

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии.

Работа допущена к защите

Директор ВШ ПИ

_____ П.Д.Дробинцев

« ____ » _____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

работа бакалавра

РАЗРАБОТКА СЕРВЕРНОГО ПРИЛОЖЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ «УМНАЯ ТЕПЛИЦА»

по направлению подготовки (специальности) 09.03.04 Программная инженерия

Направленность (профиль) 09.03.04_03 Разработка программного обеспечения

Выполнил
студент гр. в3530904/80321

И.Е. Колоянов

Руководитель
старший преподаватель

О.В. Прокофьев

Консультант
по нормоконтролю

Е. Г. Локшина

Санкт-Петербург

2021

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕР-
СИТЕТ ПЕТРА ВЕЛИКОГО**

Высшая школа программной инженерии

УТВЕРЖДАЮ

Руководитель ОП А. В. Петров

«__» _____ 2021 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Коляянову Ивану Евгеньевичу, в3530904/80321

фамилия, имя, отчество (при наличии), номер группы

1. Тема работы: Разработка серверного приложения информационной системы “Умная теплица”.
2. Срок сдачи студентом законченной работы:
3. Исходные данные по работе:
 1. документация к языку программирования Java.
URL: <https://docs.oracle.com/en/java/>
(дата обращения 20.05.2021) ;
 2. документация к фреймворку Spring.
URL: <https://docs.spring.io/spring-integration/docs/current/reference/html>
(дата обращения 20.05.2021).
4. Содержание работы (перечень подлежащих разработке вопросов): основы разработки серверной части веб-приложения, исследование функциональных возможностей фреймворка “Spring”, исследование работы MQTT протокола.
5. Перечень графического материала (с указанием обязательных чертежей):
6. Консультанты по работе:
7. Дата выдачи задания:

Руководитель ВКР _____

Прокофьев О.В.

(подпись) инициалы, фамилия

Задание принял к исполнению

Студент _____

Коляянов И. Е.

(подпись) инициалы, фамилия

РЕФЕРАТ

На 51 с., 19 рисунков, 9 таблиц, 0 приложений.

ТЕПЛИЦА, АВТОМАТИЗАЦИЯ, ИНТЕРНЕТ ВЕЩЕЙ, JAVA, MQTT, SPRING, MOSQUITO

В данной работе проводится разработка программного обеспечения для автоматизации работы умной теплицы с использованием технологий интернета вещей (IoT). В работе рассмотрены теоретические аспекты, направленные на предметную область, связанную с понятием «умная теплица», и практические аспекты, включающие в себя разработку программной части. В качестве веб-сервера для общения с клиентами для мониторинга данных в режиме реального времени, было разработано приложение на Java фреймворке Spring, которое взаимодействует с базой данных PostgreSQL. Для общения с брокером, который выполняет роль посредника между сервером и аппаратной частью, была добавлена поддержка MQTT протокола, с помощью которого происходит общение.

THE ABSTRACT

51 pages, 19 pictures, 9 tables, 0 application

GREENHOUSE, AUTOMATION, IOT, JAVA, MQTT, SPRING,
MOSQUITO

In this work, software is being developed to automate the work of a smart greenhouse using Internet of Things (IoT) technologies. The work considers theoretical aspects aimed at the subject area associated with the concept of "smart greenhouse", and practical aspects, including the development of the software part. As a web server for communicating with clients to monitor data in real time, an application was developed in the Spring Java framework that interacts with a PostgreSQL database. To communicate with the broker, which acts as an intermediary between the server and the hardware, support for the MQTT protocol has been added, through which communication takes place.

Содержание

Введение	7
Постановка задачи	10
Глава 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ	11
1.1. Теплица	11
1.2. Интернет вещей	13
1.3. Существующие решения для автоматизации теплиц.....	15
1.4. Обзор существующих способов контроля и управления теплицей	18
1.4.1 Проветривание	18
1.4.2 Влажность и полив	22
1.4.3 Освещение	26
Глава 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА	31
2.1. Архитектура	31
2.2. Выбор средств реализации.....	33
2.3. Проектирование базы данных	34
2.4. Проектирование веб-сервера	40
2.5. Общение с брокером через MQTT протокол	45
Заключение	50
Список использованных источников	51

ВВЕДЕНИЕ

Выращивание растений при помощи теплиц – это пространственный способ упрощения работ с овощными культурами в сельском хозяйстве. Плюсы данной технологии заключаются не только в том, что растения вырастают в закрытом грунте из-за чего уменьшается риск попадания на грунт насекомых или растений, которые могут помешать нормальному росту овощей, но и также в том, что помещение в теплицах можно отапливать, что благоприятно скажется на выращивании культур, которые сильно зависимы от температуры, в которой они созревают.

В современный век высоких технологий нередко можно встретить теплицы площадью больше 100 тысяч квадратных километров, что уж говорить о бесчисленном множестве теплиц с куда меньшим помещением для созревания.

Но не так давно стала развиваться индустрия “Умных теплиц”. Что бы теплица из обычной вдруг стала умной в неё добавляют технологии, которые самостоятельно и почти без помощи человека могут заниматься регулированием температуры, влажности или даже освещения в помещении, в котором выращиваются растения.

Список вещей, которые можно привнести в теплицу при помощи технологий, до сих пор расширяется, и для того, чтобы как-то систематизировать технологическое управление в отдельный модуль было создано такое понятие как “Интернет вещей”. Интернет вещей объединяет устройства в компьютерную

сеть и позволяет им собирать, анализировать, обрабатывать и передавать данные другим объектам через программное обеспечение, приложения или технические устройства.

Во всероссийском каталоге цифровых решений можно найти множество проектов, которые направлены на автоматизацию работ теплиц при помощи интернета вещей. Автоматизировано множество вещей: от чего-то банального, как полив и проверка температуры, так и более сложные элементы работы, как управление потреблением энергии или автоматическое приготовление растворов и пестицидов для обработки земли в зависимости от текущего состояния здоровья некоторых растений. Также почти все решения поставляются с уже готовыми к настройке элементами управления и датчиками.

Из предыдущего абзаца можно сделать вывод о том, что все доступные решения на данный момент выгодно использовать в теплицах огромных размеров для промышленного производства. Для частных теплиц куда более меньших размеров, которые весьма распространены на постсоветском пространстве, данные решения использовать неуместно.

Если с приобретением аппаратной части, на данный момент, не возникает проблем, то вот с программным обеспечением, которое будет реализовывать работу IoT, все не так хорошо.

Можно выделить пару популярных ПО, которые можно использовать для умных теплиц. Это “Home Assistant” и “Node-RED”.

Home Assistant – распространённое открытое ПО для настройки и получения данных элементов управления или датчиков. Хороший вариант для устройств, находящихся внутри домов или квартир, но вот для теплиц не самое удачное решение, т. к. не работает с ардуино, да и более-менее адекватной документации для настройки элементов управления, отвечающих за поливы или влажность почвы не было найдено.

Node-RED – инструмент потокового программирования, который работает в среде исполнения Node.js. Как ПО для настройки умной теплицы, почти идеальный вариант. Из минусов можно выделить отсутствие документации на русском языке, что может стать критичным для некоторых пользователей постсоветского пространства, которые решат самостоятельно заняться настройкой своей умной теплицы. Также, в отличие от предыдущего примера, имеет довольно трудный для понимания интерфейс, в котором довольно трудно разобраться не имея более-менее продвинутых навыков работы с ПК и ПО.

ПОСТАНОВКА ЗАДАЧИ

Из сказанного выше, можно сделать вывод, что нужно разработать такой программный продукт, который будет иметь понятный интерфейс пользователя для работы с аппаратной частью теплиц и реализовывать работу с данными, которые были получены во время использования теплицей разных устройств.

Цель данной работы – серверной части для общения с клиентской частью. Данная работа включает в себя следующие задачи:

1. Разработка серверной части на языке Java с помощью фреймворка Spring.
2. Настройка брокера Mosquito с помощью фреймворка Spring.
3. Настройка общения сервера и брокера с помощью MQTT протокола.
4. Работа с данными посредством базы данных Postgress.
5. Общение с клиентской частью с помощью http запросов.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Теплица

Не секрет, что уход за теплицей требует у своего владельца львиную долю времени и сил. В современном мире, где у многих не хватает времени на выполнение всех необходимых задач вовремя, автоматизация процессов высоко оценивается, так как позволяет сократить время- и трудо- затраты человека. Автоматизация работы и управления теплицей позволит ее владельцу сократить время, затрачиваемое на уход за теплицей, так как отслеживание значений, таких как температура, влажность воздуха, освещенность, могут регистрировать датчики, а такие минорные задачи, как полив и проветривание, можно доверить контроллерам на автоматический запуск при выполнении определенных условий. Суть умной теплицы состоит в объединении всех этих возможностей в одну систему, предоставление возможности отслеживать удаленно все происходящие в теплице процессы и, при необходимости, менять заданные настройки.

В подобных системах данные регистрируются датчиками, отправляются на сервер для хранения посредством брокера, взаимодействующего с аппаратной частью, и в последствии предоставляются визуально пользователю посредством клиентского приложения. Таким образом система умной теплицы обеспечивает непрерывный мониторинг данных, обеспечивает их хранение и автоматически запускает процессы ухода за культурами, выращиваемыми в теплице.

Разберем подробнее преимущества системы умной теплицы:

- Снижение затрат

Автоматизация теплицы является отличным решением экономии времени и трудозатрат. Владельцу теплицы необходимо настроить систему и задать алгоритм действий для реагирования системы в определенных ситуациях. После этого она будет работать самостоятельно и непрерывно, отсылая владельцу системы актуальные данные о состоянии теплицы. Отправка данных может быть реализована множеством вариантов: от сообщений на телефон до визуализации посредством клиентского приложения.

- Улучшение качества готового продукта

Один из важнейших пунктов – повышение качества итогового продукта. Система умной теплицы поддерживает идеальные условия, заданные владельцем системы, без перерывов, реагируя на изменения показателей теплицы. Показатели, такие как температура, регистрируются датчиками и отправляются на сервер через брокера.

Для ухода за растениями может использоваться система автоматического полива, которая позволяет предотвратить недостаточное орошение культур. Также благодаря завязке на систему умной теплицы, можно разбить пространство внутри теплицы на зоны и ухаживать за ними по разному расписанию и параметрам. Это довольно актуальная функциональность, т. к. часто встречаются ситуации выращивания в одной, пусть даже маленькой, теплице разных культур в сравнительной близости. Разбитие на зоны и гибкая настройка параметров для каждой

зоны делает систему умной теплицы еще более полезной для владельца.

- **Хранение данных**

Данные собираются с определенным интервалом, заданным владельцем теплицы, с помощью установленных в теплице датчиков. Все полученные значения в последствии хранятся в базе данных и доступны для просмотра и сбора статистики, в случае возникновения такой необходимости. К примеру, благодаря хранению собранной с датчиков информации, становится проще найти проблемы плохого роста растений в теплице, если подобная ситуация вдруг возникнет.

Можно сделать заключение, что система умной теплицы может помочь свести к минимуму трудозатраты ее владельца, сэкономить время на уходе за теплицей и, возможно, даже повысить качество получаемого продукта. Автоматизированная система умной теплицы является эффективным решением для наблюдения за показателями теплицы, выявления проблем роста растений и улучшения получаемого урожая.

1.2. Интернет вещей

Интернет вещей (IoT – Internet of Things) – концепция передачи данных между устройствами. Если описывать абстрактно, то внутри интернета вещей люди могут общаться с вещами, а вещи – общаться между собой.

Интернет вещей устроен таким образом, что позволяет объединять устройства в одну сеть, позволяет им работать с данны-

ми и передавать их через посредническое ПО. Устройства внутри IoT функционируют самостоятельно, люди могут задавать им настройки и предоставлять доступ к данным. IoT системы, чаще всего, работают в режиме реального времени. Сначала устройства собирают данные, затем данные отправляются в место хранения, полученные результаты обрабатываются системой и принимается решение, как нужно отреагировать на полученные данные.

IoT используется повсеместно во многих сферах, таких как: транспорт, здравоохранение, сельское хозяйство, логистика – с целью автоматизации процессов и снижения трудозатрат. Такой подход улучшает качество предоставляемых услуг и удешевляет процесс производства.

При прочтении описания IoT у многих выстроится ассоциация с умным домом и не просто так, это всем известный и понятный пример реализации интернета вещей.

Однако, конечно, у IoT есть и свои недостатки. Основной проблемой является безопасность. Устройства, являющиеся частью IoT, постоянно подвергаются атакам киберпреступников, поэтому при разработке своей системы IoT важно суметь ее грамотно защитить.

Если верить прогнозам развития IoT, через 5 лет будет насчитываться примерно 55,7 млрд подключенных устройств. Атаки киберпреступников продолжатся, т. к. IoT является для них быстрым способом распространить вредоносное ПО. Использование интеллектуальных технологий будет только расти,

т. к. они позволяют экономить как время, так и финансы. На данный же момент технология IoT хоть и довольно развита, но все же недостаточно стандартизирована, что делает трудным интеграцию умных решений между собой.

1.3. Существующие решения для автоматизации теплиц

Существует несколько видов автоматизированных теплиц: частная и промышленная.

Частная автоматизированная теплица – система собранная самостоятельно кем-либо. Соответствует следующим характеристикам: автоматическая регулировка температуры в теплице за счет датчиков температуры воздуха, обязательное наличие капельной системы орошения, грунт в теплице восстанавливается без вмешательства человека.

Основной аспект автоматизированной теплицы заключается в том, что все установленные системы функционируют согласованно.

Можно выделить несколько видов теплиц по способу их автоматизации:

- Гидравлическая автоматизация теплицы. Подразумевает под собой расширение жидкости при нагреве. Используется в доводчиках с термоприводами.
- Биметаллическая автоматизация теплицы. Подразумевает под собой способность различных металлов к расширению. Часто используется для автоматизации систем вентилирования.

- Электрическая автоматизация теплицы. Довольно просто монтируется и имеется возможность точной настройки. Подразумевает разных электронных модулей для обслуживания теплицы.

Промышленная автоматизация теплиц регулирует множество моментов от простой регулировки вентиляции до системы искусственного освещения или подкормки рассады. Границы возможности регулируется индивидуально под нужды каждой отдельной теплицы.

Таблица 1.1

Существующие решения автоматизации теплицы

Название	Поставляе- мые в ком- плекте дат- чики	Возмож- ность до- бавлять в систему сторонние датчики	Приклад- ное кли- ентское ПО	Лицензия на ПО
Умная теплица компаний ТЕХНОСЕРВ	Датчики для мониторинга почвы, мик- роклимата и погодных условий.	Не выяв- лена	Присут- ствует	Коммерче- ская
Программно- аппаратный комплект “IOTICA” компаний ALAN	Датчики температу- ры, влажно- сти, давле- ния, питания сети, затоп- ления, освещенно- сти, дыма	Присут- ствует	Присут- ствует	Коммерче- ская
Комплект ав- томатизации для умных теплиц “Умни- ца”	Датчики температу- ры, освещенности, влажности, атмосферно- го давления	Не выяв- лена	Отсутству- ет	-
Наш проект	Датчики влажности, освещения, температуры	Присут- ствует	Присут- ствует	Для неком- мерческого использова- ния

1.4. Обзор существующих способов контроля и управления теплицей

1.4.1 Проветривание

Вентиляция теплицы необходима по нескольким причинам:

1. Даже в прохладную погоду температура в теплице может быть очень высокой, что препятствует нормальному росту и развитию растений. Холод растениям гораздо менее вреден чем излишняя жара. Не все растения хорошо переносят жару. Некоторые могут перестать расти, могут начать увядать или даже погибнуть. С помощью вентиляции можно добиться выравнивания температуры в теплице с помощью вывода излишнего тепла.
2. Кислород и углекислый газ необходимые элементы в процессе фотосинтеза. Без хорошего проветривания в герметично закрытой теплице невозможно добиться постоянного притока свежего воздуха, из-за чего растениям не хватает питательных веществ. Правильная вентиляция позволит растениям как укрепить свои стебли, так и корни.
3. Герметично закрытые теплицы являются рассадниками вредителей, они не чувствуют какой-либо угрозы и начинают бесконтрольно питаться и размножаться. Приток свежего воздуха может негативно сказаться на них.
4. Хорошее проветривание так же является залогом хорошего опыления. Хорошая вентиляция не только

помогает само опыляемым растениям за счет поступления свежего воздуха, но также позволяет проникнуть в теплицу пчелам, которые тоже будут заниматься опылением растений.

Существует несколько способов проветривания помещений:

1. Установка доводчика с термоприводом. Состоит устройство, изображенное на рис. 1, из самого термопривода, доводчика, кронштейна и крепежных элементов. В стальном корпусе термопривода находится жидкость, которая расширяется при повышении температуры и сжимается при её понижении. Из-за чего при достижении температуры, скажем, 30 градусов жидкость вытолкнет доводчик на максимальную длину, благодаря чему тот откроет окно или дверь. Точно также при опускании температуры до 20 градусов доводчик вернется в изначальное состояние, так как жидкость ему больше не мешает, тем самым закрыв окно или дверь;



Рис. 1. Доводчик с термоприводом

2. Комплект автоматизированного проветривания, изображенный на рис. 2. В таком комплекте есть несколько элементов:

- а. Блок управления – нужен для измерения температуры и передачи приводам команды на открытие и закрытие окна или двери.
- б. Блок питания для работы блока управления от сети.
- с. Приводы, которые будут открывать и закрывать двери или окна.

Второй вариант более гибок, так как все открывающиеся механизмы будут открыты одновременно благодаря блоку управления, в отличие от доводчиков, которые могут открывать и закрывать окна и двери в зависимости от температуры в том месте, где они находятся. Но доводчики обходятся гораздо дешевле чем полно-

ценный комплект для автоматизированного проветривания.

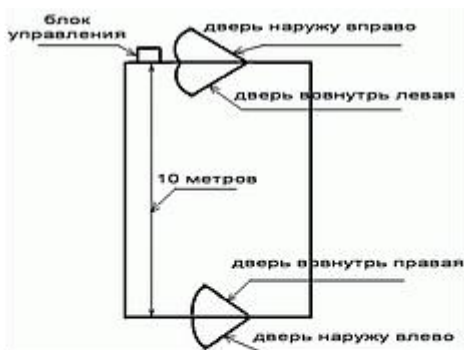


Рис. 2. Комплект автоматизированного проветривания

1.4.2 Влажность и полив

Растениям и урожаю нужен регулярный и достаточно объёмный полив, без которого они не смогут расти и просто погибнут. Но даже обычный уход за растениями можно автоматизировать с помощью технологий, основанных на электронике.

Автоматизация полива – это технический комплекс, который обеспечивает регулярный и достаточно объёмный полив без вмешательства человека. Правильный полив обеспечивается расположенными по всему участку датчиков увлажнения почвы или влажности окружающей среды, на основе показания которых производится автоматический полив.

Существуют 3 основных вида системы полива:

- дождевой;
- капельный;
- внутрипочвенный.

Дождевой способ самый популярный для автоматизации полива из-за того, что вода попадает на почву через специальные фонтанчики, которые напоминают собой осадки, что нравится растениям.

Преимущества дождевого способа, изображенного на рис. 3:

- полив рассеянной водой не вредит структуре почвы;
- почва увлажняется глубоко;
- повышается влажность воздуха;

- вода не перемещается по поверхности благодаря чему не изменяется плодородный слой;
- большой радиус действия.

Из недостатков можно выделить невозможность равномерного полива при сильном ветре.



Рис. 3. Дождевой способ полива

Капельный способ доставляет воду непосредственно в зону высадки растения, благодаря чему сначала орошается корневая система, что позволяет экономить воду. В основном используется для полива теплиц и огородных растений, также иногда для деревьев или кустарников. Комплект для капельного полива изображен на рис. 4.

Преимущества капельного способа:

- минимальный расход воды, т. к. она сразу попадает на корни;
- равномерная подача воды;
- сохраняется полноценный доступ кислорода к корням;
- нет образований твердой корки на поверхности земли.

Из недостатков можно выделить частые засорения мелких отверстий в капельном шланге из-за чего его придется периодически чистить.



Рис. 4. Комплект для капельного полива

Внутрипочвенное орошение представляет собой комплекс поливальных систем для подземного орошения, состоящий из пластиковых труб, распределенных под землей для подачи воды непосредственно к корневой системе растений. Пример такой пластиковой трубы изображен на рис. 5 Обычно используется на таких участках, где нежелательно или невозможно часто перекапывать землю.

Преимущества внутрипочвенного орошения:

- экономия воды;
- минимальный коэффициент испарения;
- исключение образования корки;
- свободный доступ кислорода к корням.

Из недостатков можно выделить отсутствие орошения надпочвенных частей растений и запрет на использование в песчаных почвах.



Рис. 5. Поливальная система для внутрипочвенного полива

1.4.3 Освещение

Сложно представить в современном мире такую отрасль промышленности, где бы отсутствовала потребность в искусственном освещении. Не секрет, что работа освещения потребляет не мало ресурсов предприятия, но в то же время оно необходимо, т. к. с его помощью обеспечиваются оптимальные условия роста культур в теплице. Совершенно естественно, если владелец теплицы желает поддерживать идеальные условия для развития растений, но также желает сэкономить ресурсы. Тогда для него оптимальным вариантом будет внедрение автоматизации управления освещением. Эффективность автоматического управления позволит сократить расходы на электроэнергию и поддерживать идеальное состояние внутри теплицы 24/7.

Система автоматизации управления освещением является комплексом технологических решений, которая способна обеспечивать нужное количество света в нужное время и нужном месте. Эффективность автоматизации управления освещением можно расценивать наравне с переходом на энергоэффективные лампы.

Для частных теплиц внедрение отдельной системы автоматизации освещения может быть связано с большими финансовыми и трудовыми затратами, т. к. нужно закупить оборудование, установить его путем технических работ, настроить полученную систему и т. д. Такая нагрузка может служить поводом для отказа частных маленьких теплиц от автоматизации управления освещением. Однако, с другой стороны, внедрение системы управления освещением обеспечивает в автоматическом режиме требуемый уровень освещенности в теплице и ее зонах, а также повышает энергоэффективность и экономит ресурсы.

Автоматизированные системы освещения делятся на 2 вида:

- дискретное управление освещением;
Плюсы – наиболее экономичный и бюджетный вариант, освещенность регулируется путем полного или частичного отключения приборов. Недостатки – срок службы ламп при постоянном включении-выключении снижается.
- плавная настройка яркости;

Плюсы – бережет оборудование за счет плавных затуханий ламп, не используя резких отключений. Недостатки – дороже в эксплуатации.

У автоматизации освещения можно выделить несколько основных функций:

- точное поддержание искусственной освещенности в помещении на заданном уровне;
- учет естественной освещенности в помещении;
- учет времени суток;
- учет присутствия людей в помещении;
- дистанционное беспроводное управление осветительной установкой.

Искусственная освещенность достигается путем добавления в систему фотоэлемента, находящегося внутри помещения теплицы. Этот фотоэлемент контролирует создаваемую освещенность. Экономия энергии происходит путем отсечки «излишка освещенности».

Для реализации учета естественной освещенности в помещении используется тот же фотоэлемент, что описывался выше. В определенное время суток или года возможно отказаться от искусственного освещения и пользоваться только естественным, что позволит экономить электроэнергию на 20–40%.

Чтобы система могла отслеживать время суток и дни, ее необходимо снабдить собственными часами реального времени. Благодаря этой функции система может эффективно экономить электроэнергию, т. к. автоматическое отключение искусственного освещения происходит в определенное время суток.

Еще одной важной функцией, которую следует рассмотреть подробнее – удаленное беспроводное управление осветительной установкой. Это не является автоматизированной функцией, т. к. требует вмешательства человека, однако часто присутствует в автоматизированных системах управления освещением ввиду простоты своей реализации.

На самом деле перечень функций системы автоматизированного управления освещением ничем не ограничен. Здесь каждый решает в индивидуальном порядке, какие из функций ему нужнее в своей системе в зависимости от требований объекта.

Давайте проведем классификацию автоматизированных систем освещения. Существует два вида таких систем:

- локальные системы управления;
- централизованные системы управления;

Локальные системы управления работают с применением датчиков движения, присутствия и освещенности. Сами датчики в этой системе имеют необходимые средства для управления освещением с воем корпусе. Они так же могут управлять и другими устройствами, например кондиционерами, вентиляторами и т. п.

У локальных систем также есть свои недостатки, например:

- ограниченное количество подключаемых светильников;
- необходимости прокладки кабеля к группе устройств для реализации управления;

- отсутствие возможности расширения и масштабирования системы в случае необходимости;
- отсутствие функции управления освещением по времени;

Можно сделать вывод, что локальные системы управления освещением могут быть удобны при использовании на малых предприятиях, как наша теплица, однако затруднительны для дальнейшего расширения системы и не идеальны в управлении освещением полностью автоматически, следовательно не экономят электроэнергию.

Централизованные системы управления освещением являются системами более высокого уровня и позволяют реализовать полноценную автоматизацию управления освещением. Система строится на основе микропроцессоров, которые обеспечивают управление большим количеством светильников путем выдачи управляющих сигналов на светильники по сигналам локальных датчиков.

Из описания централизованной системы управления освещением можно понять, что этот тип системы отлично подходит для больших предприятий, намного лучше автоматизирован нежели локальная система управления освещением, что способствует сбережению электроэнергии и поддержанию идеальной среды для культур внутри теплицы 24/7.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА

2.1. Архитектура

Для реализации ВКР была выбрана архитектура клиент-сервер. Данная архитектура подразумевает под собой наличие базы данных, в которой хранятся какие-то данные, наличие клиента, который хочет получить какие-то данные из БД или добавить новые данные, и наличие сервера, который будет осуществлять обработку запросов с клиента и выборку данных из БД для передачи их клиенту в ответ на его запрос. Архитектура данной работы изображена на рис. 6.

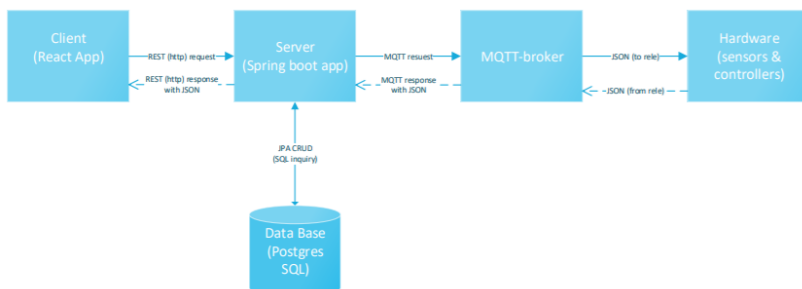


Рис. 6. Архитектура приложения

У данной архитектуры есть множество преимуществ:

1. На один сервер могут отправлять запросы множество клиентов;
2. Все вычисление происходит на сервере. Один мощный сервер стоит дешевле чем несколько клиентских машин, благодаря чему выгоднее будет хорошо оборудовать только одну машину, которая и будет сервером;

3. Так как весь код для работы с БД и алгоритмы работы самого сервера находятся только на одной машине, то клиентское ПО для работы с сервером и отрисовки данных не занимает много места на клиентской машине, что подразумевает под собой отсутствие дублирования кода;
4. Все данные находятся в безопасности в отдельной БД. Данные получает только одобренный сервером клиент по защищенному протоколу.

Минус клиент-серверной архитектуры происходит из его вышеперечисленных плюсов. Если сервер не доступен, или пропало соединение с БД, то все клиенты не могут получать данные из-за чего их работа до восстановления соединения с БД или сервером останавливается.

На рис. 7 изображена диаграмма последовательностей обмена данными между клиентом, сервером, брокером и БД.

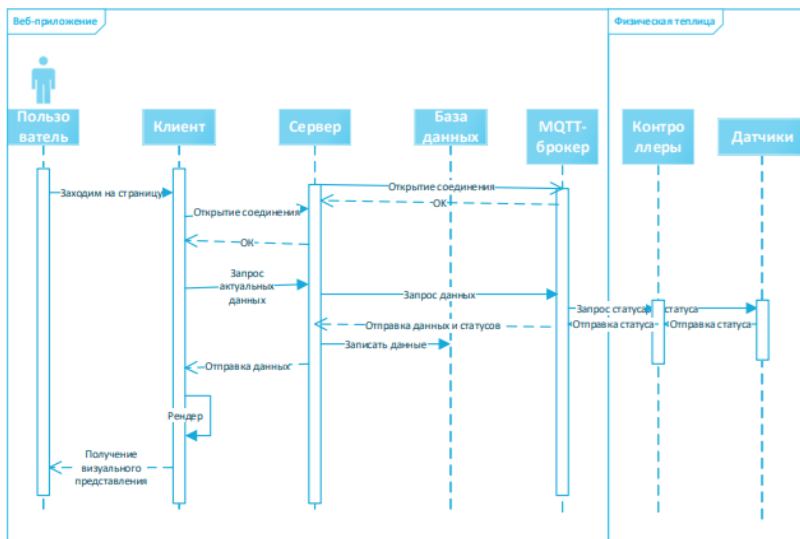


Рис. 7. Диаграмма последовательностей работы веб-приложения

2.2. Выбор средств реализации

Основная проблема выбора средств реализации программы для ВКР заключается в выборе фреймворка способного как общаться с различными базами данных, так и имеющего в себе возможность общения с брокером с помощью MQTT протокола.

В качестве основной платформы для реализации ПО был выбран фреймворк Spring языка Java. Он способен с помощью своих библиотек на общение с различными БД, что прекрасно подходит для работы с СУБД PostgreSQL, которая была выбрана для этой работы.

В качестве брокер-сервера, который должен общаться с компонентами умной теплицы, был выбран брокер Mosquito.

2.3. Проектирование базы данных

Для ВКР была разработана БД для хранения данных, полученных с датчиков и информации о работе элементов управления.

В таблице 2.1 хранятся различные названия датчиков, которые могут быть в теплице: датчики температуры, датчик влажности, датчик освещения и т. п.

Таблица 2.1

Таблица Sensor_types

Поле	Тип	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ
name	VARCHAR(255) UNIQUE NOT NULL CHECK(name != '')	Текстовая переменная для хранения названия типа. Название должно быть уникальным.

В таблице 2.2 хранятся: все датчики, их ip-адреса, идентификаторы типов, и сведения о том, включены ли какие-то датчики или нет.

Таблица 2.2

Таблица Sensors

Поле	Тип	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ

Продолжение таблицы 2.2

ip	VARCHAR(255) UNIQUE NOT NULL CHECK(name != '')	Текстовая переменная для хранения ip-адреса датчика. Ip-адрес должен быть уникальным. Поле не должно быть пустым.
Id_sensor_type	INTEGER REFERENCES sensor_types (id)	Идентификатор типа сенсора из таблицы sensor_types
Enabled	BOOLEAN	Переменная для проверки включен или выключен датчик

В таблице 2.3 хранятся все зоны и их названия.

Таблица 2.3

Таблица Areas

Поле	Тип	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ
name	VARCHAR(255) UNIQUE NOT NULL CHECK(name != '')	Текстовая переменная для названия зоны. Название должно быть уникальным. Поле не должно быть пустым.

В таблице 2.4 хранятся различные названия элементов управления, которые могут быть в теплице: элемент управления охлаждением, элемент управления полива и т.п.

Таблица 2.4

Таблица Control_types

Поле	Тип	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ
name	VARCHAR(255) UNIQUE NOT NULL CHECK(name != '')	Текстовая переменная для названия элемента управления. Название должно быть уникальным. Поле не должно быть пустым.

В таблице 2.5 хранятся: все элементы управления, их Ip-адреса, идентификаторы типов, состояния работы в данный момент и настройки для активации элементов управления. В зависимости от типа контроллера, поле “настройки” будет использоваться для передачи нужной команды брокеру.

Таблица 2.5

Таблица Controls

Поле	Тип	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ
ip	VARCHAR(255) UNIQUE NOT NULL CHECK(name != '')	Текстовая переменная для хранения ip-адреса элемента управления. Ip-адрес должен быть уникальным. Поле не должно быть пустым.
Id_control_type	INTEGER REFERENCES control_types (id)	Идентификатор типа элемента управления из таблицы control_types

Продолжение таблицы 2.5

Settings	INTEGER NOT NULL	Поле, которое содержит в себе настройку включения элемента управления. В зависимости от типа поле настройки используется для контроля на основном сервере
Enabled	BOOLEAN	Переменная для проверки включен или выключен датчик

В таблице 2.6 хранятся время и дата активации каждого контроллера. Когда брокеру передается сообщение на активацию, то в БД записывается время и идентификатор контроллера. После выполнения действия элемент управления выключается.

Таблица 2.6

Таблица Control_work

Поле	Тип	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ
Id_control	INTEGER REFERENCES controls (id)	Идентификатор элемента управления из таблицы controls
Worktime	TIMESTAMP NOT NULL	Поле для хранения время и даты, когда было произведено включение контроллера

В таблице 2.7 хранятся сведения о том, в какой зоне находится какой-либо элемент управления.

Таблица 2.7

Таблица Areas_controls

Поле	Тип	Описание
Id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ
Id_control	INTEGER REFERENCES controls (id)	Идентификатор элемента управления из таблицы controls
id_area	INTEGER REFERENCES areas (id)	Идентификатор зоны из таблицы areas

В таблице 2.8 хранятся все данные, полученные с датчиков вместе с датой и временем когда они были получены.

Таблица 2.8

Таблица Data

Поле	Тип	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор, первичный ключ
Id_sensor	INTEGER REFERENCES sensors (id)	Идентификатор типа элемента датчика из таблицы sensors
Id_area	INTEGER REFERENCES areas (id)	Идентификатор типа элемента зоны из таблицы areas
Date_time	TIMESTAMP NOT NULL	Поле для хранения время и даты, когда были получены данные
Data	TEXT NOT_NULL	Полученные данные

На рис. 8 изображена схема отношения базы данных. На рис. 9 изображена ER Диаграмма БД.

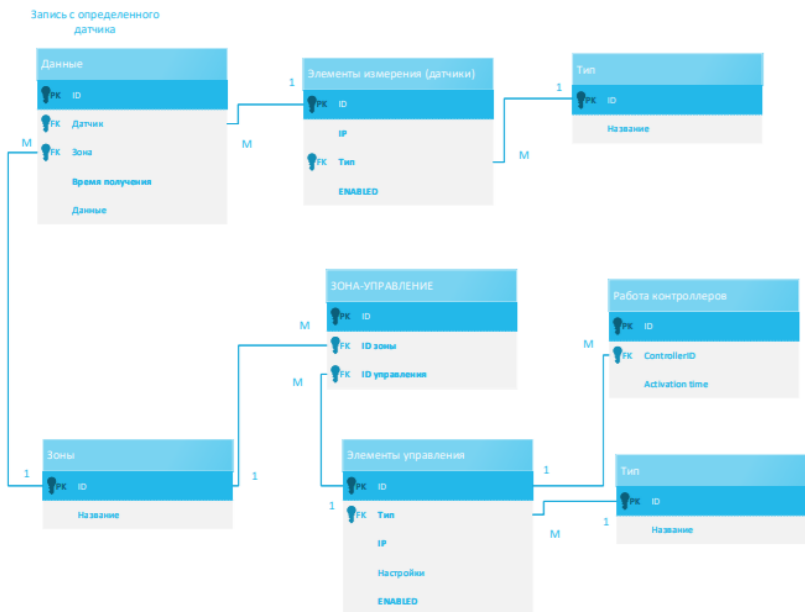


Рис. 8. Схема отношений БД

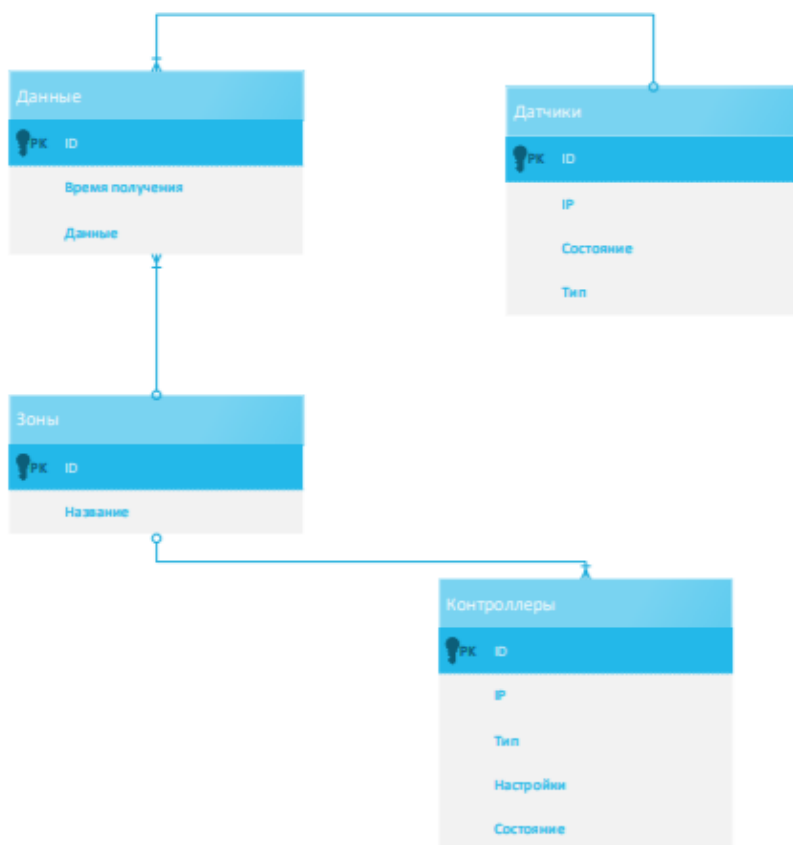
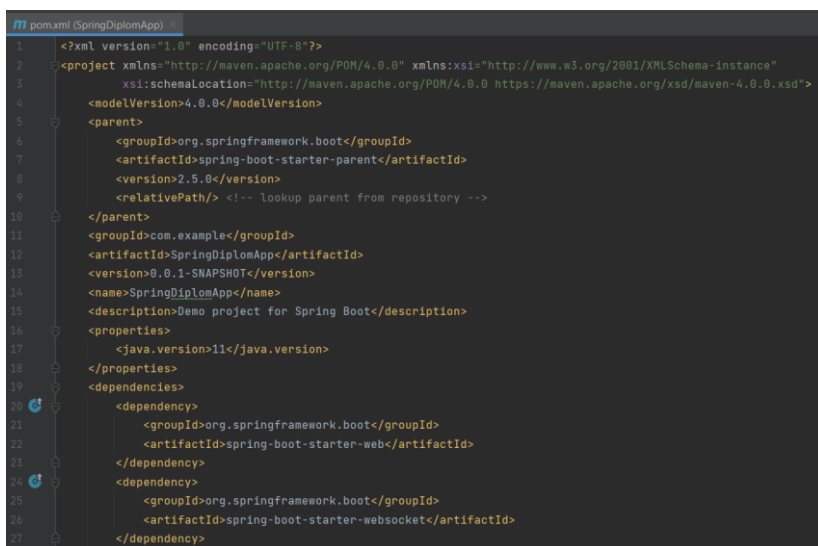


Рис. 9. ER диаграмма

2.4. Проектирование веб-сервера

Spring – это фреймворк который предоставляет комплексную модель программирования и конфигурации для современных приложений на основе языка Java.

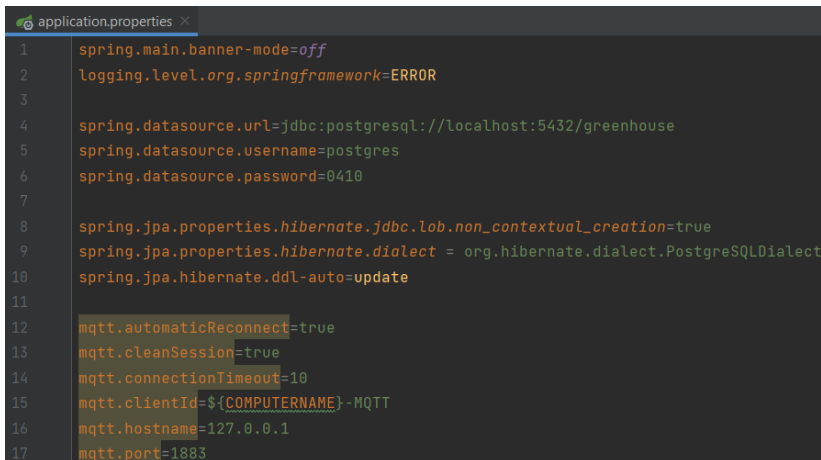
Для того, чтобы подключить к нашему Spring проекту все нужные библиотеки, был использован фреймворк Apache Maven который автоматизирует сборки проектов на основе описания их структуры в файлах. Описание структуры записано в файле POM формата xml и представлено на рис. 10. В этом файле прописываются зависимости всех нужных нам библиотек, таких как: org.postgresql, spring-integration-mqtt и т.д.

The image shows a screenshot of a code editor with a dark theme, displaying a Maven POM (Project Object Model) file named 'pom.xml (SpringDiplomApp)'. The XML content is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.5.0</version>
9         <relativePath><!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.example</groupId>
12    <artifactId>SpringDiplomApp</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>SpringDiplomApp</name>
15    <description>Demo project for Spring Boot</description>
16    <properties>
17        <java.version>11</java.version>
18    </properties>
19    <dependencies>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter-web</artifactId>
23        </dependency>
24        <dependency>
25            <groupId>org.springframework.boot</groupId>
26            <artifactId>spring-boot-starter-websocket</artifactId>
27        </dependency>
```

Рис. 10. Файл конфигурации

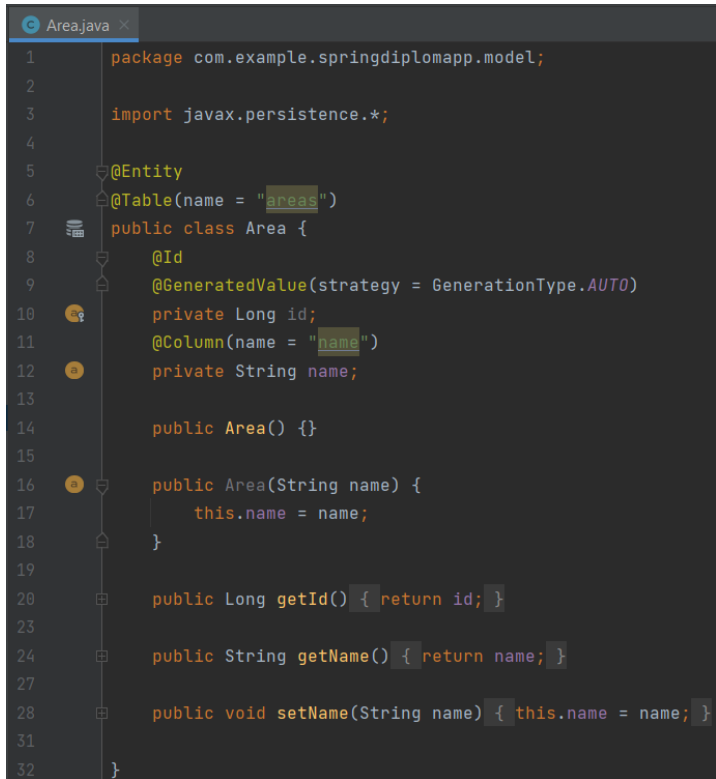
Для того, чтобы сервер знал откуда брать данные нужно ему указать в специальном файле, изображённом на рис. 11, данные для подключения к БД.

A screenshot of a code editor showing the 'application.properties' file. The file contains 17 lines of configuration for a Spring application, including database connection details for PostgreSQL, Hibernate settings, and MQTT client configuration. The text is color-coded: green for standard properties, blue for class names, and orange for placeholders like {COMPUTERNAME}.

```
1 spring.main.banner-mode=off
2 logging.level.org.springframework=ERROR
3
4 spring.datasource.url=jdbc:postgresql://localhost:5432/greenhouse
5 spring.datasource.username=postgres
6 spring.datasource.password=0410
7
8 spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
9 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
10 spring.jpa.hibernate.ddl-auto=update
11
12 mqtt.automaticReconnect=true
13 mqtt.cleanSession=true
14 mqtt.connectionTimeout=10
15 mqtt.clientId=${COMPUTERNAME}-MQTT
16 mqtt.hostname=127.0.0.1
17 mqtt.port=1883
```

Рис. 11. Настройки приложения

Для приложения были реализованы классы, которые представляют собой сущности таблиц из нашей БД. К примеру, на рис. 12 изображен класс, который представляет собой сущность таблицы `ageas` из основной БД. На 6 строчке расположена аннотация, которая указывает, что данный класс является сущностью. На 7 строчке расположена аннотация, принимающая в себя название таблицы, которой наш класс и будет являться.



```

1  package com.example.springdiplomapp.model;
2
3  import javax.persistence.*;
4
5  @Entity
6  @Table(name = "areas")
7  public class Area {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.AUTO)
11     private Long id;
12     @Column(name = "name")
13     private String name;
14
15     public Area() {}
16
17     public Area(String name) {
18         this.name = name;
19     }
20
21     public Long getId() { return id; }
22
23     public String getName() { return name; }
24
25     public void setName(String name) { this.name = name; }
26
27 }

```

Рис. 12. Класс сущности areas

Spring поддерживает библиотеку jpa, в которой доступен интерфейс Repository. Для работы с данными для каждой сущности был создан интерфейс, который наследуется от JpaRepository и получает в наследство методы, реализующие CRUD концепцию. Пример интерфейса репозитория таблицы areas представлен на рис. 13.

```

1 package com.example.springdiplomapp.repository;
2
3 import com.example.springdiplomapp.model.Area;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface AreaRepository extends JpaRepository<Area, Long> {
9 }

```

Рис. 13. Репозиторий для сущности areas

Для выборки данных по не нескольким таблицам были созданы классы, которые тоже реализуют собой сущность, но не имеют привязки к какой-либо таблице.

Для данных классов были реализованы специальные репозитории, которые имеют в себе метода с аннотацией Query которой передан нужный для БД запрос на выборку данных. К примеру, на рис. 14 изображен репозиторий для DataDTO сущности, которая является выборкой данных из 3 таблиц.

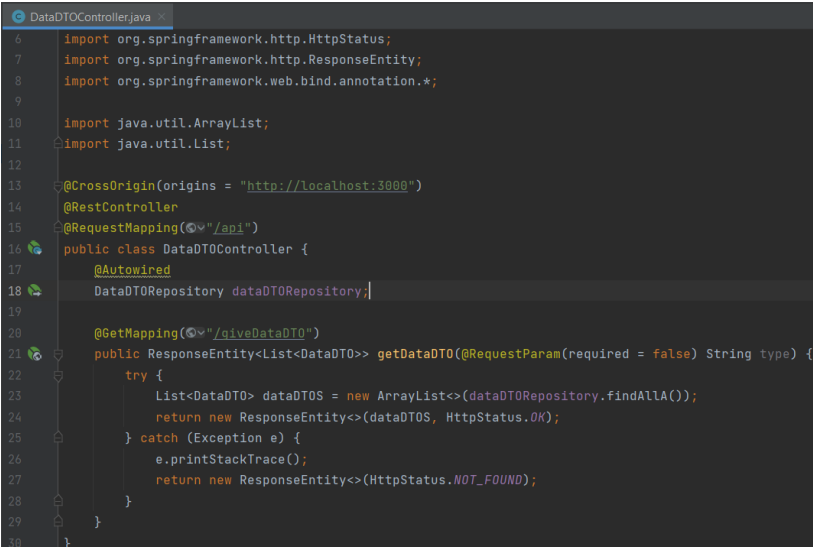
```

1 package com.example.springdiplomapp.repository;
2
3 import com.example.springdiplomapp.dto.DataDTO;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.stereotype.Repository;
7
8 import java.util.List;
9 @Repository
10 public interface DataDTORepository extends JpaRepository<DataDTO, String> {
11     @Query(value = "SELECT d.id, s.ip, st.name, a.name as area_name, d.data_time, d.data FROM data d " +
12         "JOIN sensors s ON d.id_sensor = s.id " +
13         "JOIN sensor_types st ON s.id_sensor_type = st.id " +
14         "JOIN areas a ON d.id_area = a.id", nativeQuery = true)
15     List<DataDTO> findAll();
16 }

```

Рис. 14. Репозиторий DataDTO

Для того, чтобы сервер мог принимать и получать данные от клиента, нужны специальные классы контроллеры, которые с помощью реализованных ранее репозиториях будут принимать, сохранять и редактировать данные, полученные от клиента, а также отправлять ему данные в ответ на его запрос. На рис. 15 изображен класс `DataDTOController`, для получения данных из которого, нужно отправить запрос по ссылке `/api/giveDataDTO`. Этот класс имеет метод, в котором с помощью репозитория вызывается выборка данных из нужных нам таблиц и отправляет эту выборку клиенту вместе с логом.



```
1  DataDTOController.java
2
3  6      import org.springframework.http.HttpStatus;
4      7      import org.springframework.http.ResponseEntity;
5      8      import org.springframework.web.bind.annotation.*;
6      9
7      10     import java.util.ArrayList;
8      11     import java.util.List;
9      12
10     13     @CrossOrigin(origins = "http://localhost:3000")
11     14     @RestController
12     15     @RequestMapping("/api")
13     16     public class DataDTOController {
14     17         @Autowired
15     18         DataDTORepository dataDTORepository;
16     19
17     20         @GetMapping("/giveDataDTO")
18     21         public ResponseEntity<List<DataDTO>> getDataDTO(@RequestParam(required = false) String type) {
19     22             try {
20     23                 List<DataDTO> dataDTOS = new ArrayList<>(dataDTORepository.findAll());
21     24                 return new ResponseEntity<>(dataDTOS, HttpStatus.OK);
22     25             } catch (Exception e) {
23     26                 e.printStackTrace();
24     27                 return new ResponseEntity<>(HttpStatus.NOT_FOUND);
25     28             }
26     29         }
27     30     }
```

Рис. 15. Контроллер `DataDTOController`

2.5. Общение с брокером через MQTT протокол

Общение сервера и аппаратной части происходит при помощи протокола MQTT. MQTT – это протокол обмена сооб-

щениями, который работает по шаблону издатель – подписчик. Издателями, в данном случае, выступают датчики и элементы управления теплицы, которые отправляют сообщения. Подписчиками являются конечные получатели данных.

MQTT поддерживает три уровня качества обслуживания при передаче сообщений:

1. QoS 0 – сообщение передается брокеру не более одного раза, подтверждение от брокера не ожидается.
2. QoS 1 – сообщение передается как минимум один раз, получатель должен подтвердить доставку, но сообщения могут дублироваться.
3. QoS 2 – сообщение передается только один раз без дублирования, сервер обеспечивает доставку с подтверждением.

Для того, чтобы наладить общение между издателями и подписчиками нужно использовать брокер. Брокер – это центральный узел в MQTT, который будет отвечать за общение благодаря тому, что он получает данные от подписчиков, обрабатывает их и убеждается в том, что сообщение было доставлено подписчику.

В качестве брокера был выбран брокер с открытым исходным кодом Mosquitto, С помощью него можно не только обмениваться сообщениями, но и также отслеживать их, что продемонстрировано на рис. 16.

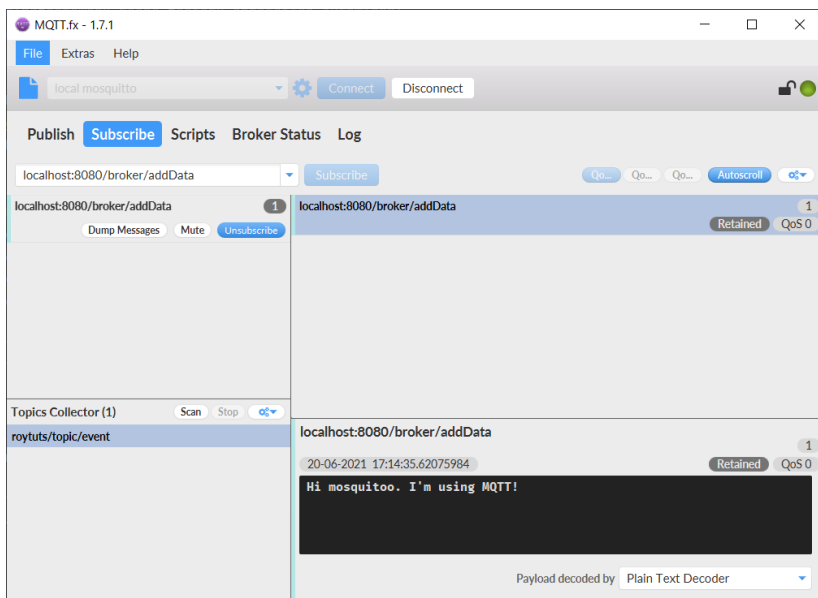


Рис. 16. Mosquito

Для работы с MQTT протоколом был реализован класс сервис, который содержит в себе методы для подписки на издателя и отправки запросов, данный класс представлен на рис. 17.

```

MQTTBrokerController.java x MessagingService.java x
import org.springframework.stereotype.Service;

9
10 @Service
11 public class MessagingService {
12     @Autowired
13     private IMqttClient mqttClient;
14
15     @
16     public void publish(final String topic, final String payload, int qos, boolean retained)
17         throws MqttPersistenceException, MqttException {
18         MqttMessage mqttMessage = new MqttMessage();
19         mqttMessage.setPayload(payload.getBytes());
20         mqttMessage.setQos(qos);
21         mqttMessage.setRetained(retained);
22
23         mqttClient.publish(topic, mqttMessage);
24         mqttClient.disconnect();
25     }
26
27     public void subscribe(final String topic) throws MqttException, InterruptedException {
28         System.out.println("Message received:");
29
30         mqttClient.subscribeWithResponse(topic, (topic, msg) -> {
31             System.out.println(msg.getId() + " -> " + new String(msg.getPayload()));
32         });
33     }
34 }

```

Рис. 17. Сервис для MQTT клиента

Для того, чтобы клиент мог отправить запрос брокеру был реализован класс контроллер, представленный на рис. 18. Клиенту нужно отправить запрос по пути “/api/sendMessage” для того, чтобы сервер передал запрос брокеру.


```

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/api")
public class MQTTPublisherController {
    @Autowired
    private MessagingService messagingService;

    @PostMapping("/sendMessage")
    public ResponseEntity<String> sendMqttMessage(@RequestBody String text) {
        final String topic = "greenhouse/topic/message";
        try {
            messagingService.subscribe(topic);
            messagingService.publish(topic, text, qos: 0, retained: true);
        } catch (MqttException | InterruptedException e) {
            e.printStackTrace();
        }

        return new ResponseEntity<>(null, HttpStatus.OK);
    }
}

```

Рис. 18. Контроллер для отправки сообщений брокеру

Данные, получаемые с датчиков, передаются по пути “/broker/addData”, после чего специальный класс изображенный на рис. 19 добавляет данные в базу данных.

```

@CrossOrigin(origins = "http://localhost:1883")
@RestController
@RequestMapping("/broker")
public class DataController {
    DataRepository dataRepository;

    @PostMapping("/addData")
    public void addNewData(@RequestBody MqttMessage mqttMessage) {
        Data data = new Data();
        data.setId_sensor((long) mqttMessage.getId());
        data.setData(Arrays.toString(mqttMessage.getPayload()));
        data.setDate_time(new Timestamp(System.currentTimeMillis()));
        data.setId_area(0L);
        dataRepository.save(data);
    }
}

```

Рис. 19 Контроллер для добавления данных в БД

ЗАКЛЮЧЕНИЕ

В ходе работы было разработано серверное приложение для обработки запросов клиента и выборки данных из базы данных по этим запросам для передачи нужной информации клиенту. Также сервер выполняет функцию подписчика на брокер через протокол MQTT для получения информации от аппаратной части проекта. Приложение обеспечило быструю работу с клиентской частью проекта и возможность добавления данных в БД и выборку данных из неё. Также был настроен брокер, который выполняет роль посредника между серверной и аппаратной частью. Программный продукт является завершённым и полностью удовлетворяет необходимым требованиям для серверного приложения, несмотря на это, он может быть улучшен с помощью перехода общения клиента с сервером на веб-сокеты или возможностью подключения нескольких клиентов и добавлением возможности для авторизации каждого клиента. Задача, поставленная в ходе выполнения работы была полностью выполнена и выбор данных средств для разработки полностью оправдал ожидания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация фреймворка Spring [электронный ресурс] сайт.
Режим доступа: свободный.
URL: <https://docs.spring.io/spring-integration/docs/current/reference/html/>
2. Примеры работы фреймворка Spring [электронный ресурс] сайт.
Режим доступа: свободный.
URL: <https://www.baeldung.com/rest-with-spring-series>
3. Документация к брокеру Mosquitto [электронный ресурс] сайт.
Режим доступа: свободный.
URL: <https://mosquitto.org/documentation/>
4. Документация к psql [электронный ресурс] сайт.
Режим доступа: свободный.
URL: <https://postgrespro.ru/docs/postgresql/9.6/app-psql>
5. Документация к MQTT [электронный ресурс] сайт.
Режим доступа: свободный.
URL: <http://www.steves-internet-guide.com/mqtt/>
6. Научная статья посвященная автоматизации теплиц [электронный ресурс] статья.
Режим доступа: свободный.
URL: <https://ap-n.com/avtomatizacija-teplic/>
7. Научная статья посвященная интернету вещей [электронный ресурс] статья.
Режим доступа: свободный.
URL: <https://trends.rbc.ru/trends/industry/-5db96f769a7947561444f118>