«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	11.03.02 - Инфокоммуникационные технологии и системы связи
Профиль	Радиоэлектронные средства информационного обмена
Факультет	РТ
Кафедра	РЭС

К защите допустить

Зав. кафедрой Малышев В.Н.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: ЛАБОРАТОРНЫЙ СТЕНД "ОРГАНИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО ВЗАИМОДЕЙСТВИЯ"

Студент		_____ *подпись*	Дубовский И.С.
Руководитель	доцент *(Уч. степень, уч. звание)*	_____ *подпись*	Воронов А.В.
Консультанты	асс. каф. РЭС *(Уч. степень, уч. звание)*	_____ *подпись*	Проценко И.М.
	к.т.н., доц. *(Уч. степень, уч. звание)*	_____ *подпись*	Иванов А.Н.

Санкт-Петербург

2023

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю
Зав. кафедрой РЭС
_____ Малышев В.Н.
«___»_____20__ г.

Студент Дубовский И.С. Группа 9181

Тема работы: Лабораторный стенд "Организация клиент-серверного взаимодействия"

Место выполнения ВКР: СПБГЭТУ «ЛЭТИ», каф. РЭС

Исходные данные (технические требования): разработать программное обеспечение на C++ для микроконтроллера ESP8266 в составе учебного макета для обеспечения клиент-серверного взаимодействия.

Содержание ВКР: анализ макета, прототипирование информационной системы, разработка программного обеспечения.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал

Дополнительные разделы: Безопасность жизнедеятельности

Дата выдачи задания
«___»_____20___ г.

Дата представления ВКР к защите
«___»_____20__ г.

Студент _____ Дубовский И.С.

Руководитель доцент _____ Воронов А.В.
(Уч. степень, уч. звание)

Консультант асс. каф. РЭС _____ Проценко И.М.
(Уч. степень, уч. звание)

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Зав. кафедрой РЭС
_____Малышев В.И.
«___»_____20___ г.

Студент(ка) Дубовский И.С. Группа 9181
Тема работы: Лабораторный стенд "Организация клиент-серверного взаимодействия"

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	25.02 – 02.03
2	Анализ макета и используемых технологий	03.03 – 15.03
3	Прототипирование программного обеспечения	16.03 – 30.03
4	Разработка программного обеспечения	31.03 – 30.04
5	Оформление пояснительной записки	01.04 – 19.04
6	Оформление иллюстративного материала	20.04 – 25.04

Студент _____ Дубовский И.С.

Руководитель доцент _____ Воронов А.В.
(Уч. степень, уч. звание)

Консультант асс. каф. РЭС _____ Проценко И.М.
(Уч. степень, уч. звание)

РЕФЕРАТ

Предмет данной работы – разработка программного обеспечения для учебного лабораторного макетов на базе микроконтроллера ESP8266 с целью организации клиент-серверного взаимодействия между макетом и компьютером. Макет представляется собой ESP8266 соединенный с набором сенсоров: детектор жестов, датчик давления и температуры, датчик освещённости, акселерометр, индикаторов: три светодиодных лампы и светодиодная лента, и элементов управления в виде трех кнопок, Тема исследования связана с изучением беспроводной связи в контексте интернета вещей.

Методология исследования включает в себя теоретическое обоснование и обзор выбранных технологий, прототипирование информационной системы n разработку ПО для макета. Используются классические методы системного анализа, дизайна и программирования.

Результатом работы является разработанное программное обеспечение для микроконтроллера ESP8266, предоставляющее интерфейс для удаленного взаимодействия с макетом в Wi-Fi сети, управления подключенными к нему устройствами и сбора данных.

Областью применения результатов работы является сфера образования, и, в частности, процесс обучения студентов основам работы с периферийными устройствами и IoT в рамках беспроводных сетей.

Разработанное программное обеспечение может быть применено как для изучения принципов работы с периферийными устройствами и основ сферы Интернета вещей, давая возможность исследовать их работу, модифицировать код и дорабатывать как для изучения устройств, так и для работы с точки зрения клиента – взаимодействия и обработки собираемых устройством данных.

ABSTRACT

This work focuses on developing software for educational laboratory model based on ESP8266, aimed at enabling client-server interaction for remote control and data collection from connected peripheral sensors. The developed software provides an interface for remote interaction with ESP8266 within a Wi-Fi network. Result of work is intended to be used in educational process as a way to learn some of the key technologies in Internet-of-Things and specifics of work with some of the devices.

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 6
ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 8
ВВЕДЕНИЕ 9
1. ТЕОРЕТИЧЕСКИЙ ОБЗОР МАКЕТА 10
1.1. Общие сведения 10
1.2. Обзор NodeMCU ESP8266 11
1.3. Обзор датчиков и индикаторов 13
1.3.1. ADXL345 14
1.3.2. BMP180 14
1.3.3. APDS-9960 15
1.3.4. Светодиоды, кнопки и датчик освещенности 15
2. ПРОТОТИПИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 17
2.1. Средства разработки 17
2.2. Сторонние библиотеки 18
2.3. Общая структура программы 19
2.4. Серверная часть приложения 21
2.4.1. HTTP сервер 21
2.4.2. RESTful HTTP 22
2.4.4. Формат передачи данных 25
2.4.5. Обработка запросов 29
2.5. Выводы 33
3. АНАЛИЗ ПРИМЕНЕНИЙ 34
4. БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ 38
4.1. Обзор стандартов 38
4.2. REST HTTP 39
4.3. JSON 40
4.4. Заключение 41

ЗАКЛЮЧЕНИЕ 43
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 45
ПРИЛОЖЕНИЕ А 46
server.ino 46
utils.h 54

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяют следующие термины с соответствующими определениями:

HTTP – Hypertext Transfer Protocol (Протокол передачи гипертекста)

UDP – User Datagram Protocol (Протокол пользовательских датаграмм)

TCP – Transmission Control Protocol (Протокол управления передачей)

IoT – Internet of Things (Интернет вещей)

XML – Extensible Markup Language (Расширяемый язык разметки)

JSON – JavaScript Object Notation (Нотация объектов JavaScript)

IDE – Integrated Development Environment (Интегрированная среда разработки)

HTML – Hypertext Markup Language (Язык гипертекстовой разметки)

API – Application Programming Interface (Интерфейс программирования приложений)

ВВЕДЕНИЕ

В современном обществе цифровизация и автоматизация процессов играют все более важную роль. Интернет вещей (Internet of Things, IoT) стал, для многих, неотъемлемым элементом повседневной жизни и промышленности, обеспечивая связь между физическими объектами и цифровыми системами. В этом контексте, разработка программного обеспечения для устройств IoT становится все более актуальной и важной задачей. Основным элементом работы является плата ESP8266, широко применяемая в качестве модуля управления и беспроводной связи за счёт своей дешевизны и эффективности.

Созданное в итоге работы программного обеспечения позволит его использование в образовательном процессе с двух сторон: серверной: для изучения непосредственного взаимодействия микроконтроллера с периферийными устройствами и передачи данных, и клиентской: для обработки и анализа собранных и переданных макетом данных.

Непосредственной целью данной дипломной работы является разработка программного обеспечения для обеспечения клиент-серверного взаимодействия на плате ESP8266, подключенной к набору сенсоров и индикаторов.

Задачи разработки включают: анализ и выбор подходящих технологий для разработки ПО, проектирование и реализация ПО, анализ возможных применений и улучшений в будущем и изучение стандартов соответствующим выбранным технологиям и проверка программы на соответствие им.

Объектом разработки является учебный макет на базе микроконтроллера ESP8266, подключенного к набору сенсоров. Предметом разработки является программное обеспечение, обеспечивающее клиент-серверное взаимодействие между учебным макетом и компьютером.

1. ТЕОРЕТИЧЕСКИЙ ОБЗОР МАКЕТА

1.1. Общие сведения

В работе рассматривается макет, предназначенный для использования в ходе лабораторных работ отдельно или вместе с аналогичными устройствами в составе беспроводной сети. Общая структурная схема лаборатории представлена на рисунке 1.

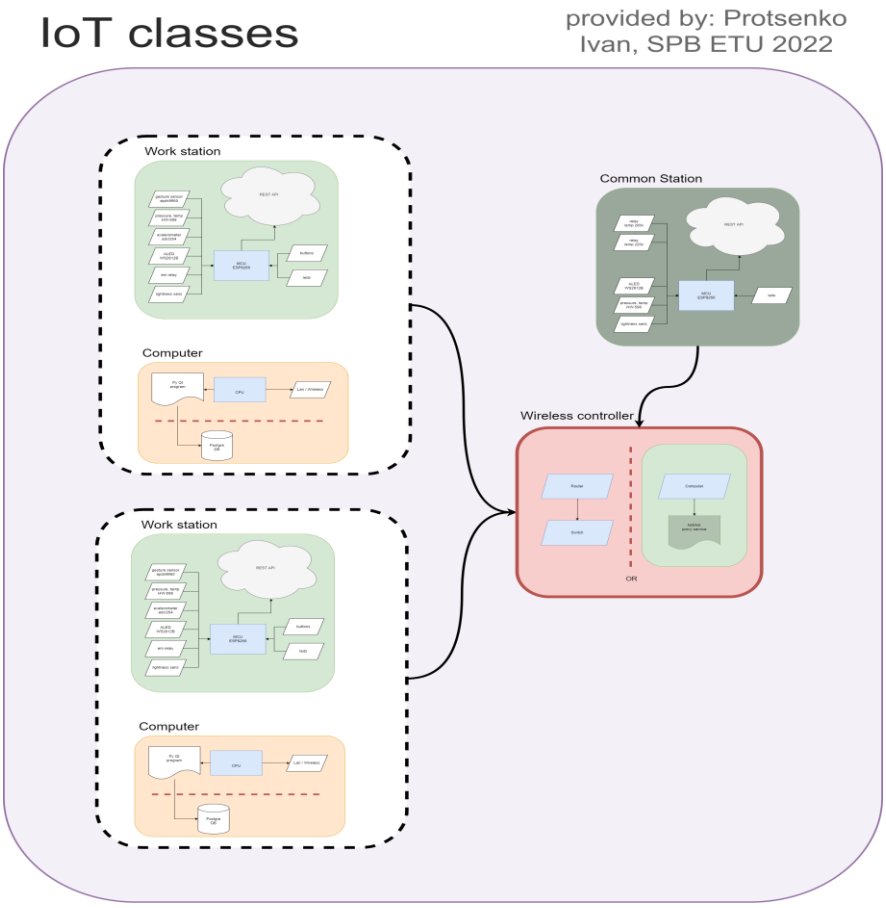

Рисунок 1. Схема лаборатории

Предполагается, что макет будет подключен к сети Wi-Fi, созданной отдельным от него устройством, а управление будет осуществляться посредством взаимодействия с сервером, предоставляемым макетом, с помощью компьютера, расположенного в той же сети. Также плата ESP8266 предоставляет последовательный порт для отправки отладочных сообщений, который позволяет подключить макет напрямую к компьютеру по USB. Этот же канал используется для загрузки программного обеспечения для прошивки платы.

Макет построен вокруг микроконтроллера ESP8266, к пинам которому подключены используемые сенсоры, индикаторы и переключатели. Он обеспечивает выполнение программы для сбора данных, управления подключен-

ными периферийными устройствами, подключение к беспроводной сети для приема и отправки данных, а также проводную связь через порт microUSB,

Сенсоры, индикаторы и переключатели, подключенные к микроконтроллеру, позволяют измерять различные параметры окружающей среды, состояния макета и контролировать состояние устройств. Например, возможно измерять температуру, атмосферное давление, уровень освещенности, ускорение устройства с помощью акселерометра и обнаруживать жесты. Индикаторы представлены тремя LED лампочками и светодиодной полоской с 8 элементами, которые могут быть использованы для отображения данных.

1.2. Обзор NodeMCU ESP8266

Рассмотрим подробнее используемую в макете плату NodeMCU ESP8266, для которой и будет производится разработка программного обеспечения.

NodeMCU ESP8266 – это открытая платформа для проектирования решений в IoT (Интернет вещей), базирующаяся на чипсете Expressif ESP8266. Основным и наиболее распространенным языком программирования для этой платформы является Lua, но также полностью поддерживается Arduino IDE, основанная на языке C++. Далее на рисунке 2 представлен внешний вид и распиновка платы из официальной документации [13].

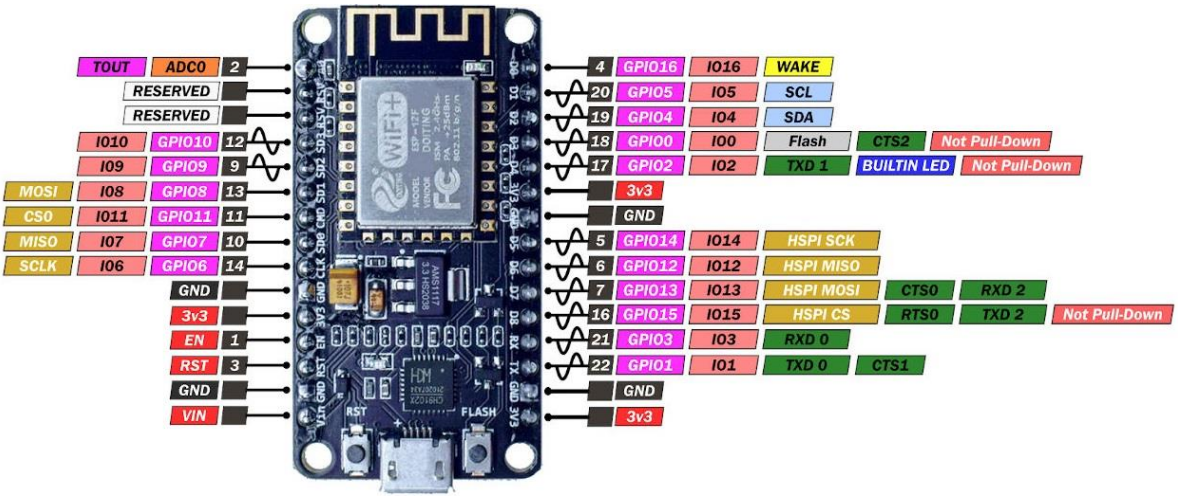

Рисунок 2. Изображение и распиновка ESP8266

Рассмотрим основные характеристики устройства.

В основе этой платы находится Wi-Fi чипсет ESP8266 от компании Espressif Systems. Модуль обладает 32-битным процессором и встроенным Wi-Fi, чтоделает его идеальным решением для различных IoT-приложений.

Устройство позволяет как подключаться к Wi-fi сетям, так и может служить самостоятельной точкой доступа для других устройств.

GPIO-пины: NodeMCU имеет несколько общих цифровых вводов-выводов (GPIO), которые можно использовать для чтения данных с сенсоров или управления другими устройствами.

Micro-USB-порт: Плата оборудована портом micro-USB для программирования и питания. Он также может использоваться в качестве последовательного порта для полученияф вывода отладочных сообщений с платы.

Также стоит отметить, что NodeMCU ESP8266 поддерживает интерфейсы I2C (шинная система передачи данных), SPI (последовательный периферийный интерфейс), что позволяет взаимодействовать с различными устройствами и сенсорами.

Простота использования: NodeMCU является одной из самых популярных платформ для разработчиков IoT из-за ее простоты использования и доступности, и как следствия для устройства существует множество библиотек и документаций, существенно упрощающих процесс разработки.

Отдельно стоит отметить один из наиболее важных для устройств IoT факторов: энергопотребление. Модуль ESP8266 обладает различными режимами работы, каждый из которых соответствует выполняемым в данный момент задачам, что позволяет существенно оптимизировать суммарное потребление устройства на протяжении работы. Далее представлены существующие режимы работы и ориентировочные значения энергопотребления для них:

1. Режим работы: При активной работе, включая передачу данных по Wi-Fi, ESP8266 потребляет от 70 до 170 мА.
2. Режим ожидания (idle mode): Когда Wi-Fi включен, но не передает данные, потребление тока составляет примерно 15 мА.
3. Режим глубокого сна (deep sleep mode): В этом режиме, который используется для максимальной экономии энергии, ESP8266 может потреблять всего около 10 мкА.
4. Режим модем-сна (modem sleep mode): Это режим сна с включенным процессором, но отключенным Wi-Fi. В этом состоянии потребление составляет около 15-20 мА.

Эти значения могут иметь небольшие отличия в зависимости от конкретной модели платы и настроек прошивки. Важно отметить, что энергопотребление может также значительно увеличиться при передаче данных, особенно при работе в сетях с большим уровнем шума.

NodeMCU ESP8266 также обладает встроенной флэш-памятью, которая может использоваться для хранения программного кода и данных. Объем флэш-памяти варьируется в зависимости от модели платы, но обычно составляет от 4 МБ до 16 МБ – в данном случае используется третья (v3) версия платы, использующая 4 МБ памяти..

В заключение, NodeMCU ESP8266 – это мощная и универсальная плата для разработки IoT-решений. Ее простота использования, поддержка популярных сред разработки и функционал делают ее отличным выбором для создания различных проектов в области Интернета вещей, а значит актуальной и достаточно удобной для использования в изучении этой сферы.

1.3. Обзор датчиков и индикаторов

К макету подключены четыре отдельных датчика, три кнопки, а также светодиодная лента из 8-ми элементов и 3 отдельных светодиода. Общая структура соединений модуля макета представлена на рисунке 4. Рассмотрим подробнее остальные используемые устройства.

Рисунок 3. Схема подключений макета

1.3.1. ADXL345

Датчик ADXL345 – это высокоточный, низкопотребляющий трехосевой акселерометр с цифровым интерфейсом. Он производится компанией Analog Devices. Рассмотрим особенности устройства.

ADXL345 является трехосевым акселерометром, то есть может измерять ускорение по трем осям: X, Y и Z. Это делает его идеальным для широкого спектра применений, включая мониторинг физической активности, навигацию, игровые устройства, промышленное оборудование и многое другое.

Низкое энергопотребление: ADXL345 был разработан для оптимизации энергопотребления, и изначально предназначен для батарейных устройств, что делает его удобным для использования в устройствам IoT.

Датчик также использует цифровой интерфейс для передачи данных, что упрощает интеграцию с микроконтроллерами и другими цифровыми устройствами, в данном случае ESP8266. Он поддерживает как I2C, так и SPI интерфейсы. В данном случае он подключен через интерфейс I2C.

Различные режимы работы: Устройство имеет режимы измерения полного диапазона ускорения ±2g, ±4g, ±8g, и ±16g.

Дополнительные функции: ADXL345 предлагает функции, такие как прерывания активности/неактивности, прерывание свободного падения, прерывание наклона и т.д.

1.3.2. BMP180

Датчик BMP180 – это высокоточный, низкопотребляющий датчик атмосферного давления от компании Bosch Sensortec. Он также может измерять температуру.

Основные характеристики и функции BMP180 включают:

Измерение давления: BMP180 может измерять атмосферное давление в диапазоне от 300 до 1100 гектопаскалей (hPa).

Измерение температуры: Кроме измерения давления, BMP180 также может измерять температуру в диапазоне от -40 до +85 градусов Цельсия.

Низкое энергопотребление: BMP180 был спроектирован для минимального энергопотребления, что делает его идеальным для батарейных устройств и устройств с низким энергопотреблением.

Цифровой интерфейс: BMP180 использует цифровой I2C интерфейс для передачи данных, что упрощает его интеграцию с микроконтроллерами и другими цифровыми устройствами.

Высокая точность: Датчик обеспечивает высокую точность измерения давления, которая может быть улучшена через калибровку и корректировку в соответствии с температурой.

1.3.3. APDS-9960

APDS-9960 – это ультра-компактный, амбиентный световой и ближний ИК датчик от компании Broadcom. Этот модуль также включает в себя полнофункциональный цветной световой датчик, датчик приближения, и датчик жестов, делая его чрезвычайно многофункциональным.

Основные характеристики и функции APDS-9960 включают:

Датчик амбиентного света и ИК: APDS-9960 может измерять уровень амбиентного света, а также интенсивность ближнего ИК света.

Датчик цвета: APDS-9960 включает в себя четырехканальный датчик цвета, который может измерять красный, зеленый, синий и инфракрасный свет, позволяя определять и отслеживать цвета в своем окружении.

Датчик приближения: Этот датчик может определить, когда объект приближается к устройству, что делает его идеальным для различных приложений, таких как отключение экрана телефона при приближении к уху.

Датчик жестов: APDS-9960 также содержит датчик жестов, который может распознавать простые жесты рук. Это делает его полезным для различных интерактивных приложений.

Цифровой интерфейс: APDS-9960 использует цифровой I2C интерфейс для передачи данных, что упрощает его интеграцию с микроконтроллерами и другими цифровыми устройствами.

1.3.4. Светодиоды, кнопки и датчик освещенности

Отдельный датчик освещенности также подключен к плате через аналоговый порт, он не предоставляет никаких данных кроме освещенности в люксах.

Макет также включает три отдельных светодиода и кнопки, подключенных напрямую к устройству, и 8-ми элементную светодиодную ленту WS2812B. Рассмотрим последнюю подробней.

WS2812B – светодиодная лента, предоставляющая управляемый интерфейс с встроенным контроллером – каждый из светодиодов на ленте может быть индивидуально адресован и управляем с помощью этого чипа.

Светодиодная лента с 8 светодиодами WS2812B имеет следующие характеристики:

Индивидуальное управление: каждый из светодиодов на ленте может быть контролирован отдельно, что позволяет менять цвет и яркость каждого светодиода независимо.

Цвет: WS2812B поддерживает полный спектр цветов. Чип использует модуляцию ширины импульса (PWM) для управления каждым из трех цветов светодиода (красный, зеленый и синий), позволяя создавать буквально миллионы цветовых комбинаций.

Однопроводной интерфейс: WS2812B использует однопроводной интерфейс для передачи данных, то есть управление производится посредством подключения подключения через единственный пин.

2. ПРОТОТИПИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1. Средства разработки

Для разработки была выбрана Arduino IDE, предоставляющая возможность разработки на C++. Так как основным языком для работы с ESP8266 часто считают Lua, необходимо рассмотреть причины данного выбора.

Arduino IDE – это интегрированная среда разработки (IDE), которая обеспечивает простоту использования и гибкость для программирования микроконтроллеров, в том числе таких как NodeMCU ESP8266.

Основные особенности:

Простота использования: Arduino IDE предоставляет простой и интуитивно понятный интерфейс, который облегчает программирование и прошивку. Среда позволяет разрабатывать, проверять и загружать программы исключительно в её контексте.

Обширная библиотека: Arduino IDE поставляется с обширной библиотекой, которая включает в себя множество полезных функций и примеров кода. Библиотека содержит готовые решения для различных задач, таких как работа с Wi-Fi, HTTP-запросы, управление пинами и другие. Это позволяет быстро и легко реализовывать различные проекты на основе ESP8266, минимизируя необходимость написания низкоуровневого кода с нуля. Многие разработчики датчиков также предоставляют библиотеки в первую очередь именно для Arduino.

Поддержка большого сообщества: Arduino IDE имеет активное сообщество разработчиков, где можно найти множество документации, учебных материалов и проектов, так как используется для существенно более широкого спектра микроконтроллеров, чем Lua.

Стоит также отметить, что плата NodeMCU ESP8266 использует для порта micro-USB чип CH340. Он служит мостом между микроконтроллером и USB-портом компьютера, который позволяет загружать прошивку в плату, а также передавать между собой данные, таким образом данный порт при разработке обеспечивает и питание, и взаимодействия с макетом, но для последнего также была выполнена установка драйвера USB-UART CH340G.\

Для поддержки работы с платой NodeMCU ESP8266 использовался фреймворк ESP8266 Arduino Core для Arduino IDE. Он включает в себя драйверы и библиотеки, позволяющие взаимодействовать с различными компонентами и модулями, такими как Wi-Fi, GPIO (General Purpose Input/Output), I2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface) и другими периферийными устройствами.

2.2. Сторонние библиотеки

Для взаимодействия платы с кнопками (считывания их состояния), датчиком освещенности (получения результатов измерений) и светодиодными лампами (установки и чтения их состояния) достаточно имеющихся в составе Arduino IDE и упомянутого выше фреймворка библиотек, но для поддержки взаимодействия с другими датчиками необходимо применение сторонних библиотек, чтобы избежать низкоуровневого взаимодействия непосредственно с устройствами и упростить процесс разработки. Рассмотрим применяемые библиотеки.

Для работы со светодиодной лентой WS2812b была использована библиотека FastLED. Она использует низкоуровневые операции и оптимизированный код, чтобы обеспечить быструю обработку цветовых данных и предоставляет простой интерфейс для записи данных о цвете, яркости и других параметров. FastLED предлагает множество встроенных функций и эффектов освещения, таких как плавное затухание, волновые эффекты и многое другое.

Для работы с датчиком давления и температуры используется библиотека SparkFun BMP180_Breakout Arduino Library, предоставляющая интерфейс для считывания температуры, представленной в градусах Цельсия, и давления, представленного в миллибарах. Библиотека также предоставляет возможность измерить высоту над уровнем моря, а также исключить эффект высоты над уровнем моря при получении значения давления.

Для акселерометра ADXL345 применяется библиотека Adafruit ADXL345. Библиотека получает не только измерять значения по осям, но и отслеживать события соответствующие резким изменениях в данных параметрах.

Для датчика жестов используется SparkFun APDS9960. Она позволяет управлять работой отдельных компонентов, получать напрямую обнаружен-

ный жест (движение объектов влево, вправо, вверх, вниз, к датчику и от него), уровни освещения и определять приближение объектов.

Также для работы с форматом JSON используется библиотека ArduinoJSON, позволяющая упростить процесс сериализации и десериализации JSON, которая будет рассмотрена подробнее в пункте 2.4.3.

2.3. Общая структура программы

Все программы в Arduino IDE используют общую базовую структуру из двух основных функций:

setup(): Этот раздел программы выполняется один раз, при первом включении платы. Он используется для инициализации оборудования, запуска сервера и установки необходимых переменных.

loop(): Этот раздел программы начинает выполнятся после завершение функции setup() и работает циклично, пока плата не будет выключена – в данном случае не используется никаких дополнительных выключателей и единственным способом штатно отключить ESP8266 будет остановка питания. На рисунке 4 приведен общий алгоритм работы программы, описывающий основные выполняемые в данных функциях задачи.

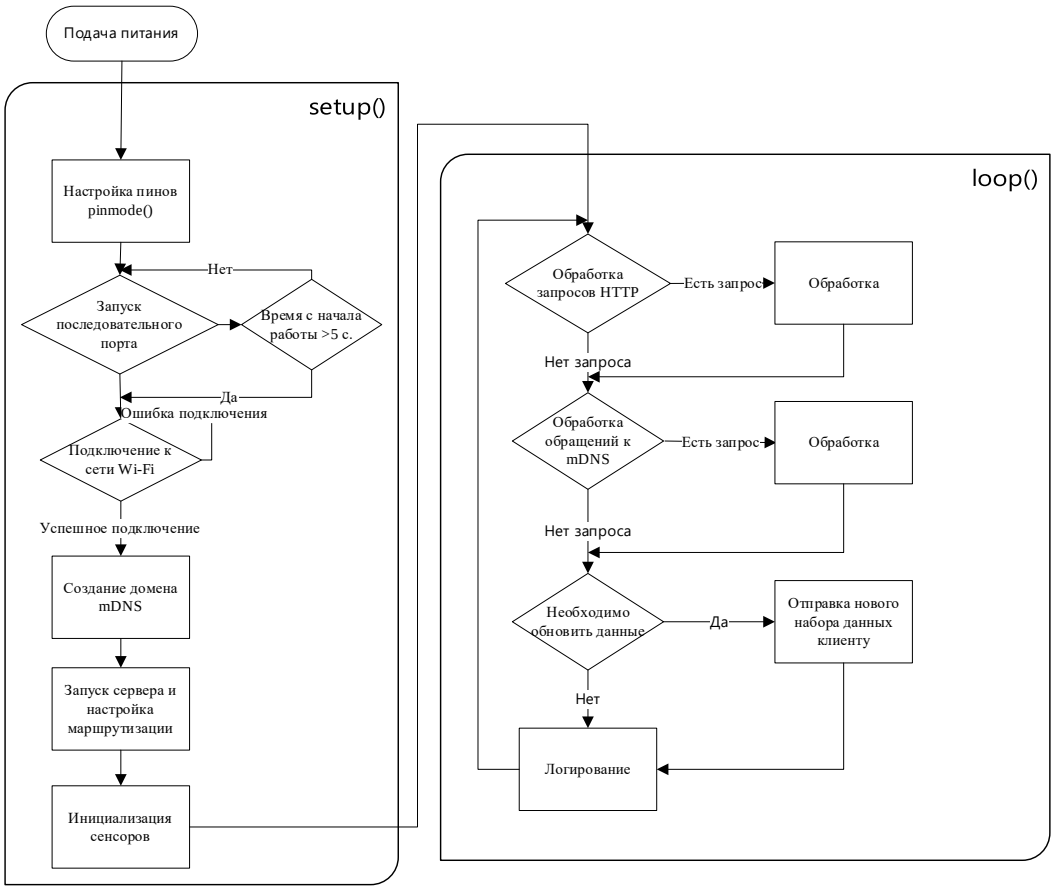

Рисунок 4. Алгоритм

Рассмотрим их немного подробнее.

В функции setup() в первую очередь выполняется настройка пинов платы таким образом, чтобы была возможна запись данных по адресам светодиодных ламп (режим OUTPUT для PIN_LED1 – PIN_LED3), а также получение данных с кнопок (режим INPUT_PULLUP для PIN_BUTT1 – PIN_BUTT3). Значения параметров PIN_ соответствуют пинам, к которым подключены элементы, и заданы директивами #define в программе.

После этого производится открытие последовательного порта со скоростью работы 115 200 бод и ожидание подключения к нему – так как макет допускает исключительно беспроводную работу, если по истечению 5 секунд успешное подключение не совершено, этот этап будет пропущен.

Далее производится подключение к WiFi сети по заранее заданным в программе данным в режиме WIFI_STA, позволяющим плате работать в качестве клиента в сети. Подключение считается необходимым для работы и попытки будут продолжаться до успешного выполнения.

После успешного соединения, в сети при помощи многоадресного протокола DNS (mDNS) создаётся домен esp8266.local, задаётся таблица маршрутизации сервера и производится его запуск, при этом доступен сервер будет как по полученному IP адресу в WiFi сети, так и по домену. Более подробно особенности его работы будут рассмотрены далее.

Последним этапом является инициализация и проверка подключенных устройств: светодиодной полосы, датчика давления, акселерометра и датчика жестов. При ошибке проверки запуск продолжится независимо от их статуса, и в итоге управление передаётся функции loop().

В loop() выполняется четыре задачи: обработка запросов к домену службой mDNS для перенаправления их непосредственно на сервер, после чего проводится обработка полученных сервером запросом, в ходе которой и проводится работа с сенсорами и т.п., далее проверяется, установлено ли клиентом время для обновление и, если это так, проверяется прошедшее время с последнего обновления данных и производится отправка пакета с новыми данными с сенсоров. В завершении цикла запускается функции для отправки записей в лог, передаваемый по последовательному порту для отслеживания работы устройства.

Функция setup(), таким образом, задаёт всю логику дальнейшей работы устройства путем настройки соответствующих объектов, а функция loop() занимается исполнением логики вызывая для них функции-обработчики.

2.4. Серверная часть приложения

2.4.1. HTTP сервер

Для взаимодействия внешних устройств (например, компьютера) с платой в рамках беспроводной сети выбран HTTP сервер. Данный протокол был выбран по следующим причинам:

HTTP хорошо известный и широко поддерживаемый протокол. Это означает, что практически все языки программирования, браузеры и операционные системы в том или ином виде поддерживают взаимодействие с ним (например, язык программирования Python предоставляет простой интерфейс для взаимодействия с HTTP в виде библиотеки requests).

HTTP – это простой протокол. Реализация, использование и модифицирование сервера на нём доступны при минимальных затратах времени, а из предыдущего пункта также следует избыток документации и образовательных материалов по протоколу.

Также HTTP использует протокол TCP, гарантируя доставку (или сообщение о её провале) сообщений между клиентом и сервером.

Основной альтернативной в сфере Интернета вещей для HTTP является протокол MQTT, разработанных специально для подобных применений. По технически параметрам он близок к HTTP, проигрывая в стоимости (с точки зрения нагрузки на сеть) установления и разрыва соединений, но выигрывая в затратах на пересылках сообщений. Это также позволяет ему выиграть с точки зрения энергопотребления, что может быть особенно полезно для применений в сфере Интернета вещей

В данном случае при работе платы в составе учебного макета подразумевается постоянного транслирования ни большого количества пакетов, ни пакетов с большим объемом данных, поэтому HTTP был выбран как наиболее удобный и легко интегрируемый протокол, который все ещё является гораздо более распространенным не только в сфере Интернета вещей, но и в целом в сфере информационных технологий, что особенно важно для применения макета в образовательных целях.

Дополнительно стоит упомянуть ещё один способ связи, часто используемый в небольших устройствах, в том числе для Интернета вещей – протокол UDP. Прямое подключение к сокету и передача данных таким образом позволила бы существенно сократить (даже по сравнению с MQTT) вес каждого отправляемого пакета и нагрузка по сети – некоторые измерения показывают, что UDP приводит к 5-10 кратному выигрышу в нагрузке на сеть и 2-3 кратному выигрышу по потребляемой мощности для передачи одного пакета [9].

Тем не менее, взаимодействие с протоколом UDP как с точки зрения разработки, так и с точки зрения образовательного процесса и взаимодействия с макетом студентов было бы существенно усложнено, так как протокол является низкоуровневым и чтение и передача нагрузки подобной данным в формате JSON в HTTP/MQTT гораздо проще. При этом протокол не гарантирует проверки доставки сообщений и является ненадежным, что делает его неприменимым в том числе и во многих реальных устройствах. На практике наиболее эффективным его использование было бы при передаче потока большого количества сообщений с небольшой нагрузкой.

2.4.2. RESTful HTTP

Для HTTP сервера в приложении был выбран архитектурный стиль REST (Representational State Transfer), описывающий принципы взаимодействия компонентов приложений в сети. приложений. REST основан на принципах, установленных в HTTP-протоколе, и обеспечивает универсальный и гибкий подход к созданию веб-сервисов.

Основные принципы стиля REST:

Клиент-серверная архитектура: REST предполагает разделение клиентской и серверной логики. Клиенты отправляют запросы на сервер, который обрабатывает эти запросы и возвращает ответы. Это позволяет достичь легковесности клиентов и повысить масштабируемость системы.

Без состояния (stateless): Каждый запрос клиента должен содержать всю необходимую информацию для его обработки на сервере. Сервер не хранит состояние между запросами от клиента. Это позволяет повысить масштабируемость и надежность системы.

Кэширование: REST поддерживает возможность кэширования ответов сервера на стороне клиента. Клиенты могут использовать кэш для повторного использования ранее полученных данных, что уменьшает нагрузку на сервер и улучшает производительность.

Единообразный интерфейс: REST опирается на использование универсальных стандартов и протоколов, таких как HTTP, URI (Uniform Resource Identifier), JSON (JavaScript Object Notation) или XML (eXtensible Markup Language). Это обеспечивает простоту, расширяемость и независимость между клиентами и серверами.

Многоуровневая система: REST поддерживает иерархическую структуру системы с промежуточными компонентами, такими как прокси, балансировщики нагрузки или кэши. Это позволяет улучшить масштабируемость и безопасность системы.

Основной причиной для использования REST в данном приложении является упрощение взаимодействия с функциями посредством HTTP за счёт создания простой и легко читаемой для человека структуры адресации. Подробнее выбор REST и требования к его имплементации в стандарте ГОСТ рассмотрены в главе 4.

Для работы создана схема адресации, приведенная далее в таблице 1. Все адреса определяются относительно корневого, он может быть как полученным устройством IP адресом, так и заданным mDNS доменом esp8266.local.

Таблица 1. Структура REST адресации.

Метод HTTP	Адрес	Действие
GET	/	Предоставляет интерфейс для отправки POST запроса, аналогичного POST /sensors/leds и отображает сведения о макете.
GET	/update	Должен передавать аргумент в целочисленном виде,

		указывающий частоту обновления данных в миллисекундах
GET	/sensors	Возвращает данные всех сенсоров
GET	/sensors/leds	Возвращает состояние светодиодов и цвета на светодиодной ленте
POST	/sensors/leds	Включает/выключает светодиоды и управляет цветами на световой ленте
GET	/sensors/buttons	Возвращает состояние кнопок
GET	/sensors/lightning	Возвращает параметры освещения, записанные датчиком жестов, и отдельным датчиком
GET	/sensors/gestures	Возвращает данные об обнаруженных жестах
GET	/sensors/pressure	Возвращает температуру и атмосферное давление
GET	/sensors/axes	Возвращает данные акселерометра

Таким образом предоставляется полноценный программный интерфейс для взаимодействия с компонентами программы. Любые запросы, обработка

которой не подпадает под описанную выше систему адресов будут перенаправлены на страницу с ошибкой.

2.4.4. Формат передачи данных

Для данного проекта в качестве формата обмена данными был выбран JSON, Это решение было обусловлено несколькими ключевыми факторами, которые делают JSON идеальным выбором для данного сценария.

Прежде всего, JSON является легким и простым для разбора, что делает его очень эффективным для устройств с ограниченными ресурсами, таких как ESP8266 и устройства Интернета вещей в целом. Поскольку JSON – это текстовый формат, он занимает минимум памяти и может легко передаваться по сети. Это важно для микроконтроллеров, который имеет ограниченную вычислительную мощность и память.

Во-вторых, JSON обеспечивает человекочитаемую и самодокументирующуюся структуру для представления данных. В нем используется простой формат пары ключ-значение, который интуитивно понятен и прост для разработчиков. Это делает его отличным выбором для обмена структурированными данными между веб-сервером ESP8266 и другими устройствами или сервисами, такими как веб-браузеры или API.

Еще одним преимуществом JSON является его совместимость с широким спектром языков программирования и платформ. Эта универсальность обеспечивает возможность легко интегрироваться с веб-сервисами, базами данных или другими устройствами, позволяя легко создавать клиенты для реализованного сервера в разных технических условиях. Популярность JSON, наличие большого количества библиотек и обширной документации, делают его удобным выбором для упрощения процесса разработки. Это также крайне важно с точки зрения использования учебного макета в образовательном процессе, так как формат является стандартом и распространён в сфере Интернета вещей и на коммерческом уровне и знакомство с ним может быть полезным навыком.

JSON также поддерживает вложенные структуры, массивы и различные типы данных, что позволяет представлять сложные иерархии данных. Такая универсальность особенно полезна при работе с данными датчиков, настройками конфигурации и другой многомерной информацией, часто встречающейся в IoT-приложениях. Способность JSON работать со структу-

рированными и разнообразными данными делает его подходящим выбором для требований проекта.

Альтернативные способы передачи данных проигрывает в простоте интеграции с другими сервисами, удобстве чтения для человека или в простоте разработки. Например, если бы данные передавались в произвольном текстовом формате, для написания клиента потребовалось бы писать парсер данных с нуля, что в некоторых языках программирования может быть крайне трудоемким процессом, при этом в процессе разработки самого сервера пришлось бы существенно усложнить программу для корректного формирования данных,

XML(«расширяемый язык разметки») предлагает существенно более сложную и детальную структуру, которая полезна для передачи больших страниц, но в данном случае необходимо передавать лишь небольшой объем данных и использование XML привело бы к разрастанию объема передаваемой информации, увеличению затрат на обработку данных платой и уменьшению его доступности для чтения человеком.

Для взаимодействия с JSON, как уже упоминалось в пункте 2.2., была выбрана библиотека ArduinoJSON. Она является фактически единственной актуальной и поддерживаемой библиотекой для обработки JSON совместимой с ESP8266, и предоставляет широкий набор функций для взаимодействия с форматом. Наиболее важными для работы является её основной функционал – сериализация и десериализация JSON файлов, т.е. конвертация данных из исходных форматов (например, целочисленного и строки) в единый объект JSON, и обратный процесс. Библиотека оптимизирована для работы с ограниченными ресурсами памяти и имеет небольшой размер кода. ArduinoJSON полностью соответствует стандартам JSON, поддерживает различные типы данных и обладает хорошей производительностью. Она обладает подробной документацией и активным сообществом пользователей, что делает ее популярным выбором для работы с JSON на ESP8266

Для отправляемых клиенту в ответ на GET запросы данных, описывающих состояние подключенных периферийных устройств, была разработана приведенная на рисунке 6 структура данных.

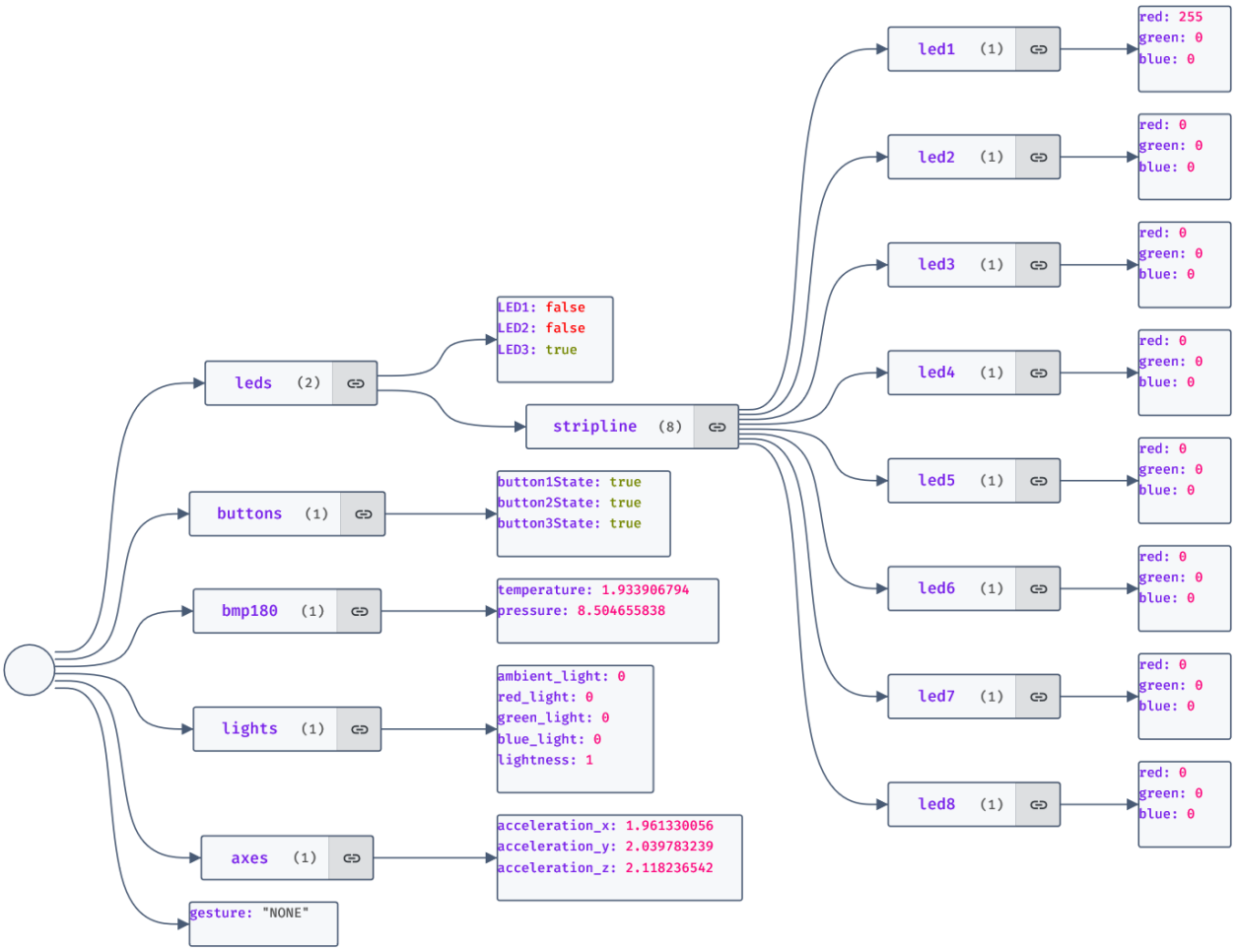

Рисунок 5. Структура JSON для GET /sensors запроса

Здесь элементы верхнего уровня соотнесены с адресами для таблицы маршрутизации, приведенной в таблице 1, и содержат в качестве дочерних элементов данные, которые соответствуют требуемых для данных адресов. Соответственно при запросе, например, GET /sensors/lights будут возвращены исключительно значения в группе lights на иллюстрации. Данный подход существенно упрощает отправку данных при запросе их для отдельного блока устройств, и также делает формат более читаемым для человека, группируя элементы по функционалу.

Для хранения данных о статусе светодиодных ламп (leds) и кнопок (buttons) используется булевый формат, для данных об уровнях света (lights) целочисленный, цвета элементов светодиодной ленты (leds -> stripline) передаются в виде трёх целых чисел, соответствующих цветовой модели RGB, акселерометр (axes) и датчик давления и температуры (bmp180) использует

формат чисел с плавающий запятой, а жесты передаются в виде строки, содержащий обнаруженное движение.

Для получения данных от клиента посредством POST запроса с его стороны, используемых для переключения светодиодных ламп и светодиодной ленты, используется формат, приведенный ниже на рисунке 7.

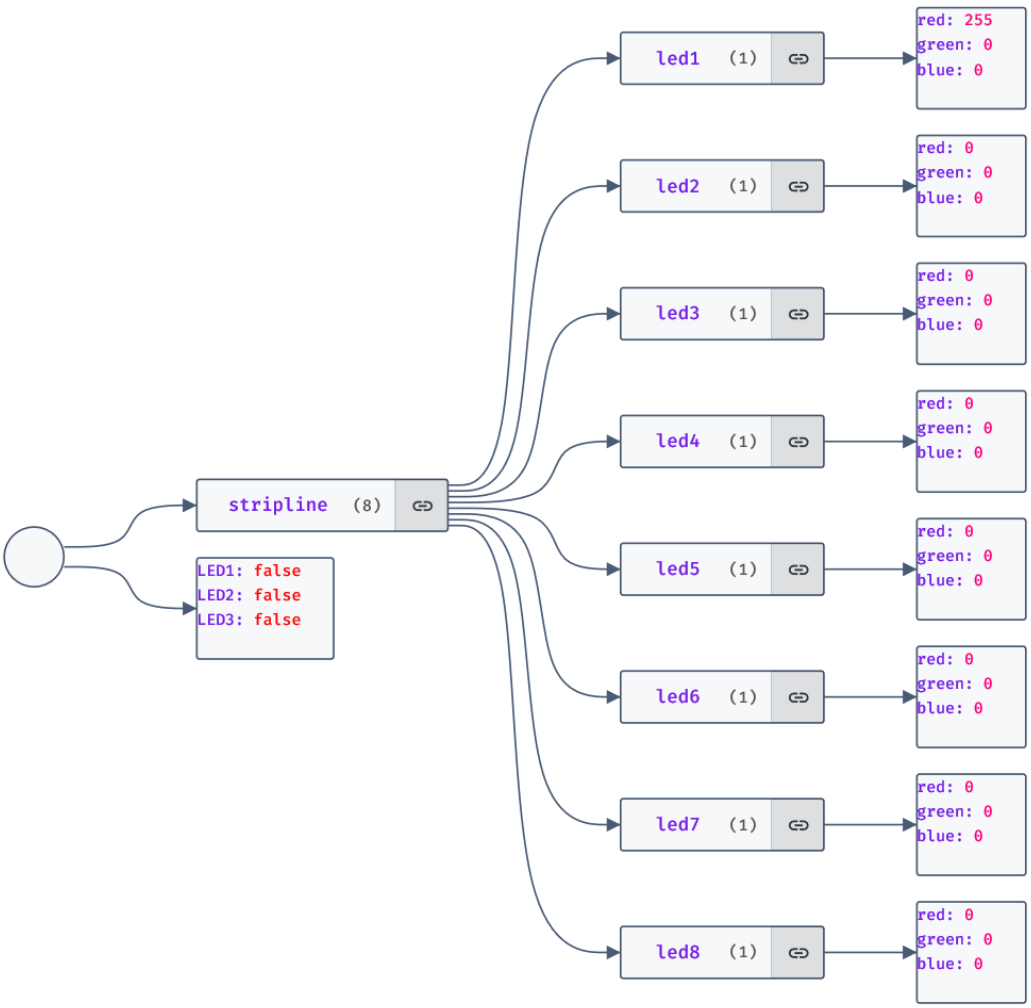

Рисунок 6. Структура JSON для POST /sensors/leds запроса

Он полностью идентичен по структуре элементу leds у формата для GET запросов. Несмотря на то, что в данном случае выделение отдельного блока stripline может показаться излишним, так как помимо него существует только один другой тип элементов, такая унификация может существенно упростить для клиента работу с данными, так как позволяя при получении ответа на GET //sensors/leds модифицировать его и сразу отправить обратно не проводя создания новых параметров и сериализации нового JSON-объекта.

Также для передачи данных используется аргумент updateTimeout GET запроса HTTP к адресу /update. При запросе формата /update?updateTimeout=1000 он устанавливает значение данной переменной, управляющее частотой отправки полных данных сенсоров клиенту (по умолчанию значение 0 и соответствует отсутствию отправки). В данном случае требуется управление единственной переменной, не имеющей отношения к данным периферийных устройств, и для простоты оно приведено к такому виду.

2.4.5. Обработка запросов

После отправки GET или POST запроса вызывается функция для его обработки, как правило собирающая данные с сенсоров и отправляющая их пользователю. Рассмотрим, как система адресации из пункта 2.4.2. соотносится с программными функциями и каким образом происходит сбор данных макета. Далее в таблице 2 приведено соотношение запросов к функциям для удобства дальнейшего объяснения.

Таблица 2. Соотношение запросов и функций-обработчиков

Метод HTTP	Адрес	Функция
GET	/	get_main()
GET	/update	set_update()
GET	/sensors	get_sensors()
GET	/sensors/leds	get_sensLeds()
POST	/sensors/leds	post_sensLeds()
GET	/sensors/buttons	get_sensButtons()
GET	/sensors/lightning	get_sensLight()
GET	/sensors/gestures	get_sensGest()
GET	/sensors/pressure	get_sensPress()
GET	/sensors/axes	get_sensAxes()

В качестве промежуточного этапа при работе с устройствами эти функции используют структуру standData, содержащую параметры и результаты измерений устройств. Она используется для чтения данных перед их сериализацией и включает в себя функцию, её выполняющую согласно описанной ранее структуре, и она же используется для записи полученных данных от пользователя перед их установкой. Структура описана в файле standData.h, ниже приведена визуализация переменных и

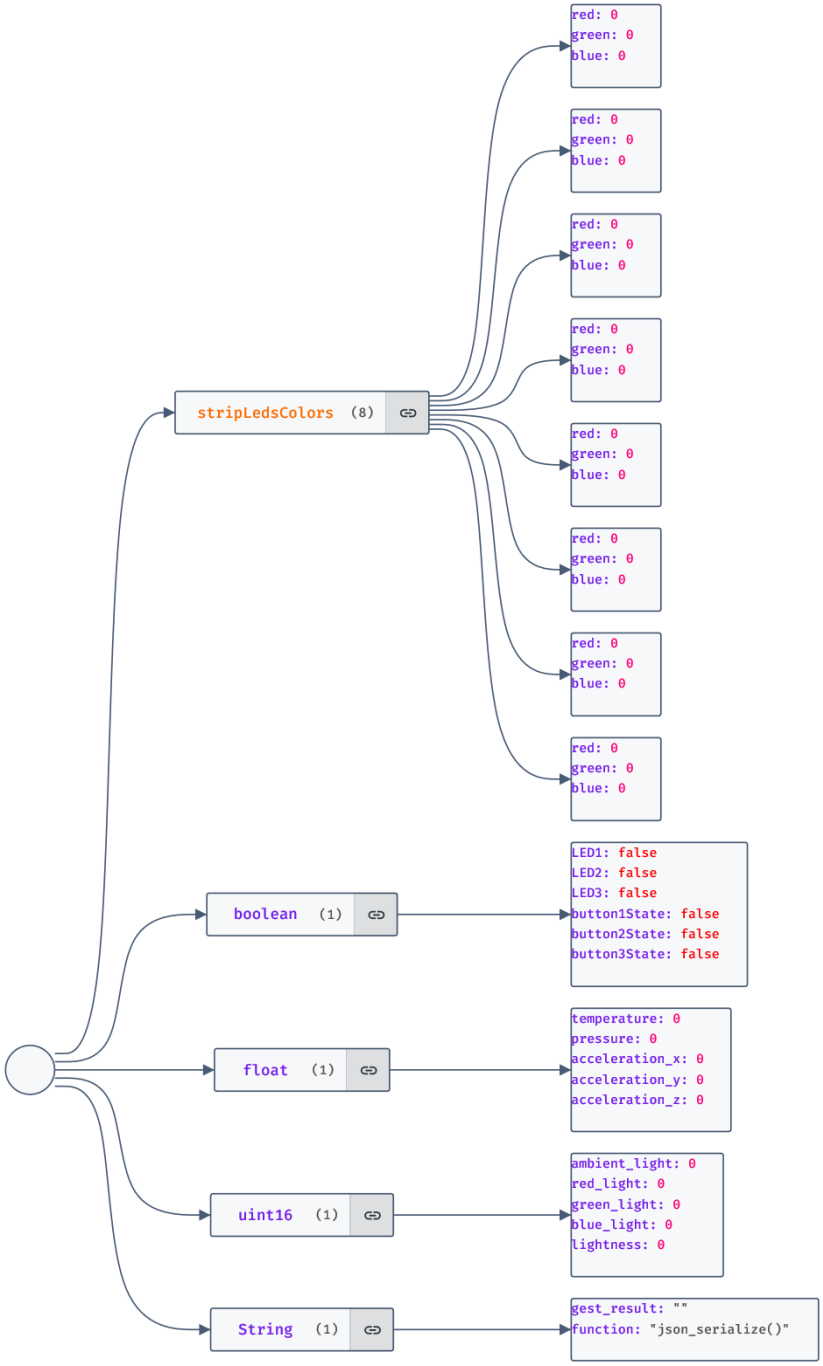

Рисунок 7. Структура standData

Рассмотрим работу функций.

Функция get_main() отправляет клиенту заранее заданную HTML страницу с общими сведениями о макете, а также полем для отправки POST запроса к /sensors/leds с возможностью записать вручную JSON строку. Данный элемент служит исключительно для простой демонстрации работы POST запросов, предоставляя для этого интерфейс в браузере без необходимости дополнительных инструментов.

Функция set_update() обрабатывает запросы GET запрос к //update, проверяет наличие аргумент updateTimeout, пытается, при наличии, перевести его в целочисленный формат, и при успехе записывает значение в одноименную переменную. Эта переменная управляет работой обновления данных в функции loop(), при ненулевом значении каждую итерацию будет проводится проверка, прошло ли updateTimeout секунд с последнего обновления. Если да – производится вызов get_sensors() с последующей отправкой данных, собранных макетом подключенному клиенту.

Функция get_sensors() выполняет основные задачи приложения – сбор и упаковку данных. Рассмотрим подробно алгоритм работы данной функции.

В первую очередь она считывает состояние (true/false) трех светодиодов и сохраняет их значения в переменные standData.LED1, standData.LED2, standData.LED3.

Далее аналогично считываются состояния трех кнопок и сохраняет их значения в переменные standData.button1State, standData.button2State, standData.button3State.

Считывает значение освещенности с аналогового пина PIN_LIGHT и сохраняет его в переменную standData.lightness.

Считывает значения освещенности с помощью датчика жестов. Значения сохраняются в переменные standData.ambient_light, standData.red_light, standData.green_light и standData.blue_light. Здесь при возникновении ошибки при считывании значений, выводится сообщение об ошибке.

Выполняет обнаружение жестов через дополнительную функцию gesture_detect(), которая возвращает обнаруженной движение в виде строки, и сохраняет результат в переменную standData.gest_result.

Считывает значения температуры и давления с помощью датчика BMP180, используя дополнительную функцию get_bmp180val() и сохраняет

их в переменные standData.temperature и standData.pressure. Значения также выводятся на последовательный порт.

Считывает значения ускорения по осям X, Y и Z с помощью акселерометра (accel) и сохраняет их в переменные standData.acceleration_x, standData.acceleration_y и standData.acceleration_z.

Копирует значения цветов светодиодов из массива ledsLine в массив standData.stripLedsColors.

Сериализует данные standData в формате JSON с помощью дочерней функции standData json_serialize() и сохраняет результат в переменную jsonMessage, после чего отправляет HTTP-ответ с кодом 200 и содержимым jsonMessage клиенту и устанавливает значение глобальной переменной json_includes в -1.

Функции get_sens*, обрабатывающие GET запросы к страницам /sensors/* также используют функцию get_sensors() для получения данных, но предварительно изменяют переменную json_includes. Ниже в таблице 3 приведено соотношение значений json_includes и передаваемых в итоге данных.

Таблица 3. Обработка json_includes

json_includes	Данные
0	Отправляется только блок leds структуры JSON
1	Только блок buttons
2	Только блок lightning
3	Только блок gestures
4	Только блок pressure
5	Только блок axes
Другие значения	Отправляется объединение всех блоков структуры

Функция post_sensLeds() проверяет наличие переданного с запросом текста, который может быть преобразован в JSON, после чего пытается прочитать соответствующие структуре leds элементы в структуру standData, после чего передать их значения по соответствующим пинам в светодиодные лампочки и в массив ledsLine, соответствующий значениям светодиодной ленты.

2.5. Выводы

По изложенным выше теоретическим данным разработана программа, предоставляющая легко интегрируемые программные интерфейсы для взаимодействия с макетом, позволяющие считывать данные всех подключенных сенсоров и управлять состоянием светодиодной ленты, светодиодных ламп, а также возможность установить постоянной периодической передачи данных от макета клиенту по сети.

Дополнительные отладка, обновление и модификации программы могут быть легко проведены без детального знания об изложенном выше процессе разработки за счёт использования распространенных интерфейсов, протоколов и форматов данных, структуры программы и применения общепринятых средств разработки.

Листинг кода разработанной программы, состоящий из двух файлов, приведен в Приложении А.

3. АНАЛИЗ ПРИМЕНЕНИЙ

Лабораторный макет предлагает широкий спектр потенциальных применений в учебного процессе. Основной целью его использования в образовании является создание интерактивной и практической среды для студентов, где они могут применить теоретические знания, приобретенные в учебных курсах, на практике. Макет позволяет им не только изучать принципы работы беспроводных сетей, микроконтроллеров и сенсоров, но и разрабатывать свои собственные проекты, исследовать и анализировать данные, а также применять полученные результаты для решения реальных задач.

Благодаря REST HTTP-серверу, лабораторный макет может предоставить интерфейс для для работы над визуализацией и обработкой данных сенсоров. Студенты могут научиться проектировать веб-интерфейсы, мобильные приложения или компьютерные программы для взаимодействия с ESP8266 и отображения данных сенсоров. Макет позволяет им отслеживать и собирать данные с подключенных сенсоров, получая представление о условиях окружающей среды.

В данном контексте применение возможно как с точки зрения базы для работы над перечисленными выше интерфейсами, так и для возможности работать над собранными с помощью макета данными для их математического анализа, моделирования по ним ситуаций и так далее. Основной для подобных задач является возможность организовать постоянную и регулярную отправку данных с помощью предоставляемого страницей /update функционала.

Проектирование экспериментов и автоматизация: Возможности макета позволяют студентам разрабатывать и реализовывать эксперименты в контролируемой среде. Например, они могут создавать проекты, в которых светодиоды реагируют на определенный уровень освещенности, обнаруженный датчиком света, или использовать датчик жестов для запуска конкретных действий. Программируя ESP8266 для автоматизации задач на основе данных сенсоров, студенты могут исследовать концепцию систем управления с обратной связью.

Прототипирование решений IoT: С помощью ESP8266 в качестве центрального контроллера, студенты могут создавать прототипы собственных решений IoT. Они могут разрабатывать проекты, такие как система умного

дома, где светодиоды представляют различные устройства, а сенсоры запускают действия в зависимости от условий окружающей среды.

Макет в целом может служить моделью небольшой системы умного дома. Плата ESP8266 является широко распространенной для обеспечения беспроводной связи в подобных проектах, но на практике редко используется для подключения такого количества одинаковых сенсоров, так как они часто расположены далеко друг от друга (а применение проводной связи для таких количеств устройств редко является приемлемым. Таким образом, можно воспринимать можно воспринимать клиента разработанного сервера не только как конечного пользователя, анализирующего итоговые данные, но и ещё одно промежуточное (возможно являющееся компонентом маршрутизатора/роутера, но как правило отдельное) устройство, взаимодействующее с другими платами и проводящее сбор и, по необходимости, первичную обработку таких данных.

Как следствие, студенты могут также работать над программами, связывающими либо один макет как модель умного дома (воспринимая потоки данных через адреса /sensors/* как отдельные и удаленные друг от друга компоненты), либо несколько макетов для более полного моделирования реальных ситуаций.

Это более комплексная задача, включающая в том числе разработку формата, отражающего наличие устройств в разных местах и отправляющих данные одного типа, написание сервера, способного поддерживать параллельно несколько подключений к разным источникам данных и корректный сбор с них информации, и веб-сервера, предоставляющего доступ к интерфейсу из глобальной сети Интернет, а не только в пределах локальной сети.

Понимание сетевого взаимодействия: Лабораторная модель может помочь понять концепции сетевого взаимодействия и роль Интернета в IoT. Макет использует широко распространенные технологий для взаимодействия с удаленными клиентами.

Разработка приложений, обрабатывающих данные в формате JSON без использования сторонних библиотек, может быть полезно в понимании работы с данным форматом, являющимся крайне распространенным в информационных технологиях, и в том числе стандартизированным для многих сфер.

Аналогично, разработка сокетов для взаимодействия с REST HTTP может быть полезна с точки зрения получения навыков, востребованных в данной сфере. Этот стиль не только является широко распространенным для многих областей информационных технологий, но и является единственным официально стандартизированным для облачных сервисов, к которым можно отнести и многие разработки в Интернете вещей.

Разработка средств для подобного сетевого взаимодействия интересна как с точки зрения взаимодействия с REST, так и с точки зрения работы с самим протоколом HTTP, являющимся, на данный момент наиболее распространенным методом обмена информации в сети.

Также стоит отметить, что из важных аспектов разработки программного обеспечения для данного макета является его модульность и расширяемость. Использование человекочитаемых форматов данных и выделение функциональных блоков в отдельные функции и структуры позволяют пользователям легко изменять код, добавлять новые типы сенсоров и устройств, а также модифицировать существующую функциональность без необходимости в глубоком понимании процесса разработки данного программного обеспечения.

Это позволяет легко интегрировать новые устройства в макет при необходимости расширить его функционал, а отключение уже имеющихся не требуют никаких дополнительных модификаций не повлияет на работоспособность устройства и сохранит работу интерфейсов других датчиков и сервера.

Учитывая сказанной выше, плата с разработанным программным обеспечением также может быть использована для обучения написанию программ для микроконтроллеров и работы в Arduino IDE. Она позволяет не писать комплексное ПО с нуля, а работать над отдельными компонентами в рамках обучения.

Это может быть как разработка новых функций, так и работа над уже имеющимся в данной программе функционале: его доработка или использование урезанной версии ПО в качестве шаблона для работы над конкретными задачами.

В частности, наличие легко модифицируемой функции сериализации JSON, работающего веб-сервера и службы mDNS для Wi-Fi сети позволяет

работать над непосредственным изучением способов работы с периферийными устройствами (или даже не связанными с ними функциями, выполняемыми на микроконтроллере) не превращая подобную задачу в более комплексный проект, требующий параллельно рабочего сервера и т.п.

С другой стороны, возможны различные доработки программы. Например, расширение функционала для первичной обработки собираемой информации, включающее возможность применения различных триггеров для отправки данных или управления элементами макета. Так, датчик жестов может использоваться для активации светодиодных элементов (в тоже время, это может быть и частью клиентской программы, взаимодействующей с макетом по сети), переключение кнопок может также использоваться для включения/выключения элементов, начала сбора данных и так далее.

В целом, программа позволяет применять макет для изучения многих базовых и ключевых для сферы Интернета вещей технологий.

4. БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

4.1. Обзор стандартов

Основным стандартом, определяющим требования к облачным приложениям, является ГОСТ Р ИСО/МЭК 19831-2017 [10], который является локализацией международного стандарта ISO/IEC 19831:2015 и полностью ему идентичен.

Стандарт Модель и протокол интерфейса управления облачной инфраструктуры (CIMI) является открытым стандартом, разработанным Distributed Management Task Force (DMTF), для управления облачной инфраструктурой. CIMI позволяет пользователям управлять жизненным циклом облачных ресурсов, таких как машины, сети и хранилища, через единый интерфейс.

Основные особенности стандарта CIMI включают:

Модель облачной инфраструктуры: CIMI определяет абстрактные модели для различных ресурсов облачной инфраструктуры, таких как машины, сети и хранилища. Эти модели предоставляют стандартные интерфейсы для управления жизненным циклом ресурсов и обеспечивают прозрачность для потребителей облачных услуг.

RESTful HTTP-протокол: CIMI определяет RESTful HTTP-протокол для коммуникации между клиентами и серверами. Это позволяет клиентам использовать простые HTTP-запросы для управления ресурсами облачной инфраструктуры. Протокол поддерживает основные HTTP-методы, такие как GET, POST, PUT, DELETE и PATCH, для управления ресурсами.

Форматы обмена данными: CIMI рассматривает несколько форматов обмена данными, включая XML и JSON. Это обеспечивает гибкость для клиентов при обмене информацией с серверами.

Безопасность: CIMI предоставляет механизмы для защиты данных и ресурсов облачной инфраструктуры. Это включает в себя авторизацию, аутентификацию и шифрование.

CIMI имеет цель упростить и стандартизировать управление облачной инфраструктурой, предоставляя простые и единые интерфейсы для различных облачных услуг.

Наиболее важными для данной работы являются описания RESTful HTTP маршрутизации и принципов сериализации JSON и использования его

для передачи данных. Далее будут подробно рассмотрены предлагаемые стандартом требования в отношении к разработанному приложению.

4.2. REST HTTP

Стандарт CIMI от DMTF продвигает архитектурный стиль REST (Representational State Transfer) для взаимодействия между клиентами и серверами.

Стиль REST существует как расширение протокола HTTP и использует его методы (GET, POST, PUT, DELETE и PATCH) для выполнения операций над ресурсами облачной инфраструктуры. REST представляет простой, универсальный интерфейс для работы с ресурсами.

Рассмотрим ключевые аспекты использования REST в стандарте CIMI:

Ресурсы: В контексте REST, ресурсы – это основные абстракции, которыми управляют клиенты. В CIMI ресурсы представляют элементы облачной инфраструктуры, такие как машины, сети и хранилища. Каждый ресурс имеет свой собственный URL, который используется для идентификации ресурса.

HTTP-методы: CIMI использует методы HTTP для выполнения операций над ресурсами. Это включает методы GET для чтения ресурса, POST для создания нового ресурса, PUT и PATCH для обновления ресурса, и DELETE для удаления ресурса.

Примером применения предыдущих двух пунктов может служить, например, способ маршрутизация адресов в соответствии с запросами:

Создать пользователя: POST /users

Удалить пользователя: DELETE /users/1

Получить всех пользователей: GET /users

Получить одного пользователя: GET /users/1

В разработанном приложении также применяются только методы GET/POST и приведенные выше два пункта реализованы в рассмотренное в разделе 2 схеме адресов, используя, например, GET /sensors для получения клиентом доступных данных о подключенных устройствах, POST /sensors для изменения параметров для тех компонентов, где это возможно и так далее.

Стандартные интерфейсы: Одно из ключевых преимуществ использования REST в CIMI – это то, что он предлагает стандартные интерфейсы для

взаимодействия с ресурсами. Это обеспечивает совместимость между различными облачными услугами и упрощает процесс интеграции и использования этих услуг. Макет использует исключительно широко распространенные стандарты HTTP и JSON.

Отсутствие состояния: REST является без состояния, что означает, что каждый запрос клиента к серверу содержит всю информацию, необходимую для обработки запроса. Это упрощает процесс обработки запросов на стороне сервера и обеспечивает большую масштабируемость. В построенном приложении все запросы будут обработаны одинаково, независимо от наличия или отсутствия предыдущих обменов информацией, т.е. данное требования к REST выполняется.

4.3. JSON

Для передачи и приема данных макет использует формат JSON. В CIMI данные об обмене между клиентом и сервером также могут быть представлены в формате JSON в теле HTTP-запросов и ответов. Это позволяет обмениваться сложными данными и структурами данных между клиентом и сервером.

JSON, или JavaScript Object Notation, является универсальным форматом обмена данными, который легко читается и генерируется как людьми, так и машинами. В стандарте CIMI JSON используется для сериализации данных о ресурсах облачной инфраструктуры, которые передаются между клиентом и сервером.

Формат обмена данными: JSON используется как основной формат обмена данными в CIMI. Когда клиенты отправляют HTTP-запросы к серверу или получают HTTP-ответы от сервера, данные в этих сообщениях обычно представлены в формате JSON. В данном макете никакие дополнительные форматы данных не используются, таким образом с этой точки зрения обмен данных полностью стандартизирован.

Модели данных: В CIMI определены различные модели данных для ресурсов облачной инфраструктуры, таких как машины, сети и хранилища. Эти модели данных могут быть сериализованы в формат JSON для передачи данных между клиентом и сервером.

Основная цель этих моделей данных – предоставить стандартный интерфейс для работы с различными ресурсами облачной инфраструктуры,

обеспечивая согласованность и совместимость между различными облачными услугами.

В данной работе макет существует в рамках заранее известной локальный сети и необходимости предоставлять детальные сведения о состоянии устройства или сети нет необходимости, отдельных от них хранилищ данных не исопльзуется.

Сериализация и десериализация. Сериализация – это процесс преобразования состояния объекта или данных в формат, который можно передать или сохранить. В контексте CIMI, это обычно включает преобразование состояния ресурсов облачной инфраструктуры, таких как машины, сети и хранилища, в формат JSON. Десериализация – это обратный процесс сериализации. Он включает преобразование сериализованных данных обратно в их исходное состояние. В контексте CIMI, это обычно включает преобразование данных в формате JSON обратно в внутренние модели данных, которые можно использовать для обработки запросов и выполнения операций над ресурсами облачной инфраструктуры. Эти методы, таким образом, исключают прямое взаимодействие с данными в формате JSON, и реализованы в разработанном приложении посредством применения библиотеки ArduinoJSON, сериализация для отправки данных клиенту, и десериализация для разбора полученных данных об управлении устройством.

Совместимость: JSON – это универсальный формат обмена данными, который поддерживается многими языками программирования. Это обеспечивает широкую совместимость между различными системами и платформами, которые используют стандарт CIMI.

4.4. Заключение

Подведем итоги по анализу соответствия программы стандарту Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol.

Соответствие стандарту REST: Программа успешно применяет принципы REST, предоставляя ресурсы, идентифицируемые через URL-ы, и используя стандартные HTTP методы для операций над этими ресурсами. Такой подход обеспечивает универсальность и расширяемость интерфейса.

Использование JSON: Программа использует JSON в качестве основного формата обмена данными. Это делает обмен данными между клиентами и

сервером эффективным и стандартизированным, позволяя передавать сложные структуры данных.

Применение моделей данных: Программа применяет модели данных, описанные в стандарте CIMI, для представления различных ресурсов облачной инфраструктуры. Это обеспечивает стандартизацию и универсальность при работе с различными ресурсами.

В целом, программа успешно соответствует стандарту и корректно применяет общепринятые технологии. Это также полезно с точки зрения учебного применения макетов, позволяя знакомить студентов со стандартами сферы и её наиболее распространенными технологиями.

ЗАКЛЮЧЕНИЕ

В ходе проведения данной дипломной работы была успешно выполнена разработка программного обеспечения для учебного макета на базе микроконтроллера ESP8266, подключенного к набору сенсоров. Разработанное ПО обеспечивает эффективное клиент-серверное взаимодействие, что позволяет студентам глубже изучить и понять концепции IoT.

Работа была выполнена в соответствии с поставленными задачами: произведен анализ и выбор подходящих технологий для разработки ПО, выполнено проектирование и реализация ПО, проведено его тестирование и отладка, а также проведен анализ возможных применений и улучшений в будущем.

Результаты работы показали, что разработанное программное обеспечение обеспечивает надежное и эффективное взаимодействие между учебным макетом и компьютером, упрощает процесс управления подключенными устройствами и сбора данных.

Программное обеспечение разработано в соответствии со стандартом ГОСТ Р ИСО/МЭК 19831-2017, использует распространенные и принятые в сфере Интернета вещей технологии: HTTP сервер, архитектурный стиль REST и формат данных JSON. Это упрощает процесс работы с макетом, его возможные отладку и доработку и позволяет легко интегрировать его с другими программными интерфейсами.

С точки зрения применения в образовательном процессе, RESTful HTTP и JSON, являясь актуальными технологиями, делают возможным для студентов изучить их в рамках работы с этим макетом. Также возможно использование макета в качестве модели системы умного дома для разработки различных проектов, связанных с обработкой данных и предоставлением доступных конечному клиенту интерфейсом для работы с устройствами.

Программа предоставляет интерфейсы для сбора различных данных: уровней освещенности, цвета, атмосферного давления, обнаруженных жестов, температуры и состояния кнопок и светодиодных элементов. Это позволяет применять его для изучения потоков данных, их анализа и математического моделирования, например, в области климатических наблюдений.

Важным аспектом разработки программного обеспечения для данного макета была его модульность и расширяемость. За счёт применение челове-

кочитаемых форматов данных и выделения функциональных блоков в отдельные функции/структуры, пользователи могут легко модифицировать код и добавлять новые типы сенсоров и устройств, а также вносить изменения в существующую функциональность, без глубокого знакомства с процессом разработки ПО. Это делает макет гибким инструментом для обучения и позволяет студентам адаптировать его под свои потребности и исследовательские задачи.

В заключение, разработанное программное обеспечение для учебного макета на базе микроконтроллера ESP8266 представляет собой эффективный инструмент для реализации концепций Интернета вещей. Оно обеспечивает клиент-серверное взаимодействие, простое управление устройствами и сбор данных, и может быть применено эффективно в учебном процессе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Скиена, С. Алгоритмы. Руководство по разработке. 2-е изд. Пер. с англ. СПБ "БХВ Петербург", 2011. 720 с.

2. Страуструп, Б. Язык программирования C++. 4-е изд. Пер. с англ..: Бином. 2021. 1136 с.

3. FastLed Wiki // Github URL: https://github.com/FastLED/FastLED/wiki/

4. ESP8266 Arduino Core Documentation // Read the Docs. URL: https://readthedocs.org/projects/arduino-esp8266/downloads/pdf/latest/ (дата обращения: 28.04.2023).

5. APDS-9960 Datasheet // Arduino. URL: https://content.arduino.cc/assets/Nano_BLE_Sense_av02-4191en_ds_apds-9960.pdf (дата обращения: 28.04.2023).

6. Documentation // ArduinoJSON. URL: https://arduinojson.org/v6/doc/ (дата обращения: 09.04.2023).

7. BMP180 Digital pressure sensor // Adafruit. URL: https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf (дата обращения: 28.04.2023).

8. Data Sheet – ADXL345 // Analog Devices. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/adxl345.pdf (дата обращения: 28.04.2023).

9. How do IoT protocols affect a device’s energy consumption? // QOITECH URL: https://www.qoitech.com/blog/how-do-iot-protocols-affect-a-devices-energy-consumption (дата обращения: 01.05.2023)/

10. ГОСТ Р ИСО/МЭК 19831-2017. Модель и протокол интерфейса управления облачной инфраструктурой (CIMI). Интерфейс для управления облачной инфраструктурой. Введ. 01.01.2018. М.: Стандартинформ, 2018.

11. Гололобов В. Н. «Умный дом» своими руками / В. Н. Гололобов. – Москва: НТ Пресс, 2007.

12. Шугаев С. Система умный дом / С. Шугаев – автоматизация технологических процессов: Выпуск No2, 2013г.

13. ESP8266 NodeMCU WiFi Development Board // HandsonTec URL: https://handsontec.com/dataspecs/module/esp8266-V13.pdf (дата обращения: 01.05.2023)

ПРИЛОЖЕНИЕ А

server.ino

```
#include <FastLED.h> //led stripline interaction
#include <Wire.h> //for spi bus
#include <SFE_BMP180.h> //pressure and temperature sensor
#include <Adafruit_Sensor.h> //axelerometer
#include <Adafruit_ADXL345_U.h> //
#include <SparkFun_APDS9960.h> //gestures sensor
#include <ESP8266WiFi.h> //for connection to wifi
#include <WiFiClient.h> //
#include <ESP8266WebServer.h> //for creating HTTP server
#include <ESP8266mDNS.h> //for setting up mDNS

#include "utils.h" //contains struct for sensors/leds/buttons data

//pins
#define PIN_LED1 12
#define PIN_LED2 13
#define PIN_LED3 15
#define PIN_LIGHT A0
#define PIN_BUTT1 14
#define PIN_BUTT2 0
#define PIN_BUTT3 2

#define TIMEOUT_SERIAL 5000 //time to wait until serial is connected

#define DATA_FASTLED_PIN 16
#define FASTLED_ESP8266_RAW_PIN_ORDER

CRGB ledsLine[NUM_LEDS_IN_STRIPLINE];

SFE_BMP180 pressureSens;
struct BMP180val {
  double  temperature,
          pressure;
};

//set unique id for sensor
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12346);

//init geustures sensor
SparkFun_APDS9960 apds = SparkFun_APDS9960();

StandData standData;

int updateTimeout;
unsigned long update_time = 0;


ESP8266WebServer server(80);
```

```
const char* ssid = "238";
const char* password = "msg54321";
const char* mdnsName = "esp8266";

//main HTML page
const String IndexPage = "<html>\
  <head>\
    <title>ESP8266 Web Server POST handling</title>\
    <style>\
      body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-Serif; Color:
#000088; }\
    </style>\
  </head>\
  <body>\
    <h1>ESP8266 laboratory model<h1>\
    <h2>POST plain text to /sensors/leds. For example: {\"LED1\": true} will enable LED1</h2>\
    <form method=\"post\" enctype=\"text/plain\" action=\"/sensors/leds\">\
      <input type=\"text\" name=\"json\" value=\'{JSON string}\'>\
      <br><br>\
      <input type=\"text\" name=\"text\" value=\'\'>\
      <input type=\"submit\" name=\"button\" value=\"Send string\" >\
    </form>\
  </body>\
</html>";

//handler for main page
void get_main()
{
  server.send(200, F("text/html"), IndexPage);
  powerled_blink(4, 40);
}

//handler for /sensors
void get_sensors ()
{
  standData.LED1 = digitalRead(PIN_LED1);
  standData.LED2 = digitalRead(PIN_LED2);
  standData.LED3 = digitalRead(PIN_LED3);

  standData.button1State = digitalRead(PIN_BUTT1);
  standData.button2State = digitalRead(PIN_BUTT2);
  standData.button3State = digitalRead(PIN_BUTT3);

  standData.lightness = analogRead(PIN_LIGHT);
  if (!apds.readAmbientLight(standData.ambient_light) ||
      !apds.readRedLight(standData.red_light) ||
      !apds.readGreenLight(standData.green_light) ||
      !apds.readBlueLight(standData.blue_light) )
    Serial.println("Error reading light values");

  standData.gest_result = gesture_detect();
```

```
  BMP180val tmp = get_bmp180val();
  standData.temperature = tmp.temperature;
  Serial.print("Temperature: "); Serial.println(standData.temperature);
  standData.pressure = tmp.pressure;
  Serial.print("Pressure: "); Serial.println(standData.pressure);

  sensors_event_t event;
  accel.getEvent(&event);
  standData.acceleration_x = event.acceleration.x;
  standData.acceleration_y = event.acceleration.y;
  standData.acceleration_z = event.acceleration.z;

  for (int i = 0; i < NUM_LEDS_IN_STRIPLINE; i++)
    standData.stripLedsColors[i] = ledsLine[i];

  String jsonMessage = standData.json_serialize();
  server.send(200, F("text/plain"), jsonMessage);
  powerled_blink(4, 40);

  json_includes = -1;
}

//APDS sensor
String gesture_detect()
{
  String temporary;
  if ( apds.isGestureAvailable() ) {
    switch ( apds.readGesture() ) {
      case DIR_UP:
        temporary = "UP";
        break;
      case DIR_DOWN:
        temporary = "DOWN";
        break;
      case DIR_LEFT:
        temporary = "LEFT";
        break;
      case DIR_RIGHT:
        temporary = "RIGHT";
        break;
      case DIR_NEAR:
        temporary = "NEAR";
        break;
      case DIR_FAR:
        temporary = "FAR";
        break;
      default:
        temporary = "NONE";
    }
  }
  else {
```

```
    temporary = "NONE";
  }
  Serial.print("Gesture detected: "); Serial.println(temporary);
  return temporary;
}

void post_sensLeds()
{
  if (server.method() != HTTP_POST) {
    server.send(405, "text/plain", "Method Not Allowed");
  }
  else {
    server.send(200, "text/plain", "POST body was:\n" + server.arg("plain"));
    String json = server.arg("plain");
    Serial.print("args : "); Serial.println(json);

    StaticJsonDocument<700> jsonDocument;
    deserializeJson(jsonDocument, json);

    standData.LED1 = jsonDocument["LED1"];
    digitalWrite(PIN_LED1, standData.LED1);
    standData.LED2 = jsonDocument["LED2"];
    digitalWrite(PIN_LED2, standData.LED2);
    standData.LED3 = jsonDocument["LED3"];
    digitalWrite(PIN_LED3, standData.LED3);

    String tmp = "led";
    for (int i = 0 ; i < NUM_LEDS_IN_STRIPLINE; i++)
    {
      String nameLeds = tmp + String(i + 1);
      JsonObject obj_led = jsonDocument["stripline"][nameLeds];
      Serial.print("Json object is; "); Serial.println(nameLeds);
      standData.stripLedsColors[i].r = obj_led["red"];
      standData.stripLedsColors[i].b = obj_led["blue"];
      standData.stripLedsColors[i].g = obj_led["green"];
      Serial.println(standData.stripLedsColors[i].b);
      Serial.println(standData.stripLedsColors[i].r);
      Serial.println(standData.stripLedsColors[i].g);
    }
    //show a new line colors
    for (int i = 0; i < NUM_LEDS_IN_STRIPLINE; i++) {
      ledsLine[i] = standData.stripLedsColors[i];
    }
    FastLED.show();
  }
  powerled_blink(4, 40);
}

void set_update()
{
  String st = "";
  if (server.hasArg("updateTimeout") == false) { //Check if body received
```

```
    server.send(200, "text/plain", "No arguments received.");
    return;
  }
  st = server.arg("updateTimeout");
  updateTimeout = st.toInt();
  Serial.print("Set timeout to (milliseconds): "); Serial.println(updateTimeout);
}


//terrible realization, but ESP8266WebServer doesnt seem to have any way to handle page
names etc
void get_sensLeds() {
  json_includes = 0;
  get_sensors();
}
void get_sensButtons() {
  json_includes = 1;
  get_sensors();
}
void get_sensLight() {
  json_includes = 2;
  get_sensors();
}
void get_sensGest() {
  json_includes = 3;
  get_sensors();
}
void get_sensPress() {
  json_includes = 4;
  get_sensors();
}
void get_sensAxes() {
  json_includes = 5;
  get_sensors();
}



// Define routing
void restServerRouting()
{
  //return HTML page
  server.on("/", HTTP_GET, get_main);
  //return JSON data value representation
  server.on(F("/sensors"), HTTP_GET, get_sensors);

  server.on(F("/sensors/leds"), HTTP_GET, get_sensLeds);
  server.on("/sensors/leds", HTTP_POST, post_sensLeds);

  server.on(F("/sensors/buttons"), HTTP_GET, get_sensButtons);
  server.on(F("/sensors/lightning"), HTTP_GET, get_sensLight);
  server.on(F("/sensors/gestures"), HTTP_GET, get_sensGest);
```

```
  server.on(F("/sensors/pressure"), HTTP_GET, get_sensPress);
  server.on(F("/sensors/axes"), HTTP_GET, get_sensAxes);

  server.on("/update", HTTP_GET, set_update);
}

// Manage not found URL
void handleNotFound() {
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) {
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);

  Serial.println("Error query from client");
  Serial.println(millis());
  powerled_blink(4, 40);
}

void write_Log() {
  static unsigned long t_in = 0;
  static const int decay_sec = 10; //write log each

  if (t_in + decay_sec * 1000 < millis())
  {
    t_in = millis();
    Serial.print("Log time: "); Serial.print(t_in / 1000 / 60); Serial.print(":"); Serial.println(t_in /
1000 % 60);
  }
}

//used for indication of erros
void led_alarmBlink()
{
  for (int i = 0; i++; i < 10)
  {
    digitalWrite(PIN_LED3, !digitalRead(PIN_LED1));
    delay(400);
  }
}

//reads temperature and pressure
BMP180val get_bmp180val() {
  BMP180val tmp;
  char status;
```

```
  status = pressureSens.startTemperature();
  if (status != 0)
  {
    delay(status);
    status = pressureSens.getTemperature(tmp.temperature);
    if (status != 0)
    {
      status = pressureSens.startPressure(3);
      if (status != 0)
      {
        delay(status);
        status = pressureSens.getPressure(tmp.pressure, tmp.temperature);
        if (status == 0)
          Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");

  return tmp;
}

//led indicator
void powerled_blink(int blinkNum, int decay)
{
  if (blinkNum < 2) blinkNum = 2;
  for (int i = 0; i < blinkNum; i++)
  {
    digitalWrite(PIN_LED3, 1);
    delay(decay);
    digitalWrite(PIN_LED3, 0);
    delay(decay);
  }
}

void setup(void) {
  pinMode(PIN_LED3, OUTPUT);
  powerled_blink(5, 50);

  Serial.begin(115200);

  //attemp to connect for TIMEOUT_SERIAL milliseconds
  unsigned long start = millis();
  while (!Serial) {
    if (millis() - start > TIMEOUT_SERIAL)
      break;
  }

  Serial.println("Serial port started");
  powerled_blink(2, 40);
```

```
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Attempting to connect to network: "); Serial.println(ssid);
  while (WiFi.status() != WL_CONNECTED) {
    delay(50);
    Serial.print(".");
  }
  powerled_blink(4, 100);
  Serial.print("Connected to ");  Serial.println(ssid);
  Serial.print("IP address: ");  Serial.println(WiFi.localIP());

  // Activate mDNS this is used to be able to connect to the server
  // with local DNS hostmane esp8266.local
  if (MDNS.begin("esp8266"))
    Serial.println("MDNS responder started");
  MDNS.addService("http", "tcp", 80);

  //define routing for server and start it
  restServerRouting();
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("HTTP server started");

  powerled_blink(2, 300);

  //pins mode setup
  pinMode(PIN_LED1, OUTPUT);
  pinMode(PIN_LED2, OUTPUT);
  pinMode(PIN_LED3, OUTPUT);

  pinMode(PIN_BUTT1, INPUT_PULLUP);
  pinMode(PIN_BUTT2, INPUT_PULLUP);
  pinMode(PIN_BUTT3, INPUT_PULLUP);

  FastLED.addLeds<WS2812B, DATA_FASTLED_PIN, RGB>(ledsLine,
NUM_LEDS_IN_STRIPLINE);

  //global lightness
  FastLED.setBrightness(64);
  ledsLine[0] = CRGB::Red;

  //pressure init
  if (!pressureSens.begin())
  {
    Serial.println("BMP180 init failure");
    led_alarmBlink();
  }

  //axeleration sensor init
  if (!accel.begin())
  {
```

```
    Serial.println("Ooops, no ADXL345 detected ... Check your wiring!");
    led_alarmBlink();
  }
  else {
    accel.setRange(ADXL345_RANGE_16_G);
  }

  //geusture sensor init
  if ( !apds.init() )
  {
    Serial.println(F("Something went wrong during APDS-9960 init!"));
    led_alarmBlink();
  }

  // Start running the APDS-9960 light sensor (no interrupts)
  if ( !apds.enableLightSensor(false) )
  {
    Serial.println(F("Something went wrong during light sensor init!"));
    led_alarmBlink();
  }

  if ( !apds.enableGestureSensor(false) ) {
    Serial.println(F("Something went wrong during gesture sensor init!"));
    led_alarmBlink();
  }

  // Wait for initialization and calibration to finish
  digitalWrite(PIN_LED3, 1);
  Serial.println("Setup finished.");

}

void loop(void) {
  server.handleClient();
  MDNS.update();
  if (millis() - update_time > updateTimeout && updateTimeout != 0) {
    Serial.println("Timed data sent");
    get_sensors();
    update_time = millis();
  }
  write_Log();
}
```

utils.h

```
#include <ArduinoJson.h>
#include <FastLED.h>

#define NUM_LEDS_IN_STRIPLINE 8
//leds, buttons, lightning, gestures, pressures, axes
int json_includes = -1;
```

```
struct StandData {
  bool  LED1 = false,
        LED2 = false,
        LED3 = false;

  CRGB stripLedsColors[NUM_LEDS_IN_STRIPLINE];

  bool button1State,
       button2State,
       button3State;

  float  temperature = 0,
         pressure = 0;

  uint16_t ambient_light = 0,
           red_light = 0,
           green_light = 0,
           blue_light = 0;

  String gest_result;

  int lightness;

  float acceleration_x,
        acceleration_y,
        acceleration_z;

  String json_serialize(void) {
    StaticJsonDocument<1500> jsonDocument;

    jsonDocument["leds"]["LED1"] = LED1;
    jsonDocument["leds"]["LED2"] = LED2;
    jsonDocument["leds"]["LED3"] = LED3;

    jsonDocument["buttons"]["button1State"] = button1State;
    jsonDocument["buttons"]["button2State"] = button2State;
    jsonDocument["buttons"]["button3State"] = button3State;

    jsonDocument["bmp180"]["temperature"] = temperature;
    jsonDocument["bmp180"]["pressure"] = pressure;

    jsonDocument["lights"]["ambient_light"] = ambient_light;
    jsonDocument["lights"]["red_light"] = red_light;
    jsonDocument["lights"]["green_light"] = green_light;
    jsonDocument["lights"]["blue_light"] = blue_light;
    jsonDocument["lights"]["lightness"] = lightness;

    jsonDocument["gesture"] = gest_result;

    jsonDocument["axes"]["acceleration_x"] = acceleration_x;
    jsonDocument["axes"]["acceleration_y"] = acceleration_y;
```

```
    jsonDocument["axes"]["acceleration_z"] = acceleration_z;

    String led_name = "led";
    for (int i = 0; i < NUM_LEDS_IN_STRIPLINE; i++)
    {
      String nameObj = led_name + (i + 1); //led_name needed as concatention before
      //declaration of one of parts gives unpredictable results
      //JsonObject color_r = jsonDocument.createNestedObject(nameObj);
      jsonDocument["leds"]["stripline"][nameObj]["red"] = stripLedsColors[i].r;
      jsonDocument["leds"]["stripline"][nameObj]["green"] = stripLedsColors[i].g;
      jsonDocument["leds"]["stripline"][nameObj]["blue"] = stripLedsColors[i].b;
    }

    String tmp;

    switch ( json_includes ) {
      case 0:
        serializeJsonPretty(jsonDocument["leds"], tmp);
        break;
      case 1:
        serializeJsonPretty(jsonDocument["buttons"], tmp);
        break;
      case 2:
        serializeJsonPretty(jsonDocument["lights"], tmp);
        break;
      case 3:
        serializeJsonPretty(jsonDocument["gesture"], tmp);
        break;
      case 4:
        serializeJsonPretty(jsonDocument["bmp180"], tmp);
        break;
      case 5:
        serializeJsonPretty(jsonDocument["axes"], tmp);
        break;
      default:
        serializeJsonPretty(jsonDocument, tmp);
    }
    return tmp;
  }
};
```