинге 5.

Листинг 5 – Скрипт запуска программы

```
#!/bin/bash
cd $APP_PATH
python3 __main__.py &
cd ~
```

Сама программа выгружается в систему из git репозитория.

Исходные коды программ представлены в приложениях Г и Д.

4 Контроль качества программного комплекса

Тестирование – это разновидность экспериментального исследования операционной системы. В экспериментальной теории тестирование называется пассивным экспериментом, поскольку исследователь действует только как наблюдатель или фиксатор параметров некоторого продолжающегося процесса.

Тестирование системы управления означает выполнение набора тестовых действий для проверки и подтверждения эффективности принципов и функций, принятых при проектировании системы, полноты задач, для которых она была создана, и качества контроля.

4.1 Модульное тестирование

Модульное тестирование – это процесс программирования, который позволяет проверить на корректность отдельные модули исходного кода программы, наборов из одного или нескольких программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки. Основная цель модульного тестирования – изолировать части программ и подтвердить их корректную работу по отдельности.

Для модульного и интеграционного тестирования была использована среда Junit, которая используется для написания unit-тестов на языке Java. В таблице 8 представлено описание модульных тестов. Исходный код тестов представлен в приложении В.

Таблица 8 – Описание модульных тестов

Класс	Тестовый метод	Содержание теста	Ожидаемый результат	Результат
LoginTest	goodAuth	Сделать попытку авторизации пользователя по логину и паролю, указать верные данные.	Должен быть найден пользователь с указанным логином и паролем.	Тест пройден.

Продолжение таблицы 8

Класс	Тестовый метод	Содержание теста	Ожидаемый результат	Результат
LoginTest	badAuth	Сделать попытку авторизации пользователя по неверному логину и паролю, указать верные данные.	Должна быть отловлена ошибка «BadCredentialsException».	Тест пройден.
	registerUser	Зарегистрировать нового пользователя в системе.	Должен быть создан новый пользователь с уникальным логином.	Тест пройден.
DeviceTest	getAllDevice	Попытка получения всех УУ.	Должен вернуться «not null» список.	Тест пройден.
	getDeviceByToken	Попытка получения УУ по токenu.	Должен вернуться УУ.	Тест пройден.
DeviceTest	getDeviceById	Попытка получения УУ по ID.	Должен вернуться УУ.	Тест пройден.
	createDevice	Попытка создания нового УУ.	Должен быть создан новый УУ.	Тест пройден.
	getDeviceListUser	Получение всех привязанных УУ.	Должен вернуться список УУ.	Тест пройден.
SensorTest	createSensorType	Попытка создания нового типа датчика.	Должен создаться новый тип датчика.	Тест пройден.
	createSensor	Попытка создания нового датчика и привязка его к УУ.	Должен создаться новый датчик с привязкой к УУ.	Тест пройден.
	sendIndicator	Попытка создания нового показателя датчика.	Должен создаться новый показатель датчика.	Тест пройден.

На рисунке 37 представлены результаты модульного тестирования.

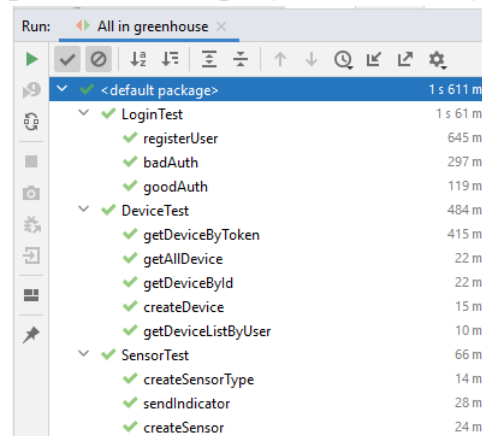


Рисунок 37 – Результаты модульного тестирования

Как видно из рисунка, все тесты были успешно выполнены.

4.2 Интеграционное тестирование

Интеграционное тестирование определяется как тип тестирования, при котором программные модули логически интегрируются и тестируются как

единая группа. Зачастую программный проект состоит из нескольких программных модулей. Цель этого уровня тестирования – выявить дефекты связи между этими программными модулями, когда они интегрированы.

В таблице 9 представлен план и результаты интеграционного тестирования.

Таблица 9 – План и результаты интеграционного тестирования

Класс	Тестовый метод	Содержание теста	Ожидаемый результат	Результат
AuthIntegrationTest	goodAuth	Отправка запроса с корректными данными на авторизацию по url: «/api/1.0/auth/login»	В теле json ответа параметр code = 010	Тест пройден.
	badAuth	Отправка запроса с не корректными данными на авторизацию по url: «/api/1.0/auth/login»	В теле json ответа параметр code = 011	Тест пройден.
	register	Отправка запроса на регистрацию пользователя по url: «/api/1.0/auth/register»	В теле json ответа параметр code = 012	Тест пройден.

На рисунке 38 представлены результаты интеграционного тестирования.

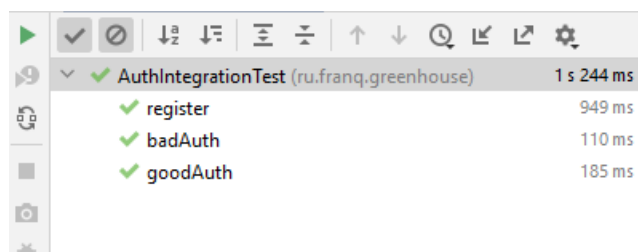


Рисунок 38 – Результаты интеграционного тестирования

Все тесты были успешно пройдены.

4.3 Функциональное тестирование

Для проверки соответствия между реальным поведением разработанного программного обеспечения и ожидаемым поведением был функциональный режим тестирования. Главная цель этого вида тестирования

закключается в том, чтобы проверить реализуемость функциональных требований, то есть способность решить возложенные на него задачи.

Используется метод функциональных диаграмм, который заключается в преобразовании входной спецификации программы в функциональную диаграмму (диаграмму причинно-следственных связей) с помощью простейших логических отношений и в построении таблицы решений, которая является основой для написания эффективных тестовых наборов данных. Построение таблицы решений осуществляется методом обратной трассировки. Описание тестов представлено в таблице 10.

Таблица 10 – Описание спецификаций тестов

№ п/п	Тестируемая ситуация	Описание теста	Тестовые данные	Ожидаемый результат
1	Авторизация с валидными учетными данными	Получение пользовательского токена и сохранение его в куки	Логин, пароль	В теле json ответа параметр code = 010, параметр message содержит токен доступа, токен сохраняется в куки
2	Получение списка устройств	Получение привязанных устройств у пользователя	Токен доступа пользователя	Список привязанных устройств
3	Получение списка датчиков	Получение списка привязанных датчиков к устройству	Токен доступа пользователя, токен устройства	Список привязанных датчиков к устройству

Для каждой тестируемой ситуации построим функциональную диаграмму. На рисунке 39 представлена диаграмма теста №1.

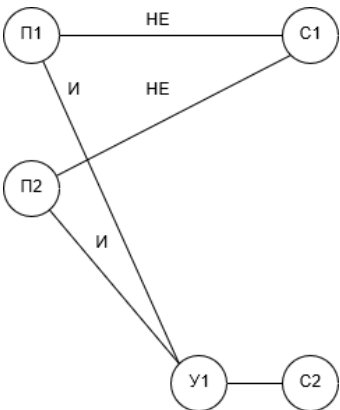


Рисунок 39 – Функциональная диаграмма теста № 1

На рисунке 40 представлена диаграмма теста №2.

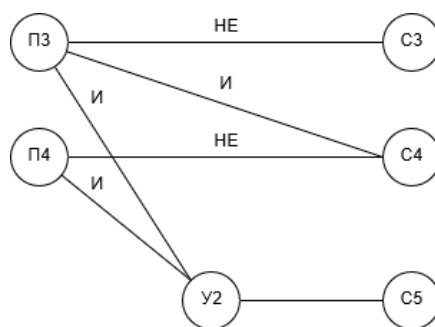


Рисунок 40 – Функциональная диаграмма теста № 2.

На рисунке 41 представлена диаграмма теста №3.

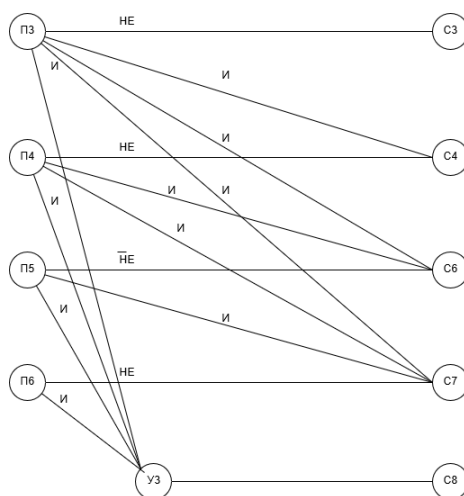


Рисунок 41 – Функциональная диаграмма теста № 3

Описание причин представлено в таблице 11.

Таблица 11 – Описание причин

Причина	Описание
П1	Корректное значение логина
П2	Корректное значение пароля
П3	Корректное значение пользовательского токена
П4	Срок действия пользовательского токена не истек
П5	Корректное значение токена устройства
П6	Срок действия токена устройства не истек

Описание следствий представлено в таблице 12.

Таблица 12 – Описание следствий

Следствие	Описание
С1	Неверное значение логина или пароля
С2	Авторизация успешна
С3	Некорректный токен пользователя
С4	Токен пользователя истек

Продолжение таблицы 12

Следствие	Описание
C5	Получен список устройств
C6	Некорректный токен устройства
C7	Токен устройства истек
C8	Получен список привязанных датчиков

В таблице 13 представлена таблица решений для первого теста, составленная на основе функциональной диаграммы.

Таблица 13 – Таблица решений теста № 1

	1	2	3
ПЗ	0	1	1
П4	0	0	1
C1	1	1	0
C2	0	0	1

В таблице 14 представлен набор тестов на основе таблице решений 6. Правильный логин и пароль – roman;roman.

Таблица 14 – Набор тестов

№	Входные данные	Ожидаемый результат
1	Логин: ssss Пароль: asdasd	(C1)
2	Логин: roman Пароль: asdasd	(C1)
3	Логин: ssss Пароль: roman	(C1)
4	Логин: roman Пароль: roman	(C2)

В таблице 15 представлена таблица решений для второго теста, составленная на основе функциональной диаграммы.

Таблица 15 – Таблица решений теста № 2

	1	2	3
П4	0	1	1
П5	0	0	1

Продолжение таблицы 15

C3	1	0	0
C4	0	1	0
C5	0	0	1

В таблице 16 представлен набор тестов на основе таблице решений 8. Правильный токен – testtoken, годен до: 10.04.2021, token2, годен до 01.01.20201.

Таблица 16 – Набор тестов

№	Входные данные	Ожидаемый результат
1	Токен: sdjfhdsjfhdsjf	(C3)
2	Токен: token2	(C4)
3	Токен: testtoken	(C5)

В таблице 17 представлена таблица решений для третьего теста, составленная на основе функциональной диаграммы.

Таблица 17 – Таблица решений теста 3

	1	2	3	4	5	6	7
ПЗ	0	1	1	1	0	1	1
П4	0	0	1	1	0	0	1
П5	0	0	0	1	1	1	1
П6	0	0	0	0	0	0	1
C3	1	0	0	0	1	0	0
C4	0	1	0	0	0	1	0
C6	0	0	1	0	0	0	0
C7	0	0	0	1	0	0	0
C8	0	0	0	0	0	0	1

В таблице 18 представлен набор тестов на основе таблице решений 17. Правильный пользовательский токен – testtoken, годен до: 10.04.2021, token2, годен до 01.01.20201. Правильный токен устройства – devtoken, годен до 10.04.2022, xtoken, годен до 01.01.2021

Таблица 18 – Набор тестов

№	Входные данные	Ожидаемый результат
1	Токен пользователя: sdfhdsjfhdsjf	(C3)
2	Токен пользователя: testtoken	(C4)
3	Токен пользователя: token2 Токен устройства: dfghdfjkgfkggh	(C6)
4	Токен пользователя: token2 Токен устройства: xtoken	(C7)
5	Токен пользователя: dsfsdfdsf Токен устройства: xtoken	(C3)
№	Входные данные	Ожидаемый результат
6	Токен пользователя: testtoken Токен устройства: xtoken	(C4)
7	Токен пользователя: token2 Токен устройства: devtoken	(C8)

Функциональное тестирование показало, что методы авторизации работают корректно. При условии неверных введенных учетных данных, система отправляет соответствующее предупреждение об ошибке.

4.4 Метрики кода

Метрика устойчивости для компоненты отражает количественную меру работы, которую необходимо произвести при изменении других зависимых компонент. Для вычисления позиционной устойчивости кода используется следующая формула:

$$I = \frac{C_e}{C_a + C_e},$$

где I – мера неустойчивости;

C_e – количество классов вне данного компонента, зависящих от классов внутри компонента;

C_a – количество классов внутри данного компонента, зависящих от классов вне компонента.

Величина I изменяется в диапазоне от $[0, 1]$. Если $I = 0$, то компонент максимально устойчив, если $I = 1$, то компонент максимально неустойчив.

В созданном приложении можно выделить 6 логических компонент:

– слой «Controller»;

- компонент бизнес логики (Service);
- компонент взаимодействия БД;
- слой «Model»;
- вспомогательный компонент;
- компонент конфигурации.

Ниже для каждой компоненты была просчитана величина – метрика неустойчивости:

$$I_1 = \frac{0}{6+0} = 0; I_2 = \frac{3}{9+3} = 0.25; I_3 = \frac{5}{8+5} = 0.38;$$

$$I_4 = \frac{8}{0+8} = 1; I_5 = \frac{1}{0+1} = 1; I_6 = \frac{0}{3+0} = 0.$$

Метрика A , являющаяся мерой абстрактности компонента, равна отношению количества абстрактных классов к общему количеству классов в компоненте,

где N_c – количество классов в компоненте;

N_a – количество абстрактных классов в нем же.

Тогда абстрактностью A называется величина.

$$A = \frac{N_a}{N_c}.$$

Метрика A изменяется от 0 до 1, причем 0 означает, что в компоненте вообще нет абстрактных классов, а 1 – что компонент содержит только абстрактные классы:

$$A_1 = \frac{0}{2} = 0; A_2 = \frac{0}{8} = 0; A_3 = \frac{8}{8} = 1; A_4 = \frac{0}{10} = 0; A_5 = \frac{0}{1} = 0; A_6 = \frac{0}{1} = 0.$$

Вычислив меру устойчивости компоненты и ее абстрактность можно отобразить их на графике. На вертикальной оси необходимо отложить меру абстрактности A , а на горизонтальной оси – меру неустойчивости I .

Компонент, который располагаются в верхнем левом углу или в нижнем правом являются максимально качественным. На практике часто случается, так что полученная точка лежит на диагонали между точкой (1,0) и точкой (0,1).

Для вычисления количественной оценки используется формула вычисления расстояние до главной последовательности:

$$D = \left| \frac{(A + I - 1)}{\sqrt{2}} \right|.$$

С помощью данной метрики категории проекта могут быть проанализированы на их полное соответствие главной последовательности. Метрика D может быть рассчитана для каждой категории. Любая категория, которая имеет значение D не близкое к нулю, может быть повторно исследована и реструктурирована.

Чем ближе точка располагается к главной последовательности, тем лучше. Был построен соответствующий график для логических компонент приложения, который отображен на рисунке 42.

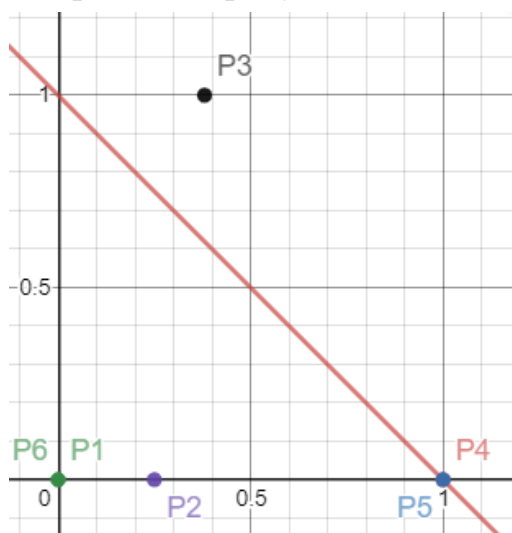


Рисунок 42 – Отображение метрик на графике

Было вычислено расстояние до главной последовательности:

$$D_1 = \left| \frac{(0 + 0 - 1)}{\sqrt{2}} \right| = 0.707; D_2 = \left| \frac{(0.25 + 0 - 1)}{\sqrt{2}} \right| = 0.53;$$

$$D_3 = \left| \frac{(0.38 + 1 - 1)}{\sqrt{2}} \right| = 0.268; D_4 = \left| \frac{(1 + 0 - 1)}{\sqrt{2}} \right| = 0;$$

$$D_5 = \left| \frac{(1 + 0 - 1)}{\sqrt{2}} \right| = 0; D_6 = \left| \frac{(0 + 0 - 1)}{\sqrt{2}} \right| = 0.707;$$

Первая и последняя компонента достаточно удалены от главной последовательности и находятся в зоне неприятностей. Это обусловлено тем, что данные компоненты являются слоем контроллера и конфигурации, их задача только вызывать методы бизнес логики, поэтому этот компонент имеет устойчивость. Нулевая абстрактность обусловлена малым количеством классов в данной компоненте.

В целом можно заметить, что только 3 компоненты имеет абстрактные классы, т.к. этот модуль является оберткой для работы с БД и используется в компоненте бизнес-логики. Также, специфика задачи, позволяет минимизировать использования абстрактных классов.

4.5 Планирование и оценка бюджета

Планирование разработки и бюджетная оценка были выполнены с использованием программного обеспечения для управления проектами Microsoft Project [15]. Начало проекта – 14 апреля 2021 года. При планировании разработки было выделено 28 задач. Список задач и их описание представлено на рисунке 43.

	Реги задач	Название задачи	Длительность	Начало	Окончание	Предшественн	Названия ресурсов
1		Начало проекта	0 дней	Ср 14.04.21	Ср 14.04.21		
2		Анализ предметной области	5 дней	Ср 14.04.21	Вт 20.04.21	1	Аналитик
3		Анализ аппаратных компонентов	1 день	Ср 14.04.21	Ср 14.04.21	1	
4		Анализ программных аналогов	1 день	Чт 15.04.21	Чт 15.04.21	3	
5		Постановка задачи	1 день	Пт 16.04.21	Пт 16.04.21	4	
6		Анализ функциональных требований	2 дней	Пн 19.04.21	Вт 20.04.21	5	
7		Анализ предметной области завершен	0 дней	Вт 20.04.21	Вт 20.04.21	6	
8		Проектирование программных средств	11 дней	Ср 21.04.21	Ср 05.05.21	2	
9		Проектирование базовой архитектуры	3 дней	Ср 21.04.21	Пт 23.04.21		Аналитик
10		Проектирование алгоритма автоматизированного управления поливом	2 дней	Пн 26.04.21	Вт 27.04.21		Старший инженер
11		Проектирование схемы развертывания аппаратных компонентов	3 дней	Пн 26.04.21	Ср 28.04.21	9	Аналитик
12		Проектирование базы данных	3 дней	Чт 29.04.21	Пн 03.05.21		Java разработчик
13		Проектирование программных средств	2 дней	Вт 04.05.21	Ср 05.05.21	9;10;11;12	Java разработчик;Python разработчик;Аналитик
14		Проектирование завершено	1 день	Ср 05.05.21	Ср 05.05.21	13	
15		Реализация программного комплекса	16 дней	Чт 06.05.21	Чт 27.05.21	8	Выделенный облачный сервер[1 ед];ДВН[3 шт];ДТВ[
16		Реализация серверного приложения	12 дней	Чт 06.05.21	Пт 21.05.21		Java разработчик
17		Реализация приложения контроллера	6 дней	Чт 20.05.21	Чт 27.05.21		Python разработчик;Старший инженер
18		Реализация завершена	1 день	Чт 27.05.21	Чт 27.05.21	17	
19		Тестирование	6 дней	Пт 28.05.21	Сб 05.06.21	15	
20		Модульное тестирование	2 дней	Пт 28.05.21	Пн 31.05.21		Java разработчик
21		Интеграционное тестирование	3 дней	Пн 31.05.21	Ср 02.06.21		Python разработчик
22		Функциональное тестирование	2 дней	Ср 02.06.21	Чт 03.06.21		Java разработчик
23		Анализ качества кода	2 дней	Пт 04.06.21	Сб 05.06.21		Старший инженер
24		Тестирование завершено	0 дней	Сб 05.06.21	Сб 05.06.21	23	
25		Завершение проекта	6 дней	Пн 07.06.21	Пн 14.06.21	19	
26		Составление документации	5 дней	Пн 07.06.21	Пт 11.06.21		Java разработчик;Python разработчик
27		Сдача в эксплуатацию	2 дней	Сб 12.06.21	Пн 14.06.21	26	Аналитик;Старший инженер
28		Проект завершен	0 дней	Пн 14.06.21	Пн 14.06.21	27	

Рисунок 43 – Список задач проекта

Для реализации проекта было выделено четыре трудовых ресурса: Java разработчик, Python разработчик, аналитик, старший инженер и 6 материальных ресурсов: контроллер, ДВП, ДТВ, реле питания, пластиковый бокс и выделенный облачный сервер. Описание ресурсов представлено на рисунке 44.

И	Название ресурса	Тип	Единицы измерения материала	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочн	Затраты на исполн.	Начисление	Базовый календарь
1	Java разработчик	Трудовой		JD		100%	200,00 Р/ч	400,00 Р/ч	0,00 Р	Пропорционал	Стандартный
2	Python разработчик	Трудовой		PD		100%	170,00 Р/ч	340,00 Р/ч	0,00 Р	Пропорционал	Стандартный
3	Аналитик	Трудовой		A		100%	210,00 Р/ч	420,00 Р/ч	0,00 Р	Пропорционал	Стандартный
4	Старший инженер	Трудовой		ME		100%	210,00 Р/ч	420,00 Р/ч	0,00 Р	Пропорционал	Стандартный
5	Контроллер RP3	Материальный	шт	K			3 400,00 Р		0,00 Р	Пропорционал	
6	ДВП	Материальный	шт	Д			200,00 Р		0,00 Р	Пропорционал	
7	ДТВ	Материальный	шт	Д			170,00 Р		0,00 Р	Пропорционал	
8	Реле питания	Материальный	шт	P			70,00 Р		0,00 Р	Пропорционал	
9	Пластиковый бокс	Материальный	шт	П			1 800,00 Р		0,00 Р	Пропорционал	
10	Выделенный облачный сервер	Материальный	ед	B			0,00 Р		570,00 Р	Пропорционал	

Рисунок 44 – Список ресурсов проекта

В результате распределения задач и необходимых ресурсов проекта была построена диаграмма Ганта, представленная на рисунке 45. Диаграмма Ганта показывает задачи, продолжительность и отношения между задачами в проекте. Кроме того, на диаграмме показаны даты начала и окончания каждой задачи и назначенные ресурсы для выполнения задачи.

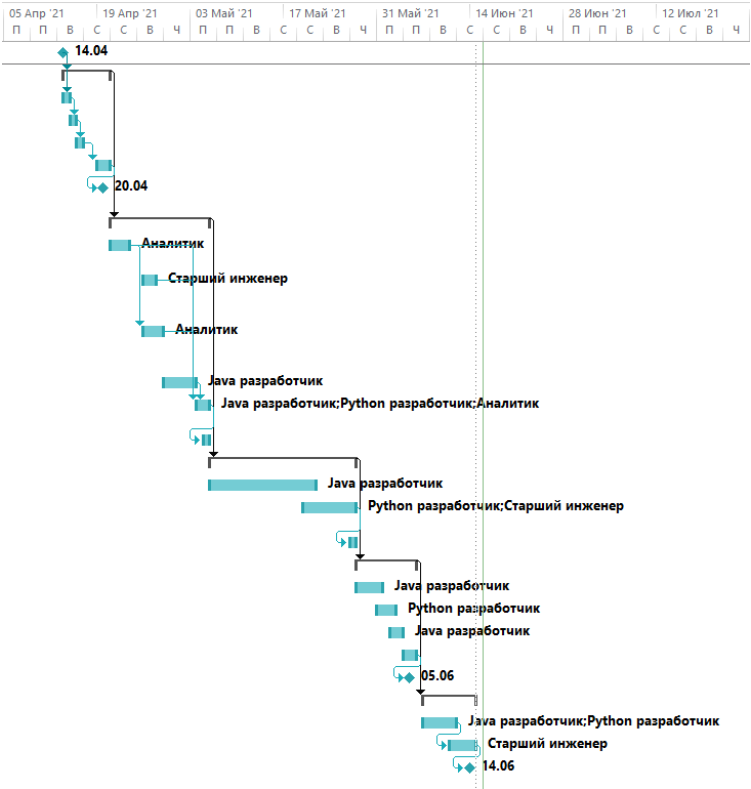


Рисунок 45 – Диаграмма Ганта

На рисунке 46 представлена гистограмма загрузки ресурсов.

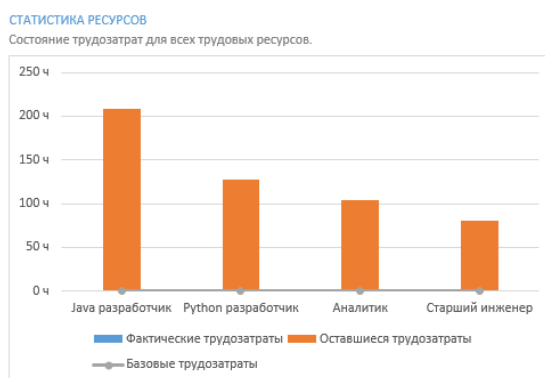


Рисунок 46 – Гистограмма загрузки ресурсов

На рисунке 47 представлена гистограмма затрат на задачи.

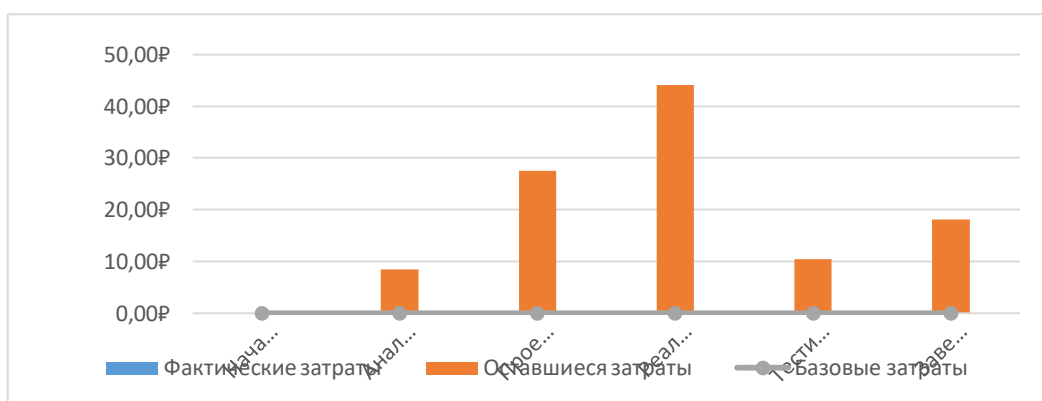


Рисунок 47 – Гистограмма затрат на задачи

Таким образом разработка проекта составит 44 полных рабочих дней с 14 апреля 2021 года по 14 июня 2021 года. Трудозатраты составят 520 человеко-часов. Оценка бюджета проекта составляет 108 610 рубля.

Заключение

В рамках данной выпускной квалификационной работы был разработан комплекс программных и аппаратных средств контроля и мониторинга микроклимата теплицы. Особенности разработанного ПО, отличающими его других программных аналогов, является использование многослойной архитектуры и высокий уровень масштабирования функционала.

Серверное приложение разработано на языке программирования Java помощью фреймворка Spring. Приложение контроллера реализована на языке Python в среде PyCharm 2020. В качестве СУБД выступает MySQL, использующий язык SQL. Брокером сообщений является RabbitMQ, основанный на стандарте AMQP.

В ходе работы было проведено функциональное и структурное тестирование системы, осуществлён качественный анализ метрик кода. Результаты тестирования показали, что разработанное ПО работает корректно и соответствует ожидаемым требованиям.

Произведено планирование и оценка бюджета программных средств в среде Microsoft Office Project 2016. На основании определенных задач и ресурсов проекта построена диаграмма Ганта, которая показывает календарный график выполнения работ по проекту. Срок выполнения проекта составил 43 дня, а бюджет – 108 610 рублей. Анализ сроков и бюджета свидетельствует о том, что разработка данного проекта является экономически выгодной.

Таким образом, в результате выполнения выпускной квалификационной работы был разработан комплекс программных и аппаратных средств контроля и мониторинга микроклимата теплицы, отвечающий всем требованиям технического задания.

Список использованных источников

1. Промышленный интернет вещей [Электронный ресурс]. – URL: <https://investmoscow.ru/media/3340535/03-промышленный-интернет-вещей.pdf#:~:text=«Интернет%20вещей»%20-%20это%20вычислительная,части%20системы%2C%20так%20и%20глобальных> (дата обращения: 01.04.2021).
2. Как работают датчики DHT [Электронный ресурс]. – URL: <https://radioprogram.ru/post/744> (дата обращения 03.04.2021).
3. Знакомство в шиной IC2 [Электронный ресурс]. – URL: <https://ph0en1x.net/94-adc-dac-pcf8591-raspberry-pi-tutorial-i2c-bus-python.html> (дата обращения 04.04.2021).
4. Технические характеристики Raspberry Pi 3 Model B [Электронный ресурс]. – URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> (дата обращения 25.04.2021).
5. Официальный сайт компании Бионика [Электронный ресурс]. – URL: <https://www.bionia.ru/teplidy/terraform/> (дата обращения 25.04.2021).
6. Официальный сайт компании 2agrocloud [Электронный ресурс]. – URL: <https://2agrocloud.com> (дата обращения 25.04.2021).
7. Арлоу Д. UML 2 и Унифицированный процесс, 2-е издание. Практический объектно-ориентированный анализ и проектирование. / Арлоу Д., Нейштадт А. – СПб: Символ Плюс, 2007. – 624 с.
8. Маклаков, С. В. Моделирование бизнес-процессов с BPwin 4.0. / С.В. Маклаков. – Москва: ДИАЛОГ-МИФИ, 2002. – 218 с.
9. Проектирование информационных систем с СА ERwin Modeling Suite 7.3: учебное пособие / В. И. Горбаченко, Г. Ф. Убиенных, Г. В. Бобрышева – Пенза: Изд-во ПГУ, 2012. – 154 с.
10. Документация «Spring Framework» [Электронный ресурс]. – URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/> (дата обращения 12.05.2021).

11. Документация «Bootstrap 5» [Электронный ресурс]. – URL: <https://bootstrap5.ru/docs/getting-started/introduction> (дата обращения 13.05.2021).
12. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение Теория и практика. / Коннолли Т., Бегг К. – Санкт Петербург: Питер, 2003. – 1440 с.
13. СУБД MySQL [электронный ресурс] – URL: https://depix.ru/articles/sistema_upravleniya_bazami_dannyh_mysql (дата обращения 17.12.2020).
14. Леоненков, А.В. Самоучитель UML2 / А.В. Леоненков [Электронный ресурс]. – URL: <https://www.e-reading.club/book.php?book=33640> (дата обращения: 23.05.2021).
15. Управление проектами в Microsoft Project: учеб.-метод. пособие для студентов по направлению специальности «Информационные системы и технологии (издательско-полиграфический комплекс)» / Т. П. Брусенцова, В. В. Смелов. – Минск: БГТУ, 2011. – 160 с.

ПРИЛОЖЕНИЕ А
ГЛОССАРИЙ
(справочное)

AMQP – открытый протокол для передачи сообщений между компонентами системы.

АО – аналоговый выход.

API – программный интерфейс приложения.

DO – цифровой выход.

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений.

GPIO – интерфейс ввода/вывода общего назначения.

I2C – последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов.

Java – строго типизированный объектно-ориентированный язык программирования общего назначения.

JDBC – платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.

MVC – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер.

MySQL – свободная реляционная система управления базами данных.

ORM – объектно-реляционное отображение.

Python – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью

RP3 – одноплатный контроллер Raspberry Pi 3.

UML – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов.

АЦП – аналого-цифровой преобразователь.

БД – база данных.

ДВП – датчик влажности почвы.

ДТВ – датчика температуры и влажности.

Интернет вещей – концепция сети передачи данных между физическими объектами, оснащёнными встроенными средствами и технологиями для взаимодействия друг с другом или с внешней средой.

Контейнеризация – метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного.

ПО – программное обеспечение.

СУБД – система управления базой данных.

Токен доступа – средство идентификации пользователя или отдельного сеанса работы в компьютерных сетях и приложениях.

УУ – устройство управления.

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

ПРИЛОЖЕНИЕ Б
Описание атрибутов сущностей БД
(рекомендуемое)

Таблица Б.1 – Описание атрибутов для сущности «Пользователь»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор пользователя	Int	11	0-9	Нет	PK	1
username	Логин пользователя	Varchar	255	«A-z», «0-9», « », «-»	Нет		roman
password	Зашифрованный пароль пользователя	Varchar	255	Любые значения	Нет		\$2a\$10\$wfZvmbQ3/d1/utLeKXicXOTCkYV.qLgIMtdk

Таблица Б.2 – Описание атрибутов для сущности «Роль»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор роли	Int	11	0-9	Нет	PK	1
name	Роль пользователя	Varchar	255	«A-z»	Нет		USER

Таблица Б.3 – Описание атрибутов для сущности «УУ»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор роли	Int	11	0-9	Нет	PK	1
name	Название УУ	Varchar	255	«A-z», «А-я», «0-9»	Нет		Дача
version	Версия установленного ПО	Varchar	255	«0-9», «.»	Нет		1.0
ip	Ip адрес контролера	Varchar	255	«0-9», «.»	Нет		192.168.1.217
is_online	Флаг активности устройства	tinyint	2	0,1	Нет		1
is_deleted	Флаг удаленного устройства	tinyint	2	0,1	Нет		0

Таблица Б.4 – Описание атрибутов для сущности «Токен доступа»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор токена	Int	11	0-9	Нет	PK	1
token	Значение токена	Varchar	255	«А-з», «А-я», «0-9»	Нет	U	c861bf3feaa800f9497bca057afbd460
device_id	Идентификатор УУ	Int	11	0-9	Нет	FK	1
user_id	Идентификатор пользователя	Int	11	0-9	Нет	FK	1
date_active	Дата активация токена	Datetime		2021-01-01 – 2100-01-01	Нет		2021-01-01 12:00:54
date_end	Дата деактивации токена	Datetime		2021-01-01 – 2100-01-01	Нет		2022-01-01 12:00:54
is_active	Флаг активности токена	tinyint	2	0,1	Нет		1

Таблица Б.5 – Описание атрибутов для сущности «Датчик»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор датчика	Int	11	0-9	Нет	PK	1
name	Название датчика	Varchar	255	«А-з», «А-я», «0-9»	Нет		ДВП 1
device_id	Идентификатор УУ	Int	11	0-9	Нет	FK	1
sensor_type_id	Идентификатор типа датчика	Int	11	0-9	Нет	FK	1
gpio	Номер порта GPIO	Varchar	10	«0-9»	Нет		40
factor	Коэффициент датчика для СПР	Double			Нет		1,1
is_deleted	Флаг удаленного датчика	tinyint	2	0,1	Нет		1

Таблица Б.6 – Описание атрибутов для сущности «Реле питания»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор реле	Int	11	0-9	Нет	PK	1
name	Название реле	Varchar	255	«А-з», «А-я», «0-9»	Нет		ДВП 1
device_id	Идентификатор УУ	Int	11	0-9	Нет	FK	1
sensor_type_id	Идентификатор типа датчика	Int	11	0-9	Нет	FK	1
gpio	Номер порта GPIO	Varchar	10	«0-9»	Нет		40
is_deleted	Флаг удаленного реле	tinyint	2	0,1	Нет		1
is_active	Имеет ли реле логическую единицу	tinyint	2	0,1	Нет		1

Таблица Б.7 – Описание атрибутов для сущности «Команда»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор команды	Int	11	0-9	Нет	PK	1
device_id	Идентификатор УУ	Int	11	0-9	Нет	FK	1
command_type_id	Идентификатор типа команды	Int	11	0-9	Нет	FK	1
value	Значение команды	Varchar	255	Любые значения	Нет		30
date_reg	Дата отправки команды	Datetime		2021-01-01 – 2100-01-01	Нет		2022-01-01 12:00:54

Таблица Б.8 – Описание атрибутов для сущности «Показатель датчика»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор показателя	Int	11	0-9	Нет	PK	1
sensor_id	Идентификатор датчика	Int	11	0-9	Нет	FK	1
value	Значение команды	Varchar	255	Любые значения	Нет		23.3
date_reg	Дата считывания показателя	Datetime		2021-01-01 – 2100-01-01	Нет		2022-01-01 12:00:54

Таблица Б.9 – Описание атрибутов для сущности «Типа датчика»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор типа датчика	Int	11	0-9	Нет	PK	1
name	Название типа датчика	Varchar	255	«A-z»	Нет		ДВП
is_deleted	Флаг удаленного типа датчика	tinyint	2	0,1	Нет		1

Таблица Б.10 – Описание атрибутов для сущности «Категория команды»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Идентификатор категории	Int	11	0-9	Нет	PK	1
name	Название категории	Varchar	255	«A-z»	Нет		Остановить
is_deleted	Флаг удаленной категории	tinyint	2	0,1	Нет		1

ПРИЛОЖЕНИЕ В

Исходный код модульного тестирования (рекомендуемое)

Листинг В.1 – Файл LoginTest.java

```
package ru.franq.greenhouse;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class LoginTest {

    @Autowired
    UserService userService;

    @Autowired
    AuthenticationManager authenticationManager;

    @Test
    public void goodAuth() {
        String login = "roman";
        String pass = "roman";
        assertNotNull(auth(login, pass));
    }

    @Test
    public void badAuth() {
        String login = "roman";
        String pass = "roman123";
        try {
            assertNull(auth(login, pass));
        } catch (BadCredentialsException ex) {
            assertNotNull(ex);
        }
    }

    @Test
    public void registerUser() {
        String login = "bbb";
        String pass = "nnn";

        User user = userService.findUserByUsername(login);
        assertNull(user);
        User user1 = new User(login, pass);
        assertTrue(userService.saveUser(user1));
    }

    public User auth(String login, String pass) {
        authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(login, pass));
        return userService.findUserByUsername(login);
    }
}
```

Листинг В.2 – Файл DeviceTest.java

```
package ru.franq.greenhouse;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class DeviceTest {
```

```

@Autowired
DeviceService deviceService;

@Autowired
UserService userService;

@Autowired
AuthenticationManager authenticationManager;

@Test
public void getAllDevice() {
    List<Device> deviceList = deviceService.getAllDevice();
    assertNotNull(deviceList);
}

@Test
public void getDeviceByToken() {
    Device device =
deviceService.getDeviceByToken("c861bf3feaa800f9497bca057afbd460");
    assertNotNull(device);
}

@Test
public void getDeviceById() {
    Device device = deviceService.getDeviceById(20L);
    assertNotNull(device);
}

@Test
public void createDevice() {
    Device device = deviceService.createDevice("YU 4");
    assertNotNull(device);
}

@Test
public void getDeviceListByUser() {
    List<Device> devices = deviceService.getDeviceListByUser(3L);
    assertNotNull(devices);
}
}

```

Листинг В.3 – Файл SensorTest.java

```

package ru.franq.greenhouse;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class SensorTest {

    @Autowired
    SensorTypeRepository sensorTypeRepository;

    @Autowired
    DeviceRepository deviceRepository;

    @Autowired
    SensorRepository sensorRepository;
}

```

```

@Autowired
SensorIndicatorRepository sensorIndicatorRepository;

@Test
public void createSensorType(){
    SensorType sensorType = new SensorType("Датчик влажности
почвы");

    SensorType st = sensorTypeRepository.saveAndFlush(sensorType);
    assertNotNull(st);
}

@Test
public void createSensor(){
    Device device = deviceRepository.getOne(20L);
    SensorType sensorType = sensorTypeRepository.getOne(2L);
    Sensor sensor = new Sensor("Датчик2", sensorType, device);
    sensorRepository.save(sensor);
}

@Test
public void sendIndicator(){
    Sensor sensor = sensorRepository.getOne(1L);
    SensorIndicator indicator = new SensorIndicator(24.1, sensor);
    sensorIndicatorRepository.saveAndFlush(indicator);
    assertNotNull(indicator);
}
}

```

ПРИЛОЖЕНИЕ Г

**Исходный код серверного приложения
(рекомендуемое)**

Листинг Г.1 – Файл GreenhouseApplication.java

```
package ru.franq.greenhouse;

@EnableScheduling
@SpringBootApplication(excludeName = "Greenhouse server")
public class GreenhouseApplication{

    public static void main(String[] args) {
        SpringApplication.run(GreenhouseApplication.class, args);
    }

}
```

Листинг Г.2 – Файл DeviceRepository.java

```
package ru.franq.greenhouse.repository;

public interface DeviceRepository extends JpaRepository<Device, Long> {

    @Query(
        value = "SELECT d.*\n" +
            "from device d\n" +
            "left join token t on d.id = t.device_id\n" +
            "where t.token = ?;", nativeQuery = true
    )
    Device getDeviceByToken(String token);

    @Query(
        value = "SELECT d.*\n" +
            "from device d\n" +
            "left join token t on d.id = t.device_id\n" +
            "left join user u on t.user_id = u.id\n" +
            "where t.token = ? and u.id = ?;", nativeQuery =
true
    )
    Device getDeviceByToken(String token, Long idUser);

    @Query(
        value = "SELECT d.*\n" +
            "from device d\n" +
            "left join token t on d.id = t.device_id\n" +
            "left join user u on t.user_id = u.id\n" +
            "where d.id = ? and u.id = ?;", nativeQuery = true
    )
    Device getDeviceByIdAndIdUser(Long id, Long idUser);

    @Query(
        value = "SELECT d.*\n" +
            "from device d\n" +
            "left join token t on d.id = t.device_id\n" +
            "where t.user_id = ?;", nativeQuery = true
    )
    List<Device> getDeviceListByUser(Long id);

}
```

Листинг Г.3 – Файл DeviceService.java

```
package ru.franq.greenhouse.service;

@Service
public class DeviceService {

    @Autowired
    private DeviceRepository deviceRepository;

    @Deprecated
    public List<Device> getAllDevice() {
        return deviceRepository.findAll();
    }

    @Deprecated
    public Device getDeviceById(Long id) {
        return deviceRepository.getOne(id);
    }

    public Device getDeviceByToken(String token, Long idUser) throws
BadTokenException {
        Device device = deviceRepository.getDeviceByToken(token,
idUser);
        if (device != null) {
            return device;
        } else {
            throw new BadTokenException();
        }
    }

    public Device getDeviceByToken(String token) throws
BadTokenException {
        Device device = deviceRepository.getDeviceByToken(token);
        if (device != null) {
            return device;
        } else {
            throw new BadTokenException();
        }
    }

    public Device createDevice(String name) {
        Device device = new Device();
        device.setName(name);
        device.setVersion("1.1.2");

        deviceRepository.saveAndFlush(device);
        return device;
    }

    public void changeTriggerValue(Device device, boolean value) {
        device.setTriggerActive(value);
        deviceRepository.saveAndFlush(device);
    }

    public List<Device> getDeviceListByUser(Long idUser) {
        return deviceRepository.getDeviceListByUser(idUser);
    }

    public void updateHostIp(Device device, String remoteHost) {
        device.setIp(remoteHost);
        deviceRepository.saveAndFlush(device);
    }
}
```

```

    }

    public void setOnline(Device device, int i) {
        device.setIsOnline(i);
        deviceRepository.saveAndFlush(device);
    }
}

```

Листинг Г.4 – Файл DeviceController.java

```

package ru.franq.greenhouse.controller.apil_0;

import java.util.List;

@RestController
@RequestMapping("/api/1.0/device")
public class DeviceController {

    @Autowired
    private DeviceService deviceService;
    @Autowired
    private DeviceTokenService deviceTokenService;
    @GetMapping("/list")
    public List<Device> getDevice(Authentication authentication) {
        Long id = ((User) authentication.getPrincipal()).getId();
        return deviceService.getDeviceListByUser(id);
    }
    @GetMapping("/token")
    public DeviceToken getToken(@RequestParam(name = "id") Long id) {
        return deviceTokenService.getTokenByDevice(id);
    }

    @GetMapping("/by_token")
    public Device getDeviceByToken(@RequestParam(name = "token") String
token) throws BadTokenException {
        return deviceService.getDeviceByToken(token);
    }
    @PostMapping("/setting/trigger")
    public BaseResponse changeTriggerActive(@RequestParam(name =
"token") String token,
                                           @RequestParam(name =
"value") boolean value,
                                           Authentication
authentication) throws BadTokenException {
        Long id = ((User) authentication.getPrincipal()).getId();
        Device device = deviceService.getDeviceByToken(token, id);
        deviceService.changeTriggerValue(device, value);
        return new BaseResponse("001", BaseResponse.Status.OK, "Trigger
active has been change");
    }
    @PostMapping("/create")
    public DeviceResponse createDevice(@RequestParam(name =
"device_name") String name, Authentication authentication) {
        User user = (User) authentication.getPrincipal();
        Device device = deviceService.createDevice(name);
        DeviceToken token = deviceTokenService.createToken(user,
device);

        System.out.println(token);
        return new DeviceResponse(device, token);
    }
}

```

ПРИЛОЖЕНИЕ Д
Исходный код приложения контроллера
(рекомендуемое)

Листинг Д.1 – Файл __main__.py

```
import asyncio
import logging
import logging.config
import sys
import config
from Logic.files import Files
from Logic.network import Network
from View.Controller import Controller

is_exit = False

file_log = logging.FileHandler('Log.log')
console_out = logging.StreamHandler()

logging.basicConfig(handlers=(file_log, console_out),
                    format=u'%(levelname)s: [%asctime)s]
%(filename)s[LINE:%(lineno)d]# %(message)s',
                    level=logging.INFO)

def callback(is_success, command, log, flag):
    global is_exit
    is_exit = flag
    if not is_success:
        logging.error(f'{command}: {log}')
    else:
        logging.info(f'{command}: {log}')

def main():
    logging.info('Запуск программы')
    logging.info('Попытка соединиться с сервером...')

    loop = asyncio.get_event_loop()
    connect_msg = loop.run_until_complete(Network.connect())
    if 'error' not in connect_msg:
        if connect_msg['code'] == "010":
            logging.info("Соединение установлено!")
        else:
            logging.error(f"Не удалось создать сессию!
{connect_msg['message']}")
            sys.exit()
    else:
        logging.error(f"Не удалось соединиться с сервером:
{connect_msg['error']}")
        sys.exit()

    sensors = loop.run_until_complete(Network.get_sensors())
    if 'error' not in sensors:
        Files.write_sensor_list(sensors['data'])
        logging.info(f"Список датчиков, полученных от сервера:
{sensors['data']}")
    else:
        logging.error(f"Не удалось получить данные по датчикам:
{connect_msg['error']}")
        sys.exit()

    relays = loop.run_until_complete(Network.get_relays())
    if 'error' not in relays:
        Files.write_relay_list(relays['relays'])
```

```

        logging.info(f"Список реле, полученных от сервера:
{relays['relays']}")
    else:
        logging.error(f"Не удалось получить данные по реле:
{connect_msg['error']}")
        sys.exit()

    logging.debug(config.start_msg)
    controller = Controller(lambda is_success, command, log, is_exit:
callback(is_success, command, log, is_exit))

    try:
        while not is_exit:
            # input_command = input()
            # console.read_command(input_command)
            continue

        logging.info('Finish program...')
        sys.exit()
    except KeyboardInterrupt as ex:
        logging.info('KeyboardInterrupt....')
        controller.read_command('exit')
        sys.exit()

if __name__ == "__main__":
    main()

```

Листинг Д.2 – Файл Controller.py

```

import config
from Logic.LogicStorage import LogicStorage
from Logic.network import Network

class Controller:
    """
    callback возвращает 4 аргумента:
    is_success - успешно ли завершилась команда
    command - какая команда выполнялась
    log - текстовый лог команды
    is_exit - требуется ли выход из программы
    """
    def __init__(self, callback):
        self._callback = callback

        self._commands = {
            'help': lambda args: self._help(),
            'exit': lambda args: self._exit(),
            'get_system_monitor': lambda args: self._show_system_info(),
            'connect': lambda args: self._sign_in(args)
        }

        self._logic_storage = LogicStorage()

    def read_command(self, command):
        arr = self._parse_command(command)
        try:
            self._commands[arr[0]](arr[1])
        except KeyError:

```

```

        self._callback(False, arr[0],
f'{config.command_not_found_msg}: {command}', False)
        except IndexError:
            self._callback(False, arr[0], f'{config.bad_args_msg}
{command}', False)

    def _parse_command(self, command: str):
        arr = command.split(' ')
        return arr[0], arr[1:]

    def _show_system_info(self):
        self._callback(True, 'get_system_monitor',
self._logic_storage.thread_service().get_system_monitor(), False)

    def _help(self):
        self._callback(True, 'help', config.command_list, False)

    def _sign_in(self, *args):
        token =
self._logic_storage.loop().run_until_complete(Network.get_token(args[0][0],
args[0][1]))
        if 'token' in token:
            self._callback(True, 'connect', config.success_connect_msg,
False)
        else:
            self._callback(False, 'connect', token["error"], False)

    def _exit(self):
        self._logic_storage.close_services()
        self._callback(True, 'exit', config.exit_msg, True)

```

Листинг Д.3 – Файл LogicStorage.py

```

import asyncio
import logging

from Logic.network import Network
from Logic.SensorService import SensorService
from Logic.ThreadService import ThreadService
from Logic.RelayService import RelayService

class MetaSingleton(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(MetaSingleton,
cls).__call__(*args, **kwargs)
        return cls._instances[cls]

class LogicStorage(metaclass=MetaSingleton):

    def __init__(self):
        logging.info("Init Logic Storage")
        self._loop = asyncio.get_event_loop()
        self._threadService = ThreadService()
        self._sensorService = SensorService()
        self._relayService = RelayService(10)

```

```

        self._threadService.create_system_monitor_thread()
        self._loop.set_exception_handler(lambda x: logging.info(x))

        for s in self._sensorService.get_sensor_list():
            self._threadService.create_indicator_thread(s)

        self._threadService.create_command_thread()

    @classmethod
    def getInstance(cls):
        if not cls.__instance:
            cls.__instance = LogicStorage()
        return cls.__instance

    def thread_service(self) -> ThreadService:
        return self._threadService

    def sensor_service(self) -> SensorService:
        return self._sensorService

    def power_on_relay(self, relay):
        if relay.on():

self._loop.run_until_complete(Network.change_relay_active(relay.get_id(),
True))

        def power_off_relay(self, relay):
            if relay.off():

self._loop.run_until_complete(Network.change_relay_active(relay.get_id(),
False))

        def relay_service(self) -> RelayService:
            return self._relayService

        def loop(self):
            return self._loop

        def close_services(self):
            logging.info("Close threads...")
            self._threadService.close()
            for r in self.relay_service().get_all_relays():
                r.off()

self._loop.run_until_complete(Network.change_relay_active(r.get_id(), False))

```

Листинг Д.4 – Файл ThreadService.py

```

from Logic.Sensors.Sensor import Sensor
from Logic.Threads.CommandThread import CommandThread
from Logic.Threads.IndicatorThread import IndicatorThread
from .Threads.SystemMonitorThread import SystemMonitorThread
import logging
import config

class ThreadService:
    _threads = {
        'system_monitor': None,
        'command_thread': None,

```

```

        'indicator_thread': []
    }
    _system_monitor = {}

    def __init__(self) -> None:
        super().__init__()

    def create_system_monitor_thread(self):
        logging.info("Creating system monitor thread")
        t = SystemMonitorThread()
        self._threads['system_monitor'] = t
        t.start()

    def create_indicator_thread(self, sensor: Sensor):
        t = IndicatorThread(sensor)
        self._threads['indicator_thread'].append(t)
        t.start()

    def create_command_thread(self):
        t = CommandThread(f"command_{config.device_token}")
        self._threads['command_thread'] = t
        t.start()

    def get_system_monitor(self):
        self._threads['system_monitor'].event.set()
        data = self._threads['system_monitor'].get_system_monitor()
        self._threads['system_monitor'].event.set()
        return data

    def close(self):
        logging.info(f"thread list: {self._threads}")

        self._threads['system_monitor'].close()
        self._threads['command_thread'].close()

        for t in self._threads['indicator_thread']:
            t.close()

```

Листинг Д.5 – Файл SensorService.py

```

from Logic.Sensors.SoilMoistureSensor import SoilMoistureSensor
from Logic.Sensors.TempHumiditySensor import TempHumiditySensor
from Logic.files import Files

class SensorService:
    __key = 'indicator'
    __tempSensor = None
    __sensors = []

    def _dict_to_sensor(self, sensor):
        if sensor['type'] == '1':
            self.__tempSensor = TempHumiditySensor(sensor['gpio'],
            sensor['tempId'], sensor['humId'])
            return self.__tempSensor
        elif sensor['type'] == '3':
            return SoilMoistureSensor(sensor['id'], sensor['gpio'])

    def get_sensor_list(self):
        file_sensor = Files.parse_list()
        for sensor in file_sensor:

```

```

        val = self._dict_to_sensor(sensor)
        self.__sensors.append(val)
    return self.__sensors

    def close(self):
        for sensor in self.__sensors:
            sensor.close_sensor()

```

Листинг Д.6 – Файл SoilMoistureSensor.py

```

# only for rb3
import smbus
from Logic.Sensors.Sensor import Sensor
import logging

class SoilMoistureSensor(Sensor):

    __main_address = 0x48

    def __init__(self, id, address) -> None:
        super().__init__()
        self._id = id
        self._address = int(address, 16)
        self._bus = smbus.SMBus(1)

    def read_data(self):
        try:
            super().read_data()
            self._bus.write_byte(self.__main_address, self._address)
            value = self._bus.read_byte(self.__main_address) * 3.3 / 255
            value = round(100 - (max(0, round(value, 3) - 1.281)) /
(1.618 - 1.281) * 100, 3)
            self._edit_param('id', self._id)
            self._edit_param('humidity', value)
            return self._get_data()
        except Exception as error:
            logging.error(f"Произошла ошибка при чтении данных от
датчика влажности почвы: {error}")
            return None

    def get_routing_key(self):
        return 'soil_moisture'

```

Листинг Д.7 – Файл TempHumiditySensor.py

```

import adafruit_dht
import board
import logging
from Logic.Sensors.Sensor import Sensor

class TempHumiditySensor(Sensor):

    __flag = True

    def __init__(self, gpio, id_temp, id_hum) -> None:
        self._id_temp = id_temp
        self._id_hum = id_hum

```

```

        self._dhtDevice =
adafruit_dht.DHT22(self.__convert_pin_to_board(gpio))

    def getSensor(self):
        return self._dhtDevice

    def read_data(self):
        super().read_data()
        if self.__flag:
            try:
                temperature_c = self._dhtDevice.temperature
                humidity = self._dhtDevice.humidity
                self._edit_param('temp', temperature_c)
                self._edit_param('humidity', humidity)
                self._edit_param('tempId', self._id_temp)
                self._edit_param('humId', self._id_hum)
                return self._get_data()
            except RuntimeError as error:
                logging.error(f"Произошла ошибка при чтении данных из
датчика температуры и влажности! {error.args[0]}")
                return None
            except Exception as error:
                logging.error(error)
                return None
        return None

    def __convert_pin_to_board(self, pin):
        if pin == '4':
            return board.D4
        elif pin == '17':
            return board.D17
        return board.D4

    def close_sensor(self):
        self._dhtDevice.exit()

    def get_routing_key(self):
        return 'temperature'

```

Листинг Д.8 – Файл IndicatorThread.py

```

import config
from Logic.Sensors.Sensor import Sensor
from Logic.Threads.BasicThread import BasicThread
from Logic.rabbitMQ import RabbitMQ

class IndicatorThread(BasicThread):
    _sensor = None
    _exchange = 'sensor.indicator'

    def __init__(self, sensor: Sensor):
        super().__init__(config.sensor_update_time)
        self._sensor = sensor
        self._rabbit_mq = RabbitMQ()

    def _call(self):
        val = self._sensor.read_data()
        if val is not None:

```

```

        self._rabbit_mq.send_value(self._sensor.read_data(),
self._exchange,
                                self._sensor.get_routing_key())

    def close(self):
        super().close()
        self._sensor.close_sensor()
        self._rabbit_mq.close()
        return None

```

Листинг Д.9 – Файл network.py

```

import functools

import requests_async as requests
from requests.auth import AuthBase

import config
from Logic.files import Files

def check_error():
    def decorator(function):
        @functools.wraps(function)
        async def wrapper(*args, **kwargs):
            try:
                return await function(*args, **kwargs)
            except requests.ConnectionError as e:
                return {'error': e}

        return wrapper

    return decorator

class TokenAuth(AuthBase):
    def __init__(self, token):
        self.token = token

    def __call__(self, r):

        r.headers['Authorization'] = f'Bearer {self.token}'
        return r

class Network:

    @staticmethod
    @check_error()
    async def get_token(login, password):
        r = await requests.post(config.host + 'api/1.0/auth/login',
                                json={'login': login, 'password':
password})
        if r.status_code == 200:
            Files.change_config_param('api_token', r.json()['message'])
            return r.json()

    @staticmethod
    @check_error()
    async def get_devices():
        r = await requests.get(config.host + 'api/1.0/device/list',

```

```

        auth=TokenAuth(config.client_token))

    return r.json()

    @staticmethod
    @check_error()
    async def connect():
        r = await requests.get(config.host + 'api/1.0/auth/check',
                                auth=TokenAuth(config.client_token),
                                params={
                                    'token': config.device_token
                                })

        return r.json()

    @staticmethod
    @check_error()
    async def get_sensors():
        r = await requests.get(config.host + 'api/1.0/sensor/list',
                                auth=TokenAuth(config.client_token),
                                params={
                                    'token': config.device_token
                                })

        return r.json()

    @staticmethod
    @check_error()
    async def get_relays():
        r = await requests.get(config.host + 'api/1.0/relay/list',
                                auth=TokenAuth(config.client_token),
                                params={
                                    'token': config.device_token
                                })

        return r.json()

    @staticmethod
    @check_error()
    async def change_relay_active(id, value):
        r = await requests.post(config.host + 'api/1.0/relay/active',
                                data={'token': config.device_token,
                                      'id_relay': id,
                                      'action': value},
                                auth=TokenAuth(config.client_token))

        return r.json()

```