Введение

Актуальность темы –

Современный мир стал свидетелем интенсивного развития технологий и автоматизации процессов во всех сферах деятельности. Актуальность темы создания удаленной системы управления переносной мини-теплицей обусловлена растущим интересом к современным технологиям в области сельского хозяйства и увеличением потребности в свежих овощах, ягодах и зелени круглый год. Данная тема может быть полезна как для обычного человека, желающего выращивать свежие продукты в домашних условиях, так и для ресторанов и сельхоз предприятий.

Разработка удаленной системы управления переносной мини-теплицей позволит обычным людям выращивать овощи, ягоды и декоративные растения в любое время года в домашних условиях, что является важным шагом в направлении экологии, автономности и самообеспечения хозяйств, экономии средств на покупку продуктов. Для ресторанов и других предприятий общественного питания, выращивание микрозелени и ягод в мини-теплицах с помощью удаленной системы управления может стать выгодным и экономически эффективным решением, позволяющим сократить затраты на закупку свежих ингредиентов и обеспечить высокое качество блюд.

Одним из основных преимуществ создания удаленной системы управления переносной мини-теплицей является возможность контролировать все процессы выращивания растений удаленно с помощью мобильного устройства или компьютера. Это позволит пользователям получать информацию о температуре, влажности почвы и воздуха, уровне освещенности и других показателях, а также управлять системой полива, освещения и вентиляции из любой точки планеты. В результате, это существенно улучшит процесс выращивания овощей и растений в условиях городской среды и сельского хозяйства, способствуя повышению качества продукции и улучшению экологической ситуации в стране.

Создание удаленной системы управления теплицами также может быть полезна для сельскохозяйственной промышленности в целом, обеспечивая более эффективное и точное управление производственными процессами, а также улучшение качества и урожайности продукции.

Цель работы –

Целью данной работы является создание удаленной системы управления переносной мини-теплицей, которая позволит обеспечить круглогодичное выращивание овощей, ягод и декоративных растений в домашних условиях. Данная система будет представлять собой комплекс из аппаратного и программного обеспечения, которое позволит управлять теплицей из любой точки мира с помощью интернета. Основная задача заключается в создании удобного и эффективного способа выращивания растений в условиях ограниченного пространства и времени.

В рамках работы будут рассмотрены существующие подходы к управлению теплицами и разработана новая система, которая будет основана на использовании современных технологий интернета вещей и удаленного управления. Результатом работы будет готовый прототип системы, который можно будет использовать для реализации проектов в области сельского хозяйства и создания удобных условий для выращивания растений в домашних условиях.

Основные задачи –

1. Создание и разработка базы данных для хранения информации о состоянии теплицы, ее настройках, пользователях системы и производимых измерениях.
2. Создание и разработка архитектуры веб-сервисов для реализации всего необходимого функционала системы, способного выдерживать необходимую нагрузку и обеспечивать заданную задержку.
3. Создание и разработка прототипа теплицы и программирование микроконтроллера, который будет обеспечивать связь между теплицей и веб-сервисами, а также обрабатывать данные с датчиков и поддерживать автономный режим работы.
4. Создание удобного и интуитивно понятного пользовательского интерфейса для использования системы.

Архитектура системы и ее сервисы

1. Разработка архитектуры

В данном разделе будут подробно описаны основные архитектурные решения, принятые при разработке системы, а также факторы, повлиявшие на их выбор.

* 1. «Тонкий клиент»

Учтенные факторы:

1. Ограничения микроконтроллера – микроконтроллеры имеют ограниченный объем памяти и вычислительных мощностей, что сильно ограничивает возможность реализации логики на стороне пользователя.
2. Обновляемость - изменения в функционале системы могут потребовать изменения кода микроконтроллера, для чего каждому пользователю необходимо будет лично подключить теплицу к ПК и перепрошить микроконтроллер.

В связи с данными факторами для системы было принято решение выбрать архитектуру «Тонкий клиент», роль клиента в которой будет исполнять микроконтроллер, а роль сервера – веб-сервисы на виртуальных машинах.

Таким образом, большая часть обработки и хранения данных будет происходить на серверной стороне, функционал микроконтроллера будет ограничен только поддержкой системных функций, отправлением данных с датчиков и исполнением максимально простых команд.

* 1. RESTful API.

Учтенные факторы:

1. Расширяемость – система должна обеспечивать возможность быстрого внесения новой функциональности и изменения уже существующей без необходимости серьезных изменений в архитектуре. Это связанно с широким спектром целевой аудитории – от сельскохозяйственной промышленности до физических лиц.
2. Вариативность ui-клиентов – дальнейшее развитие системы может потребовать создания множества различных ui клиентов для пользователей – сайт, мобильное приложение, desktop приложение, боты для социальных сетей и т.д., в связи с чем серверное приложение должно предоставлять единообразное API для всех клиентов.
3. Сложность тестирования – в процессе работы приложения, входящие данные будут проходить через большое количество операций – приведение к системе СИ, анализ для работы автоматических систем управление, запись в БД и др. Данный фактор серьезно усложняет тестирование системы и создает потребность в возможности тестирования работы на различных этапах обработки данных.

В связи с данными факторами для серверного приложения было принято решение выбрать паттерн архитектуры RESTful API c трехслойной архитектурой.

Таким образом разделение приложения на слои позволит реализовать принципы SOLID, что обеспечит легкую расширяемость функционала и возможность тестирования корректности обработки данных на различных слоях, а слой представления данных организованный в стиле REST – единообразный унифицированный интерфейс для различных ui-клиентов.

* 1. «Диспетчер»

Учтенные факторы:

1. Большой объем данных – для возможности реализации автоматического управления теплицей и фиксации статистических выбросов во входящих данных каждая теплица на постоянной основе и с достаточной частотой должна отправлять результаты измерений параметров среды. Однако, полная обработка и запись данных в БД –трудоемкий процесс, более того, теплица не является объектом критически важной инфраструктуры и для пользователя нет необходимости хранить результаты с той же точностью и частотой, которая нужна для данного функционала.
2. Масштабирование – с увеличением числа пользователей система должна легко масштабироваться для обеспечения заданного уровня задержки управления. Однако создание нескольких экземпляров API может спровоцировать «гонку за ресурсы» за ресурсы БД

В связи с данных факторами было принято решение создание отдельного сервиса между клиентом-микроконтроллером и API – диспетчера. Данный сервис будет получать от API текущую конфигурацию для подключаемого микроконтроллера, генерировать для него команды на основе входящих данных и отправлять API усредненные результаты измерений за определенный период.

* 1. Сайт и «конфигуратор»

В качестве основного ui-клиента для данной системы было принято решение разработать отдельный сайт. Он наиболее универсален, так как изначально доступен для большинства устройств и предоставляет больше возможностей для реализации функционала чем приложения и боты для социальных сетей.

Также существует потребность в создании «конфигуратора» - desktop приложения для первоначальной настройки теплицы. Приложение должно взаимодействовать с подключенной к ПК по USB порту теплицей, давая возможность пользователю настроить используемую ею Wi-Fi сеть и зарегистрировать ее в системе под своими данными.

2. Общая архитектура

В результате проектирования была принята следующая версия архитектуры:



Серверные приложения включают в себя: БД, API, сайт и один или несколько экземпляров диспетчера. Экземпляры сервисов распространяются в виде docker образов и могут располагаться как на нескольких виртуальных машинах, так и на одной, используя разные порты.

Развертывание и тестирование экземпляров приложений автоматизировано по методологии CI/CD.

В процессе разработки диплома для экономии средств все серверные приложения будут располагаться на одной ВМ. В случае дальнейшего развития проекта и его коммерческого использования целесообразно расположить все приложения на разных ВМ для повышения отказоустойчивости и масштабируемости системы.

Приложение «конфигуратор» распространяется через сайт и устанавливается на ПК пользователя вместе с необходимыми для сообщения с МК драйверами.

Общая схема взаимодействия приложений:



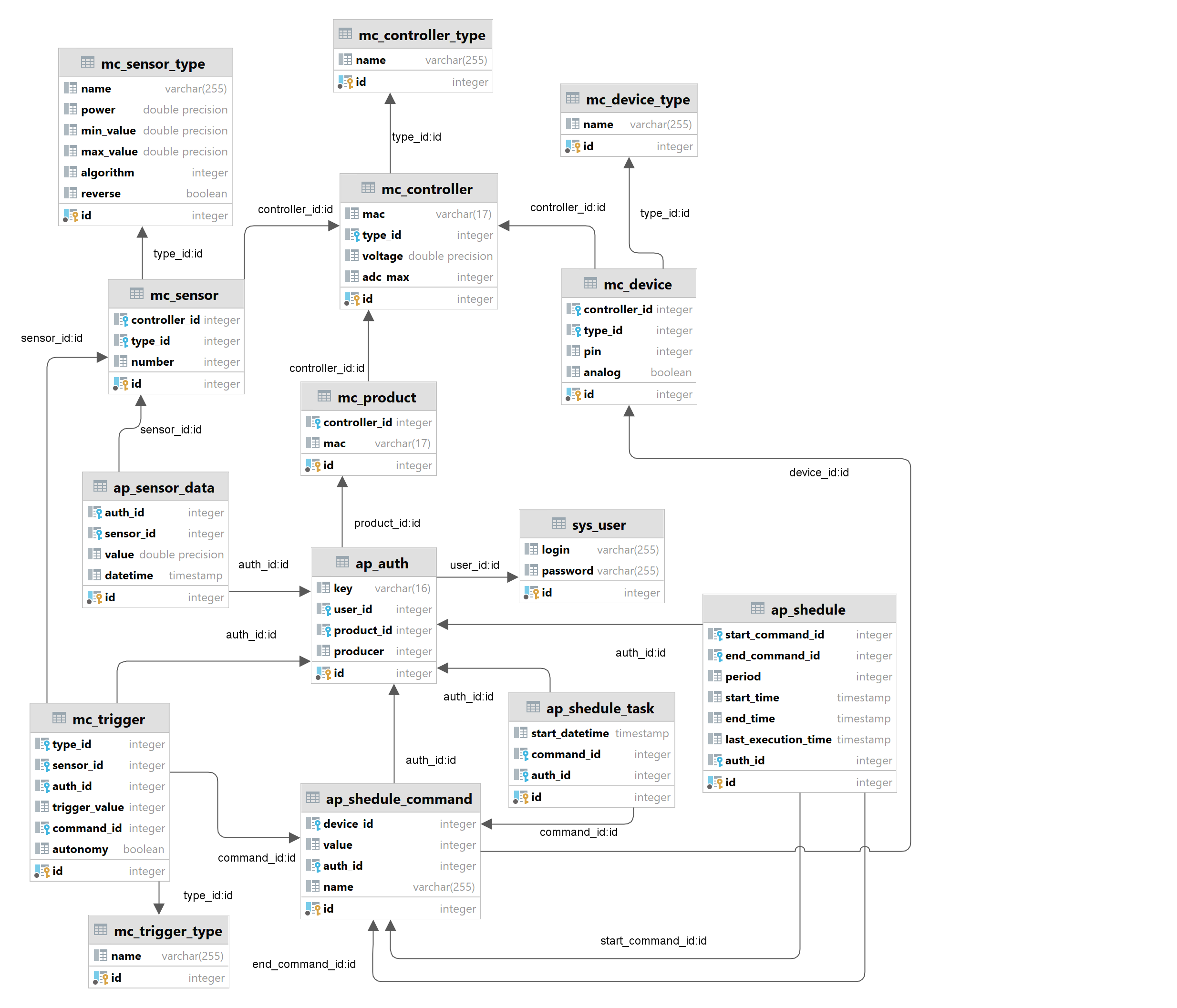
2.1. База данных

Для разработки системы выдвинуты следующие требования:

1. Высокая производительность чтения и записи данных – диспетчер периодически производит как запись усредненных результатов измерений, так и чтение текущей конфигурации контроллера, в связи с чем требуется БД с оптимальной скоростью записи/чтения данных.
2. Поддержка масштабирования и гибкости в добавлении новых таблиц и полей – см 1.2 «Расширяемость»
3. Открытый исходный код – на сегодняшний день в Российской федерации происходит массовый отказ от иностранного лицензионного ПО и переход на использование либо отечественных систем, либо систем с открытым исходным кодом.

Таким образом, с учетом всех вышеперечисленных требований, в качестве основной для реализации системы была выбрана реляционная БД PostgreSQL, однако, планируется сохранить поддержку подключения других похожих БД, таких как MySQL и MS SQL.

Схема БД:



|  |  |  |
| --- | --- | --- |
| MC\_CONTROLLER\_TYPE – модель МК | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Name | VARCHAR | Наименование типа контроллера |

|  |  |  |
| --- | --- | --- |
| MC\_SENSOR\_TYPE – модель датчика | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Name | VARCHAR | Наименование типа датчика |
| Power | DOUBLE | Рабочее напряжение датчика |
| Min\_value | DOUBLE | Минимальное значение датчика в СИ |
| Max\_value | DOUBLE | Максимальное значение датчика в СИ |
| Algorithm | INT | Дискриминатор алгоритма для приведения в СИ |
| Reverse | BOOL | Обратная зависимость возрастания напряжения к результату измерения |

|  |  |  |
| --- | --- | --- |
| MC\_DEVICE\_TYPE – модель устройства | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи, генерируемый системой, начиная с 1 |
| Name | VARCHAR | Наименование типа девайса |

|  |  |  |
| --- | --- | --- |
| MC\_CONTROLLER – версия прошивки МК | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Type\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_CONTROLLER\_TYPE].[ID] |
| Voltage | DOUBLE | Опорное напряжение микроконтроллера |
| Adc\_max | INT | Разрядность АЦП |

|  |  |  |
| --- | --- | --- |
| MC\_SENSOR – датчик | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Controller\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_CONTROLLER].[ID] |
| Type\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_SENSOR\_TYPE].[ID] |
| Number | INT | Порядковый номер датчика |

|  |  |  |
| --- | --- | --- |
| MC\_DEVICE - устройство | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Controller\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_CONTROLLER].[ID] |
| Type\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_SENSOR\_TYPE].[ID] |
| Pin | INT | Номер пина, к которому подключено устройство |
| Analog | BOOL | ШИМ- подключение |

|  |  |  |
| --- | --- | --- |
| MC\_RPODUCT – конкретный прошитый МК | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Controller\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_CONTROLLER].[ID] |
| Mac | VARCHAR(17) | MAC-адрес микроконтроллера |

|  |  |  |
| --- | --- | --- |
| AP\_AUTH – зарегистрированный МК | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Key | VARCHAR(16) | Ключ регистрации |
| User\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [SYS\_USER].[ID] |
| Product\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_RPODUCT].[ID] |
| Producer | INT | Дискриминатор режима работы теплицы |

|  |  |  |
| --- | --- | --- |
| AP\_SENSOR\_DATA – данные с датчиков | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Sensor\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_SENSOR].[ID] |
| Auth\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_AUTH].[ID] |
| Value | DOUBLE | Значение датчика |
| Datetime | DATETIME | Дата и время измерения |

|  |  |  |
| --- | --- | --- |
| SYS\_USER – пользователь системы | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Login | VARCHAR | Логин пользователя |
| Password | VARCHAR | Пароль пользователя |

|  |  |  |
| --- | --- | --- |
| AP\_COMMAND – базовые команды МК | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Name | VARCHAR | Наименование команды |
| Device\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_DEVICE].[ID] |
| Auth\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_AUTH].[ID] |
| Value | INT | Значение устройства |

|  |  |  |
| --- | --- | --- |
| AP\_TRIGGER – триггер для режима работы «по правилам» | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Type\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_TRIGGER\_TYPE].[ID] |
| Sensor\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_SENSOR].[ID] |
| Auth\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_AUTH].[ID] |
| Trigger\_value | INT | Значение срабатывания триггера |
| Command\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [AP\_COMMAND].[ID] |
| Autonomy | BOOL | Включается ли триггер в автономный конфиг |

|  |  |  |
| --- | --- | --- |
| MC\_TRIGGER\_TYPE – тип триггера | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Name | VARCHAR | Наименование типа триггера |

|  |  |  |
| --- | --- | --- |
| AP\_SHEDULE – расписание команд для режима работы «ручной» | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Start\_command\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [AP\_COMMAND].[ID] |
| End\_command\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [AP\_COMMAND].[ID] |
| Start\_time | DATETIME | Время запуска стартовой команды |
| End\_time | DATETIME | Время запуска финальной команды |
| Last\_execution\_time | DATETIME | Дата и время последнего исполнения |
| Auth\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_AUTH].[ID] |
| Period | INT | Кол-во дней между запусками |

|  |  |  |
| --- | --- | --- |
| AP\_SHEDULE\_TASK – запланированные на текущий день команды | | |
| ID | INT | (PK) Первичный ключ — автоинкрементный номер записи |
| Start\_datetime | DATETIME | Время запуска команды |
| Command\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [AP\_COMMAND].[ID] |
| Auth\_id | INT | (FK) Внешний ключ – ссылка на первичный ключ [MC\_AUTH].[ID] |

2.2 API

Как уже обсуждалось ранее (см 1.2) API должен представлять из себя трехслойное веб-приложение, в котором слой представления соответствует парадигме RESTful API.

Трехслойная архитектура – архитектурный подход к разработке ПО, в котором приложение разделяется на три слоя: представления (presentation layer), бизнес-логики (business logic layer) и данных (data storage layer).

Слой представления – отвечает за отображение данных пользователю. В нашем приложении должен представлять из себя RESTful API. Для разработки было принято решение использовать кроссплатформенный фреймворк ASP.NET платформы .NET 7.

Слой бизнес-логики – содержит логику приложения, включая все правила и операции, которые управляют поведением приложения.

Слой уровня данных – отвечает за сохранение и извлечение данных. Для поддержания расширяемости, сопровождения и безопасности кода подразумевается использование технологии ORM.

REST – это архитектурный стиль веб-сервисов, использующий протокол HTTP для передачи данных и обеспечивающий масштабируемость и гибкость системы. Для соблюдения требований безопасности ISO/EIC 27001 будет использоваться протокол HTTPS.

**Безопасность.**

Приложение API наиболее критично к вопросу безопасности, так как представляет собой ключевой узел системы и предоставляет доступ к БД. Для соблюдения стандартов ISO/EIC 27001 и OWASP были приняты следующие решения:

1. Использование протокола HTTPS.
2. Использование JWT-аутентификации. Вся аутентификация в системе происходит исключительно через API.
3. Обязательный атрибут httpOnly для всех Cookie.
4. Использование технологии ORM для слоя данных.

**Разработка.**

Слои бизнес-логики и данных было решено реализовывать в виде двух dll библиотек: Domain.dll и Repository.dll.

Domail.dll – библиотека, в которой объявлены все основные и общедоступные интерфейсы моделей без реализации. Данное решение продиктовано наличием еще как минимум одного сервиса, который будет использовать те же модели (диспетчер), а также перспективой создания desktop и мобильных клиентов.

Repository.dll – библиотека, содержащая имплементацию для всех интерфейсов из Domain.dll, а так же собственные классы логики для обеспечения работы ORM Nhibernate и планировщика задач Quartz.NET.

В ходе разработки были реализованы следующие абстрактные интерфейсы:

Domain.dll:

IEntity – обобщенный класс для всех моделей системы.

IEntityRepository<T> where T: IEntity – обобщенный класс репозитория, содержащий функции для работы с NHibernate и основными CRUD операциями БД над моделями.

IEntityManager<T> where T: IEntity – обобщенный класс менеджера, содержащий основную бизнес логику и CRUD операциями репозитория.

ApiDto<T> where T: IEntity – обобщенный класс используемый для json-сериализации моделей.

Repository.dll:

IEntityMapping<T> where T: IEntity – обобщенный класс маппингов моделей системы.

NHibernateHelper – класс предоставляющий доступ к абстрактной фабрике сессий NHibernate и др. функциям для работы с БД вне контекста моделей.

Api:

BaseCrudController<dto,entity> where dto: ApiDto<entity> where entity: IEntity – обобщенный класс контроллера, предоставляющий пользователю основные REST операции над моделями системы.

Так, диаграмма последовательности операций для классической операции создания нового объекта выглядит следующим образом:



Благодаря данной системе, добавление классического функционала для всех новых моделей требует только создания нескольких унаследованных классов. Для уникального функционала, например, каскадирования операций, достаточно просто переопределить необходимые функции на нужном уровне.

2.3 Диспетчер

Диспетчер – веб-сервис, представляющий собой промежуточное звено между API и МК. Для каждого подключенного МК в оперативной памяти должна храниться сессия, содержащая основную конфигурацию текущего МК полученную от API, а также сгенерированные команды и результаты измерений за определенный период. Диспетчер должен на постоянной основе прослушивать все сообщения от подключенных к нему МК, обрабатывать и сохранять приходящие от них результаты измерений и на основании этих данных и текущей конфигурации – генерировать ответы со списком команд, которые должен выполнить МК.

**Протокол общения с МК**

Диспетчер использует HTTPS протокол для общения с API, однако, данный протокол не подходит для общения с МК по следующим причинам:

1. SSL-шифрование слишком трудоемкий процесс для использования на ограниченных в ресурсах микроконтроллерах. Согласно проведенным тестам, шифрование/дешифрование сообщений HTTPS протокола МК ESP8266 тратит в среднем 3-4 с, а ESP32 1-2 с., что создает слишком большую задержку.
2. Большой объем ненужной информации, которую по умолчанию содержат сообщения HTTPS протокола, к примеру, заголовки запроса, которые не используются при взаимодействии МК и диспетчера.

В связи с этим, было принято решение использовать другой протокол для общения МК и диспетчера. Выбор стоял между тремя протоколами, специализированными для IoT-устройств: MQTT, AMQP и COAP.

Таблица-сравнение протоколов:

|  |  |  |  |
| --- | --- | --- | --- |
| Характеристика | MQTT | AMQP | COAP |
| Тип | Издатель-подписчик | Очередь сообщений | Запрос-ответ |
| Способ доставки | Best effort | Guaranteed | Best effort |
| Протокол транспорта | TCP | SSL/TLS | UDP |
| Формат сообщений | Бинарный | Бинарный и текстовый | Текстовый |
| Ресурсоемкость | Низкая | Высокая | Низкая |
| Пропускная способность | Высокая | Высокая | Низкая |

AMQP имеет ту же проблему, что и HTTPS – SSL шифрование слишком трудоемкий процесс для микроконтроллеров.

CoAP хорошо подходит для отправки результатов измерений с МК, однако, разрабатываемая система предполагает гарантированное получение ответа от диспетчера для корректной работы режимов управления теплицей, что не согласуется с транспортным протоколом UDP, который используется в данном протоколе.

Исходя из этого, выбор был остановлен на использовании MQTT протокола для взаимодействия диспетчера и МК.

Разработка:

В итоге, диспетчер должен выполнять 5 основных функций:

1. Обеспечивать работу MQTT брокера, а также быть его издателем и подписчиком.
2. Предоставлять REST API для обновления конфигураций со стороны API.
3. Содержать список открытых сессий, хранить в них конфигурацию и приходящие от МК результаты измерений.
4. Периодически посылать на API отчет с основными сведениями о сессии и усредненными данными измерений.
5. На основании приходящих результатов измерений и конфигурации генерировать команды и отправлять их необходимому МК.

В ходе разработки были разработаны следующие классы:

SessionList и Session – классы для управления сессиями подключаемых микроконтроллеров.

MqttServer – класс поддерживающий работу MQTT-брокера и реализующий функционал прослушивания отдельных устройств.

CommandProducer – абстрактный интерфейс, потомки которого реализуют функционал генерации команд в зависимости от дискриминатора сессии.

Таким образом, диаграмма последовательности основного функционала диспетчера выглядит следующим образом:



2.4 Датчики

Для прототипа теплицы было решено использовать следующие датчики:

1. Датчик температуры (терморезистор) – для контроля температуры воздуха внутри теплицы.

Параметры:

Тип данных: аналоговый

Рабочее напряжение: 3.3-6 В

Рабочий ток: 10 мА

Рабочая температура: 0-70 °C

1. Датчик влаги (YL-69) – для контроля влажности почвы.

Параметры:

Тип данных: аналоговый

Рабочее напряжение: 3,3 – 5 В

Рабочий ток: 35 мА

1. Датчик света (фоторезистор) – для контроля уровня освещенности внутри теплицы.

Параметры:

Тип данных: аналоговый

Рабочее напряжение: 3.3-6 В

Рабочий ток: 10 мА

1. Датчик расхода воды (YF-S201) – для контроля полива.

Параметры:

Тип данных: аналоговый, ШИМ

Рабочее напряжение: 3,3 – 18 В

Рабочий ток: 15 Ма

Диапазон работы: 1-30 л/мин

Импульсы на литр: 450

Могут использоваться как цифровые, так и аналоговые датчики. Цифровые подключаются напрямую к пинам МК. Аналоговые подключаются либо через внешний АЦП, либо, при наличии, к пинам МК подключенным к внутреннему АЦП.

2.5 Устройства

Для прототипа теплицы было решено использовать следующие устройства:

1. Фито-лента – для освещения внутри теплицы.

Параметры:

Рабочее напряжение: 5 В DC

Максимальная мощность: 11 Вт

Вид ламп: светодиод

1. Греющий кабель (SRL 16-2) – для обогрева внутри теплицы.

Параметры:

Максимальная мощность: 16 Вт/метр

Рабочее напряжение: 220 В AC

Максимальная температура: 65 °C

1. Кулер – для вентиляции воздуха внутри теплицы.

Параметры:

Количество: 2 шт

Рабочее напряжение: 12 В DC

Максимальная мощность: 0,96 Вт

Скорость вращения: 6000 об/мин

1. Насос (КЗАЭ ЭНЦ 2,5-12) – для полива.

Рабочее напряжение: 12 В

Максимальный ток: 4 А

Давление: 2,5 кг/см2

Расход воды: 25 мл/c

Могут использоваться как цифровые, так и аналоговые устройства. Аналоговые устройства подключаются с помощью реле или транзисторов, устройства, работающие от переменного напряжения, подключаются через MOSFET-транзисторы.

2.6 МК

**Выбор МК**

При разработке системы управления переносной мини-теплицей одним из ключевых компонентов является микроконтроллер, который отвечает за управление устройствами, сбор и обработку данных. При рассмотрении вариантов микроконтроллеров, которые могут быть использованы в данной системе, обязательным требованием будет наличие Wi-Fi модуля стандарта IEEE 802.11, так как разработка на таком МК упрощается и убирает необходимость дополнительно покупать и присоединять сетевой модуль, также поддержка UART для работы конфигуратора и наличие flash памяти для хранения конфигурации.

Для выбора оптимального варианта будем использовать следующие характеристики:

1. Цена – стоимость одного микроконтроллера.
2. Память – количество внутренней памяти.
3. Доступность – возможность приобретения МК в Санкт-Петербурге в течении 7 дней.
4. Количество аналоговых пинов
5. Количество цифровых пинов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| МК | Цена  (usd) | Память  (кб) | Доступность | Аналоговые пины | Цифровые пины |
| ESP8266 | 2-3 | 80 | + | 1 | 11 |
| ESP32 | 6-8 | 320 | + | 18 | 36 |
| Raspberry PI | 35-55 | 1024 | + | 0 | 40 |
| STM32WB | 4-8 | 256 | - | 10 | 36 |

ESP8266 – 5 баллов

ESP32 – 9 баллов

Raspberry PI – 7 баллов

STM32WB – 7 баллов

Таким образом, был выбран микроконтроллер ESP32 - он соответствует требованиям проекта, имеет встроенный Wi-Fi модуль и поддерживает нужный функционал для создания удаленной системы управления.

ESP32 - это микроконтроллер с двумя ядрами, основанный на архитектуре Xtensa LX6, с тактовой частотой до 240 МГц, имеющий встроенный Wi-Fi стандарта 802.11 и Bluetooth v4.2, 520 Кб ОЗУ, 4 Мб flash-памяти, 34 цифровых входа/выхода (GPIO) и 18 аналоговых входов, а также поддержку многих периферийных устройств, таких как SPI, I2C, UART и других. ESP32 использует стандарт языка C++ 11.

Разработка:

В ходе реализации функционала МК был разработан следующий алгоритм основного цикла работы:



Схема сборки теплицы:



2.7 Сайт

**Разработка**

Как и при разработке предыдущих сервисов при создании сайта необходимо было учитывать большие требования к расширяемости функционала системы, в контексте веб-разработки это означало потребность создания хорошо структурированного и организованного приложения, позволяющего быстро создавать и редактировать страницы приложения. В связи с этим, основным языком разработки был выбран строго типизированный TypeScript.

Для ускорения разработки и повышения качества UI-интерфейса было принято решение использовать набор компонент DevExtreme. Хоть данная платформа и поддерживает такие технологии как Angular, Vue и jQuery, первоочередным приоритетом для нее является React. По этой причине предпочтение было отдано именно данному фреймворку.

Так как сайт является отдельным независимым приложением, в качестве среды выполнения был выбран лёгкий и мультиплатформенный Node.js.

Сайт сообщается со всей остальной системой через API, для которого были разработаны обобщенные классы контроллеров. Таким образом, основной функционал для всех моделей представляется абсолютно идентичным набором REST-запросов, что дало возможность разработать для сайта как собственные базовые компоненты, такие как: выпадающие списки, таблицы с поиском и фильтрацией, формы редактирования и просмотра объектов и другие.

**Безопасность**

Одним из требований к безопасности API было обязательное наличие атрибута httpOnly у всех Cookie, данное требование также релевантно и для сайта. Данное требование в первую очередь продиктовано защитой от XSS-атак и, в более общем смысле, подразумевает полный запрет доступа к конфиденциальным данным из JavaScript кода.

В связи с этим возникает проблема, JWT-аутентификация производится с помощью записи токена в заголовок Authorization, однако, мы не можем давать доступ к токену из кода. Для решения данной проблемы был модифицирован API – создан middleware-компонент перехватывающий токен из текущих Cookie браузера пользователя и подставляющий его во входящие заголовки.

**Интерфейс**

Так как сайт является основным методом взаимодействия пользователя с системой – к нему предъявляются наиболее серьезные UI/UX требования.

Главными стандартами пользовательских интерфейсов ПО являются ISO 9241 и ISO 14915. Однако, данные стандарты дают лишь общие характеристики. В связи с этим для разработки интерфейса приложения за основу был также взят набор рекомендаций Material Design от разработчиков Google.

Большинство рекомендаций данного набора уже учтены в библиотеке Bootstrap, на которой базируются используемые нами компоненты Devextreme. Так, при разработке осталось использовать только следующие паттерны:

1. «Одностраничный» сайт – переход между страницами сайта происходит путем динамической перерисовки центрального элемента сайта, все элементы навигации не перерисовываются. Данный функционал предоставляется фреймворком React.
2. «Экранные заглушки» - для создания ощущения более плавного перехода пользователя между страницами сайта приложение не должно дожидаться загрузки всех необходимых данных, а сразу отрисовывать все необходимые элементы интерфейса, «заглушая» их анимацией ожидания, пока не будет получен ответ от API. Для реализации данного функционала был написан класс ShantiApiHelper – расширение AJAX, реализующие базовые асинхронные запросы к API.
3. «Гамбургер-меню» - паттерн, позволяющий скрывать меню навигации. В нашем случае используется также и для того, чтобы незаметно перерисовать меню навигации после выбора пользователем текущего МК. Так как наибольшее количество функционала сайта связано с управлением МК, необходимо блокировать пользователю доступ к соответствующим UI-элементам до тех пор, пока он не выберет МК, с которым будет производить последующую работу.

**Внешний вид**

2.8 Конфигуратор

Разработка

Последним из разработанных приложений был конфигуратор, необходимый для первоначальной настройки теплицы после приобретения ее пользователем, а также смены конфигурации при изменении параметров беспроводной сети, передаче другому пользователю и т.д.

Для данного приложения наиболее критичными были два следующих фактора:

1. Мультиплатформенность – приложение должно корректно работать на системах семейств Windows, Linux, MacOS.
2. Небольшой вес – т.к. данное приложение используется очень редко и предоставляет небольшое количество функционала, большой вес приложения может вызвать недовольство пользователей.

В связи с данными факторами, было принято решение вести разработку приложения с помощью кроссплатформенного фреймворка Qt 6.1.1 для языка C++. Альтернативные технологии, такие как JavaFX или Windows Forms либо не предоставляют мультиплатформенности, либо слишком много весят после компиляции.

Также, вместе с приложением, должны устанавливаться необходимые USB-драйвера для подключения к МК. После подключения теплицы к ПК по USB конфигуратор автоматически обнаруживает необходимый порт по метаданным и обменивается с ним сообщениями по встроенному в микроконтроллер протоколу UART. При удачном подключении диспетчер запрашивает у МК текущую конфигурацию.

Главный функционал конфигуратора разбит на 4 UI блока, 3 из которых по умолчанию заблокированы и последовательно становятся доступными пользователю только после корректного заполнения предыдущего блока.

Блоки:

1. Подключение.

Данный блок доступен по умолчанию, при запуске конфигуратора происходит автоматический поиск необходимого порта по метаданным, в случае успешного обнаружения теплицы – порт выбирается автоматически, иначе пользователь может выбрать порт подключения самостоятельно.

1. Wi-Fi сеть.

Данный блок содержит в себе информацию, необходимую для подключения теплицы к Wi-Fi сети. Данная информация получается автоматически из текущей конфигурации МК. При изменении данных пользователем МК пытается подключиться к новой сети, в случае удачного подключения МК перезаписывает текущую конфигурацию, иначе откатывается к предыдущим настройкам.

1. Авторизация.

Блок авторизации пользователя. Пользователь вводит данные своей учетной записи в системе, после чего конфигуратор отправляет запрос к API для проверки данных. Эти данные не сохраняются в МК.

1. Регистрация теплицы.

Блок идентификационных данных МК в системе, содержит постоянный MAC-адрес микроконтроллера и уникальный регистрационный ключ теплицы в системе. Данная информация также получается из конфигурации МК после подключения. Для изменения этих данных, например, для смены пользователя, конфигуратор отправляет запрос к API с постоянными данными микроконтроллера, после чего происходит привязка к новому пользователю и выдача нового регистрационного ключа, который сохраняется в конфигурации МК.