

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

**ЭЛЕКТРОННЫЙ ЗАМОК НА ОСНОВЕ
МИКРОКОНТРОЛЛЕРА STM32**

Студент

 31.05.2024 Вяльмисов М.О.
(подпись, дата)

Руководитель

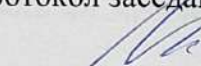
 31.05.2024 Зинкин С.А.
(подпись, дата)

Нормоконтролёр

 31.05.2024 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 27.05.2024 г. № 11)

Заведующий кафедрой

 М.А. Митрохин
(подпись)

Работа защищена с  отметкой _____ (протокол заседания ГЭК от 10.06.24 № 3)

Секретарь ГЭК

 Е.И. Гурин
(подпись)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«19» 04 2024г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Вяльмисов Максим Олегович гр.20BBB1 факультета вычислительной техники.

2. Тема работы: «Электронный замок на основе микроконтроллера STM32»

Тема утверждена приказом ПГУ № 405/о от "19" апреля 2024 г.

3. Руководитель бакалаврской работы д.т.н., профессор, каф. ВТ ПГУ Зинкин С.А.

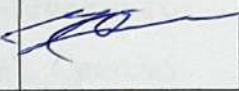
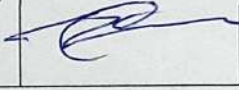
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программу «Электронный замок на основе микроконтроллера STM32», позволяющую организовать автоматизированный процесс контроля доступа в помещение при помощи умного замка. Проект должен состоять из клиент-серверного приложения, где клиент реализован на микроконтроллере STM32. Система должна обрабатывать события попыток входа и постоянно поддерживать связь с сервером.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие автоматизированные системы, выделить их достоинства и недостатки. Выбрать язык программирования, который подойдет для создания сервера. Определить архитектуру приложения и вспомогательные средства разработки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	19.04.2024 – 21.05.2024	
2	Анализ требований	32	22.04.2024 – 23.05.2024	
3	Выбор технологий, применяемых при проектировании и разработке	30	24.04.2024 – 26.04.2024	
4	Установка и настройка средств разработки	40	27.04.2024 – 30.04.2024	
5	Разработка основной части программы	66	1.05.2024 – 7.05.2024	
6	Реализация графического интерфейса	70	8.05.2024 – 23.05.2024	
7	Оформление пояснительной записки	54	24.05.2024 – 31.05.2024	

Дата выдачи задания "19" апреля 2024 г.

Руководитель бакалаврской работы  Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент  19.04.24 Вальмисов М.О.
(подпись, дата)

Работу к защите допустить
Декан факультета  06.06.2024 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 101 лист, 29 рисунков, 5 таблиц, 12 источников и 4 приложения

ЭЛЕКТРОННЫЙ ЗАМОК НА ОСНОВЕ МИКРОКОНТРОЛЛЕРА STM32

Цель работы – разработать программу «Электронный замок на основе микроконтроллера STM32», позволяющую организовать автоматизированный процесс контроля доступа в помещение, сохранение истории о проходе пользователей и с дальнейшей возможностью расширения путем добавления новых пользователей. Проект должен состоять из клиент-серверного приложения, где клиент реализован на микроконтроллере STM32. Система должна обрабатывать события попыток входа и постоянно поддерживать связь с сервером.

Объектами разработки являются:

- серверное приложение для обработки запросов, сохранения данных в базу данных (БД);
- клиентское приложение, реализованное на микроконтроллере STM32.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче, а именно: программа клиент для STM32, сервер на NestJS, брокер сообщений развернутый в docker. Также была спроектирована база данных Postgres.

					ПГУ 1.090301.05.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Электронный замок на основе микроконтроллера STM32 Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Вяльмисов М.О.		31.05.24					
Провер.		Зинкин А.С.		31.05.24			2	105	
Н. Контр		Бычков А.С.		31.05.24		ФВТ, каф ВТ гр. 20BBB1			
Утверд.		Митрохин М.А.		31.05.24					

Содержание

Перечень сокращений и обозначений	3
Введение.....	7
1 Электронный замок на основе микроконтроллера STM32.....	8
1.1 Анализ предметной области	8
1.1.1 Обзор аналогов	9
1.1.2 Анализ требований к разработке	11
1.1.3 Выбор инструментов разработки программы для STM32.....	14
1.1.4 Выбор инструментов разработки серверной части	19
1.1.5 Выбор СУБД.....	25
1.1.6 Выбор архитектуры приложения и протокола взаимодействия	29
1.2 Проектирование и разработка.....	34
1.2.1 Реализация серверной части	39
1.2.2 Проектирование и создание базы данных	47
1.2.3 Реализация клиента STM32.....	48
1.3 Анализ результатов	52
1.3.1 Тестирование серверной части.	54
1.3.2 Тестирование микроконтроллера.	56
2 Охрана труда.....	58
2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ	58
2.2 Классификация и категорирование помещения, в котором производится разработка	58
2.2.1 Классификация по взрывопожарной и пожарной опасности	58
2.2.2 Классификация по опасности поражения электрическим током	60
2.3 Санитарно-гигиеническая характеристика производства	61
Заключение	63
Список используемых источников	64

Приложение А – UML-диаграммы.....	65
Приложение Б – Рисунки схем	74
Приложение В – Листинг программы сервера.....	77
Приложение Г – Презентация.....	94

					ПГУ 1.090301.05.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения

БД – база данных

ОСРВ – операционная система реального времени

ПО – программное обеспечение

ПЭВМ – персональная электронно-вычислительная машина

СКУД – система контроля и управления доступом

СУБД – система управления базами данных

ЭВМ – электронно-вычислительная машин

API (Application Programming Interface) – программный интерфейс приложения

ARM (Advanced RISC Machine) – усовершенствованная RISC-машина

CI (Continuous Integration) – непрерывная интеграция

DI (Dependency Inversion) – инверсия зависимостей

DTO (Data Transfer Object) – объект передачи данных

GPIO (general-purpose input/output, GPIO) – интерфейс ввода/вывода общего назначения

HAL (Hardware abstraction layer) – слой аппаратных абстракций

HTML (Hyper Text Markup Language) – язык гипертекстовой разметки

IDE (Integrated Development Environment) – интегрированная среда разработки

IoT (Internet of Things) – интернет вещей

MQTT (message queuing telemetry transport) – транспорт телеметрии очереди сообщений

NFC (Near-field communication) – связь ближнего радиуса действия

NoSQL (not only SQL) – не только SQL

ORM (Object-Relational Mapping) – объектно-реляционное отображение, или преобразование

QoS (Quality of Service) – качество обслуживания

RFID (Radio Frequency Identification) – радиочастотная идентификация

					ПГУ 1.090301.05.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

SQL (Structured Query Language) – язык структурированных запросов

UART (universal asynchronous receiver-transmitter) – универсальный асинхронный приёмо-передатчик

UML (Unified Modeling Language) – унифицирован язык моделирования.

					ПГУ 1.090301.05.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Безопасность всегда являлась важной частью любой сферы жизни, начиная от частной жизни отдельного человека и заканчивая крупными производственными корпорациями. Обеспечение безопасности может осуществляться по-разному и в разных сферах деятельности, например, охрана товаров в магазине от их непосредственной кражи, охрана в общественных местах и т.д.

С развитием технологий средства для обеспечения безопасности также начали совершенствоваться, применяя все новые разработки, такие как металлоискатели, камеры видеонаблюдения, искусственный интеллект для распознавания лиц, одежды и даже поведения.

Одной из самых главных частей в пресечении каких-либо несанкционированных действий является ограничение доступа на территорию или же в помещение посторонних лиц. Механические ключи использовать на больших территориях нецелесообразно и дорогостояще. Решение данной проблемы также возможно найти в автоматизировании процесса контроля доступа, тем самым усилив меры предосторожности и снизив расходы на персонал.

Микроконтроллеры STM32 от компании «STMicroelectronics» являются самым часто применяемым решением при автоматизации большинства процессов. Данные микроконтроллеры имеют ряд преимуществ: высокая производительность, широкий спектр моделей и периферийных модулей, малое электропотребления, низкая стоимость и др.

В данной работе была поставлена цель создать умный электронный замок на основе микроконтроллера STM32. Актуальность темы обусловлена нехваткой данных решений по контролю доступа на рынке в большинстве регионов, в частности в Пензенской области.

1 Электронный замок на основе микроконтроллера STM32

1.1 Анализ предметной области

На данный момент практически все частные компании, муниципальные заведения и другие организации начинают внедрять электронные замки и системы управления и контроля доступа, так как механические ключи перестали соответствовать требованиям безопасности. Также делается это с целью сокращения затрат на персонал по охране территории или помещения. Первоочередным клиентом, нуждающимся в разрабатываемом решении, является пензенская компания «Максофт», которая и является заказчиком подобной системы. В дальнейшем планируется интеграция электронных замков и в другие пензенские организации и выход на глобальный рынок.

Возможными клиентами данных систем могут являться и физические лица, которые хотят иметь удобный и инновационный функционал для оборудования своего дома.

Разрабатываемая система предлагает ряд преимуществ, делающих ее перспективной альтернативой:

- повышенная безопасность;
- удобство в использовании;
- простота управления;
- возможность масштабируемости;
- широкое кол-во возможностей;
- более низкая стоимость в сравнении с содержанием персонала.

Умные электронные замки со считывателем RFID имеют более высокий уровень безопасности чем другие средства за счет трудоемкого взлома самого устройства и повышенной трудности изготовления поддельных ключей. Помимо усиления безопасности резко увеличивается количество возможностей и соответственно увеличивается удобство использования. В системе, основанной на микроконтроллере STM32 имеется функционал для

					ПГУ 1.090301.05.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

расширения, путем добавления модулей wifi, bluetooth и др., таким образом обеспечивая возможность упрощения взаимодействия с системой разными способами в виде мобильных приложений и сайтов. Так модуль wifi можно использовать для установления соединения с сервером, отправкой данных о всех событиях и получения настроек, служебной информации, обновлений.

В сравнении с другими вариантами обеспечения безопасности, установление систем на базе микроконтроллеров фирмы «STMicroelectronics» обеспечивает минимальное количество затрат.

1.1.1 Обзор аналогов

С развитием отрасли информационных технологий системы контроля доступа и электронные замки начали разрабатываться и применяться повсеместно. На рынке присутствует большое количество систем по контролю и управлению доступом и электронных замков. Каждая система имеет как ряд своих преимуществ, так и ряд недостатков. Основным, самым масштабным и технологичным в России решением является разработка компании «PERCo»: система контроля доступа PERCo-Web предназначена для усиления безопасности и повышения дисциплины труда персонала. СКУД PERCo-Web позволяет организовать защиту от доступа посторонних, разграничение прав доступа сотрудников и посетителей и учет рабочего времени сотрудников. В качестве идентификаторов в системе контроля и управления доступом используются биометрические данные, карты доступа с защитой от копирования и смартфоны с NFC-модулем.

Данный продукт имеет сразу несколько интерфейсов взаимодействия с пользователем, полный контроль над системой из обширного интерфейса, предназначенного для администратора, большое кол-во внешних модулей и возможность интеграции в разные системы.

Разработка компании «PERCo» несмотря на большой ряд преимуществ, имеет также ряд недостатков таких как сложность в управлении системой и

					ПГУ 1.090301.05.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

дорогая стоимость интеграции технологии, что не может быть приемлемым решением для небольших компаний и частных территорий или помещений.

Подобными решениями занимались также и крупные зарубежные корпорации: «Google» (doorlock google home), «Amazon» (smart door lock), «Samsung», «Xiaomi» и другие. Все разработанные решения имеют одинаковый ряд недостатков: ненадежность, неспособность к масштабированию, изготовление их массово без интеграции и поддержки в определенные предприятия. Краткий анализ сильных и слабых сторон систем, разработанных крупными корпорациями представлен в таблице 1.

Таблица 1 - Обзор аналогов

Конкурент	Сильная сторона	Слабая сторона
Google	Известный бренд, широкий ассортимент продукции, высокое качество	Высокая цена
Samsung	Инновационные технологии, удобство использования	Сложность установки
Xiaomi	Доступная цена, простота использования	Ограниченный функционал
Aqara	Широкий ассортимент продукции, интеграция с экосистемой Apple HomeKit	Невысокая надежность
Nuki	Красивый дизайн, простота установки	Небольшая распространенность на рынке

На основании проведенного анализа можно сделать вывод, что на рынке умных электронных замков существует большой потенциал для разработки новых продуктов. Разрабатываемая система умного электронного замка на

базе микроконтроллера STM32 имеет все шансы стать конкурентоспособной на рынке.

1.1.2 Анализ требований к разработке

Заказчиком была поставлена задача разработки универсального умного электронного замка состоящего из устройства управления на основе микроконтроллера STM32 и клиент-серверного приложения на базе mqtt протокола, как для крупных промышленных предприятий, так и для небольших территорий.

Целью проекта является создание прототипа по требованиям предъявляемым компанией со следующим первоначальным функционалом:

- считывание метки;
- сравнение считанной метки с метками, сохраненными в базе данных;
- управление смарт-реле (открытие/закрытие);
- отправка события на сервер;
- сохранение события в базу данных;

Плата для разработки устройства была предоставлена компанией-заказчиком.

Под текущий функционал был определен следующий стек технологий: C, HAL, FREERtos, STM32CubeIDE, MQTT, JS, NodeJs, NestJs, Sequelize, Postgres, docker, WEBStorm, DBViewer, Git, Linux Ubuntu.

Выбор технологий был обусловлен требуемыми определенными достоинствами перед другими системами:

- производительность;
- офлайн режим;
- использование современных технологий реализации;
- возможность расширения и добавления новых модулей;
- сохранение истории на внешнем сервере;
- универсальность использования;
- низкое энергопотребление.

Микроконтроллеры STM32 от «STMicroelectronics» обладают широким спектром функций, которые делают их универсальным решением для разработки встраиваемых систем. STM32 основаны на архитектуре ARM Cortex-M [1], которая обеспечивает высокую производительность, низкое энергопотребление и широкий спектр периферийных устройств. ARM Cortex-M представляет собой семейство 32-х разрядных RISC-процессоров (компьютеров с сокращенным набором команд), оптимизированных для встраиваемых систем.

Структура ядра Cortex-M:

- ядро процессора, которые выполняют инструкции программы;
- регистры, они хранят временные данные, используемые ядром процессора;
- блок управления памятью (MMU - Memory Management Unit), обеспечивающий управление виртуальной памятью;
- блок обработки прерываний (NVIC - Nested Vectored Interrupt Controller), который обрабатывает запросы прерываний от периферийных устройств;
- шина, обеспечивающая передачу данных между различными компонентами ядра.

Выбор данной платы обусловлен дальнейшими расширениями проекта.

Модуль для связи и установки соединения – ESP32. ESP32 - это универсальный модуль Wi-Fi, который подходит для широкого спектра задач [2]. ESP32 может подключаться к Wi-Fi, Bluetooth и другим сетям. Контроллер имеет поддержку большинства сетей Wi-Fi работающих на частоте 2.4 ГГц и 5 ГГц. Так как система будет интегрироваться в какую-либо организацию, контроллер был специально подобран с требованиями поддержки большинства стандартов сети Wi-Fi, для лучшей интеграции.

Диаграмма компонентов контроллера показана на рисунке 1.



Рисунок 1 - Диаграмма компонентов контроллера

Микроконтроллер stm32h753vit6 обладает повышенной производительностью, что необходимо для дальнейшего расширения системы и добавления новых модулей. Данный контроллер имеет ядро Cortex-M7 работающее на частоте 480МГц. Устройства STM32H753xI имеют высокоскоростную встроенную память с двухбанковой флэш-памятью объемом 2 Мбайт и до 1 Мбайт оперативной памяти.

Весь процесс разработки будет производится на операционной системе Linux Ubuntu 22.04. Linux – это семейство Unix-подобных операционных систем с открытым исходным кодом, основанное на ядре Linux. Ubuntu – это популярный дистрибутив Linux, простой в использовании и предлагающий одно из самых больших сообществ пользователей. Выбор в пользу данной системы обусловлен множеством факторов. Linux более устойчив к вирусам и вредоносным программам, что повышает надежность разработки. Данная операционная система легко изменяется, ее можно настроить под потребности текущего проекта. Также Linux Ubuntu является бесплатной системой с открытым исходным кодом, что и стало решающим фактором в выборе. Другие операционные системы с должным инструментарием для разработки, такие как Windows и MacOS ограничили доступ к своей покупке.

Для управления версиями приложения и удобной дальнейшей совместной разработки используется Git с облачным хранилищем предоставляемым компанией «Максофт» – GitLab.

1.1.3 Выбор инструментов разработки программы для STM32

В качестве языка реализации проекта был выбран язык Си с использованием библиотеки HAL и оперативной системы реального времени FreeRTOS.

Язык C появился значительно раньше и изначально использовался для системного программирования [3]. Хотя существуют и другие языки программирования, пригодные для разработки на микроконтроллерах, Си остается практически единственным решением по ряду причин:

– Си обеспечивает прямой доступ к аппаратным ресурсам микроконтроллеров, позволяя оптимизировать код для достижения максимальной производительности;

– Си-код компилируется непосредственно в машинный код, который может быть выполнен процессором микроконтроллера без интерпретатора, что обеспечивает быстрое и эффективное выполнение;

– Си обладает полной экосистемой, включающей в себя компиляторы, библиотеки, инструменты разработки и большое сообщество пользователей;

– производители микроконтроллеров, включая STMicroelectronics, Microchip, Atmel, предоставляют обширные ресурсы и документацию по разработке на C для своих устройств.

HAL (Hardware Abstraction Layer) – это библиотека с открытым исходным кодом, разработанная компанией «STMicroelectronics» в 2014 году для упрощения разработки программного обеспечения (ПО) для микроконтроллеров STM32. HAL абстрагирует работу с низкоуровневыми регистрами и аппаратными ресурсами. Библиотека сама выполняет инициализацию периферийных устройств и для дальнейшей работы с ними предоставляет удобный интерфейс в виде готовых функций чтения и записи. Также HAL предоставляет функции для обработки прерываний от периферийных устройств и функции работы с питанием. Но для дальнейшей возможности расширения системы, добавления нового функционала, удобной читаемости кода и стабильности в разработке также необходима операционная система для STM32.

Операционные системы реального времени (ОСРВ) используются во многих устройствах и системах, где критична быстрая и точная реакция на события в реальном времени. Именно таким устройством и является электронный замок.

ОСРВ обеспечивают:

- предсказуемость, так как задачи выполняются в гарантированные временные интервалы;
- детерминированность, так как поведение системы не зависит от случайных факторов;
- быстрое время реакции, система мгновенно реагирует на внешние события;
- надежность, система устойчива к сбоям и обеспечивает бесперебойную работу;
- структурированность, система сама распределяет задачи, разработчикам легче сопровождать код и работать в команде.

Операционные системы реального времени (ОСРВ) чаще всего используются для процессоров ARM и Cortex. Это связано с их распространенностью во встраиваемых системах. Но среди ОСРВ с открытым исходным кодом лидирует бесплатная система FreeRTOS. Существует также коммерческая версия FreeRTOS под названием SafeRTOS. FreeRTOS обеспечивает высокую производительность так как код системы написан на ассемблере. Она соответствует стандарту безопасности IEC 61508.

FreeRTOS включает в себя инструменты для обеспечения многозадачности и распределение выполнения задач по приоритету. Также в FreeRTOS можно обеспечить правильную синхронизацию задач и распределение памяти. Основные компоненты для работы с задачами и памятью в FreeRTOS: ядро, задачи, очереди, семафоры, мьютексы и таймеры. При разработке текущего решения данных инструментов будет достаточно, но в дальнейшем при расширении имеется возможность увеличить функционал FreeRTOS использованием множества сторонних библиотек с открытым исходным.

Основными компонентами операционной системы FreeRTOS являются [4]:

- задачи;
- события;
- мьютексы;
- семафоры;
- очереди событий;
- таймеры;
- ядро FreeRTOS.

Для написания кода, сборки и отладки проекта необходимо выбрать интегрированную среду разработки (IDE). IDE включает в себя компоненты для удобной разработки программного обеспечения. Она обеспечивает комплексный подход к разработке, упрощая и оптимизируя процесс создания программ. В основные функции IDE входит:

- редактор кода, он позволяет писать, редактировать и форматировать код;
- компилятор и интерпретатор, который преобразует код в машинный код, который может быть выполнен устройством;
- отладчик, он позволяет выявлять и устранять ошибки в коде;
- инструменты сборки, которые автоматизируют процесс создания исполняемых файлов из исходного кода;
- средства управления версиями, позволяют отслеживать изменения в коде и работать над проектом совместно с другими разработчиками;
- инструменты профилирования, они помогают оптимизировать код для повышения производительности.

При коммерческой разработке, написание кода, не используя IDE почти невозможно. С помощью множества инструментов среда разработки позволяет ускорить процесс написания кода в несколько раз, избегая

множество синтаксических ошибок, сохранить единый стиль написания программ и форматирования кода.

В качестве среды разработки управляющей программы для микроконтроллера STM32 была выбрана бесплатная IDE с открытым исходным кодом, разработанная компанией «STMicroelectronics» – STM32CubeIDE.

STM32CubeIDE - обладает широким набором функций и преимуществ, которые делают ее отличным выбором. STM32CubeIDE включает в себя поддержку компилятора gcc, имеет встроенный отладчик позволяющий отлаживать код непосредственно в IDE [5]. Кроме среды разработки у компании «STMicroelectronics» имеется инструмент для простого конфигурирования и инициализации проектов под платформу STM32 - STM32CubeMX. Он позволяет разработчикам быстро и легко настроить периферийные устройства, часы, GPIO и другие функции микроконтроллера, не прибегая к написанию кода вручную. С данным программным обеспечением имеется интеграция у среды разработки STM32CubeIDE, что является преимуществом. Сравнение аналогов STM32CubeIDE представлено в таблице 2.

Таблица 2 - Обзор IDE

IDE	Недостаток
Keil MDK от Arm Ltd	Платная IDE
Eclipse	Ограниченная поддержка STM32 Сложность освоения
CoDeSys	Платная Ограниченная поддержка STM32

Таким образом для разработки управляющей программы для микроконтроллера STM32 был выбран следующий стек технологий:

- язык программирования Си
- библиотека HAL
- операционная система реального времени FREERtos
- интегрированная среда разработки STM32CubeIDE

1.1.4 Выбор инструментов разработки серверной части

Существует огромное множество инструментов разработки серверного программного обеспечения. Выбор фреймворка для разработки серверной части является критическим этапом в любом проекте. Необходимо тщательно взвесить все факторы, прежде чем принимать решение, так как правильный выбор инструмента может значительно повлиять на успех проекта. Приоритетом в выборе фреймворка являются удобочитаемость кода, гибкость в разработке и возможность расширения системы. Был проведен анализ основных фреймворков на разных языках программирования подходящие под данные требования и сравнение их с NestJS, чтобы обосновать выбор наиболее подходящего решения.

Первым возможным очевидным решением является фреймворк Django использующий python для написания кода. Основным явным перевес в пользу данного решения является низкий порог вхождения и простота в написании кода. Он подходит для разработки веб-приложений любого масштаба, имеет огромную кодовую базу с большим сообществом и поддержкой библиотек. Но проанализировав скорость работы, было определено, что язык программирования python значительно проигрывает по скорости среди других платформ. На рисунке 2 приведены результаты исследований по сравнению языка python и JavaScript с сайта «jelvix.com».

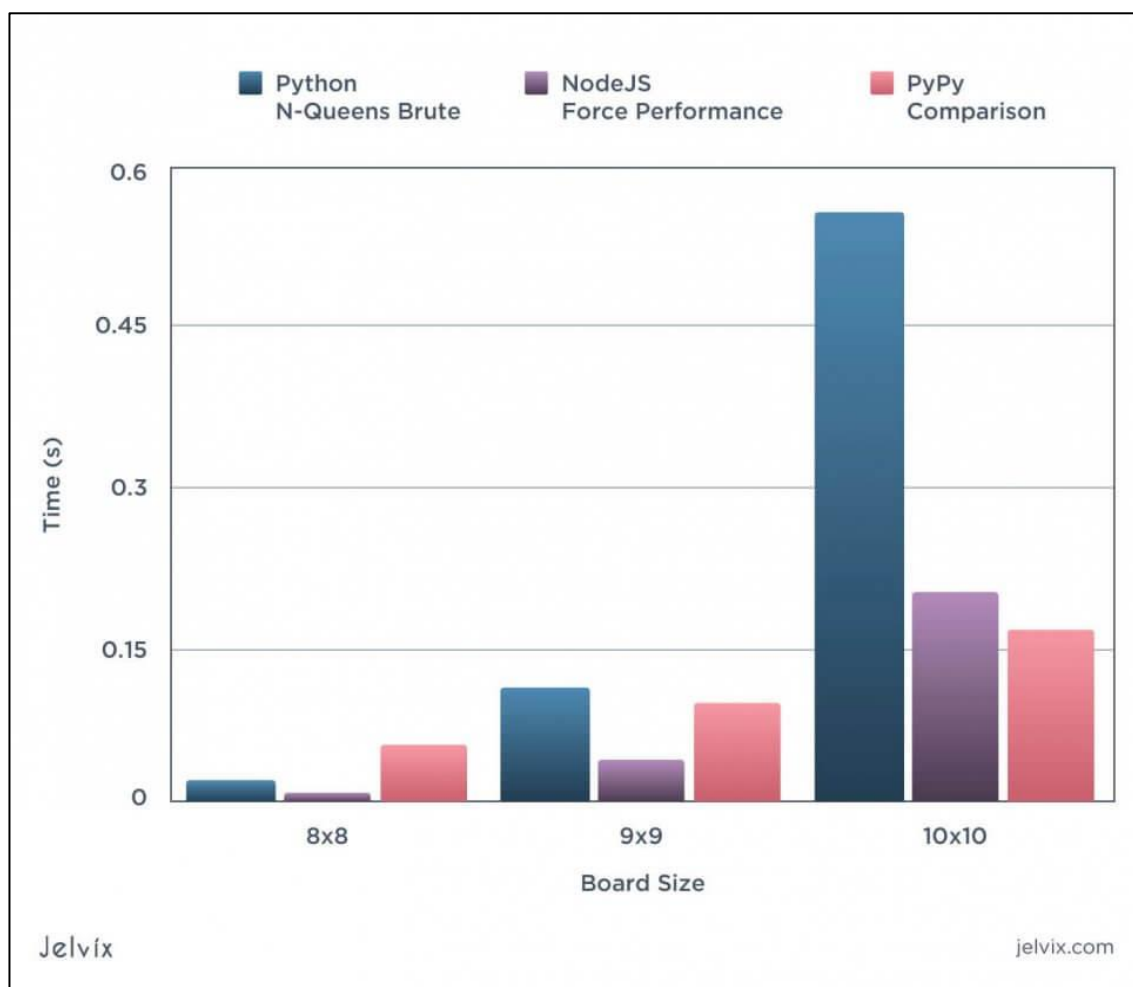


Рисунок 2 - Сравнение производительности

График показывает время обработки множественных запросов на сервер. Так как разрабатываемая система предполагает использование одновременно большого кол-ва устройств, например, при интеграции в учреждения с большой территорией, сервер обязан хорошо справляться решением подобных задач. Фреймворк Django на python не обеспечивает достаточной скорости работы для имеющихся требований к проекту. Также данный инструмент имеет мало средств для разработки приложения микросервисной архитектуры, что затрудняет дальнейшие возможности на расширение системы.

Следующим предполагаемым решением был рассмотрен фреймворк на java - SpringBoot. SpringBoot предназначен для корпоративной разработки, обеспечивает высокую надежность, имеет большую кодовую базу. При

детальном изучении SpringBoot был определен как достаточно сложный в освоении и поддержке кода. Фреймворк имеет многослойную архитектуру, включающую различные модули и абстракции, что может затруднить понимание его внутренней работы. Множество конфигурационных файлов: Для сложных приложений может потребоваться множество конфигурационных файлов, что усложняет их управление и поддержку. Из-за многослойной архитектуры и использования большого количества различных абстракций может быть сложно отслеживать и решать проблемы.

ASP.NET Core – достаточно известный фреймворк написанный на C# и поддерживаемый компанией Microsoft, также имеет ряд преимуществ и недостатков, но основной причиной отказа послужило его плохая интегрируемости с операционной системой Linux. Далее будет подробно описан данный выбор в пользу операционной системы Linux.

Таким образом компанией заказчиком разрабатываемой системы по предоставленному анализу был сделан выбор в пользу фреймворка NestJS работающем на платформе NodeJS по ряду причин:

- платформа NodeJS имеет возможность добавления аддонов (динамически связанные общие объекты, написанные на C++ и может загружать дополнения как обычные модули Node.js [6]);
- модульная архитектура;
- чистая архитектура, NestJS придерживается принципов чистой архитектуры, разделяя код на самостоятельные модули. Это повышает читаемость, упрощает разработку и поддержку сложных приложений, а также стимулирует повторное использование кода [7];
- инверсия управления зависимостями (DI), NestJS использует паттерн DI, который упрощает тестирование единичных компонентов приложения;
- TypeScript и статическая типизация, TypeScript, лежащий в основе NestJS, обеспечивает статическую типизацию, что позволяет выявлять ошибки на ранних стадиях разработки. Это повышает надежность кода и уменьшает

количество ошибок, с которыми можно столкнуться в процессе эксплуатации. Статическая типизация TypeScript делает код более понятным и читаемым, что упрощает его изучение и поддержку;

– производительность и масштабируемость, NestJS оптимизирован для максимальной производительности и эффективного использования ресурсов. Он использует современные технологии NodeJS и оптимизирован для масштабирования под увеличение нагрузки. Приложения, разработанные на NestJS, демонстрируют высокую скорость выполнения и низкую латентность. Это положительно влияет на пользовательский опыт и позволяет обрабатывать больше запросов в единицу времени;

– сообщество и поддержка, NestJS обладает активным и быстрорастущим сообществом разработчиков, которые всегда готовы оказать помощь и поддержку. На различных форумах и в чатах можно найти ответы на вопросы, получить советы и поделиться своим опытом;

– обширная документация. Доступна детальная и понятная документация, которая охватывает все аспекты разработки с помощью NestJS. Она включает в себя обучающие материалы, примеры кода и ссылки на дополнительные ресурсы. NestJS имеет развитую экосистему библиотек и инструментов, которые расширяют его функциональность и позволяют решать различные задачи;

– универсальность и гибкость. NestJS совместим с различными архитектурами приложений, включая монолитную, микросервисную и сервер безголовый (headless). Это позволяет разработчикам выбрать наиболее подходящий подход для своего проекта. NestJS подходит для разработки различных типов серверных приложений, включая API, веб-сайты, реальные приложения и игры. Он обеспечивает необходимую функциональность для реализации самых разных проектов.

На основании проведенного анализа можно сделать вывод, что NestJS обладает рядом преимуществ и перед другими популярными фреймворками

					ПГУ 1.090301.05.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

написанными на javascript, такими как ExpressJS, KoaJS, SailsJS и HapiJS. NestJS зарекомендовал себя как надежный, эффективный и универсальный фреймворк.

ORM (Object-Relational Mapping) – это паттерн программирования и программное обеспечение, которое упрощает взаимодействие между объектно-ориентированными приложениями и базами данных. ORM позволяет уменьшить количество кода, необходимого для работы с базами данных, делая код более читаемым, понятным и поддерживаемым. ORM может повысить производительность приложения за счет оптимизации запросов, кэширования и использования асинхронных операций. Также данный паттерн программирования улучшает читаемость кода и его модульность, за счет отделения логики работы базы данных в независимый модуль. Тестирование приложения при использовании ORM становится проще и быстрее сокращая время разработки за счет быстрой и понятной отладки. В совокупности все вышеперечисленные факторы позволяют значительно сократить время разработки и сосредоточиться на написании бизнес-логики приложения. По последним соображениям также было решено взять одну из готовых ORM предоставленных фреймворком NestJS - Sequelize.

Sequelize – это универсальный ORM, который хорошо подходит для различных типов приложений и баз данных. Sequelize позволяет создавать модели данных, которые представляют собой объектные представления таблиц базы данных. Это упрощает работу с данными, делая код более читаемым, поддерживаемым и масштабируемым. При помощи Sequelize легко осуществлять миграции баз данных, которые позволяют изменять их структуру с течением времени без потери информации.

Сравнение Sequelize с другими ORM:

– TypeORM более сложен в освоении и использовании, чем Sequelize, может быть менее производительным, чем Sequelize, особенно при работе с

большими объемами данных, менее гибкий, чем Sequelize, и может не подходить для некоторых случаев использования;

– MikroORM более молодой проект, который еще не до конца отлажен, из-за чего имеет меньшее сообщество, чем Sequelize, что может затруднить поиск помощи, имеет меньшую поддержку со стороны библиотек и инструментов, чем Sequelize.

Для разработки кода серверной части было принято решение использовать WebStorm IDE на основе JavaScript от компании «JetBrains». Среда предлагает удобный и целенаправленный интерфейс, поддерживает системы контроля версий, такие как Git и Mercurial, имеет инструменты рефакторинга. А также она интуитивно понятна и проста в использовании и интегрируется со всеми популярными инструментами CI [8]. Также IDE имеет особые инструменты, не встроенные в другие похожие приложения. WebStorm автоматически подсказывает названия функций, переменных, модулей и других элементов кода, что позволяет писать код быстрее и с меньшим количеством ошибок. Она предлагает мощные инструменты для рефакторинга кода, которые позволяют изменять структуру кода без внесения изменений в его функциональность. WebStorm интегрирован с npm, что позволяет легко устанавливать и управлять пакетами NodeJS. А также выбор данной IDE обусловлен уже имеющимся опытом работы с данным программным обеспечением от компании «JetBrains». Обзор аналогов представлен в таблице 3.

Таблица 3 - Сравнение IDE

IDE	Преимущества	Недостатки
Visual Studio Code	Бесплатная, открытая, расширяемая	Менее функциональная, чем WebStorm
IntelliJ IDEA	Мощная, богатая	Платная для коммерческого

IDE	Преимущества	Недостатки
	функциональностью	использования
Sublime Text	Легкая, быстрая	Менее функциональная, чем WebStorm

Для отладки и дальнейшей работы приложения была выбрана такая технология как Docker. Docker – это платформа виртуализации с открытым исходным кодом, которая позволяет создавать, управлять и развертывать приложения в изолированных контейнерах. Изоляция позволяет одновременно запускать несколько версий одного приложения без конфликтов. Принцип работы Docker является подобным виртуальной машине, контейнеры Docker легковесны и занимают меньше места, чем виртуальные машины и контейнеры Docker легко развертывать и масштабировать. Также и виртуальные машины Docker позволяет разработчикам легко делиться образами контейнеров, что упрощает совместную работу. В целом, использование Docker предлагает множество преимуществ, таких как изоляция, легковесность, повторяемость, простота развертывания, портативность, экономия средств, улучшенное сотрудничество, широкий выбор инструментов, поддержка различных языков программирования, открытость и прозрачность.

1.1.5 Выбор СУБД

Выбор СУБД (системы управления базами данных) является критическим этапом в процессе разработки. Правильный выбор СУБД оказывает влияние на производительность, функциональность, масштабируемость и безопасность приложения. Существует несколько типов баз данных: реляционные, NoSQL, гибридные базы данных, временные базы данных, геопространственные базы данных, графовые базы данных. В процессе выбора было рассмотрено две основные категории: реляционные базы данных и NoSQL.

Реляционные базы данных - хранят данные в виде таблиц, где каждая таблица состоит из строк и столбцов, связи между таблицами устанавливаются с помощью первичных и внешних ключей. Такие базы данных подходят для хранения структурированных данных и аналитических запросов к ним, например, поиск информации о конкретном пользователе системы или товаре в случае интернет магазинов. Примерами таких баз являются: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.

NoSQL базы данных - не имеют фиксированной схемы данных, что делает их более гибкими, они хранят данные в различных форматах, таких как JSON, документы или графы. NoSQL базы данных подходят для неструктурированных данных, таких как изображения, видео, тексты и используются в системах где объем данных может быстро меняться. Примерами таких баз являются: MongoDB, Cassandra, CouchDB, RavenDB.

Так как данные в текущем проекте представляют собой структурированную информацию и не подразумевают хранение больших медиа файлов с часто меняющимся размером, целесообразно выбор на использование был сделан в пользу реляционных баз данных. Работа с такими базами подразумевает большую производительность, что не менее важно в разрабатываемой системе. Изучение работы с реляционными базами данных, проектирование их схем и написание кода приложения также будет производиться намного быстрее, что позволит ускорить процесс разработки.

Анализ реляционных баз данных представлен в таблице 4.

Таблица 4 - Анализ баз данных

Название базы данных	Преимущество	Недостаток
MySQL	Бесплатная и открытая, простота использования, распространенность	Ограниченная функциональность, слабая безопасность

Название базы данных	Преимущество	Недостаток
PostgreSQL	Надежность, безопасность	Сложность, расширенные возможности, масштабируемость
Microsoft SQL Server	широкий спектр функций, высокая производительность, надежность и безопасность	Стоимость, закрытый код, сложность
Oracle Database	Масштабируемость, надежность и безопасность	Стоимость, сложность, жесткость схемы

Базы данных от «Microsoft» и «Oracle» предлагают отличные решения для разрабатываемой системы, но в текущих реалиях использование этих баз данных не представляется возможным, так как компании разработчики ограничили доступ к покупке. База данных MySQL является слабо безопасной и предлагает недостаточно функционала для дальнейшего расширения системы.

Таким образом единственным подходящим и доступным вариантом для проекта становится база данных PostgreSQL.

PostgreSQL – это мощная, надежная и бесплатная реляционная СУБД (система управления базами данных), которая пользуется большой популярностью среди разработчиков, администраторов баз данных и компаний. PostgreSQL создана на основе некоммерческой СУБД Postgres, разработанной как open-source проект в калифорнийском университете в Беркли [9]. Она существует в реализациях для множества UNIX-подобных платформ, включая AIX, различные BSD-системы, HP-UX, IRIX, Linux, macOS, Solaris/OpenSolaris, Tru64, QNX, а также для Microsoft Windows.

Преимущества использования PostgreSQL:

- бесплатность и открытость: PostgreSQL доступна бесплатно для всех, что делает ее доступной для проектов с ограниченным бюджетом;
- надежность. PostgreSQL известна своей надежностью и низким риском сбоев;
- безопасность. PostgreSQL обеспечивает высокий уровень безопасности, что защищает ваши данные от несанкционированного доступа;
- производительность. PostgreSQL демонстрирует высокую производительность, даже при работе с большими объемами данных и высокой нагрузкой;
- масштабируемость. PostgreSQL легко масштабируется путем добавления дополнительных серверов, что позволяет ей расти вместе с вашими потребностями;
- расширенные возможности. PostgreSQL поддерживает широкий спектр функций, что делает ее универсальной для различных типов приложений;
- сообщество. PostgreSQL имеет большое и активное сообщество разработчиков.

Помимо IDE необходимо программное обеспечение для удобной работы с базой данных. Таким инструментом может послужить программа DBeaver. DBeaver – это отличный инструмент, который позволяет легко и эффективно работать с базами данных. Он поддерживает широкий спектр СУБД, имеет простой в использовании интерфейс и предоставляет множество функций, которые позволяют управлять своими данными, просматривать их, изменять и т.д. Выбор подобного решения не имеет особого значения для дальнейшего развития проекта и не влияет на работоспособность системы, поэтому данное программное обеспечение было выбрано по личному усмотрению и в любой момент подлежит замене без каких-либо явных трудностей.

1.1.6 Выбор архитектуры приложения и протокола взаимодействия

Архитектура всего приложения была определена выбором сделанных технологий. Использование фреймворка NestJS предполагает использование модульной архитектуры приложения. Модульная архитектура – это метод проектирования программного обеспечения, который предполагает деление приложения на отдельные, независимые модули. Каждый модуль выполняет определенную функцию и имеет свой собственный интерфейс. Модули взаимодействуют друг с другом через свои интерфейсы, что позволяет создавать слабосвязанные и легко тестируемые приложения. Модульная архитектура приложения позволяет повторно использовать код в разных частях приложения, тем самым уменьшая дублирование кода. В дальнейшем, при расширении приложения, предполагается добавление микросервисной архитектуры, для взаимодействия разных частей приложения.

Использование в микроконтроллере операционной системы реального времени FreeRTOS обязывает разделить все приложение на задачи, каждая из которых выполняет определенную функцию. Задачи могут быть простыми или сложными, периодическими и непериодическими. Каждой задаче назначается приоритет, который определяет очередность ее выполнения. Задачи могут нуждаться в доступе к общим ресурсам, таким как переменные или периферийные устройства. Для синхронизации доступа к общим ресурсам используются мьютексы, семафоры и другие механизмы синхронизации. Задачи могут обмениваться данными друг с другом через очереди сообщений, события и другие механизмы связи.

MQTT (Message Queuing Telemetry Transport) – это легковесный протокол обмена сообщениями, специально разработанный для использования в приложениях Интернета вещей (IoT) [10]. Он позволяет устройствам подключаться к серверам и обмениваться данными эффективно и надежно. MQTT использует минимальный объем данных, что делает его идеальным для устройств с ограниченными ресурсами, таких как

					ПГУ 1.090301.05.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

микроконтроллеры STM32. Его использование позволяет уменьшить нагрузку на сеть и снизить энергопотребление. MQTT поддерживает несколько уровней качества обслуживания (QoS), что гарантирует доставку сообщений. Это важно для умных замков, где недоставка сообщений может привести к задержке прохода людей и создания очередей. MQTT прост в реализации и использовании и существует множество библиотек и инструментов для разработки приложений MQTT, что упрощает процесс разработки. Данный протокол легко масштабируется и позволяет подключать к серверу множество устройств без многократного увеличения нагрузки. Для электронных замков необходимо быть частью большой сети устройств. В дальнейшем при развертывании проекта на сервере необходимо будет применять шифрование во время передачи данных, что вполне возможно при использовании данного протокола.

Протокол MQTT имеет ряд преимуществ для разрабатываемой системы перед другими протоколами. Сравнение с другими протоколами представлено в таблице 5.

Таблица 5 - Сравнение протоколов

Протокол	Описание	Преимущество	Недостаток
HTTP	Универсальный протокол передачи данных	Простой в использовании, широко распространен	Не оптимизирован для IoT, высокая нагрузка на сеть, низкая отказоустойчивость
CoAP	Легковесный протокол, основанный на HTTP	Экономит ресурсы устройства, низкая нагрузка на сеть	Менее распространен, чем MQTT
AMQP	Надежный, гибкий, масштабируемый протокол обмена	Обеспечивает гарантии доставки сообщений	Сложнее реализовать, чем MQTT

Протокол	Описание	Преимущество	Недостаток
	сообщениями		
WebSocket	Протокол постоянного двунаправленного соединения	Позволяет реализовать приложения IoT в режиме реального времени	Требует серверной поддержки WebSocket и затрат на производительность за счет постоянной поддержки соединения

Протокол MQTT требует дополнительное программное обеспечение при доставке сообщений называемое брокером. Таким образом данный протокол диктует общую архитектуру всей системы.

Ключевые компоненты MQTT:

- клиенты. Устройства IoT, которые подключаются к серверу MQTT и обмениваются сообщениями;
- серверы. Брокеры сообщений, которые перенаправляют сообщения между клиентами;
- темы. Категории, по которым группируются сообщения;
- сообщения. Данные, которые обмениваются между клиентами и серверами.

Последовательность работы компонентов протокола:

- клиент MQTT подключается к брокеру MQTT;
- клиент подписывается на темы, которые ему интересны;
- клиент публикует сообщение, он отправляет его на сервер вместе с темой;
- брокер перенаправляет сообщение всем клиентам, подписанным на эту тему;
- клиенты обрабатывают полученные сообщения;
- клиенты могут ответить, опубликовав новое сообщение.

Типы сообщений MQTT:

- publish, сообщения, которые отправляются клиентами на сервер;
- subscribe, сообщения, которые отправляются клиентами серверу, чтобы подписаться на темы;
- unsubscribe, сообщения, которые отправляются клиентами серверу, чтобы отписаться от тем;
- publishAck, сообщения, которые отправляются сервером клиентам, чтобы подтвердить получение сообщений;
- pubRec, сообщения, которые отправляются клиентами серверу, чтобы запросить подтверждение получения сообщения;
- pubRel, сообщения, которые отправляются сервером клиентам, чтобы подтвердить получение запроса на подтверждение;
- pubComp, сообщения, которые отправляются клиентами серверу, чтобы подтвердить получение подтверждения.

Уровни качества обслуживания (QoS) MQTT:

- QoS 0, сообщения не гарантируются и могут быть потеряны;
- QoS 1, сообщения гарантируются однажды;
- QoS 2, сообщения гарантируются дважды.

Визуальное представление архитектуры системы представлено на рисунке 3.

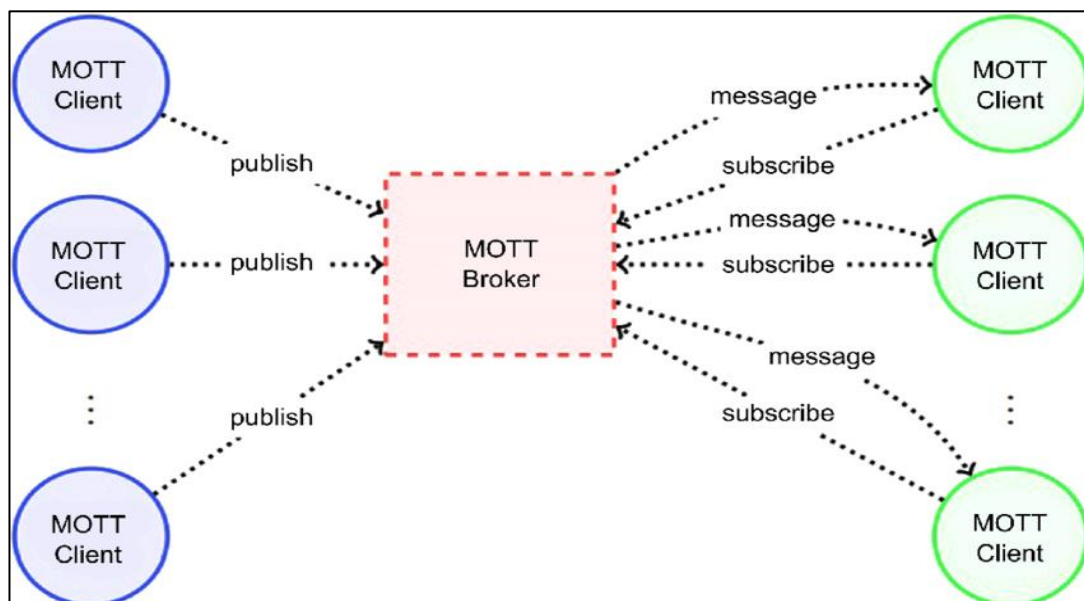


Рисунок 3 - Взаимодействие по протоколу MQTT

Брокер сообщений возможно реализовать самостоятельно, но в целях экономии времени разработки было решено использовать существующие уже отлаженные решения. Так как система в дальнейшем будет расширяться предполагается использование брокеров с открытым исходным кодом, для возможности увеличения функционала брокера.

Для первоначальной версии проекта был выбран брокер EMQX. EMQX -самый популярный брокер MQTT с 11,4 тысячами звёзд на GitHub, распространяемый по лицензии Apache 2.0. EMQX — самый масштабируемый в мире брокер MQTT на кластерной основе, который поддерживает расширяемость с помощью плагинов, использует протоколы MQTT 3.1, MQTT 5.0. В дальнейшем брокер может быть заменен, на аналоги (Mosquitto, NanoMQ, VerneMQ) без проблем изменения кодовой базы и архитектуры приложения.

Архитектура полной разрабатываемой микросервисной системы, с частями, не разрабатываемыми в данной работе представлена на рисунке 4.

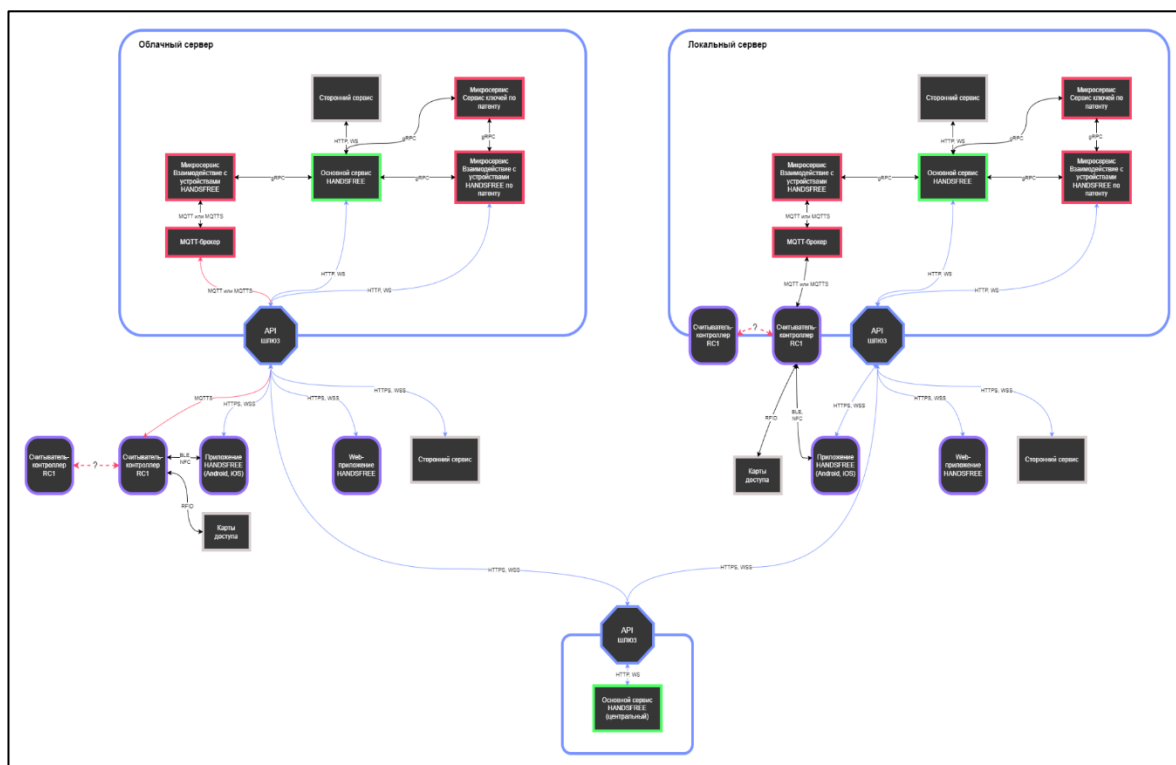


Рисунок 4 – Полная архитектура системы

1.2 Проектирование и разработка

Приложение состоит из сервера, работающего с базой данных и занимающимся обработкой запросов от клиентов, брокера передающего сообщения и клиентов, работающих на микроконтроллерах STM32. Серверное приложение построено на модульной архитектуре.

При разработке системы согласно требованиям, кодовая база разрабатывалась с возможностью расширения и дальнейшего сопровождения проекта в командной разработке. При командной разработке важно писать чистый код [11].

Диаграмма поведения приложения согласно требованиям, к разработке представлена на рисунке 5. Изначально была произведена установка всех компонентов разработки (рисунок 6 - 14) и при помощи docker был развернут MQTT брокер EMQX (рисунок 15). Docker - это платформа виртуализации,

которая позволяет упаковывать приложения вместе с их зависимостями в изолированные контейнеры [12].

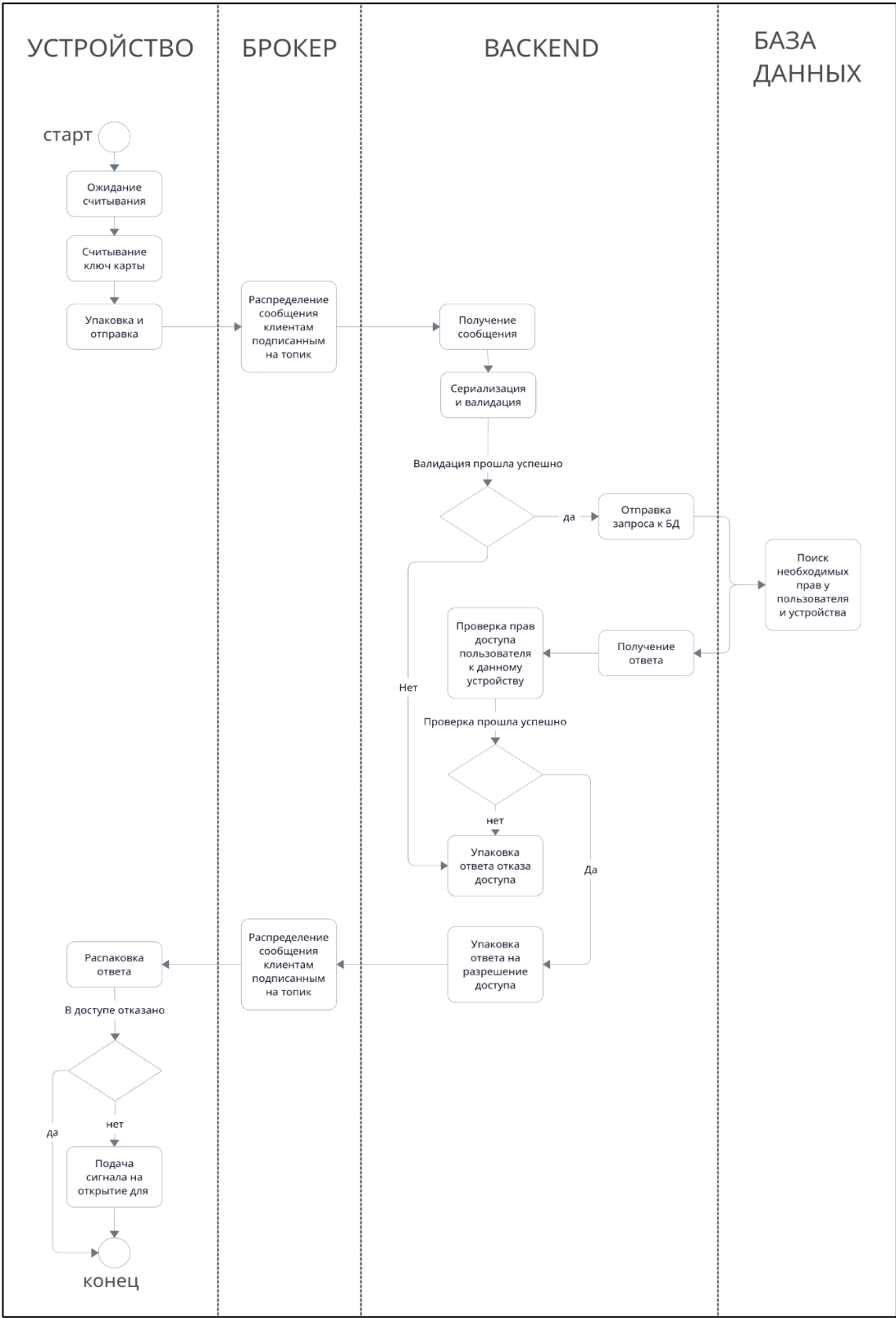


Рисунок 5 – UML-диаграмма поведения

```

root@89ce422bce2d:/# sudo apt-get install -y nodejs
sudo: setrlimit(RLIMIT_CORE): Operation not permitted
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib python2 python2-minimal python2.7 python2.7-minimal
Suggested packages:
  python2-doc python-tk python2.7-doc binutils binfmt-support
The following NEW packages will be installed:
  libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib nodejs python2 python2-minimal python2.7 python2.7-minimal
0 upgraded, 8 newly installed, 0 to remove and 5 not upgraded.
Need to get 21.9 MB of archives.
After this operation, 109 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 libpython2.7-minimal amd64 2.7.18-rc1-2 [335 kB]
Get:2 https://deb.nodesource.com/node_12.x focal/main amd64 nodejs amd64 12.18.2-deb-1nodesource1 [18.0 MB]
Get:3 http://archive.ubuntu.com/ubuntu focal/universe amd64 python2.7-minimal amd64 2.7.18-rc1-2 [1302 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal/universe amd64 python2-minimal amd64 2.7.17-2ubuntu4 [27.5 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal/universe amd64 libpython2.7-stdlib amd64 2.7.18-rc1-2 [1880 kB]

```

Рисунок 6 - Установка NodeJS

```

root@89ce422bce2d:/# node -v
v12.18.2
root@89ce422bce2d:/# npm -v
6.14.5
root@89ce422bce2d:/#

```

Рисунок 7 - Проверка установки NodeJS

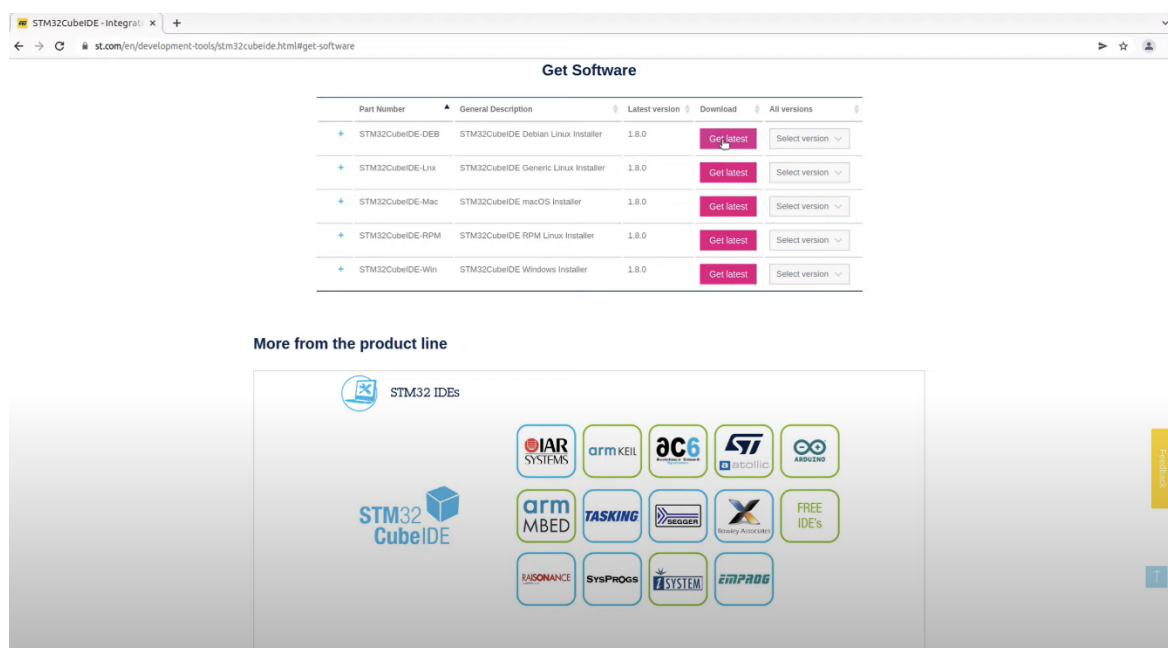


Рисунок 8 - Скачивание IDE

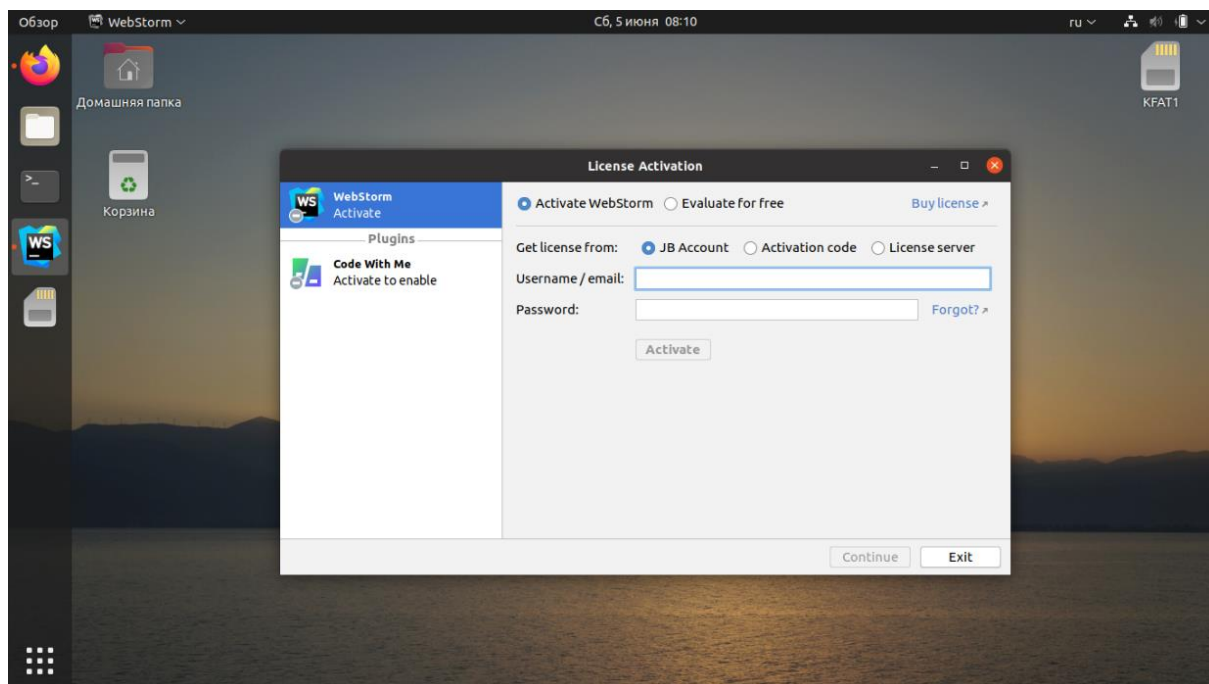


Рисунок 9 - Установка IDE

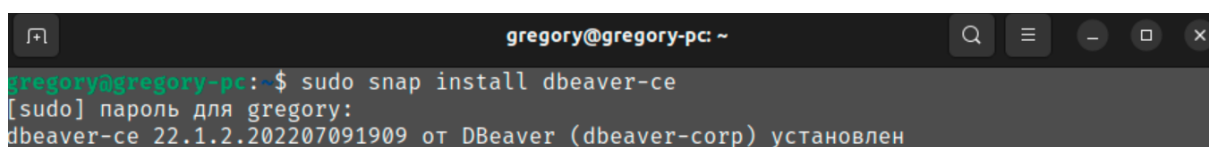


Рисунок 10 - Установка dbveawer

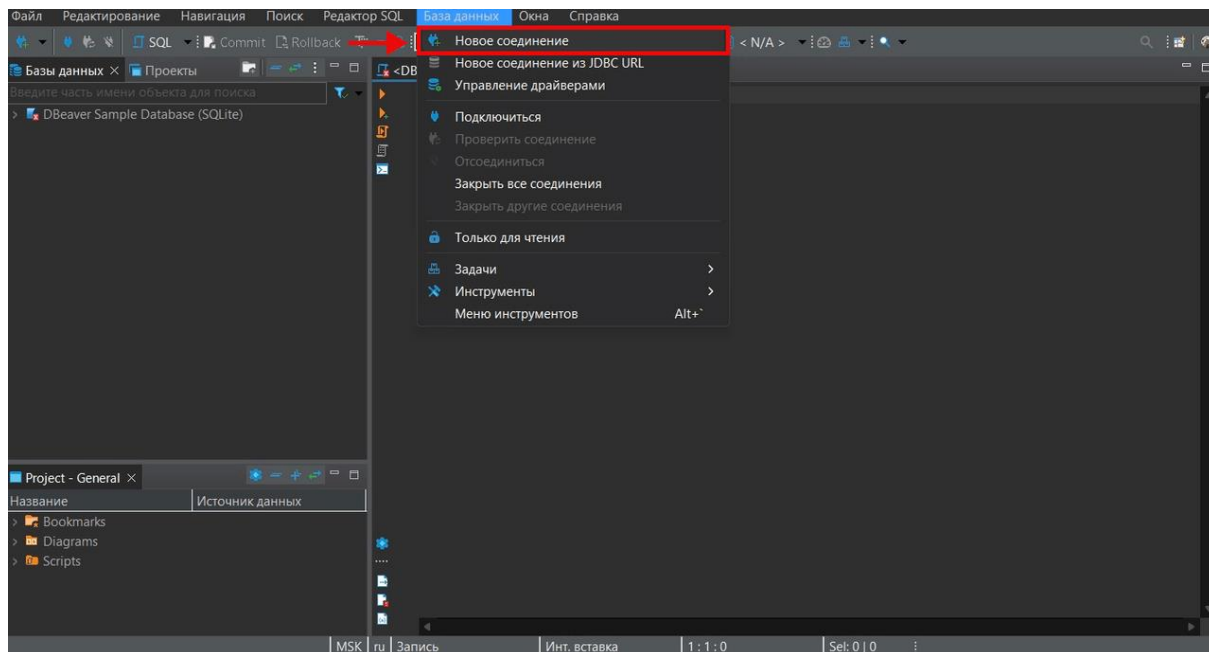


Рисунок 11 - Подключение к базе данных

Connect to a database

Connection Settings

PostgreSQL connection settings

PostgreSQL

Main PostgreSQL Driver properties SSH Proxy SSL

Server

Host: 10.92.6.36 Port: 5432

Database: books_store

Authentication

Authentication: Database Native

Username: postgres

Password: ☒ Save password locally

Advanced

Session role: Local Client: <not present>

ⓘ You can use variables in connection parameters. [Connection details \(name, type, ...\)](#)

Driver name: PostgreSQL [Edit Driver Settings](#)

Test Connection ... < Back Next > Cancel Finish

Рисунок 12 - Настройки подключения к базе данных

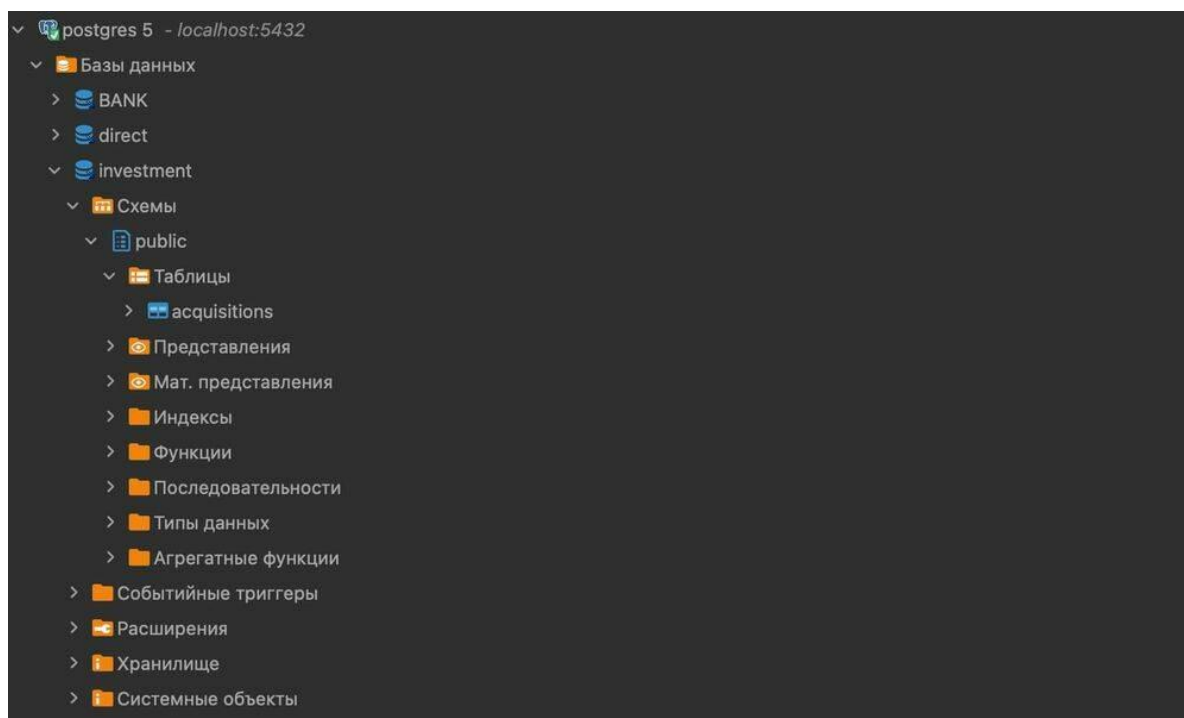


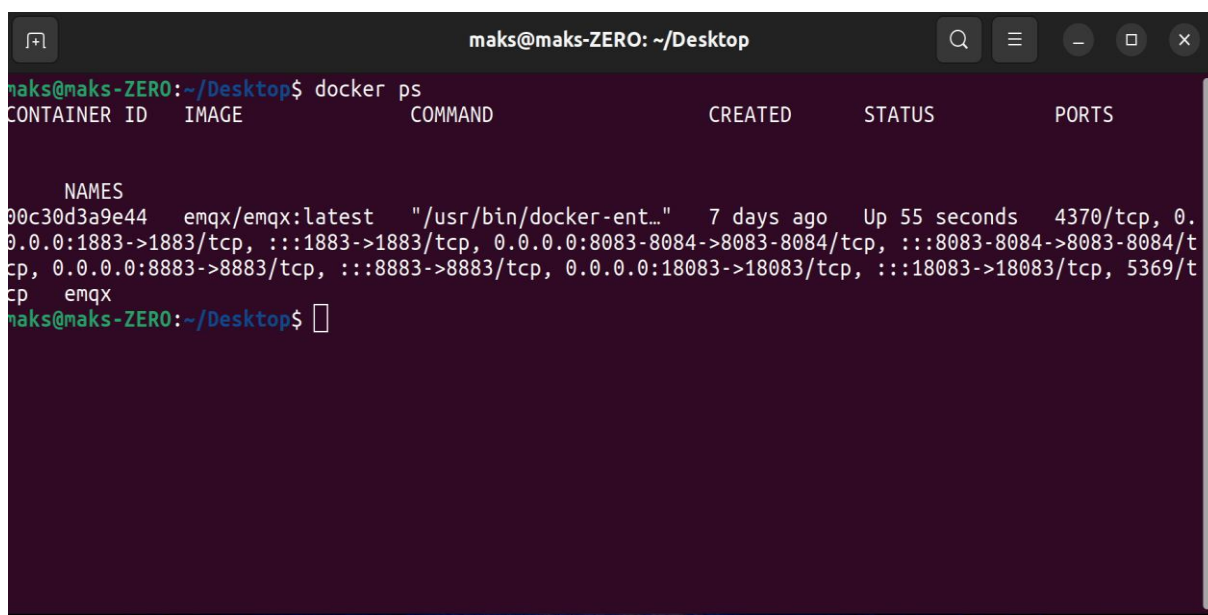
Рисунок 13 - Открытие базы данных

```

nick@U20Desktop:~$ cd Downloads/
nick@U20Desktop:~/Downloads$ ls
en.st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.deb_bundle.sh_v1.8.0
en.st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.deb_bundle.sh_v1.8.0.zip
nick@U20Desktop:~/Downloads$ cd en.st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.deb_bundle.sh_v1.8.0/
nick@U20Desktop:~/Downloads/en.st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.deb_bundle.sh_v1.8.0$ ls
st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.deb_bundle.sh
nick@U20Desktop:~/Downloads/en.st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.deb_bundle.sh_v1.8.0$ sudo ./st-stm32cubeide_1.8.0_11526_20211125_0815_amd64.deb_bundle.sh
[sudo] password for nick:
Creating directory makeself_dir_XQ0oyH
Verifying archive integrity...

```

Рисунок 14 - Установка IDE для STM32



```

maks@maks-ZERO: ~/Desktop
maks@maks-ZERO:~/Desktop$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
00c30d3a9e44   emqx/emqx:latest  "/usr/bin/docker-ent..."  7 days ago    Up 55 seconds  4370/tcp, 0.0.0.0:1883->1883/tcp, :::1883->1883/tcp, 0.0.0.0:8083-8084->8083-8084/tcp, :::8083-8084->8083-8084/tcp, 0.0.0.0:8883->8883/tcp, :::8883->8883/tcp, 0.0.0.0:18083->18083/tcp, :::18083->18083/tcp, 5369/tcp
NAME
00c30d3a9e44   emqx
maks@maks-ZERO:~/Desktop$

```

Рисунок 15 - Разворачивание брокера

1.2.1 Реализация серверной части

Архитектура основных классов программы представлена на рисунке 16.

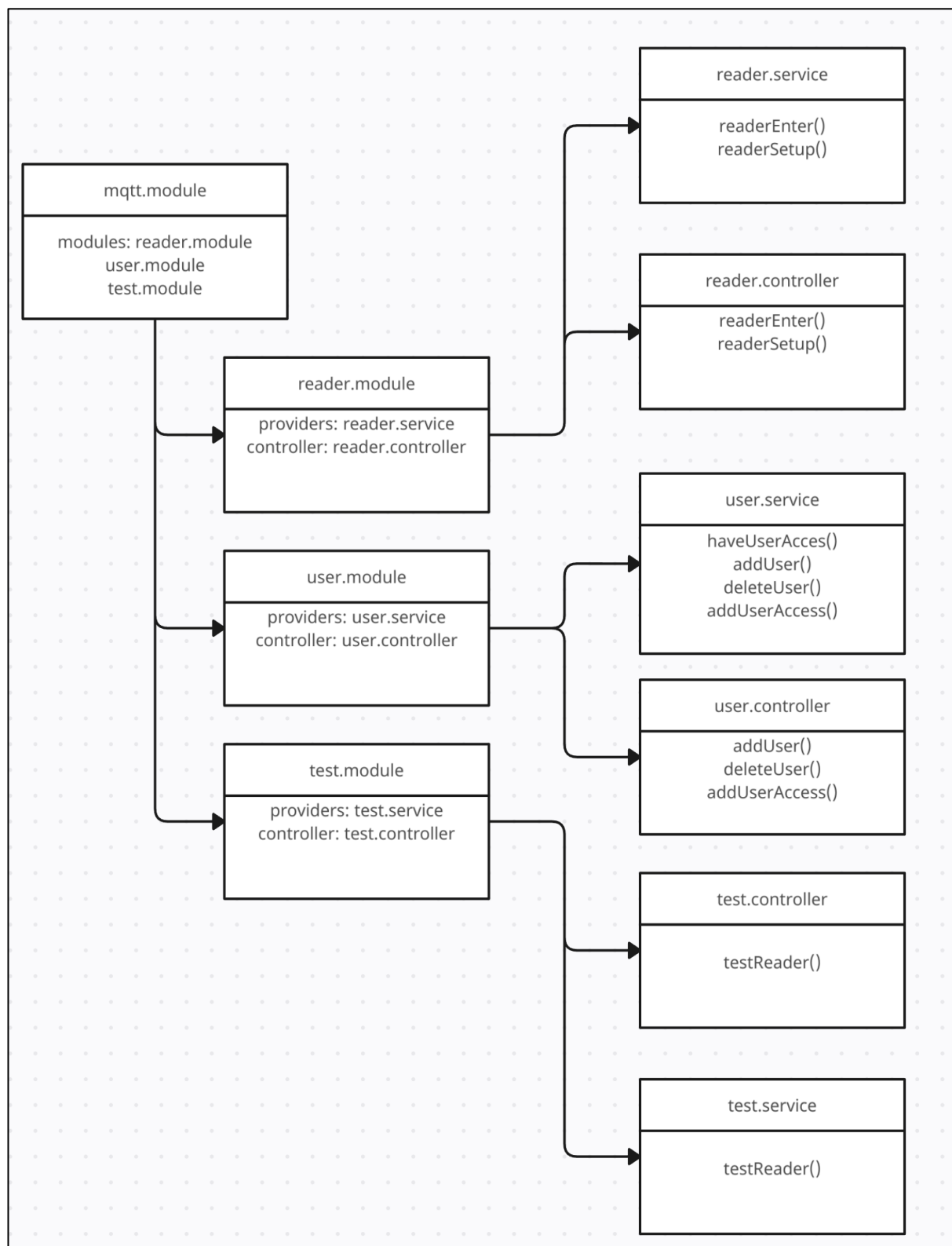


Рисунок 16 – UML-диаграмма основных классов

Полная диаграмма классов представлена на рисунке 17.

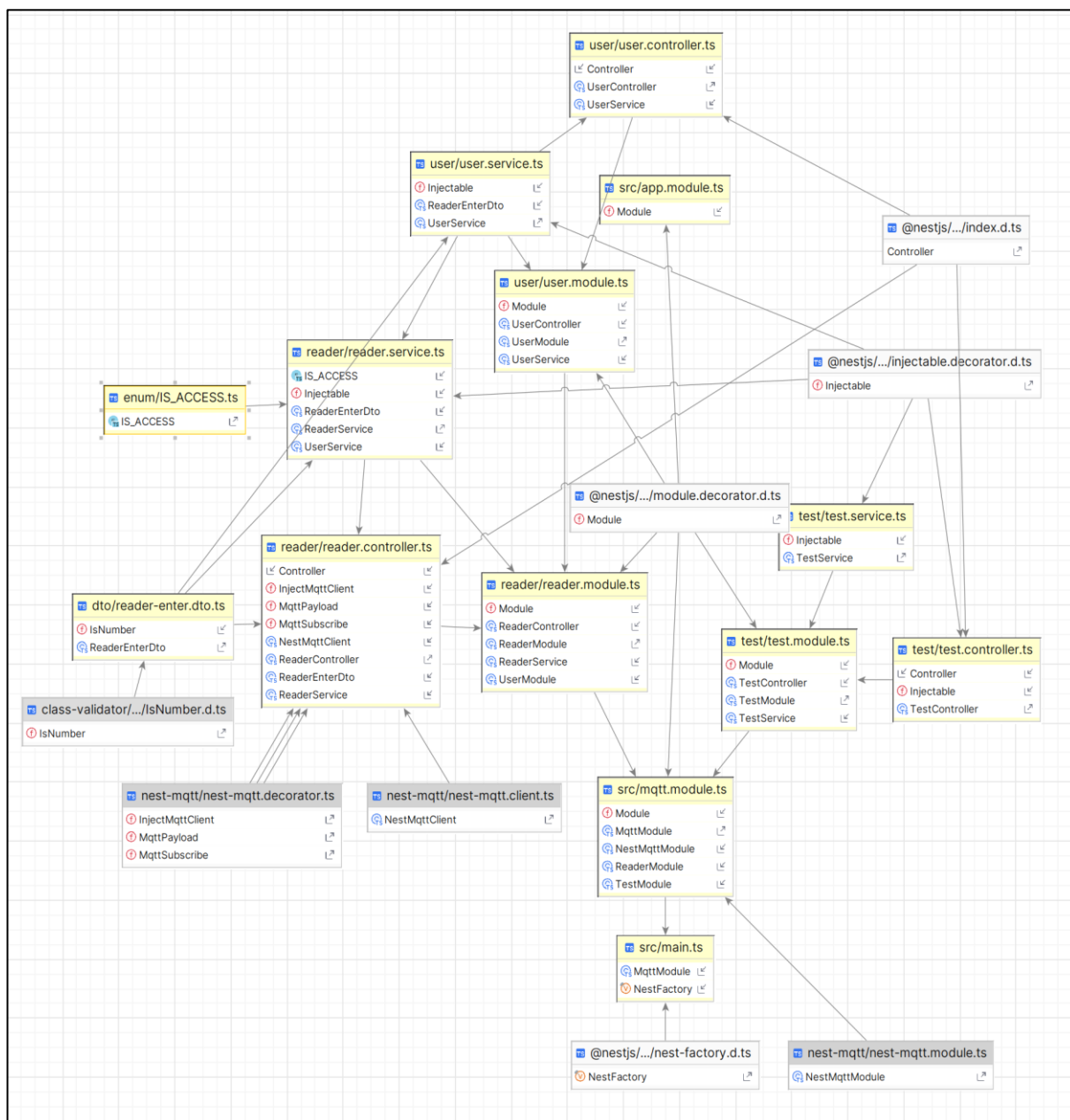


Рисунок 17 – UML-диаграмма всех классов

Для общения по протоколу MQTT была написана библиотека, вынесенная в отдельный модуль:

```

@Global()
@Module({
  imports: [DiscoveryModule],
  exports: [NestMqttClient],
})
export class NestMqttModule {
  public static forRoot(options: MqttModuleOptions):
  DynamicModule {
    return {

```

```

        module: NestMqttModule,
        providers: [
            {
                provide: MQTT_OPTION_PROVIDER,
                useValue: options,
            },
            NestMqttClient,
            NestMqttExplorer,
        ],
    },
};
}
}

```

В приведенном модуле регистрируются 3 класса использующих паттерн провайдер. Первый класс содержит параметры настройки MQTT клиента и ассоциируется с меткой MQTT_OPTION_PROVIDER. Класс NestMqttClient содержит все API функции для работы с данной библиотекой. Декораторы для API прописаны в отдельном файле nest-mqtt.decorator.ts. Основным ядром библиотеки является класс провайдер NestMqttExplorer, в данном классе реализована обработка декораторов и предобработка сообщений, получаемых от брокера. Обработка декораторов происходит при запуске приложения в функции explore() с помощью технологии рефлексии предоставляемой NestJS:

```

explore() {
    const controllers: InstanceWrapper[] =
this.discoveryService.getControllers();
    controllers.forEach((wrapper: InstanceWrapper) => {
        const {instance} = wrapper;
        if (!instance) {
            return;
        }
        this.metadataScanner.scanFromPrototype(
            instance,
            Object.getPrototypeOf(instance),
            key => {
                const subscribeOptions: MqttSubscribeOptions =
this.reflector.get(
                    SUBSCRIBE_OPTIONS,
                    instance[key],
                );
                const parameters = this.reflector.get(
                    SUBSCRIBER_PARAMS,
                    instance[key],
                );
                if (subscribeOptions) {

```

```

        this.subscribe(subscribeOptions, parameters,
instance[key], instance);
    }
    },
);
});

```

В приведенном выше участке кода, при инициализации приложения происходит получение всех контроллеров (именно в них располагается код по обработке запросов) и проверка сопутствующих декораторов для их функций. Далее функции добавляются в массив обработчиков. В функции explore() происходит также создания обработчика сообщений:

```

this.client.on(
    'message',
    (topic: string, payload: Buffer, packet: Packet) => {
        const subscriber = this.getSubscriber(topic);
        if (subscriber) {
            const parameters = subscriber.parameters ||
[];
            const scatterParameters:
MqttSubscriberParameter[] = [];
            for (const parameter of parameters) {
                scatterParameters[parameter.index] =
parameter;
            }
            try {
                const transform =
getTransform(subscriber.options.transform);
                if (this.options.beforeHandle) {
                    this.options.beforeHandle(topic,
payload, packet);
                }

subscriber.handle.bind(subscriber.provider) (
                    ...scatterParameters.map(parameter =>
{
                        switch (parameter?.type) {
                            case 'payload':
                                return
transform(payload);
                            case 'topic':
                                return topic;
                            case 'packet':
                                return packet;
                            case 'params':

```

```

        return
NestMqttExplorer.matchGroups(topic, subscriber.regex);
        default:
            return null;
    }
    })),
    );
    } catch (err) {
        this.logger.error(err);
    }
    }
    },
);

```

При помощи внедрения функции `getTransform(transform: 'json' | 'text' | MqttMessageTransformer)`, удалось создать сериализацию данных из формата `json` и `text` на уровне библиотеки предоставляю разработчику готовый класс для его `.dto` объектов. При дальнейшей разработки такое решение даёт возможность легкого добавления новых форматов представления данных, например формата `protobuf`, который является бинарным представлением данных и ускоряет процесс их передачи.

Для обработки запросов от контроллера были созданы `reader.module`, `reader.controller`, `reader.service`. В `reader.controller` для подписки на определенный `topic`, получения с него сообщения и отправки ответа используется написанная библиотека `nest-mqtt`:

```

@Controller('reader')
export class ReaderController {
    constructor(private readonly readerService: ReaderService,
        @InjectMqttClient() private readonly mqttService:
NestMqttClient) {
    }

    @MqttSubscribe({
        topic: 'reader/enter/',
        transform: 'json',
    })
    async readerEnter(@MqttPayload() readerEnterDto:
ReaderEnterDto) {
        const response = await
this.readerService.readerEnter(readerEnterDto);
        console.log(`For user ${readerEnterDto.userId} access is
${response}`);
    }
}

```

```

        await
this.mqttService.publish(`reader/${readerEnterDto.deviceId}/acce
ss/`, response.toString());
    }
}

```

DTO – это классы или структуры, используемые для передачи данных между разными уровнями приложения или между разными приложениями. Они служат общим интерфейсом для обмена данными, отделяя внутренние структуры данных одного уровня или приложения от другого. Для передачи информации о попытке входа была создана следующая структура:

```

export class ReaderEnterDto {
  @IsNumber({}, {message: 'must be number'})
  readonly userId: number;

  @IsNumber({}, {message: 'must be number'})
  readonly deviceId: number;
}

```

Для валидации данных был использован такой инструмент как pipe. Pipes в NestJS – это инструменты, предназначенные для трансформации и валидации данных в приложениях NestJS. Они позволяют изменять и проверять данные, поступающие в контроллеры и сервисы, перед их использованием (рисунок 18). Pipes обеспечивают гибкость и настраиваемость при работе с входными данными, делая код более читаемым и легко поддерживаемым.

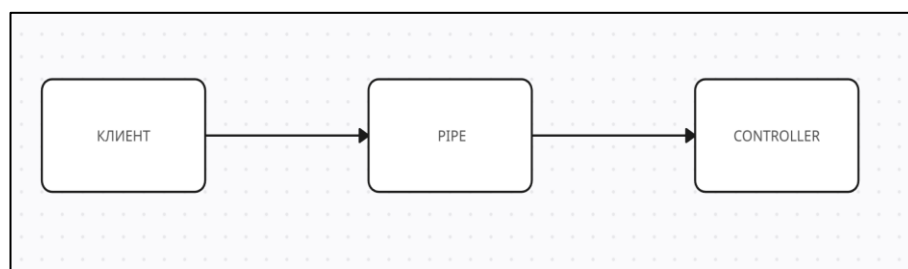


Рисунок 18 - Работа pipe

С помощью декораторов подобных `@IsNumber` перед поставкой запроса на обработку все поля проверяются на валидность и в случае несоответствия типов полей выдается ошибка:

```
@Injectable()
export class ValidationPipe implements PipeTransform<any> {
  async transform(value: any, metadata: ArgumentMetadata):
  Promise<any> {
    const obj = plainToClass(metadata.metatype, value);
    const errors = await validate(obj);

    if (errors.length) {
      let messages = errors.map(err => {
        return ` ${err.property} -
${Object.values(err.constraints).join(', ')} `;
      });
      throw new ValidationException(messages);
    }
  }
}
```

Все основные операции с пользователями были вынесены в `user.module`. Так работа с базой данных происходит в `user.service` с помощью функций `sequelize`.

UML диаграмма классов стандартного модуля на примере `reader.module` представлена рисунке 19.

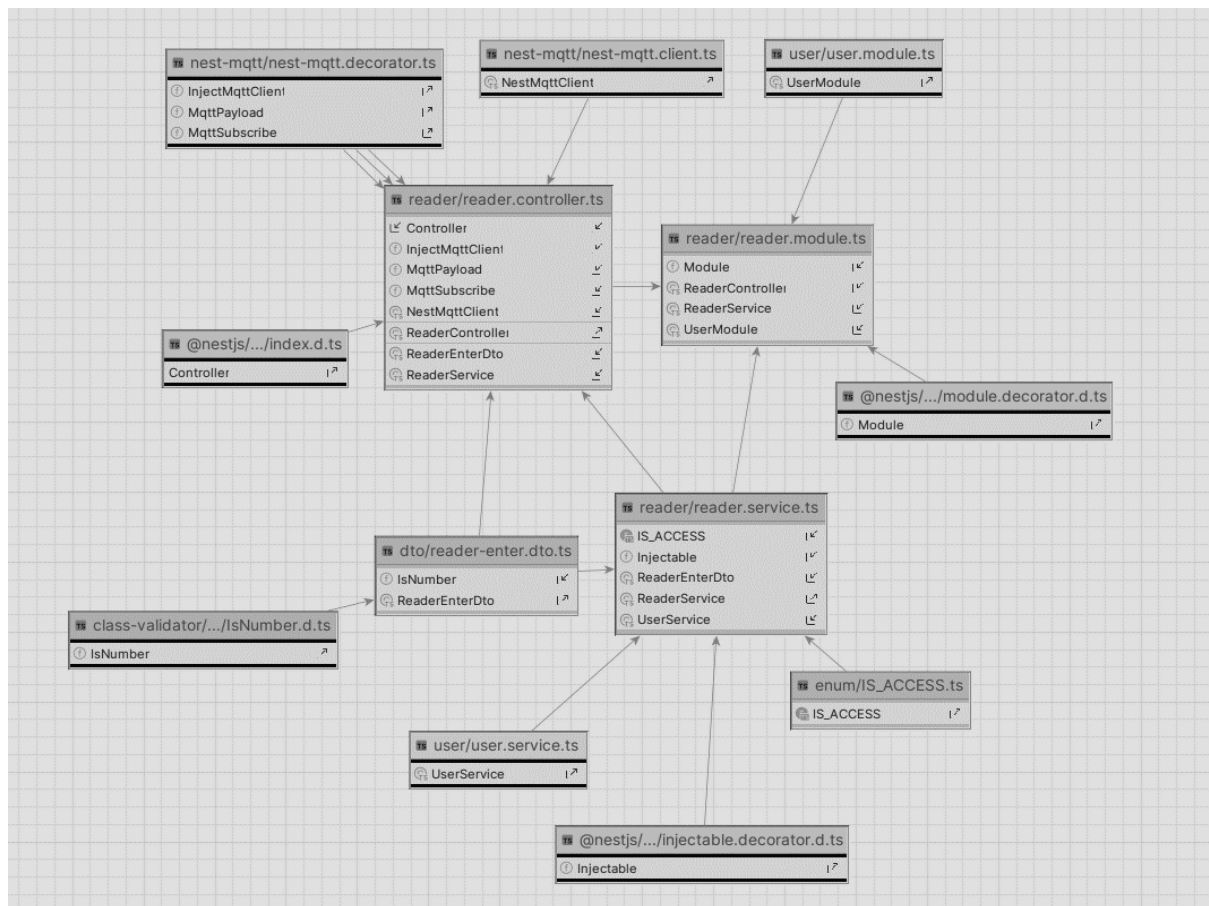


Рисунок 19 – UML-диаграмма классов модуля

Структура всех остальных модулей идентична.

1.2.2 Проектирование и создание базы данных

При проектировании базы данных была создана следующая диаграмма сущностей и зависимостей. Диаграмма представлена на рисунке 20.

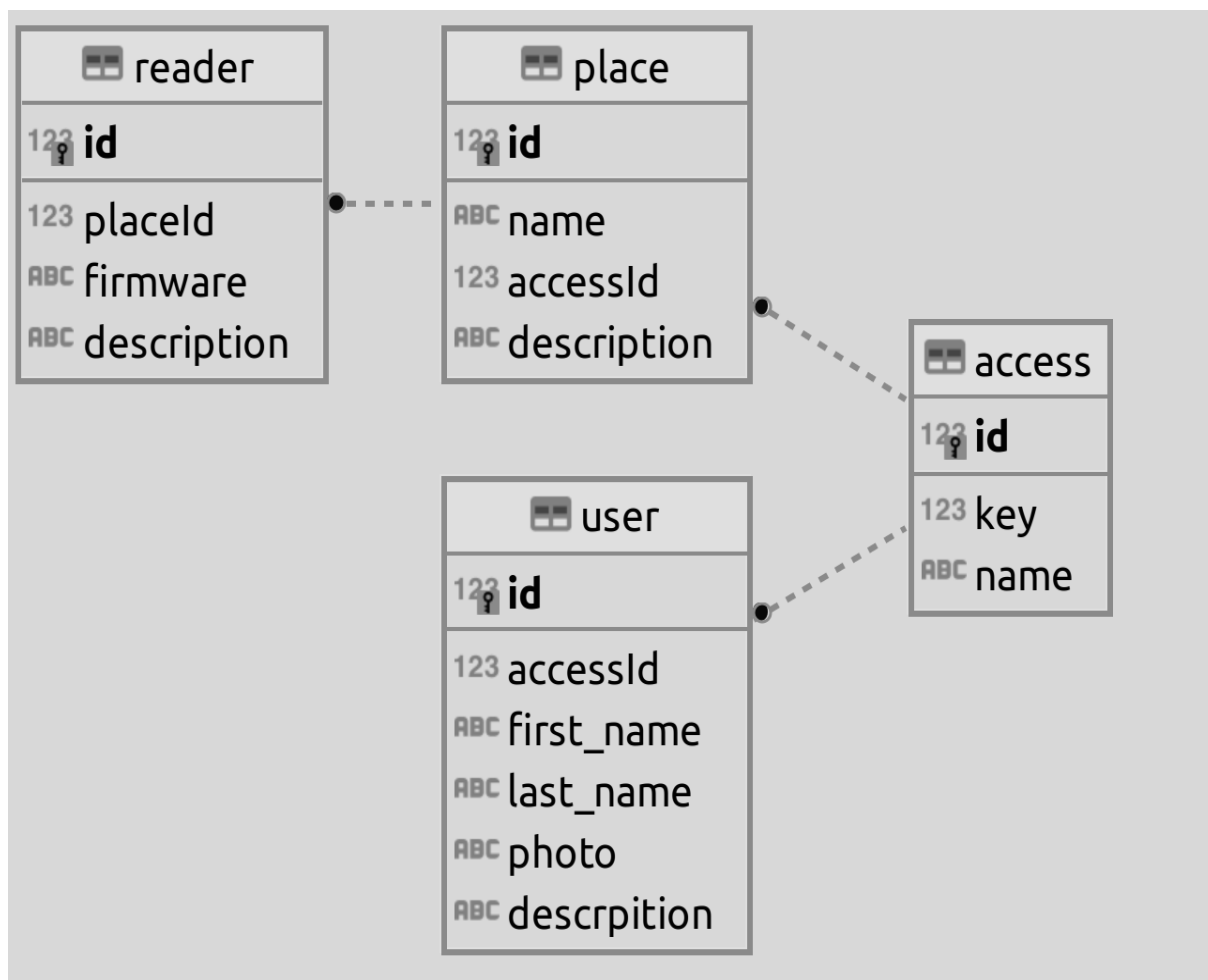


Рисунок 20 - ER-диаграмма

Сущность access представляет собой права доступа и имеет колонку key в которой находится позиция в иерархии разрешений (чем vyšло число, тем выше права доступа). Таблица user представляет собой сущность пользователя и имеет несколько колонок для идентификации человека, а также поле accessId, которое ссылается на таблицу access. У каждого пользователя в базе данных прописаны максимальные права доступа, которые применяются при проверке входа. Каждому пользователю была добавлена колонка photo, в которой в дальнейшем будет храниться фотография в формате base64, для возможности идентификации по лицу.

Таблица reader имеет колонки для описания устройства и ключ ссылающийся на таблицу place, которая в свою очередь имеет ссылку на таблицу с правами доступа. Так как в одном месте может стоять несколько

устройств и считывателей (например, на контрольно-пропускном пункте имеется несколько турникетов), правда доступа разделяются по местам нахождения электронных замков.

1.2.3 Реализация клиента на STM32

Для разработки устройства компанией заказчиком была спроектирована и предоставлена схема, представленная в приложениях 1-3.

Основные компоненты схемы:

- модуль питания обеспечивает питание всех остальных компонентов устройства;
- центральный процессор (ЦП) является платой STM32 и выполняет все вычисления и обработку данных;
- модуль Ethernet обеспечивает подключение устройства к сети Ethernet;
- модуль Wi-Fi является платой ESP32, обеспечивает подключение устройства к сети Wi-Fi;
- модуль RFID используется для считывания информации с RFID-меток;
- модуль смарт-реле необходим для работы с самим механизмом замка.

Для реализации проекта использовалась система реального времени FreeRTOS. Для исполнения необходимого функционала были созданы несколько задач:

- `tsk_LED`, задача индикации устройства о перепрошивки и другой служебной информации;
- `tsk_status_LED`, задача индикации о разрешении и запрете доступа при прикладывании карты;
- `tsk_clk`, системная задача для работы с таймерами;
- `tsk_esp32`, задача для опроса платы ESP32 и общения с сервером;
- `tsk_smart_relay`, задача общения с устройством закрытия и открытия;
- `tsk_update`, задача обновления прошивки контроллера;
- `tsk_main`, системная основная задача.

Диаграмма задач представлена на рисунке 21.

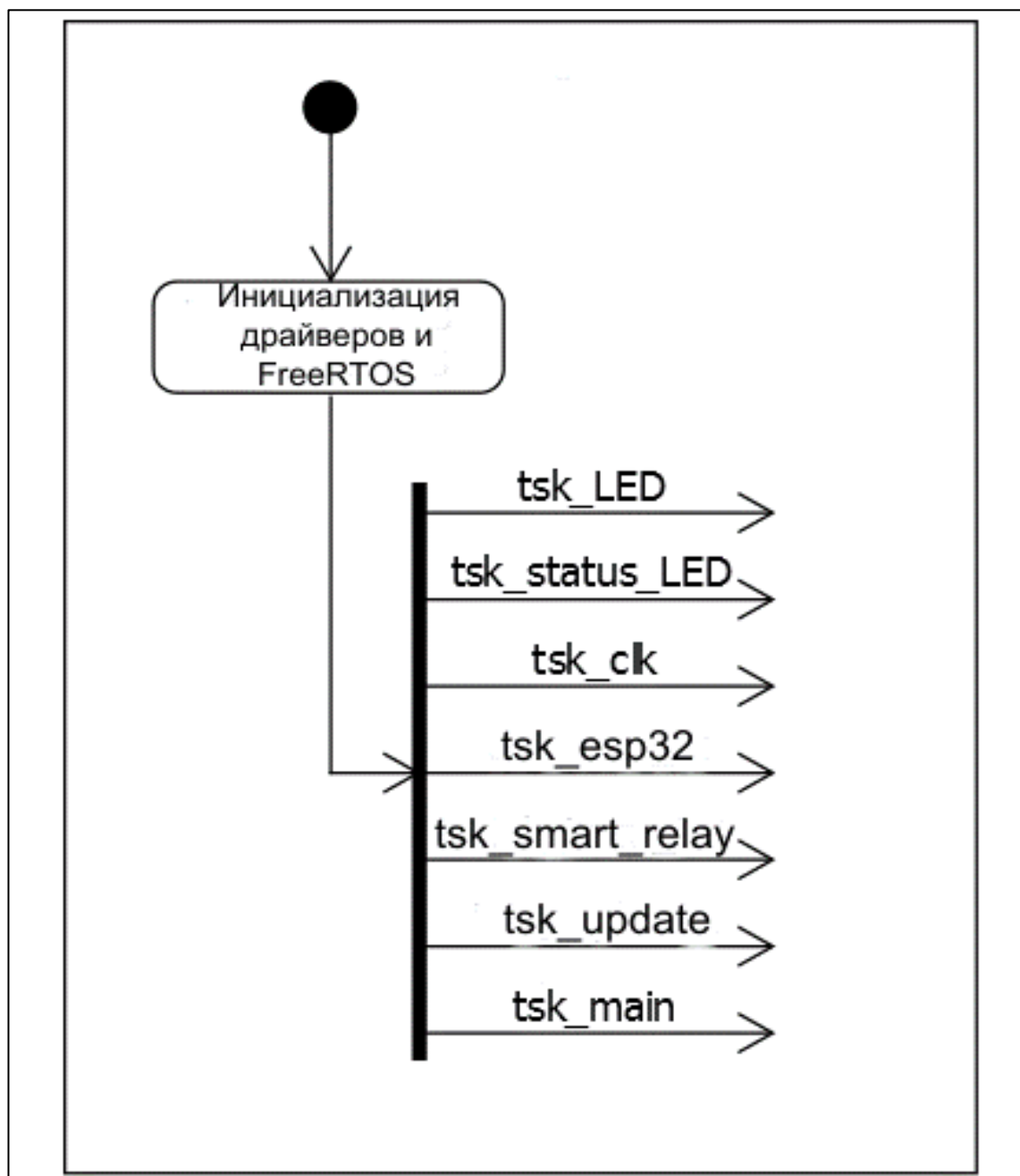


Рисунок 21 – UML-диаграмма задач

Инициализация задач и запуск операционной системы происходит в функции main:

```

int main() {
    // Инициализация HAL
    HAL_Init();
    // Инициализация UART
    MX_UART_Init();
    // Инициализация GPIO
    MX_GPIO_Init();
    // Создание семафора

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

50

```

xSemaphoreDataReady = xSemaphoreCreateBinary(NULL);
// Создание задач
xTaskCreate(readRFIDTask, "tsk_LED", 128, NULL, 1, NULL);
xTaskCreate(sendDataTask, "tsk_status_LED", 128, NULL, 2, NULL);
xTaskCreate(manageLockTask, "tsk_clk", 128, NULL, 3, NULL);
xTaskCreate(manageLockTask, "tsk_esp32", 128, NULL, 3, NULL);
xTaskCreate(manageLockTask, "tsk_smart_relay", 128, NULL, 3,
NULL);
xTaskCreate(manageLockTask, "tsk_update", 128, NULL, 3, NULL);
xTaskCreate(manageLockTask, "tsk_main", 128, NULL, 3, NULL);
// Запуск задач
vTaskStartScheduler(); for (;;) {}
}

```

Для обозначения получения данных с сервера используется семафор:

```

void readRFIDTask(void *pvParameters) {
while (1) {
// Считывание RFID-метки
uint8_t tagID = readRFIDTag();
// Проверка наличия метки
if (tagID != 0) {
//Сигнализация другим задачам о наличии данных
xSemaphoreGive(xSemaphoreDataReady);
}
vTaskDelay(100); // Задержка на 100 мс
}
}

```

В фоновом режиме, задачей простоя является ожидание получения сигнала о считывания карты.

Работа по протоколу mqtt и любые другие взаимодействия с сетью реализованы с помощью встроенного функционала в плате ESP32 – AT-команд.

ESP32 – это микроконтроллер, разработанный компанией «Espressif Systems». Он обладает широким спектром функций, что делает его идеальным для различных задач. В особенности для работы с сетью. ESP32 может подключаться к Wi-Fi, Bluetooth и другим сетям, что делает его идеальным для создания устройств IoT, таких как умные дома, носимые устройства и системы мониторинга.

АТ-команды (Attention commands) - это набор команд, используемых для управления и конфигурирования устройств ESP32. Использование АТ-команд:

					ПГУ 1.090301.05.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

- AT-команды отправляются ESP32;
- через последовательный порт (UART) с помощью терминальной программы;
- через Wi-Fi с помощью команды AT+CWDHCPS;
- ESP32 интерпретирует команду;
- распознает тип команды и ее параметры;
- ESP32 выполняет команду;
- выполняет заданное действие (подключение к Wi-Fi, отправка данных);
- ESP32 отправляет ответ;
- сообщает о результате выполнения команды (OK, ERROR).

С использованием данной разработки ESP32 используется как модуль для взаимодействий с сетью. Для подключения к тестовой сети вай фай была написана следующая функция:

```
void connectToWifi(void){
    uint8_t receiveBuffer[BUFFER_SIZE] = "";

    HAL_UART_Transmit(&huart1, "ATE0\r\n", strlen("ATE0\r\n"),
    10000);
    volatile int status = HAL_UART_Receive(&huart1, receiveBuffer,
    BUFFER_SIZE, 10000);
    memset(receiveBuffer, 0, sizeof(receiveBuffer));

    HAL_UART_Transmit(&huart1, "AT+CWINIT=1\r\n",
    strlen("AT+CWINIT=1\r\n"), 10000);
    status = HAL_UART_Receive(&huart1, receiveBuffer,
    BUFFER_SIZE, 10000);
    memset(receiveBuffer, 0, sizeof(receiveBuffer));

    HAL_UART_Transmit(&huart1, "AT+CWMODE=1\r\n",
    strlen("AT+CWMODE=1\r\n"), 10000);
    status = HAL_UART_Receive(&huart1, receiveBuffer,
    BUFFER_SIZE, 10000);
    memset(receiveBuffer, 0, sizeof(receiveBuffer));

    HAL_UART_Transmit(&huart1, "AT+WPS=1,7\r\n",
    strlen("AT+WPS=1\r\n"), 10000);
    status = HAL_UART_Receive(&huart1, receiveBuffer,
    BUFFER_SIZE, 10000);
    memset(receiveBuffer, 0, sizeof(receiveBuffer));
}
```

```

        HAL_UART_Transmit(&huart1,
        "AT+CWJAP=\"Maks\", \"25252525\"\\r\\n",
        strlen("AT+CWJAP=\"Maks\", \"25252525\"\\r\\n"),
        10000);
        status      =      HAL_UART_Receive(&huart1,      receiveBuffer,
        BUFFER_SIZE, 10000);
        memset(receiveBuffer, 0, sizeof(receiveBuffer));
    }

```

Таким же образом были написаны функции для подключения по протоколу mqtt.

1.3 Анализ результатов

В рамках выполнения проекта реализованы: серверное приложение, написанное на языке javascript с использованием фреймворка NestJS, клиент на плате STM32, а также была спроектирована и создана база данных.

Диаграмма развертывания системы представлена на рисунке 22. Диаграмма использования приложения представлена на рисунке 23. В дальнейшем при доработке системы, будет добавлена панель администратора для доступа к функциям настройки. Диаграмма поведения администратора представлена на рисунке 24.

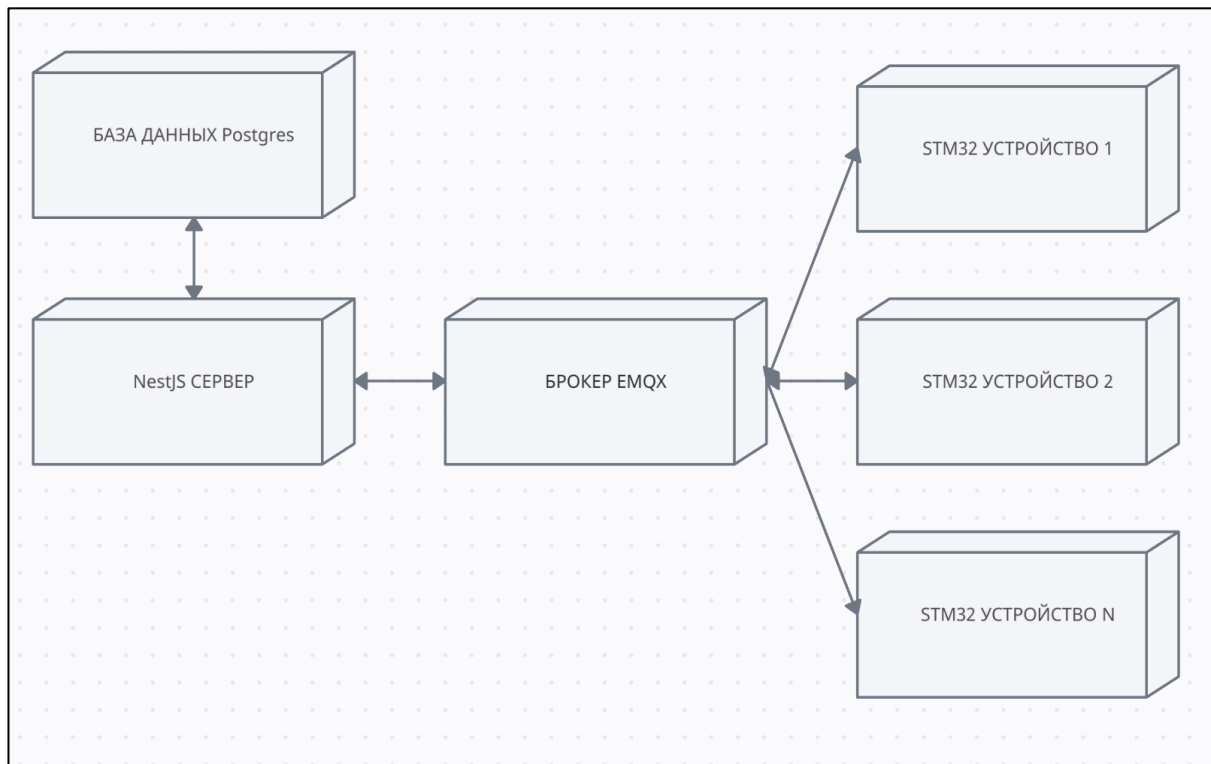


Рисунок 22 – UML-диаграмма развертывания

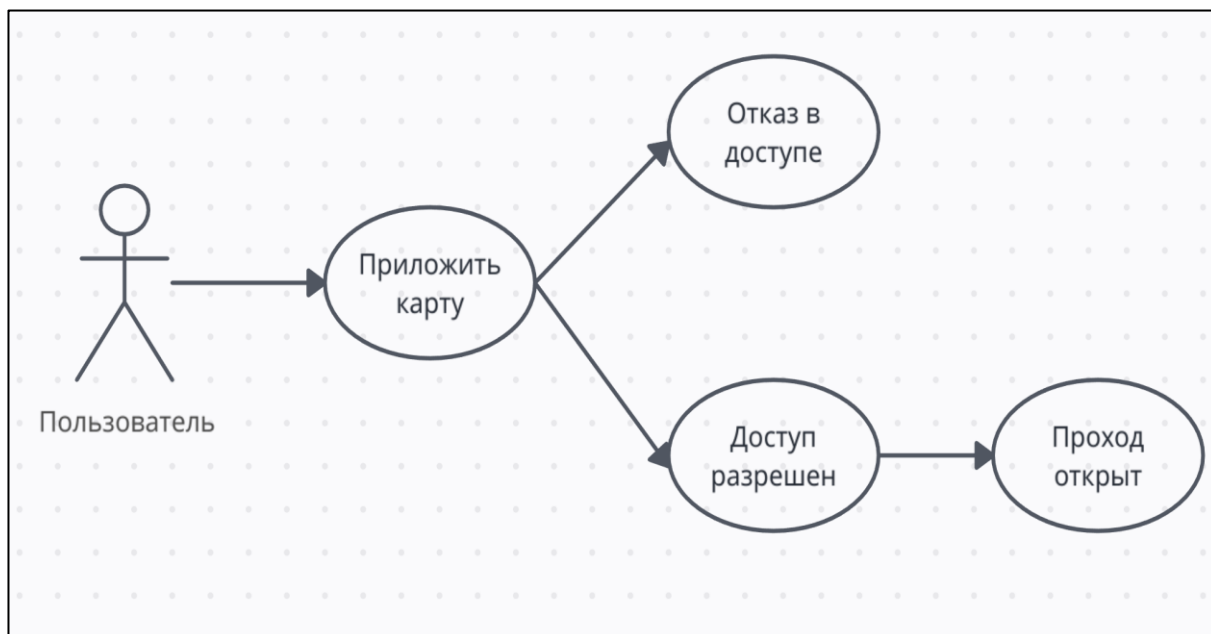


Рисунок 23 – UML-диаграмма использования

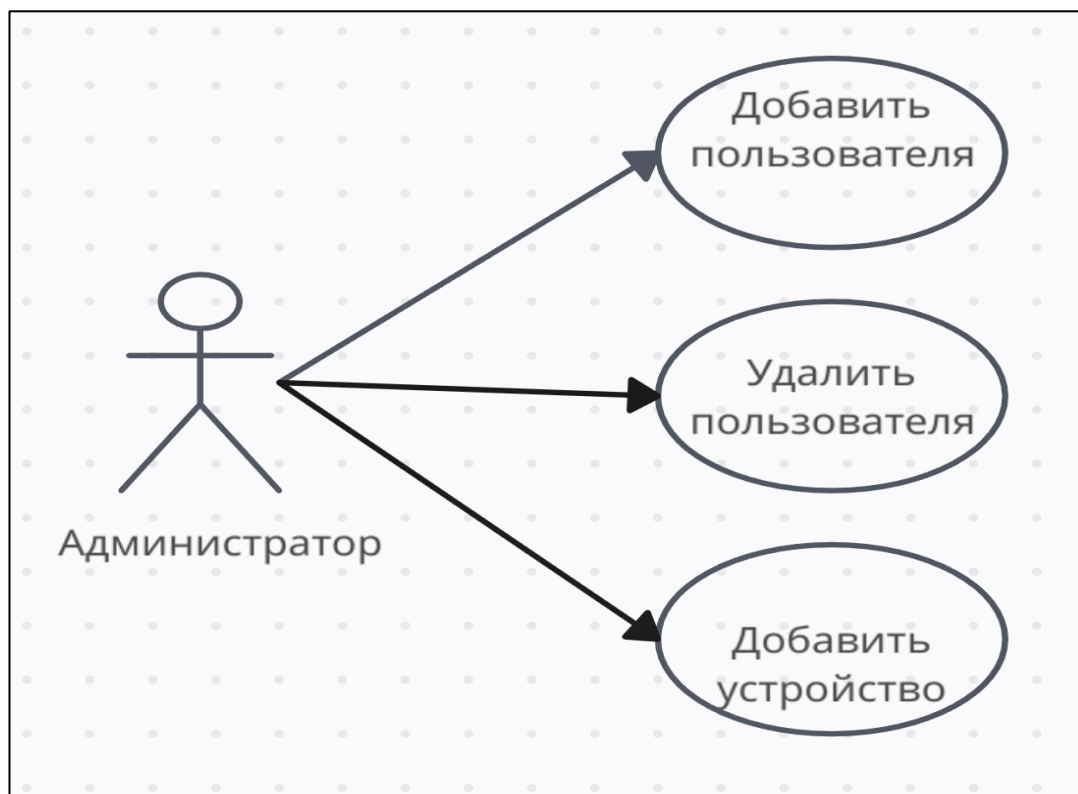


Рисунок 24 – UML диаграмма использования со стороны администратора

1.3.1 Тестирование серверной части.

Для тестирования серверной части использовалось программное обеспечение Postman. Postman – это популярная платформа разработчиков API, которая используется для разработки, тестирования, документирования и управления API.

Ключевые функции Postman:

- создание запросов. Позволяет создавать и отправлять HTTP-запросы (GET, POST, PUT, PATCH, DELETE) к API. Возможно легко указать URL-адрес, заголовки, тело запроса и параметры;
- тестирование API. Позволяет тестировать различные сценарии использования API, проверяя его функциональность и производительность. Возможно легко просматривать ответы API, включая коды статуса, заголовки и тело ответа.

Данный инструмент имеет также возможности создания запросов по mqtt протоколу.

Подключение клиента postman и подписка на необходимые топики показано на рисунке 25.

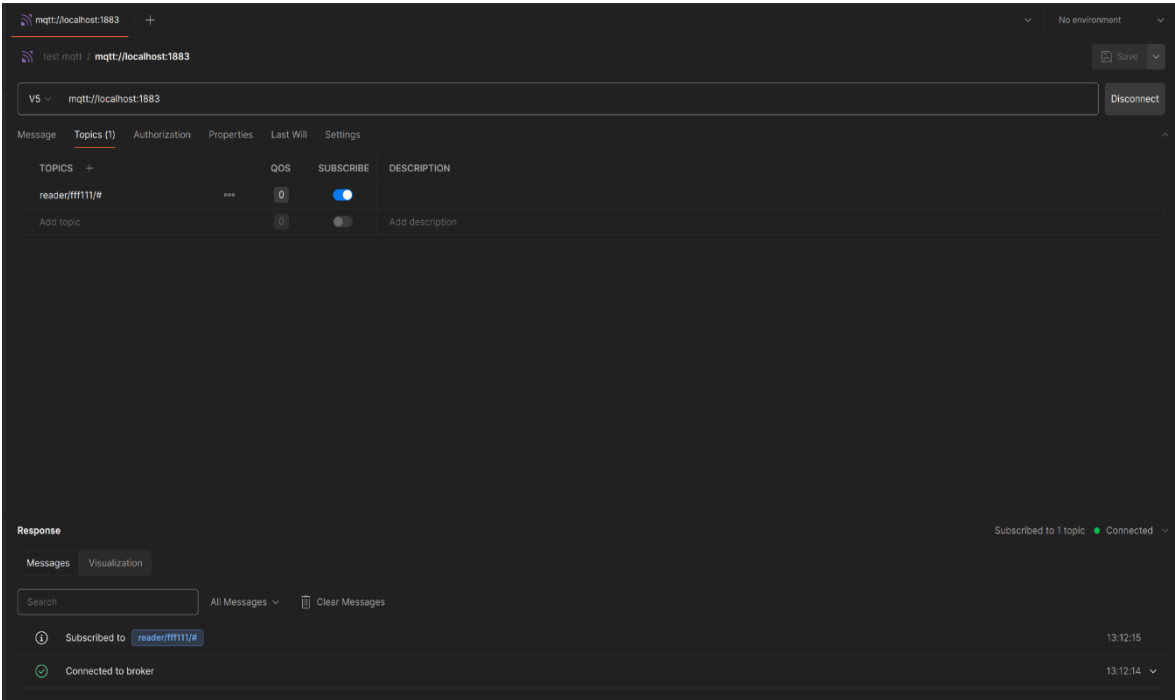


Рисунок 25 - Подключение клиента

Отправка сообщения и получение ответа представлено на рисунке 26.

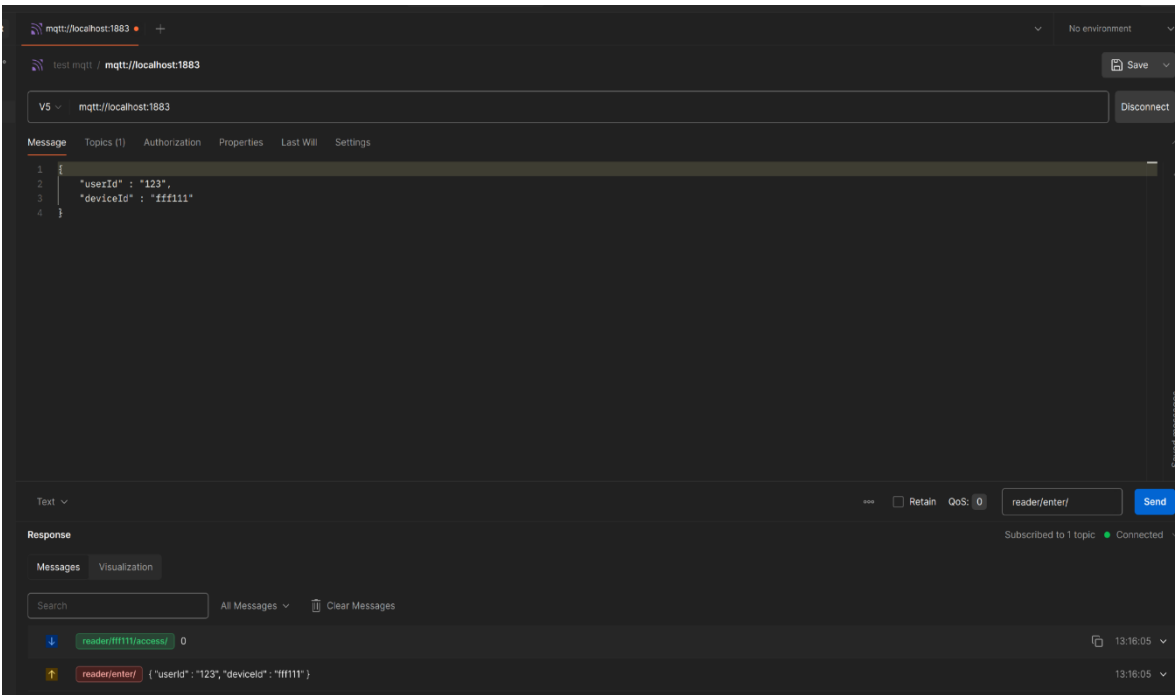


Рисунок 26 - Отправка сообщения

Вывод сервера представлен на рисунке 27.

```
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [InstanceLoader] UserModule dependencies initialized +0ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [InstanceLoader] DiscoveryModule dependencies initialized +0ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [InstanceLoader] NestMqttModule dependencies initialized +1ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [InstanceLoader] ReaderModule dependencies initialized +0ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [RoutesResolver] ReaderController {/reader}: +2ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [RoutesResolver] UserController {/user}: +0ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [RoutesResolver] TestController {/test}: +0ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [MQTT_LOGGER_PROVIDER] NestMqttClient initialized
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [MQTT_LOGGER_PROVIDER] NestMqttModule dependencies initialized
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [NestApplication] Nest application successfully started +1ms
(Nest) 8753 - 05/22/2024, 1:14:11 PM LOG [MQTT_LOGGER_PROVIDER] MQTT connected

For user 123 access is 0
[]
```

Рисунок 27 - Вывод сервера

1.3.2 Тестирование микроконтроллера

Вывод переменных микроконтроллера при подключении к сети wifi представлен на рисунке 28.

```
117 void connectToWifi(void){
118     uint8_t receiveBuffer[BUFFER_SIZE] = ""; receiveBuffer: uint8_t [255]
119
120     HAL_UART_Transmit(&huart1, pData: "ATE0\r\n", Size: strlen("ATE0\r\n"), Timeout: 10000);
121     volatile int status = HAL_UART_Receive(&huart1, pData: receiveBuffer, Size: BUFFER_SIZE, Timeout: 10000); status: 3
122     memset(receiveBuffer, 0, sizeof(receiveBuffer));
123
124     HAL_UART_Transmit(&huart1, pData: "AT+CWINIT=1\r\n", Size: strlen("AT+CWINIT=1\r\n"), Timeout: 10000);
125     status = HAL_UART_Receive(&huart1, pData: receiveBuffer, Size: BUFFER_SIZE, Timeout: 10000);
126     memset(receiveBuffer, 0, sizeof(receiveBuffer));
127
128     HAL_UART_Transmit(&huart1, pData: "AT+CWMODE=1\r\n", Size: strlen("AT+CWMODE=1\r\n"), Timeout: 10000);
129     status = HAL_UART_Receive(&huart1, pData: receiveBuffer, Size: BUFFER_SIZE, Timeout: 10000); status: 3
130     memset(receiveBuffer, 0, sizeof(receiveBuffer)); receiveBuffer: uint8_t [255]
131
132     HAL_UART_Transmit(&huart1, pData: "AT+WPS=1,7\r\n", Size: strlen("AT+WPS=1\r\n"), Timeout: 10000);
133     status = HAL_UART_Receive(&huart1, pData: receiveBuffer, Size: BUFFER_SIZE, Timeout: 10000);
134     memset(receiveBuffer, 0, sizeof(receiveBuffer));
135
136     HAL_UART_Transmit(&huart1, pData: "AT+CWJAP=\"Raks\", \"25252525\" \r\n", Size: strlen("AT+CWJAP=\"Raks\", \"25252525\" \r\n"),
137                       Timeout: 10000);
138     status = HAL_UART_Receive(&huart1, pData: receiveBuffer, Size: BUFFER_SIZE, Timeout: 10000);
139     memset(receiveBuffer, 0, sizeof(receiveBuffer));
140 }
```

connectToWifi

Threads & Variables Console GDB Memory View Peripherals

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

receiveBuffer = (uint8_t [255])

```
00 [0] = (uint8_t) 13 '\r'
01 [1] = (uint8_t) 10 '\n'
02 [2] = (uint8_t) 79 'O'
03 [3] = (uint8_t) 75 'K'
04 [4] = (uint8_t) 13 '\r'
05 [5] = (uint8_t) 10 '\n'
06 [6] = (uint8_t) 0 '\000'
07 [7] = (uint8_t) 0 '\000'
08 [8] = (uint8_t) 0 '\000'
09 [9] = (uint8_t) 0 '\000'
10 [10] = (uint8_t) 0 '\000'
```

Рисунок 28 -Получение ответа от ESP32

Получение сообщения от устройства на postman представлено на рисунке 29.

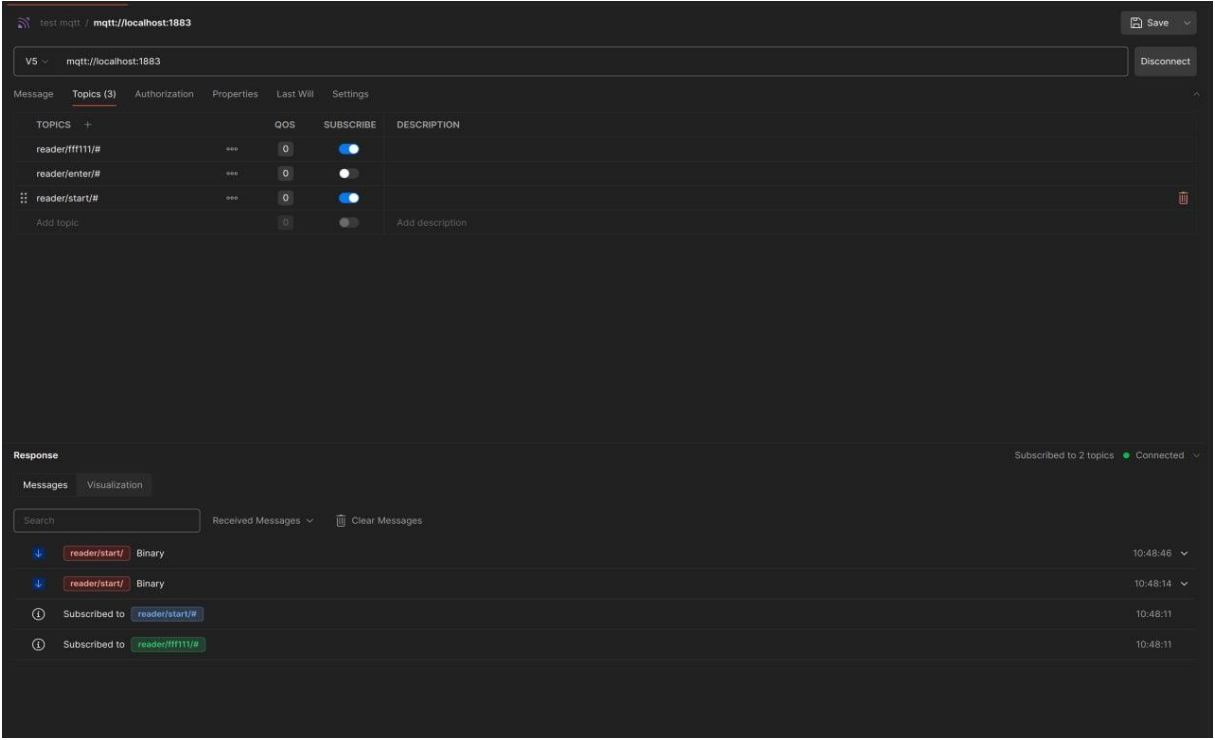


Рисунок 29 - Получение сообщения от устройства

2 Охрана труда

В данном разделе рассматриваются требования по обеспечению безопасности жизнедеятельности и охраны труда пользователей ПЭВМ, участвующих в разработке и эксплуатации программного обеспечения, а также способы минимизации пагубных факторов на здоровье.

2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Компьютеры прочно вошли в нашу жизнь, помогая нам в работе, учебе и отдыхе. Но, как и любой инструмент, они могут быть не только полезны, но и опасны. Длительное сидение за компьютером может привести к:

- проблемам со зрением: напряжение глаз, сухость, близорукость;
- нарушениям осанки: боли в спине, шее, суставах;
- головным болям и утомлению;
- стрессам и нервным расстройствам;
- снижению иммунитета.

Для минимизации рисков ухудшения здоровья необходимо строго соблюдать технику безопасности и придерживаться правил обращения с техникой. Необходимо соблюдать режим работы, правильно обустроить рабочее место, следить за положением своего тела, ограничивать время нахождения за компьютером, а также в нерабочее время следить за состоянием своего здоровья.

2.2 Классификация и категорирование помещения, в котором производится разработка

2.2.1 Классификация по взрывопожарной и пожарной опасности

По взрывопожарной и пожарной опасности помещения подразделяются на категории А, Б, В1-В4, Г и Д, а здания - на категории А, Б, В, Г и Д. Чем выше категория, тем строже требования к пожарной безопасности:

					ПГУ 1.090301.05.001 ПЗ	Лист
						59
Изм.	Лист	№ докум.	Подпись	Дата		

строительным конструкциям, системам противопожарной защиты, хранению ГВМ, мерам безопасности. Определение категории регламентируется Федеральным законом № 123-ФЗ и СП 1.13130.2020. Знание категории пожарной опасности необходимо для обеспечения безопасности людей, сохранности имущества, разработки и реализации противопожарных мер. Категория пожарной опасности помещений и зданий определяется комплексно, с учетом:

- горючих веществ и материалов (ГВМ): их вида, агрегатного состояния, способа горения, температуры вспышки и количества;
- планировочных решений помещений: объема, высоты, наличия и количества проемов, степени деления на пожарные отсеки;
- характеристик проводимых процессов: вида (с выделением тепла, использованием ГВМ, образованием взрывоопасных смесей), опасности (наличие открытого огня, высокого давления, электрооборудования).

Ниже приведены категории помещения и характеристик веществ и материалов, находящихся в помещении:

– «А» повышенная взрывопожароопасность - горючие газы, легковоспламеняющиеся жидкости с температурой вспышки не более 28 °С в таком количестве, что могут образовывать взрывоопасные парогазовоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 кПа, и (или) вещества и материалы, способные взрываться и гореть при взаимодействии с водой, кислородом воздуха или друг с другом, в таком количестве, что расчетное избыточное давление взрыва в помещении превышает 5 кПа;

– «Б» взрывопожароопасность - горючие пыли и волокна, легковоспламеняющиеся жидкости с температурой воздуха более 28 °С, горючие жидкости, способные образовать взрывоопасные пылевоздушные

или паровоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 кПа;

– «В1-В4» пожароопасность - горючие и трудногорючие жидкости, вещества и материалы, способные при взаимодействии с водой, кислородом воздуха или друг с другом только гореть;

– «Г» умеренная пожароопасность - Негорючие вещества и материалы в горячем, раскаленном состоянии, при обработке которых выделяется лучистое тепло, искры и пламя;

– «Д» пониженная пожароопасность - материалы в холодном состоянии и негорючие вещества.

2.2.2 Классификация по опасности поражения электрическим током

Условия повышенной опасности поражения людей электрическим током:

– наличие влажной (пары или конденсирующаяся влага выделяется в виде мелких капель, и относительная влажность воздуха превышает 75%);

– наличие проводящей пыли (технологическая или другая пыль, оседая на проводах, проникая внутри аппаратов или машин ухудшает условия изоляции и охлаждения, но не вызывает опасности пожара или взрыва);

– наличие металлических, земляных, железобетонных кирпичных оснований;

– наличие повышенной температуры (независимо от времени года температура превышает длительно 35°C или кратковременно 40°C).

Условия особой опасности поражения людей электрическим током:

– сырость (снег, дождь, повышенная влажность и т.д.);

– наличие химических веществ;

– наличие одновременно двух или более условий повышенной опасности.

Таким образом можно вынести три категории опасности помещений: безопасные, с повышенной опасностью и особо опасные.

					ПГУ 1.090301.05.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Для сокращения рисков поражения электрическим током рекомендуется работать в помещениях, соответствующих следующим критериям:

- температура воздуха не ниже 30°C, рекомендованный диапазон 18-20°C;
- полы должны быть выполнены из диэлектрического материала;
- помещение должно хорошо проветриваться;
- влажность воздуха не выше 55%.

2.3 Санитарно-гигиеническая характеристика производства

Обязательными санитарно-гигиеническими критериями являются: сухость помещения, температура внутри него и количество света. Также существует перечень основных требований:

- каждое рабочее место должно быть в объёме не менее 15 м³;
- высота помещения не менее 3,2 м;
- ширина дверей не менее 0,8 м;
- ширина коридоров не менее 1,4 м.

Требования по освещению:

- окна должны освещать каждое рабочее место;
- окна должны регулярно открываться для проветривания;
- окна необходимо очищать от пыли и грязи для обеспечения максимальной освещенности;
- в условиях недостаточной естественной освещенности должны быть установлены электроосветительные приборы;
- фонари должны быть изготовлены из армированного стекла и защищены металлическими сетками.

Оптимальный диапазон температур для человека составляет 18-21°C а влажности 40-60%. Также необходимо обеспечивать каждое рабочее помещение чистой питьевой водой и регулярно менять воду, не реже одного раза в сутки.

					ПГУ 1.090301.05.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

Соблюдение всех этих требований создает благоприятные условия труда и способствует сохранению здоровья и работоспособности сотрудников, тем самым повышая производительность труда.

Заключение

В ходе реализации данной работы были изучены принципы проектирования клиент-серверных приложений и баз данных, основы разработки программ для микроконтроллера STM32 архитектуры ARM Cortex-M и принципы реализации приложений, использующих протокол mqtt.

Также были получены навыки разработки на языке программирования javascript и фреймворке NestJS, навыки работы с базой данных Postgres. Были усовершенствованы навыки написания программ для встраиваемых систем.

Во время выполнения данной работы были разработаны: программа клиент для STM32 и сервер на NestJS развернутый в docker. Также была спроектирована база данных Postgres.

В дальнейшем проект будет совершенствоваться и расширяться.

					ПГУ 1.090301.05.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

Список используемых источников

1. Документация по МК stm32h753vi [Электронный ресурс] / Режим доступа: <https://www.st.com/en/microcontrollers-microprocessors/stm32h753vi.html> свободный. (дата обращения: 25.04.2024).
2. Документация по модулю wifi esp32-s2-mini [Электронный ресурс] / Режим доступа: <https://www.espressif.com/en/support/documents/technical-documents> свободный. (дата обращения: 25.04.2024).
3. Си для встраиваемых систем / Александр Тарасов. – Санкт-Петербург: Leanpub, 2019. – 267 с.
4. Документация по операционной системе реального времени FreeRTOS [Электронный ресурс] / Режим доступа: <https://www.freertos.org/a00106.html> свободный. (дата обращения: 30.04.2024).
5. Документация по STM32CubeIDE [Электронный ресурс] / Режим доступа: <https://www.st.com/en/development-tools/stm32cubeide.html> свободный. (дата обращения: 30.04.2024).
6. Документация по фреймворку NodeJS [Электронный ресурс] / Режим доступа: <https://nodejs.org/docs/latest/api/> свободный. (дата обращения: 5.05.2024).
7. Документация по фреймворку NestJS [Электронный ресурс] / Режим доступа: <https://docs.nestjs.com/> свободный. (дата обращения: 5.05.2024).
8. Руководство пользователя и обучающие материалы по WEBStorm [Электронный ресурс] / Режим доступа: <https://www.jetbrains.com/webstorm/learn/> свободный. (дата обращения: 7.05.2024).
9. Документация по postgresSQL [Электронный ресурс] / Режим доступа: <https://www.postgresql.org/docs/> свободный. (дата обращения: 10.05.2024).
10. Документация по протоколу MQTT [Электронный ресурс] / Режим доступа: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> свободный. (дата обращения: 8.05.2024).

11. Чистый код / Мартин Роберт С. – Санкт-Петербург: Питер, 2020. – 464 с.

12. Документация по docker [Электронный ресурс] / Режим доступа: <https://docs.docker.com/> свободный. (дата обращения: 25.05.2024).

					ПГУ 1.090301.05.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А

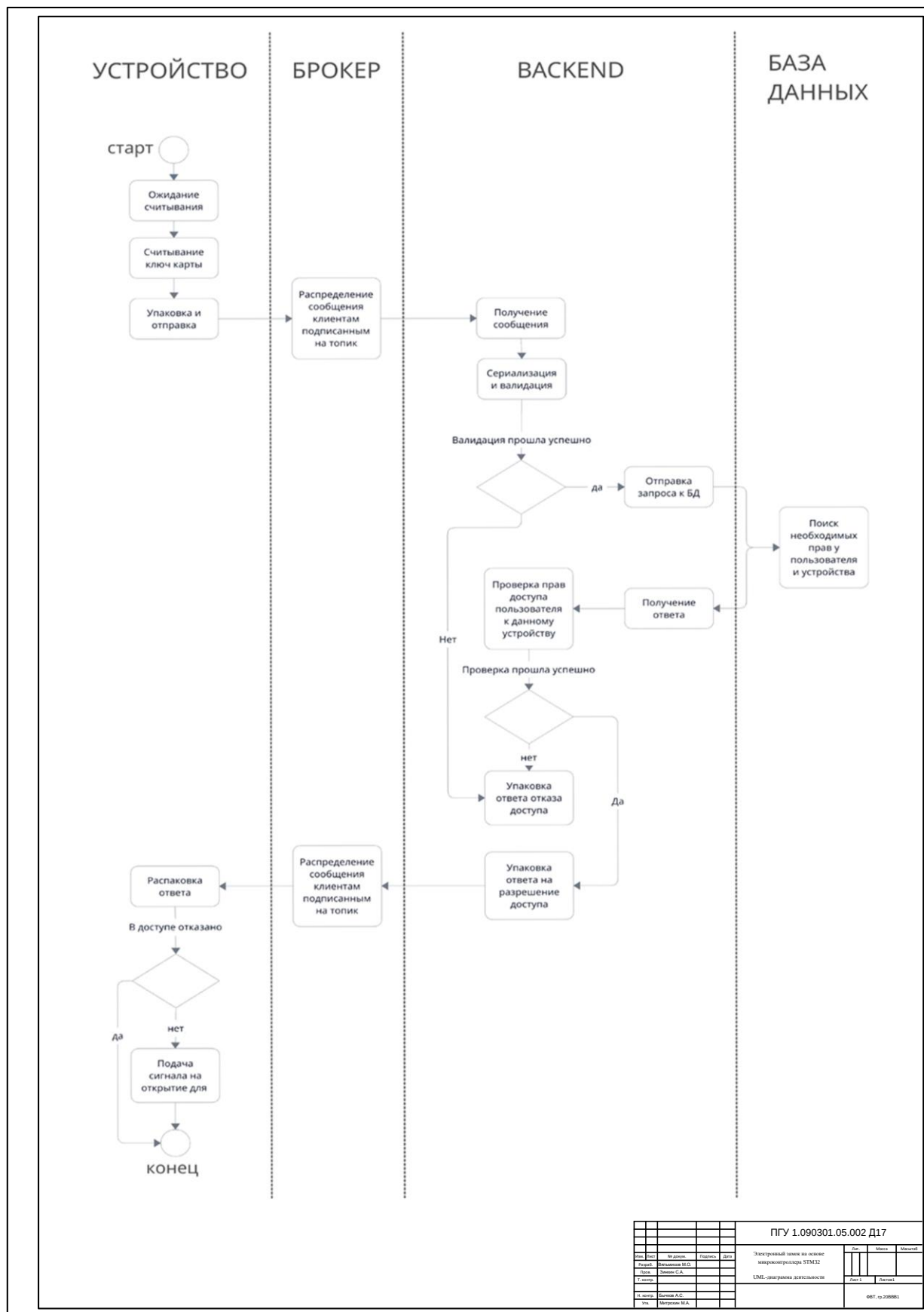
(Обязательное)

Электронный замок на основе микроконтроллера STM32

UML-диаграммы

					ПГУ 1.090301.05.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма деятельности



ПГУ 1.090301.05.002 Д17					Изм.	Лист	№ докум.	Подпись	Дата
Изм.	Лист	№ докум.	Подпись	Дата	Электронный текст на основе: информационной системы СТМ-СД				
Изд.	Лист	№ докум.	Подпись	Дата	ИМ-диагностика деятельности				
Изд.	Лист	№ докум.	Подпись	Дата	Лист 1				
Изд.	Лист	№ докум.	Подпись	Дата	001.09.00001				

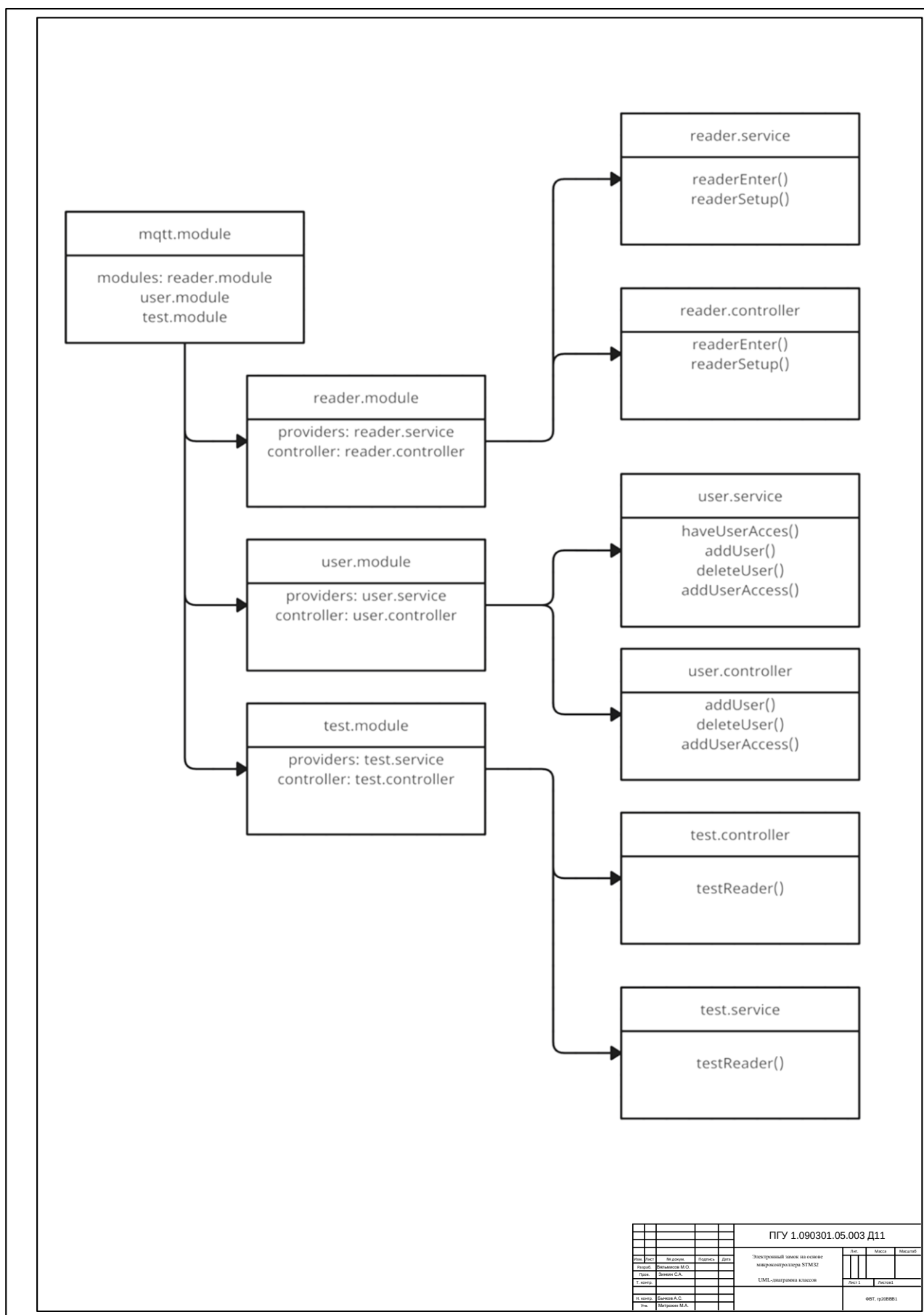
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

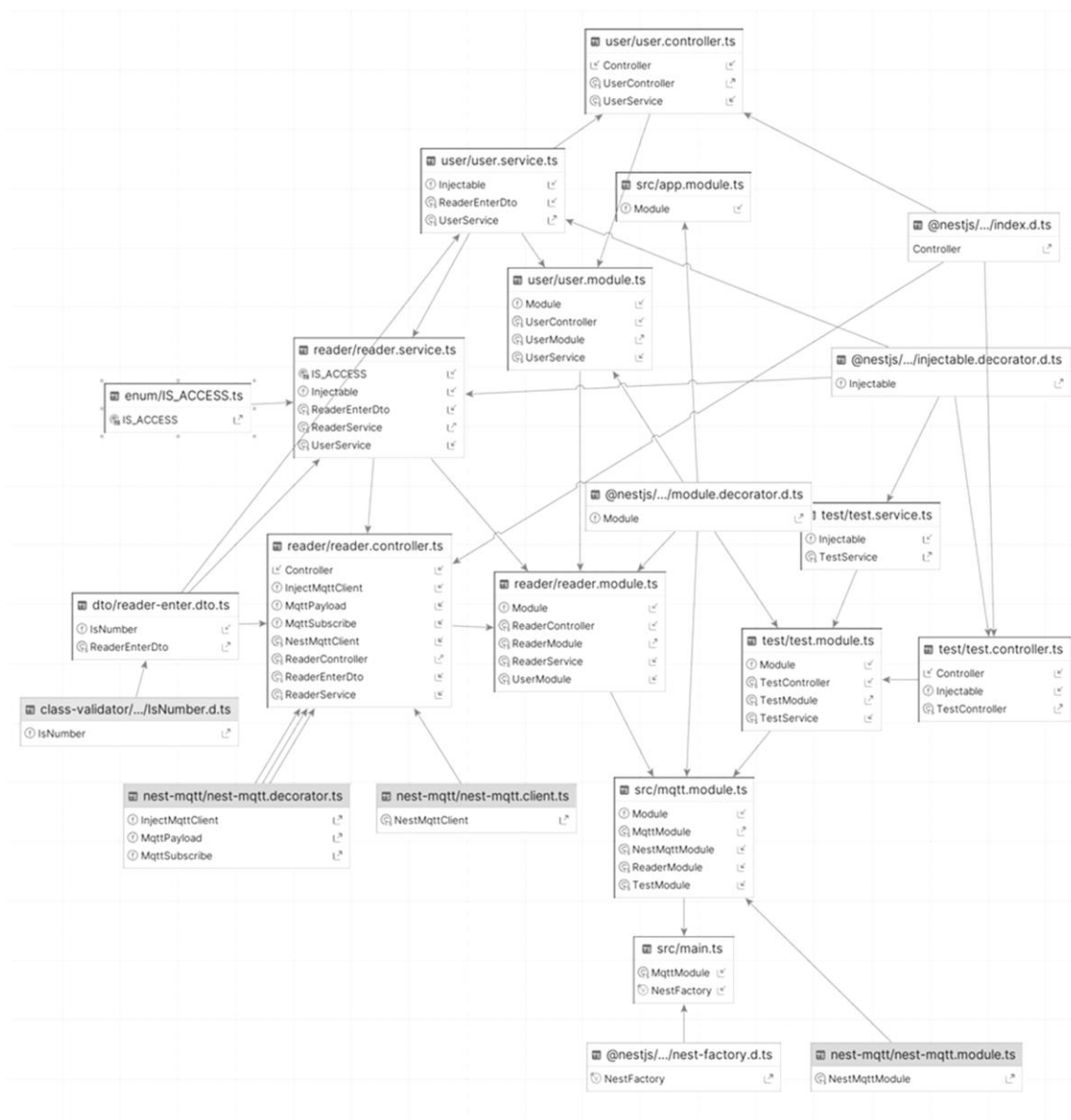
Лист

68

Приложение А.2 - Диаграмма основных классов



Приложение А.3 - Диаграмма всех классов



ПГУ 1.090301.05.004 Д11			
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия

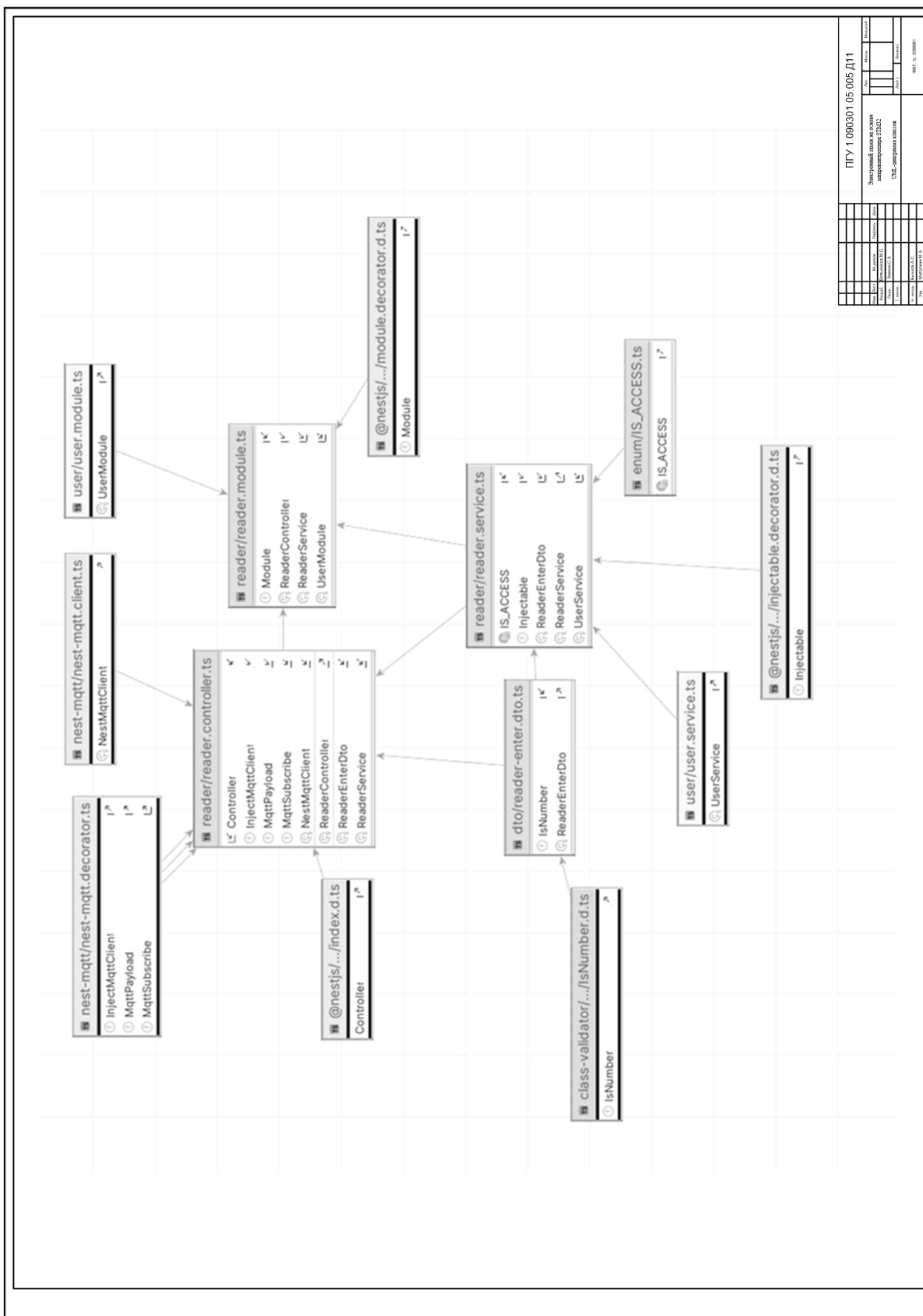
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

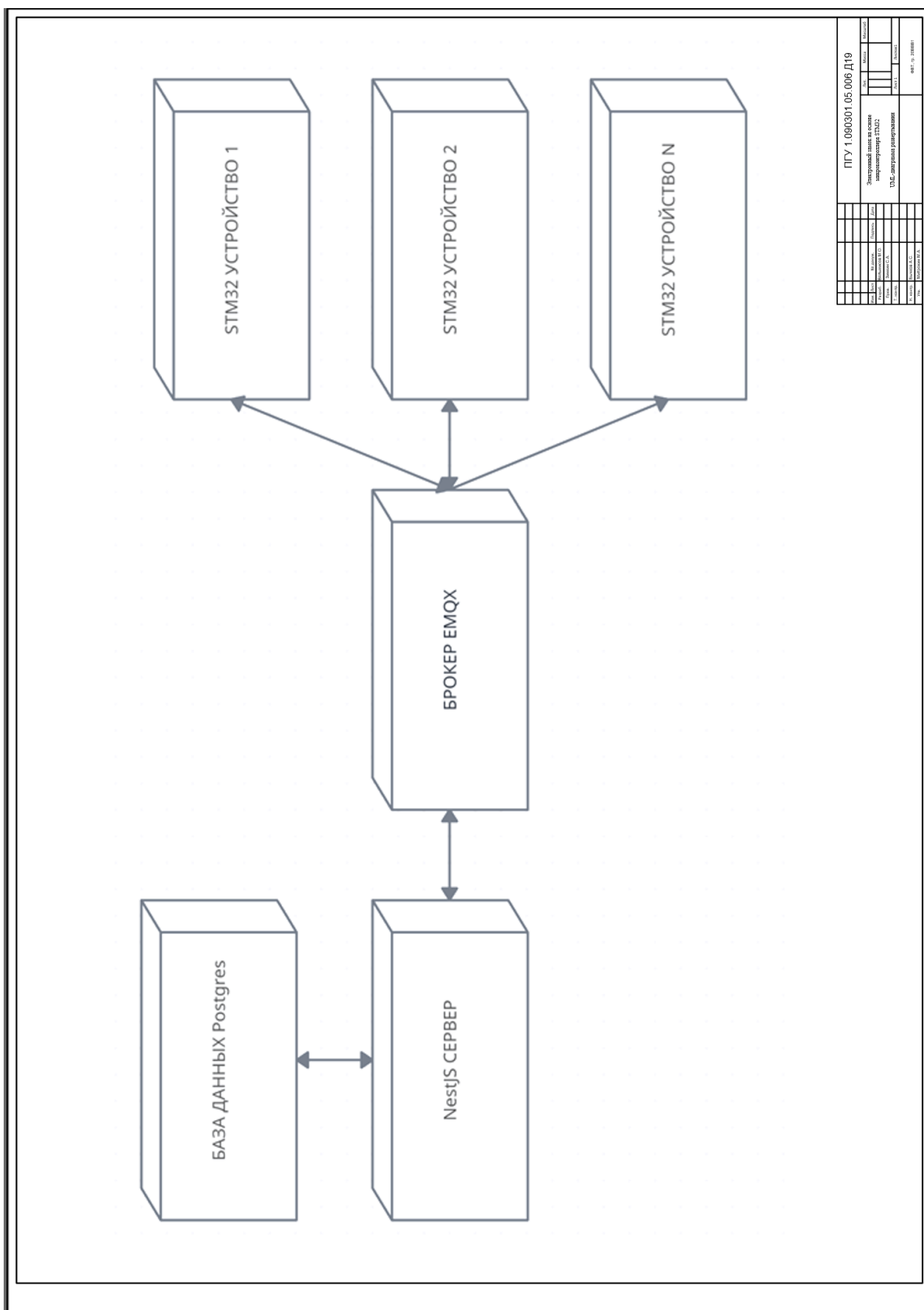
Лист

70

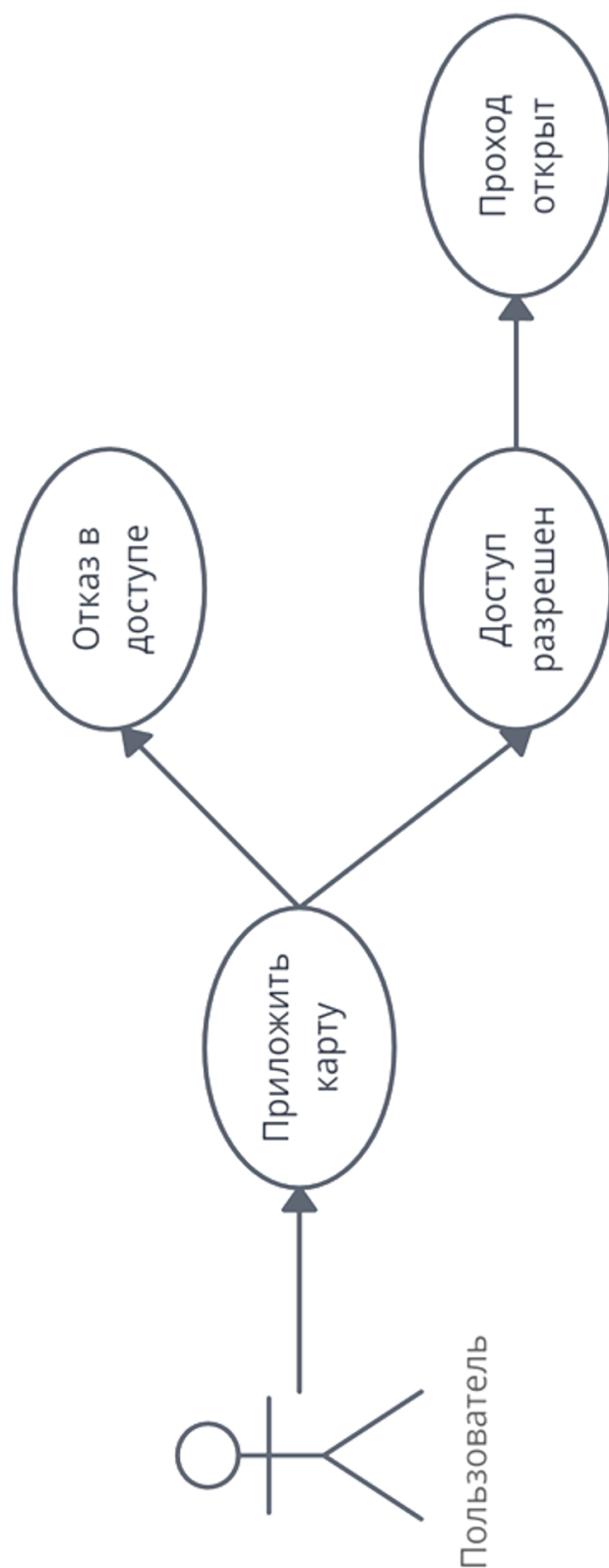
Приложение А.4 - Диаграмма классов модуля



Приложение А.5 – Диаграмма развертывания



Приложение А.6 – Диаграмма использования

[illegible]

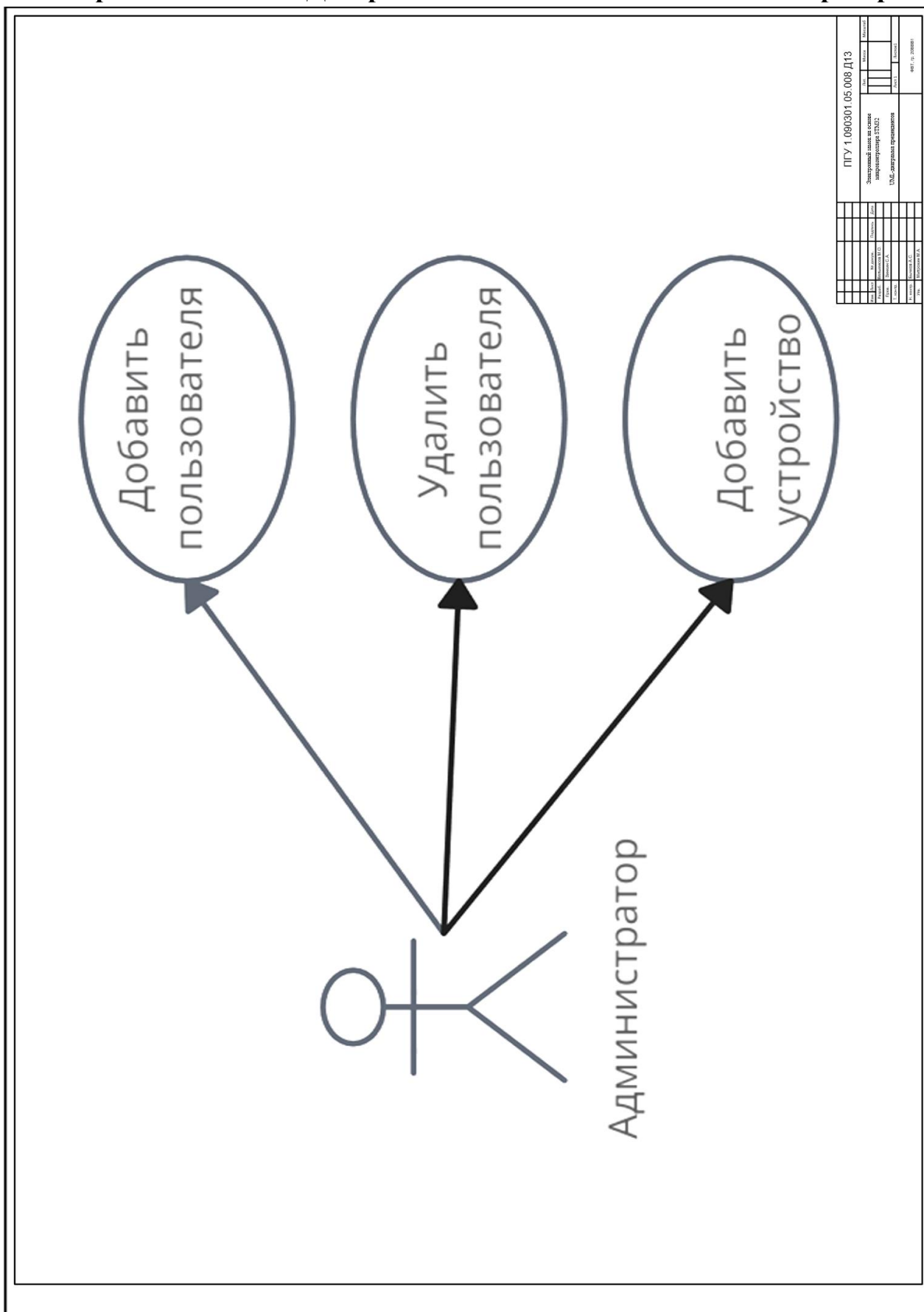
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

73

Приложение А.7 – Диаграмма использования для администратора



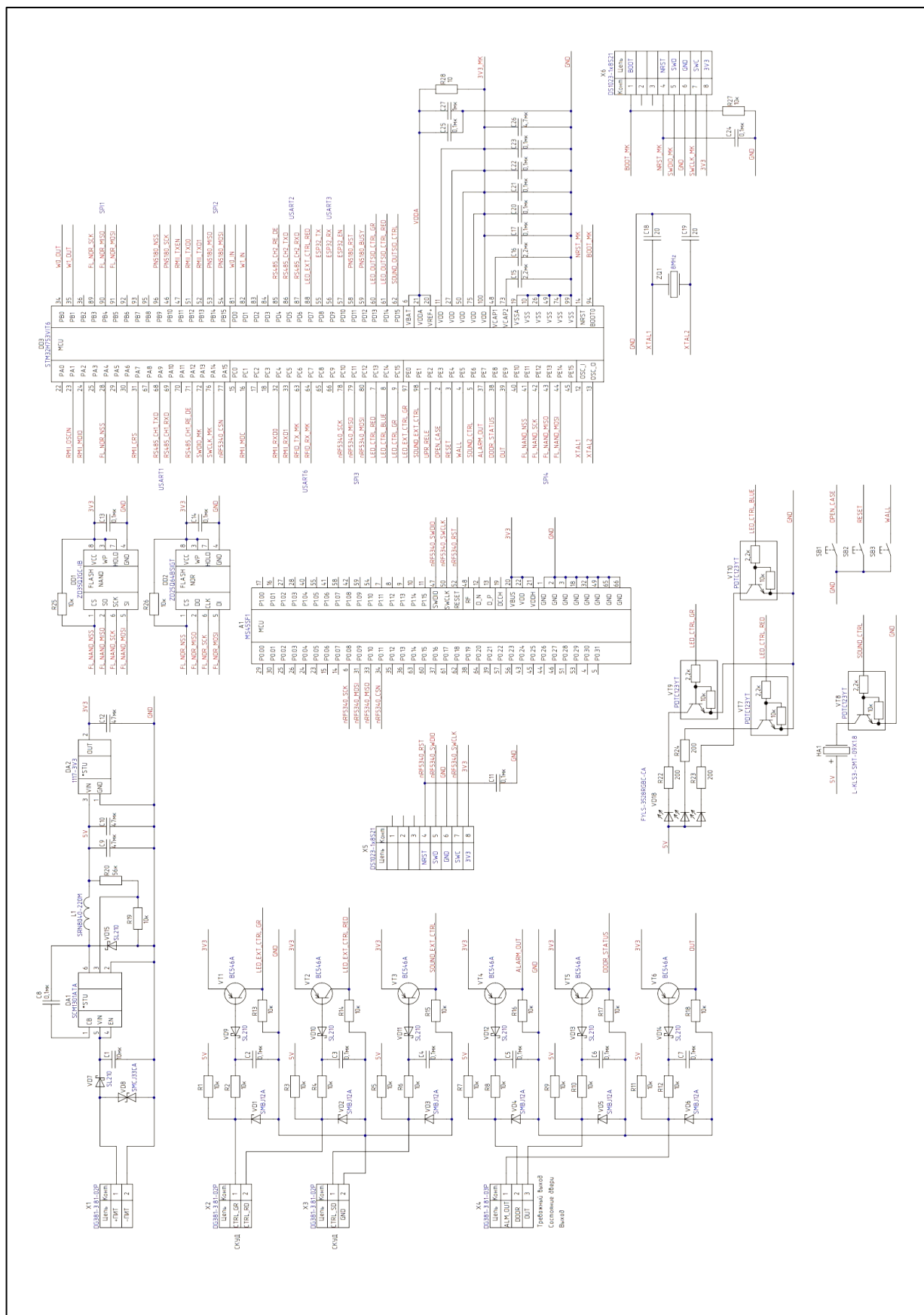
Приложение Б

(Обязательное)

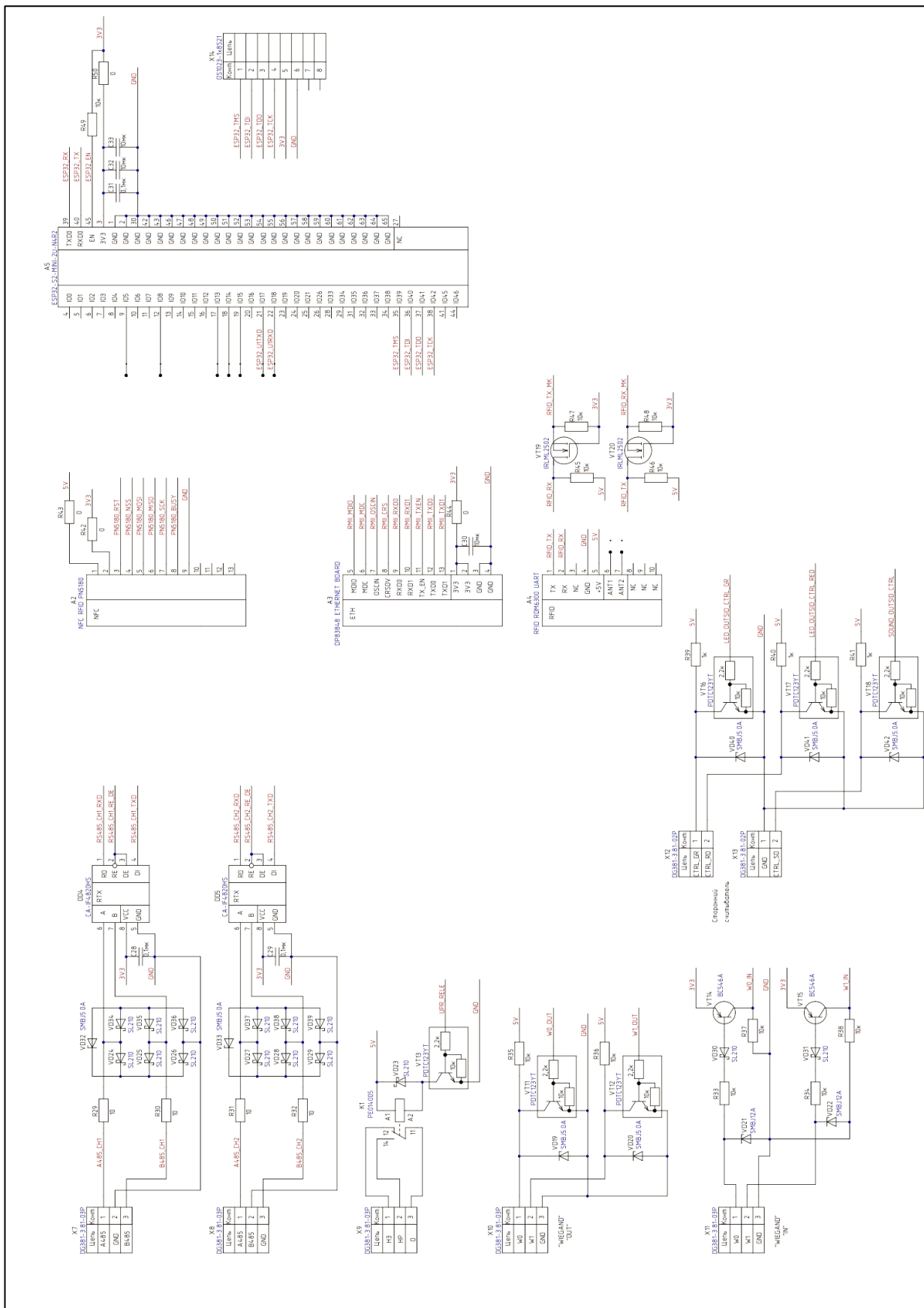
Электронный замок на основе микроконтроллера STM32

Рисунки схем

Приложение Б.1 – Рисунок схемы 1



Приложение Б.2 – Рисунок схемы 2



Приложение В

(Обязательное)

Электронный замок на основе микроконтроллера STM32

Листинг программы сервера

					ПГУ 1.090301.05.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

Файл Main.ts

```
import {NestFactory} from '@nestjs/core';
import {MqttModule} from './mqtt.module';

async function bootstrap() {
  const app = await NestFactory.create(MqttModule);
  await app.init();
}

bootstrap();
```

Файл mqtt.module.ts

```
import {Module} from '@nestjs/common';
import {ReaderModule} from '../reader/reader.module';
import {TestModule} from '../test/test.module';
import {NestMqttModule} from '../core/nest-mqtt/nest-mqtt.module';

@Module({
  imports: [
    ReaderModule,
    TestModule,
    NestMqttModule.forRoot({
      host: '127.0.0.1',
      queue: false,
      share: ''
    })
  ],
})
export class MqttModule {
}
```

Файл reader.module.ts

```
import {Module} from '@nestjs/common';
import {ReaderService} from '../reader.service';
import {ReaderController} from '../reader.controller';
import {UserModule} from '../user/user.module';

@Module({
  providers: [ReaderService],
  controllers: [ReaderController],
  imports: [
    UserModule,
  ],
})
export class ReaderModule {
}
```

Файл reader.controller.ts

```
import {Controller} from '@nestjs/common';
import {ReaderService} from '../reader.service';
import {ReaderEnterDto} from '../dto/reader-enter.dto';
import {InjectMqttClient, MqttPayload, MqttSubscribe} from
"../core/nest-mqtt/nest-mqtt.decorator";
import {NestMqttClient} from "../core/nest-mqtt/nest-
mqtt.client";

@Controller('reader')
export class ReaderController {
  constructor(private readonly readerService: ReaderService,
    @InjectMqttClient() private readonly
mqttService: NestMqttClient) {

    @MqttSubscribe({
      topic: 'reader/enter/',
      transform: 'json',
    })
    async readerEnter(@MqttPayload() readerEnterDto:
ReaderEnterDto) {
      const response = await
this.readerService.readerEnter(readerEnterDto);
      console.log(`For user ${readerEnterDto.userId} access is
${response}`);

      await
this.mqttService.publish(`reader/${readerEnterDto.deviceId}/acce
ss/`, response.toString());
    }
  }
}
```

Файл reader.service.ts

```
import {Injectable} from '@nestjs/common';
import {ReaderEnterDto} from '../dto/reader-enter.dto';
import {UserService} from '../user/user.service';
import {IS_ACCESS} from '../enum/IS_ACCESS';

@Injectable()
export class ReaderService {
  constructor(private readonly userService: UserService) {

    async readerEnter(readerEnterDto: ReaderEnterDto) {
      if (await
this.userService.haveUserAccess(readerEnterDto)) {
        return IS_ACCESS.ALLOWED;
      }

      return IS_ACCESS.DENIED;
    }
  }
}
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }
}

```

Файл reader-enter.dto.ts

```

import { IsNumber } from 'class-validator';

export class ReaderEnterDto {
  @IsNumber({}, {message: 'must be number'})
  readonly userId: number;

  @IsNumber({}, {message: 'must be number'})
  readonly deviceId: number;
}

```

Файл IS_ACCESS.ts

```

export enum IS_ACCESS { ALLOWED , DENIED}

```

Файл user.controller.ts

```

import { Controller } from '@nestjs/common';
import { UserService } from '../user.service';

@Controller('user')
export class UserController {
  constructor(private readonly userService: UserService) {}

}

import { Module } from '@nestjs/common';
import { UserController } from '../user.controller';
import { UserService } from '../user.service';

@Module({
  controllers: [UserController],
  providers: [UserService],
  exports: [
    UserService,
  ],
})
export class UserModule {}

Файл user.service.ts
import { Injectable } from '@nestjs/common';
import { ReaderEnterDto } from '../reader/dto/reader-enter.dto';

@Injectable()
export class UserService {
  async haveUserAccess(readerEnterDto: ReaderEnterDto) {
    return readerEnterDto.userId == 123;
  }
}

```

Файл user.module.ts

					ПГУ 1.090301.05.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

```
import { Module } from '@nestjs/common';
import { UserController } from '../user.controller';
import { UserService } from '../user.service';
```

```
@Module({
  controllers: [UserController],
  providers: [UserService],
  exports: [
    UserService,
  ],
})
export class UserModule {}
```

Файл test.controller.ts

```
import { Controller, Injectable } from '@nestjs/common';

@Controller('test')
export class TestController {
  constructor() {}
}
```

Файл test.module.ts

```
import { Module } from '@nestjs/common';
import { TestController } from '../test.controller';
import { TestService } from '../test.service';

@Module({
  controllers: [TestController],
  imports: [],
  providers: [
    TestService
  ],
})
export class TestModule {}
```

Файл test.service.ts

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class TestService {}
```

Файл validation.exception.ts

```
import { } from '@nestjs/common';
import { InvalidMessageException } from '@nestjs/microservices/errors/invalid-message.exception';

export class ValidationException extends InvalidMessageException{
```

```

    messages;

    constructor(response) {
        super();
        this.messages = response;
    }
}

```

Файл validation.pipe.ts

```

import { ArgumentMetadata, Injectable, PipeTransform } from
'@nestjs/common';
import { validate } from 'class-validator';
import { plainToClass } from 'class-transformer';
import { ValidationException } from
'../exceptions/validation.exception';

@Injectable()
export class ValidationPipe implements PipeTransform<any> {
    async transform(value: any, metadata: ArgumentMetadata):
Promise<any> {
        const obj = plainToClass(metadata.metatype, value);
        const errors = await validate(obj);

        if (errors.length) {
            let messages = errors.map(err => {
                return `${err.property} -
${Object.values(err.constraints).join(', ')}`;
            });
            throw new ValidationException(messages);
        }
    }
}

```

Файл nest-mqtt.module.ts

```

import { DynamicModule, Global, Module } from '@nestjs/common';
import { NestMqttExplorer } from '../nest-mqtt.explorer';
import { DiscoveryModule } from '@nestjs/core';
import {
    MqttModuleOptions,
} from '../nest-mqtt.interface';
import { MQTT_OPTION_PROVIDER } from '../nest-mqtt.constant';
import { NestMqttClient } from '../nest-mqtt.client';

@Global()
@Module({
    imports: [DiscoveryModule],
    exports: [NestMqttClient],
})
export class NestMqttModule {
    public static forRoot(options: MqttModuleOptions):
DynamicModule {

```

```

        return {
            module: NestMqttModule,
            providers: [
                {
                    provide: MQTT_OPTION_PROVIDER,
                    useValue: options,
                },
                NestMqttClient,
                NestMqttExplorer,
            ],
        };
    }
}

```

Файл nest-mqtt.client.ts

```

import {Inject, Injectable, Logger, OnModuleInit} from
"@nestjsjs/common";
import {MqttModuleOptions} from "../nest-mqtt.interface";
import {connect, IClientPublishOptions, IClientSubscribeOptions,
ISubscriptionGrant, MqttClient, Packet} from "mqtt";
import {MQTT_LOGGER, MQTT_OPTION_PROVIDER} from "../nest-
mqtt.constant";

@Injectable()
export class NestMqttClient implements OnModuleInit {
    private client: MqttClient;

    private readonly logger: Logger = new Logger(MQTT_LOGGER);

    constructor(@Inject(MQTT_OPTION_PROVIDER) private readonly
options: MqttModuleOptions) {
    }

    onModuleInit(): void {
        this.client = connect(this.options);

        this.client.on('connect', () => {
            this.logger.log('MQTT connected');
        });

        this.client.on('disconnect', packet => {
            this.logger.log('MQTT disconnected');
        });

        this.client.on('error', error => {
        });

        this.client.on('reconnect', () => {
            this.logger.log('MQTT reconnecting');
        });

        this.client.on('close', () => {

```

```

        this.logger.log('MQTT closed');
    });

    this.client.on('offline', () => {
        this.logger.log('MQTT offline');
    });
    this.logger.log('NestMqttClient initialized');
}

subscribe(topic: string | string[], opts?:
IClientSubscribeOptions): Promise<ISubscriptionGrant[]> {
    return new Promise((resolve, reject) => {
        this.client.subscribe(topic, opts || null, (err,
granted) => {
            if (err) {
                reject(err);
            } else {
                resolve(granted);
            }
        });
    });
}

unsubscribe(topic: string, opts?: Record<string, any>):
Promise<Packet> {
    return new Promise<Packet>((resolve, reject) => {
        // @ts-ignore
        this.client.unsubscribe(topic, opts || null, (error,
packet) => {
            if (error) {
                reject(error);
            } else {
                resolve(packet);
            }
        });
    });
}

publish(topic: string, message: string | Buffer | object,
opts?: IClientPublishOptions): Promise<Packet> {
    return new Promise<Packet>((resolve, reject) => {
        if (typeof message === 'object') {
            message = JSON.stringify(message);
        }
        this.client.publish(topic, message, opts || null,
(error, packet) => {
            if (error) {
                reject(error);
            } else {
                resolve(packet);
            }
        });
    });
}

```

```

    });
  }

  getClient() {
    return this.client;
  }
}

```

Файл nest-mqtt.constant.ts

```

export const SUBSCRIBE_OPTIONS = '__mqtt_subscribe_options';
export const SUBSCRIBER_PARAMS = '__mqtt_subscriber_params';
export const MQTT_OPTION_PROVIDER = 'MQTT_OPTION_PROVIDER';
export const MQTT_LOGGER = 'MQTT_LOGGER_PROVIDER';

```

Файл nest-mqtt.decorator.ts

```

import {CustomDecorator, Inject, SetMetadata} from
 '@nestjsjs/common';

import {
  MqttMessageTransformer,
  MqttSubscribeOptions,
  MqttSubscriberParameter,
} from './nest-mqtt.interface';
import {SUBSCRIBE_OPTIONS, SUBSCRIBER_PARAMS} from './nest-
mqtt.constant';
import {NestMqttClient} from './nest-mqtt.client';

export function InjectMqttClient() {
  return Inject(NestMqttClient);
}

export function MqttSubscribe(topic: string | string[] |
MqttSubscribeOptions): CustomDecorator;
export function MqttSubscribe(topicOrOptions): CustomDecorator {
  if (typeof topicOrOptions === 'string' ||
Array.isArray(topicOrOptions)) {
    return SetMetadata(SUBSCRIBE_OPTIONS, {
      topic: topicOrOptions,
    });
  } else {
    return SetMetadata(SUBSCRIBE_OPTIONS, topicOrOptions);
  }
}

function SetParameter(parameter:
Partial<MqttSubscriberParameter>) {
  return (
    target: object,
    propertyKey: string | symbol,
    paramIndex: number,
  ) => {
    const params =

```

```

        Reflect.getMetadata(SUBSCRIBER_PARAMS,
target[propertyKey]) || [];
        params.push({
            index: paramIndex,
            ...parameter,
        });
        Reflect.defineMetadata(SUBSCRIBER_PARAMS, params,
target[propertyKey]);
    };
}

export function MqttTopic() {
    return SetParameter({
        type: 'topic',
    });
}

export function MqttPacket() {
    return SetParameter({
        type: 'packet',
    });
}

export function MqttPayload(transform?: 'json' | 'text' |
MqttMessageTransformer) {
    return SetParameter({
        type: 'payload',
        transform,
    });
}

export function MqttParams() {
    return SetParameter({
        type: 'params',
    });
}

```

Файл nest-mqtt.explorer.ts

```

import {Inject, Injectable, Logger, OnModuleInit} from
'@nestjs/common';
import {DiscoveryService, MetadataScanner, Reflector} from
'@nestjs/core';
import {InstanceWrapper} from '@nestjs/core/injector/instance-
wrapper';

import {MqttClient} from 'mqtt';
import {Packet} from 'mqtt-packet';
import {getTransform} from './nest-mqtt.transform';
import {
    MqttModuleOptions,
    MqttSubscribeOptions,
    MqttSubscriber,

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

    MqttSubscriberParameter,
} from './nest-mqtt.interface';

import {InjectMqttClient} from './nest-mqtt.decorator";
import {MQTT_LOGGER, MQTT_OPTION_PROVIDER, SUBSCRIBE_OPTIONS,
SUBSCRIBER_PARAMS} from './nest-mqtt.constant";
import {NestMqttClient} from './nest-mqtt.client";

@Injectable()
export class NestMqttExplorer implements OnModuleInit {
    private readonly reflector = new Reflector();
    private readonly logger: Logger = new Logger(MQTT_LOGGER);
    private client: MqttClient;
    subscribers: MqttSubscriber[];

    constructor(
        private readonly discoveryService: DiscoveryService,
        private readonly metadataScanner: MetadataScanner,
        @InjectMqttClient() private readonly nestMqttClient:
NestMqttClient,
        @Inject(MQTT_OPTION_PROVIDER) private readonly options:
MqttModuleOptions,
    ) {
        this.subscribers = [];
    }

    onModuleInit() {
        this.client = this.nestMqttClient.getClient();
        this.logger.log('NestMqttModule dependencies
initialized');
        this.explore();
    }

    preprocess(options: MqttSubscribeOptions): string | string[]
{
        const processTopic = (topic) => {
            const queue = typeof options.queue === 'boolean' ?
options.queue : this.options.queue;
            const share = typeof options.share === 'string' ?
options.share : this.options.share;
            topic = topic.replace('$queue/', '')
                .replace(/^\$share\/([A-Za-z0-9]+)\/\//, '');
            if (queue) {
                return `$queue/${topic}`;
            }

            if (share) {
                return `$share/${share}/${topic}`;
            }

            return topic;
        };
};

```

```

        if (Array.isArray(options.topic)) {
            return options.topic.map(processTopic);
        } else {
            // this.logger.log(options.topic);
            return processTopic(options.topic);
        }
    }

    subscribe(options: MqttSubscribeOptions, parameters:
MqttSubscriberParameter[], handle, provider) {
        this.client.subscribe(this.preprocess(options), err => {
            if (!err) {
                // put it into this.subscribers;
                (Array.isArray(options.topic) ? options.topic :
[options.topic])
                    .forEach(topic => {
                        this.subscribers.push({
                            topic,
                            route: topic.replace('$queue/', '')
                                .replace(/^\$share\/([A-Za-z0-
9]+)\:\/\/, ''),
                            regexp:
NestMqttExplorer.topicToRegex(topic),
                            provider,
                            handle,
                            options,
                            parameters,
                        });
                    });
            } else {
                this.logger.error(
                    `subscribe topic [{options.topic} failed]`,
                );
            }
        });
    }

    explore() {
        const controllers: InstanceWrapper[] =
this.discoveryService.getControllers();
        controllers.forEach((wrapper: InstanceWrapper) => {
            const {instance} = wrapper;
            if (!instance) {
                return;
            }
            this.metadataScanner.scanFromPrototype(
                instance,
                Object.getPrototypeOf(instance),
                key => {
                    const subscribeOptions: MqttSubscribeOptions
= this.reflector.get(
                        SUBSCRIBE_OPTIONS,

```

```

        instance[key],
    );
    const parameters = this.reflector.get(
        SUBSCRIBER_PARAMS,
        instance[key],
    );
    if (subscribeOptions) {
        this.subscribe(subscribeOptions,
parameters, instance[key], instance);
    }
    },
    );
});

this.client.on(
    'message',
    (topic: string, payload: Buffer, packet: Packet) =>
{
    const subscriber = this.getSubscriber(topic);
    if (subscriber) {
        const parameters = subscriber.parameters ||
[];
        const scatterParameters:
MqttSubscriberParameter[] = [];
        for (const parameter of parameters) {
            scatterParameters[parameter.index] =
parameter;
        }
        try {
            const transform =
getTransform(subscriber.options.transform);

            if (this.options.beforeHandle) {
                this.options.beforeHandle(topic,
payload, packet);
            }

            subscriber.handle.bind(subscriber.provider) (
                ...scatterParameters.map(parameter
=> {
                    switch (parameter?.type) {
                        case 'payload':
                            return
transform(payload);

                        case 'topic':
                            return topic;
                        case 'packet':
                            return packet;
                        case 'params':
                            return
NestMqttExplorer.matchGroups(topic, subscriber.regexp);

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

        default:
            return null;
        }
    })),
    );
    } catch (err) {
        this.logger.error(err);
    }
}

},

);

}

private getSubscriber(topic: string): MqttSubscriber | null
{
    for (const subscriber of this.subscribers) {
        subscriber.regexp.lastIndex = 0;
        if (subscriber.regexp.test(topic)) {
            return subscriber;
        }
    }
    return null;
}

private static topicToRegex(topic: string) {
    // compatible with emqtt
    return new RegExp(
        '^' +
        topic
            .replace('$queue/', '')
            .replace(/^$share\/([A-Za-z0-9]+)\/$/, '')
            .replace(/([\[\]\?\\(\)\$\^\*\.|])/g, '\\$1')
            .replace(/\+/g, '([^\+] +)')
            .replace(/\/#$/ , '(/.*)?') +
        '$',
        'y',
    );
}

private static matchGroups(str: string, regex: RegExp) {
    regex.lastIndex = 0;
    let m = regex.exec(str);
    const matches: string[] = [];

    while (m !== null) {
        // This is necessary to avoid infinite loops with
        zero-width matches
        if (m.index === regex.lastIndex) {
            regex.lastIndex++;
        }
    }
}

```

```

        // The result can be accessed through the `m`-
variable.
        m.forEach((match, groupIndex) => {
            if (groupIndex !== 0) {
                matches.push(match);
            }
        });
        m = regex.exec(str);
    }
    return matches;
}
}

```

Файл nest-mqtt.interface.ts

```

import {IClientOptions, Packet} from 'mqtt';

export type MqttMessageTransformer = (payload: Buffer) => any;

export interface MqttSubscribeOptions {
    topic: string | string[];
    queue?: boolean;
    share?: string;
    transform?: 'json' | 'text' | MqttMessageTransformer;
}

export interface MqttSubscriberParameter {
    index: number;
    type: 'payload' | 'topic' | 'packet' | 'params';
    transform?: 'json' | 'text' | MqttMessageTransformer;
}

export interface MqttSubscriber {
    topic: string;
    handle: any;
    route: string;
    provider: any;
    regexp: RegExp;
    options: MqttSubscribeOptions;
    parameters: MqttSubscriberParameter[];
}

export interface MqttModuleOptions extends IClientOptions {
    queue?: boolean;

    share?: string;

    beforeHandle?: (topic: string, payload: Buffer, packet:
Packet) => any;
}

```

Файл nest-mqtt.transform.ts

```

import {MqttMessageTransformer} from './nest-mqtt.interface';

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

export const JsonTransform: MqttMessageTransformer = payload =>
{
    return JSON.parse(payload.toString('utf-8'));
};

export const TextTransform: MqttMessageTransformer = payload =>
{
    return payload.toString('utf-8');
};

export function getTransform(
    transform: 'json' | 'text' | MqttMessageTransformer,
) {
    if (typeof transform === 'function') {
        return transform;
    } else {
        if (transform === 'text') {
            return TextTransform;
        } else {
            return JsonTransform;
        }
    }
}

```

Файл nest-cli.json

```

{
  "$schema": "https://json.schemastore.org/nest-cli",
  "collection": "@nestjs/schematics",
  "sourceRoot": "src",
  "compilerOptions": {
    "deleteOutDir": true
  }
}

```

Файл package.json

```

{
  "name": "nest-mqtt-client",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\"",
    "test": "jest",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
  }
}

```

```

    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register
-r ts-node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^10.0.0",
    "@nestjs/core": "^10.0.0",
    "@nestjs/microservices": "^10.3.8",
    "@nestjs/platform-express": "^10.0.0",
    "@nestjs/swagger": "^7.3.1",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.1",
    "mqtt": "^5.5.3",
    "reflect-metadata": "^0.2.0",
    "rxjs": "^7.8.1"
  },
  "devDependencies": {
    "@nestjs/cli": "^10.0.0",
    "@nestjs/schematics": "^10.0.0",
    "@nestjs/testing": "^10.0.0",
    "@types/express": "^4.17.17",
    "@types/jest": "^29.5.2",
    "@types/node": "^20.3.1",
    "@types/supertest": "^6.0.0",
    "@typescript-eslint/eslint-plugin": "^6.0.0",
    "@typescript-eslint/parser": "^6.0.0",
    "eslint": "^8.42.0",
    "eslint-config-prettier": "^9.0.0",
    "eslint-plugin-prettier": "^5.0.0",
    "jest": "^29.5.0",
    "prettier": "^3.0.0",
    "source-map-support": "^0.5.21",
    "supertest": "^6.3.3",
    "ts-jest": "^29.1.0",
    "ts-loader": "^9.4.3",
    "ts-node": "^10.9.1",
    "tsconfig-paths": "^4.2.0",
    "typescript": "^5.1.3"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "json",
      "ts"
    ],
    "rootDir": "src",
    "testRegex": ".*\\.spec\\.ts$",
    "transform": {
      "^.+\\.?(t|j)s?$": "ts-jest"
    }
  }
}

```

```

    },
    "collectCoverageFrom": [
      "**/*.@(t|j)s"
    ],
    "coverageDirectory": "../coverage",
    "testEnvironment": "node"
  }
}

```

Файл tsconfig.json

```

{
  "compilerOptions": {
    "module": "commonjs",
    "declaration": true,
    "removeComments": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "allowSyntheticDefaultImports": true,
    "target": "ES2021",
    "sourceMap": true,
    "outDir": "./dist",
    "baseUrl": "./",
    "incremental": true,
    "skipLibCheck": true,
    "strictNullChecks": false,
    "noImplicitAny": false,
    "strictBindCallApply": false,
    "forceConsistentCasingInFileNames": false,
    "noFallthroughCasesInSwitch": false
  }
}

```

Приложение Г

(Обязательное)

Электронный замок на основе микроконтроллера STM32

Презентация

					ПГУ 1.090301.05.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		



Выпускная квалификационная работа бакалавра

Электронный замок на основе микроконтроллера STM32

Hands FREE

Автор работы: Вяльмисов М. О.

Научный руководитель: профессор д.т.н Зинкин С. А.



О проекте

Проблема

Управление контролем доступа в помещение и на территорию



Задача

Разработать часть СКУД системы - программу для электронного замка и серверного обеспечения



Ценность

Универсальное решение с полноценной интеграцией системы

Целью работы являлось реализация следующих функций:

✓ Считывание метки



✓ Отправка метки на сервер

✓ Получение ответа и его обработка

✓ Получение данных о пользователе из БД

✓ Обработка запроса на сервере

3

Выбор решения

Код

C
Typescript

Фреймворки

NodeJs
NestJs

Протокол

MQTT

СУБД

Postgres

Среды разработки

STM32CubeIDE
WEBStorm

Библиотеки

HAL
FreeRTOS
Sequelize

Другие инструменты

DBViewer
Git
docker



4

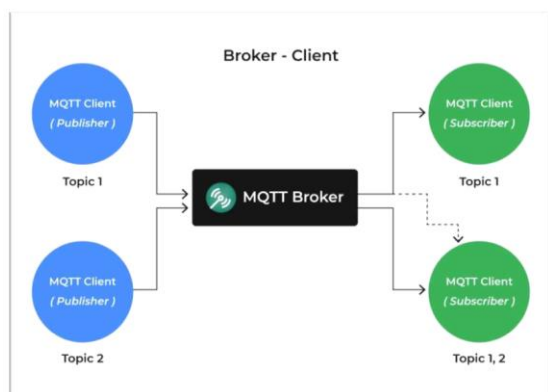
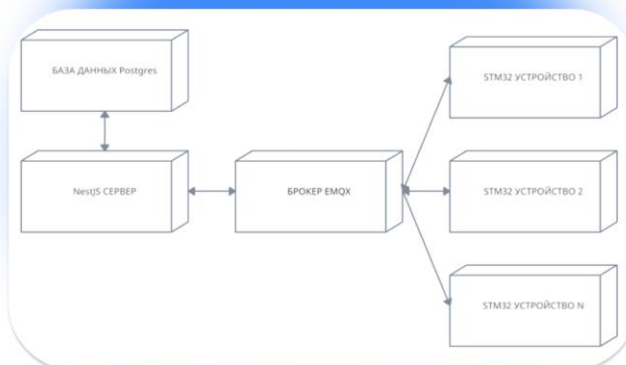
Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.090301.05.001 ПЗ

Лист

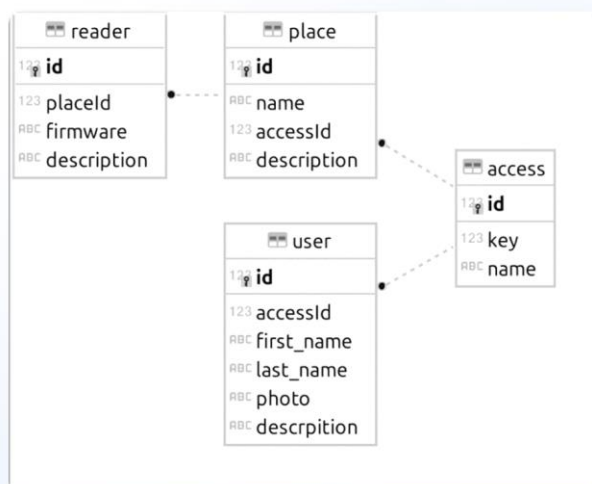
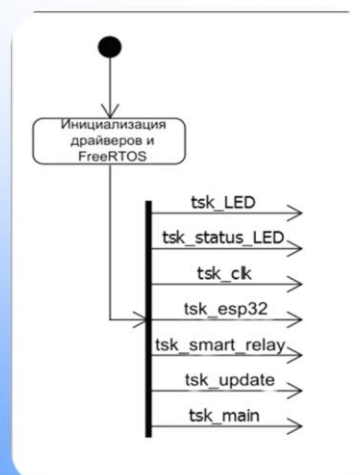
98

Архитектура проекта

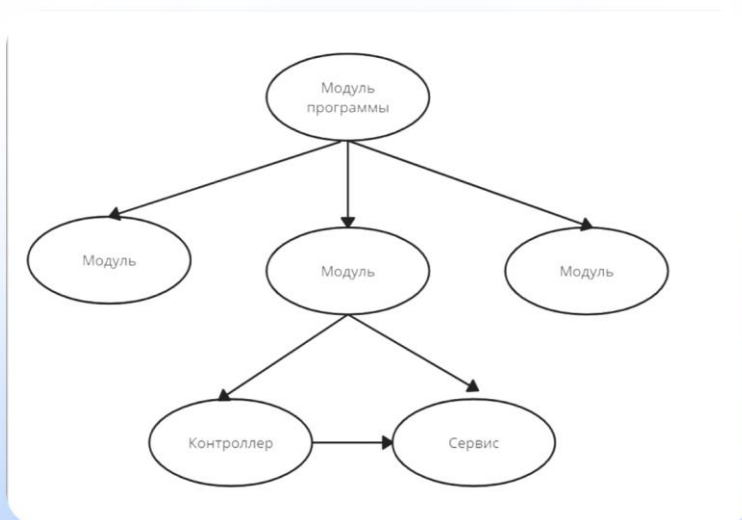


5

Процесс разработки



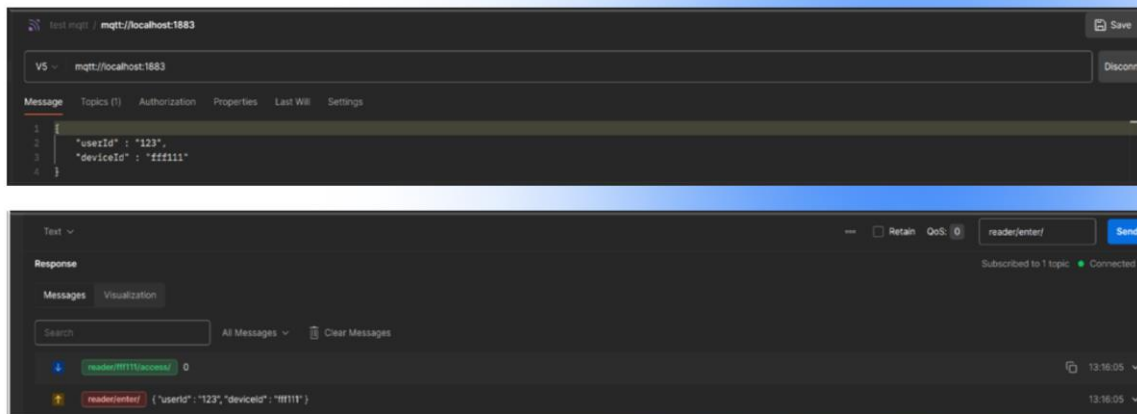
6



7

Тестирование

Тестирование проекта производилось с помощью специального ПО – Postman



8

Подведение ИТОГОВ

- ✓ В рамках проекта были разработаны: программа клиент для STM32 и сервер на NestJS развернутый в docker. Также была спроектирована база данных Postgres.

					ПГУ 1.090301.05.001 ПЗ	Лист
						101
Изм.	Лист	№ докум.	Подпись	Дата		